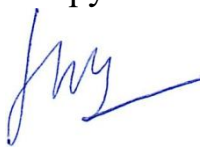
МИНОБРНАУКИ РФ
ФГБОУ ВО «ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

На правах рукописи



МАКСИМОВ ЯРОСЛАВ АЛЕКСАНДРОВИЧ

МЕТОДЫ, МОДЕЛИ И МЕТРИКИ СБОРА ДАННЫХ О ПРОИЗВОДИТЕЛЬНОСТИ КЛИЕНТ-СЕРВЕРНЫХ ВЕБ-ПРИЛОЖЕНИЙ

Специальность 2.3.8. – Информатика и информационные процессы
(технические науки)

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель:
кандидат технических наук,
доцент МАРТЫШКИН А.И.

Пенза – 2025

ОГЛАВЛЕНИЕ

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

БД – база данных.

ВАК – высшая аттестационная комиссия.

ГОСТ – государственный стандарт.

ИС – информационная система.

ПО – программное обеспечение.

СУБД – система управления базами данных.

ЦП – центральный процессор.

ЭВМ – электронная вычислительная машина.

AJAX (*Asynchronous JavaScript and XML)* – асинхронный *JavaScript* и *XML*.

API (Application Programming Interface) – программный интерфейс приложения.

ASCII (American Standard Code for Information Interchange) – стандарт кодирования букв латинского алфавита, цифр, некоторых специальных знаков и управляющих символов.

AST (*Abstract Syntax Tree*) – абстрактное синтаксическое дерево.

B2B (business-to-business) – модель ведения бизнеса, при которой как заказчики, так и поставщики товаров или услуг являются юридическими лицами.

CLF (*Common Log Format)* – общий формат логов.

CLS (*Cumulative Layout Shift*) – общий сдвиг макета.

CSS (*Cascading Style Sheets)* – каскадные страницы стилей.

DOM (*Document Object Model*) – объектная модель документа.

FID (*First Input Delay*) – время от первого ввода до действия.

GRASP (General Responsibility Assignment Software Patterns) – шаблоны ПО для назначения главных ответственностей.

HTML (HyperText Markup Language) – язык гипертекстовой разметки.

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста.

IE (Internet Explorer) – интернет эксплорер.

IETF (Internet Engineering Task Force) – Инженерный совет Интернета,

IP (Internet Protocol) – интернет протокол.

IT (Information Technology) – информационные технологии.

JPEG (Joint Photographic Experts Group) – растровый графический формат изображений.

JVM (Java Virtual Machine) – виртуальная машина *Java*.

LCP (*Largest Contentful Paint*) – время рендеринга самого большого элемента.

NPM (*Node Package Manager*) – пакетный менеджер.

OSI (Open Systems Interconnection) – сетевая модель стека.

REST (Representational State Transfer) – передача состояния представления.

RFC (Request for Comments) – документ, описывающий технические аспекты и стандарты интернета, включая протоколы, процедуры, программы.

SEO (Search Engine Optimization) – поисковая оптимизация.

SOLID (*Single responsibility, Open–closed, Liskov substitution, Interface Segregation, Dependency inversion*) – принцип единой ответственности, принцип открытости-закрытости, подстановка Лисков, разделение интерфейсов, инверсия зависимостей.

SPA (Single Page Application) – одностраничное приложение.

SSR (Server-Side Rendering) – рендеринг на стороне сервера.

TCP (*Transmission Control Protocol*) – транспортный протокол.

URI (Uniform Resource Identifier) – унифицированный идентификатор ресурса.

URL (Uniform Resource Locator) – единообразный указатель местонахождения ресурса.

W3C (World Wide Web Consortium) – консорциум всемирной паутины.

XML (EXtensible Markup Language) – расширяемый язык разметки.

ВВЕДЕНИЕ

Для современного мира характерен стремительный информационно-технологический прогресс коммерческих организаций с расширением их поля деятельности за счет интернет-ресурсов, при этом как государственные сервисы, так и торговые и банковские интернет-услуги представлены в основном посредством веб-приложений, большая часть из которых реализована на базе архитектуры «клиент-сервер», доступной для пользователей через всемирную сеть.

Интернет, который сложился к сегодняшнему времени, развивался на протяжении многих лет благодаря созданию новых подходов и технологий разработки веб-приложений. На ранних этапах развития всемирной паутины основным методом генерации веб-страниц оставался серверный рендеринг (*Server-Side Rendering, SSR*), когда формирование итоговой *HTML*-страницы выполнялось непосредственно на стороне сервера. При каждом запросе клиента сервер выполнял обработку данных, сборку компонентов приложения и генерацию *HTML*-страницы, содержащей как статический контент, так и динамические элементы. Сформированный документ передавался в браузер пользователя, где мог быть отображен без дополнительных операций по изменению контента внутри страницы.

Переломным моментом в эволюции веб-приложений стало появление концепции *Web* 2.0, ознаменовавшей собой переход от статических страниц к интерактивным веб-приложениям. Именно в этот период веб-сайты стали предоставлять пользователям возможность динамического изменения внутренней структуры веб-страницы в режиме реального времени. Существенную роль в этом сыграло распространение асинхронного *JavaScript* (*AJAX*) – новой парадигмы обработки клиент-серверных запросов, основанной на асинхронной модели взаимодействия. Реализации *AJAX* быстро получили распространение в популярных *JavaScript*-библиотеках, таких как *jQuery* и *BackboneJS*, которые предоставляли разработчикам удобные абстракции для работы с *DOM* и для организации интерактивных интерфейсов. Данный прорыв в сфере *веб-технологий* способствовал формированию устойчивых

архитектурных шаблонов пользовательских интерфейсов, что, в свою очередь, стимулировало переход от многостраничных веб-приложений к архитектуре одностраничных приложений (*SPA, Single Page Application*). Основной идеей этого перехода стало смещение части бизнес-логики на сторону клиента, что позволило снизить нагрузку на веб-серверы и повысить их производительность при обработке большого количества запросов.

Однако с ростом сложности клиентской логики возникла необходимость в более масштабируемых и контролируемых архитектурных решениях. Появление компонентных подходов к построению интерфейсов, таких как архитектура *Flux*, представленная инженером-программистом *J. Chen* в 2014 г., ознаменовало отказ от традиционной модели *Model-View-Controller* (*MVC*) в пользу однонаправленного потока данных и централизованного управления состоянием приложения. Использование данного подхода обеспечивало предсказуемость поведения клиентских приложений. Спустя десятилетие в современных веб-разработках вновь наблюдается возрождение интереса к серверному рендерингу, который позволяет быстрее загружать страницы, причем сгенерированные страницы индексируются поисковыми системами. Современные реализации *SSR*, такие как фреймворк *Next.js*, сочетают преимущества серверного и клиентского рендеринга, обеспечивая улучшенную *SEO*-оптимизацию и предоставляя интерфейс для создания веб-страниц. Популярность метода *SSR* в современных условиях объясняется также тем, что он решает задачи первоначальной загрузки, доступности и индексации поисковыми системами, что делает его важным компонентом гибридных архитектур веб-приложений.

Серверный рендеринг ныне рассматривается как один из наиболее эффективных подходов для повышения масштабируемости веб-приложений и обеспечения консистентности передаваемых данных, поскольку логика формирования страницы и агрегации данных выполняется централизованно на стороне приложения. Пользователь получает уже полностью подготовленный *HTML*-документ, снижая риск ошибок, связанных с несовместимостью

клиентских сред или ограничениями вычислительных ресурсов конечного устройства. Однако вместе с этими преимуществами возникли и новые проблемы, связанные с усложнением логики генерации страниц, увеличением числа обращений к базе данных и ростом объема передаваемого контента. В результате время отклика стало зависеть не только от характеристик сети, но и от структуры веб-страницы.

В процессе работы веб-приложения формируются данные о производительности, которые содержат следующие метрики: на стороне клиента – время отклика, время рендеринга, значение сдвига элементов пользовательского интерфейса, а на стороне сервера – количество запросов в минуту, время задержки, загрузка процессора. Компании используют указанные метрики для оценки корректности работы и повышения качества предоставляемых сервисов. Практика показывает, что своевременный анализ подобных данных помогает сократить число отказов и повысить устойчивость приложений за счет внедрения проверенных инженерных решений.

Исследования в области производительности клиент-серверных веб-приложений опираются на ряд фундаментальных работ, авторами которых являются *Raj Jain* и *David J. Lilja*, которые заложили теоретические основы для этой области. Работы *Jakob Nielsen* и *Feng F.-H. Nah* связали метрики отклика с влиянием на пользователей. Из современных исследователей в этой области можно выделить *Patrick Meenan, Tammy Everts,* А.Е. Кучерявого, А.М. Сухова, Л. Черкасова, *Albert Greenberg*, *Kimura Hideaki*, И. Григорика.

Исследования, направленные на анализ производительности веб-приложений, играют ключевую роль в формировании методологических и инструментальных основ оценки эффективности функционирования клиент-серверных систем.

Полученные в ходе исследования результаты позволяют улучшить метрики веб-приложений, повысить надежность и предсказуемость работы клиент-серверных систем, а также усовершенствовать пользовательское взаимодействие за счет сокращения времени отклика и повышения стабильности загрузки контента. Улучшение данных аспектов веб-приложений способствует

не только удержанию сложившейся аудитории благодаря улучшению качества обслуживания, но и расширению пользовательской базы, поскольку качество программного обеспечения и его надежность являются определяющими факторами конкурентоспособности современных веб-решений.

Объектом исследования является быстродействие веб-приложений на клиентской стороне, рассматриваемое в контексте анализа производительности в условиях взаимодействия с сервером.

В качестве **предмета исследования** рассматриваются совокупность методов и алгоритмов, применяемых для сбора данных метрик, характеризующих быстродействие клиентской части веб-приложений в клиент-серверной архитектуре.

Цель работы заключается в снижении времени отклика клиент-серверных веб-приложений на основе разработанных моделей и методик сбора данных путем создания технических решений, обеспечивающих получение и интерпретацию информации, полученной в виде метрик.

Для достижения поставленной цели решен ряд **задач**.

1. Проведен обзор существующих решений, методов анализа и оценки производительности клиент-серверных веб-приложений.

2. Разработана математическая модель генерации веб-страницы, которая учитывает влияние таких метрик как количество строк кода, сложность программных конструкций и состояние приложения (количество подключений к базе данных, количество переменных *cookie*) на время отклика для сгенерированной веб-страницы.

3. Разработана математическая модель контента веб-страницы, описывающая влияние таких метрик контента как общий размер *HTML*-документа, количество дополнительных объектов, размер и количество *JavaScript* и *CSS*-ресурсов на время отклика.

4. Разработан метод численной оценки производительности клиент-серверных веб-приложений, учитывающий свойства клиентской части взаимодействия.

5. Разработана методика анализа производительности веб-страниц клиент-серверных приложений, основанная на интеграции метода сбора данных о производительности в программном комплексе, математических моделей и алгоритмов.

6. Выполнена верификация разработанного метода, моделей, методики и предложены пути их применения в реальных условиях использования.

В процессе выполнения исследований и разработки алгоритмического и программного обеспечения, направленных на решение задач, поставленных в данной диссертационной работе, использованы методы вычислительной математики, линейной алгебры, математического моделирования.

Научная новизна работы заключается в следующем.

1. Создан метод сбора данных о производительности клиент-серверных приложений, который реализуется с помощью программного комплекса, состоящего из отдельных масштабируемых компонентов микросервисной архитектуры, отличающийся встроенной в клиентские веб-страницы библиотекой-агентом, что позволяет снизить накладные расходы на выполнение сбора данных.

2. Разработана математическая модель генерации веб-страницы, отражающая влияние на время отклика структурных метрик веб-страниц, число строк кода, сложность программных конструкций и состояние приложения, отличающаяся формализацией синтаксических конструкций, что дает возможность количественно оценить сложность формирования страницы на сервере еще до передачи клиенту.

3. Разработана математическая модель контента веб-страницы, отличающаяся представлением контент веб-страницы активным фактором, влияющим на работу клиентской части приложения одновременно с метриками контента, что позволяет осуществить количественную оценку производительности.

4. Предложена новая методика и алгоритм оценки свойств веб-страницы по внутренним метрикам страницы, которые в отличие от аналогов, учитывающих лишь функции сетевых и серверных факторов, учитывают синтаксическую

структуру кода, динамическую логику генерации и содержимое страницы, что дает возможность получения единого и полного критерия оценки.

Соответствие паспорту научной специальности. Область научного исследования, регламентированная в паспорте специальности 2.3.8. – Информатика и информационные процессы, включает следующие направления:

– разработка компьютерных методов и моделей описания, оценки и оптимизации информационных процессов и ресурсов, а также средств анализа и выявления закономерностей на основе обмена информацией пользователями и возможностей используемого программно-аппаратного обеспечения (п. 1);

– разработка архитектур программно-аппаратных комплексов поддержки цифровых технологий сбора, хранения и передачи информации в инфокоммуникационных системах, в том числе, с использованием «облачных» интернет-технологий и оценок их эффективности (п. 9).

Теоретическая значимость. Совершенствование методов, моделей и метрик для сбора и анализа данных о производительности клиент-серверных веб-приложений.

Практическая ценность. Применение предложенных в диссертации методов, моделей и алгоритмов помогает выявить факторы, негативно влияющие на время отклика клиент-серверных веб-приложений. Используя предложенные методы, модели и метрики, разработчик сократит затраты времени на улучшение работы клиент-серверных приложений, а также их поддержку.

Реализация и внедрение результатов работы. Разработанные методы и алгоритмы внедрены в учебный процесс на кафедре «Программирование» ФГБОУ ВО ПензГТУ и используются при подготовке студентов по направлениям бакалавриата 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия» в рамках дисциплин «Основы *DevOps* и *DataOps*», «*Я*зыки интернет программирования», «Сети и телекоммуникации», «Администрирование информационно-коммуникационных систем», «Компьютерные сети», а также при подготовке студентов по направлению магистратуры 09.04.04 «Программная инженерия» в рамках

дисциплин «Распределенные системы обработки информации», «Протоколы вычислительных сетей». Отдельные программно-технические решения, созданные в ходе диссертационного исследования, использованы в АО «НПП «Рубин» (г. Пенза) при модификации специального программного обеспечения автоматизированной системы управления материально-технического обеспечения, в АО «Радиозавод» (г. Пенза) при разработке и поддержке программного обеспечения производственного процесса.

Достоверность результатов работы подтверждается их непротиворечивостью результатам других исследователей, внедрением в практическую и научно-исследовательскую деятельность ряда организаций, а также апробацией и одобрением на всероссийских и международных научных конференциях.

На защиту выносятся:

1. Метод сбора данных о производительности клиент-серверных веб-приложений, который реализуется с помощью программного комплекса и позволяющий фиксировать ключевые метрики производительности в реальном времени, организовывать их централизованный сбор и последующую агрегацию без необходимости модификации серверной логики анализируемых приложений. Программный комплекс обеспечивает снижение времени отклика веб-страниц в среднем на 30% по сравнению с традиционным методом, основанным на прокси-сервере, а также его применение позволило снизить время на диагностику проблем на 13,7% в АО «Радиозавод» (г. Пенза).

2. Математическая модель генерации веб-страницы, позволяющая количественно оценивать вычислительную сложность формирования страницы на стороне сервера с помощью метрик числа строк кода, сложности программных конструкций и состояния приложения, а также выявлять потенциально узкие места при ее создании и определять на этапе проектирования влияние серверной нагрузки на время отклика.

3. Математическая модель контента, количественно описывающая влияние метрик, связанных с контентом страницы: общий размер *HTML*-

документа, количество дополнительных объектов, размер и количество *JavaScript* и *CSS*-ресурсов, позволяющая оценивать влияние наполнения веб-страницы на ее производительность.

4. Методика анализа производительности веб-страниц и алгоритм на ее основе, позволяющие реализовать непрерывный контроль производительности веб-приложений, автоматизировать сбор и интерпретацию данных в процессе их использования.

Апробация работы. Основные результаты, полученные в рамках диссертационного исследования, были опубликованы в научных журналах и апробированы на международных и всероссийских научных конференциях: V Всероссийская научно-практическая конференция «Актуальные вопросы современной науки: теория и практика научных исследований» (Пенза, 2021); Международная научно-практическая конференция разработчиков веб-технологий «*Web guild conference*» (Цюрих, 2021); Международная научно-практическая конференция «Современные информационные технологии» (Пенза, 2020, 2023, 2024, 2025); Международная научно-практическая конференция «Проминжиниринг» (*ICIE*) (Сочи, 2025); XXIV Всероссийская научно-практическая конференция «Молодые учёные России» (Пенза, 2025).

По результатам диссертационного исследования опубликовано 15 научных работ, в том числе 5 статей в журналах, рекомендованных ВАК Минобрнауки России, 1 статья, индексируемая в международной базе данных *Scopus*, получено 2 свидетельства о государственной регистрации программ для ЭВМ.

Личный вклад автора. Все представленные в работе результаты исследования являются оригинальными и были получены автором самостоятельно. Данные, заимствованные у других авторов, сопровождаются ссылками на соответствующие опубликованные источники.

Объем и структура диссертации. Работа состоит из введения, четырех глав, заключения, списка литературы, который включает 115 наименований, и 2 приложений. Общий объем диссертации составляет 143 страницы. Диссертация содержит 23 таблицы и 23 рисунка.

1. ОБЗОР И АНАЛИЗ ПОДХОДОВ К СБОРУ ДАННЫХ О ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

В современном цифровом мире бизнес и производства активно внедряют цифровые программные продукты и системы для автоматизации процессов через сеть интернет. Успех компаний напрямую связан с качеством используемого программного обеспечения. Оно является важным активом для разработчиков и специалистов в области бизнеса и клиентского сервиса.

Передовые программные продукты и компьютерные системы, функционирующие на основе веб-технологий, охватывают множество заинтересованных сторон, среди которых разработчики, заказчики, пользователи, а также компании, предоставляющие высококачественные *IT*-решения для других компаний (*B2B*). Для создания современного надежного программного обеспечения важно учитывать современные нужды рынка информационных систем, проводить их тестирование (функциональное, тестирование безопасности, приемочное), а также не менее важно проводить проверку качества, особенно в системах, основанных на программных решениях. Перед проектированием любой системы определяем необходимые качественные атрибуты, ориентированные на достижении целей заинтересованных сторон и выполнение функциональных задач. Такие атрибуты включают характеристики качества, связанные как с программной системой и данными, так и с взаимодействием пользователей с ней.

При разработке современных систем следование актуальным требованиям и стандартам в области качества программного обеспечения является важнейшим приоритетом для архитектора и разработчика программного обеспечения. Для них основной целью является то, как предоставить на выходе продукт, максимально соответствующий таким стандартам. Одним из таких стандартов является ГОСТ Р ИСО/МЭК 25010–2015, содержащий набор заранее предустановленных характеристик компьютерных систем. В них входят эффективность, производительность, удовлетворенность, свобода от риска, покрытие контекста (рисунок 1.1).

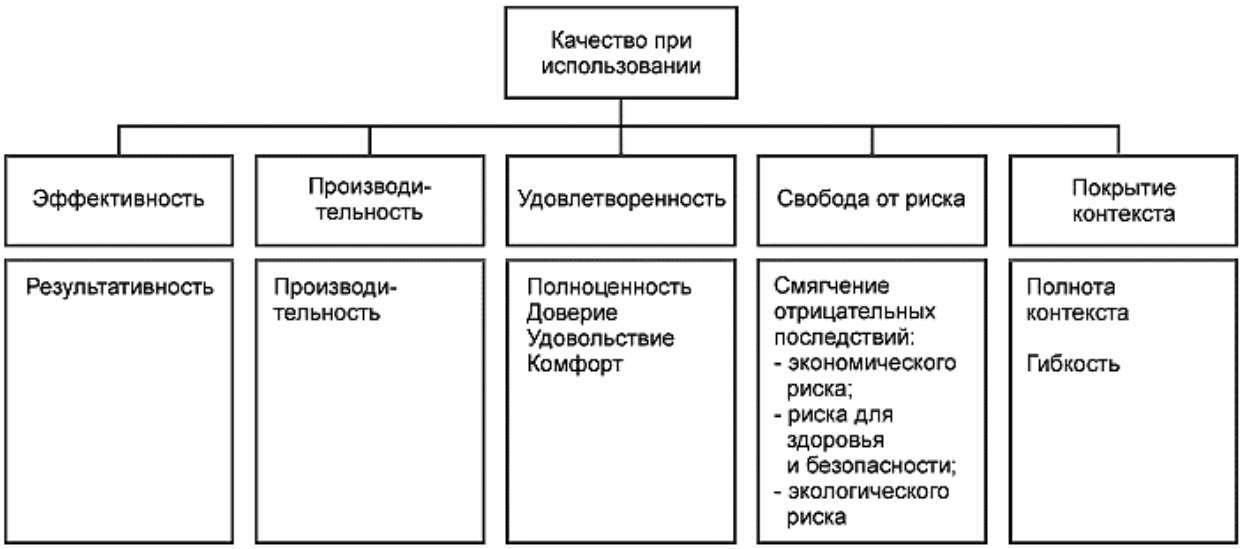


Рисунок 1.1 – Модель качества при использовании современных программных систем

Следует подчеркнуть, что эксплуатационные характеристики программной системы оказывают непосредственное влияние на заинтересованные стороны. Это влияние формируется под действием совокупности факторов, включая свойства программного обеспечения, конфигурацию аппаратных компонентов, особенности операционной среды, а также характеристики пользователей и специфику решаемых задач. Эти элементы в комплексе обуславливают уровень качества функционирующей системы и степень ее соответствия требованиям заинтересованных сторон. Хотя конкретные требования к качеству могут варьироваться в зависимости от целей и ожиданий различных категорий пользователей, в контексте веб-приложений особую значимость приобретает такая характеристика, как производительность. Она является ключевым критерием оценки эффективности веб-приложений, функционирующих на основе протокола прикладного уровня *HTTP*.

Протокол передачи гипертекста (*HTTP*) пережил несколько обновлений и модификаций, а архитектура веб-приложений стала намного сложнее. С каждым годом разработчики и архитекторы программного обеспечения реализовывают технологически-сложные сервисы приложения. Так, например, на серверной части веб-приложений стало популярным использование

микросервисного подхода в разработке, помогающего добиваться большей устойчивости, лучшего масштабирования и модульности. Кроме этого, вместо статических сайтов стали популярны клиент-серверные приложения, использующие рендеринг на стороне сервера. Следует отметить, что серверный рендеринг веб-страниц – не новое явление: он активно применялся на ранних этапах развития интернета, когда *HTML*-контент формировался на сервере и передавался клиенту в виде готовой страницы. Однако с распространением архитектуры одностраничных приложений (SPA), которая переносит основную нагрузку на клиентскую часть (толстый клиент) и обеспечивает более динамичное взаимодействие, интерес к серверному рендерингу на время ослаб. Тем не менее, в условиях роста требований к скорости загрузки, поисковой оптимизации (*SEO*) и пользовательскому опыту, подход к рендерингу на стороне сервера вновь приобретает актуальность. Особенно это заметно в рамках современных фреймворков, таких как *Next.js* или *Nuxt.js*. Возвращение к концепции *SSR* можно рассматривать как эволюционную адаптацию, направленную на достижение баланса между производительностью, гибкостью и удобством использования веб-приложений.

Каждое современное веб-приложение в процессе своей работы порождает десятки потоков. Использование многопоточного подхода широко распространено в современных программных решениях. Высоконагруженные системы разбивают работу на большое количество задач, которые должны выполняться параллельно. Наряду с этой тенденцией в исследованиях корпоративные программные системы и клиент-серверные приложения также опираются на парадигмы параллельной обработки информации и большой кодовой базы. Хорошим примером являются современные веб-браузеры, которые имеют как многопроцессорную архитектуру для изоляции задач в целях безопасности и предотвращения поломки приложений, так и многопоточную архитектуру для распределения задач рендеринга между ядрами центрального процессора.

Основной проблемой для комплексных и параллельно работающих сред является высокая степень неоднородности характеристик производительности, которая обусловлена взаимодействием различных компонентов, особенностями выполнения кода и трудностями в идентификации первопричины деградации производительности. Для решения этой проблемы широко используется статический анализ кода. Он позволяет выявлять узкие места производительности приложений. Однако такой подход имеет определенные недостатки, а именно такой анализ может давать ложные срабатывания, если код написан без применения лучших практик. Очень важно учитывать существенный факт: статический анализ не заменяет тестирование, а является его дополнением.

Частые релизы программного обеспечения и его обновления затрудняют и усложняют тщательную и подробную оценку производительности обновленного приложения. Попадание некачественного кода в финальную сборку программного продукта приводит к снижению производительности веб-приложения, что, в свою очередь, обусловливает увеличение времени отклика системы. Данный фактор способствует возрастанию эксплуатационных затрат и усложнению процесса технического сопровождения программного обеспечения в организации.

Для детального представления картины быстродействия веб-страницы недостаточно лишь профилирования кода во время его выполнения. Современные инструменты как *Lighthouse* или *JMeter*, предоставляют такие метрики как, например, задержки, потребление ресурсов и сетевую активность. Они, как правило, ориентированы на внешние индикаторы, отражающие текущее состояние производительности системы. Однако они не объясняют, какие структурные характеристики самой веб-страницы (количество дополнительных объектов, программные конструкции, количество строк кода, наличие *cookie* и так далее) влияют на показатели времени отклика. Поэтому анализ производительности должен включать моделирование внутренних параметров страницы, а не полагаться исключительно на классическое профилирование. Такой подход позволяет выделить причинно-следственные связи, например: как именно избыточное использование встроенных объектов

или тяжелые запросы к базе данных увеличивают время отклика, независимо от текущей сетевой среды.

В первые годы существования вычислительной техники производительность приложений напрямую зависела от ее комплектующих. В качестве примера можно привести центральный процессор, который и по сей день является важной деталью компьютера. Важными показателями производительности в этом случае являются тактовая частота, количество выполняемых операций в секунду. Большинство этих показателей относятся к эффективности оборудования. Быстрое совершенствование и развитие компьютерного оборудования привело к тому, что оно стало способно поддерживать программное обеспечение, которое имеет в основе сложные абстракции для предоставления разработчикам высокоуровневые конструкции для разработки [1]. Однако, с ростом сложности самого программного обеспечения, его производительность становится столь же значимой, как и производительность аппаратного обеспечения.

При функционировании современных веб-приложений как аппаратные, так и программные компоненты оказывают значительное влияние на общую производительность системы [2]. Однако при анализе клиентской части важно учитывать не только такие ключевые метрики, как время ответа сервера или объем переданных данных, но и такую метрику как восприятие пользователя [3]. Эта метрика отражает субъективное ощущение скорости и отзывчивости интерфейса, напрямую влияющее на удовлетворенность пользователя и качество системы в целом. Восприятие времени отклика зависит не только от фактической скорости работы системы, но и от таких факторов, как анимации, размера исходных файлов приложения, типа соединения и так далее. Поэтому, даже если метрики производительности улучшаются, пользователь может замечать медленную работу пользовательского интерфейса, так как отсутствует плавность в работе системы [4]. Иными словами, это описывает как быстро, по мнению пользователя, интерфейс реагирует на нажатия, ввод данных и другие взаимодействия. Поэтому исследование производительности веб-приложений

требует системного подхода, в котором технические характеристики дополняются анализом влияния клиентской структуры и логики интерфейса на восприятие времени отклика.

На основании проведенного анализа можно сделать вывод о том, что производительность остается критически важным аспектом, и вопросы в данной области приобретают дополнительную сложность. В то же время интернет, являясь одним из наиболее значимых изобретений современности, демонстрирует стремительный рост количества пользователей, который в свою очередь растет в геометрической прогрессии [5]. Данные факторы оказывают существенное влияние на требования к качеству в области современного программного обеспечения для веба. В результате вопросы производительности веб-приложений в последние годы привлекают большой интерес исследователей и разработчиков программного обеспечения [6].

Проблемы производительности представляют собой одну из существенных трудностей для веб-индустрии. Основной целью владельцев веб-сервисов является привлечение большего числа пользователей, повышение их вовлеченности, увеличение времени пребывания на клиентском веб-приложении, стимулирование использования предоставляемых услуг и совершения целевого действия. Низкие показатели производительности могут стать критическим ограничением на пути реализации поставленных задач. Одним из критических аспектов производительности веб-приложений является время отклика [7].

Для проведения анализа производительности необходимо выбрать соответствующие методы, которые позволяют эффективно оценить метрики системы. В рамках данной работы использовались математическое моделирование и эмпирический сбор данных о метриках существующих систем.

Моделирование представляет собой процесс разработки математической модели, которая является формализованной копией исследуемой системы. Этот подход позволяет получить общее представление о производительности системы, фокусируясь на ее ключевых компонентах и факторах, оказывающих наибольшее

влияние на функционирование. Результаты, получаемые посредством аналитического моделирования, обладают меньшей степенью точности по сравнению с данными, полученными в результате эмпирического сбора данных [8].

Для проведения моделирования требуется разработка модели процесса или системы, которая представляет собой специализированную компьютерную программу. Такая модель фиксирует лишь те метрики, которые являются важными для достижения целей работы. Подобное упрощение поведения системы, реализуемое в процессе моделирования, способствует облегчению проектирования, разработки и модификации модели. Однако это упрощение также накладывает ограничения на точность результатов, так как некоторые детали поведения системы могут быть исключены из модели.

Эмпирический сбор данных представляет собой процесс сбора данных, которые содержат информацию о метриках исполнения приложения. Данный подход обеспечивает более точные результаты по сравнению с моделированием, поскольку основан на фактических показателях, получаемых в реальных условиях.

1.1 Протоколы сети-интернет

Появление веба стало важным событием, которое сделало интернет доступным для широкой публики, а появление графических браузеров способствовало привлечению пользователей, которые не были из исследовательских сфер. Интернет является сетью, основанной на принципе коммутации пакетов, которая в настоящее время является преобладающей парадигмой связи в области компьютерных сетей и телекоммуникаций [9, 10]. В сетях с коммутацией пакетов данные передаются как отдельные пакеты между узлами, используя коммутаторы, повторители сигнала, мосты и маршрутизаторы. Пакеты перемещаются по каналам передачи данных, которые разделяются с другими видами трафика, обеспечивая эффективное использование сетевых ресурсов. Пакет представляет собой структурированный блок данных, который передается по сети. Использование

такого подхода дает крупным сообщениям возможность быть разделенными на небольшие фрагменты, что повышает надежность и эффективность передачи данных по сети [9]. Обычный пакет содержит три основных элемента: заголовок, полезную нагрузку и так называемая последовательность контрольных битов канала связи. Заголовок указывает на начало пакета и включает сведения, которые необходимы для его маршрутизации, проверки целостности и восстановления в корректном порядке. Полезная нагрузка хранит передаваемые данные. Последовательность контрольных битов канала связи служит для указания на завершение пакета.

Протоколы *TCP* и *IP* являются фундаментальными протоколами интернета и имеют определенные форматы заголовков [12, 13]. Актуальный стандарт для *IP* является *IPv6*, выпущенный в 1996 году Инженерной группой интернета (*IETF*) [9]. Он предусматривает использование 128-битных адресов источника (отправителя) и назначения (получателя) вместо 32-битных полей в *IPv4*. Увеличение объема адресного пространства в *IPv6* обусловлено потребностью в значительно большем количестве *IP*-адресов для современных сетевых инфраструктур.

В целом работа интернета основывается на нескольких протоколах, которые в свою очередь организованы в стек *TCP/IP*. Данный набор состоит из четырех уровней, где каждый уровень занимается решением определенных проблемы при передаче данных. Так, например, протоколы нижнего уровня отвечают за передачу цифровых данных и сигналов по физическим каналам: оптоволоконные или медные кабели. Протоколы верхнего уровня опираются на высокоуровневые абстракции, которые обеспечивают наглядное и удобное представление для конечного пользователя. Канальный уровень - описывает физическое оборудование и то, как данные передаются по физическим каналам, сетевой уровень определяет *IP*-адреса и схемы маршрутизации для передачи пакетов с одного *IP*-адреса на другой, транспортный уровень - отвечает за доставку данных и установку *TCP* соединения, а прикладной уровень обеспечивает взаимодействие на высоком уровне посредством *HTTP* протокола

[9, 11]. В дальнейшем стек протоколов *TCP/IP* эволюционировал в модель, состоящую из пяти слоев [10].

Оригинальный четырехуровневый набор протоколов интернета в дальнейшем развился в пятиуровневую модель, которая разделяет канальный уровень на физический уровень и уровень сетевого доступа, что соответствует физическому уровню и уровню канала передачи данных семиуровневой эталонной модели взаимодействия открытых систем, или так называемой модели *OSI*. Семь уровней данной модели включают в себя физический уровень передачи данных, сетевой, транспортный, сеансовый, представления и прикладной. В таблице 1.1 сравнивается набор протоколов *TCP/IP* и модель *OSI*.

Таблица 1.1 – Сравнение *TCP/IP* и *OSI*

<table>
<tr><td>Прикладной уровень</td><td rowspan="2">Прикладной уровень</td></tr>
<tr><td>Уровень представления</td></tr>
<tr><td rowspan="2">Уровень сессии</td><td rowspan="2">Транспортный уровень</td></tr>
<tr><td>Транспортный уровень</td></tr>
<tr><td rowspan="2">Сетевой уровень</td><td>Сетевой уровень</td></tr>
<tr><td rowspan="2">Уровень канальный</td><td rowspan="2">Канальный</td></tr>
<tr><td>Физический уровень</td><td>Физический уровень</td></tr>
</table>

Протоколы, действующие на каждом уровне сетевой модели, добавляют к отправляемым данным управляющую информацию. К уже имеющейся информации в дальнейшем добавляется управляющая информация, переданная протоколами более высоких уровней. Такая информация играет важную роль в обеспечении корректной доставки данных.

1.2 Обзор российских и зарубежных аналогов

В данной главе представлен обзор существующих отечественных и зарубежных решений, которые ориентированы на исследование и повышение производительности веб-страниц. Основной целью данного обзора является выявление актуальных тенденций, определение наиболее результативных

методов и инструментальных средств, а также анализ недостатков и ограничений современных методов сбора данных производительности веб-приложений. Изучение российских и зарубежных разработок позволит сформировать целостное представление о текущем состоянии исследуемой темы, а также определить перспективные направления дальнейших теоретических исследований и практической реализации разработанных подходов и моделей.

Первый патент *RU2669172C2* за авторством Канцева И. В. и Егорова Д. С. предлагает систему и метод мониторинга согласованности веб-сайтов, ориентированные на автоматизированное выявление ошибок в выводе данных и других эксплуатационных проблем. Изобретение базируется на анализе поведенческих моделей пользователей, которые формируются на основании их взаимодействий с веб-ресурсом. Этот подход обеспечивает точное определение потенциальных несоответствий, таких как неработающие ссылки, некорректное отображение контента или иные аномалии в функционировании веб-сайта [14]. Предлагаемое решение представляет собой существенный шаг вперед в автоматизации анализа качества веб-сайтов, преодолевая ограничения традиционных методик, например, пассивного мониторинга трафика или ручных проверок. Ключевым преимуществом изобретения является способность интеграции информации о поведении пользователей, позволяющая идентифицировать ошибки, которые могут быть пропущены альтернативными методами. Система сопоставляет наблюдаемую поведенческую модель с эталонной и определят проблему. Данный подход особенно актуален для ресурсов со сложной архитектурой, обширным контентом и высокой интерактивностью. Технология включает оценку пользовательских взаимодействий, построение соответствующих поведенческих моделей, анализ их отклонений от предполагаемой нормы, а также последующую передачу информации об обнаруженных проблемах ответственным специалистам [14]. Таким образом, обеспечивается повышение качества функционирования веб-сайтов, снижение вероятности сбоев и улучшение пользовательского опыта. В

качестве примера можно привести своевременное выявление некорректно работающих гиперссылок или форм ввода данных и последующую передачу информации об этих несоответствиях разработчикам для оперативного исправления. Наряду с достоинствами, система может сталкиваться и с ограничениями. В частности, качество мониторинга в значительной мере определяется точностью и полнотой данных, содержащихся в логах. Кроме того, внедрение системы в сложные инфраструктурные среды, например, сложные веб-приложения, может потребовать много ресурсов и участия квалифицированных специалистов. В целом, патент *RU2669172C2* демонстрирует инновационный подход к автоматизации контроля согласованности и стабильности работы веб-сайтов, предоставляя эффективный инструмент для повышения надежности и улучшения пользовательского опыта.

Следующий патент *RU2680746C2* за авторством Цзыню Ч. из Китая и опубликованный в 02.06.2019 описывает метод и устройство для формирования моделей качества веб-страниц, направленных на повышение точности их оценки [15]. Основная идея изобретения заключается в анализе признаков качества веб-страниц, извлеченных из логов поисковых систем, а также особенности поведения пользователей. Важным отличием предлагаемого подхода выступает автоматизация процесса построения модели качества на основе методов машинного обучения, что существенно повышает точность оценки по сравнению с традиционными методами, опирающимися на ручное обобщение эвристических правил. Система извлекает важные метрики пользовательской активности, включающие долю просмотров, продолжительность посещений, завершающие просмотры и навигационные клики, а также признаки, отражающие качество веб-страниц (структуру, содержание, оценки сторонних ресурсов) [15]. Эти данные применяются для создания высокоточных моделей, способных обрабатывать миллионы наблюдений, что существенно превосходит классические выборки из сотен или тысяч страниц. Подобный подход особенно актуален для крупных поисковых систем и высоконагруженных веб-платформ. К преимуществам данной

технологии относится улучшение пользовательского опыта за счет повышения релевантности результатов поиска и снижения вероятности ошибок. Автоматизированная модель позволяет выделять страницы с высокими показателями качества и релевантности, облегчая пользователям доступ к нужной информации. Кроме того, система снижает вероятность отображения некачественных или мошеннических ресурсов в поисковой выдаче, что способствует укреплению доверия со стороны аудитории, но внедрение данного решения может потребовать значительных вычислительных ресурсов, особенно в условиях обработки обширных данных в реальном времени. Кроме того, возможны затруднения при адаптации предложенного подхода для сайтов со сложной или динамически изменяющейся структурой.

Известна также разработка, направленная на повышение производительности веб-страниц через управление задержкой тегов, по патенту *RU2763000C2* [16]. Техническое решение базируется на концепции контроля состояний тегов и их асинхронного мониторинга, что позволяет блокировать или приостанавливать теги, замедляющие загрузку страницы. Использование различных состояний тегов (например, «нормальное», «помеченное», «приостановленное») позволяет контролировать их поведение. Это достигается путем получения данных времени задержки для каждого тега и изменения его состояния в зависимости от результатов мониторинга. Изобретение обеспечивает эффективное управление производительностью страниц благодаря возможности блокировки медленно работающих тегов, что предотвращает их негативное влияние на работу страницы и снижает риск полного отказа загрузки из-за зависших сторонних элементов [16]. Кроме того, система поддерживает интеграцию с внешними платформами, включая уведомления пользователей и администраторов о состоянии тегов. Это делает решение универсальным и применимым для различных сторонних модулей, таких как рекламные или аналитические теги. Пользователь может гибко настраивать пороговые значения задержки для блокировки или приостановки тегов, а также использовать алгоритмы мониторинга, адаптируясь к

конкретным потребностям. Достоинствами изобретения являются его уникальный подход к управлению медленными тегами сторонних производителей, высокая практическая применимость и широкий потенциал применения в таких областях, как электронная коммерция, реклама и аналитика. Однако решение имеет свои ограничения. Его внедрение требует изменений в существующих системах управления контентом, что может вызвать дополнительные сложности. Также могут возникнуть трудности с применением технологии на страницах с динамически изменяющимися элементами, где блокировка тегов может привести к сбоям в функциональности.

Метод сбора данных производительности, основанный на использовании прокси-сервера, вводит промежуточное звено в цепочке взаимодействия клиента и сервера. Все запросы и ответы проходят через использование «прямого прокси», через который проходят *HTTP(S)*-запросы (*WebScarab, Muffin*). Это позволяет фиксировать их временные характеристики, размеры передаваемых объектов, последовательность загрузки и другие параметры. Применение стандартизированного формата логов обеспечивает дальнейшую автоматизированную обработку данных и интеграцию с системой анализа. Данный метод обладает рядом преимуществ: он независим от реализации клиентской части и не требует модификации кода веб-приложения, при этом позволяя получать детализированную информацию обо всех этапах обмена данными.

Метод сбора данных, основанный на использовании фреймов, основывается на встраивании в веб-страницу специальных *HTML*-элементов (*iframe*), позволяющих регистрировать временные характеристики загрузки отдельных компонентов и этапов рендеринга. Таким образом, происходит измерение клиентской части и обеспечивается фиксация такой информации как время начала и окончания загрузки ресурсов, длительность выполнения скриптов. Преимуществом метода является его относительная простота интеграции и возможность получения данных непосредственно в среде браузера без необходимости модификации серверной инфраструктуры. Однако

фреймы обладают и существенными ограничениями: они не позволяют охватить весь спектр характеристик взаимодействия клиент–сервер, не фиксируют сетевые задержки и накладные расходы соединений, а также являются риском для безопасности клиентских веб-приложений.

В современных метриках *Web Vitals*, которые характеризуют восприятие пользователем отзывчивости и плавности работы пользовательского интерфейса. *Largest Contentful Paint* (*LCP*), *First Input Delay* (*FID*) и *Cumulative Layout Shift* (*CLS*) являются основными метриками, на которых основан данный набор метрик. Они получили широкое распространение благодаря простоте интерпретации и интеграции в популярные браузеры. Однако их использование имеет ряд ограничений. *Web Vitals* не позволяют напрямую учитывать структурную сложность кода веб-страницы, количество элементов контента и особенности серверной части, которые оказывают значимое влияние на производительность. В связи с этим возникает необходимость в разработке методов, обеспечивающих более детализированное и воспроизводимое измерение характеристик, что было реализовано в данном исследовании.

Сравнивая вышеописанные аналоги с текущей разработкой, описанной в диссертационном исследовании, можно отметить схожие цели применения: анализ веб-приложений, однако их подходы к решению проблемы различаются. Текущая разработка имеет комплексную методику, где сбор данных, моделирование и мониторинг объединяются в единую методику. В отличие от решений, направленных на локальные задачи (например, контроль тегов или выявление несоответствий через поведенческие модели), предложенная методика предоставляет более полную картину производительности веб-приложений на всех этапах их жизненного цикла. Она позволяет детализировать время отклика, анализировать влияние структуры кода и контент на время отклика работы веб-приложений, а также предоставлять обоснованные рекомендации для улучшения их работы.

1.3 Определение метрик производительности для серверной части веб-приложений

Корректное определение показателей производительности особенно важно, поскольку термин «производительность» охватывает широкий спектр свойств системы. Начиная с использования центрального процессора и подсистемы хранения данных до пропускной способности и задержек [2]. В контексте современных веб-приложений значимы также скорость запуска и масштабируемость, однако данные аспекты выходят за рамки настоящего исследования. В данной работе основное внимание уделено метрикам, описывающим воспринимаемое пользователем время отклика клиент-серверного взаимодействия.

В качестве релевантных метрик для сбора данных производительности веб-приложений используются метрики, рассмотренные далее.

Первой метрикой является время отклика. Это время между завершением запроса пользователем и началом или завершением выполнения системой.

Второй метрикой является пропускная способность, рассматриваемая в широком смысле. Обозначает количество операций, выполняемых компьютерной системой за единицу времени. В контексте передачи данных пропускная способность определяется как объем данных в байтах, который передается по сети за определенный промежуток времени [17].

Метрика загруженности системы отражает такое время, в течение которого компьютерный ресурс занят обработкой запросов, по отношению к общему доступному времени. Более высокий уровень загрузки свидетельствует о высокой нагрузке на ресурс и может указывать на тенденцию к снижению производительности. В то же время низкий уровень загрузки может свидетельствовать о неэффективном использовании ресурса, по причине его простаивания.

Доступность показывает время простоя или время безотказной работы системы, рассчитанное как среднее время наработки на отказ или среднее время восстановления. Среднее время работы безотказной системы отражает

продолжительность функционирования системы между случаями возникновения отказов [17]. Между отказами формируется время восстановления, которое представляет собой среднее время, необходимое для восстановления работоспособности системы после сбоя. Период бесперебойной эксплуатации представляет собой интервал времени, в рамках которого система функционирует без необходимости проведения технического обслуживания.

Существует метод выполнения сбора определенных метрик производительности на веб-сервере, основываясь на данных, полученных из его логов. Они содержат поля, которые описывают каждый запрос, отправленный браузером к серверу. Логи способны хранить информацию об *IP*-адресах, имени хоста, с которого поступил запрос, дата и время выполнения веб-запроса иди запроса к базе данных [18, 19]. Более того можно узнать методы запросов, которые были использованы клиентов в соответствии с *HTTP* протоколом.

Из подобных логов можно извлечь различную информацию для анализа производительности. К примеру, в логах существует поле, содержащее *IP*-адрес. Оно может быть использовано для анализа географического распределения пользователей веб-приложения, в то время как поле «агент пользователя» позволяет определить, какие браузеры преобладают среди посетителей [20, 21]. Лог-файлы веб-сервера могут содержать значительные объемы данных с детализированными атрибутами, которые могут быть преобразованы в значимые метрики и обеспечивают возможность для более глубокой и точной оценки пользовательской активности, способствуя получению обоснованных выводов о поведении пользователей.

Поскольку серверы хранят и записывают только отдельные запросы, то загрузка веб-страницы может привести к нескольким непоследовательным строчкам в логах. Подобные данные не предоставляют детальной информации о том, сколько времени, необходимо для загрузки веб-страницы. Исключением могут быть случаи, когда применяются эвристические методы для идентификации сеансов, основанные на серверных логах. Следует также отметить, что при ситуации, когда несколько клиентов подключены через один

и тот же прокси-сервер то, в логах они будут идентифицированы под одним и тем же *IP*-адресом прокси [21]. Дополнительно, кэшированные клиентские запросы не отображаются в серверных логах, что затрудняет полное отслеживание их активности.

Отмеченные выше ограничения не отменяют того, что обработка и анализ серверных логов остается неизменно важным методом для сбора данных производительности благодаря своей доступности и отсутствию необходимости увеличения сетевого трафика. Среди других преимуществ можно выделить возможность независимой идентификации каждого клиента. То есть без необходимости введения дополнительного инструментария на его стороне или использования дополнительных агентов в сети, а также реализацию на уровне *HTTP*, что исключает добавление лишних задержек в результате взаимодействия с протоколами нижнего уровня.

Для решения проблемы отсутствия информации о сеансах в серверных логах можно использовать дополнительные шаги для извлечения данных из логов и формирования последовательности запросов, соответствующих процессу загрузки веб-страницы. Путем введения такого понятия как основной объект. По сути, он является *HTML*-документом. При выполнении первого запроса клиентом для получения базового объекта передаются *IP*-адрес клиента и имя хоста. В дальнейшем каждый следующий запрос от клиента выполняется к дополнительным объектам. Такие объекты являются изображениями, таблицами стилей и скриптами *JavaScript* [22]. Если в запросах между клиентом и сервером задано поле *referer*, то для дополнительных объектов оно должно совпадать с унифицированным идентификатором ресурса (*URI)* базового объекта. Устанавливается порог в 60 секунд между временем загрузки основного объекта и остальными последующими запросами [23]. Запросы, превышающие этот временной порог, не считаются частью исходной веб-страницы. Этот порог является важным параметром, определяющим точность аппроксимации времени отклика. На рисунке 1.2 представлена упрощенная схема использования анализа серверных логов для оценки времени отклика.

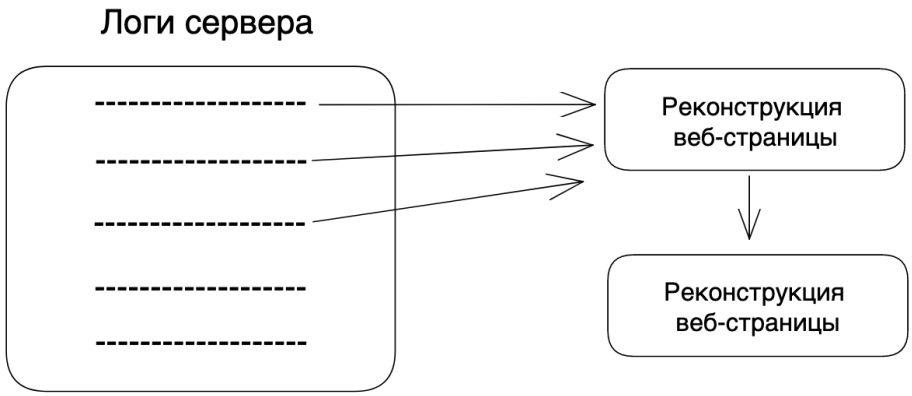


Рисунок 1.2 – Шаги для анализа логов сервера для оценки времени отклика

Использование модуля реконструкции веб-страниц позволяет объединять в группы основные объекты в логические веб-страницы и сохранять их в логах сеансов веб-страниц. Процесс реконструкции веб-страницы аналогичен процессу по извлечению данных из серверных логов, который направлен на идентификацию запросов, составляющих веб-страницу. На основе такой веб-страницы выполняется аналитический обзор. При этом точность реконструкции веб-страницы определяется применяемыми эвристическими методами и объемом данных, доступных для интерпретации из исходных данных.

1.4 Формирование метрик производительности для клиентской части веб-приложений

Веб-приложения давно заняли ведущие позиции среди наиболее часто используемых типов приложений, как для настольных, так и для мобильных устройств. Среди них выделяются клиентские, которые выполняются в браузерной среде выполнения, популярны благодаря своей кроссплатформенной совместимости и упрощенному процессу разработки. В то же время, для того чтобы конкурировать с нативными приложениями, разработанными на более низкоуровневых языках, они должны демонстрировать высокие показатели производительности [24]. Однако, несмотря на развитие веб-технологий, современные браузеры все еще показывают относительно высокое время загрузки страниц [24, 25].

Исследования показывают, что в архитектуре браузеров сохраняются критические узкие места. В результате сформировалось обоснованное мнение, что сетевое подключение перестало быть главным фактором задержек, а ключевые проблемы сместились в область клиентской части [24].

До сих пор нет единого мнения, какие модули современных веб-браузеров являются проблемными местами при работе приложений. Для определения источника накладных расходов и, чтобы дать ответы на вопросы, связанные с ухудшением работы системы, требуется анализ процесса загрузки страницы в веб-браузере. Одним из важных моментов в современных клиентских веб-приложениях является загрузка ресурсов (дополнительных объектов). Веб-приложения или веб-сайты, которые загружаются медленно, имеют более низкую популярность и больше подвержены атакам хакеров. Более того веб-браузер, который не обеспечивает достаточную скорость работы теряет свою долю рынка [24, 26].

Исследователи и разработчики, работающие в области веб-технологий, постоянно стремятся к повышению эффективности функционирования веб-систем. Это может включать как настройку параметров кэширования, так и рефакторинг кода с целью приведения его в соответствие с современными рекомендациями и лучшими практиками. Однако в условиях усложняющейся структуры веб-приложений и браузеров, реализация таких улучшений требует применения методов, способных учитывать архитектуру приложения и компонентов, задействованных в процессе загрузки и отображения страниц.

Первые веб-браузеры отображали только статические веб-страницы с документами с гиперссылками, но современные браузеры способны загружать веб-страницы с анимацией, мультимедийным содержимым и *JavaScript* для взаимодействия с пользователем. Кроме того, тенденции в клиентских веб-приложениях с момента появления *HTML5* и асинхронного *JavaScript* и *XML* (*AJAX*) изменили подход к разработке. Веб-браузеры превратились в важнейшую платформу, которая используется большим количеством современных пользователей и предустановлена во все новые устройства [27].

Важной метрикой для функционирования клиент-серверного приложения является время загрузки страницы. Время отклика – это время от начала инициированного пользователем запроса страницы до момента загрузки всего содержимого страницы. Оно напрямую влияет на пользовательский опыт и на успех проекта. Пользователи могут покинуть веб-страницу, если время, затраченное на загрузку страницы, занимает большое количество времени или даже могут прекратить использование определенного браузера или веб-сайта, если он не удовлетворяет современным требованиям. По данным компании *Google*, 53% посетителей мобильного сайта покидают страницу, которая занимает более трех секунд. В 2016 году *AliExpress* заявила, что увеличила количество заказов на 10,5% за счет того, что уменьшила время загрузки страницы [29]. На время загрузки страницы влияют два фактора: время, затрачиваемое на сетевые действия, такие как установление соединения *TCP* или выполнение поиска *DNS* и время, затрачиваемое на вычислительные действия, такие как анализ *HTML*, применение правил *CSS* и т. д.

Несмотря на значительный объем работ по анализу первопричин узких мест производительности в браузерах, среди них нет единого мнения. С одной стороны, исследователи пришли к выводу, что работа с сетью является основным источником снижения производительности, и в нескольких исследованиях изучалось влияние загрузки ресурсов на загрузку страницы в браузере [30]. Соответственно, были предложены различные реконфигурации сетевой инфраструктуры и решения на стороне клиента, чтобы уменьшить этот источник накладных расходов.

С другой стороны, более поздние исследования показали, что задачи с интенсивным использованием центрального процессора (ЦП), такие как синтаксический анализ *HTML* и манипулирование *DOM*, вносят более значительный вклад во время загрузки страницы. По этой причине исследователи пытались улучшить производительность различных этапов рендеринга страницы. Разработчики браузеров также распараллеливают работу

браузера, которая требует больших вычислительных ресурсов, и тонко настраивали параллелизм, чтобы смягчить замедление загрузки страниц [30].

Более того, современные браузеры эволюционируют и становятся все более и более сложными с точки зрения, как архитектуры, так и организации кода. Они постоянно выполняют вычисления и сетевые операции в различных потоках. В некоторых случаях и в нескольких процессах одновременно. Взаимозависимость между этими операциями определяет критический путь рендеринга, который слабо поддается анализу.

Как было отмечено ранее, имеется множество метрик, предназначенных для описания различных аспектов производительности системы. Выбор определенной метрики для применения в системе зависит от ее метрик и целей проведения анализа. Учитывая тот факт, что процесс функционирования веб-приложений напрямую связан с их откликом на действия пользователя, лучшей метрикой для анализа производительности веб-системы является такая метрика, которая отражает эффективность работы системы с точки зрения пользователя. Единственным релевантным показателем в данном случае является время отклика.

1.5 Механизм рендеринга современных браузеров

За последние 10 лет на рынок браузеров вышло несколько продуктов с богатым функционалом, современным пользовательским интерфейсом и уровнями безопасности. Вне зависимости от их внешнего вида и скорости работы, они преимущественно используют одну и ту же архитектуру и рабочий процесс для рендеринга веб-страниц. Основным программным модулем веб-браузеров является механизм рендеринга (также известный как механизм компоновки или рендеринга). Он преобразует содержимое веб-страницы в его визуальное представление. *Firefox* и *Microsoft Internet Explorer* (*IE*) имеют свои собственные механизмы, называемые *Gecko* и *Trident*. Новый браузер *Microsoft, Edge* использует *EdgeHTML* (ответвление от *Trident*) [31, 32]. Остальные

известные браузеры, такие как *Google Chrome, Opera и Safari*, разработаны на основе механизма рендеринга *Webkit* [33].

На рисунке 1.4 показано, как браузеры загружают и отображают веб-страницы. Процесс начинается с того, что пользователь вводит *URL*-адрес в строке браузера. Сразу после этого загрузчик ресурсов браузера инициирует *HTTPS*-запрос на получение основного *HTML*-файла с веб-сервера. Как правило, загрузчик ресурсов загружает этот файл постепенно, чтобы максимизировать перекрытие сетевой задержки с обработкой полученных фрагментов. Рисунок 1.3 демонстрирует внутренний рабочий процесс загрузчика ресурсов. Когда загружается первый фрагмент *HTML*, механизм рендеринга начинает синтаксический анализ тегов *HTML* и построение объектной модели документа или *DOM*, который является промежуточным представлением содержимого страницы. Демонстрируется в виде древовидной структуры.

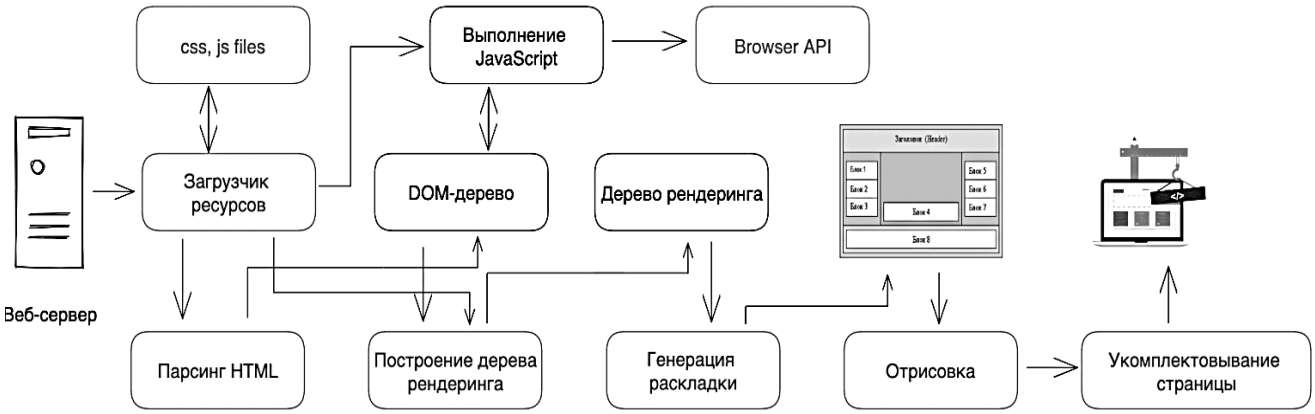


Рисунок 1.3 – Критический путь рендеринга

Анализ синтаксиса *HTML* является первым этапом вычислений в критическом пути рендеринга. Во время построения *DOM* синтаксический анализатор *HTML* может запросить дополнительные ресурсы, такие как другой *HTML*-файл, *CSS*-файл, *JavaScrip*-файл, изображения и так далее. Для каждого запроса загрузчик ресурсов может применить поиск через сервер доменных имен (*DNS)* и установить соединение *TCP* для загрузки объекта с сервера или получить объект непосредственно из кэша (рисунок 1.4).

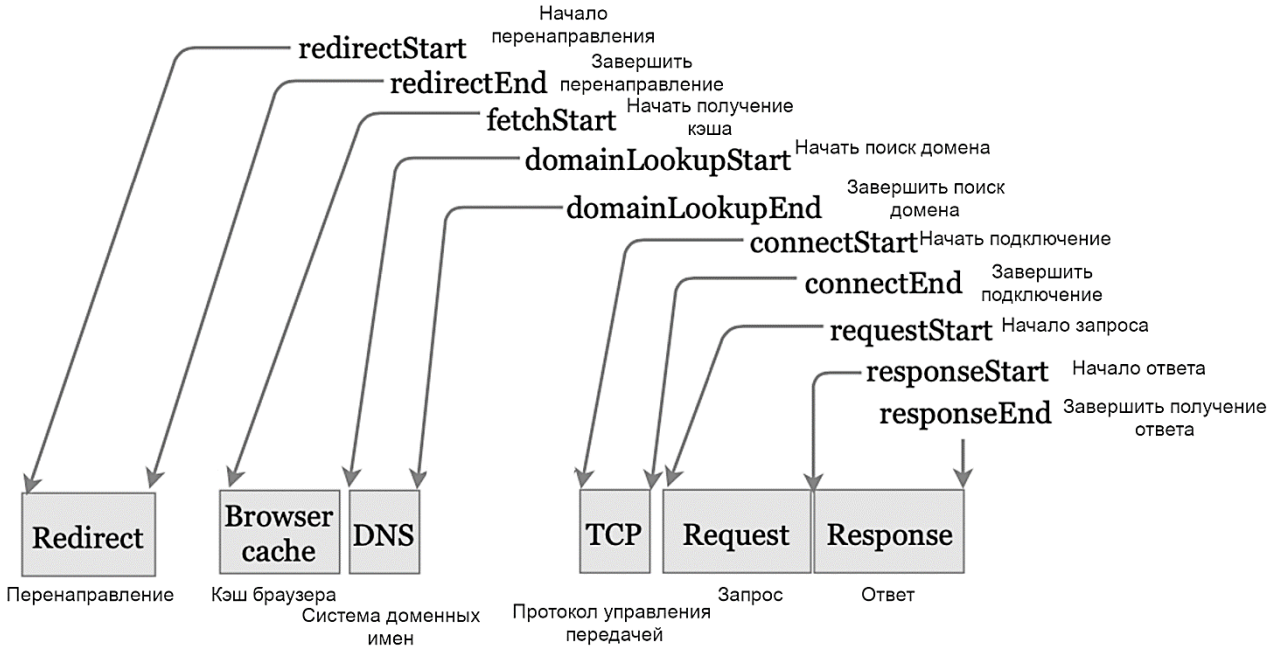


Рисунок 1.4 – Схема, показывающая атрибуты времени, определенные в интерфейсе *PerformanceTiming* и *PerformanceNavigation*, которые влияют на производительность в браузере

Среди ресурсов, которые участвуют в создании критического пути рендеринга, существуют каскадные таблицы стилей (*CSS*), содержащие набор правил, определяющих формат и атрибуты таких элементов страницы как шрифт и цвет текста [34]. Браузер анализирует эти правила и добавляет атрибуты стиля к узлам *DOM*. Этот этап, называемый «Построение дерева рендеринга», приводит к построению другого дерева, называемого деревом рендеринга. Узлы в дереве рендеринга – это визуальные элементы с описаниями стиля для отображения. На третьем этапе, «Генерация раскладки», выполняется обход дерева рендеринга для расчета относительного размера и геометрического положения элементов на экране. Четвертым этапом критического пути рендеринга является «Рендеринг», который представляет собой процесс отображения каждого визуального элемента в пикселях. Заполнение пикселей часто выполняется в несколько слоев. В конце процесса эти слои объединяются для создания финального вида веб-страницы. *JavaScript* или в целом сценарии – это еще один этап вычислений в браузере, который реагирует на действия пользователя и обрабатывает динамическое поведение

веб-страницы. Этот этап состоит из оценки, компиляции и выполнения сценариев и, обычно, имеет отдельную среду выполнения, такую как *V8* в *Google Chrome* или *SpiderMonkey* в *Mozilla Firefox* [35, 36]. Стадия выполнения *JavaScript* имеет доступ к *DOM* и может изменять его в момент, когда происходит загрузка страницы.

1.6 Исследование рынка веб-браузеров

По данным *StatCounter* – *Chrome* является самым популярным веб-браузером, используемым как для настольных компьютеров, так и для мобильных устройств. По состоянию на май 2021 года на него приходится 64,7% доли рынка браузеров, и ни один другой браузер не приближается к нему. В частности, *Safari* от *Apple* занимает второе место с долей рынка 18,4%, а *Firefox* сильно отстает с долей рынка всего 3,4% [37].

Большинство современных браузеров основаны на движке *Chromium*, который является проектом с открытым исходным кодом. На *Chromium* базируются такие популярные браузеры, как *Google Chrome*, последние версии *Microsoft Edge, Opera*, Яндекс браузер, *Brave.*

Движок рендеринга *Chrome* – *Blink*, является ответвлением популярного движка *Webkit*. *Chrome* использует архитектуру, в которой выделяется процесс для каждого экземпляра сайта, для защиты всего браузера от сбоев или вредоносных программ на веб-страницах. В этой архитектуре основной единицей является процесс браузера. Он запускает пользовательский интерфейс и управляет вкладками. Для каждого экземпляра веб-страницы создается один процесс визуализации. Процессы *Chrome* имеют несколько потоков, которые одновременно обрабатывают рендеринг страниц, взаимодействие процессов, операции ввода-вывода.

Такие этапы пути рендеринга как стилизация и макет, выполняются в основном потоке рендерера в процессе рендерера. Однако синтаксический анализ нового *HTML*-контента занимает отдельный поток, аналогичный рендерингу. *JavaScript* также работает в основном потоке рендерера, но с

потоковой передачей скриптов (новый метод, начиная с версии *Chrome* 41) *JavaScript* анализирует скрипты в отдельном потоке. *JavaScript* также взаимодействует с потоком пользовательского интерфейса в процессе браузера, чтобы реагировать на данные, которые вводятся пользователем [38]. Он также может порождать новые потоки, называемые *Web Workers* для обработки ресурсоемких задач в фоновом режиме [39]. В дополнение к этому загрузка ресурсов и другие сетевые действия управляются потоками ввода-вывода.

1.7 Проблемы критического пути рендеринга

Между этапами критического пути рендеринга существуют взаимозависимости из-за того, что некоторые из них требуют взаимодействия с *DOM*. Например, *JavaScript* использует программный интерфейс *DOM*-дерева для вставки/изменения содержимого *HTML*. В результате этап изменения стилей не может продолжаться до тех пор, пока *DOM* не будет обновлен. Для того, чтобы обеспечить согласованность *DOM*-дерева, браузеры следуют определенным правилам. Синтаксический анализ *HTML* блокируется, когда он достигает тега *script* [38, 40]. Данный тег (в отличие от *async* и *defer*) указывает, что *JavaScript* может изменять узлы *DOM* и, поэтому браузер выполняет код *JavaScript*, а только потом возобновляет синтаксический анализ *HTML*. Это гарантирует получение синтаксическим анализатором *HTML* доступа к обновленной модели *DOM* в порядке, объявленном в контексте. В другом сценарии *JavaScript* может изменить формат стилей некоторых узлов *DOM*. Это требует от браузера завершения все имеющихся процессов *CSS* перед выполнением *JavaScript* запроса. Все зависимости ограничивают планировщик задач браузера и не позволяют динамически изменять порядок этапов. В свою очередь это влияет на время загрузки страницы [40]. Рисунок 1.5 показывает, как шаги в критическом пути рендеринга влияют на время загрузки страницы в браузере.

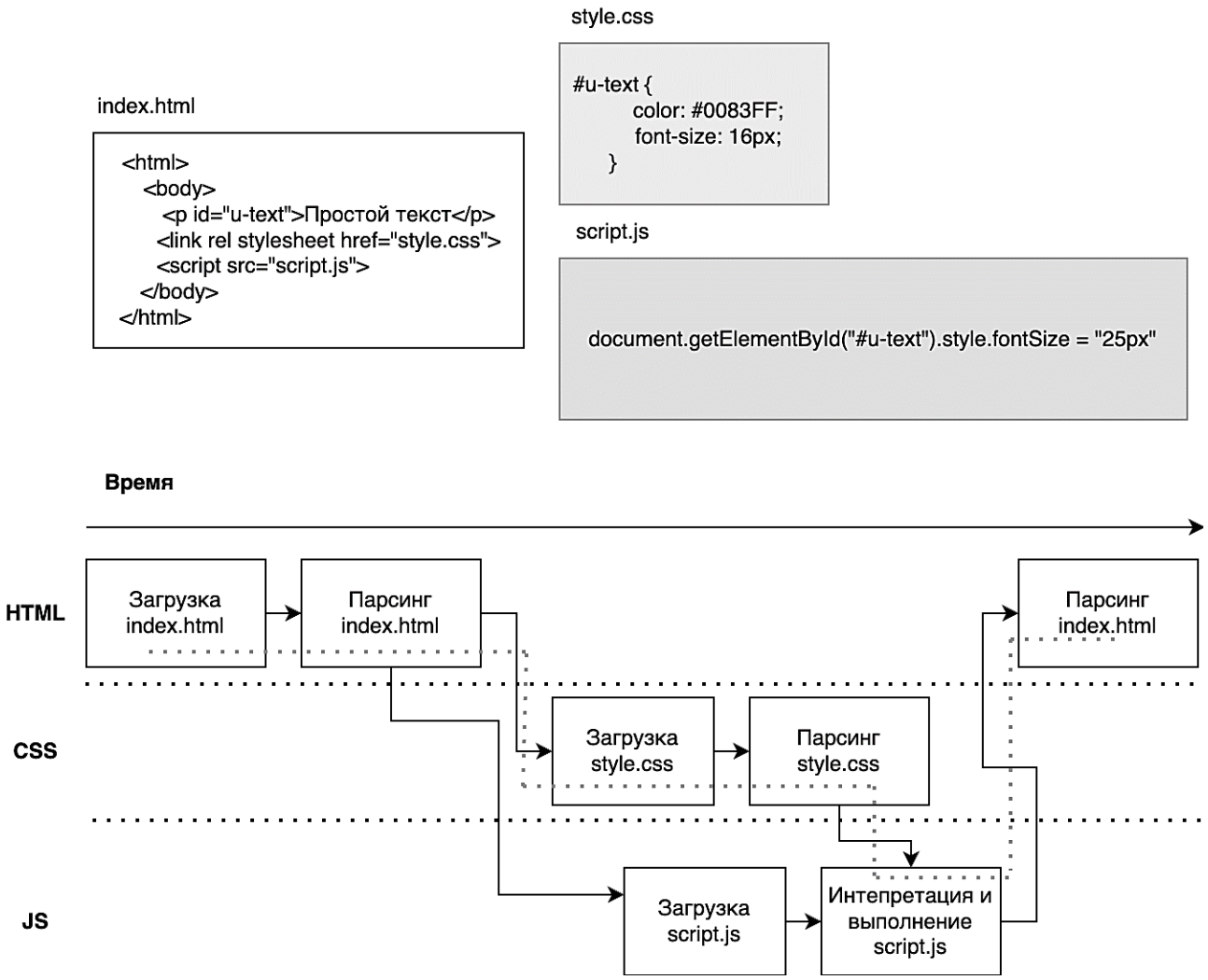


Рисунок 1.5 – зависимости между этапами рендеринга

Из приведенного примера видно, что браузер сначала загружает основной файл *HTML* (*index.html*), а затем начинает синтаксический анализ и создание модели *DOM*. Анализатор *HTML* обнаруживает внешнюю таблицу стилей *style.css* в строке 4 и начинает ее загрузку. Затем он анализирует тег *script* в строке 5, который ссылается на внешний скрипт *script.js*. Этот тег блокирует синтаксический анализ *HTML*. Однако интерпретация и оценка внешнего *JavaScript*-скрипта не может быть продолжена, поскольку обработка *CSS*-файла все еще находится в процессе. Из-за этой взаимозависимости *JavaScript* ожидает загрузки и обработки *style.css* перед переходом к следующему этапу - критического рендеринга. После того, как обработка *CSS* завершена, скрипт (*script.js*) полностью интерпретирован, и парсинг *HTML*-файла продолжается. Черные стрелки на временной шкале в нижней части рисунка 1.5 представляют

зависимости между этапами. В идеальном сценарии при отсутствии зависимостей три действия могли бы выполняться параллельно, а время загрузки страницы определялось бы самым медленным действием. Однако на практике зависимости ведут к критическому пути, показанному красной пунктирной линией. В указанном примере синтаксический анализ *HTML* и части *CSS* и *JavaScript* относятся к критическому пути [41]. Модификация данного примера путем замены строк 4 и 5 в файле *index.html* для анализа тега *script* перед тегом ссылки или изменение продолжительности действий повлияет на работу критического пути и, следовательно, на время загрузки страницы. Эти взаимозависимости между этапами делают анализ узких мест критического пути и загрузки страницы чрезвычайно сложным.

Выводы по главе

В данной главе были рассмотрены исторические и технические предпосылки возникновения интернета, а также развитие веб-технологий привело к усложнению логики браузеров и приложений. Более того были затронуты вопросы, связанные с производительностью клиент-серверных веб-приложений. В числе этих вопросов особое внимание было уделено времени отклика, с целью более глубокого понимания проблемы производительности в современных приложениях. Было отмечено, что традиционные средства профилирования не позволяют в полной мере предоставить детали снижения производительности, и что структура веб-страницы оказывает значительное влияние на восприятие времени отклика пользователем.

Также были рассмотрены современные отечественные патенты, связанные со сбором производительности веб-систем. Их анализ способствует определению аспектов, которые необходимо улучшить в рамках инструментов, позволяющих собирать данные о производительности клиент-серверных веб-приложений.

Было проведено исследование существующих клиентских и серверных метрик. Существующие метрики являются ориентиром для многих веб-разработчиков и бизнес-аналитиков в определении стандартов качества

программного обеспечения, и, которые можно использовать вновь, и адаптировать к новым реалиям современных систем для сбора данных о производительности.

Был проведен анализ рынка современных браузеров, что позволяет определить наиболее подходящее программное обеспечение в мире браузеров, которое послужит основой для проведения сбора данных и выполнения тестов. Было отмечено, что почти все популярные современные браузеры основаны на движке *Chromium*, который является проектом с открытым исходным кодом и обладает повышенной безопасностью, высокой скоростью исполнения *JavaScript* кода и архитектурной гибкостью.

В следующей главе будет проведен анализ проблем, которые будут рассматриваться в рамках настоящего исследования, а также описаны структура и методы их решения.

2. РАЗРАБОТКА МОДЕЛЕЙ, МЕТОДА, АЛГОРИТМИЧЕСКОЙ РЕАЛИЗАЦИИ МЕТОДИКИ ДЛЯ СБОРА ДАННЫХ О ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Модель является представлением реального мира. В информатике моделирование часто применяется для описания компьютерных систем, их структуры и поведения. В целом модель является максимально приблизительным представлением системы, а также она включает только те параметры, которые представляют для нее интерес, а остальные могут быть упразднены [42].

Разработанный метод, модели и методика, основанная на алгоритме, позволяют опираться на результаты, которые были получены как на клиенте, так и сервере для анализа производительности клиент-серверных приложений. В данном исследовании предлагаются модели, позволяющие объяснить взаимосвязь между временем отклика и метриками веб-страницы.

В следующей подглаве рассматривается метод для сбора данных производительности, включая время отклика. В рамках данной работы понятие «сложность» описывает вычислительные затраты на формирование и отображение веб-страницы, а также ее контента в контексте разработанных в рамке данной работы моделей. Однако для них накладываются ограничения применимости. Они связаны с классической архитектурой клиент-серверных веб-приложений, в которых *HTML*-код формируется на стороне сервера. Модели отражают взаимосвязь метрик кода и контента с временем отклика в условиях контролируемого эксперимента и стабильной сетевой среды. При этом предполагается постоянство аппаратной конфигурации сервера и клиента и исключается влияние механизмов кэширования и сетей доставки контента. Более того они учитывают количество и типы запросов к базе данных, однако не описывают такие внутренние механизмы систем управления баз данных как индексация, параллельное выполнение.

2.1 Метод сбора данных производительности с использованием программного комплекса

2.1.1 Описание метода

В рамках подглавы описывается метод сбора данных производительности клиент-серверных веб-приложений, который основан на программном комплексе. Данный комплекс основан на микросервисной архитектуре, обеспечивающей гибкость масштабирования, легкость сопровождения и возможность независимого развертывания отдельных компонентов. Более подробно разработка программного решения данного метода описана в третьей главе. Центральным элементом комплекса выступает программная библиотека-агент, реализованная в виде легковесного встраиваемого модуля на *JavaScript*.

Важной особенностью предлагаемого метода является встраивание программной библиотеки непосредственно в клиентскую часть исследуемого приложения, что позволяет осуществлять сбор данных о производительности с высокой степенью точности и с минимальным влиянием на саму систему в виде задержек. Интеграция самой библиотеки не вносит дополнительных сложностей при интеграции в клиент, так как она размещена в реестре пакетов *npm* и может быть легко установлена посредством одноименного менеджера пакетов для клиентского веб-приложения.

Библиотека-агент выполняет сбор и мониторинг ключевых метрик веба, которые отражают качество веб-системы. Среди них можно выделить метрики, которые связаны с тем, сколько времени занимает блокировка основного потока нагруженными скриптами, на то как сильно произошел сдвиг элементов, а также время отклика. Оно является критическим показателем, определяющим качество пользовательского взаимодействия с веб-приложением и его техническую и коммерческую эффективность. Подобные аспекты указывают на то, что метрика представляет наибольшую значимость для конечного пользователя, что в свою очередь делает ее не менее значимой для данного исследования.

Для реализации данного метода необходимо использовать веб-страницы, генерируемые сервером или статическим генератором сайтов с доступом к

программному интерфейсу браузера (*API*), серверные логи, фиксирующие время, затраченное сервером на обработку каждого клиентского запроса и библиотеку-агент, входящую в разработанный программный комплекс. На рисунке 2.1 представлена схема работы предложенного метода.

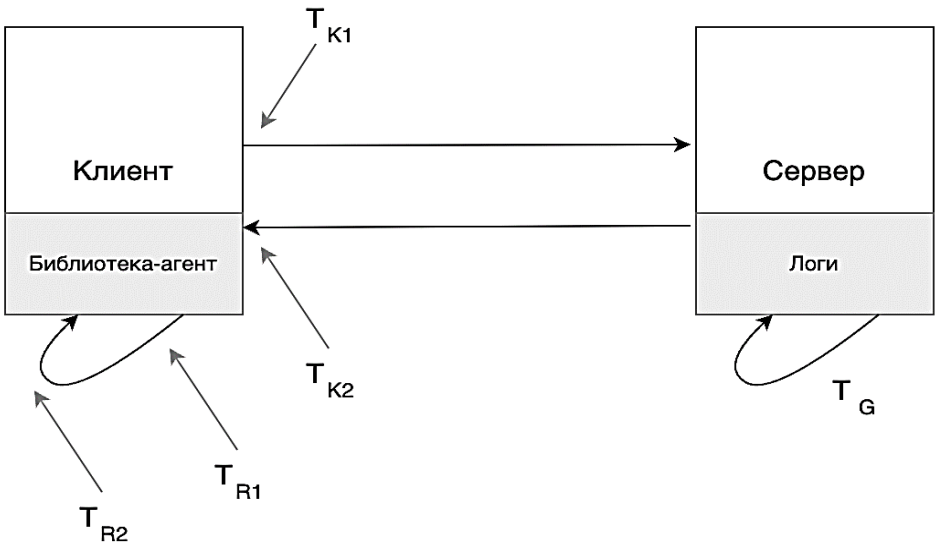


Рисунок 2.1 – Сбор данных производительности клиент-серверных веб-приложений на основе программного комплекса

Прежде всего, необходимо ввести базовые обозначения, описывающие временные метрики системы. Так временем рендеринга страницы в браузере (T_R) является отрезок времени от начала визуализации первых элементов веб-страницы в браузере до ее полной загрузки. Время генерации (T_G) – это временной интервал, в рамках которого происходит обработка всех пользовательских запросов, необходимых для загрузки сгенерированной веб-страницы. Время простоя (T_{PR}) – это время между последовательными запросами, в течение которых ни один из предыдущих запросов не находится в состоянии обработки сервером. Время обработки (T_{OBR}) – это временной интервал, в течение которого сервер выполняет обработку пользовательского запроса для загрузки конкретной *HTML*-страницы или ее дополнительного объекта. Время ответа (T_{OTV}) – это время, прошедшее с момента отправки пользователем запроса на получение веб-страницы до ее полного отображения. Время обслуживания (T_{OBSL}) – это время, отсчитываемое с момента отправки *HTML*-страницы сервером.

Временем работы сервера (T_{SERV}) является временной интервал, который устанавливается с отправки первого байта запроса пользователя и завершается в момент получения последнего байта соответствующего ответа от сервера. Время (T_{ZO}), необходимое для обмена запрошенными сервером объектами, – это временной промежуток, требуемый для полной передачи всех запросов пользователя и соответствующих ответов сервера, которые связаны с веб-страницей [44]. Взаимодействие клиента и сервера в рамках данного метода можно выразить следующим образом:

$$T_{OTVSERV} = T_{R2} - T_{K1}, \tag{2.1}$$

$$T_{RB} = T_{R2} - T_{R1}, \tag{2.2}$$

$$T_{PD} = \left(T_{K2} - T_{K1}\right) - T_G, \tag{2.3}$$

где $T_{OTVSERV}$ – время, в рамках которого было сформирована страница,

T_{RB} – время, затраченное на формирование веб-страницы,

T_{PD} – затраченное время на передачу данных,

T_{K1} – время отправки запроса на сервер,

T_{K2} – время получения ответа от сервера,

T_{R1} – время начала отображения содержимого страницы в браузере,

T_{R2} – время завершения отображения контента в браузере,

T_G – генерация ответа сервере.

При обращении клиента к серверу запрос проходит через библиотеку-агент, где фиксируется время инициализации запроса. Затем запрос отправляется на сервер для дальнейшей обработки. По завершении процесса клиент получает ответ, и библиотека-агент фиксирует время завершения запроса. Помимо сбора данных на клиенте сервер хранит систематизированные записи взаимодействия клиента и сервера в логах. В соответствии со стандартом расширенных полей файла логов *W3C*, формат файла логов должен обязательно содержать поле, которое содержит значение времени, затраченное сервером на обработку запроса [45]. Сервер отправляет клиенту ответ, и он выполняет расчет время отклика. На

клиентской части веб-приложения происходит анализ и запись времени, которые были затрачены на формирование веб-страницы с использованием программного интерфейса *Performance*.

Несмотря на то, что оценка времени создания веб-страницы и общего времени отклика не вызывает сложностей, получение данных о времени, затраченном на генерацию контента и его передачу, сопряжено с рядом особенностей:

1. С точки зрения клиента, временной промежуток между отправкой запроса и получением соответствующего ответа, который обозначается как время обслуживания T_{OBSL} в реальности включает два основных компонента: времени на обработку запроса и генерацию ответа T_{OBR} и времени на передачу запроса, и ответа по сети T_{ZO}. Таким образом, время передачи конкретной запрошенной страницы или ресурса определяется как разница между временем ответа T_{OBSL} и временем отправки запроса клиентом для данной страницы или ресурса T_{OBR} [44].

2. Сервер может приступить к отправке ответа клиенту до завершения обработки клиентского запроса, что подразумевает возможность параллельного выполнения процессов генерации и передачи страницы. Вследствие этого простое вычитание T_{OBSL} и T_{OBR} не покажет полную картину о времени обслуживания T_{ZO}.

3. В интервалах между последовательными запросами, когда предыдущий запрос не обрабатывается или не обслуживается, возникают так называемое время простоя T_{PR}. Это время может быть использовано клиентом для установки нового соединения с сервером и отправки параллельных запросов. В другом случае клиент может дождаться отображения страницы браузером перед отправкой дополнительных запросов [44]. Таким образом, вычитание T_G из времени, прошедшего между отправкой первого запроса и получением последнего байта ответа, а именно времени обслуживания на сервере T_{SERV}, дает значение, которое отражает комбинацию T_{ZO} и T_{PR}, а не только сам T_{ZO}.

4. При взаимодействии клиента и сервера возможно параллельное выполнение запросов: клиент может начать загрузку одного объекта, в то время как сервер продолжает обработку других запросов. В то же время браузер

осуществляет отображение ранее полученных ресурсов. Протокол *HTTP* позволяет использовать один и тот же сетевой канал для последовательной передачи и получения данных, не дожидаясь завершения предыдущих транзакций. Вместе с тем браузер *Google Chrome* накладывает ограничение на количество одновременно открытых соединений: не более шести подключений на одно доменное имя и не более десяти подключений в общей сложности. Это означает, что в типичных условиях браузер способен одновременно обслуживать до шести параллельных запросов к одному хосту, а оставшиеся четыре соединения могут быть использованы для запросов к другим источникам.

Подход позволяет клиентскому браузеру запрашивать параллельно несколько объектов с одного сервера. Общее времени работы сервера при обработке запроса для веб-страницы можно представить следующим образом:

$$T_{OP} = T_{PS} + T_{OBRSTRSERV}, \quad (2.4)$$

$$T_{OBSL1} = T_{ZO1} + T_{OBR1}, \quad (2.5)$$

$$T_{SERV} = T_{OP} + T_{OBSL1} + T_{PR} = T_{PS} + T_{OBRSTRSERV} + T_{ZO1} + T_{OBR1} + T_{PR}, \quad (2.6)$$

где T_{PS} – это промежуток времени, за который была передана страница, T_{OP} – это время с момента отправки и до момента получения веб-страницы, T_{SERV} – общее время, за которое произошло обслуживание запроса, T_{ZO1}– время передачи для дополнительного объекта веб-страницы, $T_{OBRSTRSERV}$ – время обработки *HTML*-страницы на сервере, T_{OBR1} – время обработки дополнительного встроенного ресурса веб-страницы, T_{PR} – время простая между выполнением запросов. Выражение позволяет оценить воздействие каждого этапа на время отклика и понять, где именно возникают проблемные места на этапе генерации, обработке вложенных объектов. При увеличении числа дополнительных объектов (i) для веб-страницы выражение 2.6 примет следующий вид:

$$T_{SERV} = \left(T_{PS} + T_{OBRSTRSERV}\right) + \sum_{i=1}^{n} T_{PERDOPRES} + \sum_{i=1}^{n} T_{OBRDOPRESSERV} + \sum_{i=1}^{n} T_{PR} \quad (2.7)$$

При анализе составляющих, представленных в выражении 2.7, следует принять во внимание, что $T_{PERDOPRES}$ – временной интервал, необходимый для

передачи одного дополнительного встроенного объекта веб-страницы, а $T_{OBRDOPRESSERV}$ – время, затраченное сервером на обработку одного дополнительного объекта страницы. $\sum_{i=1}^{n} T_{PR}$ отражает совокупное время простоя, возникающее между запросами.

В процессе исследования было обнаружено, что время генерации T_G и временные затраты на передачу веб-страницы T_{ZO} в сумме могут превосходить значение T_{OP}, что является некорректным. В результате чего был введен такой показатель как «полезное время» или T_{US}, который представляет собой разность между T_{PR} и T_{OP}. Важно отметить, что время простоя T_{PR}, является приблизительным значением времени между запросами, поскольку он фиксирует только время начала поступления ответов клиенту, но не их завершения.

Отношение T_{US} и T_{OP} определяет насколько эффективно происходит передача данных по сети. Под этим аспектом понимается совокупность метрик, таких как задержка, пропускная способность, уровень потерь пакетов и влияние используемого протокола на общее время загрузки страницы и скорость взаимодействия пользователя с интерфейсом. Небольшое значение T_{US} по отношению к T_{OP} является индикатором эффективности. Дополнительно отношение T_{ZO} и T_G дает представление о времени передачи. Чем больше значения отношения $T_{ZO} : T_G$, тем больше влияние на $T_{OTVSERV}$.

2.1.2 Сквозной сбор данных

Сквозной сбор данных производительности предполагает комплексное рассмотрение всех этапов обработки пользовательского запроса. От момента его инициирования на клиенте до формирования и доставки ответа сервером. В рамках данного подхода осуществляется мониторинг ключевых метрик как на стороне клиента (время загрузки страницы, рендеринг), так и на стороне сервера (время обработки запроса, обращения к базе данных). Это обеспечивает целостное представление о поведении веб-приложения в реальных условиях

эксплуатации и позволяет выявлять узкие места, возникающие как в клиентской, так и в серверной частях системы [50].

Блок-схема, представленная на рисунке 2.2, описывает последовательность этапов обработки данных в рамках разработанного программного комплекса для сбора данных производительности клиент-серверных веб-приложений.

Рисунок 2.2 – Сквозной сбор данных

На начальном этапе осуществляется ввод данных, который подразумевает фиксацию пользовательской активности пользователя и событий, связанных с взаимодействием с веб-системой. В процессе взаимодействия пользователя с веб-страницей инициируются события, сопровождающиеся генерацией метрик системы. Важно отметить, что данные не ассоциируются с конкретным пользователем, поэтому не затрагивает персональные данные, что исключает необходимость применения соответствующих норм законодательства. Собираемые метрики привязаны исключительно к текущему веб-приложению и конкретной странице без идентификации личности.

На стороне клиента выполняется предварительная обработка, включающая извлечение ключевых метрик библиотекой-агентом (время начала загрузки, продолжительности выполнения сценариев, получения ресурсов) и их преобразование в структурированный формат. После этого собранные данные передаются на сервер программного комплекса для дальнейшего анализа. На серверной стороне происходит агрегация и унификация данных, полученных от различных клиентов. Объединенный набор данных подвергается аналитической обработке. Заключительный этап включает вывод результатов в виде визуализаций, отчетов и рекомендаций, направленных на улучшение производительности веб-приложения. Данный алгоритм демонстрирует сквозной подход к сбору и анализу данных о производительности, который принимает во внимание как клиентскую, так и серверную стороны, и служит основой для метода сбора данных производительности, основанного на программном комплексе.

2.1.3 Достоинства и недостатки

При использовании метода сбора данных, основанного на программном комплексе и клиентской библиотеке, которая реализует логику замера на основе *JavaScript.* Их преимуществами относятся низкие временные затраты: используется только код, который фиксирует время запроса и отображения страницы. При этом передача данных и обработка запросов происходят в штатном режиме. Благодаря этому полученное время отклика практически

совпадает с фактическим временем, которое воспринимается конечным пользователем. Кроме того, данный метод отличается минимальными затратами на внедрение, поскольку не требует дополнительной настройки окружения. Вместе с тем подход имеет и ряд ограничений: корректная работа метода возможна только при условии, что в отличие от метода на основе прокси-сервера данный подход не позволяет получить более детальную информацию о переданных пакетах, так как работает на более высоком уровне в модели *OSI*.

2.2 Мониторинг времени отклика веб-страниц

В процессе мониторинга осуществляется сбор метрик системы и ее среды выполнения с последующей их обработкой. Программный комплекс использует данные метрик, агрегирует их и хранит в формате расширенных логов *W3C*, которые позволяют людям извлекать важную информацию для их системы [51]. Ее использование позволяет оперативно реагировать на отклонения значения метрик от нормы.

Несмотря на свою важную роль в цикле разработке ПО системы мониторинга имеют свои особенности. Так, например, система мониторинга, которая информирует о том, что страница имеет значение времени отклика, не превышающее порог, не гарантирует, что это значение в действительности является корректным и система работоспособна [52].

Быстрое время отклика может быть также связано с тем, что веб-сервер не способен обрабатывать клиентские запросы из-за большой нагрузки на сервер или из-за перегрузки сети или кода страницы, который содержит ошибки. Такая страница будет требовать дальнейшей модификации, при этом необходимо выявить страницы с низким значением времени отклика и предпринять меры по исправлению ситуации.

Время отклика возможно оценить на основе анализа логов сервера. Мониторинг включает в себя извлечение данных во время работы веб-приложения. Инструменты мониторинга можно классифицировать как чисто

аппаратные, чисто программные или гибридные. В целом мониторинг производительности направлен на достижение трех целей:

– сбор данных производительности системы и сопоставление полученных значений с заданными порогами для проверки их соответствия;

– выявление потенциальных или существующих проблем, а также проблемных мест в работе веб-приложения с точки зрения производительности;

– разработка набора мер, направленных на устранение выявленных проблем или уязвимостей и реализация решений для повышения общей производительности системы.

Процесс мониторинга веб-приложений тесно связан с задачами сбора данных и надежности системы. Одним из наиболее популярных методов для этих целей является активный мониторинг, который основывается на создании искусственных клиентов, размещенных в различных географических точках по всему миру [52, 53]. Эти клиенты осуществляют взаимодействие с веб-серверами для анализа их метрик, таких как доступность, надежность и время отклика [48]. Важным преимуществом активного мониторинга является возможность раннего обнаружения проблем, таких как недоступность серверов или чрезмерное время отклика. На современном рынке представлено множество компаний, предлагающих подобные услуги. Они обеспечивают веб-приложениям постоянный мониторинг и поддержку на основе собранных данных, что является неотъемлемой частью стратегии управления качеством сервисов.

2.3 Логи доступа к серверу

Современные веб-серверы генерируют и поддерживают файлы логов, содержащие информацию о выполненных операциях. Логи могут быть организованы в виде одного общего файла или нескольких специализированных файлов. Наиболее распространенными типами логов являются логи доступа и логи ошибок. Логи доступа фиксируют историю запросов к серверу, включая такие данные, как *IP*-адрес клиента, дату и время обращения, запрашиваемую страницу, *HTTP*-статус обработанных запросов,

объем переданных данных (в байтах) и идентификатор пользовательского агента [52]. Эти данные позволяют анализировать активность пользователей и метрик сетевого трафика [54]. Логи ошибок, в свою очередь, содержат записи об ошибках, возникающих в процессе обработки запросов или работе сервера. Они включают подробную диагностическую информацию, которая помогает в выявлении причины сбоев и улучшении работы серверного приложения.

Логи сервера активно используется для проведения детального анализа и извлечения необходимой информации, связанной с функционированием веб-приложения. Например, анализ логов позволяет определить количество уникальных посетителей, а также вычислить конверсионный коэффициент – долю пользователей, совершивших целевые действия, такие как покупка товаров или оформление подписки [55]. Такая аналитика предоставляет ценные данные для оценки эффективности веб-приложения, позволяет выявлять закономерности поведения пользователей и формировать стратегию повышения производительности и прибыльности системы.

Логи доступа к серверу могут содержать конфиденциальную информацию. Например список часто посещяемых веб-ресурсов или с каких браузеров выполняются запрос к серверу. Эти сведения могут быть использованы для анализа пользовательского поведения, персонализации взаимодействия или выявления потенциальных угроз безопасности. Однако из-за чувствительного характера этих данных многие организации внедряют строгие меры контроля доступа к логам серверов [56, 57]. Обычно доступ к этим файлам ограничивается ролью веб-администратора или уполномоченным административным персоналом, чтобы минимизировать риск утечек данных и соблюсти требования конфиденциальности.

Статистический анализ логов доступа к серверу предоставляет ценные данные, которые могут быть использованы для глубокого понимания поведения пользователей и проблемных мест в работе веб-приложения [57]. Такой анализ позволяет получить следующую ключевую информацию:

– популярные посещаемые страницы;

– типы используемых браузеров для сбора сведений о популярных веб-браузерах пользователей;

– количество посещений для анализа общего числа посещений и уникальных посетителей;

– временные паттерны для изучения моделей доступа в зависимости от времени суток, дней недели и других временных параметров;

– географическое распределение запросов для выявления и определения мест-источников откуда поступают запросы;

Перечисленные типы данных важны не только для администрирования и получения технической информации о системе, но также для бизнес-целей, таких как маркетинг, разработка рекламных кампаний, персонализация контента и улучшения пользовательского опыта [55].

Проблема разработки инструмента анализа логов доступа к серверу для получения этих данных заключается в том, что каждый сервер имеет свой собственный формат для логов. Это означает, что инструмент должен быть адаптирован для различных форматов логов или необходимо создать специальные инструменты поддерживающих разные форматы. Чтобы решить эту проблему, Консорциум Всемирной паутины (*W3C*) предложил общий формат файла логов, который содержит информационные поля, описанные в таблице 2.1 [58].

Таблица 2.1 – Структура общих полей формата логов *W3C*

Поле	Описание
remotehost	Имя удаленного хоста (или *IP*-адрес, если имя хоста *DNS* недоступно или если *DNSLookup* выключен
rfc931	Метод аутентификации *rfc931* использует *RFC931* для получения имени пользователя и повышает надежность правил контроля доступа на основе имени хоста
authuser	Имя пользователя, под которым он аутентифицировал себя
date	Время, когда был выполнен запрос
request	Строка запроса
status	*Http*-статус, который пришел клиенту
bytes	Размер данных в байтах

Разработка инструментов для анализа логов доступа к серверу и извлечения данных из них имеет ряд сложностей, главной из которых является разнообразие форматов логов, используемых различными серверами. Каждый сервер имеет уникальный формат записи логов, что требует либо адаптации анализирующего инструмента под каждый конкретный формат, либо создания универсальных инструментов, поддерживающих широкий спектр форматов. *Common Log Format* содержит унифицированный набор информационных полей, обеспечивая совместимость между различными системами и упрощая обработку данных. Поля, включенные в стандарт *CLF*, описаны в таблице 2.1. Они включают такие параметры как *IP*-адрес клиента, временная метка, метод запроса, код состояния *HTTP* и объем переданных данных. Использование стандартизированного формата существенно упрощает разработку аналитических инструментов, снижает затраты на их интеграцию и способствует унификации процессов мониторинга и анализа веб-серверов [58].

Несмотря на то, что использование общего формата логов упрощает разработку инструментов для анализа логов доступа к серверу благодаря своему формату, он не является гибким и содержит набор строго определенных полей. Расширенный формат логов помогает решить данную проблему. Для расширенного анализа клиент-серверных веб-приложений он включает два дополнительных поля *referer* и *user_agent* [45].

Поле *referer* содержит *URL*-адрес ресурса, с которого был инициирован запрос к серверу. Это позволяет отслеживать источники трафика и анализировать переходы пользователей. В свою очередь, поле *user_agent* предоставляет информацию о программном обеспечении клиента, в частности о версии веб-браузера и платформе, которые используются при выполнении запроса [45]. В первую очередь такая информация важна для определения характеристик клиентских устройств, а также может быть использована для анализа совместимости и улучшения работы веб-приложений.

Целью расширенного формата файла логов является обеспечение сбора широкого спектра данных для достижения таких целей как гибкость при работе

с логами и обеспечение большего количества собираемых данных. Заголовок, указывающий типы записываемых данных, приписывается в начале каждого файла логов. Это позволяет использовать настраиваемый формат файла логов и считывать с помощью универсальных анализаторов.

Расширенный формат файла логов предоставляет дополнительные поля, позволяющие точно идентифицировать стороны, участвующие в транзакции, включая клиент и сервер. Общий формат файла логов не имеет строго определенных символов для разделения полей, что допускает их использование внутри самих полей [59]. Это приводит к потенциальной неоднозначности при обработке данных и затрудняет работу анализаторов файлов логов, снижая их точность и усложняя разработку инструментов для анализа. Более того, для записи непечатаемых или для символов, которые не являются *ASCII*-символами, происходит экранирование для того, чтобы их можно было использовать в полях или в *URL*. Одним из методов экранирования символов является замена символа его шестнадцатеричным значением и добавление к нему другого символа, например, знака процента «%». Факт отсутствия четко определенных рекомендаций по экранированию символов в общем формате файлов логов, значительно осложняет их обработку. С другой стороны расширенный формат файла логов предоставляет четкие спецификации для полей, разделителей и способов их записи, что позволяет эффективно справляться с проблемой неоднозначной интерпретации данных и упрощает последующий анализ данных [58]. Кроме того, он поддерживает обработку унифицированного идентификатора ресурса (*URI*) для идентификации *HTTP*-сеансов, что упрощает группировку *HTTP*-транзакций для последующего анализа [11, 60]. В дальнейшем это позволяет исследовать такие показатели, как количество уникальных пользователей, посещающих веб-сайт, их поведение, включая просмотр или чтение содержимого. Такой подход помогает учитывать каждый запрос и хранить информацию о его состоянии на серверной стороне. Это является важным моментом в данной работе, потому протокол *HTTP* не хранит никакого состояния о сеансах пользователей, что значит при клиент-

серверном взаимодействии каждый запрос рассматривается как независимый [61]. Это позволяет сделать вывод о том, что невозможно узнать информацию о сеансах пользователей.

Следует отметить, что в расширенном файле логов существуют уникальные поля при формировании логов. Одно из таких полей – *count*. Оно содержит значение, которое отражает количество наступивших событий. *Time-from* и *time-to* позволяют указать интервалы, в течение которых выполняется сбор данных в файл логов. Среди всего прочего у расширенного файла логов существует поддержка множества инструментов анализа, что позволяет более подробно изучить происходящие в системе события. Это способствует более подробному ее анализу. Расширенный формат все еще находится на рассмотрении по состоянию на май 2024 года, но можно предвидеть, что реализация расширенного формата файла логов в конечном итоге изменит анализ логов [45].

2.4 Методика анализа производительности

Была разработана методика анализа производительности, которая заключалась в модель-управляемом процессе выявления и устранения регрессий производительности веб-приложений. В рамках методики алгоритма анализа, метод на основе программного комлекса и модели сложности создания и сложности контента, составляют ее теоретическую основу. При превышении заданного моделью порога метрики в соответствующей подвыборке выполняется детальный анализ приложения. Основным механизмом здесь выступает дельта-анализ. Для каждой метрики в рамках разработанных моделей сравниваются агрегированные текущие значения с их базовыми значениями, рассчитанными с использованием той же агрегатной функции, для той же подвыборки. После этого осуществляется проверка на превышение порогов модели и определение факта срабатывания, а также выполняется разложение вкладов дочерних метрик в изменение. При обнаружении изменений алгоритм (рисунок 2.3) последовательно выполняет сбор данных о производительности, формирование и актуализацию профилей приложений,

выполнение дельта-анализа, идентификацию причин и генерацию рекомендаций по улучшению.

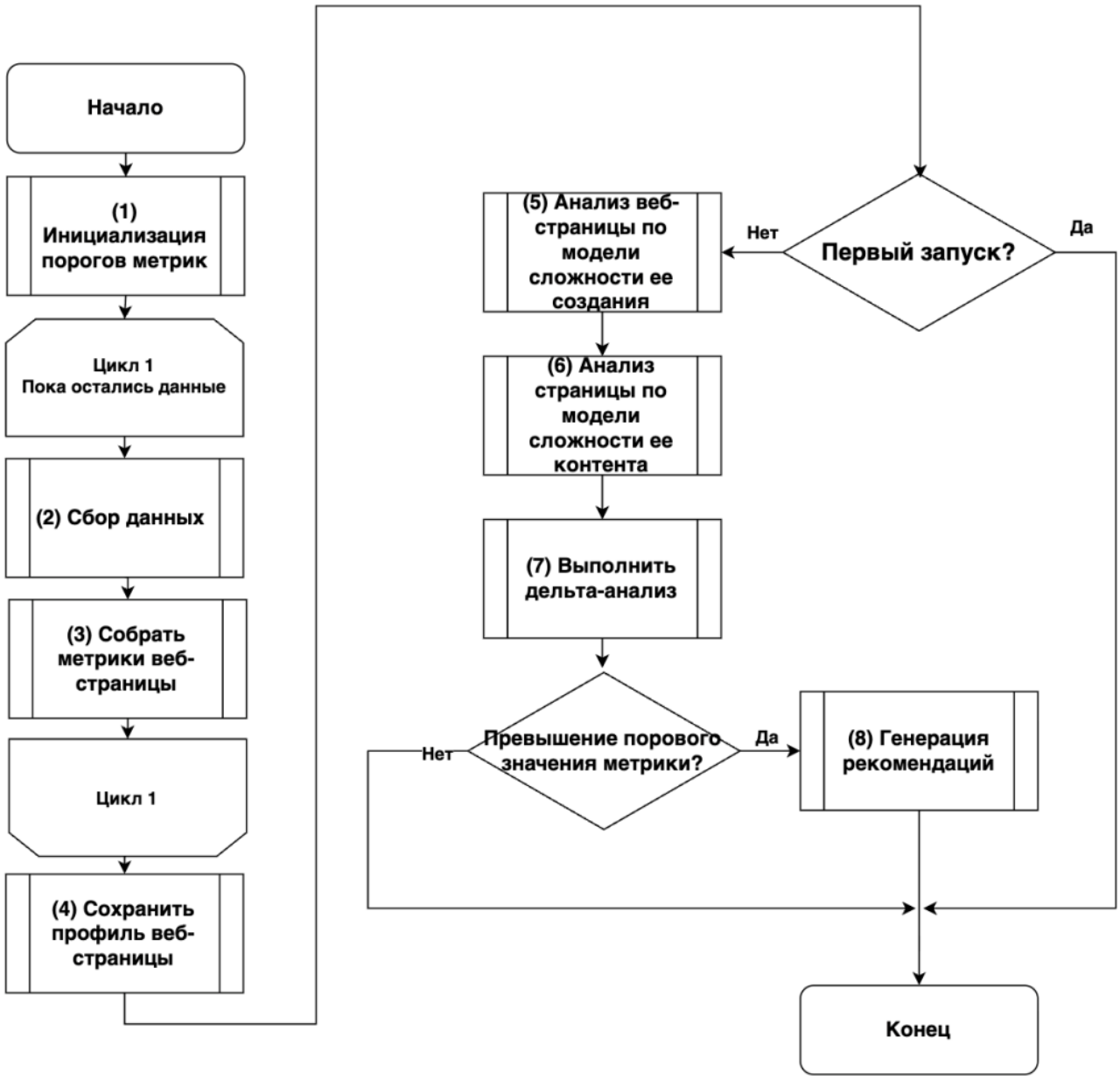


Рисунок 2.3 – Алгоритм улучшения веб-страницы на основе модели времени ответа

В разработанном алгоритме, лежащем в основе методики m, является множество простых метрик, которые являются частью составной метрики: $M = \{m_1, m_2, \dots m_n\}$, где M – это составная метрика. На первом шаге выполняется инициализация порогов τ для всех метрик модели в каждой соответствующей подвыборке условий, а также фиксируются параметры сравнения: агрегирующие функции $agg_m(x)$. Затем выполняется сбор данных о

производительности и формируется текущий профиль страницы P_t, содержащий наблюдения по метрикам модели. Для каждой метрики m вычисляется сводное текущее значение $x_t(m) = agg_m(P_t)$ и проверяется превышение заданного порога. Если его нет, то профиль сохраняется, что обеспечивает накопление данных для дальнейшего анализа. После чего алгоритм завершает работу. При наличии превышения происходит запуск проверки для данной подвыборки. Если ранее опорный профиль не существовал, то он используется как эталон P_0 для всех последующих проверок. Однако, если это не первый запуск, то выполняется сбор данных, при котором повторно фиксируется P_t, анализируется веб-страница по модели сложности ее создания и по модели сложности ее контента. После этого происходит выполнение дельта-анализа: $m = x_t(m) - Q_m$, где Q_m – эталонное значение метрики, с которым сравнивается текущее значение при анализе. Далее выполняется проверка порогов срабатывания в соответствии с порогами и правилами модели и регистрируются результаты проверки вместе с контекстом подвыборки. На основании полученных дельт и значений метрик сложности формируются рекомендации по улучшению производительности, после чего алгоритм завершает цикл и готов к следующему запуску при неизменности определений и агрегаторов, что гарантирует воспроизводимость и корректность сравнения.

2.5 Применение методики и локализация проблемных мест

При использовании приложения, имеющего большое время загрузки доплнительных объектов, возможно превышение порога метрики «Время отклика». В таком случае пользователь получает уведомление, что существует один или несколько дополнительных объектов, загрузка которых требует много времени. Каждый дополнительный объект, загрузка которого занимает количество времени, которое превышает порог, следует проверить. Необходимо оценить его значение и, при необходимости, уменьшить его путем изменения исходного кода. Этот подход позволяет браузеру быстрее загружать веб-

страницы, содержащие меньший объем информации. Таким образом, ресурсоемких дополнительных объектов либо их замена на более меньшие аналоги может существенно снизить общее время отклика веб-страницы. Например, имея изображения *JPEG*-формата с последующим преобразованием в формат *WebP* позволяет уменьшить их объем в среднем на 30 %, что, в свою очередь, снижает объем передаваемых данных и ускоряет загрузку страницы [63].

Существуют ситуации, когда среднее время загрузки дополнительных объектов соответствует пороговому значению, но общее время отклика страницы превышает установленное пороговое значение. Это может свидетельствовать о другом аспекте сложности контента, а именно о количестве дополнительных объектов. Увеличение количества таких объектов существенно влияет на время отклика страницы, поскольку веб-браузер затрачивает дополнительные ресурсы и время на обработку каждого из них [64]. Кроме того, запросы для загрузки каждого объекта создают дополнительную нагрузку на сетевое соединение, что также увеличивает общее время отклика страницы [65]. Предполагается, что запросы для получения дополнительных объектов равномерно распределяются и, поэтому данный фактор не является основной причиной увеличения времени отклика. Для решения описанной проблемы можно сократить количество дополнительных объектов на странице или задействовать несколько веб-серверов для их обработки.

Альтернативный подход включает использование протоколов с поддержкой постоянных соединений и конвейерной обработки, таких как *HTTP/1.1*, что способствует минимизации времени загрузки объектов и снижению времени отклика [66]. Внедрение технологии конвейерной обработки способствует ускорению загрузки веб-страниц при использовании низкоскоростных сетевых каналов, а также снижению общего количества передаваемых пакетов *TCP/IP*, что уменьшает нагрузку на сеть. Подобный способ особенно эффективен для пользователей, у которых задержка времени отклика составляет значительную долю от общего времени установки соединения. Однако, если время загрузки отдельного дополнительного объекта оказывается долгим, это может свидетельствовать о том, что основной причиной является задержка, связанная с

обработкой страницы на сервере [67]. В частности, если речь идет о временных затратах на извлечение или генерацию данного объекта, а также его доставку клиенту, отправившему запрос. Указанная проблема, как правило, связана со сложностью процесса создания такого объекта. В ситуации, когда выполнение всех вышеописанных шагов не привело к сокращению времени отклика, рекомендуется проанализировать время рендеринга страницы. Особое внимание следует уделить идентификации *JavaScript* или *CSS* элементов на странице, которые существенного увеличивают время рендеринга, а, следовательно, и общее время отклика [68, 69].

Данные о временных затратах загрузки и обработки основной *HTML*-страницы и связанных с ней вспомогательных объектов целесообразно собирать раздельно. Данный способ позволяет определить и сравнить влияние самой *HTML*-страницы и каждого дополнительного элемента в формировании совокупного времени отклика [70]. Несмотря на возможные вычислительные затраты, которые вносятся используемыми инструментами мониторинга, полученные данные обладают большой информативностью для анализа относительных задержек, обусловленных структурой *HTML*-страницы и ее дополнительными объектами [71]. Результаты анализа таких данных позволяют выявлять места веб-приложения, которые увеличивают общее время отклика.

2.6 Сложность генерации веб-страницы

Современные клиент-серверные приложения используют два основных подхода для генерации веб-страниц, которые запрашиваются клиентом:

1. Страница создается до поступления запроса. Такая страница называется статической [72, 73].

2. Страница создается на лету при получении запроса. Это называется динамической страницей [72].

Два описанных подхода позволяют сформировать веб-страницы на стороне сервера и доставить их клиенту. Самым распространенным способом создания страниц в современной разработке является комбинация статических и динамических элементов [72, 73], где статические страницы включают

динамические компоненты, тогда как динамические страницы нередко создаются с использованием заранее подготовленных статических шаблонов.

Дополнительно динамические элементы могут быть предварительно сгенерированы перед их запросом конечными пользователями, что позволяет сократить время отклика. Однако, при таких условиях как загрузка сервера, пропускная способность сети и объем трафика, обработка динамической веб-страницы неизбежно требует большего количества времени по сравнению с ее статическим аналогом даже в случае, если содержимое двух страниц идентично [74, 75]. Такое использование ресурсов обусловлено дополнительными вычислительными затратами, связанными с генерацией динамического контента в реальном времени, что делает улучшение такого процесса критически важным для создания условий для быстрой и эффективной работы клиент-серверных веб-приложений. Во-вторых, динамические страницы определяются тем, что их создание часто включает выполнение запросов к базе данных [75, 76]. Взаимодействие между веб-сервером и базой данных увеличивает время отклика, что обусловлено дополнительными вычислительными затратами и задержками, связанными с обработкой запросов и передачей данных.

2.7 Отклик и сложность веб-страницы

Сложность веб-страницы – это термин, который обычно используется для описания вычислительной нагрузки по обработке и визуализации страницы в браузере. Некоторые разработчики связывают их сложность внешним дизайном и такими особенностями как графические, динамические элементы, а также интерактивность. В некоторых случаях данное понятие определяют как результат человеческого познания, необходимого для использования системы, ее содержания (в частности, информации), доступного в рамках системы, и ее дизайн-системы, в которой она построена [59]. Иногда сложность воспринимается как когнитивная нагрузка, которая возникает при восприятии и использовании системы, включая взаимодействие с ее контентом и особенности ее архитектуры [64, 77].

В ходе исследования была выявлена взаимосвязь между сложностью веб-страницы и ее временем отклика, а также были разработаны модели, описывающие данную взаимосвязь. Подобная зависимость может быть формализована в виде пары взаимосвязанных переменных, имеющих причинно-следственные отношения между ними. В контексте такой связи переменные могут выступать как в роли источника воздействия, так и в качестве его объекта.

Существует два типа взаимосвязей между переменными: прямая зависимость, при которой изменение значения одной переменной (увеличение или уменьшение) приводит к аналогичному изменению другой переменной, и обратная зависимость, при которой изменение одной переменной вызывает противоположное изменение другой [78, 79]. Эти типы взаимосвязей являются фундаментальными при построении причинно-следственных связей в исследовании производительности веб-приложений, где анализ влияния одного параметра на другой позволяет формализовать процессы, происходящие в системе. Например, увеличение объема передаваемых данных может прямо коррелировать с ростом времени отклика (прямая зависимость), в то время как увеличение количества кэшированных объектов может привести к снижению времени отклика (обратная зависимость) [80]. Идентификация характера зависимости между компонентами системы позволяет не только глубже понять поведение клиент-серверного приложения в условиях нагрузки, но и построить модели, описывающие процесс работы системы и по ней выявить узкие места производительности [81]. Таким образом, точная интерпретация взаимосвязей представляет собой ключевой этап при формализации и проверке эмпирических закономерностей, выявленных в ходе экспериментального анализа.

Одним из примеров причинно-следственной зависимости в технической литературе является связь между сложностью веб-страницы и временем ее отклика. Исследователи подчеркивают, что повышение сложности страницы может негативно сказываться на ее производительности [82]. В технической литературе часто упоминается то, как сложность веб-страницы влияет на время ее отклика [1, 2, 82]. Для повышения скорости загрузки страницы рекомендуется уменьшить ее

размер путем сведения к минимуму использования графических элементов и мультимедийных, а также ограничивать количество изображений и других ресурсоемких медиа-элементов на веб-странице, чтобы ее общий размер не превышал установленное пороговое значение, что способствует улучшению производительности. Отсюда следует, что понятие «сложность веб-страницы» можно определить как совокупность характеристик ее структуры и содержания, оказывающих прямое влияние на время отклика.

2.8 Метрика количества строк кода

Сложность генерации статических веб-страниц определяется как величина, пропорциональная числу строк исходного кода, за исключением комментариев. Использование данного определения объясняется тем, что статические веб-страницы предполагают последовательный анализ и парсинг браузером текущей *HTML*-страницы и загрузку скриптов. С увеличением количества строк кода возрастает объем данных, подлежащих передаче от сервера клиенту, что ведет к увеличению времени в рамках процесса рендеринга страницы в браузере пользователя, а, следовательно, будет увеличено и время отклика [83, 84].

Метрика количества строк кода, которая используется для оценки сложности разработки как статических, так и динамических веб-страниц, требует дифференцированного подхода с учетом их типа. В частности, строки кода для статических страниц преимущественно содержат инструкции для клиентского браузера, связанные с отображением содержимого и внешним видом [84]. В то же время строки кода динамических страниц часто включают фрагменты кода, который выполняет работу на сервере (например, обращение за ресурсами). Такие операции требуют значительных вычислительных ресурсов и временных затрат, превышающих аналогичные показатели для статических страниц.

Если обозначить вес строки кода статической страницы как $weight_1$, то сложность разработки статической страницы, содержащей k строк кода, может быть выражена с использованием формулы 2.10.

$$linecode = k \cdot weight_1 \tag{2.10}$$

Более того стоит учитывать, что для динамической страницы с k строк кода сложность по отношению k строк кода описана в формуле 2.11.

$$linecode = x \cdot k \cdot weight_1 \quad (2.11)$$

В рамках формулы 2.11 должно выполняться условие $x \geq 1$. Значение x зависит от характеристик серверного окружения и технологий, применяемых при создании динамической страницы, что отражает дополнительные вычислительные затраты на обработку.

2.9 Метрика сложность программных конструкций

Для динамических страниц, помимо строк кода, необходимо учитывать и другие показатели. Это связано с тем, что динамические страницы дополнительно включают такие конструкции как функции, циклы. Современные веб-страницы в той или иной мере похожи на традиционную программу, написанную на таких языках программирования как *C, Java* или *Python*. Для выявления влияния различных аспектов таких программ были предложены метрики, включая цикломатическую сложность, размер кода программы, вложенность конструкции.

Цикломатическая сложность – это метрика, которая используется в программировании для оценки сложности кода. Она определяет количество путей ветвления, используемых в программе, и основана на графе управления выполнением программы [85, 86]. Этот показатель помогает понять, насколько сложна логика программы и сколько тестов необходимо для полного охвата всех возможных сценариев выполнения.

В данной работе для выражения сложности динамической веб-страницы используется двухуровневая мера. На первом уровне учитываются строки исходного кода страницы. Данный код не содержит комментариев, подлежит выполнению и не включает элементы, относящиеся ко второму уровню анализа, который охватывает использование программных конструкций [86]. Кроме того, код, отвечающий за создание веб-страницы, может быть распределен между несколькими файлами, что усложняет его интеграцию и анализ [87]. В этом случае

код помещается в один файл для общего подсчета строк кода. Второй уровень оценки сложности разработки основывается на использовании нотации «Большое О» (*Big O*), предложенной немецким математиком Полом Бахманом [88]. Эта нотация широко применяется для выражения асимптотической оценки роста функций и представляется в форме *O(f(n))*. В соответствии с определением, приведенным Дональдом Кнутом, нотация используется в тех случаях, когда *f(n)* является функцией от целого положительного числа *n* [89].

Данная нотация используется для описания асимптотического поведения функций, позволяя проводить их сравнение. Она предоставляет оценку количества операций, необходимых для выполнения алгоритма, и, таким образом, связана с временными затратами на решение задачи [90]. В контексте определения сложности создания веб-страниц данная нотация применяется для аппроксимации и сравнения времени, требуемого для разработки динамических веб-страниц.

Ниже приводится метрика сложности создания динамической страницы, которая включает в себя нотацию «Большое О». Циклы и функции являются конструкциями программирования, которые оцениваются с помощью нотации «Большое О» [89]. Для каждой конструкции ее сложность обозначается следующим образом: *O(f(n)) и* характеризует зависимость от размера входных данных. Сложность всех программных синтаксических конструкций кода динамической веб-страницы выражается в формуле 2.12.

$$skprog = \sum_{i=1}^{j} f\left(n_i\right) \cdot linecode_i \cdot weight_{2,i} \cdot d_i \qquad (2.12),$$

где *j* – количество конструкций, используемых на странице;

d_i – коэффициент вложенности каждой конструкции в структуре кода (например, вложенные циклы, условия внутри циклов).

Для модели вводится классификация конструкций *JavaScript* по типам с характерными оценками сложности и весами (таблица 2.2).

Таблица 2.2 – Классификация конструкций *JavaScript* по типам с характерными оценками сложности и весами

Тип конструкции	Пример	Сложность в нотации «Большое О»	Вес
Линейная	const v = map[key]	O(1)	1
Условная	if, switch	O(1)-O(n)	2
Циклическая	for, while, do while, forEach	O(n), $O(n^2)$	3-5
Рекурсивная	Рекурсивные вызовы	$O(2^n)$, O(n!)	6-10
Асинхронная	async/await, fetch()	O(n)	3-5
Встроенные функции	map, filter, find	O(n), $O(n^2)$	1-3

Для количественной оценки уровня вложенности каждой конструкции *JavaScript* (цикла, условия, функции) применяется абстрактное синтаксическое дерево (*AST*), которое отображает иерархию блоков и логики в коде [91]. На основе обхода *AST* для каждой конструкции i вычисляется глубина вложенности (вес из таблицы 2.2), затем нормализуется в коэффициент d_i (формула 2.12).

$$d_i = 1 + cd \cdot \left(depth_i - base \right), \tag{2.12}$$

где i – текущая программная конструкция, используемая на странице;

$base$ – это константа, равная единице.

Стоит отметить, что операция вычитания применяется для того, чтобы влияние глубины вложенности начиналась только со 2-го ее уровня. Это позволяет устранить влияние конструкций, расположенных на первом уровне вложенности.

Коэффициент cd в формуле (2.12) нормализации глубины вложенности конструкций *JavaScript*-кода определяется как масштабирующий параметр и

зависит от цели оценки. При $cd = 1$ каждый дополнительный уровень вложенности добавляет единицу к базовому значению 1. При $0 < cd < 1$ влияние применяется, если незначительные уровни вложенности не критичны.

В случаях, когда определенные участки кода динамической страницы недоступны для анализа их сложности, этим частям присваивается оценочная сложность rc_e, чтобы компенсировать недоступность $f(n) \cdot linecode \cdot weight_2 \cdot d_i$. При этом $rc_e \in [0,1]$. Чем выше rc_e, тем менее надежна исходная оценка сложности, и тем выше итоговый коэффициент. Определение значения, которое должно быть присвоено rc_e, требует специальных навыков и является несколько субъективным. С помощью двухуровневой меры, описанной выше, сложность генерации веб-страницы, независимо от того, является ли она статической, динамической или гибридом статической и динамической, может быть достаточно подробно описана вышеуказанными подходами.

2.10 Состояние приложения

Любое современное веб-приложение не имело бы смысла без возможности работы с данными. Работа с системами управления базами данных и подключение баз данных к приложениям. Однако внедрение таких технологий ведет к внесению задержек и влияние на время отклика. Информация о состоянии веб-приложения является фундаментальной и в то же время критически важной метрикой при генерации веб-страниц. Как уже было сказано, существует возможность поддержки состояния с помощью базы данных на сервере или с помощью куки (*cookie)* на клиентской машине [92]. В отношении базы данных рассматриваются следующие аспекты:

Метрика, обозначаемая как col_{db}, отражает количество обращений к базе данных. В отличие от простой количественной оценки числа выполненных запросов данный показатель учитывает количество индивидуальных обращений к каждой таблице базы данных [93, 94]. Например, если один запрос осуществляет доступ к двум различным таблицам, то в рамках этой метрики

будет зафиксировано два обращения, а не одно. Аналогично, если к одной таблице обращаются два разных запроса, очевидно, что запросов будет два.

Размер базы данных sz_{db} – это количественная характеристика объема хранимых в СУБД данных, которая может выражаться в байтах, мегабайтах или килобайтах, количестве записей, таблиц или других логических структур [95]. Чем больше база данных, к которой осуществляется доступ, тем больше времени потребуется для построения вывода по заданному запросу.

Количество установленных соединений, k_{soed}. Соединение с базой данных является задачей, которая занимает большое количество ресурсов и времени. Исходя из этого, можно сказать, что чем больше соединений установлено для создания веб-страницы, тем больше времени потребуется на ее создание [96].

Более того важно отметить, что операции над *cookie* данными или работа с данными текущей сессии также влияет на время отклика. Сессия является временным интервалом, в течение которого пользователь, который идентифицирован под уникальным *IP*-адресом или другим идентификатором, взаимодействует с веб-приложением. В рамках сессии могут использоваться специальные переменные для сохранения информации о пользователе, что необходимо для восполнения ограничений протокола *HTTP*. Поскольку *HTTP* является протоколом без состояния [11], то он не обеспечивает возможность хранения данных пользователя между несколькими соединениями, устанавливаемыми между клиентом и сервером в рамках единой сессии. Применение таких переменных позволяет создать псевдосеанс, обеспечивая сохранение данных о действиях пользователя при последовательных переходах между страницами одного веб-сайта.

Данные *cookie* фактически являются небольшими текстовыми фрагментами, передаваемые веб-сервером на клиентский браузер и возвращаемые браузером серверу при повторных запросах к тому же сайту или домену [92]. Этот механизм позволяет сохранять состояние сессии и передавать переменные, связанные с ней. Однако обмен данными через *cookie*, включая их передачу и обработку для извлечения сессионных переменных или другой

информации, может увеличивать время отклика веб-страницы. Влияние использования сессионных переменных и данных *cookie* веб-страницы обозначается как vl.

Метрика состояния соединений с базой данных k_{soed} не зависит от размера базы данных sz_{db}. Однако последняя влияет на продолжительность операции по формированию результатов запроса ql из базы данных. Следовательно, сложность, связанная с доступом к базе данных, а также с обработкой данных сессий и данных *cookie*, может быть выражена следующим образом: $g_{conn}(k_{soed}) + g_{req}(ql, sz_{db}) + vl$. При этом производительность базы данных, которая определяется такими дополнительными факторами, как скорость физического носителя, размер страниц базы данных и схема размещения базы данных на физических дисках, в данной модели сложности не учитывается [96].

2.11 Моделирование сложности генерации веб-страницы

Пусть сложность генерации веб-страницы обозначается как C_1. Для статических веб-страниц величина C_1 определяется в виде функции, зависящей от количества строк кода, непосредственно задействованных в работе веб-приложения. Таким образом, $C_1 = f(k)$, где k является строками кода, которые не были закомментированы, и они являются исполняемыми в рамках *HTML*-разметки страницы. В общем случае получается $f(k) = k \cdot weight_1$, где $weight_1$ – это вес выполнения, присваиваемый каждой строке кода, чтобы отразить время выполнения строки.

Для динамических страниц C_1 выражается как функция, которая включает в себя количество строк кода ($linecode$), сложность конструкций языка программирования *JavaScript* ($skprog$) и скорости доступа к базе данных ($dbacc$). Следовательно, $C_1 = g_{line}(linecode) + g_{prog}(skprog) + g_{db}(dbacc)$, где $linecode$ – количество строк кода; $skprog$ – сложность программных

конструкций; $dbacc$ – влияние состояния базы данных; $g_{line}(linecode)$ – вклад количества строк кода приложения; $g_{prog}(skprog)$ – вклад программных конструкций; $g_{db}(dbacc)$ – вклад состояния базы данных.

Данную функцию можно представить следующим образом:

1. Страница, которая содержит k строк кода и ее сложность генерации, связанная с $linecode$, $g_{line}(linecode)$, определяется как $g_{line}(linecode) = x \cdot linecode \cdot weight_1$, где $x \geq 1$.

2. Страница, которая содержит j-программных конструкций (циклы, функции). Каждая подобная конструкция имеет $linecode_i$ строк кода, где $1 \leq i \leq j$, $g_{line}(linecode) = \sum_{i=1}^{j} f(n_i) \cdot linecode_i \cdot weight_{2,i} \cdot d_i$, где $weight_2$ является весом выполнения, присвоенный строке кода конструкции, а $\sum_{i=1}^{j} f(n_i) \cdot linecode_i \cdot weight_{2,i} \cdot d_i$ аппроксимация для общего количества строк кода, которые необходимо выполнить для программных конструкций.

3. Во время работы приложения, когда сервер генерирует страницу, может быть установлено k_{soed} к j базам данных. Каждая такая база данных с размером sz_{db_i}, где $1 \leq i \leq j$ и для каждого db_i существует n_i запросов, сложность генерации которых зависит от количества баз данных j, определяется следующим образом: $g_{db}(dbacc) = weight_3 \cdot k_{soed} + \sum_{i=1}^{j}\left(n_i \cdot T_{sz_{db_i}}\right) + vl$, где $weight_3$ является весом, отражающим время, необходимое для установления соединения с базой данных, а $T_{sz_{db_i}}$ отражает время обработки запроса к базе данных с размером sz_{db_i}, а с другой стороны vl является показатель влияния сессионных переменных *cookie*. Таким образом, можно количественно оценить три параметра, связанных со сложностью создания динамической страницы, а именно $linecode$, $skprog$ и $dbacc$.

2.12 Моделирование контента

На ранних этапах развития интернета веб-страницы преимущественно содержали текстовую информацию и характеризовались относительно небольшим объемом данных и не имели большого количества медиа-элементов. С развитием веб-технологий в их структуру стали активно добавляться графические элементы, а позже и мультимедийные компоненты, которые включают аудио- и видео-контент, интерактивные анимации [98]. В результате чего современный размер веб-страниц значительно увеличился. Это привело к тому, что контент страницы превратился в ключевой фактор, который влияет на время ее отклика. Кроме того, на время отклика оказывают воздействие такие элементы как скрипты и каскадные таблицы стилей. Сложность контента, обозначаемая как C_2, определяется на основе трех аспектов, которые будут рассмотрены далее.

Первый аспект сложности контента веб-страницы заключается в количестве дополнительных объектов, обозначаемом как $elm_{o,i}$. Веб-страница, содержащая большое число дополнительных объектов, считается более сложной по сравнению со страницей, имеющей меньшее число таких элементов. Дополнительными объектами в данном случае являются медиа-элементы. Сложность контента C_2 может быть представлена как функция от $elm_{o,i}$, то есть $C_2 = f\left(elm_o\right)$.

Второй аспект – это общий размер страницы z_{sp}. Чем больше страница по размеру в байтах, тем выше ее уровень сложности. C_2 можно определить как функцию от z_{sp}, то есть $C_2 = g_{size}\left(z_{sp}\right)$.

Третий аспект сложности контента веб-страницы связан с элементами, оказывающими влияние на время ее отклика. Каждый из таких элементов вносит вклад в общий уровень сложности страницы. Для количественного выражения этого вклада каждому элементу назначается вес, отражающий его относительную сложность. Например, сложность каскадной таблицы стилей может быть определена на основе ее размера в килобайтах, а также числа

содержащихся в ней правил и каскадов. Аналогично, сложность *JavaScript*-кода может зависеть от количества строк кода или от числа выполняемых операций. Таким образом C_2 можно выразить как сумму всех весов сложности для всех элементов, то есть $C_2 = \sum_{i=1}^{j} w_{elm_{o,i}}$, где j – количество элементов, участвующих в создании веб-страницы, а $w_{elm_{o,i}}$ – вес сложности, присвоенный элементу $elm_{o,i}$. Следовательно, C_2 может быть определен как функция от $elm_{o,i}$, то есть $C_2 = f\left(elm_o\right)$. На основе вышеизложенного можно вывести следующую формулу сложности для веб-страниц: $C_2 = g_{obj}\left(n_o\right), g_{size}\left(z_{sp}\right), g_{assets}\left(elm_o\right)$.

Выводы по главе

В данной главе был предложен метод для сбора данных, выполняющий сбор данных о производительности веб-приложений, основанных на архитектуре клиент-сервер. Метод основан на программном комплексе, который состоит из отдельных масштабируемых компонентов. Он позволяет извлекать данные с использованием библиотеки-агента и логов сервера, обеспечивая корреляцию событий в рамках используемой архитектуры и формирование профилей производительности для последующего их анализа. В отличие от аналогов он имеет низкие задержки и накладные расходы на выполнение сбора данных.

Разработаны модели, описывающие время отклика веб-страницы как функцию метрик ее кода и контента. В частности, метрики веб-страницы, оказывающие влияние на время отклика, которые были выделены в две основные категории: сложность генерации и сложность контента веб-страницы. Предложенная модель позволяет описывать синтаксические конструкции и элементы логики приложения, которые влияют на время создания веб-страницы сервером, а также объединяет синтаксическую и алгоритмическую сложность и позволяет количественно оценивать сложность формирования страницы на сервере еще до передачи клиенту.

Разработан метод сбора данных производительности клиент-серверных приложений с использованием программного комплекса, состоящего из отдельных масштабируемых компонентов, основным из которых является библиотека-агент, которая встраивается в веб-страницы на клиенте. В отличие от известных аналогов такой метод обладает низкими накладными расходами на выполнение сбора данных.

Разработана методика на основе пяти взаимосвязанных элементах: метод сбора данных производительности клиент-серверных веб-приложений, модель сложности создания, модель сложности контента веб-страницы и мониторинг времени отклика, а также реализующий данную методику алгоритм. В рамках данного исследования производительность рассматривается как результат внутренних метрик страницы, включая синтаксическую структуру кода, динамическую логику генерации и содержимого, а не только как функция сетевых и серверных факторов.

3. РАЗРАБОТКА СИСТЕМЫ ДЛЯ СБОРА ДАННЫХ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Архитектура «клиент-сервер» является наиболее популярной в современной разработке и на ней основана большая часть сетевых приложений. Наиболее распространенная реализация этой модели – веб-приложение. Сервер генерирует веб-страницы на основе данных из базы данных в ответ на запросы клиентов [99].

Метод сбора информации о производительности клиент-серверных веб-приложений с использованием программного комплекса был использован для получения данных о производительности. Для того, чтобы провести сравнение двух методов, был создан прототип веб-приложения для отображения списка *JavaScript*-библиотек. Доступ к нему осуществлялся с другой машины, выступающей в роли клиента. Детали оборудования, на котором выполнялось тестирование следующие:

Серверное оборудование включало процессор *Intel Core i5*, 4 ГБ оперативной памяти и операционную систему *Ubuntu Server* 22.04. Клиентская машина также была оборудована процессором *Intel Core i5* и 4 ГБ оперативной памяти, работала под управлением *Windows* 10. Веб-браузер *Google Chrome* 133 использовался для проведения тестов. Для тестирования было создано клиент-серверное веб-приложение для отображения популярных *JavaScript*-пакетов.

Динамические веб-страницы были разработаны с использованием фреймворка *Next.js* [100]. Системы управления базами данных *PostgreSQL* и *MongoDB* применялись для хранения разных типов данных.

Первая из них употребляется для транзакционных и строго структурированных данных, для которых требуются *ACID*-требования и внешние ключи. Более того они позволяют выполнять аналитику данных производительности, используя сложные запросы.

MongoDB применялась как основа масштабируемого хранилища временных рядов. Документная модель позволяет добавлять новые метрики, теги, версии браузеров без простоев и лишних миграций. Это является критичным фактором при быстро меняющихся трендах и требованиях в мире веб-разработки. Подобная схема обеспечивает обратную и прямую

совместимость данных, где для одного хранилища длительно существуют разные версии модели данных производительности, а аналитические сервисы продолжают работать, используя значения по умолчанию для отсутствующих полей. Как результат - программный комплекс является гибким для появления новых показателей и сценариев без модификации схемы на уровне системы управления базами данных и без остановки сервиса, сохраняя целостность и пригодность данных для последующего анализа.

Для оценки эффективности разработанного метода сбора данных о производительности клиент-серверных веб-приложений с использованием программного комплекса было выполнено его сравнение с уже существующим методом, основанным на применении прокси-сервера. Он позволяет перехватывать и анализировать сетевой трафик между клиентом и сервером, обеспечивая детализацию процессов передачи и обработки данных.

Был выполнен сравнительный анализ трех методов для сбора данных производительности: метод на основе программного комплекса с встраиваемым агентом и сервером обработки, метод на основе фреймов и метода на основе прокси-сервера и представленным в данной работе методом, использующим программный комплекс. Сравнение проводилось на унифицированном наборе воспроизводимых тестовых сценариев, реализованных для веб-приложения, отображающего список популярных *JavaScript*-библиотек. Такие условия позволили обеспечить сопоставимость условий эксперимента и исключили появление различий в прикладной логике.

Созданное клиент-серверное веб-приложение, генерирующее динамическую веб-страницу, позволило выполнить сравнение методов сбора данных с помощью прокси-сервера, фреймов и программного комплекса. Условия работы клиентской, серверной, а также локальной сетей были максимально согласованы, что значит, были исключены случайные или нерелевантные колебания в условиях тестирования. Полученные данные не подвергнуты влияниями сторонних факторов. На рисунке 3.1 показан пользовательский интерфейс веб-приложения, отображающего популярные

библиотеки для *JavaScript* из сервиса для хранения пакетов *JavaScript – npm* (*node package manager*) [101].

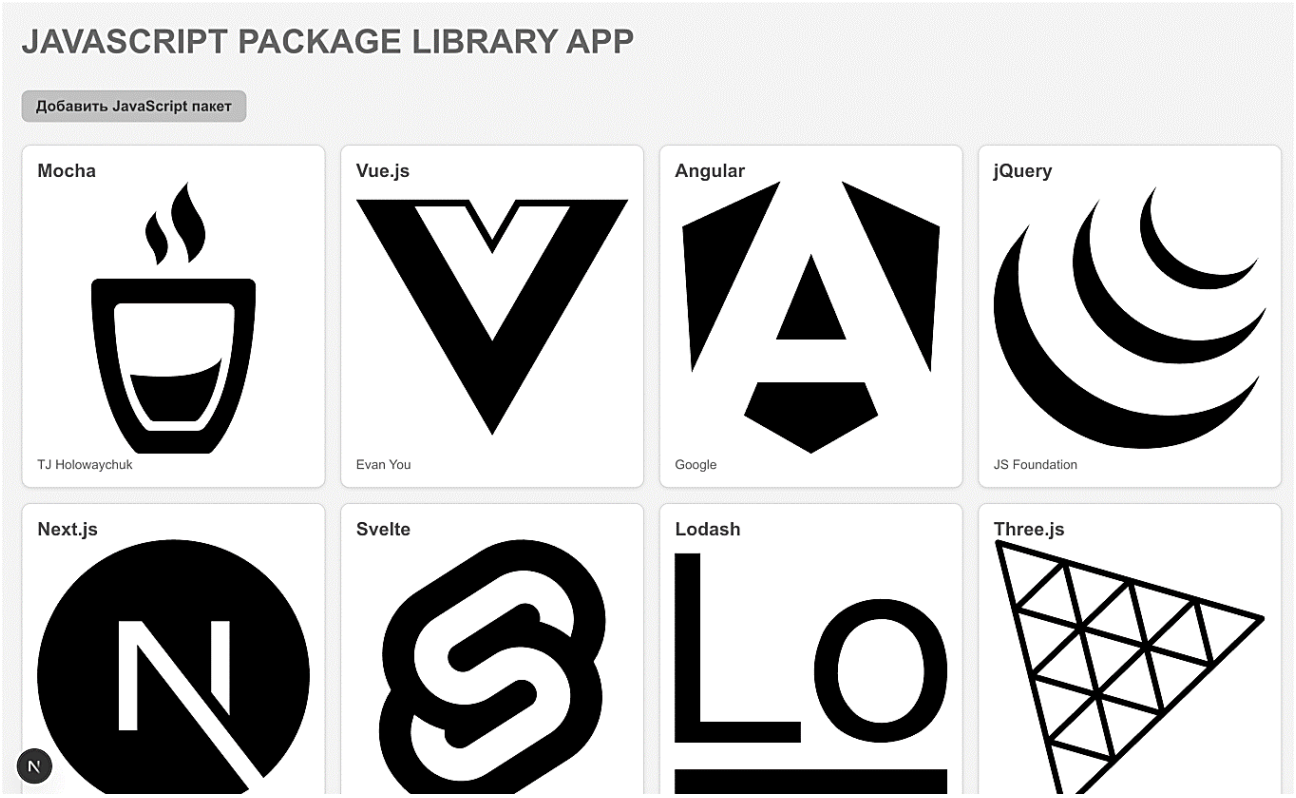


Рисунок 3.1 – Пользовательский интерфейс страницы веб-приложения, отображающего популярные библиотеки для *JavaScript*.

Для страницы были рассмотрены три сценария. Для каждого сценария были получены 100 значений. Сценарии описаны в следующих подразделах.

3.1 Программный комплекс

Программный комплекс для сбора данных о производительности клиент-серверных веб-приложений является многокомпонентной системой, реализующей микросервисный подход. Она обеспечивает сбор, обработку и анализ метрик производительности в реальном времени. При разработке программного комплекса в качестве технологического стека были выбраны: фреймворк *Spring Boot*, язык программирования *Kotlin*, *SQL*-база данных *PostgreSQL* и *NoSQL* база данных *MongoDB*.

Выбор *Spring Boot* обусловлен тем, что он имеет поддержку модульной архитектуры, встроенной реализацией инверсии управления, управления жизненным циклом компонентов, что позволяет эффективно реализовывать архитектуру, которая будет соответствовать принципам *SOLID (Single responsibility, Open–closed, Liskov substitution, Interface segregation и Dependency inversion*. Кроме того, использование *Spring Boot* способствует применению *GRASP*-паттернов (*General Responsibility Assignment Software Patterns*), обеспечивающих логичное распределение обязанностей между компонентами системы [18]. В совокупности применение лучших практик и паттернов позволяет добиться высокой степени абстракции, повторного использования кода и соответствия архитектурных решений, которые соответствуют современным требованиям к качеству программного обеспечения [102, 103]. Более того данный фреймворк является проверенным годами решением в разработке программного обеспечения любой сложности.

Выбор языка программирования *Kotlin* обоснован его современным и лаконичным синтаксисом. Он обеспечивает поддержку нескольких парадигм и предоставляет современные механизмы для работы с *null*-значениями и виртуальными потоками или корутины, которые позволяют уменьшить потребление ресурсов при выполнении в параллельных вычислениях. [103, 104]. Данная технология работает на *JVM (Java Virtual Machine)*, а значит, он полностью совместим с *Java*, что позволяет использовать уже имеющийся набор существующих *Java*-библиотек и фреймворков без дополнительной адаптации. Перечисленные достоинства делают данный язык программирования идеальным инструментом для разработки асинхронных и высоконагруженных веб-приложений [104 ,105].

В рамках данной работы выполняется сбор данных производительности. Для их хранения были выбраны два типа баз данных. Первая БД *PostgreSQL* хранит информацию о пользователях, проектах и клиентах. Вторая – *MongoDB* хранит информацию о данных производительности в виде временных рядов [106].

Диаграмма контекста, изображенная на рисунке 3.2, демонстрирует взаимодействие участников (менеджер проекта, разработчик и инженер-тестировщик) с программным комплексом, который интегрируется со сторонними веб-приложениями.

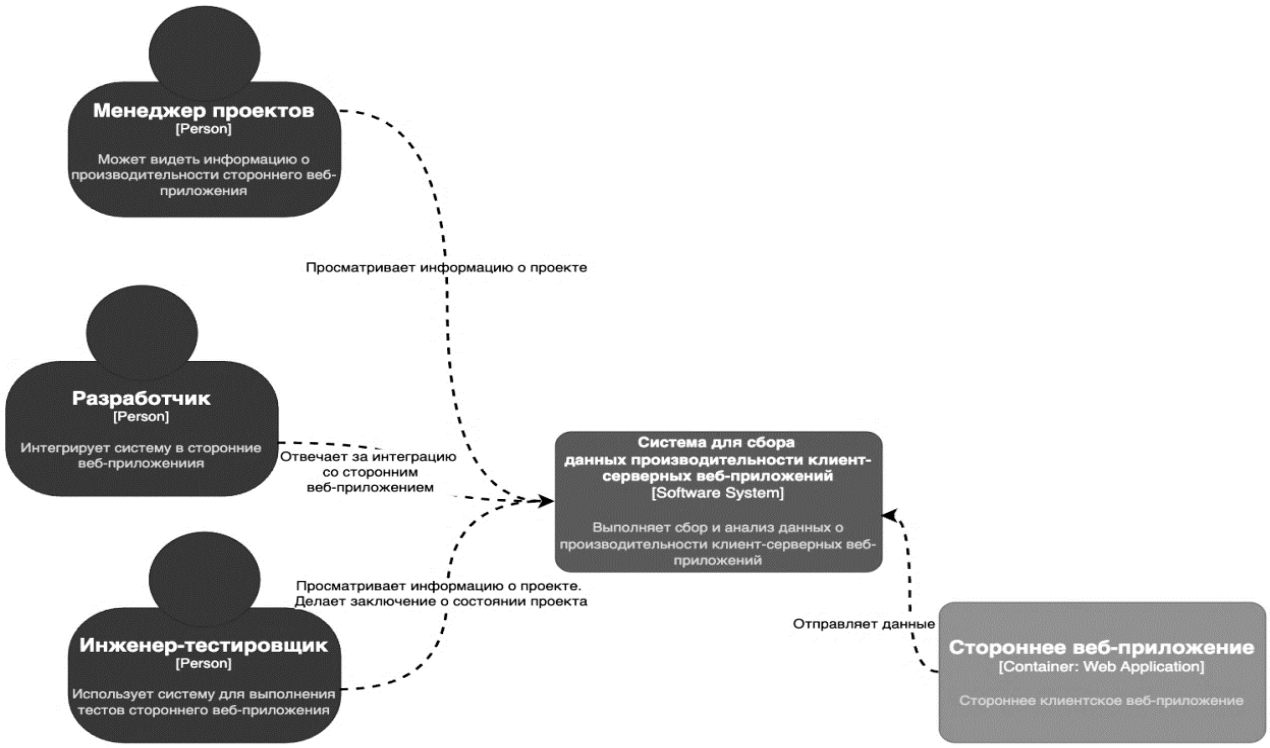


Рисунок 3.2 – Диаграмма контекста

Менеджер проекта получает обобщенную информацию о производительности и состоянии приложения, а также он делегирует команде разработки и тестирования задачи по работе над проблемными местами приложения. Разработчик отвечает за внедрение библиотеки в клиентскую часть стороннего приложения, а инженер-тестировщик использует собранные данные и анализирует состояние текущего приложения для тестирования и формирования заключений о его качестве.

Диаграмма контейнеров на рисунке 3.3 более подробно описывает внутреннюю архитектуру программного комплекса. Она состоит из нескольких ключевых контейнеров – развертываемых единиц: клиентский интерфейс, бэкенд-сервис и библиотека-агент. Клиент представляет собой веб-интерфейс, разработанный для отображения и анализа производственных данных.

Балансировщик, выставленный перед *REST API,* принимает входящие запросы и распределяет нагрузку между несколькими экземплярами сервера по причине того, что он обрабатывает большое количество запросов. Так несколько экземпляров сервиса будут способны выстоять при интенсивной нагрузке [106]. Это позволяет обеспечить высокую доступность сервиса. *REST API*-сервис содержит конечные точки, которые принимают данные или возвращают их пользователю-клиенту по его запросу.

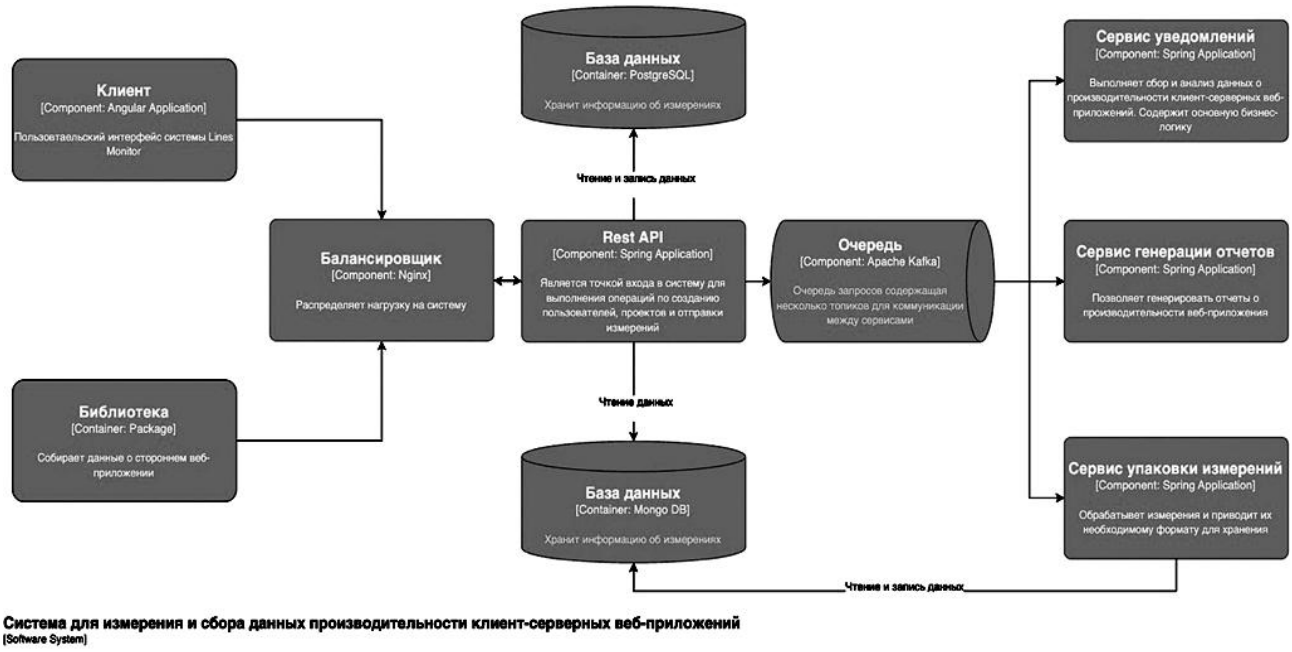


Рисунок 3.3 – Диаграмма контейнеров

Библиотека-агент встраивается в клиентскую часть стороннего приложения и собирает данные о его времени рендеринга, времени отклика и времени загрузки дополнительных объектов. Сервис уведомлений позволяет сообщить пользователям программного комплекса информацию о том, что, если их приложения превышают определенные пороговые значения. Сервис генератор отчетов позволяет создавать отчеты на основе полученных данных. *Measurements bundler* обрабатывает и преобразует данные в формат, необходимый для дальнейшего хранения [106].

На диаграмме компонентов видно, что каждая функциональная единица отвечает за строго определенный набор задач в рамках всей системы (рисунок 3.4) [108]. Компонент «Библиотека» реализует клиентскую логику сбора метрик.

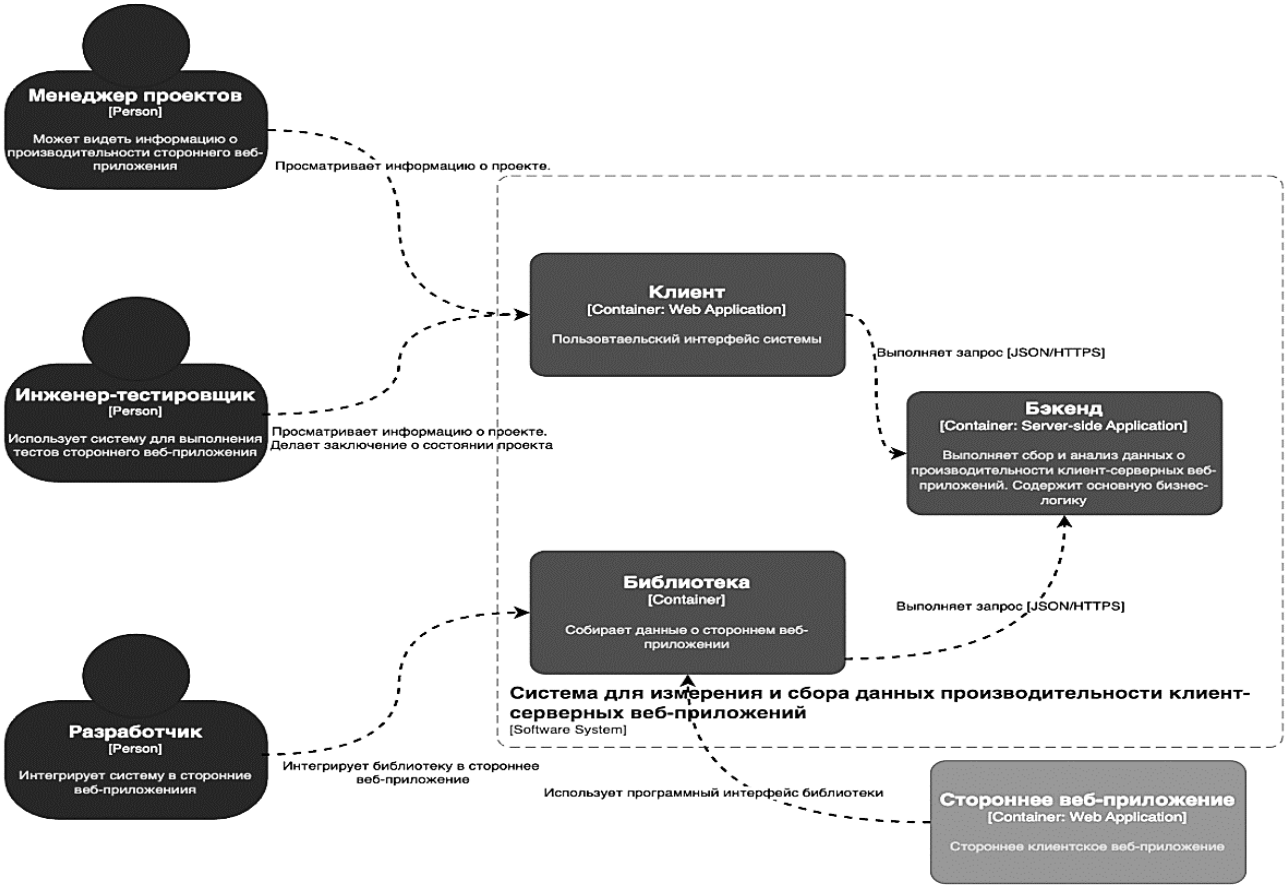


Рисунок 3.4 – Диаграмма компонентов

Компоненты клиентского интерфейса предоставляют визуализацию и позволяют осуществлять фильтрацию, анализ и экспорт собранных данных. Компонент «Бэкенд» управляет потоками данных, осуществляет их хранение и передачу в очереди сообщений для дальнейшей их обработки.

Диаграмма кода на рисунке 3.5 демонстрирует модульную организацию и взаимосвязи между программными классами. На диаграмме описана классовая структура, которая является частью *Spring Boot*-приложения [107, 108]. Представленная диаграмма показывает взаимосвязи между основными компонентами системы и подчеркивает их ответственность в рамках функционального деления. Центральным элементом системы является класс *LinesMonitorApplication*, инициализирующий конфигурацию и загрузку всех компонентов приложения. Через механизмы аннотаций *@SpringBootApplication*, *@EntityScan*, *@EnableJpaRepositories* и *@ConfigurationPropertiesScan* фреймворк обеспечивает сканирование компонентов, репозиториев и конфигурационных

свойств, включая регистрацию *JpaRepositoriesRegistrar*, *EntityScanPackages.Registrar*, и другие инфраструктурные классы *Spring* для достижения слабой связанности кода и передачи управления зависимостями стороннему контейнеру [102]. Стоит отметить, что в работе использовались слои контроллеров, сервисов и взаимодействия данными. Каждый из слоев обладает своими обязанностями. Так контроллеры отвечают за работу с *HTTP*-запросами. Они выполняют валидацию входящих данных и передают данные на уровень сервисов, делегируя логику обработки данных им. Слой взаимодействия с данными является тем местом, в котором происходит взаимодействие с базами данных.

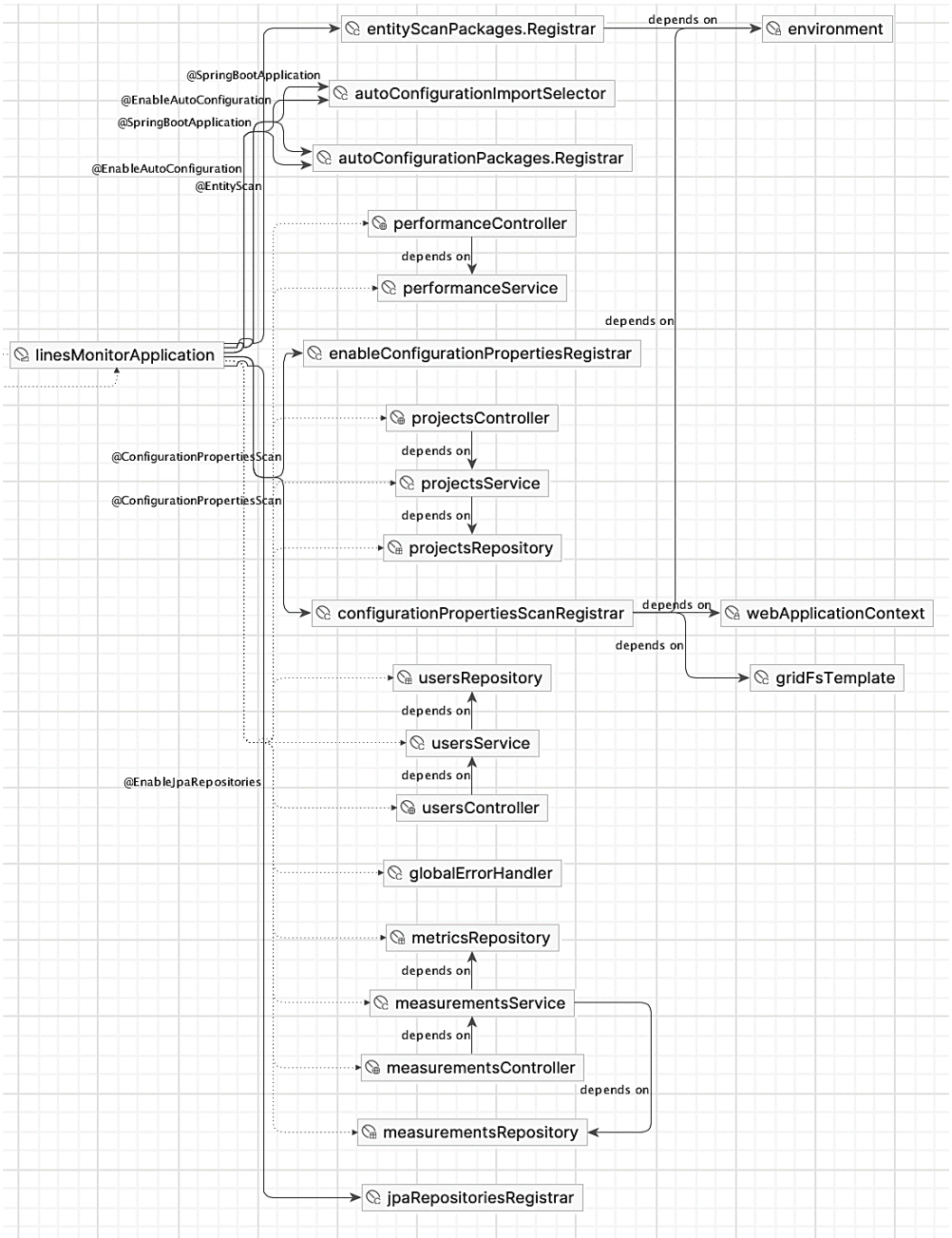


Рисунок 3.5 – Диаграмма классов

Модуль *Users* представлен тремя классами: *usersController*, *usersService* и *usersRepository*, каждый из которых реализует модель контроллер-сервис-репозиторий. Они отвечают за управление данными пользователей, их хранение и доступ к информации, необходимой для авторизации и аутентификации пользователей в системе.

Модуль *Measurements* включает *measurementsController*, *measurementsService*, *measurementsRepository*, а также *metricsRepository*. Его основными задачами являются сохранение, обработка данных производительности, поступающих из внешних клиентских приложений. Здесь реализуется логика анализа входящих данных, их агрегация и подготовка к последующим этапам обработки.

Модуль *Projects* реализует учет и управление проектами, к которым относятся полученные данные. Он представлен следующими классами *projectsController*, *projectsService* и *projectsRepository*. Эта часть системы позволяет относить данные к определенным проектам, управлять конфигурацией и структурой проекта. Под проектом понимается система, данные которой поступают в программный комплекс. Как правило, одному проекту соответствует одна анализируемая система.

Модуль *Performance* содержит *performanceController* и *performanceService*, отвечающие за выполнение анализа приложения. *Performance* взаимодействует с *Measurements* и *Projects* для получения и интерпретации данных, обеспечивая проведение комплексного анализа, выявление отклонений и генерацию показателей работы системы. Вспомогательные классы, такие как *globalErrorHandler*, обеспечивают централизованную обработку исключений, а также стабильность и отказоустойчивость системы. Компоненты *Spring*, такие как *webApplicationContext* и *gridFsTemplate*, обеспечивают доступ к контексту приложения и работе с файловыми хранилищами.

3.2 Мониторинг времени отклика

Время отклика веб-страницы представляет количественную метрику производительности и ее анализ, является простым, но важным шагом для анализа работы системы. Полученные результаты содержат набор значений, отражающих время отклика страницы в конкретные моменты времени. Однако недостаточно полагаться исключительно на эти значения для получения полного представления о потенциальном ухудшении работы системы. Проблемы в клиент-серверном приложении могут возникать при изменении его состояния в момент его эксплуатации. Приложение может выйти из строя и возвращать ошибку при этом обладать минимальным временем отклика. Такой случай является ложно позитивным срабатыванием, что ведет к неправильному обнаружению проблем в программном обеспечении. Отсюда следует, что для более глубокого анализа системы необходимо учитывать ее дополнительные метрики и факторы, влияющие на производительность, и применять комплексный подход к интерпретации данных.

В рамках данной работы сбор данных, моделирование и мониторинг времени отклика являются основными элементами исследования. Они направлены на минимизацию времени отклика веб-страниц в клиент-серверных веб-приложениях как на этапе их разработки, так и в процессе эксплуатации. На рисунке 3.6 показана схема работы мониторинга времени отклика.

Рисунок 3.6 – Процесс мониторинга времени отклика

Разработанный программный комплекс собирает данные, полученные от клиента, агрегирует и записывает в *MongoDB*. В дальнейшем эта информация используется для ведения мониторинга и дальнейшего анализа. На основе

полученных данных были выделены характеристики, влияющие на работу веб-страниц. На их основе были разработаны модели, представляющие время отклика как функцию от данных параметров. Эти модели служат инструментом для оценки качества проектирования веб-страниц с точки зрения производительности, обеспечивая возможность анализа и улучшения их структуры.

Мониторинг времени отклика осуществляется на стороне сервера посредством получения записей из базы данных, что позволяет наблюдать производительность веб-страниц в реальном времени. Такой подход предоставляет средства для выявления страниц, время отклика которых не соответствует ожидаемым показателям, а также для определения приоритетных объектов для улучшения с целью повышения общей производительности системы.

Общий отчет предоставляет пользователю сводную информацию о веб-страницах, включая их список, среднее время отклика и количество запросов, полученных для каждой страницы. Дополнительно, система позволяет получить детализированный отчет о конкретной странице. Для этого достаточно выбрать соответствующую запись в таблице, которая содержит дополнительную информацию для выбранной записи.

Для минимизации влияния внешних факторов, которые не связаны с характеристиками веб-страницы, на время ее отклика, эксперименты проводились в контролируемой среде, а именно в пределах локальной сети. Такой подход позволяет существенно уменьшить влияние переменных сетевых условий и сосредоточиться на изучении свойств самой веб-страницы. Изоляция факторов, не связанных с характеристиками страницы, является ключевым условием для точного определения их значимости в формировании времени отклика. Для этого был развернут выделенный сервер, что позволило исключить возможное воздействие сторонних пользовательских запросов и обеспечить строгое соблюдение экспериментальных условий. Собранные в ходе экспериментов эмпирические данные были проанализированы с целью выявления взаимосвязей между характеристиками веб-страницы и ее временем отклика.

На основе результатов анализа были разработаны модели, позволяющие описать эти взаимосвязи в простой и понятной форме, сохраняя при этом их точность и практичность. Для реализации исследования были созданы программные модули, предназначенные для сбора данных, мониторинга производительности и анализа. Метод сбора данных с использованием программного комплекса был выбран в силу его гибкости и адаптивности, что обеспечивает возможность масштабирования и модификации системы под различные требования.

3.3 Сценарии тестирования

Динамическая страница и клиент-серверное веб-приложение было разработано с использованием *JavaScript*-фреймворка *Next.js,* который является одним из самых популярных решений для разработки согласно статистике на сайте пакетного менеджера для *NodeJS* [101]. Для проведения тестов страниц были разработаны три тестовых сценария.

При первом сценарии пользователям предоставляется доступ к дополнительным объектам страницы без кэширования. Объекты запрашиваются и обслуживаются сервером и в его логах указывается код 200 после успешного ответа в рамках протокола *HTTP*.

При втором сценарии происходит кэширование дополнительных объектов. Они проверяются сервером на актуальность до того, как происходит извлечение из кэша. Ответы на запросы имеют статус 304. Данный статус указывает на то, что объекты являются актуальными и могут использоваться повторно.

При третьем сценарии также происходит кэширование дополнительных объектов, но проверка от сервера не выполняется. Сервер мгновенно извлекает их из локального кэша. Запросы не попадают в логи сервера и не проходят через прокси сервер. При таком сценарии получается только время рендеринга страницы.

Наличие пустых полей в результирующем наборе значений обусловлено ограничениями каждого из рассматриваемых методов. Прокси-сервер, способный перехватывать и анализировать *HTTP*-трафик, имеет расширенный

набор сетевых и объектных параметров, но создает дополнительную задержку из-за промежуточной обработки запросов и ответов [109].

Для метода, который основан на фреймах (фреймы), целевая страница загружается во вложенном *iframe*-элементе, а отдельный управляющий фрейм инициирует навигацию, фиксирует момент начала, ожидает событие завершения загрузки (*load* или *DOMContentLoaded*) во вложенном фрейме. Данный метод ограничен политиками безопасности браузера и контекстом исполнения и оценивает лишь суммарные тайминги загрузки и рендеринга, что приводит к неполноте данных по отдельным встроенным объектам и внутренним этапам обработки.

Разработанный программный комплекс обладает низкими вычислительными расходами и предоставляет отчет по собранным данным, но не учитывает захват сетевого трафика и не располагает механизмами детального профилирования сетевых взаимодействий, что делает невозможным сбор таких данных, как временные затраты на передачу страницы по сети, общее время между отправкой запроса и получением ответа, а также полезное время, связанное с обработкой и отображением контента.

3.4 Влияние кэширования

В рамках первого сценария оценивается поведение веб-приложения при условии отсутствия механизма кэширования встроенных объектов. Каждый дополнительный объект, который необходим для корректного отображения и функционирования страницы, запрашивается непосредственно с сервера при каждом обращении. Все обращения сопровождаются полной передачей содержимого, что фиксируется в логах с кодом состояния 200 в соответствии с протоколом *HTTP*. Был проведен сбор данных с использованием прокси-сервера для сценария, когда доступ к странице происходит без кэширования (таблица 3.1).

Таблица 3.1 – Доступ к веб-странице без кэширования

Параметр	Первый сбор данных динамической страницы без использования кэширования			Повторный сбор данных динамической страницы без использования кэширования		
	Прокси-сервер	Программный комплекс	Фреймы	Прокси-сервер	Программный комплекс	Фреймы
T_{OTV}	3126	2941	2946	424	321	323
T_{RB}	2490	2291	2284	447	296	293
T_{ZO}	2519	-	-	86	-	-
T_G	4501	4411	4408	253	276	267
T_{US}	2463	-	-	76	-	-
T_{OP}	4426	-	-	395	-	-
T_{US} / T_{OP}	0,45	-	-	0,19	-	-
T_{ZO} / T_G	0,55	-		0,33	-	-

Данные, полученные с помощью метода, основанном на методе фреймов, практически совпадают с результатами программного комплекса и существенно ниже значений, полученных через прокси-сервер. При первичном доступе к странице значение T_{OTV} для фреймов и программного комплекса равны 2946 и 2941 миллисекунды соответственно. При повторном доступе 323 и 321 миллисекунды. Разница не превышает 0,2-0,6%, что укладывается в допустимую область разброса результатов и отражает тот факт, что методы являются схожими по той причине, что данные собираются на клиентской части. Такой подход не имеет дополнительный сетевой узел, поэтому исключается дополнительное время задержки.

При проведении эксперимента с использованием прокси-сервера время отклика составляет 3126 миллисекунд, а при использовании программного комплекса 2941 миллисекунд. Время отклика для первичного доступа к странице больше, чем для последующего в 7 раз (3126 / 424), а для программного комплекса в 9 раз (2941 / 321). Такая разница заключается в том, что прокси-сервер вносит дополнительные временные расходы. Он создает дополнительное звено в цепочке взаимодействия, пересылая данные через себя,

что искажает время отклика. Дополнительное время, которое добавляется прокси-сервером, составляет 185 миллисекунды для первичного доступа (3126 – 2941). Для последующего доступа оно составляет 103 миллисекунды. В процентах для первичного доступа это равно 6 % (185 / 2941). Для последующего доступа это равно 32 % (103 / 321).

Увеличенное время отклика (T_{OTV}) при первом запуске клиент-серверного приложения было вызвано так называемым «холодным запуском», при котором необходимо больше времени для инициализации приложения и его зависимостей. Это происходит при первом запросе после того, как приложение было неактивно или перезапущено. Здесь стоит отметить, что дополнительные временные затраты являются несущественными при первом доступе, тогда как при последующем доступе они увеличиваются. Несмотря на то, что время отклика является уменьшенным при последующем доступе к странице вследствие малого количества пользователей и использовании локальной сети.

При первом доступе к веб-странице отношение T_{ZO} / T_G является 0,55. Полученное значение означает, что время, занимаемое сервером для обработки запроса, имеет значительное влияние на время отклика (T_{OTV}). Время, затраченное на обработку клиентского запроса, составляет 4394 миллисекунд, что занимает 97% (3494 / 4501) от времени генерации (T_G), а время получения ответа в среднем составляет 2122 миллисекунд, что является 84% (2122 / 2519) от T_{ZO}.

На основе полученных данных сделаем вывод о том, что время ожидания времени отклика для первого ответа от сервера для клиентской части значительно выше по причине «холодного запуска» сервера. Выявлено, что время для обработки сервером запроса вносит дополнительное время задержки в общее время отклика.

Показатель полезного времени (T_{US}), который составляет 2023 миллисекунды, что меньше T_{ZO}, который составляет 2519 миллисекунд на 56 миллисекунд. Данное значение дает понять, что происходит несущественное наложение запросов. Отношение T_{US} / T_{OP} дает результат 0,45, что является относительно эффективным значением для передачи данных.

При последующих запросах отношение T_{ZO} / T_G составляет 0,33, что является меньшим значением по сравнению со значением 0,55, полученным при первом запросе к странице. Обработка клиентского запроса составила 242 миллисекунды, что в свою очередь образует 95% (242 /253) от времени генерации. Дополнительно отношение T_{US} / T_{OP} имеет значение 0,19, что является показателем эффективной передачи. Полученные данные дают понимание того, что время обработки на сервере при последующих запросах составляет большую часть для времени генерации и при отсутствии «холодного запуска» произошло уменьшение времени отклика от сервера.

3.5 Проведение сбора данных с использованием кэширования и без него

Во втором сценарии анализировалось влияние механизма кэширования на время отклика веб-страницы. Для обеспечения корректности и воспроизводимости результатов сбора данных процесс осуществлялся в условиях предварительно инициализированного окружения, что позволило исключить эффект «холодного старта». Полученные результаты представлены в таблице 3.2.

Таблица 3.2 – Полученные данные с использованием кэширования и без

Параметр	Предыдущие значения сбора данных динамической страницы без использования кэширования			Следующий повторный сбор данных динамической страницы с использованием кэширования		
	Прокси-сервер	Программный комплекс	Фреймы	Прокси-сервер	Программный комплекс	Фреймы
T_{OTV}	424	321	317	240	190	191
T_{RB}	447	296	297	210	160	158
T_{ZO}	86	-		40	-	
T_G	253	276	270	110	90	90
T_{US}	76	-		30	-	
T_{OP}	395	-		190	-	
T_{US} / T_{OP}	0,19	-		0,16	-	

T_{ZO}/T_G	0,33	-		0,36	-	

Данные, полученные методом, который основан на фреймах, подтверждают «клиентский» характер полученных данных. Полученные значения практически совпадают с данными программного комплекса как при повторном доступе без кэширования. Разброс в пределах 1 и 3 миллисекунд объясняется микрозадержками обмена сообщениями и планированием задач браузера. Он не носит систематического характера. Однако, прокси-сервер добавляет, вносит дополнительные временные задержки. Для повторного доступа без кэширования разница с клиентскими методами составляет 103 миллисекунды (32%), а при кэшировании – 50 миллисекунд (26%). Задержка для прокси снижается благодаря переиспользованию соединений и устранению эффекта «холодного старта».

Полученные результаты с использованием прокси-сервера и программного комплекса, позволяют определить величину дополнительных временных расходов, которые обусловлены функционированием прокси-сервера. В частности, для первого доступа к странице без использования кэширования дополнительные временные затраты для времени отклика (T_{OTV}) составляют 103 мс, в то время как при последующих обращениях с активированным кэшированием величина накладных расходов снижается до 50 мс.

Из полученных данных можно увидеть, что все-таки существуют временные затраты, вносимые прокси-сервером, но с более низкими значениями времени отклика. Показатели характерны для локальной сети. Согласно данным, полученным с использованием программного комплекса, применение механизма кэширования способствует сокращению времени отклика на 131 мс, что эквивалентно снижению показателя на 40 % по сравнению с обращениями без кэширования (131/321). Улучшение во времени заметно и для T_{RB}, T_{ZO} и T_G. С другой стороны, для T_{ZO}/T_G улучшение произошло с изменением T_{ZO} на 36%, что превышает T_G. Такое поведение объясняется тем, что применение кэширования позволяет существенно

сократить объем данных, передаваемых от сервера к клиенту, за счет локального извлечения ранее загруженных объектов из клиентского кэша. Более того механизм кэширования снижает вычислительную нагрузку на сервер, освобождая его от операций поиска запрашиваемых ресурсов в файловой системе и формирования ответа, что, в свою очередь, приводит к сокращению времени T_G [110].

Политика кэширования веб-ресурсов является критически-важным параметром, определяющим общую эффективность использования кэша в клиент-серверных веб-приложениях. В рассматриваемом эксперименте кэшированные объекты подлежали предварительной проверке на актуальность со стороны сервера до момента их извлечения и отображения в браузере на клиентской стороне веб-приложения. Такой механизм проверки вносит дополнительную нагрузку для коммуникации между клиентом и сервером, что приводит к увеличению времени рендеринга.

3.6 Сравнение полученных метрик для статической и динамической страниц

В таблице 3.3 представлено сравнение результатов полученных значений для статических и динамических веб-страниц. Проведенное исследование позволяет более точно определить вклад каждой составляющей в общее время отклика, а также выявить особенности, влияющие на производительность при работе с различными типами веб-страниц.

Таблица 3.3 – Полученные значения для динамической и статической страниц без кэширования

Параметр	Предыдущие значения для динамической страницы без использования кэширования			Значения для статической страницы без использования кэширования		
	Прокси-сервер	Программный комплекс	Фреймы	Прокси-сервер	Программный комплекс	Фреймы

Продолжение таблицы 3.3

T_{OTV}	424	321	314	324	165	162
Параметр	Предыдущие значения для динамической страницы без использования кэширования			Значения для статической страницы без использования кэширования		
	Прокси-сервер	Программный комплекс	Фреймы	Прокси-сервер	Программный комплекс	Фреймы
T_{RB}	447	296	295	205	118	116
T_{ZO}	86	-		74	-	
T_G	253	276	278	8	8	11
T_{US}	76	-		70	-	
T_{OP}	395	-		260	-	
T_{US} / T_{OP}	0,19	-		0,27	-	
T_{ZO} / T_G	0,33	-		15,1	-	

Значения, полученные с помощью фреймов и программного комплекса, являются практически идентичными и существенно ниже времени, регистрируемого прокси-сервером. Для динамической страницы без кеширования T_{OTV} равно 314 и 321 миллисекунд соответственно (расхождение около двух процентов), T_{RB} является 295 и 296 миллисекунд, T_G 278 и 276 миллисекунд. Для статической страницы (без кеширования) тенденция сохраняется: 162 и 165 миллисекунд для T_{OTV}, 116 и 118 миллисекунд для T_{RB}, 11 и 8 мс для T_G. Переход от динамической к статической странице резко уменьшает влияние времени генерации на сервере.

Сбор данных, выполненный с помощью программного комплекса, показывает меньшие временные затраты, чем при использовании прокси-сервера. Так, например, T_{OTV} сократился для статической страницы на 156 миллисекунд или на 48% (154/321). Стоит отметить, что T_G минимально для

статической страницы. Данное поведение объясняется тем, что генерация сервером динамической страницы является гораздо более интенсивной задачей.

По причине того, что T_G уменьшилось, то время T_{ZO} стало иметь более важную роль влияния на T_{OP}, что отражено в отношении T_{ZO} / T_G (0,33 и 15,1 соответственно). Отношение T_{US} / T_{OP} имеет значение 0,27, что говорит о том, что передача все еще является эффективной. Причина этому является использование локальной сети, в которой такие метрики как пропускная способность сети и эффективность передачи данных не влияют на T_{OTV}.

3.7 Исследование взаимосвязей между полученными результатами

Время генерации страницы на сервере в основном зависит от характеристик сервера, а также его загруженности, а время передачи зависит от характеристик соединения, в рамках которого выполняется данная передача. Однако время рендеринга, традиционно рассматриваемое как параметр, определяемый клиентским браузером, в действительности определяется рядом факторов, включая, насколько быстро может быть выполнена передача страницы и всех ее дополнительных объектов, что, в свою очередь, зависит от эффективности генерации и передачи. Такая особенность работы браузера несмотря на относительную простоту получения данных о времени рендеринга, создает необходимость в изоляции времени рендеринга от остальных временных компонентов, связанных с передачей и генерацией страницы, с целью корректной интерпретации значимости каждого временного компонента в общем времени отклика.

Для того, чтобы решить данную проблему, был выполнен корреляционный анализ между метриками T_G, T_{ZO} и T_{RB} при помощи коэффициента корреляции Пирсона для изучения связи между ними. В таблице 3.4 представлены полученные значения.

Таблица 3.4 – Значения корреляции между T_{ZO}, T_{RB}, T_G

Сценарий/Параметры	T_{ZO},T_{RB}	T_{ZO},T_G	T_{RB},T_G
Холодный запуск	-0,13	0,17	0,72
Последующий доступ к динамической странице с использованием кэширования	-0,05	0,22	0,71
Последующий доступ к динамической странице без использования кэширования	-0,11	0,11	0,64

Параметр T_{ZO} не имеет выраженной корреляции с T_{RB} и T_G. Стоит отметить, что T_{ZO} и T_{RB} имеют отрицательную взаимосвязь, а T_{ZO} и T_G имеют слабую взаимосвязь, что обусловлено спецификой их определения. T_{ZO} – совокупное время обслуживания веб-страницы и всех ее встроенных ресурсов, однако, учитываемое время охватывает лишь интервал от отправки запроса до начала получения первого байта ответа, не охватывая полный объем передаваемых данных.

Замена T_{ZO} на T_{OP} в качестве показателя передачи данных изменяет ситуацию, делая корреляцию между временными компонентами T_{OP}, T_{RB} и T_G положительной (таблица 3.5). Использование T_{OP} вместо T_{ZO} способствует изменению взаимосвязей между временными компонентами. В частности, наблюдается положительная корреляция между T_{OP}, T_G и временем рендеринга на клиентской стороне T_{RB}. Это объясняется тем, что T_{OP} охватывает более широкий диапазон процессов, включая возможные задержки в сети и начальные этапы получения данных, что делает его более информативным показателем в контексте анализа общей производительности системы.

Таблица 3.5 – Значения корреляции между T_{OP}, T_{RB}, T_G

Сценарий/Параметры	T_{OP},T_{RB}	T_{OP},T_G
Холодный запуск	0,60	0,49
Последующий доступ к динамической странице с использованием кэширования	0,54	0,94
Последующий доступ к динамической странице без использования кэширования	0,51	0,71

Полученные коэффициенты корреляции могут быть использованы в качестве диагностического инструмента для оценки состояния производительности веб-системы. В частности, если для веб-страницы наблюдается более высокая корреляция между длительностью передачи данных T_{RB} и временем генерации ответа на сервере T_G, по сравнению с эталонными или идеальными условиями, это может указывать на наличие проблем в серверной части системы. Такая аномалия, как правило, обусловлена снижением производительности сервера при формировании и выдаче содержимого страницы, что ведет к увеличению времени отклика.

Выводы по главе

Несмотря на то, что разработанные методы сбора данных о производительности клиент-серверных веб-приложений имеют общую цель, каждый из них обладает своими особенностями. Каждый ориентирован на специфичные аспекты работы системы. Их различия обуславливают выбор метода в зависимости от требований к точности получаемых данных, масштабируемости решения и характера исследуемой нагрузки.

Метод, основанный на прокси-сервере, дает возможность проводить детализированный анализ временных характеристик, составляющих общее время

отклика веб-страницы. Более того он позволяет раздельно извлекать данные о задержках, обусловленные получением основного *HTML*-документа и его дополнительных объектов (изображений, скриптов, стилей). Это в свою очередь, дает возможность определить вклад структуры веб-страницы и каждого из дополнительных объектов в совокупное время отклика. Несмотря на дополнительные временные затраты, связанные с самим процессом сбора данных, данный подход является высокоэффективным для анализа относительных задержек и проведения сравнения между временными компонентами. Дополнительно следует отметить, что развертывание данного инструмента не требует модификации клиентской или серверной части веб-приложения, что делает его универсальным и независимым от специфики тестируемой среды.

С другой стороны метод сбора данных о производительности с использованием программного комплекса имеет низкие дополнительные временные затраты и позволяет достоверно зафиксировать общее время отклика с позиции пользовательского опыта. И хотя данный метод не показывает влияние дополнительных объектов на время отклика страницы, он является простым и эффективным методом сбора данных, а также не требует изменений в коде сервера и клиента.

Разработанное программное обеспечение для сбора данных о производительности применялось для анализа различных аспектов, влияющих на время отклика веб-страниц, включая кэширование, степень динамичности контента.

Разработанный программный комплекс для сбора данных производительности использовался в исследовании для анализа факторов, которые влияют на время отклика веб-страниц, включая особенности кэширования и степень динамичности отображаемого контента. В ходе экспериментов была проведена оценка временных характеристик при различных сценариях загрузки страниц, с учетом наличия или отсутствия кэшированных ресурсов, а также характера взаимодействия с серверной частью. Полученные результаты позволили выявить закономерности и количественно оценить

влияние архитектурных решений и механизмов управления состоянием на общую производительность клиент-серверных веб-приложений.

4. ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТАЛЬНОГО ТЕСТИРОВАНИЯ

На основе представлений сложности создания веб-страницы C_1 и сложности представления веб-страницы C_2 можно определить время ответа для страницы как функция от C_1 и C_2. Были составлены эталонные наборы веб-страниц и применены модели, которые отражают сложность генерации веб-страниц и сложность их контента. В качестве веб-браузера использовался *Google Chrome Browser* версии 133. Веб-страницы обслуживались с одного и того же сервера, и доступ к ним осуществлялся в одной локальной сети с сервером, чтобы минимизировать влияние внешних факторов, особенно сетевых условий, на время отклика [111].

Имея сложность генерации страницы C_1 и сложность контента C_2, которые определены для веб-страницы, время отклика страницы *R* может быть определено как функция C_1 и C_2. Отсюда можно сформировать следующее выражение для времени отклика веб-страницы: $R = f(C_1, C_2)$. Для эмпирической проверки построенных моделей используется метод сбора данных, основанный на программном комплексе, описанный во второй главе.

4.1 Количество строк кода

Первый набор из четырех статических веб-страниц отличается по количеству строк кода. Страница 1а содержит 104 строки и ее размер составляет 2 КБ, в то время как страница 1б содержит 14 строк и ее размер также составляет 2 КБ. Страница 1в содержит 104 строки и 15КБ, а страница 1г имеет 1000 строк и размер 15КБ. Каждая строка кода создает контент, отображаемый в веб-браузере [112]. Однако строка кода на страницах 1б и 1в приводит к более длинной строке текста, отображаемого в веб-браузере, чем строка кода на страницах 1а и 1г. Однако, страницы 1а и 1б в конечном итоге, отображают одинаковое общее количество символов. Страницы 1в и 1г

отображают одинаковое общее количество символов в браузере, но больше, чем страницы 1а и 1б. Страница 1г является наиболее ресурсоемкой, а за ней следуют страницы 1а и 1в. Страница 1б является самой небольшой из них. На страницах нет дополнительных объектов или элементов, таких как *CSS*. В таблице 4.1 можно увидеть детали приведенных страниц, которые используются для моделирования их сложности.

Таблица 4.1 – Детали страниц

Параметр / Название страницы	Страница 1а	Страница 1б	Страница 1в	Страница 1г
Строки кода	104	14	104	1000
Программные конструкции	0	0	0	0
Количество дополнительных элементов	0	0	0	0
Внутренние состояния	0	0	0	0
Размер (КБ)	2	2	15	15
Количество элементов (JS/CSS)	0	0	0	0

Поскольку четыре страницы статичны, их C_1 является функцией от количества строк кода в форме $linecode \cdot w_1$ где $linecode$ обозначает количество строк кода, а w_1, обозначает вес выполнения, который назначен этой метрике. Таким образом, C_1 для страниц 1а, 1б, 1в и 1г составляет $104w_1$, $14w_1$, $104w_1$ и $1000w_1$ соответственно.

Для каждой из этих страниц было собрано 100 значений времени отклика и рендеринга. Таблица 4.2 показывает средние значения (μ) и стандартные отклонения (σ) для полученных данных, а Рисунок 4.1 показывает их распределение.

Таблица 4.2 – Полученные временные показатели веб-страниц для сравнения влияния строк кода

Значение / Номер страницы	Страница 1а (μ)	Страница 1а (σ)	Страница 1б (μ)	Страница 1б (σ)	Страница 1в (μ)	Страница 1в (σ)	Страница 1г (μ)	Страница 1г (σ)
Время рендеринга (мс)	15,8	0,58	15,2	0,48	16,1	0,57	31,3	0,48
Время ответа (мс)	31,9	0,32	31,2	0,42	47,2	0,42	62,4	0,52
Время до первого рендеринга(мс)	15,5	0,42	15,9	0,32	31,1	0,32	31,1	0,29

Рисунок 4.1 отражает распределение полученных временных показателей веб-страниц для сравнения влияния количества строк кода.

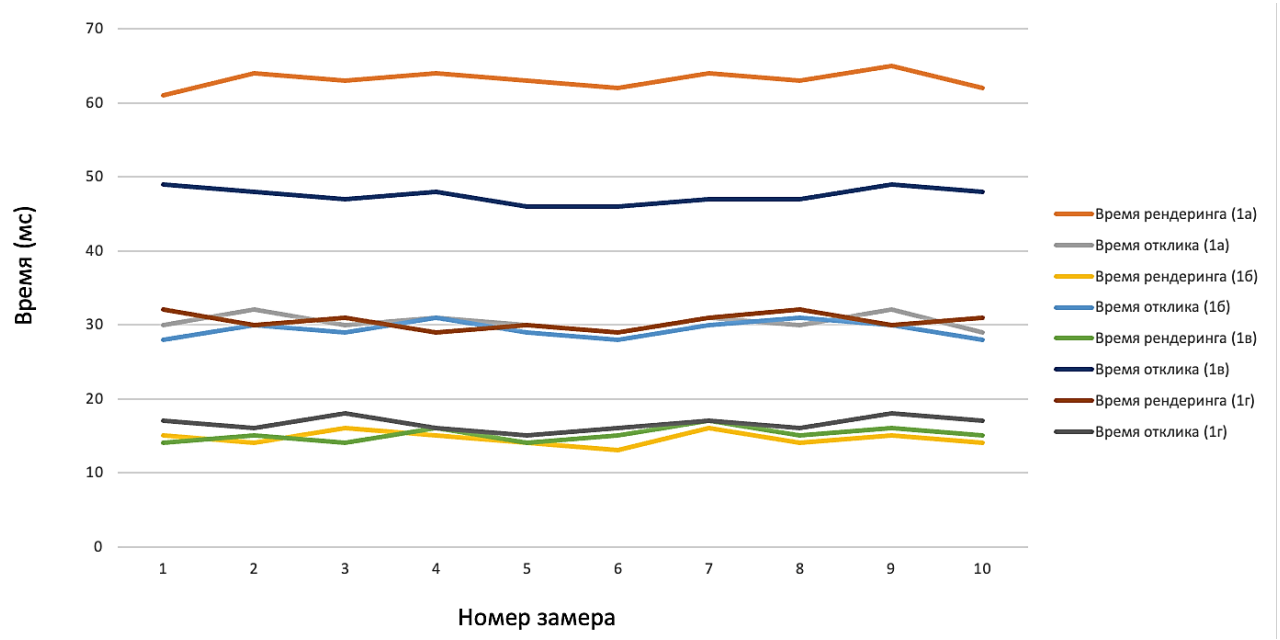


Рисунок 4.1 – Распределение полученных временных показателей веб-страниц для сравнения влияния количества строк кода

Из полученных данных видно, что различия во времени, зафиксированном для страниц 1а и 1б, незначительны и вряд ли будут заметны конечным пользователям. При сравнении страниц 1а и 1в, где каждая из которых содержат 104 строки кода, наблюдается одинаковый показатель времени рендеринга. Возникающее при этом расхождение во времени отклика в основном связано с тем, сколько было затрачено времени от отправки запроса до начала отображения страницы. Время между отправкой запроса и началом отображения страницы остается практически неизменным при увеличении количества строк кода с 104 (страница 1в) до 1000 (страница 1г). Аналогично, даже при возрастании количество строк кода с 104 (страница 1а) до 1000 (страница 1г) время отклика увеличивается всего на 29 миллисекунд. Согласно полученным значениям, величина w_1 остается весьма незначительной и определяется преимущественно производительностью клиентского браузера при рендеринге страниц. Более того различия в продолжительности времени от момента отправки запроса до полного отображения страницы в первую очередь обусловлены различиями в размере самих веб-страниц. Говоря о размере страниц 1а и 1б, можно сказать, что они одинаковые, а страницы 1в и 1г схожи. Среднее время, необходимое браузеру для начала отображения, составляет около 16 миллисекунд, чтобы начать отображать страницы 1а и 1б с момента отправки запроса, а для страниц 1в и 1г продолжительность составляет около 30 миллисекунд.

Данные результаты свидетельствуют о том, что условия на стороне сервера и сети оставались неизменными для обработки запроса и передачи ответа. Тем не менее, страница 1а демонстрирует более высокое время отклика по сравнению со страницей 1б, а страница 1г является более быстрой по сравнению со страницей 1в. Это указывает на то, что страница, имеющая большее количество строк кода, обычно характеризуется несколько увеличенным временем рендеринга и отклика, чем страница аналогичного размера с меньшим количеством строк кода. Следовательно, полученные

значения подтверждают влияние этой метрики на время отклика веб-страницы R, соответствуя зависимости: $R = g_{line}(linecode)$.

4.2 Количество дополнительных объектов

Второй набор из трех статических веб-страниц в основном различается количеством дополнительных объектов (*JPEG*-изображений). Страница 2а содержит 4 объекта, страница 2б – 10, а страница 2в – 13. Все изображения обладают одинаковым разрешением (1024 × 1024 пикселей), а их общий размер на каждой странице идентичен и составляет 2500 КБ. Таблица 4.3 содержит подробные характеристики этих страниц, учитываемые при моделировании их сложности.

Таблица 4.3 – Сведения о страницах, различающихся по числу дополнительных объектов

Параметр / Название страницы	Страница 2а	Страница 2б	Страница 2в
Строки кода	13	18	23
Программные конструкции	0	0	0
Внутренние состояния	0	0	0
Количество дополнительных объектов	4	10	13
Размер (КБ)	2501	2501	2501
Количество элементов (JS/CSS)	0	0	0

Для каждой страницы были получены 100 значений времени отклика. В таблице 4.4 приведены средние значения (μ) и стандартные отклонения (σ), полученные на основе результатов этих значений.

Таблица 4.4 – Влияние количества дополнительных объектов

Значение / Номер страницы	Страница 2а (μ)	Страница 2а (σ)	Страница 2б (μ)	Страница 2б (σ)	Страница 2в (μ)	Страница 2в (σ)
Время рендеринга (мс)	920	24	1474	33	1992	37
Время отклика (мс)	953	25	1525	25	2058	35
Время до первого рендеринга (мс)	32	6	49	16	63	19

Полученные данные подтверждают, что время отклика веб-страницы увеличивается пропорционально числу дополнительных объектов. Такое поведение связано с тем, что с увеличением количества таких объектов браузеру требуется больше времени на их загрузку и рендеринг, а также возрастает количество запросов между клиентом и сервером (включая установку или повторное установление *TCP*-соединений). Чем сложнее визуальная часть страницы и активнее обмен данными между клиентом и сервером, тем сильнее происходит колебание значений стандартного отклонения времени отклика [113]. В публичной сети подобное увеличение времени отклика может проявляться еще более заметно, поскольку сетевое окружение характеризуется непредсказуемостью и ограниченно контролируется. Таким образом, при росте числа дополнительных объектов

становится сложнее гарантировать надлежащее качество обслуживания веб-страницы в аспекте времени ее отклика.

Полученные данные подтверждают, что рост времени отклика в определенной степени демонстрирует линейную корреляцию с увеличением количества дополнительных объектов, что продемонстрировано на рисунке 4.2. Дополнительно установлено, что при возрастании числа дополнительных объектов браузеру требуется более продолжительное время, чтобы инициировать процесс отображения ответа, поступившего от сервера после отправки запроса.

Такое поведение обусловлено необходимостью предварительного анализа *HTML*-содержимого и построением структуры страницы до ее полной компоновки [1]. При увеличении количества дополнительных объектов этапы рендеринга страницы становятся более ресурсоемкими, что приводит к небольшому, но заметному возрастанию времени, требуемого на отображение страницы.

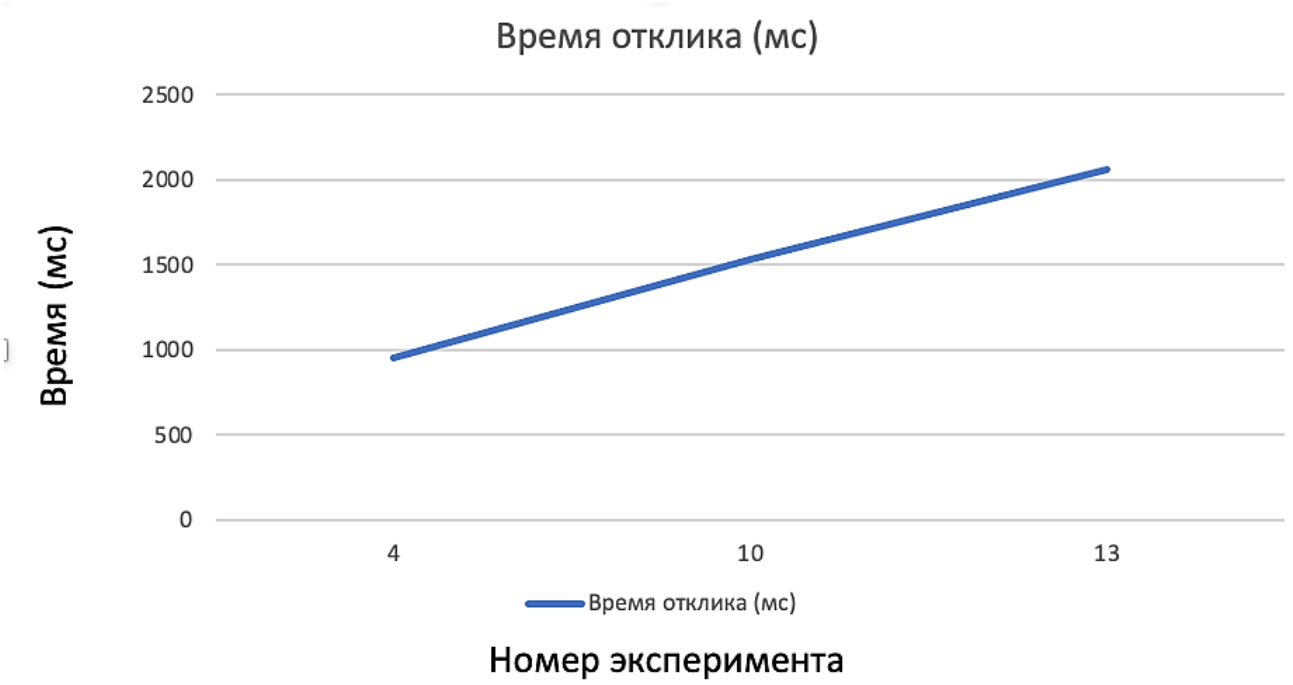


Рисунок 4.2 – Время отклика как функция количества дополнительных объектов

Рисунок 4.3 отражает распределение полученных значений и их тесную корреляцию между временем отклика и временем рендеринга для страниц. Изучение корреляции между двумя временными метриками показывает, что коэффициенты корреляции составляют 0,982 (страница 2а), 0,852 (страница 2б) и 0,886 (страница 2в).

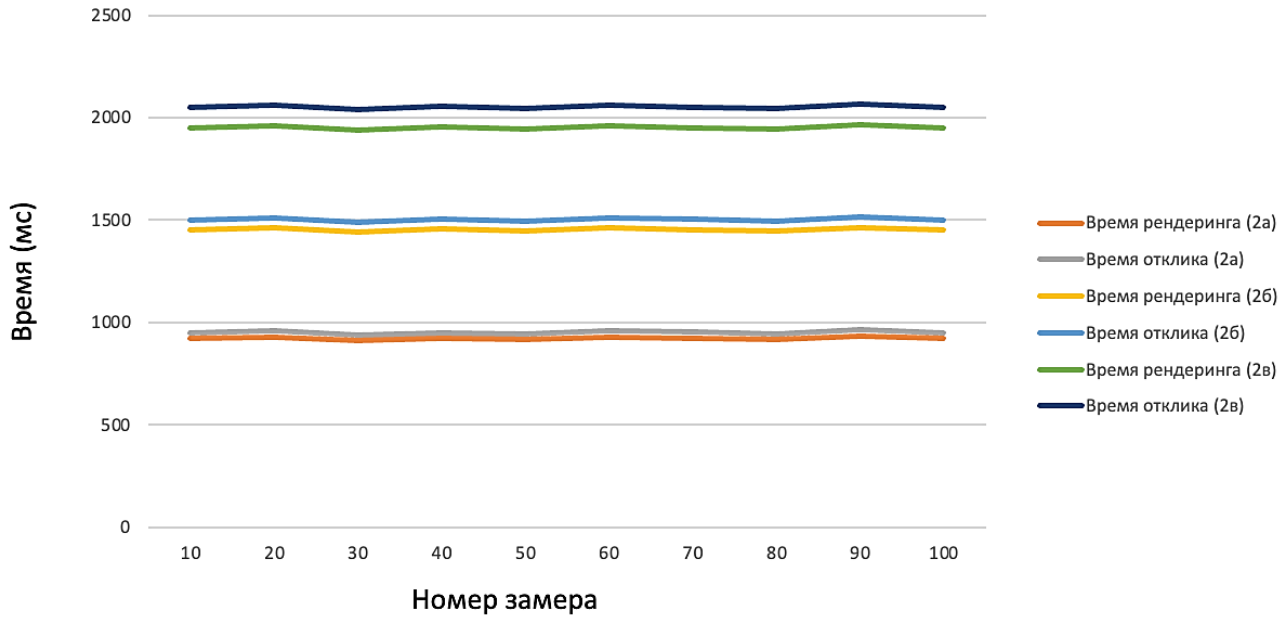


Рисунок 4.3 – Сравнения влияния количества дополнительных объектов

Данные также подтверждают, что время отклика R веб-страницы можно выразить как функцию количества дополнительных объектов $R = g_{assets}\left(elm_o\right)$.

4.3 Общий размер страницы

Третий набор из трех статических веб-страниц характеризуется различиями в общем размере. Каждая страница включает по 5 дополнительных объектов (*JPEG*-изображений) с одинаковым разрешением для 3а (512 × 512 пикселей), 3б (1024 × 1024 пикселей), 3в (1280 × 1280). Однако размеры таких объектов изменены, что обусловливает различия в общем объеме данных страниц. Детальные характеристики общего размера страниц представлены в таблице 4.5.

Таблица 4.5 – Общий размер тестовых страниц

Параметр / Название страницы	Страница 3а	Страница 3б	Страница 3в
Строки кода	13	13	13
Программные конструкции	0	0	0
Внутренние состояния	0	0	0
Количество дополнительных объектов	5	5	5
Размер (КБ)	1511	1998	2509
Количество элементов (JS/CSS)	0	0	0

Веб-страницы 3а, 3б и 3в отличаются только по их размеру. При обозначении общего размер страницы как z_{sp}, то ее сложность может быть выражена как $C_2 = g_{size}\left(z_{sp}\right)$. Для каждой страницы было получено 100 значений времени отклика. Любое изменение во времени отклика происходит по причине изменения метрики z_{sp} соответствующей страницы. Таблица 4.6 содержит сведения о средних значениях и стандартных отклонениях результатов.

Таблица 4.6 – Сравнение влияния общего размера страницы

Значение / Номер страницы	Страница 3а (μ)	Страница 3а (σ)	Страница 3б (μ)	Страница 3б (σ)	Страница 3в (μ)	Страница 3в (σ)
Время рендеринга (мс)	613	17	669	11	711	15
Время отклика (мс)	651	14	702	9	750	16
Время до первого рендеринга (мс)	41	6	35	7	30	5

С увеличением общего размера страницы z_{sp}, возрастает и ее время отклика R. Это обусловлено, в частности, увеличением объема данных, передаваемых по сети от сервера к клиенту. На рисунке 4.4 представлена диаграмма, где видна линейная зависимость между R и z_{sp}. В сети Интернет, где объем передаваемых данных возрастает из года в год, влияние внешних факторов, таких как сетевые задержки и потеря пакетов, становится более заметным. Это может привести к тому, что зависимость между R и z_{sp} станет экспоненциальной. Тем не менее, представленные результаты демонстрируют возможность описания времени отклика R как функции от общего размера страницы z_{sp}, что можно выразить как $R = g_{size}\left(z_{sp}\right)$.

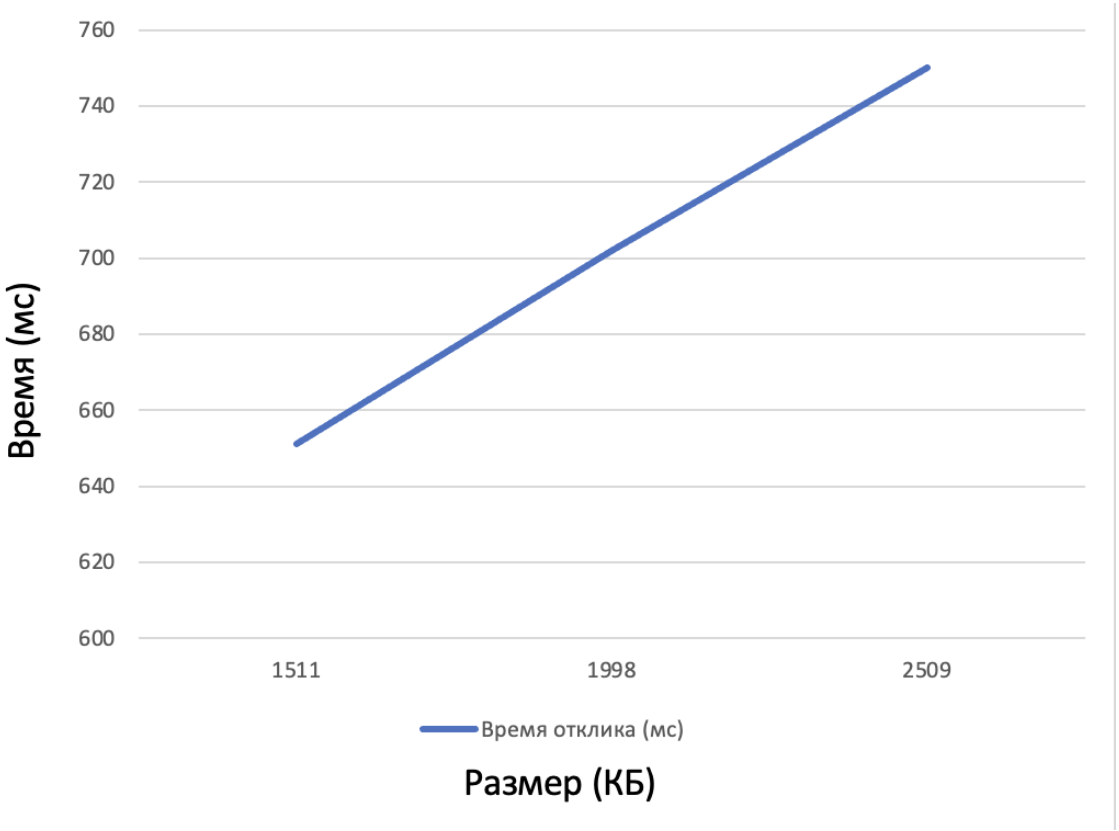


Рисунок 4.4 – Сравнение влияния количества размера страницы и ее дополнительных объектов на время отклика

Рисунок 4.5 иллюстрирует распределение полученных результатов. Коэффициенты корреляции между временем рендеринга и временем отклика для всех трех страниц превышают значение 0,73. Это наблюдение сохраняется для большинства веб-страниц при условии, что задержки обработки на сервере минимальны, как, например, в среде локальной сети, в которой проводился сбор данных.

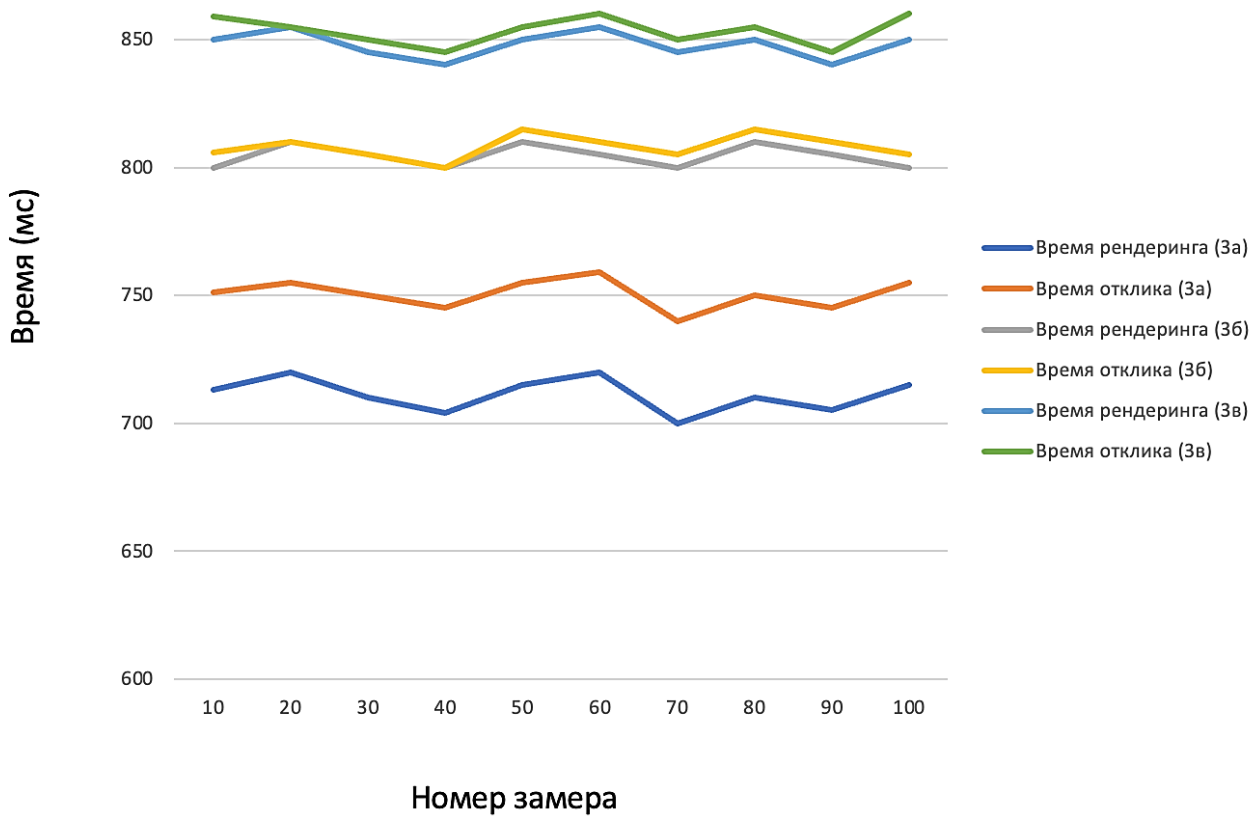


Рисунок 4.5 – Распределение временных показателей веб-страниц относительно общего размера страницы

4.4 Количество *JavaScript* и *CSS* элементов

Интеграция таких элементов как *JavaScript* и *CSS* в веб-страницу может привести к увеличению времени отклика. Дополнительные элементы увеличивают объем передаваемых данных, что влияет на время загрузки [114]. Интегрированные элементы, такие как скрипты, выполняют определенные вычислительные задачи, которые требуют времени на обработку, тем самым увеличивая общее время отклика, воспринимаемое пользователями [2].

Для того, чтобы проиллюстрировать влияние данного фактора страница 1а была преобразована в две измененные версии, обозначенные как страницы 1д и 1е, где каждая содержит *JavaScript* или *CSS*, где страница 1д содержит *JavaScript*-код, предназначенный для отображения текущей даты и времени в начале страницы, а страница 1е включает *CSS*, который задает параметры оформления страницы, такие как цвет фона и шрифта. Таблица 4.7 отражает детальную информацию об этих страницах.

Таблица 4.7 – Детальная информация о страницах 1д и 1е

Параметр / Название страницы	Страница 1д	Страница 1е
Строки кода	112	113
Программные конструкции	0	0
Внутренние состояния	0	0
Количество дополнительных объектов	0	0
Размер (КБ)	2	2
Количество элементов (JS/CSS)	1	1

Страницы 1д и 1е отличаются от исходной страницы 1а дополнением 8 и 9 строк кода соответственно. Это привело к увеличению общего размера страниц менее чем 1 КБ. Для каждой из страниц было получено 100 значений времени отклика, и выявленные различия по сравнению со страницей 1а обусловлены дополнительными элементами, которые включены в их структуру. Таблица 4.8 представляет средние значения и стандартные отклонения полученных значений.

Таблица 4.8 – Полученные временные показатели веб-страниц для изучения влияния *JavaScript* и *CSS*

Значение / Номер страницы	Страница 1а (μ)	Страница 1а (σ)	Страница 1д (μ)	Страница 1д (σ)	Страница 1е (μ)	Страница 1е (σ)
Время рендеринга (мс)	15,9	0,57	16,1	0,47	16,2	0,62
Время отклика (мс)	31,9	0,30	46,9	0,30	47,0	0,47
Время до первого рендеринга (мс)	15,5	0,42	30,9	0,31	30,8	0,41

Рисунок 4.6 иллюстрирует распределение полученных значений. Анализ показал, что различия во времени рендеринга между исследуемыми страницами незначительны.

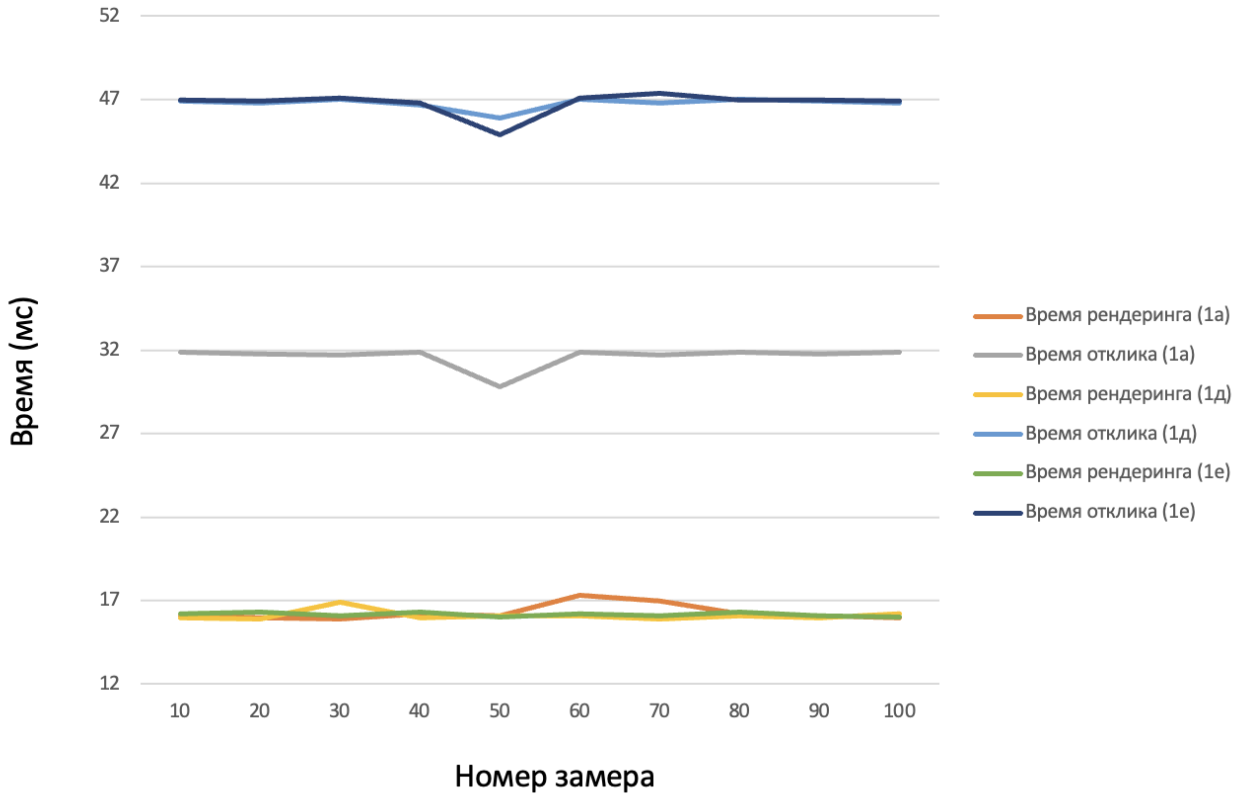


Рисунок 4.6 – Распределение полученных временных показателей веб-страниц влияния *JavaScript* и *CSS*

Однако различия во времени отклика R между страницами 1д и 1е по сравнению со страницей 1а в основном связаны с предварительной обработкой браузером *JavaScript* и *CSS* для выполнения заданных операций или корректной компоновки страниц. Степень интенсивности обработки элемента *CSS/JS* (elm_o). браузером существенно влияет на это предварительное время рендеринга, которое также входит в состав времени отклика. Следовательно оно может быть представлено как функция элемента *CSS/JS*, что выражается как $R = g_{assets}(elm_o)$.

4.5 Количество программных конструкций

В данном разделе особое внимание уделяется такой метрике, как количество программных конструкций. Для анализа влияния этой метрики были разработаны три динамические веб-страницы. Эти страницы имеют три раскрывающихся списка, которые соответствуют трем полям даты: день, месяц и год. Отображение содержимого каждого из указанных полей осуществляется с использованием трех циклов *for*. При этом два из них предназначенны для формирования списков дня и месяца остаются идентичными на всех трех страницах. Различия наблюдаются в реализации цикла *for*, отвечающего за отображение поля года, который меняется для каждой страницы.

Страница 4а имеет 100 элементов в выпадающем списке, которое представляет собой поле года, с использованием одного цикла *for*. С точки зрения нотации *Big-O*, сложность данного цикла оценивается как $O(n)$, где n = 100. В свою очередь страница 4б содержит 100 элементов в выпадающем списке, которое представляет поле «год», с использованием цикла *for*. Сложность данного цикла оценивается как $O(n)$, где n = 100. Однако внутри цикла *for* присутствует вложенный цикл *while*, который выполняет 450 повторений, выполняя операцию сложения на каждой итерации. Таким образом, в отличие от трех программных конструкций, используемых на других страницах, страница 4б включает в себя четыре программные конструкции. Сложность в данном случае $O(n \cdot m)$, где n = 1000, m=450. Страница 4в содержит 1000 элементов в выпадающем списке. Он представляет поле «год», с использованием одного цикла *for*. С точки зрения вычислительной сложности, данный цикл характеризуется сложностью $O(n)$, где n = 1000. Информация о страницах приведена в таблица 4.9.

Таблица 4.9 – Информация о страницах с различной сложностью программных конструкций

Параметр / Название страницы		Страница 4б	Страница 4в
Строки кода	48	55	50
Программные конструкции	3	4	3
Внутренние состояния	0	0	0
Количество дополнительных объектов	0	0	0
Размер (КБ)	6	6	20
Количество элементов (JS/CSS)	0	0	0

Для каждой страницы были получены 100 значений времени отклика. В таблице 4.10 приведены средние значения (μ) и стандартные отклонения (σ), полученные на основе результатов этих данных.

Таблица 4.10 – Значения страниц с различной сложностью программных конструкций

Значение / Номер страницы	Страница 4а (μ)	Страница 4а (σ)	Страница 4б (μ)	Страница 4б (σ)	Страница 4в (μ)	Страница 4в (σ)
Время рендеринга (мс)	71,1	7	71,1	7	85	8
Время отклика (мс)	374	36	473	30	596	21
Время до первого рендеринга (мс)	302	41	401	30	509	16

Так как страницы 4а и 4б имеют идентичную разметку, то существенной разницы во времени их рендеринга не наблюдается. Однако браузеру требуется больше времени для начала отображения страницы 4б (401 мс) по сравнению со страницей 4а (302 мс) после отправки запроса. Данное различие можно объяснить тем, что для страницы 4б был добавлен дополнительный цикл *for*.

Страница 4в, в отличие от страниц 4а и 4б, имеет значительно больший объем данных, что приводит к увеличению времени рендеринга. Кроме того, размер количество итераций для цикла *for*, используемого на странице 4в, было больше, что существенно увеличило время обработки на сервере по сравнению с двумя другими страницами. Как результат наблюдается более длительная задержка перед началом отображения страницы в браузере. Коэффициенты корреляции между временем отклика и промежутком времени от отправки запроса до начала отображения страницы в браузере демонстрируют высокие значения, составляющие 0,92 для страницы 4а, 0,94 для страницы 4б и 0,91 для страницы 4в.

Эти показатели свидетельствуют о сильной взаимосвязи между временем обработки на сервере и полученных временем отклика. Рисунок 4.7 демонстрирует распределение результатов. Так как время рендеринга для всех трех страниц незначительно отличается, то в данном случае метрика времени между моментом начала запроса и моментом начала отображения страницы является более значимой и информативной метрикой для анализа. Именно она представлена для сравнения и иллюстрации эффектов различных программных конструкций, поскольку она оказывает более заметное влияние на производительность.

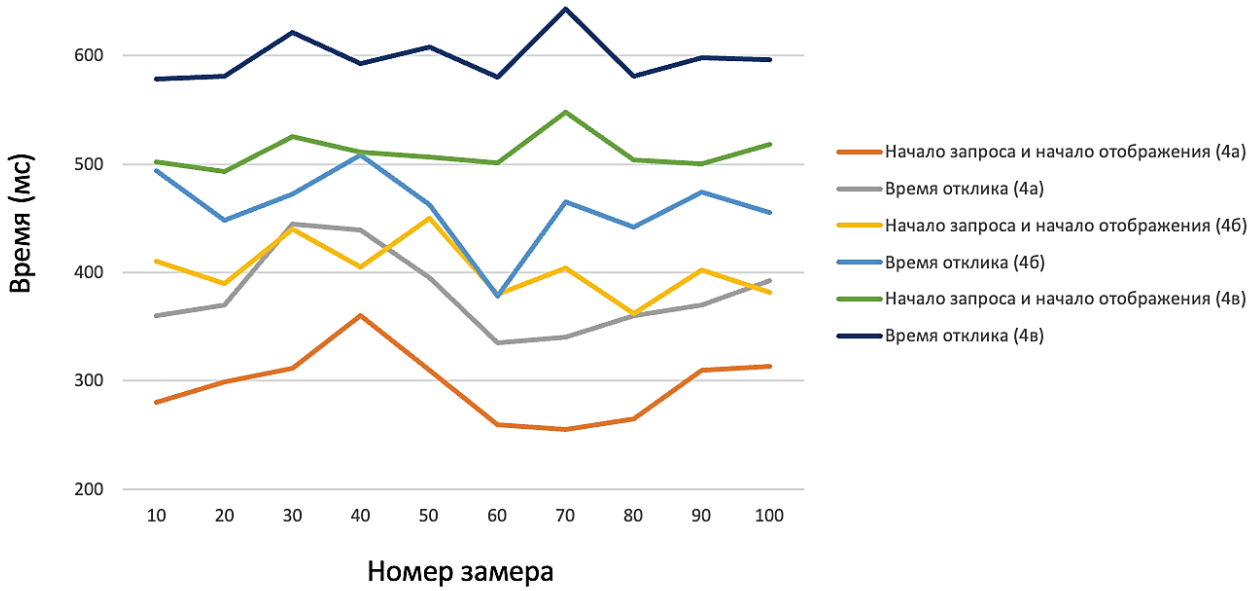


Рисунок 4.7 – Влияние программных конструкций на метрики производительности

Программные конструкции с *linecode* строк кода и сложность *Big-O*, которая обозначается как *O(f(n))*. Сложность отражает влияние время отклика. Такое отношение можно выразить как $w_2 \cdot f(n) \cdot linecode$, где w_2 является весом, который присвоен строке кода. Таблица 4.11 содержит информацию о *f(n)*, *n* и *linecode*.

Таблица 4.11 – Значения страниц с различной сложностью программных конструкций

Страница	Цикл	$f(n)$	n	$linecode$	$f(n) \cdot linecode$
Страница 4а	Цикл 1	n	31	3	92
	Цикл 2	n	12	3	35
	Цикл 3	n	100	4	400
Страница 4б	Цикл 1	n	31	3	93
	Цикл 2	n	12	3	36
	Цикл 3	n	100	5	500
	Цикл 4	n·m	100, 450	2	100000
Страница 4в	Цикл 1	n	31	3	93
	Цикл 2	n	12	3	36
	Цикл 3	n	450	4	2000

На основе данных из таблицы 4.12 можно сделать заключение о том, что n для цикла 3 оказывает значительное влияние на время отклика. Кроме того, цикл 4 имеет меньшее влияние на время выполнения по сравнению с циклом 3, поскольку его работа ограничивается выполнением простых арифметических и логических операций. Это подтверждает, что время отклика R веб-страниц напрямую зависит от сложности используемых программных конструкций и его можно описать как функцию от сложности программных конструкций.

4.6 Состояние базы данных

Влияние базы данных анализируется через использование трех веб-страниц: 5а, 5б, 5в, которые генерируются на сервере. Для минимизации времени на влияние передачи по сети на время отклика контент страниц ограничен исключительно текстовым содержимым.

Для генерации страницы 5а устанавливается соединения с базой данных и выполняется запрос к таблице. Сформированная страница имеет общий размер 5 КБ и содержит раскрывающийся список, включающий 45 элементов.

Для того чтобы сгенерировать страницу 5б устанавливается соединение с базой данных и выполняются два запроса к двум таблицам. Полученная страница имеет общий размер 6 КБ и содержит два раскрывающихся списка, которые охватывают по 45 и 25 элементов соответственно. Различие во времени отклика между страницами 5б и 5а можно объяснить дополнительным запросом к базе данных, увеличенным количеством строк кода и дополнительными программными конструкциями, которые необходимы для обработки возвращаемых запросом данных, а также увеличенным объемом конечной страницы, которая отображается в браузере.

При генерации страницы 5в устанавливаются 2 подключения к базе данных и выполняются два запроса к двум таблицам. Сформированная страница имеет общий размер 8 КБ и содержит два раскрывающихся списка, включающих 45 и 25 элементов соответственно. Запросы и таблицы, которые используются на странице 5в, идентичны запросам и таблицам, применяемым

на странице 5б, а конечный контент обеих страниц полностью совпадает. Таким образом, различие во времени отклика между страницами 5в и 5б обусловлено дополнительным подключением к базе данных, выполненным в процессе генерации страницы 5в. Таблица 4.12 показывает подробные характеристики страниц.

Таблица 4.12 – Информация о страницах с разными состояниями БД

Параметр / Название страницы	Страница 5а	Страница 5б	Страница 5в
Строки кода	67	84	94
Программные конструкции	2	4	4
Внутренние состояния (БД)	1 соединение, 1 запрос	1 соединение, 2 запроса	2 соединения, 2 запроса
Количество дополнительных объектов	0	0	0
Размер (КБ)	5	6	6
Количество элементов (JS/CSS)	0	0	0

Для каждой из страниц было проведено 100 экспериментов для получения времени отклика. Различия в полученных значениях времени отклика преимущественно обусловлены различиями в количестве установленных подключений к базе данных и выполненных запросов, необходимых для генерации страниц. В таблице 4.13 приведены средние значения (μ) и стандартные отклонения (σ), полученные на основе результатов сбора данных.

Таблица 4.13 – Сравнение эффекта доступа к базе данных

Значение / Номер страницы	Страница 5а (μ)	Страница 5а (σ)	Страница 5б (μ)	Страница 5б (σ)	Страница 5в (μ)	Страница 5в (σ)
Время рендеринга (мс)	43	6	69	8	69	7
Время отклика (мс)	2730	40	2872	61	3091	59
Время до первого рендеринга (мс)	2685	41	2801	62	3021	54

Страницы 5б и 5в показывают большее время отклика по сравнению со страницей 5а. Это объясняется их большим размером. При этом две идентичные страницы показывают одинаковые значения времени рендеринга, что подтверждает их идентичность в контексте этой метрики. Разница во времени отклика между страницами 5а и 5б составляет 142 мс, а между страницами 5б и 5в – 219 мс. Полученные данные свидетельствуют о том, что процедура установления соединения с базой данных занимает больше времени, чем выполнение запроса.

Это подтверждается тем, что процесс установления соединения с базой данных является вычислительно затратным и требует много времени. Кроме того, значения имеют высокую степень корреляции между временем отклика и интервалом времени от отправки запроса до начала отображения страницы в браузере. Коэффициенты корреляции для всех трех страниц составляют 0,991, что указывает на тесную взаимосвязь этих параметров. Рисунок 4.8 наглядно представляет распределение полученных значений.

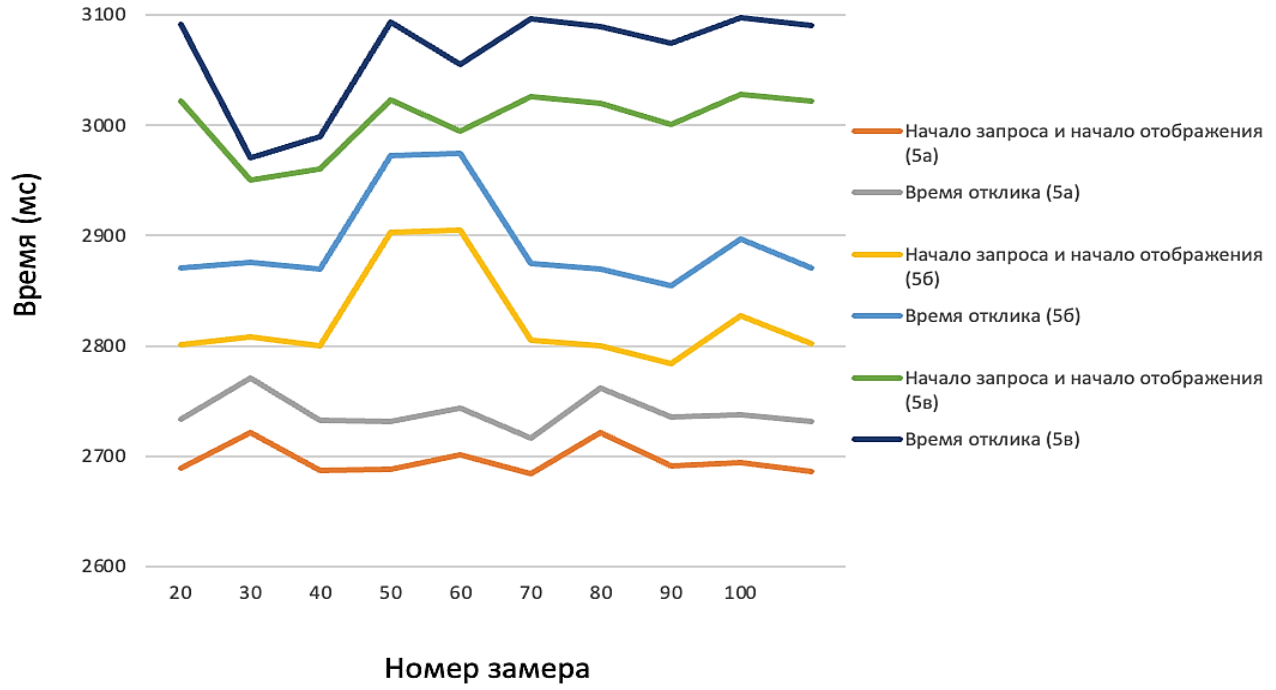


Рисунок 4.8 – Влияние внутреннего состояние БД на время отображения и отклика

Как было описано ранее метрики и количество соединений к базе данных k_{soed} и размер базы данных z_{sp} важны при генерации веб-страницы на сервере. Время отклика зависит от параметров взаимодействия с базой данных. Сложность генерации веб-страницы в данном случае можно описать как $w_3 t + \sum_{i+1}^{j}\left(n_i \cdot w_{sz_{db,i}}\right) + vl$, где w_3 вес для времени установки соединения к базе данных. k_{soed} обозначает количество установленных соединений к j базам данных, n количество соединений и $w_{sz_{db,i}}$ отражает время обработки запроса/обновления и vl представляет собой показатель влияния переменных сессии и *cookie* на время отклика.

4.7 Состояние данных *cookie*

Влияние использования небольших фрагментов данных *cookie* на время отклика веб-страницы анализируется на примере четырех страниц. Первая страница 6а не имеет связанных данных *cookie* и представляет случай без участия механизмов сохранения состояния. Страница 6б имеет *cookie* в виде 17

переменных сессии, которые создаются сервером при выполнении запроса страницы и передаются в браузер клиента. Страница 6в извлекает 20 переменных сессии из данных *cookie*, связанных со страницей 6б, выполняет их модификацию и отправляет обновленный файл обратно клиенту. Страница 6г аналогична странице 6б, однако файл *cookie* в данном случае содержит 45 переменных сессии вместо 17. Таким образом, страницы 6б и 6г моделируют сценарий первоначального создания *cookie* на сервере и их передачи клиентскому браузеру, тогда как страница 6в демонстрирует процесс двустороннего обмена данными *cookie* между клиентом и сервером с выполнением операций над их содержимым. Сводная информация о характеристиках указанных страниц представлена в таблице 4.14.

Таблица 4.14 – Информация о страницах с разными состояниями *cookie*

Параметр / Название страницы	Страница 6а	Страница 6б	Страница 6в	Страница 6г
Строки кода	23	30	35	35
Программные конструкции	0	1	1	1
Состояние *cookie* (переменные)	0	17	20	45
Количество дополнительных объектов	0	0	0	0
Размер (КБ)	1	1	1	1
Количество элементов (JS/CSS)	0	0	0	0

Для каждой страницы было получено 100 значений времени отклика. Таблица 4.15 средние значения (μ) и стандартные отклонения (σ) для полученных значений, а рисунок 4.9 показывает распределение значений.

Таблица 4.15 – Влияние *cookie*

Значение / Номер страницы	Страница 6а (μ)	Страница 6а (σ)	Страница 6б (μ)	Страница 6б (σ)	Страница 6в (μ)	Страница 6в (σ)	Страница 6г (μ)	Страница 6г (σ)
Время рендеринга (мс)	17	0,54	17	0,42	17	0,54	17	0,53
Время отклика (мс)	32	0,33	32	0,53	47	0,43	32	0,52
Время до первого рендеринга (мс)	17	0,51	17	0,48	34	0,47	17	0,33

Количество строк кода для всех четырех страниц находятся на схожем уровне, что сводит к минимуму влияние их количества на различия во времени отклика страниц. Программная конструкция, которая используется на странице 6б, вносит дополнительную задержку ко времени отклика между страницами 6а и 6б. Программные конструкции на страницах 6б и 6в реализованы как циклы *for* с вычислительной сложностью *O(n)*, где максимальное значение *n* составляет 50. Основным фактором, который реально повлиял на результат, послужил увеличению времени отклика для страницы 6в, является дополнительным шагом, связанным с передачей *cookie*-данных от клиентского браузера на сервер, выполнением операций над переменными, содержащимися в *cookie*, и дальнейшей передачей обновленного *cookie*-данных назад в клиентский браузер [115].

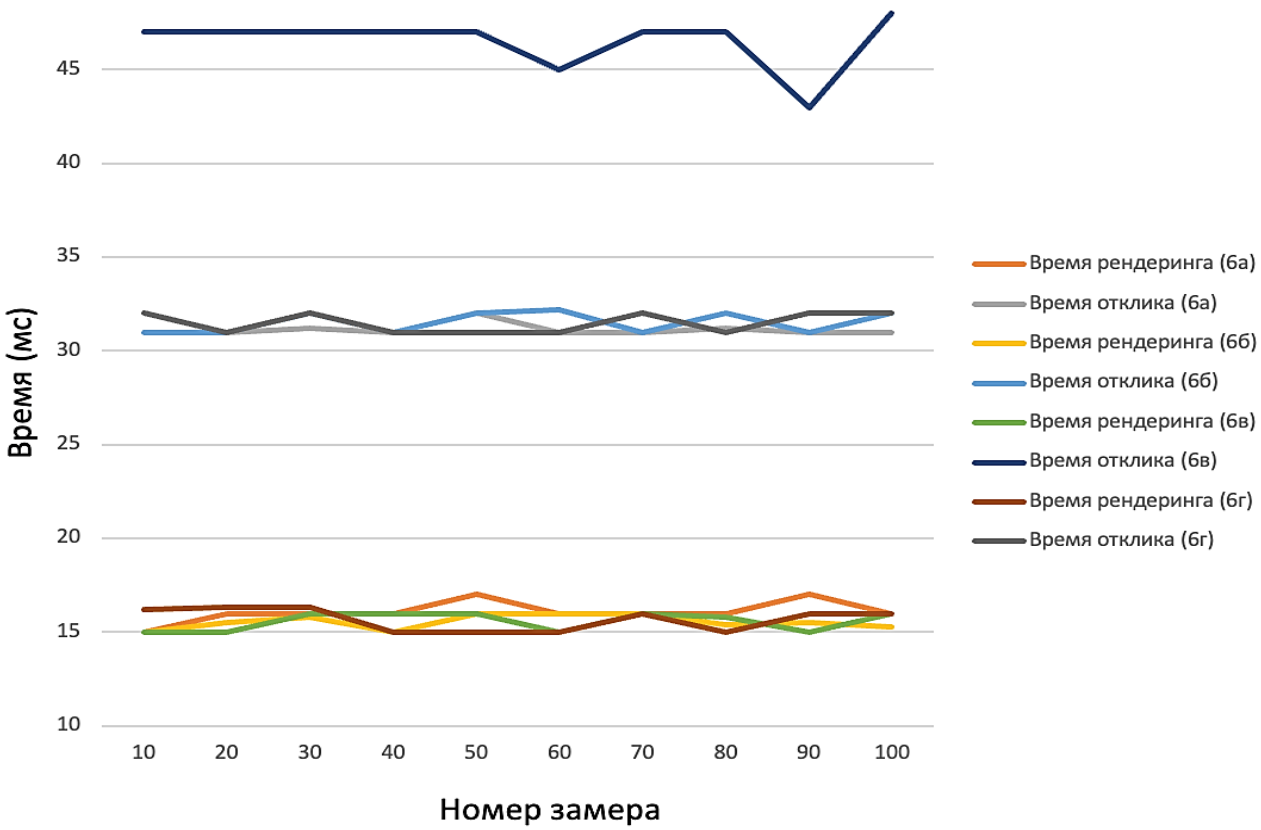


Рисунок 4.9 – Распределение полученных временных характеристик для изучения влияния файлов *cookie*

Все переменные сессии, которые хранятся в файле *cookie*, не оказывают значительного влияния на время отклика веб-страницы, что подтверждается значениями времени отклика для страниц 6б и 6г. Это связано с тем, что файл *cookie* представляет собой компактный текстовый фрагмент, и даже при увеличении числа переменных сессии ее общий размер остается небольшим. По этой причине влияние на производительность, которое связано с передачей *cookie*-данных, минимальны. Однако следует отметить, что основной целью использования *cookie* является *HTTP* сессия между клиентом и сервером. Переменные сессии, содержащиеся в *cookie*, обычно подвергаются операциям для поддержания пользовательской сессии.

Веб-страницы, имеющие логику работы с *cookie*, увеличивают время их отклика [92]. В конечном итоге можно выделить два момента, через которые файлы *cookie* влияют на время отклика: влияние на производительность при передаче файлов *cookie* между клиентским браузером и сервером и операции,

выполняемые над переменными сессии. Таким образом, общее время отклика веб-страницы, связанной с файлами *cookie*, может быть представлено как функция, зависящая от характеристик файлов *cookie* и операций над ними.

Выводы по главе

Время отклика веб-страницы может быть описано при помощи модели, отражающей ее сложность. Она определяется двумя ключевыми аспектами: процессом генерации (C_1) и характеристиками содержимого (C_2). Каждый из них имеет три аспекта, влияющих на них. Для генерации выделяются количество строк кода, програмные конструкции и состояние (включая доступ к базам данных и обработку файлов *cookie*). Метрика «строки кода» может быть применена как к статическим, так и к динамическим веб-страницам, а метрика «программные конструкции» и «состояние» относятся исключительно к динамическим страницам. Содержимое (C_2) включает метрику количество дополнительных объектов, общий объем страницы (в килобайтах) и использование дополнительных элементов *JavaScript* и *CSS*.

Проведенные эксперименты для сбора данных о времени отклика различных веб-страниц позволили не только количественно оценить влияние отдельных метрик на производительность, но и эмпирически подтвердить работоспособность предложенных моделей. Разработанный метод, основанный на программном комплексе, обеспечил высокую точность полученных значений и возможность декомпозиции времени, отклика на составные компоненты, что принципиально отличает его от существующих подходов.

Разработанные модели предоставляют систематизированный подход к анализу отдельных моментов, влияющих на время отклика. Каждая из представленных моделей может быть использована для разработки специализированных программных инструментов, которые оценивают веб-страницы по соответствующим метрикам, влияющим на время отклика. Такие инструменты способны выявлять недостатки веб-страницы, которые приводят к увеличению времени отклика.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ

В ходе подготовки диссертации получены следующие результаты, имеющие существенное значение для развития области информационных технологий:

1. Был проведен анализ существующих систем и методов анализа производительности клиент-серверных веб-приложений. Выявлены ограничения существующих методов и недочеты схожих технических решений. Полученная в результате обзора и анализа информация послужила созданию требований для разработанного программного комплекса.

2. Разработана математическая модель генерации веб-страницы, которая учитывает влияние таких метрик как количество строк кода, сложность программных конструкций и состояние приложения (количество подключений к базе данных, количество переменных *cookie*) на время отклика.

3. Разработана математическая модель контента веб-страницы, описывающая влияние таких метрик контента как общий размер *HTML*-документа, количество дополнительных объектов, размер и количество *JavaScript* и *CSS*-ресурсов на время отклика. Модель позволяет определять влияние факторов, связанных с контентом веб-страницы, на время отклика.

4. Разработан метод сбора данных о производительности клиент-серверных веб-приложений, основанный на программном комплексе и использующий микросервисную архитектуру для автоматизированного сбора, анализа и визуализации данных, результатом внедрения которого стало улучшение качества оценки производительности. Использование программного комплекса обеспечивает снижение времени отклика веб-страниц в среднем на 30% по сравнению с традиционным методом, основанным на прокси-сервере.

5. Предложена методика анализа производительности веб-страниц клиент-серверных приложений, основанная на интеграции метода сбора данных производительности на основе программного комплекса, математических моделей генерации и контента веб-страниц с использованием алгоритма классификации и выявления закономерностей, что позволяет установить

взаимосвязь между характеристиками веб-страниц и их временем отклика, а также обеспечить верифицируемую оценку производительности.

6. Проведено экспериментальное исследование, подтвердившее применимость предложенных методов, моделей и методики для анализа факторов, влияющих на производительность клиент-серверных приложений, и обеспечивающих поддержку на этапе их разработки и в процессе эксплуатации.

7. Разработан программный комплекс для сбора и анализа данных о производительности клиент-серверных приложений, применение которого в АО «Радиозавод» позволило снизить время на диагностику проблем на 13,7%.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ И ПЕРВОИСТОЧНИКОВ

1. Killelea P. Web Performance Tuning: speeding up the web. – Sebastopol, CA: O'Reilly Media, Inc., 2002. – 432 p.

2. Souders S. High Performance Web Sites: Essential Knowledge for Frontend Engineers. – Farnham: O'Reilly, 2007. – 168 p.

3. Рахматулина Р. Ш. Объекты цифрового дизайна: особенности правового регулирования // Юридический вестник Самарского университета. – 2020. – Т. 6, № 4. – С. 33–37. DOI: https://doi.org/10.18287/2542-047X-2020-6-4-33-37.

4. Grigorik I. High Performance Browser Networking. – Sebastopol, CA: O'Reilly Media, 2013. – 400 с.

5. Digital 2024: основные выводы из ежегодного отчета DataReportal. – 2024 – Текст: электронный // BYYD: [сайт]. – URL: https://www.byyd.me/ru/blog/2024/02/digital-2024-datareportal/ (дата обращения: 10.05.2024).

6. Everts T. Time Is Money. – Sebastopol, CA: O'Reilly Media, 2016. – 56 с.

7. Яблонски Дж. Законы UX-дизайна: понимание психологии пользователя – ключ к успеху / пер. с англ. А. Логунова. – СПб.: БХВ-Петербург, 2022. – 160 с.

8. Кондратьев А. С., Ляпцев А. В. Математическое моделирование: аналитические и вычислительные методы // Компьютерные инструменты в образовании. – 2007. – № 5. – С. 20–24.

9. Таненбаум Э. С., Фимстер Н., Уэзеролл Д. Дж. Компьютерные сети / пер. с англ. С. Черников; ред. М. Капранов. – 6-е изд. – Санкт-Петербург: Питер, 2024. – 992 с.

10. Куроуз Дж. Ф., Росс К. В. Компьютерные сети. Нисходящий подход / пер. с англ. М. Райтман. – М.: Эксмо, 2016. – 912 с.

11. Протокол передачи гипертекста (HTTP) – Текст: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/HTTP (дата обращения: 8.02.2024).

12. Протокол управления передачей (TCP) – Текст: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Glossary/TCP (дата обращения: 8.02.2024).

13. Интернет-протокол (IP) – Текст: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Glossary/IP (дата обращения: 8.02.2024).

14. RU2669172C2 Способ и система мониторинга согласованности веб-сайта – Текст: электронный // Яндекс патенты [сайт]. – URL: https://yandex.ru/patents/doc/RU2669172C2_20181008 (дата обращения: 21.05.2024).

15. RU2763000C2 Способ и устройство для создания модели качества веб-страницы – Текст: электронный // Национальная электронная библиотека [сайт]. – URL: https://rusneb.ru/catalog/000224_000128_0002680746_20190226_C2_RU (дата обращения: 23.05.2024).

16. RU2669172C2 Мониторинг задержки тега и система управления для повышения производительности веб-страницы – Текст: электронный // Яндекс патенты [сайт]. – URL: https://yandex.ru/patents/doc/RU2763000C2_20211224 (дата обращения: 25.05.2024).

17. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман. – СПб.: Питер, 2020. – 640 с.

18. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – СПб.: Питер, 2022. – 352 с.

19. Мартин Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – СПб.: Питер, 2019. – 464 с.

20. Fu Y., Yan M., Xu Z., Xia X., Zhang X., Yang D. An empirical study of the impact of log parsers on the performance of log-based anomaly detection // Empirical Software Engineering. 2023. Vol. 28(1). № 6. DOI: 10.1007/s10664- 022-10214-6 EDN: AAKUBP

21. Goel N., Jha C.K. (2013) Analyzing users behavior from web access logs using automated log analyzer tool. International Journal of Computer Applications, vol. 62, no. 2, pp. 29–33.

22. Доронов В. JavaScript. Народные советы [Текст] / Доронов В. — 1-е изд. – СПб.: БХВ, 2007 — 448 с.

23. Иванов, А. С. Разработка веб-приложений с использованием HTML, CSS и JavaScript [Текст] / А. С. Иванов — М.: Издательство "Техносфера", 2018 – 1120 с.

24. Online Retail Performance: Milliseconds Are Critical // APM digest. URL: https://www.apmdigest.com/state-ofonline-retail-performance (дата обращения: 10.02.2025).

25. Page Weight | Part IV Chapter 21 // Web Almanac by HTTP Archive. URL: https://almanac.httparchive.org/en/2022/page-weight (дата обращения: 15.02.2025).

26. Web Vitals 2024 // Web.dev. URL: https://web.dev/articles/vitals (дата обращения: 7.02.2025).

27. Думченков И. А. Обзор методов интеграции информационных систем, их преимуществ и недостатков // Молодой ученый. 2018. № 23(209). С. 176–177. EDN: URLVVO.

28. Google is making mobile ads much faster – Текст: электронный // Axious [сайт]. – URL: https://www.axios.com/2017/12/15/google-is-making-mobile-ads-much-faster-1513302514 (дата обращения: 26.05.2024).

29. AliExpress reduced load time by 36% and saw a 10.5% increase in orders – Текст: электронный // WPO Stats [сайт]. – URL: https://wpostats.com/2016/12/27/aliexpress-load-time/ (дата обращения: 26.05.2024).

30. L. Titchkosky, M. Arlitt, C. L. Williamson., A performance comparison of dynamic Web technologies, SIGMETRICS Performance Evaluation Review, 31(3), pages 2-11, December 1, 2003. DOI: 10.1145/974036.974037

31. Trident – Текст: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/en-US/docs/Glossary/Trident (дата обращения: 11.06.2024).

32. Microsoft Edge – Текст: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/en-US/docs/Glossary/Microsoft_Edge (дата обращения: 11.06.2024).

33. How Chromium Displays Web Pages: электронный // Chromium: [сайт]. – URL: https://www.chromium.org/developers/design-documents/displaying-a-web-page-in-chrome/ (дата обращения: 11.06.2024).

34. Время загрузки страницы и ресурсов // MDN Web Docs: [сайт]. – URL:https://developer.mozilla.org/ru/docs/Web/Performance/Guides/Navigation_and_resource_timings (дата обращения: 17.06.2024).

35. Documentation // V8: [сайт]. – URL: https://v8.dev/docs (дата обращения: 18.06.2024).

36. SpiderMonkey// Firefox Source docs: [сайт]. – URL: https://firefox-source-docs.mozilla.org/js/index.html (дата обращения: 18.06.2024).

37. Browser Market Share Worldwide // Starcounter: [сайт]. – URL: https://gs.statcounter.com/browser-market-share (дата обращения: 18.05.2021).

38. Souders S. Even Faster Web Sites: Performance Best Practices for Web Developers. 1st ed. O'Reilly Media, 2009.

39. Web Workers API: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/API/Web_Workers_API (дата обращения: 11.06.2024).

40. Критические этапы рендеринга // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/Performance/Guides/Critical_rendering_path (дата обращения: 5.07.2024).

41. Мартышкин А.И., Максимов Я.А. Проблемы измерения производительности в современных клиент-серверных Веб-приложениях // Современные информационные технологии. – 2023. – № 37 (37). – С. 105-109.

42. Горлач, Б.А. Математическое моделирование. Построение моделей и численная реализация // СПб.: Лань, 2016. – 292 с.

43. Harrison, P. G., and Patel, N. M. (1993). Performance Modelling of Communication Networks and Computer Architectures. Wokingham: Addison-Wesley Publishing Company.

44. Пышкина И.С., Максимов Я.А., Мартышкин А.И. Метод расчета производительности Веб-приложения // XXI век: итоги прошлого и проблемы настоящего плюс. – 2024. – Т. 13. – № 2 (66). – С. 43-48.

45. Extended Log File Format // W3C: [сайт]. – URL: https://www.w3.org/TR/WD-logfile.html (дата обращения: 15.07.2024).

46. Blocking: The Web Performance Villain // W3C: [сайт]. – URL: https://bluetriangle.com/blog/blocking-web-performance-villain#:~:text=Chrome%20has%20a%20limit%20%20of,host%20at%20the%20same%20time (дата обращения: 15.07.2024).

47. Hofmann, R., Klar, R., Mohr, B., Quick, A., and Siegle, M. (1994). Distributed Peformance Monitoring: Methods, Tools and Applications. IEEE Transactions on Parallel and Distributed Systems, 5(6): 585-598.

48. Максимов Я.А., Мартышкин А.И. Возможности оптимизации производительности веб-приложений // Современные информационные технологии. – 2024. – № 39 (39). – С. 189-192.

49. Малахов В.В. Большие данные и аналитика в образовании. Сборник трудов конференции. СПб.: ООО "ВВМ", 2020: 282-287.

50. Линдигрин А.Н. Анализ специфики и проблематики процессов поиска аномалий в сетевых данных / А.Н. Линдигрин // Известия ТулГУ. Технические науки. – 2021. – № 5. – С. 304-309.

51. Максимов Я.А., Гудкова Е.А. Развертывание современных веб-приложений // Современные информационные технологии. – 2020. – № 32 (32). – С. 69-71.

52. T. M. L. Ivanova E. V., "Overview of modern time series processing systems," Bulletin of the South Ural State University. Series: Computational Mathematics and Informatics, vol. 9, no. 4, pp. 79-97, 2020.

53. Tokar D., "The IoT Applications Productivity: Data Management Model and ELK Tool Based Monitoring and Research," 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), pp. 162-167, 2022.

54. Клеппман М., Высоконагруженные приложения. Программирование, масштабирование, поддержка. // СПб.: Питер, 2020. – С. 640.

55. Nagappan M., Vouk MA (2010) Abstracting log lines to log event types for mining software system logs. In: 2010 7th IEEE working conference on mining software repositories (MSR 2010). IEEE, pp 114– 117

56. Рудометкин В.А. Мониторинг и поиск неисправностей в распределенных высоконагруженных системах. – Кибернетика и программирование, 2020. – № 2. – С. 1-6. – DOI 10.25136/2644-5522.2020.2.32996.

57. Сильнов Д.С. Актуальность современных систем удаленного мониторинга вычислительных ресурсов / Сильнов Д.С. - Санкт-Петербург, известия РГПУ им. а.и. Герцена, 2020, 55-59.

58. The Common Logfile Format // W3C: [сайт]. – URL: https://www.w3.org/Daemon/User/Config/Logging#common-logfile-format (дата обращения: 22.08.2024).

59. Максимов Я.А., Мартышкин А.И. Обзор современных программных решений в области измерения производительности клиентской части веб-приложений // Современные наукоемкие технологии. – 2021. – № 12-2. – С. 348-354.

60. URI | MDN // MDN: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/URI (дата обращения: 29.10.2024).

61. Gourley D., Totty B. HTTP: The Definitive Guide. O'REILLY Media Inc., 2002. 617 p.

62. Рудометкин В.А. Мониторинг и поиск неисправностей в распределенных высоконагруженных системах. – Кибернетика и программирование, 2020. – № 2. – С. 1-6. – DOI 10.25136/2644-5522.2020.2.32996.

63. How to Make Your Website Load Faster With WebP Images // TechStacker: [сайт]. – URL: https://techstacker.com/load-website-faster-with-webp-images/MHzDWJFko9EB4fBep/ (дата обращения: 29.10.2024).

64. Максимов Я.А., Мартышкин А.И. Влияние интернет-рекламы на производительность клиентской части веб-приложений // Современные наукоемкие технологии. – 2022. – № 11. – С. 52-56.

65. ECMAScript 2015 Language Specification // ECMA [сайт]. – URL: https://262.ecma-international.org/6.0/ (дата обращения 4.11.2024).

66. Деарт В.Ю., Кожухов И.С. Влияние современных способов оптимизации загрузки интернет-страниц в браузерах на параметры качества обслуживания http-рафика // Технологии информационного обществ. – 2012. – № 7. – С. 83-84.

67. Модели и методы разработки крупномасштабных веб-приложений // UGATU [Электронный ресурс]. – URL: http://journal.ugatu.su/index.php/Vestnik/article/view/2468 (дата обращения: 28.01.2023).

68. Vidgen R. Developing Web Information Systems: From Strategy to Implementation. Oxford, Butterworth-Heinemann, 2002. 274 p.

69. Рахмани Д., Баранов М.Д., Кузьмин Д.А. Разработка метода оптимизации веб-приложений с целью повышения производительности // Технические науки. – 2025. – № 3 (1). – С. 237–241. – DOI: 10.24412/2500-1000-2025-3-1-237-241.

70. Рендеринг в веб // Habr [Электронный ресурс]. – URL: https://habr.com/ru/articles/548382 (дата обращения: 5.12.2024).

71. Кравцов Е. П. Эффективное отображение изображений в браузере: техники и стратегии / Кравцов Е.П. - Проблемы науки, 2024, 55-59.

72. Vepsäläinen J., Vuorimaa P. Bridging Static Site Generation with the Dynamic Web // Web Engineering. ICWE 2022. Lecture Notes in Computer Science. – 2022. – Vol. 13362. – P. 437–442. – DOI: 10.1007/978-3-031-09917-5_32

73. Using Static Site Generators for Scholarly Publications and Open Educational Resources // Code4Lib Journal – URL: https://journal.code4lib.org/articles/13861 (дата обращения: 7.12.2024).

74. P. Kishore, Mahendra B.M. Evolution of Client-Side Rendering over Server-Side Rendering, recent Trends in Information Technology and its Application, vol.3, no.2 (2020).

75. Taufan Fadhilah Iskandar и другие, Comparison between client-side and server-side rendering in the web development, IOP Conference Series: Materials Science and Engineering, vol. 801 (2019).

76. Разворачиваем приложение Next.js с базой данных PostgreSQL и задачей Cron на облачном сервере Ubuntu Linux // Habr [Электронный ресурс]. – URL: https://habr.com/ru/companies/timeweb/articles/858094 (дата обращения: 8.12.2024).

77. Бурукина И. П., Привалов А. Э. Исследование современных подходов к проектированию цифровых интерфейсов // Известия высших учебных заведений. Поволжский регион. Технические науки. 2022. № 1. С. 78–87. doi:10.21685/ 2072-3059-2022-1-7.

78. Гмурман В. Е. Теория вероятностей и математическая статистика 12-е изд. Учебник для вузов / В. Е. Гмурман. – М.: Изд-во Юрайт, 2020. – 480 с.

79. Кибзун А. И. Теория вероятностей и математическая статистика. Базовый курс с примерами и задачами. Учебник для вузов / А.И.Кибзун, Е.Р.Горяинова, А.В.Наумов, А.Н.Сиротин. – М.: Изд-во Юрайт, 2020. – 224 с.

80. Чулков, В. А. Методология научных исследований: учебное пособие / В. А. Чулков. — Пенза: ПензГТУ, 2014. – 200 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/62796 (дата обращения: 10.01.2025). – Режим доступа: для авториз. пользователей.

81. Website Response Times // Nielsen Norman group [Электронный ресурс]. – URL: https://www.nngroup.com/articles/website-response-times/ (дата обращения: 14.12.2024).

82. The Need for Speed // Nielsen Norman group [Электронный ресурс]. – URL: https://www.nngroup.com/articles/the-need-for-speed-1997 // (дата обращения: 14.12.2024).

83. Григорьев С.В. Статический анализ программного кода / С. В. Григорьев – СПб.: БХВ-Петербург, 2008, 416.

84. Nielson F., Nielson H., Hankin C. Principles of Program Analysis. – Berlin: Springer, 2005. – 852 p.

85. McCabe T.J. A Complexity Measure. IEEE Transactions on Software Engineering. 1976, SE-2(4):308–320.

86. Маккейб Т. Измерение сложности программ // Программирование. – 1976. – № 3. – С. 28–36.

87. Фленаган Д. JavaScript. Подробное руководство. – 7-е изд. – М.: Вильямс, 2021. – 1072 с.

88. Firdous Ahmad Mala. The Big-O of Mathematics and Computer Science // Journal of Applied Mathematics and Computation. – 2020. – № 3. – С. 1–3. DOI: 10.26855/jamc.2022.03.001.

89. Кнут Д. Искусство программирования: Основные алгоритмы. – М.: Вильямс, 2019. – 800 с.

90. Фаулер М. Рефакторинг: улучшение дизайна существующего кода. – СПб.: Питер, 2021. – 448 с.

91. Grune D., van Reeuwijk K., Bal H. E., Jacobs C. J. H., Langendoen K. Modern compiler design. Springer Science & Business Media, 2012.

92. HTTP-куки – Текст: электронный // MDN Web Docs: [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Guides/Cookies (дата обращения: 23.12.2024).

93. Глушкова О.О. Выявление с помощью нагрузочного тестирования и устранение узких мест в работе с базами данных // Вестник науки. – 2025. – Т. 3, № 4 (85). – С. 737-742.

94. Юлкина Н.С., Шмелев М.С., Фомина И.К. Сравнение и анализ методов и инструментов, применяемых для сбора данных // Молодой исследователь Дона. – 2023. – № 8 (6). – С. 49-54.

95. Дейт К. Дж. Введение в системы баз данных. – Классическое издание – М.: Диалектика-Вильямс, 2005. – 720 с.

96. Гарсия-Молина Х., Ульманн Дж. Д., Уидом Дж. Системы баз данных: полный курс. – М.: Вильямс, 2003. – 1248 с.

97. Каримов Т.Ш. Оптимизация запросов в реляционных базах данных: подходы и инструменты // Вестник науки. – 2025. – Т. 3, № 4 (85). – С. 780-786.

98. Berners Lee Tim. Weaving the Web: Th e past, Presentand Futureof the World Wide Web by its Inventor. Texere. London, 2000.

99. Таненбаум Э. С., Фимстер Н., Уэзеролл Д. Компьютерные сети. 6-е изд. М.: Питер, 2023. – 992 с.

100. Next.js by Vercel – Текст: электронный // Nextjs: [сайт]. – URL: https://nextjs.org/ (дата обращения: 14.1.2025).

101. npm | Home – Текст: электронный // npm: [сайт]. – URL: https://www.npmjs.com (дата обращения: 14.1.2025).

102. Spring Boot – Текст: электронный // Spring IO: [сайт]. – URL: https://spring.io/projects/spring-boot (дата обращения: 16.1.2025).

103. Kotlin Docs – Текст: электронный // Kotlin: [сайт]. – URL: https://kotlinlang.org/docs/home.html (дата обращения: 23.1.2025).

104. Жемеров Д.Б. Kotlin в действии / Д.Б. Жемеров, С.С. Исакова — М.: МДК-Пресс, 2021. — 402 с.

105. Лонг, Джош. Java в облаке. Spring Boot, Spring Cloud, Cloud Foundry / Джош Лонг, Кеннет Бастани. – Санкт-Петербург: Питер, 2019. – 624 с.

106. Maximov Y.A., Martyshkin A.I. Fault Prediction In Server-Side Of Web-Applications By Using A Software System and Long Short-Term Memory Neural Networks // Empirical Software Engineering. 2025. Vol. 28(1). № 6. DOI: 10.1007/s10664- 022-10214-6 EDN: AAKUBV

107. C4 model for software architecture – Текст: электронный // C4 model: [сайт]. – URL: https://c4model.info/#abstractions (дата обращения: 7.2.2025).

108. Нагорный Н.Н. Современные технологии программного обеспечения для веб-приложений // Цифровая экономика. – 2023. – №2. – С. 81-84.

109. HOME TCPDUMP – Текст: электронный // TCPDUMP [сайт]. – URL: https://www.tcpdump.org/ (дата обращения: 7.2.2025).

110. Булгаков М.В., Носов В.П. Исследование и разработка методов построения и кэширования веб-приложений // Открытое образование. – 2008. – №2. – С. 28-40.

111. Аверкиев А.А., Камбулов Д.А. Анализ локальной вычислительной сети // StudNet. – 2022. – №1 . – С. 678-887.

112. Производительность фронтенда: разбираем важные метрики – Текст: электронный // Habr [сайт]. – URL: https://habr.com/ru/companies/vk/articles/454920 (дата обращения: 12.2.2025).

113. Geissler, G., Zinkhan, G. M., Watson, R. T. Web Home Page Complexity and Communication Effectiveness. Journal of the Association for Information Systems // Information Systems. – 2001. – №2 . – С. 1-46.

114. Яремчук А.В. Роль современных методов хранения клиентских данных / Яремчук А.В. // Научно-исследовательский центр "Science Discovery". – 2024. – № 17. – С. 80-84.

115. Barry P. Exploring New Frontiers in Web Performance. M.: Manning, 2020. – 160 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1. Свидетельства о государственной регистрации программ для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025664004

Программа для сбора метрик в браузерных приложениях

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Пензенский государственный технологический университет» (RU)***

Авторы: ***Максимов Ярослав Александрович (RU), Мартышкин Алексей Иванович (RU)***

Заявка № **2025662530**

Дата поступления **20 мая 2025 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***02 июня 2025 г.***

Руководитель Федеральной службы по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат 0692e7b1a6300bf54f240f670bcc2026

Владелец **Зубов Юрий Сергеевич**

Действителен с 10.07.2024 по 03.10.2025

Ю.С. Зубов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025665430

Программа для измерения метрик производительности клиент-серверных систем

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Пензенский государственный технологический университет» (RU)***

Авторы: ***Максимов Ярослав Александрович (RU), Мартышкин Алексей Иванович (RU)***

Заявка № **2025662573**
Дата поступления **20 мая 2025 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***17 июня 2025 г.***

Руководитель Федеральной службы по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат: 0692e761a6300bf54f240f670bcca2026
Владелец: Зубов Юрий Сергеевич
Действителен с 10.07.2024 по 03.10.2025

Ю.С. Зубов

ПРИЛОЖЕНИЕ 2. Акты внедрения результатов кандидатской диссертации

«УТВЕРЖДАЮ»
И.о. ректора ФГБОУ ВО «Пензенский государственный технологический университет»
_____ Д.В. Пащенко
«29» августа 2025 г.

АКТ
о внедрении результатов диссертационной работы
Максимова Ярослава Александровича

Комиссия в составе:

председатель комиссии – к.т.н., доцент Сёмочкина И.Ю. – начальник учебно-методического управления ФГБОУ ВО «Пензенский государственный технологический университет»;

члены комиссии:

к.т.н., доцент Жашкова Т.В. – заместитель декана факультета автоматизированных информационных технологий ФГБОУ ВО «Пензенский государственный технологический университет»;

к.т.н., профессор Бершадская Е.Г. – профессор кафедры «Программирование» ФГБОУ ВО «Пензенский государственный технологический университет»;

к.т.н., доцент Печерский С.В. – доцент кафедры «Программирование» ФГБОУ ВО «Пензенский государственный технологический университет»,

составила настоящий акт о том, что результаты диссертационной работы Максимова Я.А. на тему «Методы, модели и метрики сбора данных о производительности клиент-серверных веб-приложений», представленной на соискание ученой степени кандидата технических наук, внедрены в учебный процесс кафедры «Программирование» ФГБОУ ВО «Пензенский государственный технологический университет».

Автором получены новые научные результаты:

1. Разработан метод сбора данных производительности клиент-серверных приложений с использованием программного комплекса, состоящего из отдельных масштабируемых компонентов микросервисной архитектуры.

2. Разработана математическая модель генерации веб-страницы, которая отражает как структурные метрики веб-страниц, количество строк кода, сложность программных конструкций и состояние приложения

(количество подключений к базе данных, количество переменных *cookie*) влияют на время отклика.

3. Разработана математическая модель контента веб-страницы, отражающая то, как метрики контента веб-страницы (общий размер *HTML*-документа, количество дополнительных объектов, размер и количество *JavaScript* и *CSS* ресурсов) позволяют выполнить количественную оценку производительности.

4. Предложена методика, заключающаяся в интеграции пяти взаимосвязанных элементов: метод сбора данных производительности клиент-серверных веб-приложений, математическая модель генерации веб-страницы, математическая модель контента веб-страницы и мониторинг, а также алгоритм, реализующий данную методику.

Указанные результаты внедрены в учебный процесс кафедры «Программирование» по направлениям подготовки:

09.03.01 «Информатика и вычислительная техника» при проведении лекционных и лабораторных работ по дисциплинам: «Основы DevOps и DataOps», «Языки интернет программирования», «Сети и телекоммуникации»;

09.03.04 «Программная инженерия» при проведении лекционных и лабораторных работ по дисциплинам: «Администрирование информационно-коммуникационных систем», «Компьютерные сети», .

09.04.04 «Программная инженерия» при проведении лекционных и лабораторных работ по дисциплинам: «Распределенные системы обработки информации», «Протоколы вычислительных сетей».

Внедрение полученных автором научных результатов позволило повысить качество учебного процесса.

Председатель комиссии _____ И.Ю. Сёмочкина

Члены комиссии _____ Т.В. Жашкова

_____ Е.Г. Бершадская

_____ С.В. Печерский

УТВЕРЖДАЮ
Заместитель генерального
директора – главный конструктор
АО «НПП «Рубин», к.т.н., доцент
_____ Сериков И.В.
«14» апреля 2025 г.

М.П.

А К Т

о внедрении результатов диссертационной работы аспиранта Пензенского государственного технологического университета (ПензГТУ) Максимова Ярослава Александровича

Комиссия в составе:

председатель комиссии – и.о. заместителя генерального директора по ГОЗ АО «НПП «Рубин» - начальник научно-технического центра №1 Мясин Петр Юрьевич;

члены комиссии:

и.о. заместителя начальника научно-технического центра №1 АО «НПП «Рубин» Фролов Николай Александрович;

начальник отделения АО «НПП «Рубин» Зинин Сергей Михайлович;

начальник отдела АО «НПП «Рубин» Шеланков Олег Евгеньевич

свидетельствует о том, что в ходе модификации специального программного обеспечения автоматизированной системы управления материально-технического обеспечения войск (сил) (изделие 83т775) были апробированы и реализованы следующие научные положения кандидатской диссертации Максимова Ярослава Александровича:

1. Модели времени отклика веб-страницы, использующие метрики количество строк кода, количество программных конструкций и состояние базы данных.
2. Метод измерения производительности с использованием программного комплекса.

Разработанный программный комплекс сбора и анализа данных производительности для клиент-серверных веб-приложений позволяет в реальном времени регистрировать и реагировать на значения

пользовательских метрик, таких как полное время загрузки страницы, время до первого отображения контента, время до полной интерактивности и другие.

Внедренная программная библиотека успешно интегрируется в клиентскую часть веб-приложений, не нарушая их архитектуру и не требуя модификации серверного кода.

Председатель комиссии: _____ П.Ю. Мясин

Члены комиссии: _____ Н.А. Фролов

_____ С.М. Зинин

_____ О.Е. Шеланков

«14» апреля 2025 г.

ГОСУДАРСТВЕННАЯ КОРПОРАЦИЯ «РОСТЕХ», АКЦИОНЕРНОЕ ОБЩЕСТВО «РОССИЙСКАЯ ЭЛЕКТРОНИКА»
АКЦИОНЕРНОЕ ОБЩЕСТВО «РАДИОЗАВОД»
ул.Байдукова,1,Пенза,440039,Россия, тел.:(8412) 92-80-96, факс:(8412) 49-60-24
ИНН 5835049799, ОГРН 1045802500336, телетайп 155261 «Волна», e-mail: radio@rf58.ru

АКТ

внедрения результатов диссертационной работы аспиранта Пензенского государственного технологического университета (ПензГТУ) Максимова Ярослава Александровича

Следующие результаты диссертационного исследования Максимова Ярослава Александровича:

1. модели времени отклика веб-страницы, использующие метрики количество строк кода, количество программных конструкций и состояние базы данных;
2. метод измерения производительности с использованием программного комплекса;

были использованы при разработке и поддержки программного обеспечения.

Разработанный программный комплекс сбора и анализа данных производительности для клиент-серверных веб-приложений позволяет в реальном времени регистрировать и реагировать на значения пользовательских метрик, таких как полное время загрузки страницы, время до первого отображения контента, время до полной интерактивности и другие.
Внедренная программная библиотека успешно интегрируется в клиентскую часть веб-приложений, не нарушая их архитектуру и не требуя модификации серверного кода. Комплекс позволил измерять производительность даже в высоконагруженных системах с динамически генерируемыми веб-страницами.
Результатом внедрения стало улучшение качества оценки производительности, снижение времени на диагностику проблем на 13.7%. Система была протестирована на реальных проектах и показала стабильную работоспособность в различных условиях эксплуатации

Заместитель генерального директора по ГП Е.А. Евсеев