МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка модели оценки привлекательности Web интерфейса приложения с технологией vue.js

Студент Т.Б. Холдаров
(И.О. Фамилия) (личная подпись)

Руководитель С.В. Митин
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема бакалаврской работы: «Разработка модели оценки привлекательности Web интерфейса приложения с технологией vue.js»

Актуальность бакалаврской работы заключается в проблеме эффективности пользовательского интерфейса сотового оператора связи.

Объект исследования: поиск оптимального набора свойств и качеств интерфейса для выявления его эффективности и построение модели.

Предмет исследования: построение математическая модели и набора тестовых данных.

Цель работы является сравнение модели, практических результатов и практическая реализация веб интерфейса с применением технологии VUE.JS.

В введение приводится актуальность, необходимость и современность бакалаврской работы.

Первая глава содержит постановку задачи, анализ существующих web технологий для разрабатываемого приложения, анализ существующих веб интерфейсов, методы его тестирования, а также проектирование и назначение биллинг системы.

В второй главе осуществляется анализ набора тестовых данных и разработка модели оценки пользовательского интерфейса, происходит выбор среды и практическая реализация интерфейса с применением технологии VUE.JS, а так же реализация серверной части приложения биллинг системы.

В третьем главе выполняется оценка качества пользовательского интерфейса, а так же ручное и автоматизированное тестирование интерфейса.

Результатом работы является биллинг система для мобильного сотового оператора связи.

Abstract

The present graduation work is devoted to developing a model for evaluating the attractiveness of the Web interface of an application using VUE.JS technology.

The relevance of the research consists in the problem of the mobile network operator user interface effectiveness.

The object of the graduation work is searching for the optimal set of properties and features of the interface to identify its effectiveness and to construct a model.

The subject of the graduation work is constructing a mathematical model and choosing a test data set.

The purpose of the work is to compare the model, the practical results and the practical implementation of the web interface using the VUE.JS technology.

The introduction reveals the relevance and the scientific novelty of the investigation.

The first chapter states the problem, as well as analyzes the existing web technologies for the suggested application, the existing web interfaces and the methods of testing. This chapter also designs the billing system and explains its applicability.

The second chapter analyzes the user interface test data set, develops a model of evaluating the user interface, selects the environment, as well as implements the interface using VUE.JS technology and the server part of the billing system.

The third chapter assesses the quality of the user interface, as well as manual and automated testing of it.

The result of the research is the billing system designed for a mobile network operator.

Оглавление

Введение

В современном мире мобильные сотовые операторы стали неотъемлемой частью нашей жизни. Каждый из них имеет своё веб приложение. Слово «веб-приложение» относится к пользовательскому интерфейсу, а именно к «сайту». С каждым днём число пользователей продолжает расти в геометрической прогрессии. Причина этого роста заключается в предоставлении широких возможностей пользователям с интенсивным развитием информационных технологий в мире.

В настоящее время при разработке программного продукта особое внимание уделяется, скорости обработки информации, доступности, хранение и передача информации. Тем не менее, создание пользовательского интерфейса играет не малую роль, поскольку интерфейс определяет, насколько легко и удобно будет пользоваться сайтом.

Пользовательский интерфейс представляет собой набор средств ввода данных, методы отображения информации и элементы управления, используемые в программе.

Эффективность программного обеспечения определяется не только его функциональностью, но и наличием элементов интерфейса, с которым, взаимодействует пользователь. Таким образом, интуитивно понятный, удобный интерфейс – важная задача при разработке программного продукта.

Глава 1 Проектирование биллинг системы и пользовательского интерфейса

1.1 Постановка задачи

Прежде чем приступить к разработке пользовательского интерфейса, первое что необходимо сделать это разобрать на компоненты будущую биллинг систему (БС) и сформировать требования, к которым данное приложение должно соответствовать.

Формирование требований необходимы процесс для любой разработки программного обеспечения в том числе веб-разработки, они позволяют избежать множество проблем и служат своего рода «ориентиром» в реализации любых приложений. Для того что бы сформировать требования к веб-приложению необходимо понять для каких целей данное приложение будет использоваться.

Данная БС будет осуществлять различные виды транзакций, расчётного и обслуживающего плана. Эта информационная система (ИС) будет разделена на три компоненты, а именно АТС, CRM, SSP.

Каждая компонента биллинг системы будет выполнять определённую роль в веб-сервисе, а именно АТС – будет служить для эмуляции сотового телефона клиента, CRM – для изменения тарифов, создания их и удаления, SSP – личный кабинет пользователя, необходим для переключения между тарифами и пополнения лицевого счёта. Также для создания полноценной биллинг системы обязательными компонентами будут являться Billing Api и Billing DB. Billing Api не обходим для дальнейшей возможности взаимодействия, например с мобильными приложениями на базе операционных систем: Android, iOS, Windows Phone. В компоненте Billing DB будет хранится база данных приложения. На рисунке 1 изображён пример взаимодействия компонентов ИС.

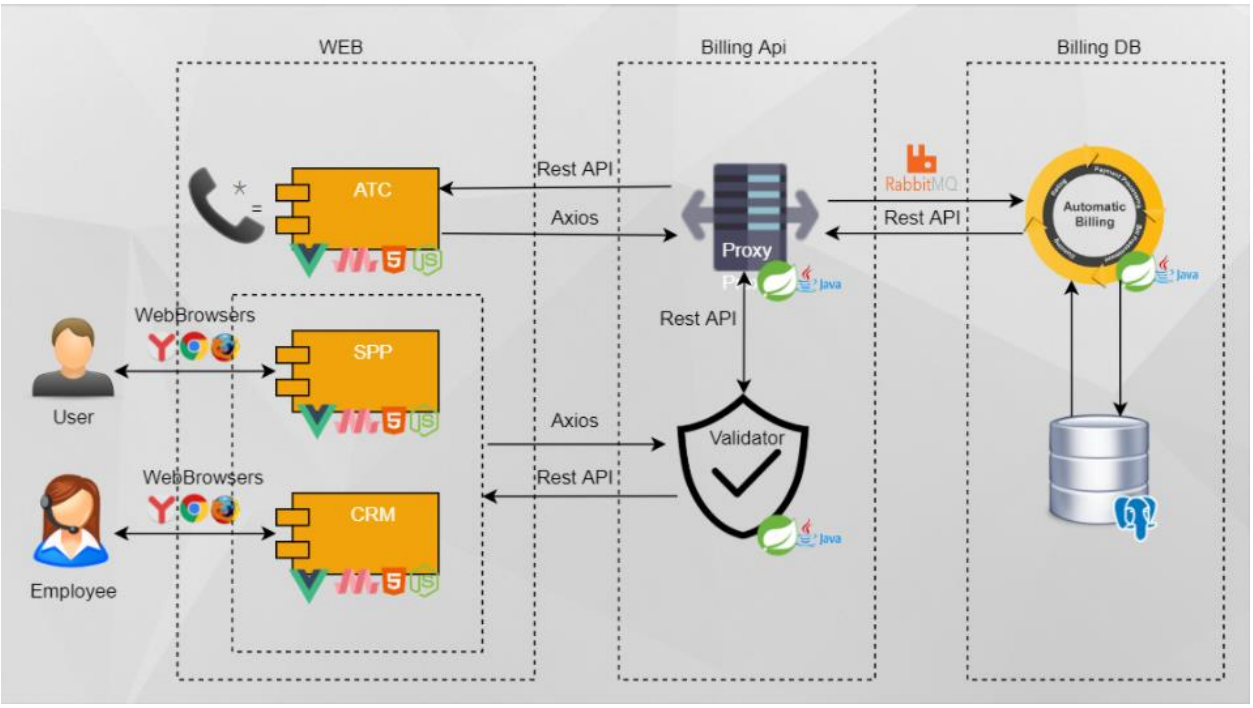

Рисунок 1 – Взаимодействие компонентов ИС

После того как были сформированы цели данного веб-сервиса можно приступить к формированию требований к данному приложению.

В 1990-годах Робертом Грейди впервые была сформирована классификация уровней требований, которая актуальна до сих пор. Классификация имеет название FURPS она содержит такие критерии как:

– Функциональность,
– Производительность,
– Удобство использования,
– Надёжность,
– Поддерживаемость.

Перечислив каждую классификацию, приступим к формированию требований для каждого из перечисленных разделов.

Требования к функциональности биллинг системы:

– Расчёт использования ресурсов по тарифу, такие как минуты, мегабайты и sms,

– Пополнение лицевого счёта,

– Создание и редактирование тарифов.

Требования к удобству использования (как правило, применительно к интерфейсу):

– эстетическая привлекательность,

– интуитивная понятность.

Требования к надежности веб-приложения:

– режим доступности 24/7,

– точность проводимых расчетов,

– пригодность системы к восстановлению данных и информации,

– отказоустойчивость.

Требования к производительности системы:

– быстрый отклика системы (время реакции системы на действие пользователя),

– масштабируемость (в случае роста числа пользователей),

– малое потребление ресурсов (финансовые затраты).

Требования к поддержке системы:

– совместимость с программно-аппаратным обеспечением и другими системами,

– простота установки.

В данном разделе были сформированы необходимые требования для разрабатываемой биллинг системы, при соблюдении которых, можно получить гарантированно качественное приложение.

1.2 Анализ существующих WEB-технологий

На сегодняшний день существуют сотни тысяч веб приложений построенных на различных видах архитектурных решений. Каждое такое решение для той или иной задачи является оптимальным, в какой-то степени

технологий реализация проходят путь проб и ошибок, таким образом формируются особенности, преимущества и недостатки тех или иных решений.

Чтобы создать качественное ПО необходимо придерживаться сформированным требованиям к программному обеспечению, а это значит технологии, которые должны быть использованы в создание биллинг системы не должны противоречить этим требованиям.

Технологии разработки программных систем совершенствуются с каждым днём, день за днём появляются новые языки программирования, новые фреймворки и новые библиотеки. Все это благодаря тому, что с каждым днём все больше и больше людей увлекаются разработкой ПО, тем самым способствуя появлению этих технологий. Например, фреймворк VUE.JS, созданный всего одним человеком, имеет на сегодняшний день огромное количество разработчиков. Фреймворк VUE.JS разрабатывается по принципу открытого кода. В 2013 году разработчик Эван Ю, работая на компанию Google, параллельно стал заниматься разработкой нового фреймворка. Тогда он придумал для него название Seed, но так как данное имя было занято библиотекой NPM, он назвал его VUE, в переводе с французского «вид». Эван создал легкий и интуитивно понятный инструмент, который стал привлекать все больше и больше внимания со стороны. Он опубликовал данный фреймворк на платформу Github, открыв доступ всем разработчикам в мире. С каждым днём количество людей, присоединяемых для разработки данного фреймворка растёт, а самое интересное в этом то, что люди, которые помогают в разработке этого фреймворка не получают никаких денежных вознаграждений.

По такому принципу разработки, как фреймворк VUE.JS, на сегодняшний день строиться огромное количество новых технологий, фреймворков, библиотек, языков программирования. Конечно, есть технологии, за которыми стоят большие интернет-гиганты, такие как

Facebook(React.JS), Google(AngularJS). Они способствуют развитию этих технологических решений, потому что сами используют их в разработке своих продуктов. Конечно, компании, которые решили разработать свой веб-сайт, допустим для интернет-магазинов, пока что не очень доверяют таким технологиям как VUE.JS, потому что за ними стоят ряд «добровольных людей» нежели технологий, за которыми стоят большие интернет-гиганты, такие как Google, которые гарантируют поддержку данной технологий ещё долгие годы.

Для проектировании биллинг системы было выбрано три современных популярных фреймворков для веб разработки, а именно: Angular.JS, React.JS, VUE.JS. Так как для разработки качественного ПО необходимо придерживаться требований FURPS (быстродействие) Для этого, ниже на рисунках 1.2-1.4, представлены таблицы тестирования быстродействия фреймворков. Тестирование производилось на устройстве MacBook Pro 15. (i7 2,5 ГГц, 16 ГБ ОЗУ, OSX 10.13.6, Chrome 69.0.3497.100 (64-разрядная версия)[19].

Name	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed
create rows Duration for creating 1000 rows after the page loaded.	182.1 ± 7.6 (1.0)	185.2 ± 10.2 (1.0)	180.5 ± 7.3 (1.0)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	158.8 ± 2.7 (1.0)	161.2 ± 2.7 (1.0)	157.3 ± 2.0 (1.0)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	156.4 ± 9.6 (2.3)	68.8 ± 3.7 (1.0)	81.9 ± 2.7 (1.2)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	10.6 ± 2.0 (1.0)	7.9 ± 4.3 (1.0)	10.3 ± 2.1 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	20.0 ± 2.9 (1.0)	105.8 ± 1.8 (5.3)	106.5 ± 1.9 (5.3)
remove row Duration to remove a row. (with 5 warmup iterations).	54.2 ± 2.2 (1.2)	47.1 ± 3.0 (1.0)	49.6 ± 0.8 (1.1)
create many rows Duration to create 10,000 rows	1,603.2 ± 34.8 (1.0)	1,693.9 ± 70.1 (1.1)	1,935.4 ± 33.6 (1.2)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	342.5 ± 6.0 (1.4)	243.3 ± 6.3 (1.0)	268.6 ± 6.9 (1.1)
clear rows Duration to clear the table filled with 10.000 rows.	191.9 ± 6.1 (1.1)	263.9 ± 3.0 (1.5)	175.4 ± 4.1 (1.0)
slowdown geometric mean	**1.17**	**1.27**	**1.27**

Рисунок 2 – Тест вычислительных операций

Name	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,252.7 ± 0.2 (1.0)	3,278.5 ± 4.0 (1.5)	2,477.6 ± 0.6 (1.1)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	55.1 ± 1.5 (1.0)	235.2 ± 8.5 (4.3)	65.6 ± 2.3 (1.2)
main thread work cost total amount of time spent doing work on the main thread. Includes style/layout/etc.	420.6 ± 67.5 (1.0)	679.3 ± 5.3 (1.6)	466.0 ± 3.9 (1.1)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	215,445.0 ± 0.0 (1.0)	365,497.0 ± 0.0 (1.7)	251,915.0 ± 0.0 (1.2)

Рисунок 3 – Тест запуск метрик

Name	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed
ready memory Memory usage after page load.	2.7 ± 0.2 (1.0)	6.0 ± 0.0 (2.2)	2.8 ± 0.2 (1.0)
run memory Memory usage after adding 1000 rows.	7.1 ± 0.0 (1.1)	9.8 ± 0.0 (1.5)	6.7 ± 0.0 (1.0)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	7.2 ± 0.0 (1.0)	9.8 ± 0.0 (1.4)	7.6 ± 0.0 (1.1)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	7.2 ± 0.0 (1.0)	10.1 ± 0.0 (1.4)	7.9 ± 0.0 (1.1)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	3.0 ± 0.0 (1.0)	6.4 ± 0.0 (2.1)	3.8 ± 0.0 (1.3)

Рисунок 4 – Тест распределения памяти

На рисунках 2 - 4 в левом столбце изображены названия тестов, а также их описание, в первой строке отображено название фреймворков, ниже в цифрах изображена их производительность соответственно каждому тесту. Проанализировав полученные данные изображенные на рисунке 2 - 4 можно сделать вывод, что Фреймворк VUE.JS более быстрый по сравнению с Фреймворками Angular.js и React.js, а значит он подходит больше для разработки ИС мобильного сотового оператора связи.

Также сравним подробнее эти технологии, во многом React и Vue схожи, они оба используют Virtual DOM, представляют реактивную и компонентную структуру, выносят в дополнительные библиотеки такие вопросы как роутинг или управление глобальным состоянием. Конечно, React превосходит Vue в богатстве экосистемы и изобилия доступных пользовательских средств отрисовки. Например, в фреймворке React, когда состояние компонента изменяется, он запускает повторную отрисовку всего поддерева компонента. Конечно, этого можно избежать, используя метод «PureComponent» везде, где это возможно. В отличии от Vue, там зависимость компонента отслеживается автоматически при каждой отрисовке, поэтому система точно знает, какие компоненты действительно необходимо повторно отрисовывать при изменения состояния, а какие нет. В целом это устраняет необходимость написания лишнего кода, что позволяет больше сосредоточиться на построении самого приложения или его масштабирования.

Некоторые части синтаксиса Vue выглядят очень схожими с Angular, например условие «если» v-if на Vue.js и ng-if на Angular.JS. Это не случайно, многие идеи при разработке фреймворка Vue были взяты именно с Angular, так как создатель фреймворка Vue, Эван Ю, работал на компанию Google, где разрабатывался Angular. Несмотря на то, что части синтаксиса выглядят похожими, оба фреймворка отличаются разным уровнем вхождения, например у Angular фреймворка API просто очень огромное, и

как новому пользователю нужно будет разбирается с большим количеством концепций, прежде чем стать продуктивным разработчиком. В отличии от технологии Vue, где полученных за день знаний достаточно.

Итак, в данном разделе были рассмотрены основные преимущества и недостатки технологий для веб-разработки, а таже был проведён их сравнительный анализ производительности.

1.3 Проектирование и назначение биллинг системы

Биллинг система – это система, которая по своей сущности должна решать ряд задач финансового, расчётного и обслуживающего плана.

Так она должна использовать денежные средства клиентов, а именно пополнять баланс, тратить, а также предоставлять различные услуги по тарифам.

Данную БС можно использовать малым сотовым операторам, у которых ещё не было своей биллинг системы, а также тем у кого нет большого количества пользователей.

Биллинг система ориентирована для реализации задач связанных расчётного обслуживающего плана. Она представляет из себя систему сотового оператора связи, которая обеспечивает процесс обмена сообщений вычислений трат по тарифу. Своего рода является вычислительным ядром для мобильной связи.

Определим основные функции биллинг системы:

– Настройка параметров тарификации услуг,

– Тарификация предоставления услуг.

Прежде чем приступить к разработке биллинг системы необходимо построить блок схему различных сценарий взаимодействия. Сценарии пользования необходимы для проверки функциональности

пользовательского интерфейса. Ниже описаны основные типы сценарий системы.

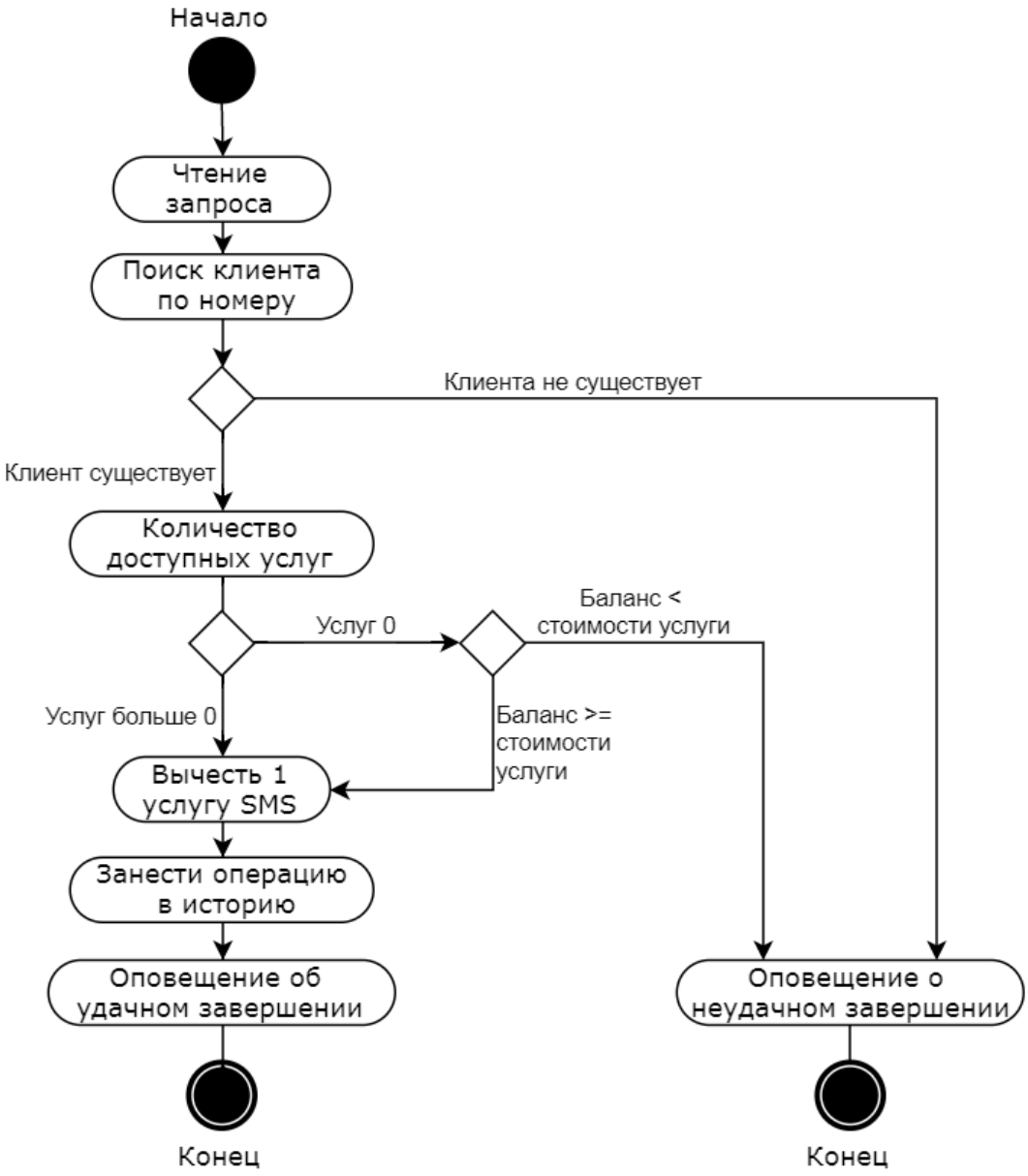

Рисунок 5 – Диаграмма деятельности отправки смс

На рисунке 5 демонстрируется сценарий, при котором пользователь отправляет смс другому пользователю. В данном сценарии используется проверка на существование текущего клиента в системе, так же проверяется количество доступных ему услуг по данному тарифу, в данном случае в сценарии рассмотрено именно смс, и если у данного пользователя доступны

услуги или баланс более стоимости услуги, происходит списывание денежных средств и занесении данной операции в историю.

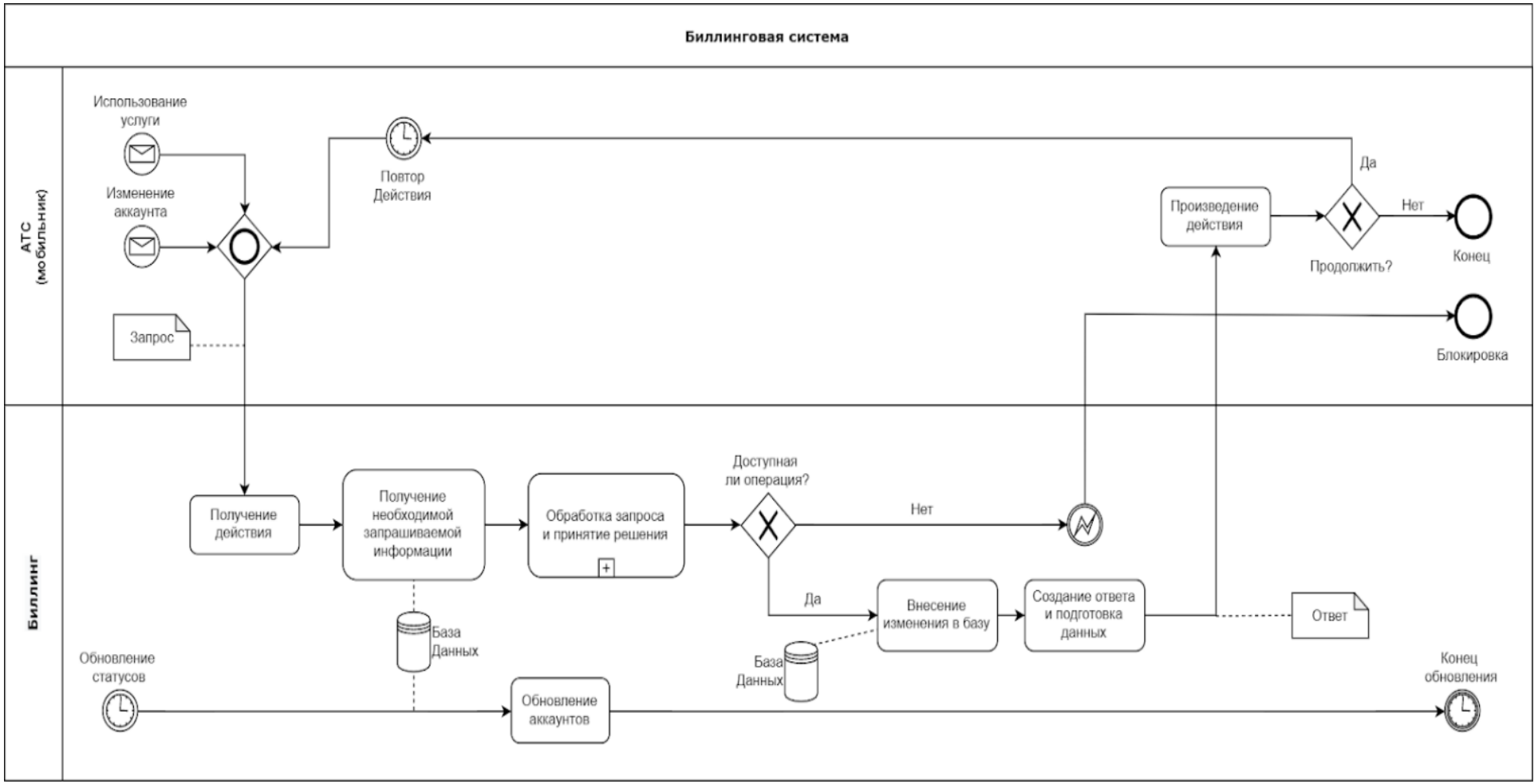

Рисунок 6 – Взаимодействие биллинг системы с компонентой АТС

На рисунке 6 демонстрируется функционал биллинг системы который показывает изменение статуса услуги текущего клиента, а также изменения аккаунта при взаимодействии с базой данных клиента.

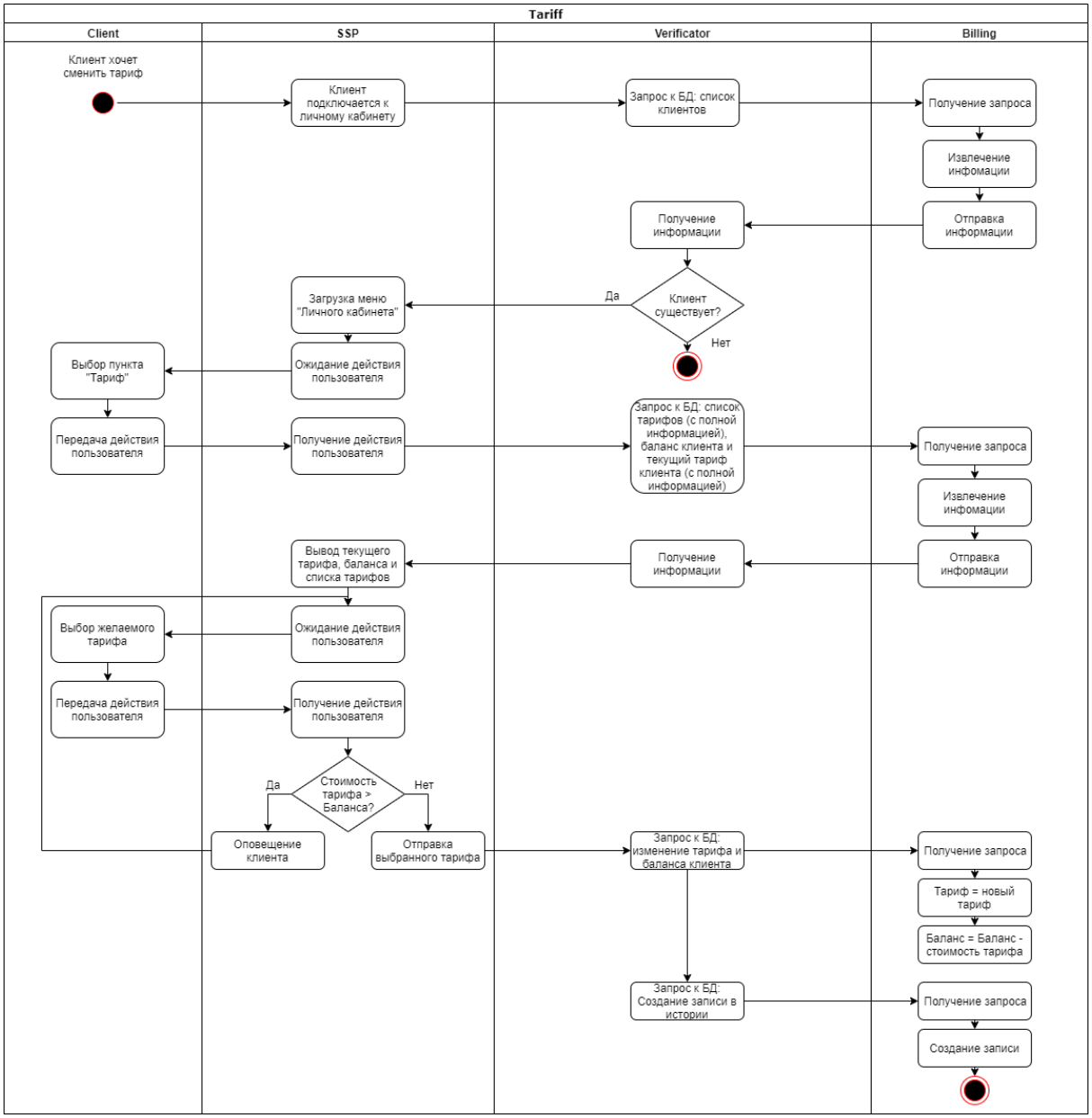

Рисунок 7 – Диаграмма смены тарифа клиента

На рисунке 7 демонстрируется взаимодействие компонентов SSP Verificatory и Billing, в последовательном порядке при смене тарифа пользователя.

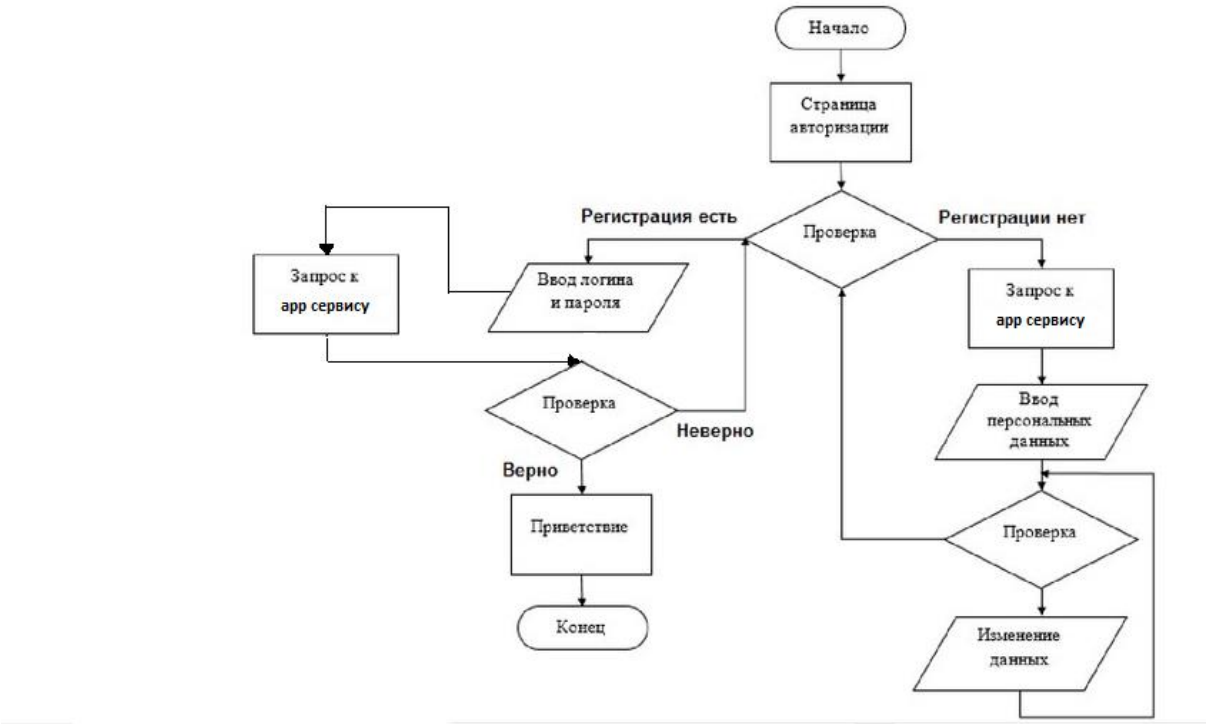

Рисунок 8 – Сценарий авторизации пользователя

На рисунке 8 демонстрируется блок схема сценария, при котором пользователь осуществляет авторизацию в системе. Если же пользователь ранее не был зарегистрирован, то пользователь направляется на регистрацию в системе. Если же пользователь зарегистрирован, то он направляется на проверку правильности введённых данных. После успешно авторизации его перенаправляет на домашнюю страницу сайта.

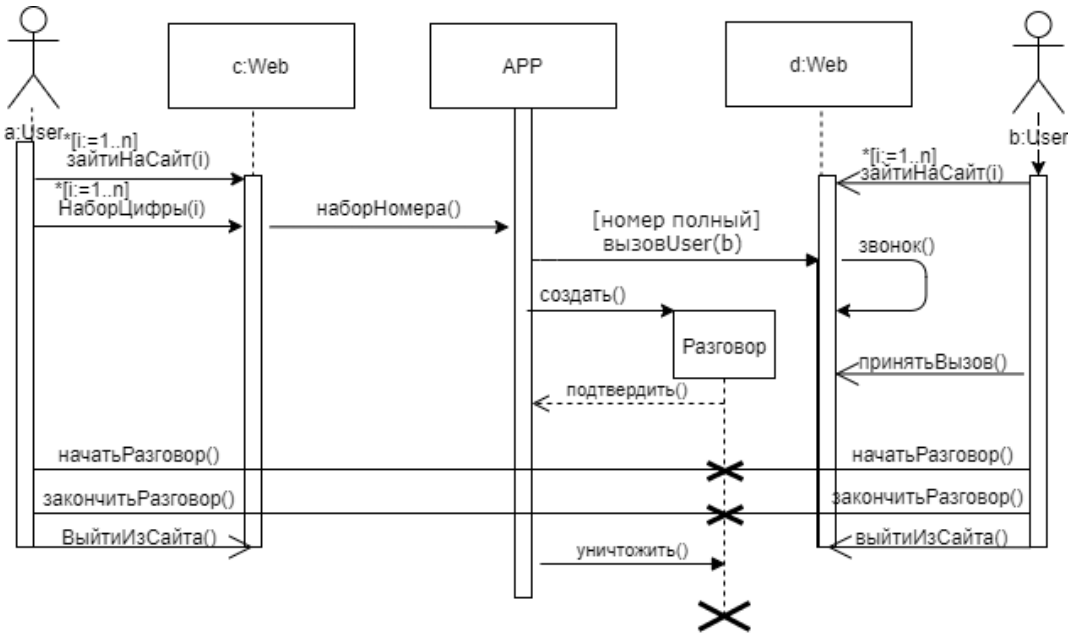

Рисунок 9 – Сценарий звонка пользователя

На рисунке 9 изображена диаграмма последовательностей, в которой демонстрируется сценарий, при котором пользователь осуществляет звонок другому пользователю.

Так как компонента расчётной биллинг системы тесно связана с базой данных, содержание базы данных таблиц продемонстрированно на рисунке 10.

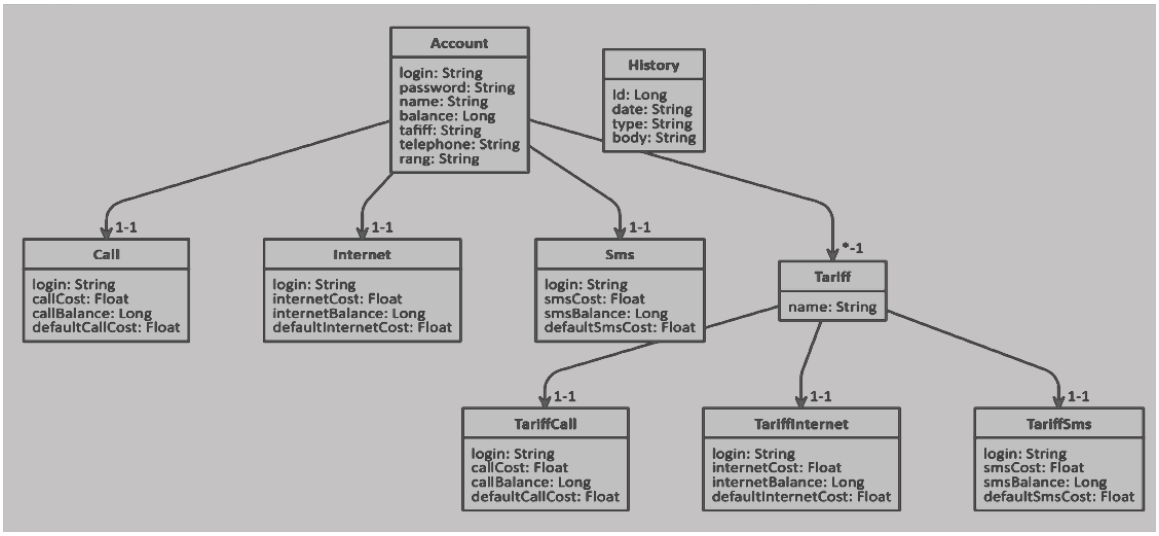

Рисунок 10 – Схема базы данных биллинг системы

На рисунке 10 продемонстрирована схема базы данных в MySQL с связями и типами полей в таблицах. Таблица Account служит для хранения данных об аккаунте пользователя, таблица history для хранения истории, а именно: дата, тип, количество потраченных ресурсов. Таблица tariff для хранения названий тарифов. Значения по тарифам вынесены в отдельные таблицы TariffCall, TariffInternet, TariffSms.

Итак, в данном разделе были рассмотрены основные сценария биллинг системы, которые необходимо учитывать при веб-разработке приложения, был проведён анализ, а также продемонстрированы таблицы базы данных системы.

1.4 Анализ пользовательского интерфейса

После формулировки сценарий пользователя можно перейти к формированию пользовательского интерфейса, а значит и к его анализу.

Каждый пользовательский интерфейс разрабатывается в соответствии требованиями к будущей системе. Пользовательский интерфейс представляет собой совокупность используемых в программе средств ввода данных, способов отображения информации и элементов управления. Эффективность работы программного продукта определяется не только его функциональными возможностями, но и доступностью этих возможностей. Поэтому создание интуитивно-понятного дружественного интерфейса является важной задачей при разработке программного продукта.

Третьей компанией в Европе по сбору данных в пользовательском интерфейсе является компания Яндекс, с сервисом Яндекс.Метрика [15].

Яндекс.Метрика – это инструмент веб-аналитики, который помогает получать наглядные отчеты, видеозаписи действий посетителей, отслеживать источники трафика и оценивать эффективность. На рисунках 11-13 продемонстрированы некоторые возможности данного инструмента.

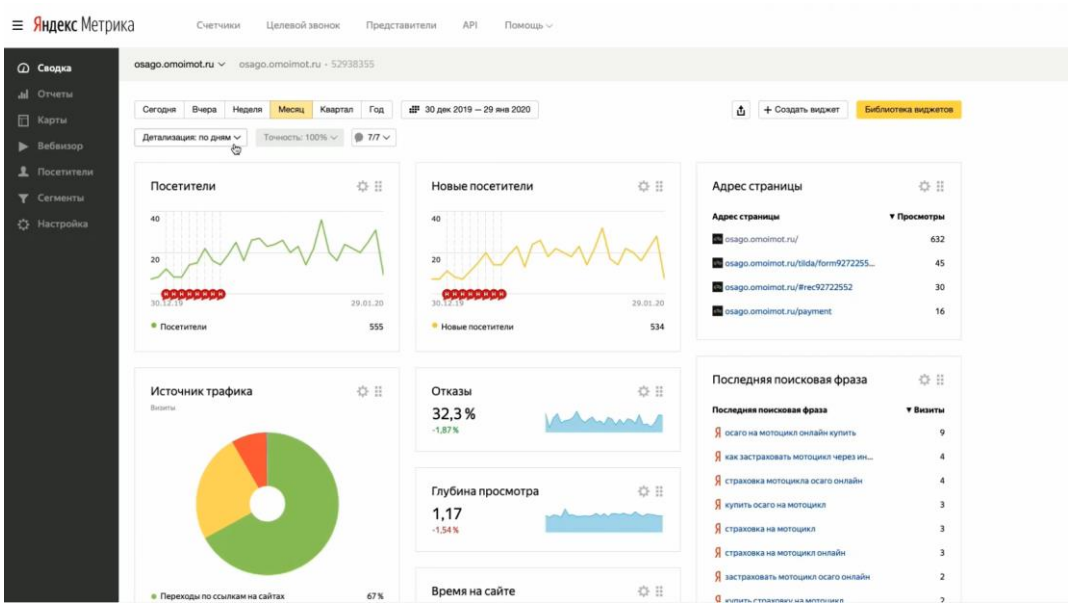

Рисунок 11 – Инструмент аналитике статистических данных

На рисунке 11 продемонстрирован функционал Яндекс.Метрики по сбору статистических данных. На данном рисунке демонстрируются статистические данные о количестве посетителей на сайте за определённый промежуток времени, количество новых посетителей, а также по какой фразе из поискового запроса пользователь посетил на сайт.

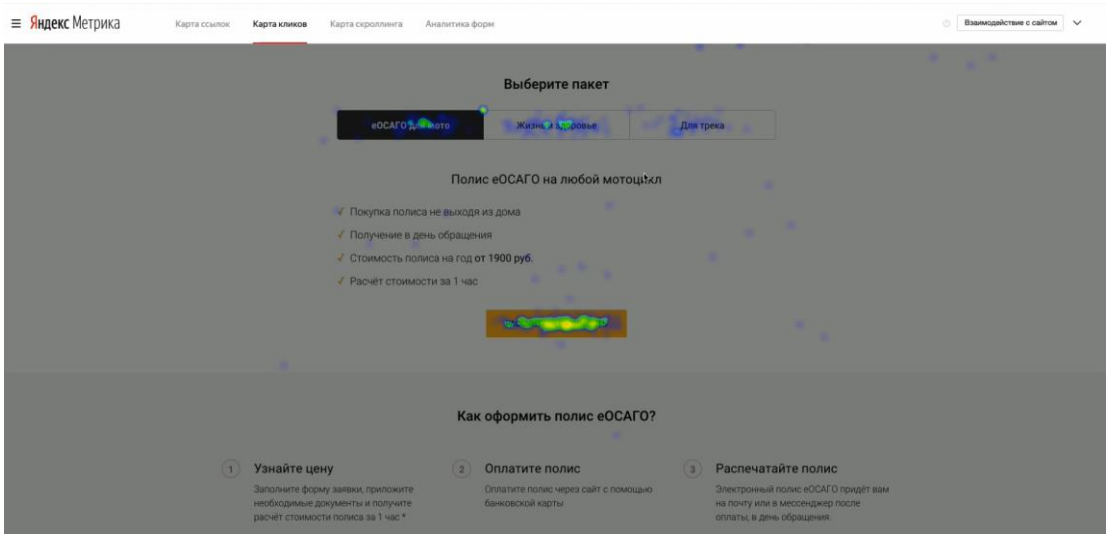

Рисунок 12 – Инструмент аналитики кликабельности страницы

На рисунке 12 демонстрируется аналитика кликабельности страницы. На нём от темно-синего до ярко-красного демонстрируются активные зоны сайта, там, где пользователь чаще всего «кликает». Анализируя данную статистику можно прийти к выводу эффективности того или иного элемента интерфейса сайта.

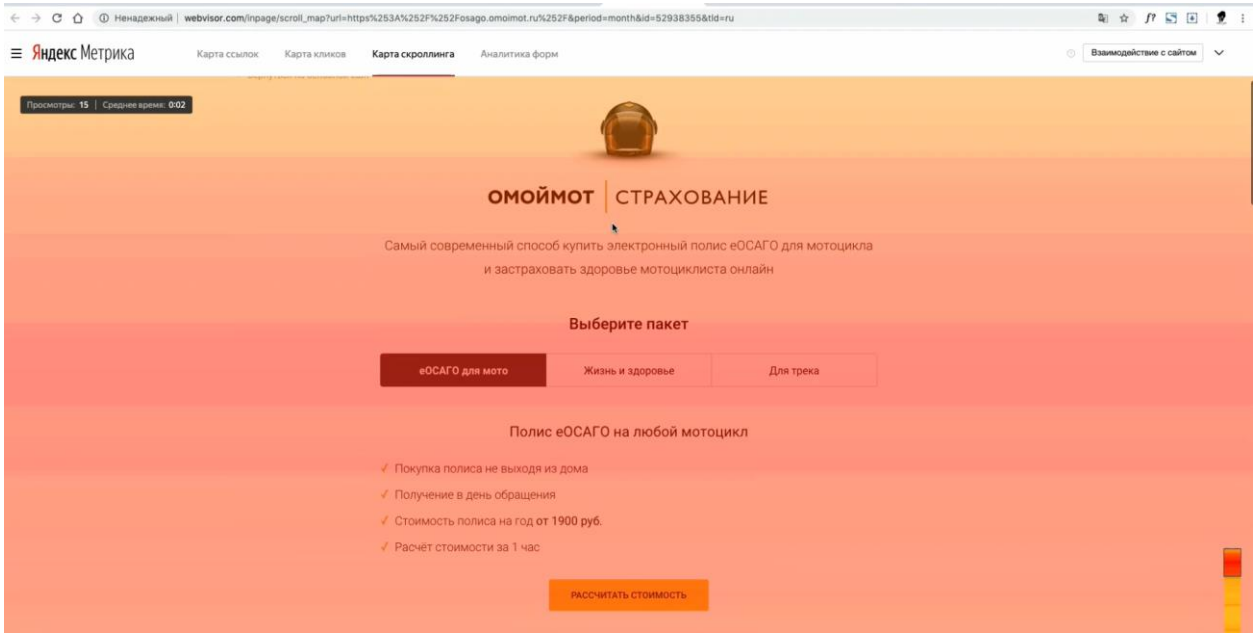

Рисунок 13 – Инструмент аналитики «карта скролла страницы»

Инструмент аналитики «карта скролла страницы» изображен на рисунке 13 на данном рисунке от темно-синего до ярко-красного показаны места активных зон страницы, там где пользователи чаще всего просматривают страницу. Удивительно, но чем ниже скролить сайт, тем темнее линии активных зон будут. Возможно это объясняется тем что пользователям не хочется скролить страницу вниз, они сразу хотят получить результат, особенно с мобильного телефона, возможно, так как движение скролла страницы сложнее, чем просто клик.

Прежде приступить к разработке пользовательского интерфейса, необходимо исследовать существующие виды решений пользовательского

интерфейса мобильных сотовых операторов связи, для этого воспользуемся сайтами сотовых операторов связи, например: мтс, билайн, мегафон, йота, теле2. Ниже на рисунках 13-17 изображены главные страницы данных сайтов.

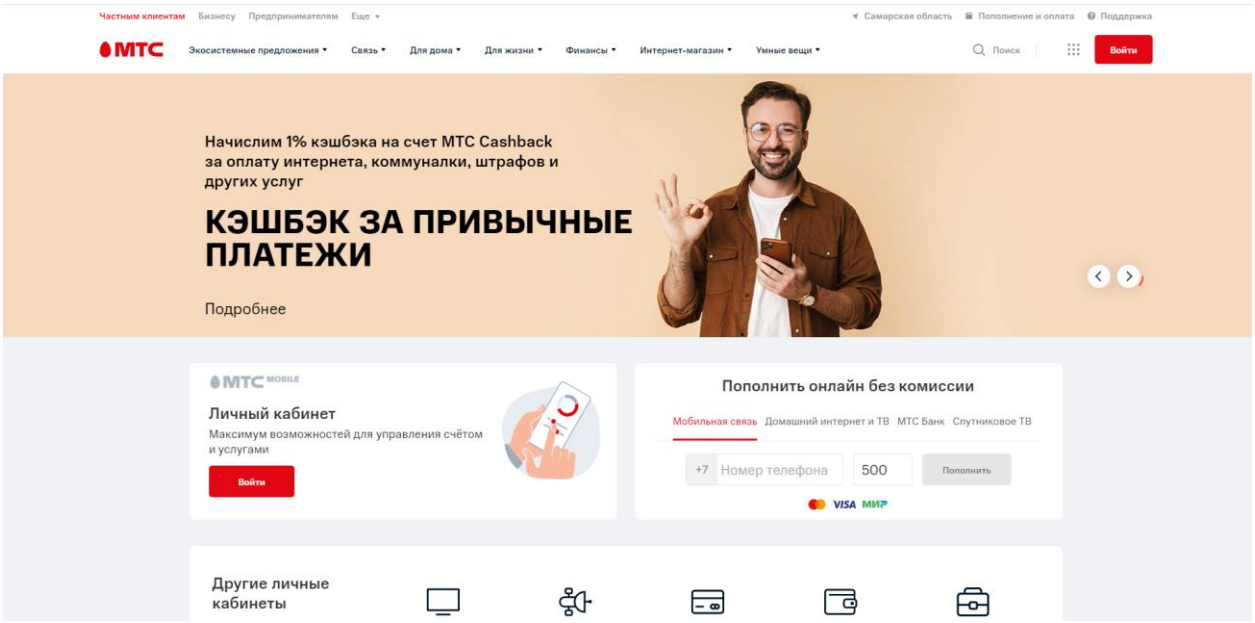

Рисунок 13 – Главная страница МТС

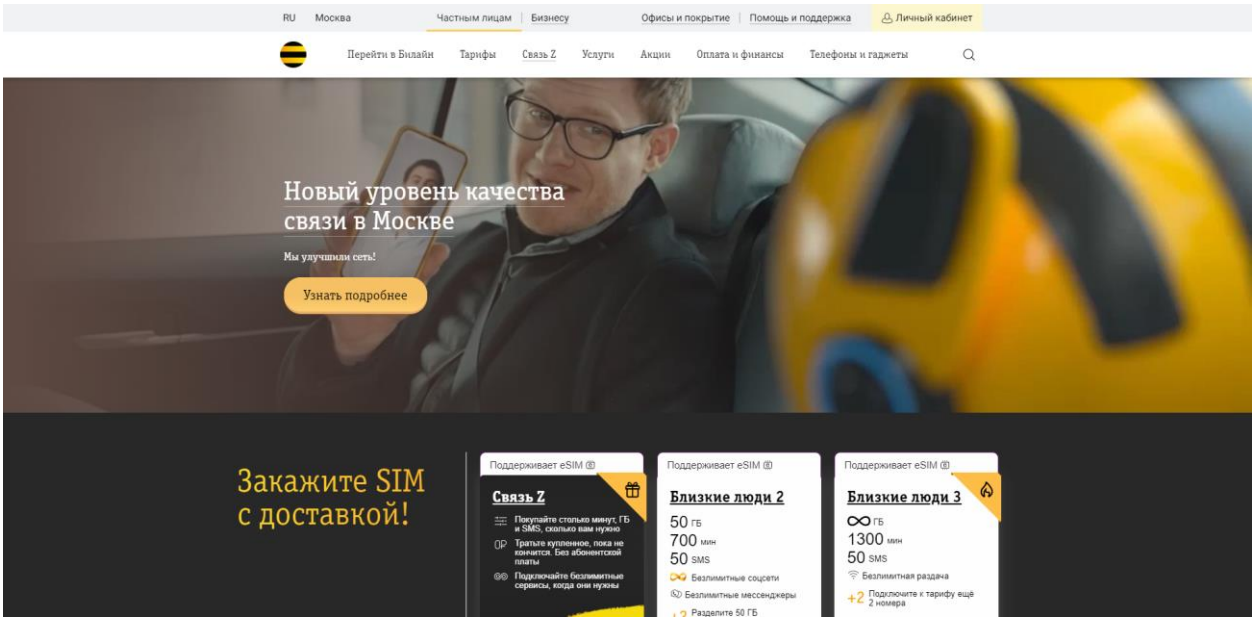

Рисунок 14 – Главная страница Билайн

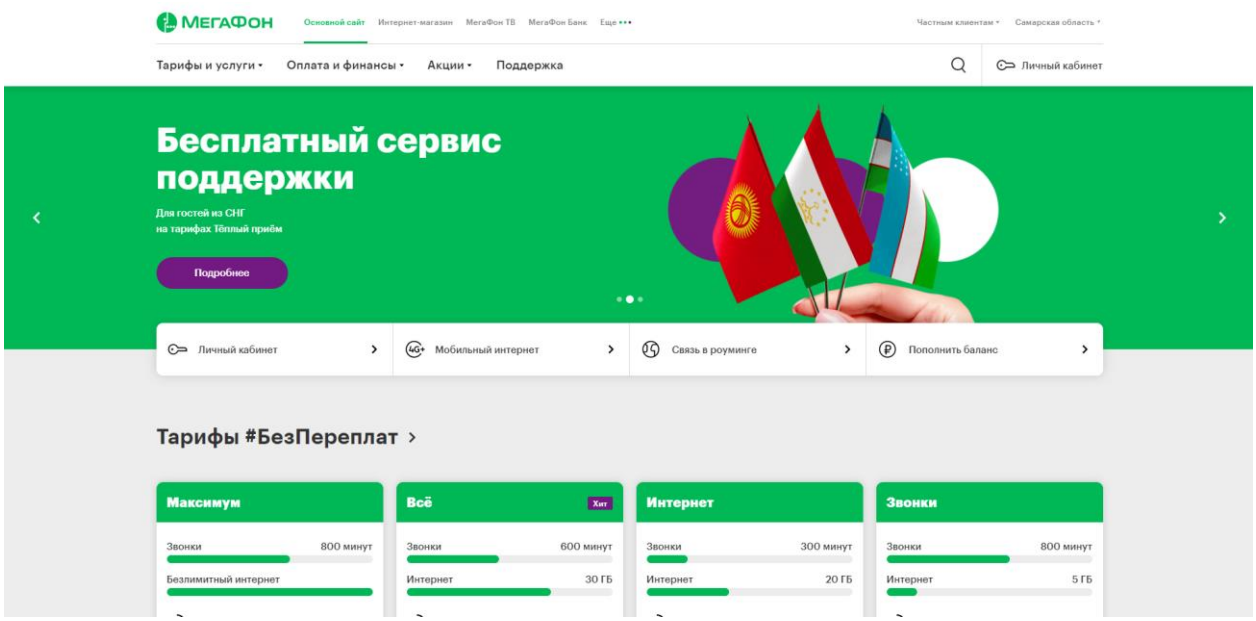

Рисунок 15 – Главная страница Мегафон

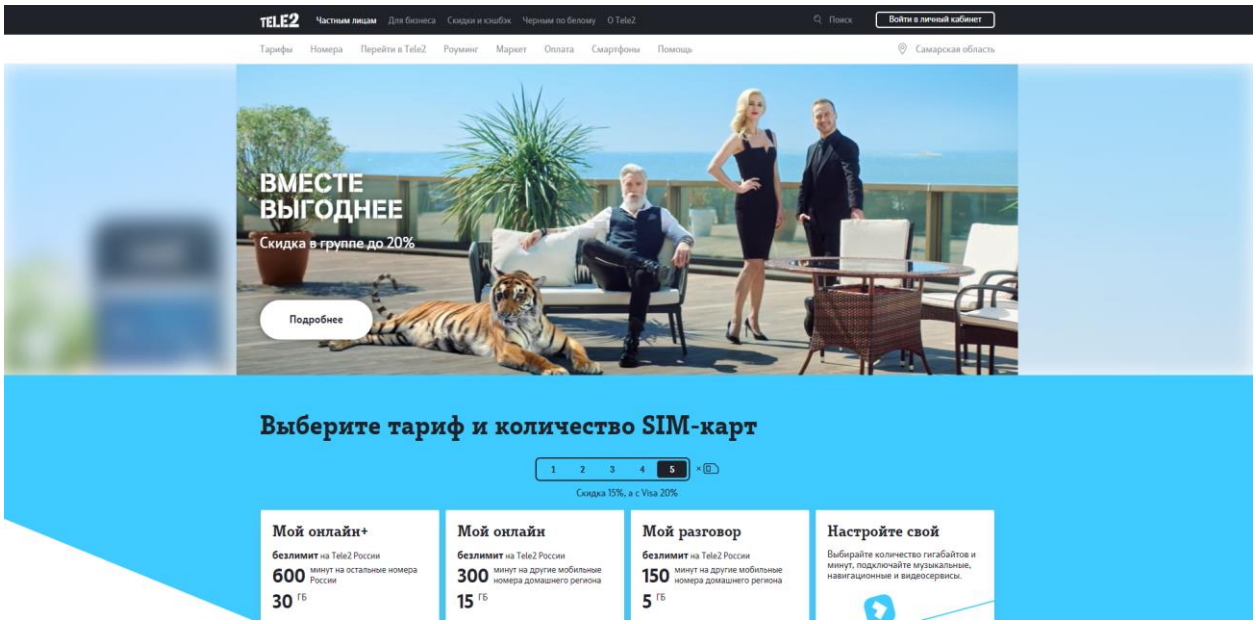

Рисунок 16 – Главная страница Теле2

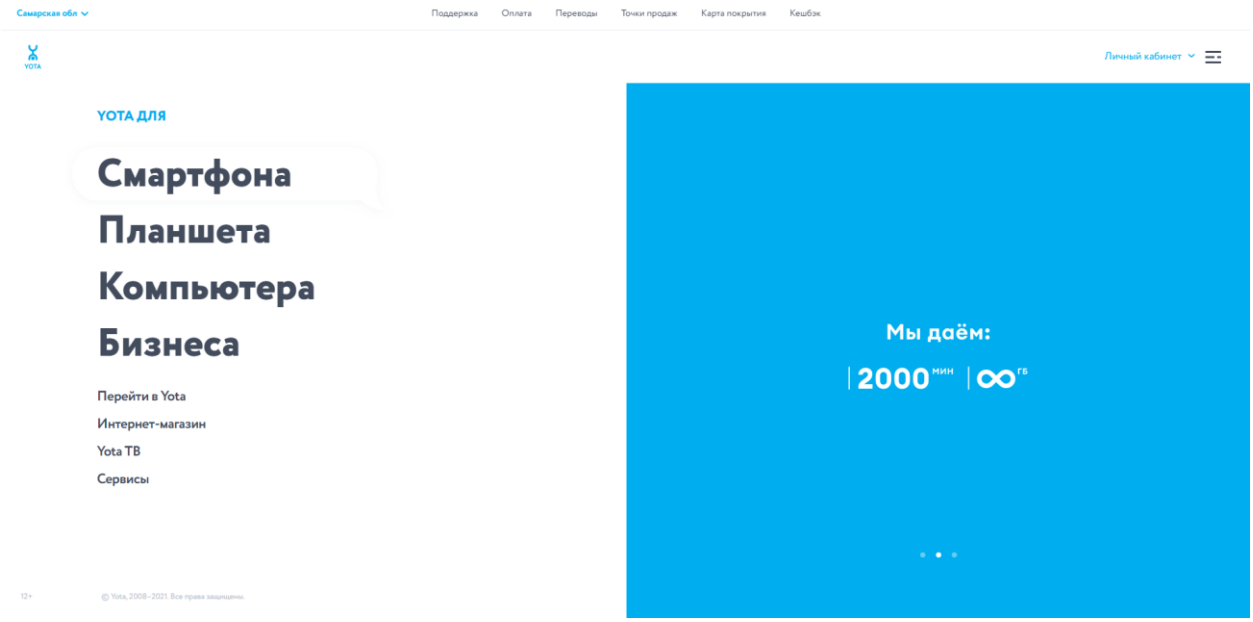

Рисунок 17 – Главная страница Yota

В трёх сайтах из пяти, цены тарифов присутствуют на главной странице, причем не скроля страницу значительно вниз можно увидеть тарифы с их соответствующими ценами, это является эффективным решением, потому что тарифы располагаются почти на самой активной карте скроллинга, что является интуитивно понятным для пользователя, а значит эффективность данных сайтов будет значительно превосходить тех у кого данный элемент отсутствуют на главной странице.

В данном разделе был проведён анализ дизайна пользовательского интерфейса сотовых операторов. В процессе анализы была выдвинута гипотеза о том, что наличие тарифов в активной карте скроллинга позволяет упростить пользователю нахождению тарифов, а значит и улучшить эффективность пользовательского интерфейса сайта.

1.5 Требования к тестированию программного обеспечения

Качественное программное обеспечение является важным параметром в любой информационной системе. Допустим если интерфейс отображается с ошибкой - «кнопка не работает», и пользователь не сможет получить ожидаемый результат при её нажатии, то с определенной долей вероятности у пользователя сложится плохое впечатление и он покинет сайт или откажется от посещения сайта в будущем. Возможно это объяснится тем, что пользователи доверяют свои данные, денежные средства сайту, а значит и сайт должен соответствовать требованиям качественного ПО, что бы пользователь был уверен в надёжности предоставляемых услуг сайта.

Чтобы БС советовала качественному ПО, необходимо сформировать требования к тестированию:

- Тесты в первую очередь должны основываться на требованиях к программному обеспечению,
- Тесты должны разрабатывается для проверки правильности функционирования и создания условий для выявления потенциальных ошибок,
- Поиск очевидных ошибок в функционировании пользовательского интерфейса.

Виды тестирования:

- Ручное тестирование,
- Автоматизированное тестирование.

Автоматизированное тестирование заключается в проверке основного функционала сайта, такого как отображение модального окна авторизации, проверки корректной смены тарифа, отображения страницы истории и многих других функций пользовательского интерфейса. При использовании автоматизированных средств, таких как веб драйвера Selenium, можно задать последовательность действий команд браузеру, тем самым покрыть тестами

основной функционал сайта. Преимущество автоматизированного тестирования заключается в том, что оно не требует постоянного контроля со стороны разработчика и может быть запущено через определённый промежуток времени, для выявлении критических ошибок в функциональности сайта и внутренних систем.

Ручное тестирование заключается в проверке полного и общего функционала сайта только тогда, когда автоматизированное тестирование не актуально, или не может быть использовано.

Выводы по первой главе

В первой главе были сформированы требования к БС, проведён анализ существующих web технологий со сравнением преимуществ и недостатков каждой технологии. Проведено проектирование БС, результатом которого являются диаграммы, описывающие функциональные требования к системе, которые станут основополагающими в функциональности разрабатываемого веб интерфейса, а так же сформированы, требования к тестированию БС и произведён анализа существующих пользовательских интерфейсов сотовых операторов, результатом которого была сформирована гипотеза о том, что эффективным решением пользовательского интерфейса мобильных сотовых операторов связи является присутствие информации о тарифах на активной карте скроллинга.

Глава 2 Разработка модели оценки привлекательности и веб-приложения на основе технологии vue.js

2.1 Анализ статистических данных

Прежде чем приступить к написанию веб-приложения, нужно определится с дизайном сайта. В параграфе 1.4 была сформирована гипотеза, что наличие тарифов в активной карте скроллинга позволяет упростить пользователю нахождение тарифов, а значит и улучшить эффективность пользовательского интерфейса сайта.

Эффективность пользовательского интерфейса – это продуктивность использования пользовательского интерфейса, с целью достижения каких-либо результатов. Так же эффективность заключается в удобстве для пользователя, например, она тесно связано с принципом ожидания.

Принцип ожидания заключается в том, что каждый человек от проведенной им деятельности ожидает, то есть надеется, что его усилия будут оправданы и вознаграждены в той или иной степени.

Для того чтобы отвергнуть или принять гипотезу, было опрошено 20 человек среднего возрастного диапазона от 18 до 35 лет. Опросы содержат в себе различные грани действующей гипотезы для того, чтобы упростить опрашивающему прохождение опроса, а также упростить понимание вопроса в целом. В процессе опроса были получены результаты, которые были сгруппированы в диаграммы, для упрощённого их анализа.

Рисунок 18 – Статистические данные опроса

На рисунке 18 изображена диаграмма с результатами на ответы пользователей на вопрос: «На каком сайте сотового оператора Вам проще найти нужную Вам информацию?». Результаты отображены в количественном и в процентом соотношении. Количественное значение отображено по ширине от 0 до 10. Проанализировав данную диаграмму, можно прийти к выводу, что у сотовых операторов теле2 и мегафон значительно превосходят других операторов, также можно заметить, что сайты YOTA и MTS получили наименьшее количество голосов, а значит являются аутсайдерами данного опроса. Удивительно, но информация о тарифах на сайтах YOTA и MTS отсутствует на главной странице, а у сотового оператора Билайн предоставлены не все тарифы на главной странице, возможно это и повлияло на низкий процент голосования за оператора БИЛАЙН в отличии от ТЕЛЕ2 и МЕГАФОН.

Рисунок 19 – Статистические данные опроса

На рисунке 19 отображена диаграмма с результатами на ответы пользователей на вопрос: «На каком сайте сотового оператора Вам проще найти информацию о тарифах?». В фаворитах оказались 2 сотовых оператора, это ТЕЛЕ2 и МЕГАФОН. Аутсайдерами оказались YOTA, МТС и БИЛАЙН. На диаграмме заметно, что сайт БИЛАЙН превосходит сотового оператора МТС, хотя, на сайте МТС в активной карте скроллинга не присутствуют значения о тарифах. На сайте сотового оператора БИЛАН присутствуют не все тарифы, а только выше ценового сегмента, и чтобы найти тарифы «конкурентные» приходится поискать их по сайту, что возможно усложняет пользователю их нахождение.

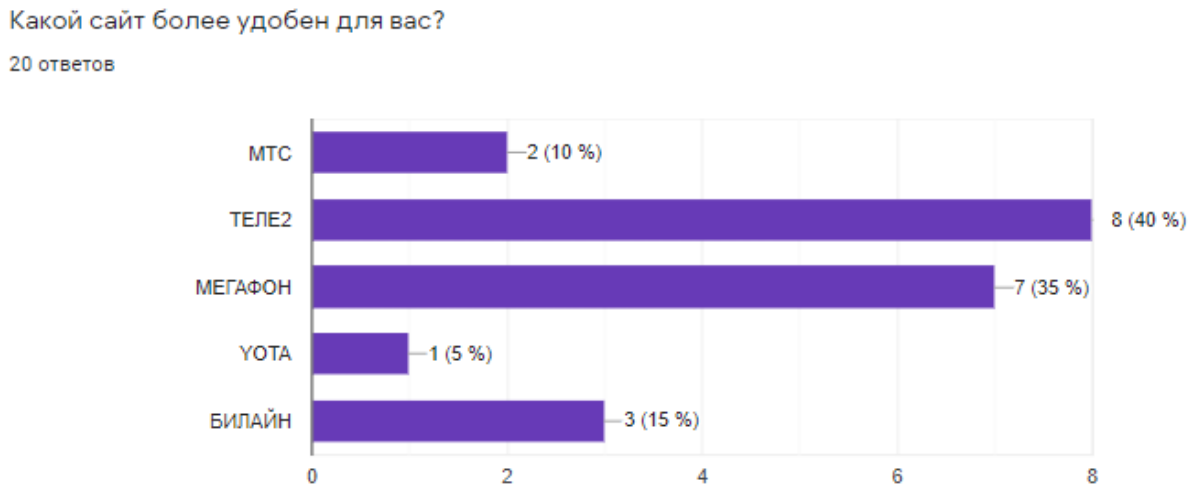

Рисунок 20 – Статистические данные опроса

На рисунке 20 отображена диаграмма с результатами на ответы пользователей на вопрос «Какой сайт более удобен для вас?». Фаворитами данного опроса оказались МТС, Теле2, а также, с большим отставанием от них, МЕГАФОН. Аутсайдерами оказались YOTA и МТС.

Рисунок 21 – Статистические данные опроса

На рисунке 21 отображена диаграмма с результатами на ответы пользователей на вопрос: «Какой дизайн сайта вам нравится больше всего?». Фаворитами данного опроса оказался Теле2. Примерно поровну набрали сотовые операторы YOTA, БИЛАЙН, МТС и МЕГАФОН.

Прежде чем сгруппировать данные и привести их к общему виду, необходимо определиться с метрикой для принятия гипотезы. Установим условие для метрики: «Если больше 70% подтвердят удобство интерфейса, на сайтах которых наличие тарифов присутствует в активной карте экрана, гипотезу считаем подтверждённой.»[13].

Что бы сгруппировать данные необходимо воспользоваться формулами математической статистики для обработки статистической информации и для получения научных и практических выводов.

Группировка – это процесс образования однородных групп на основе расчленения статистической совокупности на части или объединения изучаемых единиц в частные совокупности по существенным признакам. На рисунке 22 представлена таблица с группированными данными опроса.

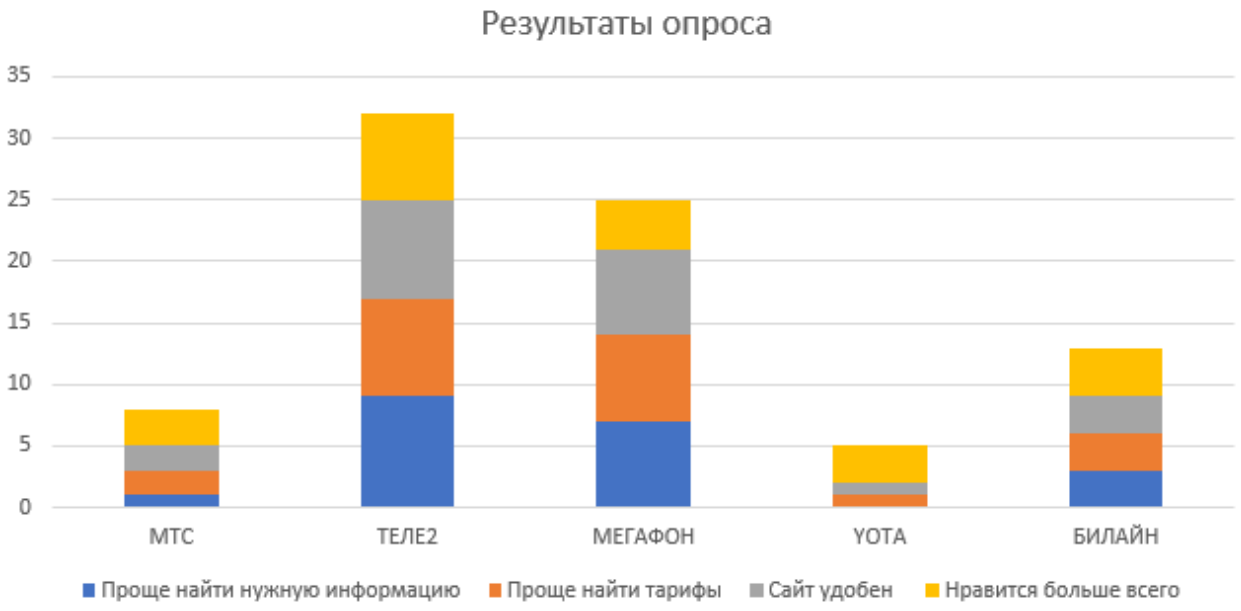

Рисунок 22 – Сгруппированные данные опроса

На диаграмме видно, что данные по вопросу: «Проще найти тарифы» соотносятся с общими данными почти в одинаковом соотношении. Также можно заметить, что у операторов, у которых присутствует какая-либо информация о тарифах в активной карте скроллинга, находятся в топе опроса. Также необходимо выразить данные в процентном соотношении для принятия и опровержении гипотезы.

Чтобы выразить данные в процентном соотношении воспользуемся формулой 1:

$$P = \frac{B}{A} * 100\% \quad (1)$$

где P - количество процентов, B и A - отношение чисел по которым нужно найти процент.

Результаты вычислений данной формулы изображены на рисунке ниже.

	Прост	Нашёл Тарифы	Удобен	Нравится	Общий процент
МТС	5,00%	9,52%	10,53%	14,29%	<70%
YOTA	0,00%	4,76%	5,26%	14,29%	
ТЕЛЕ2	45,00%	38,10%	36,84%	33,33%	>70%
МЕГАФОН	35,00%	33,33%	31,58%	19,05%	
БИЛАЙН	15,00%	14,29%	15,79%	19,05%	

Рисунок 23 – Данные в процентном соотношении

Общий процент высчитывался по формуле 2:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i = \frac{(X_1 + X_2 + X_3 + \cdots + X_n)}{n} \quad (2)$$

где N - количество параметров X - значение параметров.

Также, более 70% проголосовавших отдали свои голоса за операторов, у которых в активной части экрана присутствует информация о тарифах. Гипотезу считаем подтверждённой.

В данном разделе была не опровергнута гипотеза о том, что наличие тарифов в активной карте скроллинга позволяет упростить пользователю нахождению тарифов, а значит и улучшить эффективность пользовательского интерфейса сайта, а значит решение использования тарифов в активной части скроллинга принимаем для интерфейса биллинговой системы малого сотового оператора связи.

2.2 Построение модели оценки пользовательского интерфейса

Любой элемент пользовательского интерфейса можно представить в виде математической модели, а именно графа конечного автомата, вершины которого являются состояниями интерфейса, а ребра - переходы между ними[12].

Прежде чем построить математическую модель оценки пользовательского интерфейса, были проведены опросы оценки графического интерфейса и его функционала, в котором были определены фавориты и аутсайдеры соответственно. Для построение модели оценки пользовательского интерфейса возьмем одного фаворита и одного аутсайдера среди данных опросов, а именно: Мегафон и МТС соответственно.

Для построение модели оценки пользовательского интерфейса, необходимо провести анализ пользовательского интерфейса сайтов выбранных сотовых операторов, чтобы определить количество шагов, для достижение цели пользователей «смены тарифов».

Основания мысль заключается в том, что чем ближе значения модели биллинг системы будет ближе к значению модели фаворита, тем эффективнее и привлекательнее для пользователя построен интерфейс.

Под значением модели будет принимать конечное число шагов, которое необходимо совершить для достижения одной определённой цели.

Для математического моделирования элементов интерфейса был выбран конечный автомат, где Q - множество состояний, E – множество переходов, S - начальное состояние автомата.

Построим граф конечного автомата для аутсайдера сайта МТС, данный граф изображён на рисунке 24.

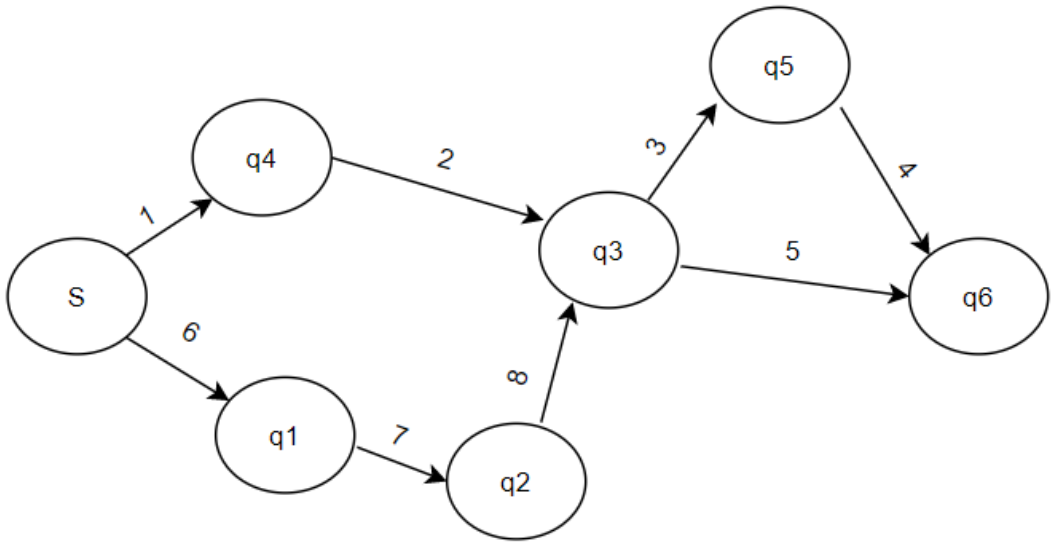

Рисунок 24 – Граф конечного автомата для сайта МТС

Состояния автомата:

S – стартовая страница сайта,

q_1 - поля для авторизации через e-mail и пароль,

q_2 - пользователь вошел в “личный кабинет”,

q_3 – окно экран выбора тарифа,

q_4 – выпадающий список,

q_5 – окно выбора тарифа,

q_6 – подключение тарифа.

Переходы автомата:

1 – нажатие кнопки связь,

2 – тарифы и подписки,

3 – подробнее,

4 – подключить,

5 – перейти,

6 – войти,

М.7 – бургер меню,

М.8 – тарифы и подписки.

Таблица 1 - Таблица переходов автомата поиска тарифов на сайте МТС.

	S	**q1**	**q2**	**q3**	**q4**	**q5**	**q6**
1	q4						
2					q3		
3				q5			
4						q6	
5				q6			
6	q1						
7		q2					
8			q3				

Количество переходов для достижении цели смены тарифа соответственно равно: 1 – 2 – 3 – 4, либо 6 – 7 – 8 – 5, наименьшее количество переходов четыре.

Построим граф конечного автомата для фаворита сайта Мегафон, данный граф изображён на рисунке 25.

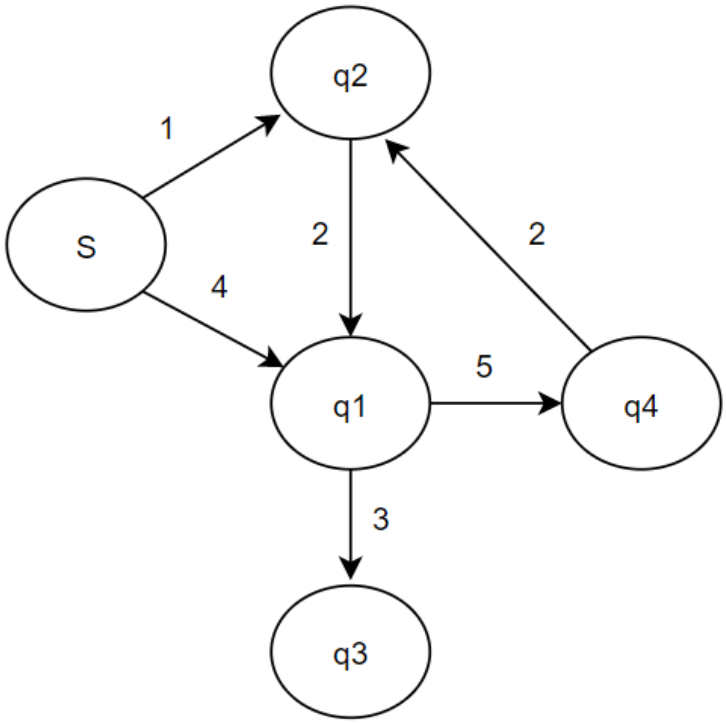

Рисунок 25 – Граф конечного автомата для сайта Мегафон

Состояния автомата:

S – стартовая страница сайта,

q_1 - поля для авторизации через e-mail и пароль,

q_2 - пользователь вошел в “личный кабинет”,

q_3 – список тарифов,

q_4 – список тарифов.

Переходы автомата:

1 – нажатие SSP,

2 – нажатие войти,

3 – изменить тариф,

4 – сразу изменить тариф без авторизации,

5 – изменение тарифа из личного кабинета.

Таблица 2 - Таблица переходов автомата поиска тарифов на сайте Мегафон.

	S	**q1**	**q2**	**q3**	**q4**
1	q2		q1		
2					q2
3		q3			
4	q1				
5		q4			

Количество переходов для достижении цели смены тарифа соответственно равно: 1 – 2 – 3, либо 4 – 5 – 2 – 2 – 3, либо 4 – 3, наименьшее количество переходов два.

Построим граф конечного автомата для разрабатываемого веб приложение биллинг системы, данный граф изображён на рисунке 2.4.

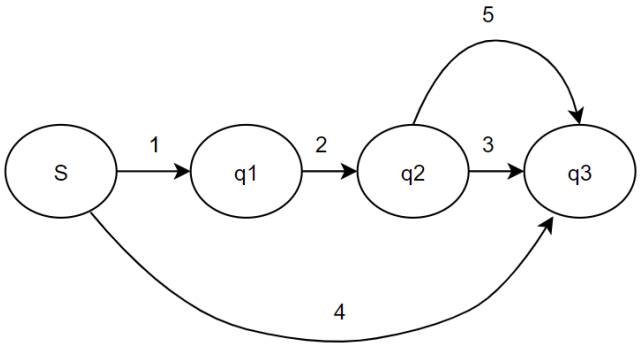

Рисунок 26 – Граф конечного автомата

Состояния автомата:

S – стартовая страница сайта,

q_1 - поля для авторизации через e-mail и пароль,

q_2 - пользователь вошел в “личный кабинет”,

q_3 – окно экран выбора тарифа.

Переходы автомата:

1 – нажатие SSP,

2 – нажатие войти,

3 – изменить тариф,

4 – сразу изменить тариф без авторизации,

5 – изменение тарифа из личного кабинета.

Таблица 3 - Таблица переходов автомата биллинг системы.

	S	**q1**	**q2**	**q3**
1	q1			
2		q2		
3			q3	
4	q3			
5			q3	

Количество переходов для достижении цели смены тарифа соответственно равно: 1 – 2 – 3, либо 1 – 2 – 5, либо 4, наименьшее количество переходов один.

Количество переходов в решении биллинг системы близко к значению сотового оператора Мегафон, а также на графе 24 заметно, что количество переходов с сотовым оператором МТС находится в существенном диапазоне, а именно имеет большую разницу.

В данном разделе была построена и использована математическая модель для оценки привлекательности пользовательского интерфейса.

2.3 Анализ инструментов разработки

Для того что бы приступить к разработке биллинговой системы необходимо определиться с выбором средств разработки IDE (Интегрированной средой разработки). Выбор IDE является важным этапом в разработке ПО, потому что IDE позволяет быстрее, качественнее писать и менять код. На сегодняшний день существует огромное количество сред разработки программного обеспечения, каждая среда имеет свой набор преимуществ и недостатков. Важным критерием выбора IDE служит наличие автоматической загрузки необходимых библиотек, нахождение и исправление ошибок в коде, не выходя за пределы IDE, данный критерий продемонстрирован на рисунке 29 в среде Intellij idea, а также наличие автозаполнение «понимания контекста», понимание контекста продемонстрировано на рисунке 27 в среде Intellij idea.

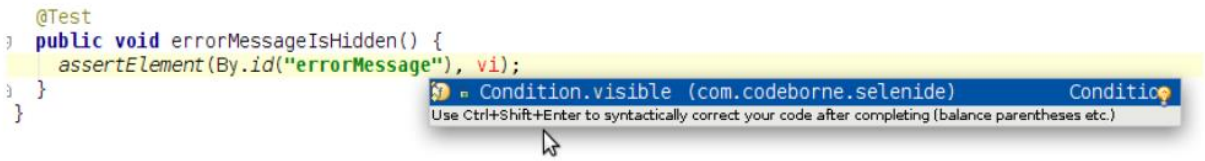

Рисунок 27 – Понимание контекста IDE

На рисунке 27 среда IDEA понимает, что метод assertElement хочет получить вторым параметром объект класса Condition, а в этом классе как раз есть статистическая переменная типа Condution с именем visible, соответственно IDEA предлагает единственный возможный вариант. Так же благодаря пониманию контекста IDE может показать целевую документацию для текущего ожидаемого метода.

Непонимание контекста продемонстрированно на рисунке 28, на примере ide Eclepse.

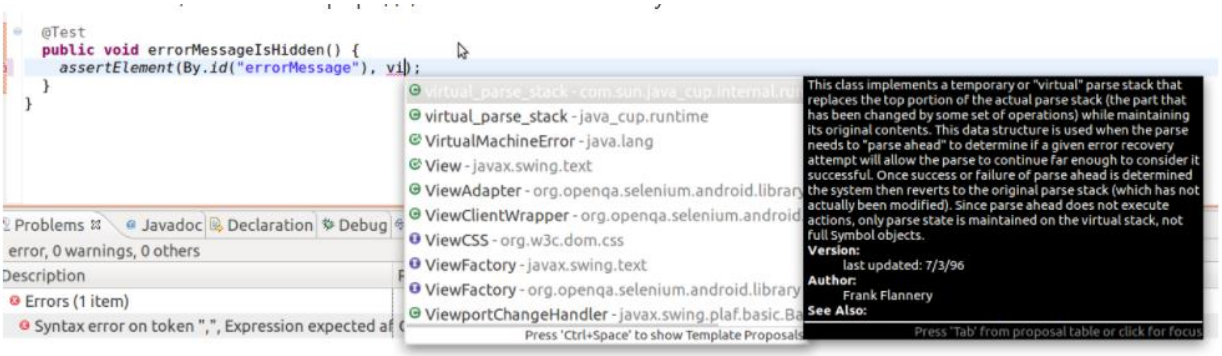

Рисунок 28 – Не понимание контекста

На рисунке 28 продемонстрирована ситуация непонимания контекста средой Eclipse, среда не знает, что курсор находится на месте второго параметра метода assertElenent, поэтому при нажатии на CTRL+SPACE, Eclipse показывает все что начинается на буквы «vi».

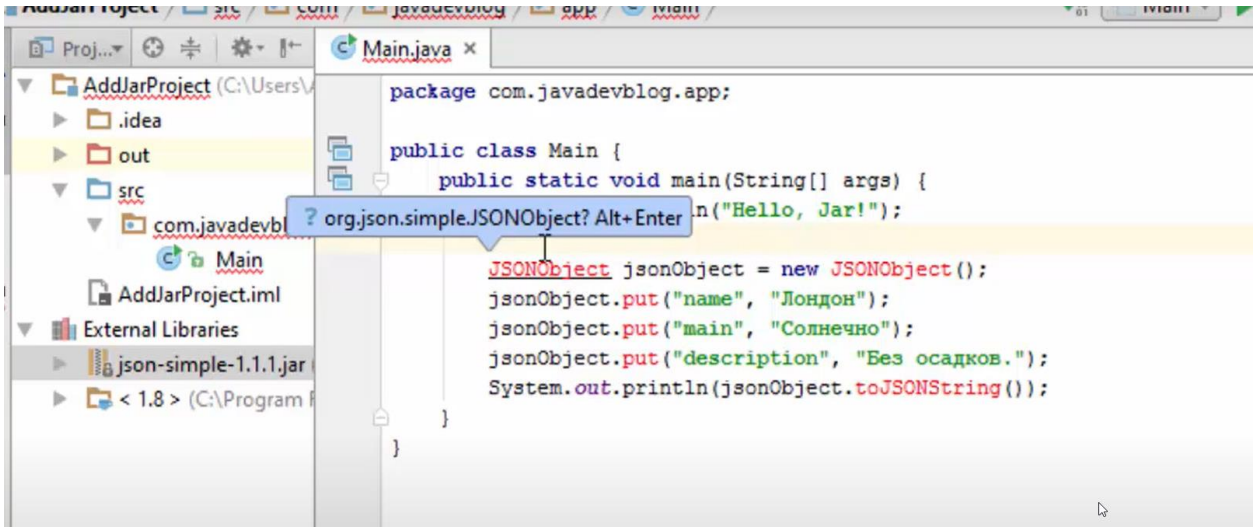

Рисунок 29 – Автоматическое исправление ошибок в коде

На рисунке 29 продемонстрирована ситуация в котором по мнению idea кроется ошибка. Красным цветом idea выделяет синтаксис, который, по её мнению, содержит ошибку, при наведении мыши всплывает подсказка о

решении данной проблемы, при нажатии сочетаний клавиш Alt+Enter idea исправит ошибку в коде.

После перечисления всех достоинств среды IDEA было принято решение разрабатывать серверную часть с использование среды intellij idea.

Для разработки клиентской части было решено использовать IDE Visual Studio code, так как в данной среда разработки присутствует огромное количество плагинов для комфортной веб разработки.

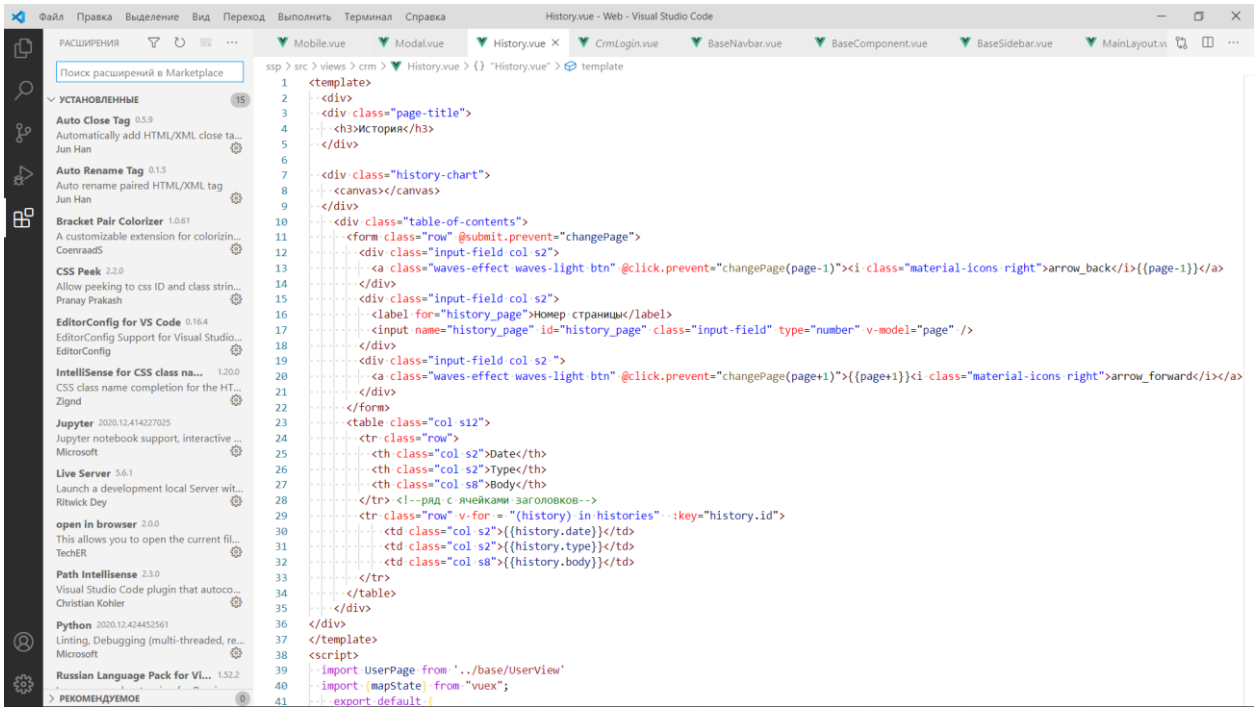

Рисунок 30 – Среда visual studio code

На рисунке 30 изображена страница установленных плагинов в среде разработки visual studio code.

В данном разделе был проведён анализ инструментов разработки для реализации биллинг системы. В процессе анализа были выделены основные преимущества и недостатки сред разработки.

2.4 Разработка серверной части приложения

После того как была выбрана среда разработки необходимо приступить к написанию серверной части приложения. Бэкенд (в переводе с английского «Backend») - программно-аппаратная часть сервиса, которая отвечает за осуществление функционирования внутренней части веб-сайта. Бэкенд в себя включает такие компоненты как DB, Proxy, Validator.

DB (data base) – база данных приложения она необходима для хранение информации.

Proxy – компонента необходимая для выполнения основной логики БС, например, замены тарифа «А» пользователя на базовый при удаления тарифа «А» из системы, а также все сложные действия разбиваются на простые, базовые и отправляются в виде сообщений в BD через Rabbit.

Validator – необходим для создания сессий и проверок входящих данных, именно с ней связанны веб компоненты АТС и CRM.

Серверная часть приложения будет взаимодействовать с веб частью с помощью запросов REST API.

Наглядное взаимодействие данных компонентов продемонстрированно на рисунке 1.

REST API – является одним из способов взаимодействия клиента с сервером, в свою очередь клиент может представляется как веб версия сайта, мобильная версия сайта или мобильное приложение. Все запросы с разных платформ от ios, andoid, и любой другой технологии, будут приходить на один сервер, который читает их и выдаёт соответствующий запросу ответ.

Запросы бывают нескольких типов:

– get,

– post,

– put,

– delete,

– head,

– connect,

– options,

– trace,

– patch.

Метод «get» (в переводе с английского «get») - запрашивает у сервера определенный ресурс. Результатом является только набор данных, который этот метод возвращает на сервер.

Метод «post» (в переводе с англ. «Send») - отправляет на сервер конкретную сущность, которую нужно добавить в базу данных или обработать. Результатом может быть либо состояние об успешном выполнении операции, либо сущность, соответствующая успешной обработке объекта.

Метод «put» (в переводе с англ. «Положить») - отправляет на сервер сущность или некоторые ее поля, например как и метод «post», при этом внутри него также должен быть параметр, который будет определяться сущностью, требующий обновлений (при условии, что она существует). Этот метод также поддерживает параметризацию, как и у метода "get".

Метод «delete» (в переводе с англ. «Удалить») - отправляет на сервер набор параметров или сущность для ее определения, которые необходимо удалить (например, в контексте баз данных).

После выполнения запроса со стороны клиенты приходит ответ с сервера в виде кода состояния. Данные коды сопровождаются словесным пояснением предоставленными ниже, данные кодов сгруппированы по причине их возникновения:

а) информационные:

- 100 Continue,

- 102 Processing.

б) успешные:

- 200 Ok,
- 201 Created,
- 202 Accepted,
- 204 No content.

в) перенаправление:

- 300 Multiple choices,
- 301 Moved permanently,
- 302 Moved temporarily,
- 304 Not modified,
- 305 Use proxy,
- 307 Temporary redirect,
- 308 Permanent redirect.

г) ошибка клиента:

- 400 Bad request,
- 401 Unauthorized,
- 403 Forbidden,
- 404 Not found,
- 405 Method not allowed.

д) ошибка сервера:

- 500 Internal server error,
- 502 Bad gateway,
- 503 Service unavailable,
- 504 Gateway timeout.

Фрагменты классов обрабатывающие запросы API от клиента представлены на рисунке ниже

```
import ...

/*@RestController*/
@Service
public class TariffController {

    @Autowired
    private RabbitMQSender rabbitMQSender;

    @Autowired
    OperationsService operationsService;

    Logger LOG = LoggerFactory.getLogger(TariffController.class);

    public Boolean isTariffExists(String tariffName) {
        return operationsService.request("/getTariffByName/?name=" + tariffName, HttpMethod.GET, Tariff.class) != null;
    }

    public Boolean createTariff(CollectedTariff tariff) throws JsonProcessingException {...}

    public CollectedTariff getTariff(String tariffName) {
        if (isTariffExists(tariffName)) {
            try {
                final TariffCall tariffCall = operationsService.request("/getTariffCallByName/?name=" + tariffName, HttpMethod.GET, TariffCall.class);
                final TariffInternet tariffInternet = operationsService.request("/getTariffInternetByName/?name=" + tariffName, HttpMethod.GET, TariffInternet.class);
                final TariffSms tariffSms = operationsService.request("/getTariffSmsByName/?name=" + tariffName, HttpMethod.GET, TariffSms.class);
                return new CollectedTariff(tariffName, tariffCall, tariffInternet, tariffSms);
            } catch (Exception e) {
                LOG.error("getTariff failed", e);
            }
        }
        return null;
    }

    public List<Tariff> getAllBaseTariffs() {
        return operationsService.requestList("/getAllTariff", HttpMethod.GET, Tariff[].class);
    }

    public List<CollectedTariff> getAllCollectedTariff() {...}
    public List<Map> getAllCollectedTariffAsMapList() {...}
    public Boolean updateTariff(Tariff newTariff) {...}
    public Boolean deleteTariff(String name) {...}
}
```

Рисунок 31 – Реализация класса TariffController

TariffController является основным классом для взаимодействия с тарифами. Он позволяет выполнять основные действия такие как: удалить, обновить и изменить тариф.

На рисунке 32 продемонстрирован класс AccountController. С помощью этого класса можно получить информацию о состоянии основных параметров аккаунта, таких как текущий остаток по минутам, смс и мегабайтам.

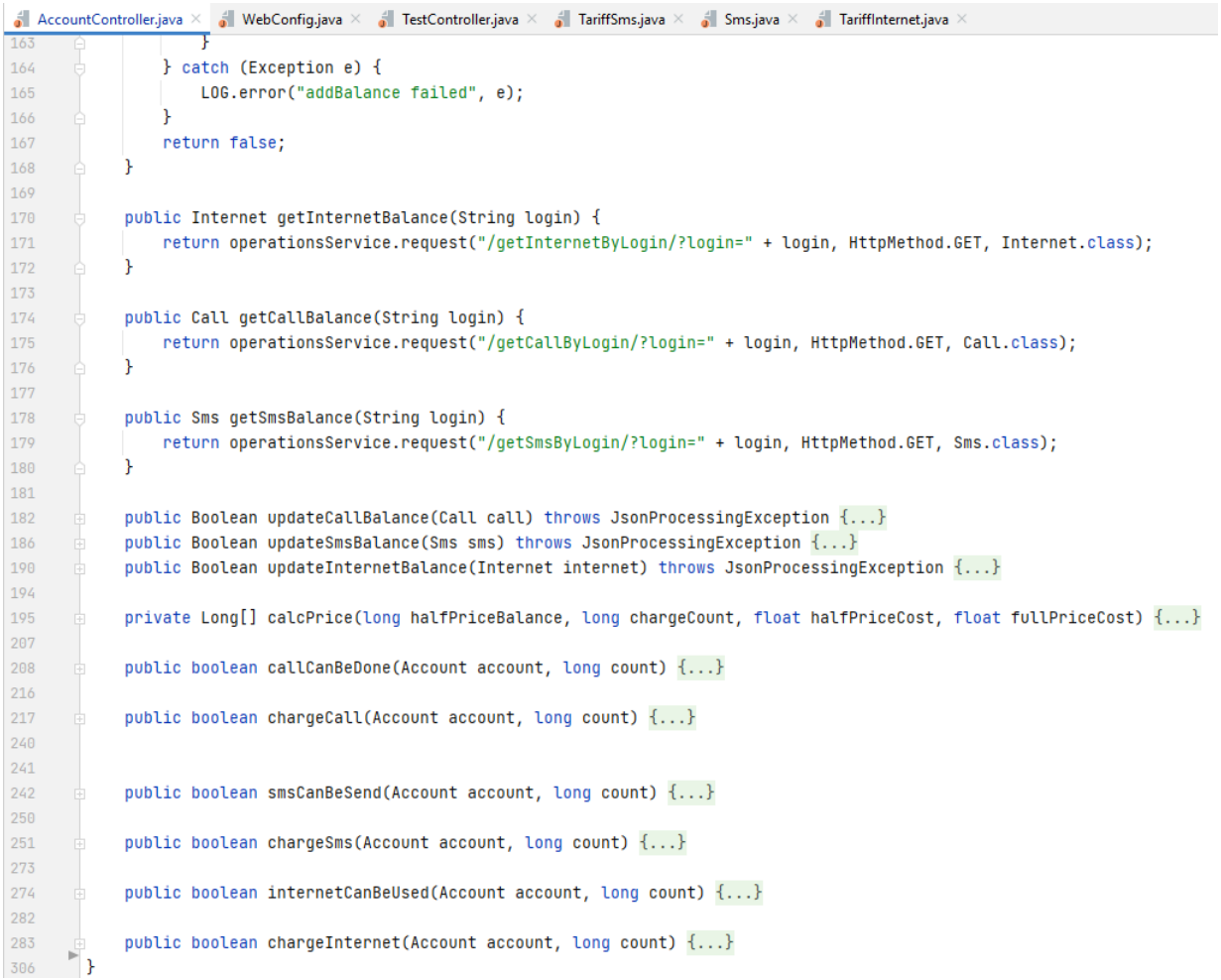

Рисунок 32 – Реализация класса AccountController

Так же для получения истории трат по тарифу был реализован класс HistoryController. С помощью данного класса можно получить историю расходов у текущего аккаунта, данный класс принимает в запросе номер страницы, он изображён на рисунке 33.

```java
package com.edunetcracker.billingservice.ProxyProxy.proxy;

import ...

/*@RestController*/
@Service
public class HistoryController {

    @Autowired
    OperationsService operationsService;

    public List<History> showHistory(Integer page) {
        return operationsService.requestList("/getHistory?page=" + page, HttpMethod.GET, History[].class);
    }

}
```

Рисунок 33 – Реализация класса HistoryController

Также сервер от клиента в каждом запросе принимает токен.

Токен – ключ который используется для идентификации владельца и безопасного удалённого доступа к информации, он выдаётся после успешной авторизации пользователю.

После обработки запроса сервер вернёт ответ, если же запрос предполагает от сервера ответа. Ответ от сервера отправляется в формате json и формируется по структуре, которую ожидает получить клиент, пример ответа изображён на рисунке 34.

```
{ “students” :
  [
      {“id":1, "name":"Adnan Sohail"},
      {“id":2, "name":"Irfan Razzaq"}
  ]
}
```

Рисунок 34 – json ответ от сервера

Json – формат данных для клиента, которые содержат в себе информацию в структурированном виде. Данные могут содержать: числовые, строковые, болевые значения, объекты и списки. На рисунке 34 изображена

группа student в которой содержатся уникальный унификатор и имя студента.

На стороне клиента происходит разбиение json ответа и подставление данных туда, где они должны располагаться в интерфейсе пользователя.

После реализации основных классов, которые обрабатывают запросы пользователя необходимо приступить к написанию клиентской части приложения.

В данном разделе была разработана основная серверная часть приложения для мобильного сотового оператора связи.

2.5 Разработка клиентской части приложения с использованием технологии VUE.JS

После того как было разработана серверная часть необходимо перейти к созданию веб интерфейса, с которым будет взаимодействовать клиент.

Данный раздел содержит в себе разработку интерфейса приложения с помощью технологии VUE.JS. Преимущество данной технологии были описаны в разделе 1.2.

Первое что необходимо сделать это создать окно авторизации пользователя. Окно авторизации представляет в себе модальное окно, в котором присутствуют два поля ввода, каждое из которых отвечает за логин и пароль соответственно. Так как в данной биллинг системе присутствуют сразу 3 веб модуля, это: SRM, SSP, ATC, модальное окно должно присутствовать в каждом из них.

Чтобы приступить к разработке на технологии VUE.JS необходимо ознакомиться с жизненным циклом данной технологии. Понятие жизненный цикл заключается в вызове определённых методов при загрузки той или иной страницы. Данные методы можно использовать для чистки временных

данных или на оборот для их получения с помощью API. Последовательность вызова этих методов продемонстрированно на рисунке 35.

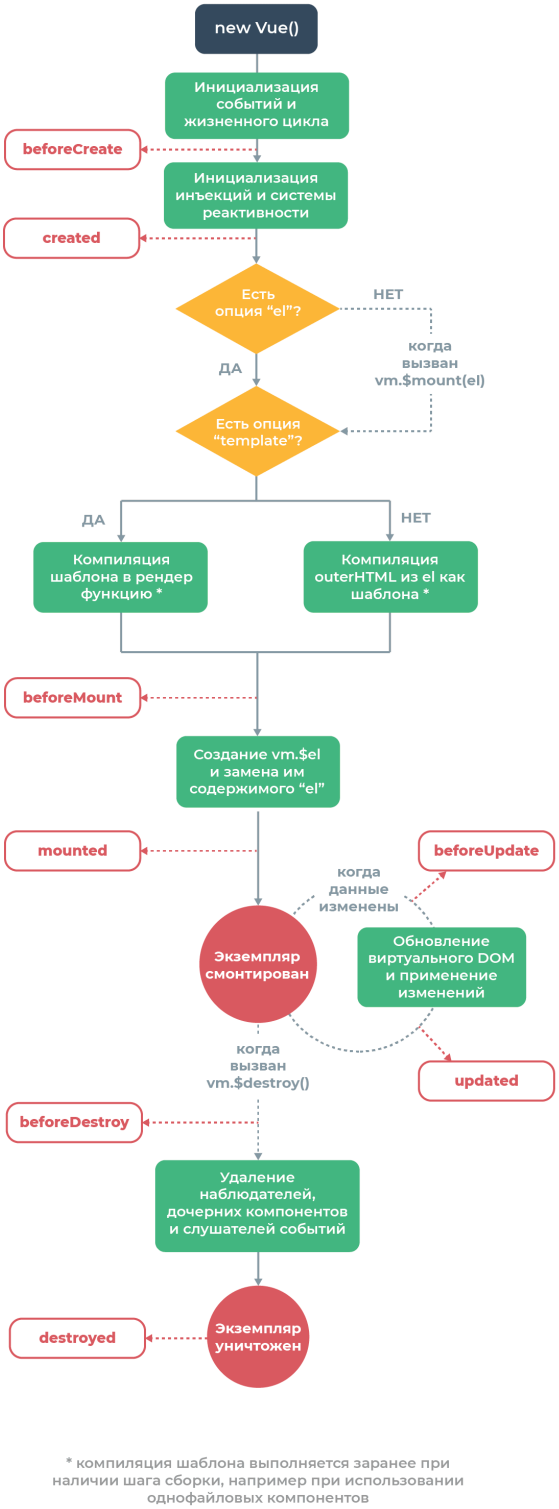

Рисунок 35 – Хуки жизненного цикла VUE

Что бы уменьшить нагрузку на сервер и увеличить скорость загрузки страниц, было принято использовать особенность vue.js динамического отображение контента, по принципу «скачай и пользуйся». Для этого шапку и бургер меню страницы сделаем основными, чтобы в дальнейшем подгрузка страницы осуществлялась не целиком, а только тех элементов, которых не хватает, для этого на домашней странице оставим соответствующий тег component благодаря которому другие элементы смогут погружаться на данной странице. Данный программный код изображён на рисунке 36.

```
<template>
    <div id="app-container">
        <div id="app" v-if="appInitialized">
            <component :is="layout"> <!-- @контент меняется от условия layout()@ -->
                <!-- <router-view/> @контент меняется от адресной строки@ Наличие имени у <router-view>
            </component>
        </div>
        <div id="appLoading" v-if="!appInitialized && appInitializationInProgress">
            <h3 class="">Приложение загружается</h3>
        </div>
        <div id="appLoadFailed" v-if="!appInitialized && !appInitializationInProgress">
            <h3 class="">Не удалось загрузить приложение</h3>
        </div>
    </div>
</template>
```

Рисунок 36 – Программный код

Для отображения модального окна авторизации воспользуемся хуком munted() который генерируется после загрузки страницы. Который в свою очередь вызывает метод loginCheck. Logincheck отслеживает состояния поля email. Данный пример изображён на рисунке 37.

```vue
<script>
  import BasePage from "../../components/base/BaseComponent"
  import {mapState} from 'vuex'
  import {email, minLength, required} from 'vuelidate/lib/validators'

  export default {
    extends: BasePage,
    name: 'login', //имя данной странице
    data: () => ({
      title: 'Личный кабинет',
      email: '',
      password: '',
      loginFailed: false,
      loginStarted: false,
    }),
    computed: mapState({
      login_done: state => state.CURRENT_USER != null,
    }),
    mounted() {
      this.$store.dispatch("WAIT_INITIALIZATION").then(() => {
        this.email = this.$store.state.DEFAULT_USER_NAME;
        this.password = this.$store.state.DEFAULT_USER_PASS;
        this.loginCheck();
      })

    },
    validations: {
      email: {email, required}, /* required - пустое поле не принимаем*/
      password: {required, minLength: minLength(6)}
    },
    methods: {
      loginCheck() {
        if(this.login_done){
          this.goToLastPage();
        }else {
          const that = this;
          setTimeout(() => that.loginCheck(), 1000);
        }
      },
```

Рисунок 37 – Программный код модального окна

Во vue.js разметка html, стили CSS и код JavaScript могут находиться на одной странице, что упрощает написание кода, пример показан на рисунке 38.

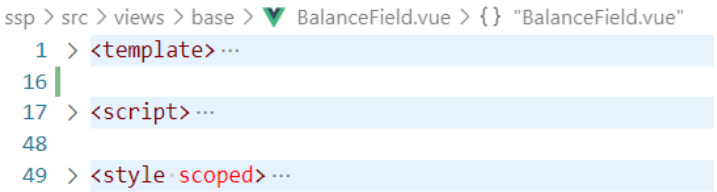

Рисунок 38 – Размещение разных типовых подходов

Что бы осуществить проверка валидности входных данных, а именно логина и пароля, необходимо подключить библиотеку validator, её можно

легко подключить с помощью интерфейса vue.js или набрать в консоли npm install validator. После установки библиотеки довольно просто её использовать в коде html. Пример использования изображён на рисунке 39.

```
<div class="input-field">
    <input id="email" type="text" v-model.trim="email"
        :class="{invalid: ($v.email.$dirty && !$v.email.required)||($v.email.$dirty && !$v.email.email)}">
    <label for="email">Email</label>
    <small class="helper-text invalid"
        v-if="($v.email.$dirty && !$v.email.required)||($v.email.$dirty && !$v.password.minLength)">Email</small>
</div>
<div class="input-field">
    <input id="password" type="password" v-model.trim="password"
        :class="{invalid: ($v.password.$dirty && !$v.password.required)||($v.password.$dirty && !$v.password.minLength)}">
    <label for="password">Пароль</label>
    <small class="helper-text invalid"
        v-if="($v.password.$dirty && !$v.password.required)||($v.password.$dirty && !$v.password.minLength)">Минимум
        {{$v.password.$params.minLength.min}} символов.</small>
</div>
</div>
```

Рисунок 39 – Проверка валидности входных данных

Внешний вид страницы модального окна регистрации изображён на рисунке 40.

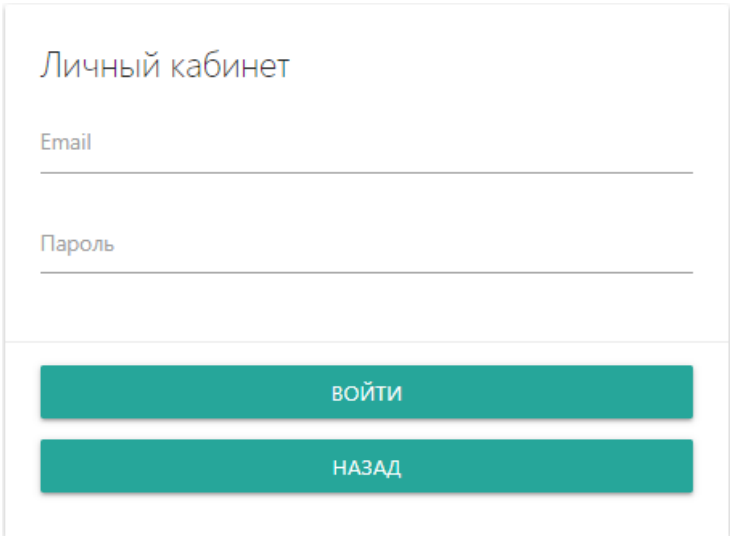

Рисунок 40 - Окно авторизации

После формирования модального окна регистрации приступим к отображению тарифов в личном кабинете пользователя. В разделе 2.1 была

принята гипотеза о наличии тарифов в активной карте экрана, следовательно размещение тарифов в биллинг системе будет располагаться так чтобы не противоречить гипотезе. Пример дизайна страницы отображения тарифов изображен на рисунке 41.

Рисунок 41 – Доступные тарифы для пользователя

На рисунке 41 изображены тарифы которые может выбрать пользователь, новые тарифы могу быть добавлены, изменены или удалены администраторами через модуль CRM. Данный функционал изображен на рисунке 42.

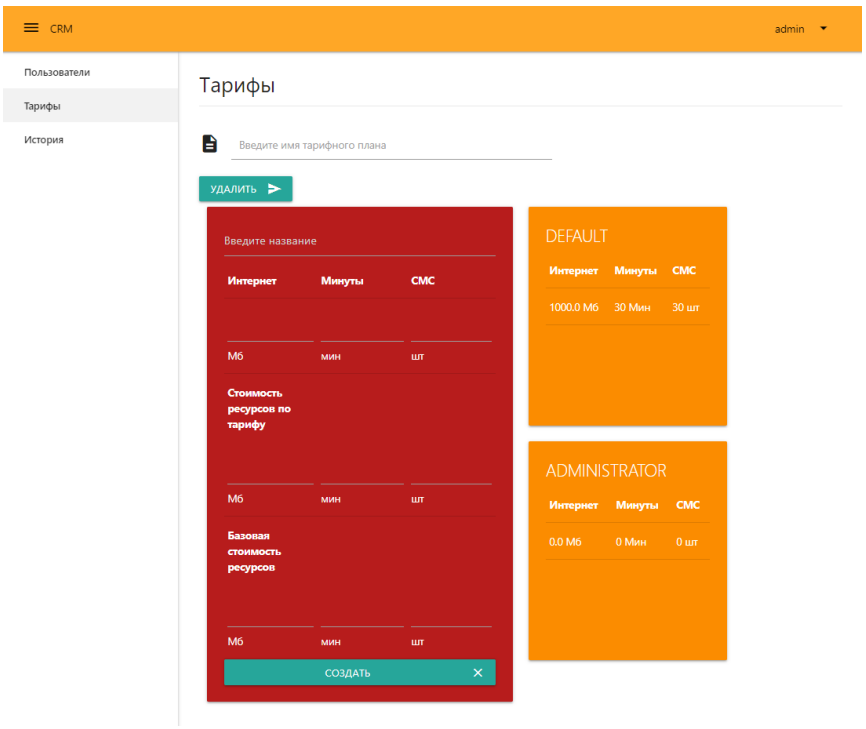

Рисунок 42 – Модуль CRM изменение тарифов

Домашняя страница личного кабинета у пользователя выглядит следующим образом, она изображена на рисунке 43.

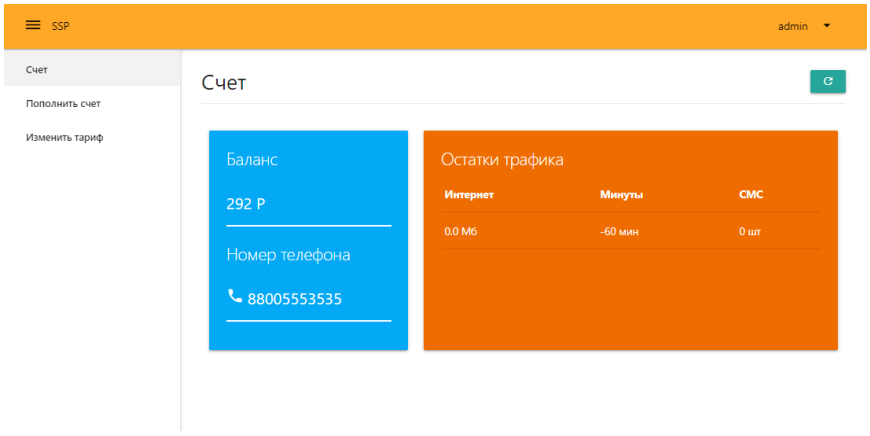

Рисунок 43– Домашняя страница пользователя компоненты SSP

Для того, чтобы получить данные о состоянии баланса, о количествах минут, мегабайт и смс необходимо обратиться к классу AccountControler. Для

этого на домашней странице укажем обращение к данному классу через хранилище state во vue.js.

Mobile.vue | Modal.vue | History.vue | Home.vue | App.vue | BalanceField.vue | LoginView.vue

ssp > src > views > ssp > Home.vue > {} "Home.vue" > template

```
<script>
  import UserPage from '../base/UserView'
  import {mapState} from "vuex";
export default {
  extends: UserPage,
  data: () => ({ loginRoute: '/ssp/login' }),
  computed: mapState({
    balance: state => state.CURRENT_USER!=null ? (parseFloat(state.CURRENT_USER.balance)).toFixed(0) : 0,
    minutes: state => state.CURRENT_USER!=null ? (parseFloat(state.CURRENT_USER.minutes)).toFixed(0) : 0,
    sms: state => state.CURRENT_USER!=null ? (parseFloat(state.CURRENT_USER.sms)).toFixed(0) : 0,
    internet: state => state.CURRENT_USER!=null ? (parseFloat(state.CURRENT_USER.internet)).toFixed(1) : 0,
    userPhone: state => state.CURRENT_USER!=null ? state.CURRENT_USER.telephone : ''
  }),
  mounted() {
    this.refreshData();
  },
  methods: {
    refreshData(){
      this.$store.dispatch("SYNC_CURRENT_USER");
    }
  }
}
</script>
```

Рисунок 44 – Получение данных для домашней страницы

Страница истории и её программный код изображен на рисунке 45 и 46 соответственно.

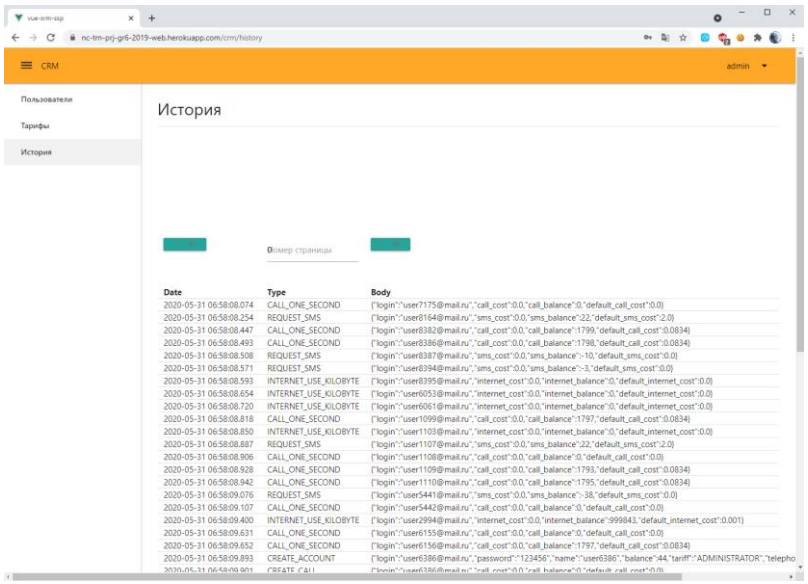

Рисунок 45 – Страница истории

Рисунок 46 – Программный код страницы истории

На странице истории пользователь получает информацию о любых действиях в системе. Чтобы при загрузке страницы отобразить данные которые принадлежат истории необходимо воспользоваться хуком VUE.JS – mounted() который хранит в себе метод LOAD_HISTORY который в свою очередь обращается к классу CRM на сервере по соответствующему адресу, в результате сервер вернёт json формат который и хранит данные которые и необходимо отобразить на странице.

```
LOAD_HISTORY: async (context) => {
    let compare = function(a, b) {
        if (a.id<b.id) {
            return -1;
        }
        if (a.id>b.id) {
            return 1;
        }
        // a должно быть равным b
        return 0;
    }
    if(context.state.STUB_MODE) {
        await context.commit("SET_ACTIONS_HISTORY", []);
    } else {
        try{
            if(context.state.TOKEN!=null){
                const historyResponse = await Vue.axios.get(`${context.state.VALIDATOR_URL}/showHistory/?token=${context.state.TOKEN}&page=${context.state.HISTORY_PAGE}`);
                if(historyResponse==null || historyResponse.data==null){
                    await context.commit("SET_ACTIONS_HISTORY",[]);
                }else{
                    let data = historyResponse.data;
                    data.sort(compare);
                    await context.commit("SET_ACTIONS_HISTORY", data);
                }
            }else{
                await context.commit("SET_ACTIONS_HISTORY", []);
            }

        }catch(e){
            await context.commit("SET_ACTIONS_HISTORY", []);
        }
    }
},
```

Рисунок 47 – Программный код метода LOAD_HISTORY

На рисунке 47 продемонстрирован метод LOAD_HISTORY который выполняет запрос к серверу для получения данных по истории, запрос содержит в себе значение ключа – токена, ключ на сервере идентифицирует пользователя тем самым сервер разрешает конкретному пользователю получить ответ.

Так же что бы эмулировать траты по ресурсам тарифов была создана страница АТС которая выполняет эту задачу. Вписав номер телефона в поле ввода «from the telephone» и нажав кнопку «НАЧАТЬ ЗВОНОК» у соответствующего пользователя произойдёт списание пакета минут, до тех пор пока пользователь в модальном окне не нажмёт завершить.

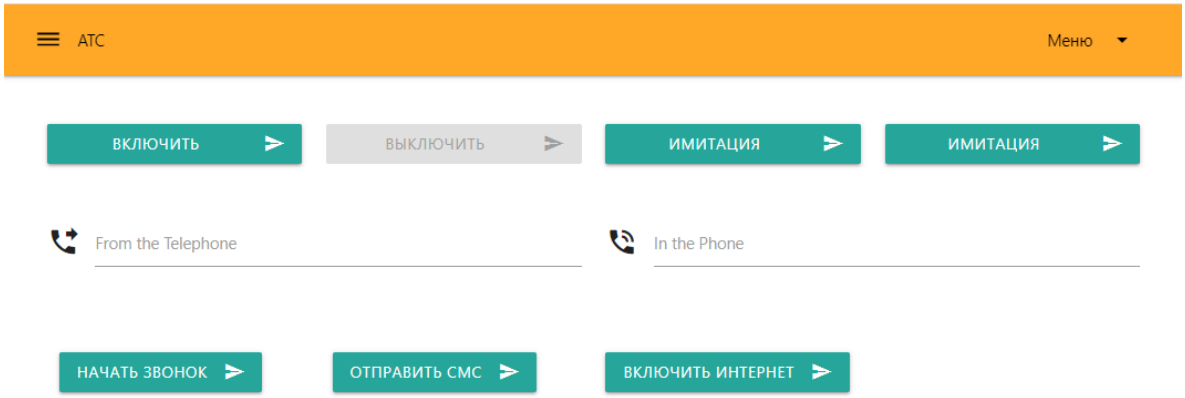

Рисунок 48 – Страница АТС

На рисунке 48 представлен пример страницы АТС. Стоит добавить, что каждая кнопка на данной странице также обращается на сервер по соответствующему url адресу, в котором сервер принимает и отправляет его соответствующему классу.

Удобным в использовании технологии VUE.JS также является его router. Router (от английского «маршрутизатор») – он позволяет задать страницам соответствующие url адреса, router продемонстрирован на рисунке 49.

```vue
import Vue from 'vue'
import VueRouter from 'vue-router'

Vue.use(VueRouter)
const routes = [
  {
    path: '',
    name: 'swith',
    meta: {layout: 'Main'},
    component: () => import('../views/Swith.vue') //главная страница
  },
  {
    path: '/ssp/login', //для адресной строки //при этом имени <router-view/> на главной странице App.vue будет менять контент
    name: 'sspLogin', //для себя имя
    meta: {layout: 'Main'}, //задаём лаяуты //свойства имя, имя
    component: () => import('../views/ssp/SspLogin') //импортируем контент
  },
  {
    path: '/crm/login',
    name: 'crmLogin',
    meta: {layout: 'Main'},
    component: () => import('../views/crm/CrmLogin')
  },
  {
    path: '/crm/tariffadmin',
    name: 'tariffadmin',
    meta: {layout: 'Main', auth: true},
    component: () => import('../views/crm/TariffAdmin.vue')
  },
  {
    path: '/crm',
    name: 'crmUser',
    meta: {layout: 'Main', auth: true},
    component: () => import('../views/crm/User.vue')
  },
  {
    path: '/crm/history',
    name: 'crmHistory',
    meta: {layout: 'Main', auth: true},
    component: () => import('../views/crm/History.vue')
  },
```

Рисунок 49 – Программный код маршрутизатора

Выводы по второй главе

В данной главе была принята гипотеза о том, что наличие у сотовых операторов тарифов в активной карте экрана позволяет увеличить эффективность пользовательского интерфейса. Была построена математическая модель для оценки привлекательности веб интерфейса. Также в данном разделе был проведён анализ сред разработки и сама разработка приложения, а именно серверная и внешняя её часть.

Глава 3 Оценка качества интерфейса

Для того, что бы убедится, что гипотеза верна, необходимо провести повторный анализ статистики сайта, так как данные статистик анализируемых сайтов сотовых операторов являются закрытыми. Данные, которые были получены в результате опроса и соответствующей ей гипотезу, примем за истину. Поэтому, в данной главе, будет рассмотрены варианты тестирования внедрённого функционала.

3.1 Ручное тестирование интерфейса

Целью данного раздела является ручное тестирование веб интерфейса приложения.

Ручное тестирование «manual testing» - является частью контроля качества, тестирования, при разработке ПО. Это процесс, выявления дефектов в тестируемом приложении, в нашем случаи тестирование выполняется разработчиком, в информативно технологических компаниях эту роль выполняет тестировщик по алгоритму тест плана, который ранее был написан до того, как была выполнена соответствующая задача. Цель ручного тестирования заключается в сравнения ожидаемого результата с действительным.

Достоинство и недостатки ручного тестирования в сравнении с автоматизированным тестированием:

- в ручном тестировании функционала можно получить гораздо больше информации о состоянии продукта, в отличии от автоматизированного, потому что автоматизированное тестирование не может дать результат о степени удобства и простоты использования интерфейса;

- В ручных тестах можно импровизировать, создавая различные сочетания действий, которые никогда не придут в голову пользователю, но могут быть совершены им случайно. Это позволяет охватить больше

функционала, потому что при тестировании тестировщик задаёт себе вопрос «а что, если?». Таким образом он находит необыкновенные способы взаимодействия с интерфейсом, даже если данного сценария тестирования не было.

После формулировки достоинств и недостатков ручного тестирования пользовательского интерфейса приступим к практической части тестирования, а именно написание сценарий тестирования.

Первое что необходимо проверить - модальное окно авторизации, а также корректного отображения тарифов на главной странице у пользователя. Чтобы их проверить необходимо выполнить набор последовательных действий на сайте их называют ещё «шагами». Запишем эти действия в последовательном порядке:

1) Зайти на сайт

2) Ввести в поле «Email» существующий Email пользователя, а именно: admin@mail.ru

3) Ввести в поле «Пароль» существующий пароль пользователя, а именно: 123456

4)Нажать на кнопку войти

5)Найти информацию о тарифах

После успешного выполнения данной последовательности действий, тест считается завершённым успешно, а значит функционал, который охватывает тест исправно работает. Данный тест захватывает сразу две проверки пользовательского интерфейса, а именно отображение тарифов на странице пользователя и модального окна авторизации.

В данном разделе произведён анализ ручного и автоматизированного тестирования, также было сформировано основное понятие ручного тестирования. Результатом данного раздела является сформированный сценарий тестирования.

3.2 Автоматизированное тестирование интерфейса

Целью данного раздела является автоматизированное тестирование веб интерфейса приложения.

Автоматизированное тестирование – процесс, при котором тестирование, а именно: шаги, запуск, выполнение, анализ и выдача результата производятся автоматически при помощи инструментов для автоматизированного тестирования.

На сегодняшний день существует множество технологий для написания тестов, а также языков программирования поддерживающие данные технологии.

Чтобы написать автоматизированный тест для проверки окна авторизации и отображение тарифа на главной странице пользователя, был выбран язык python и библиотека selenium.

Так как основной сценарий тестирования был сформирован в разделе 3.1 для написания автоматического теста формулировка сценария не потребуется.

Программный код автоматического теста продемонстрирован на рисунке 50.

```
from selenium import webdriver #автоматизация с использованием браузера
import time as t
# Получаем в переменную browser указатель на браузер
browser=webdriver.Firefox()
# Переходим на страницу, на которой нужно что-то сделать
browser.get('https://nc-trn-prj-gr6-2019-web.herokuapp.com/ssp/login')
#задержка в течение 1.5 секунды
t.sleep(4)
# Поиск по имени
username=browser.find_element_by_name("email")
username.send_keys('admin@mail.ru')
password=browser.find_element_by_name("password")
password.send_keys('123456')
#Получаем указатель на кнопку "Войти"
submit=browser.find_element_by_css_selector('submit')
#Нажимаем эту кнопку
submit.click()
#Получаем список тарифов
tarifs = submit=browser.find_elements_by_id('tarifs')
#Проверяем наличие тарифов
if tarifs>0:
    print("Тесты пройдены успешно!")
else:
    print("Тесты не пройдены!")
```

Рисунок 50 – Программный код автоматического теста

На рисунке 50 автоматический тест выполняет шаги в последовательном порядке. Что бы автоматический тест понимал с каким именно элементом в интерфейсе ему нужно взаимодействовать программе необходимо передать наименование этих элементов в разметке страницы. Так например в 10 строке кода передаётся название html тега который отвечает за поле ввода «email» пользователя.

Результат выполнения автоматического теста продемонстрирован на рисунке 51.

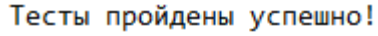

Рисунок 51 – Результат автоматического теста

В данном разделе было сформировано основное понятие автоматизированного тестирования, а также было проведено успешно тестирование пользовательского интерфейса.

Выводы по третьей главе

В данной главе был проведён анализ двух видов тестирования, а именно автоматизированного тестирования и ручного тестирования в котором были сформированы преимущества и недостатки каждого из видов. Был написан сценарий тестирования и автоматический тест, который проверяет модальное окно авторизации и отображение тарифов у пользователя. Тестирование пользовательского интерфейса было завершено успешно.

Заключение

Цель выпускной квалификационной работы заключалась в разработке привлекательного и эффективного веб-интерфейса путём построения модели оценки веб приложения для биллинг системы мобильного сотового оператора, разрабатываемого на основе технологии VUE.JS.

Для реализации поставленной цели были решены следующие задачи:

1. Сформированы требования к разрабатываемому приложению.

2. Проанализированы существующие веб- технологии.

3. Построены блок схемы и сценарий пользовательского интерфейса.

4.Приведен анализ существующих пользовательских интерфейсов, следствием которого была сформирована гипотеза.

5.Проанализированы статистические данные опроса, касающихся пользовательского интерфейса с помощью математической статистики, тем самым гипотеза была не опровергнута.

6. Была построена математическая модель оценки веб интерфейса.

7. Проведен анализ сред разработки пользовательского интерфейса.

8.Успешно выполнено внедрение и тестирование пользовательского интерфейса.

В качестве среды разработки веб-интерфейса использовалась среда программирования visual studio code 2021, а в качестве разработки серверной части использовалась среда intellij idea языком программирования java. Операционной системой для разрабатываемого приложения послужил windows 10. В результате выполнения данной бакалаврской работы, разработана биллинг система мобильного сотового оператора с практической реализацией веб интерфейса с применением технологии VUE.JS.

Цель бакалаврской работы в полной мере была достигнута.

Список используемой литературы

1. Бэрри, П. Изучаем программирование на Python / П. Бэрри ; [перевод с английского М.А. Райтман]. – Москва : Эксмо, 2017. – 624 с. : ил. ; Библиогр.: с. 182–228. – Заказ экз. – ISBN 978-5-699-98595-1. – Текст : непосредственный.

2. Белов, В.В. Проектирование информационных систем: Учебник / В.В. Белов. - М.: Академия, 2018. - 144 с

3. Зубов А.Н. Математика кодов аутентификации / А.Н. Зубов. - М.: Гелиос АРВ, 2014. 152 с.

4. Ишмухаметов Ш.Т. Математические основы защиты информации: учеб. пособие / Ш.Т. Ишмухаметов, Р.Г. Рубцов — Казань: Казанский федер. Унт, 2012. 138 с. 41

5. Константин М. Платежные технологии, системы и инструменты / М. Константин, 2015. – 282 с.

6. Куликов, С.С. Тестирование программного обеспечения. Базовый курс / С.С. Куликов. – Минск: Четыре Четверти, 2017. – 312 с.

7. Литвинская О. С. Основы теории передачи информации. Учебное пособие / О.С. Литвинская, Н.И. Чернышев. - М.: КноРус, 2015. 168 с.

8. Макаров А. Теория и практика хакерских атак / Макаров А. - М.: МИК, 2015. 384 с.

9. Неруш, Ю.М. Проектирование логистических систем: Учебник и практикум для бакалавриата и магистратуры / Ю.М. Неруш, С.А. Панов, А.Ю. Неруш. - Люберцы: Юрайт, 2016. – 422 с.

10. Перлова, О.Н. Проектирование и разработка информационных систем: Учебник / О.Н. Перлова, О.П. Ляпина, А.В. Гусева. - М.: Academia, 2017. - 416 с.

11. Перлова, О.Н. Проектирование и разработка информационных систем: Учебник / О.Н. Перлова. - М.: Академия, 2018. - 272 с.

12. Сакаева И.А. Математическое моделирование и анализ пользовательских интерфейсов: Научная статья / И.А. Сакаева – Казань: Казанский федеральный Университет, 2016г. URL: https://sci-article.ru/stat.php?i=1480765008 (дата обращения: 24.05.2021)

13. Хохлова Д. Тестирование гипотез через интервью целевой аудитории [Электронный ресурс] // URL: https://vc.ru/flood/8651-customer-dev-interview (дата обращения: 24.05.2021).

14. Шаньгин В. Информационная безопасность и защита информации / В.Ф. Шаньгин. - Москва: Гостехиздат, 2016. 470 с.

15. Яндекс.Метрика [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/%D0%AF%D0%BD%D0%B4%D0%B5%D0%BA%D1%81.%D0%9C%D0%B5%D1%82%D1%80%D0%B8%D0%BA%D0%B0 (дата обращения: 24.05.2021).

16. Alvarez C. Lean Customer Development/ Cindy Alvarez.2017. 240 с.

17. IntelliJ IDEA the Java IDE – JetBrains. [Электронный ресурс] // URL: https://www.jetbrains.com/idea/ (дата обращения: 24.05.2021).

18. Rajaraman V. Analysis and Design of Information Systems/ V. Rajaraman, 2018. 328 с.

19. Results for js web frameworks benchmark – round 8. [Электронный ресурс] // URL: https://stefankrause.net/js-frameworks-benchmark8/table.html (дата обращения: 16.06.2018).

20. Scott T. Systems Analysis and Design. / T. Scott, Harry J. Rosenblatt. 2016. 752 с.

21. William E. Software Testing and Continuous Quality Improvement. / E. William. 2017. 688 с.