**Реферат**

Утверд.

Н. контр.

Провер.

Разраб.

*Пацей Н.В.*

*Рыжанкова А.С*

*Куванова О.Р.*

*Шиман Д.В.*

Ф.И.О.

Подпись

Дата

1

*Реферат*

Лист

1

*БГТУ ДП 00.00.ПЗ*

Лит.

Листов

*БГТУ 74417058, 2020*

Дипломный проект содержит 60 страниц пояснительной записки, 10 таблиц, 26 рисунков, 14 источников литературы, 3 приложения.

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ ОФоРМЛЕНИЯ КРЕДИТНЫХ И ЛИЗИНГОВЫХ ПРОДУКТОВ И ЗАЯВОК, js, nODE.JS, СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ NOSQL, REACT, REDUX

Целью данного дипломного проекта является создание веб-приложения для формления кредитных и лизинговых заявок, создание самих кредитных и лизинговых продуктов, добавление банков и лизинговых компаний и точек продаж, а также в веб-приложении представлен кредитный и лизинговый калькулятор.

Пояснительная записка состоит из введения, шести разделов и заключения.

Во введении представлена общая информация о текущем состояние в сфере рассматриваемой проблемы.

В первом разделе представлены результаты аналитического обзора предметной области и технологий разработки.

Во втором разделе представлен процесс проектирования архитектуры системы.

В третьем разделе показан процесс реализации программного средства.

В четвертом разделе представлены результаты тестирования.

В пятом разделе представлены инструкции по установке приложения, а также руководство пользователя.

В шестом разделе представлены результаты себестоимости и отпускной цены разработанного программного средства.

В заключении представлены итоги дипломного проектирования и задачи, которые были решены в ходе разработки программного средства.

Abstract

Утверд.

Н. контр.

Провер.

Разраб.

*Пацей Н.В.*

*Рыжанкова А.С*

*Куванова О.Р.*

*Шиман Д.В.*

Ф.И.О.

Подпись

Дата

1

*Реферат*

Лист

1

*БГТУ ДП 00.00.ПЗ*

Лит.

Листов

*БГТУ 74417058, 2020*

The explanatory note of the diploma project contains 100 pages of explanatory note, 11 illustrations, 12 tables, 58 sources used, 5 appendices.

WEB APPLICATION FOR PROCESSING CREDIT AND LEASING PRODUCTS AND APPLICATIONS JS, NODE.JS, DATABASE MANAGEMENT SYSTEM NOSQL, REACT, REDUX

The purpose of the graduation project is to create a web-application for submitting credit and leasing applications, create credit and leasing products themselves, add banks and leasing companies and points of sale, and the web application provides a credit and leasing calculator.

The result of this work is a web platform for optimizing the operation of service halls of catering establishments. The user can use the app for web browsers.

The explanatory note consists of an introduction, seven sections, conclusion, and a list of references.

The introduction sets out the goal and objectives of the project.

In the first section, you will find a problem statement, a review of similar solutions, and a patent search.

The second section is devoted to the justification of technical techniques and tech-nologies.

The third section describes the main steps for designing a web platform.

The fourth section describes the details of implementing the server and client parts of the application.

The fifth section describes the main steps for testing the project for different com-ponents.

The sixth section describes the user's guide for understanding how to interact with the client application interface.

The seventh section provides a calculation of the economic parameters and cost of the software developed within the framework of the diploma project.

In conclusion, the results of the diploma design and the tasks that were solved dur-ing the development of the project are presented.

**Содержание**

ФИО

Подпись

Дата

Лист

1

ДП 00.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

Рыжанкова А.С

Утверд.

*Пацей Н.В.*

Содержание

Лит.

Листов

2

*БГТУ 74417058, 2020*

[Введение 7](#_Toc41002567)

[1 Аналитический обзор литературы и постановка задачи 8](#_Toc41002568)

[1.1 Описание и разработка web-сервиса 8](#_Toc41002569)

[1.2 Обзор аналогов разрабатываемого программного средства 11](#_Toc41002570)

[1.2.1 Сервис Директ Кредит 11](#_Toc41002571)

[1.2.2 Онлайн оформление кредитов в системе банков 12](#_Toc41002572)

[1.2.3 Онлайн оформление лизингов 13](#_Toc41002573)

[1.3 Обзор используемых технологий 14](#_Toc41002574)

[1.3.1 Язык программирования JavaScript 14](#_Toc41002575)

[1.3.2 Стандарт EcmaScript 15](#_Toc41002576)

[1.3.3 Библиотека ReactJS 16](#_Toc41002577)

[1.3.4 Библиотека React-Redux 17](#_Toc41002578)

[1.6.5 Среда выполнения Node.js 18](#_Toc41002579)

[1.3.6 Базы данных NoSQL 19](#_Toc41002580)

[1.3.7 Микросервисная архитектура 20](#_Toc41002581)

[1.4 Постановка задачи 21](#_Toc41002582)

[1.5 Вывод по разделу 22](#_Toc41002583)

[2 Проектирование программного средства 23](#_Toc41002584)

[2.1 Основные технические требования к разработке 24](#_Toc41002585)

[2.2 Проектирование базы данных 24](#_Toc41002586)

[2.2.2 Контекст данных приложения 24](#_Toc41002587)

[2.2.3 Модели базы данных 25](#_Toc41002588)

[2.3 Проектирование веб-сервисов 26](#_Toc41002589)

[2.3.1 Разработка диаграммы вариантов использования 27](#_Toc41002590)

[2.3.2 Структура приложения 29](#_Toc41002591)

[2.5 Вывод 31](#_Toc41002592)

[3 Реализация программного средства 32](#_Toc41002593)

[3.1 Разработка клиентской части приложения 32](#_Toc41002594)

[3.1.1 Маршрутизация, используемая в приложении 34](#_Toc41002595)

[3.1.2 Компоненты приложения 36](#_Toc41002596)

[3.1.3 Действия (actions) для манипуляции данными приложения 37](#_Toc41002597)

[3.1.4 Сервисы и утилиты, реализованные в приложении 38](#_Toc41002598)

[3.1.5 Зависимости клиентской части приложения 38](#_Toc41002599)

[3.2 Реализация серверной части 39](#_Toc41002600)

[3.2.1 Маршрутизаторы для обработки запросов, авторизация 40](#_Toc41002601)

[3.2.2 Сервисы, реализованные в приложении 40](#_Toc41002602)

[3.2.3 Модели данных приложения 42](#_Toc41002603)

[3.2.3 Модели данных приложения 42](#_Toc41002604)

[3.2.5 Поставщики данных (провайдеры) приложения 43](#_Toc41002605)

[3.3 Вывод по разделу 43](#_Toc41002606)

[4 Тестирование приложения 44](#_Toc41002607)

[4.1 Тестирование серверной части 44](#_Toc41002608)

[4.1.1 Аутентификация пользователя 44](#_Toc41002609)

[4.1.1 Создание нового аккаунта пользователя 45](#_Toc41002610)

[4.1.2 Доступ к запрещенным ресурсам 45](#_Toc41002611)

[4.2 Тестирование клиентской части 46](#_Toc41002612)

[4.4 Вывод по разделу 49](#_Toc41002613)

[5 Руководство по установке и использованию программного обеспечения 50](#_Toc41002614)

[5.1 Системные требования оборудования 50](#_Toc41002615)

[5.2 Необходимое программное обеспечение и библиотеки 50](#_Toc41002616)

[5.3 Подготовка инфраструктуры и развертывание приложения 51](#_Toc41002617)

[5.4 Инструкция пользователя 53](#_Toc41002618)

[5.5 Вывод по разделу 62](#_Toc41002619)

[6 Экономический раздел 63](#_Toc41002620)

[6.1 Общая характеристика разрабатываемого программного средства 63](#_Toc41002621)

[6.2 Исходные данные и маркетинговый анализ 63](#_Toc41002622)

[6.3 Методика обоснования цены 64](#_Toc41002623)

[6.3.1 Объём программного средства 64](#_Toc41002624)

[6.3.2 Основная заработная плата 65](#_Toc41002625)

[6.3.3 Дополнительная заработная плата 66](#_Toc41002626)

[6.3.4 Отчисления в Фонд социальной защиты населения 66](#_Toc41002627)

[6.3.5 Расходы на материалы 67](#_Toc41002628)

[6.3.6 Расходы на оплату машинного времени 67](#_Toc41002629)

[6.3.7 Прочие прямые затраты 68](#_Toc41002630)

[6.3.8 Накладные расходы 68](#_Toc41002631)

[6.3.9 Сумма расходов на разработку программного средства 68](#_Toc41002632)

[6.3.10 Расходы на сопровождение и адаптацию 68](#_Toc41002633)

[6.3.11 Полная себестоимость 69](#_Toc41002634)

[6.3.12 Определение цены, оценка эффективности 69](#_Toc41002635)

[6.4 Выводы по разделу 70](#_Toc41002636)

[Заключение 71](#_Toc41002637)

[Список использованных источников 72](#_Toc41002638)

[ПРИЛОЖЕНИЕ А 73](#_Toc41002639)

[ПРИЛОЖЕНИЕ Б 74](#_Toc41002640)

[ПРИЛОЖЕНИЕ В 75](#_Toc41002641)

[ПРИЛОЖЕНИЕ Г 76](#_Toc41002642)

[ПРИЛОЖЕНИЕ Д 77](#_Toc41002643)

[ПРИЛОЖЕНИЕ Е 78](#_Toc41002644)

Введение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С*.

Утверд.

*Пацей Н.В.*

Введение

Лит.

Листов

1

*БГТУ 74417058, 2020*

У

На сегодняшний день можно действительно сказать, что информационные технологии играют большую и довольно значимую роль в жизни каждого человека. Разве можно уже сейчас представить современного человека без каких-либо гаджетов: мобильных телефонов, планшетов, ноутбуков и т.д.

Большую и довольно значимую роль в развитие информационных технологий внесло появление и повсеместное распространение интернета. Благодаря возможностям предоставляемым интернетом появилась и сформировалась новая ниша в информационных технологиях, такая как веб-технологии. Множество программного обеспечения было перепроектировано из монолитных десктопных приложений в веб-сервисы, веб-приложения и веб-сайты. Теперь тот функционал, который раньше мог быть использован и получен пользователем на компьютере, может быть доступен на любом устройстве, у которого имеется возможность подключения к интернету.

В современном мире у человека есть множество разных возможностей, всегда есть выбор, какие-то альтернативы. И во многом богатство выбора в окружающем нас мире зависит от денежных средств. А если учесть высокую скорость развития человечества, быстрый рост потребностей, развитие IT-индустрии, то приходит осознание того, что без некоторой суммы денежных средств человеку в нашем мире будет тяжело, ибо без них человек не сможет приобретать как необходимые для поддержания жизни ресурсы, так и вещи, потакающие его прихотям. Иногда для этого требуется брать деньги в долг, а также оформлять кредиты и лизинги ради приобретения желаемого. Учитывая, что в нашем мире всё чаще необходимые человеку услуги оцифровываются, то разные банковые возможности теперь доступны и онлайн.

В представленном дипломном проекте будет разработано программное средство для создания и управления кредитными/лизинговыми продуктами и заявками. Программное средство будет реализовано в виде веб-приложения, чтобы предоставлять пользователям доступ через интернет при помощи персонального компьютера или ноутбука.

Для достижения поставленной цели необходимо решить следующие задачи:

* разработать клиентскую часть, которая позволит создавать кредитные и лизинговые продукты и заявки;
* разработать серверную часть, которая будет осуществлять валидацию входных данных, генерировать электронные документы, сохранять данные в базу данных.

1 Аналитический обзор литературы и постановка задачи

ФИО

Подпись

Дата

Лист

1

ДП 01.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С.*

Утверд.

*Пацей Н.В.*

1 Аналитический обзор литературы и постановка задачи

Лит.

Листов

8

*БГТУ 74417058, 2020*

Сегодня банковская отрасль неумолимо растёт и развивается. Но, как известно, чем больше система – тем выше шанс возникновения ошибки. Увеличение количества персонала повышает шанс возникновения ошибок со стороны человека, который следит за оформлением кредитных и/или лизинговых заявок сотрудниками.

Оформление кредитных/лизинговых заявок в банковской отрасли – необходимая составляющая. В повышении качества может сыграть положительную роль автоматизация самых различных процессов.

Несмотря на довольное большое количество схожих платформ, на данный момент нет аналогичной платформы, которая ориентируется исключительно на белорусский рынок.

Поэтому главные критерии для данного приложения – это реализация веб-приложения, которая дает возможность добавления нового продукта для банка или лизинговой компании, создание точек продаж, а также пользователей для управления системой. А также, что не менее важно – реализации заполнения кредитной или лизинговой заявки. В контексте данного приложения будет доступна строго валидированная форма для оформления заявки, которая содержит в себе данные по кредиту или лизингу, данные о клиенте, его работе и контактном лице.

* 1. Описание и разработка web-сервиса

Сервер *API* (*Application Programming Interface*) – это определенное представление данных для взаимодействия между приложениями. В частном случае, в качестве ответного приложения, может выступать сервер. *API* – это описанный формат, которому должны соответствовать обе стороны обмена данными.

Все популярные онлайн-сервисы и системы имеют публичные *API*. Это позволяет сторонним разработчикам интегрировать в свои программные продукты функционал, предоставляемый публичным *API* - сервером.

Большинству разработчиков сайтов, веб-сервисов и мобильных приложений рано или поздно приходится иметь дело с клиент-серверной архитектурой, а именно разрабатывать web API или интегрироваться с ним. Чтобы не изобретать каждый раз что-то новое, важно выработать относительно универсальный подход к проектированию *web API*, основываясь на опыте разработки подобных систем.

Сервером, в случае *web API*, считается абстрактная машина в сети, способная получить *HTTPS*-запрос, обработать его и вернуть корректный ответ. В общем случае совершенно не важны физическая суть и внутренняя архитектура сервера, будь это студенческий ноутбук или огромный кластер из промышленных серверов, разбросанных по всему миру.

В той же мере совершенно неважны технологии, с помощью которых осуществляется обработка запросов, *Apache* или *Nginx*, *PHP*, *Python* или *Ruby* выполняет его обработку и формирует ответ, какое хранилище данных используется: *Postgresql*, *MySQL* или *MongoDB*. Главное, чтобы сервер отвечал главному правилу – принять запрос, обработать его и вернуть ответ клиенту.

Архитектура *REST* (*Representational state transfer*) изначально был задуман как простой и однозначный интерфейс для управления данными, предполагавший всего несколько базовых операций с непосредственным сетевым хранилищем (сервером): извлечение данных (*GET*), сохранение (*POST*), изменение *(PUT/PATCH*) и удаление (*DELETE*). Разумеется, этот перечень всегда сопровождается такими опциями, как обработка ошибок в запросе (корректно ли составлен запрос), разграничение доступа к данным и валидация входящих данных, и другими возможными проверками, которые сервер выполняет перед тем, как выполнить запрос клиента [1].

Помимо этого, *REST* имеет ряд архитектурных принципов, перечень которых представлен ниже:

* независимость сервера от клиента – серверы и клиенты могут быть мгновенно заменены другими независимо друг от друга, так как интерфейс между ними не меняется. Сервер не хранит состояний клиента.
* уникальность адресов ресурсов – каждая единица данных (любой степени вложенности) имеет свой собственный уникальный *URL*, который, по сути, целиком является однозначным идентификатором ресурса;
* независимость формата хранения данных от формата их передачи;
* сервер может поддерживать несколько различных форматов для передачи
* одних и тех же данных (*JSON*, *XML* и т.д.), но хранит данные в своем внутреннем формате, независимо от поддерживаемых;
* присутствие в ответе всех необходимых метаданных – помимо самих данных сервер должен возвращать детали обработки запроса, например, сообщения об ошибках, различные свойства ресурса, необходимые для дальнейшей работы с ним, например, общее число записей в коллекции для правильного отображения постраничной навигации.

Ключевым типом данных в общении между клиентом и сервером выступает объект. По сути, объект – это перечень свойств и соответствующих им значений. В процессе своей работы клиент может отправить объект на сервер в запросе и получить результат запроса в виде объекта. При этом объект не обязательно будет реальной сущностью, хранящейся в базе данных, по крайней мере, в том виде, в котором он отправлен или получен. Например, учетные данные для авторизации передаются в виде объекта, но не являются самостоятельной сущностью. Даже хранимые в БД объекты склонны обрастать дополнительными свойствами внутрисистемного характера, например, датами создания и редактирования, различными системными метками и флагами. Свойства объектов могут быть как собственными скалярными значениями, так и содержать связанные объекты и коллекции объектов, которые не являются частью объекта. Часть свойств объектов может быть редактируемой, часть системной, доступной только для чтения, а часть может носить статистический характер и вычисляться в процессе выполнения (например, количество лайков в социальной сети). Некоторые свойства объекта могут быть скрыты, в зависимости от прав пользователя.

В процессе работы с сущностями практически всегда возникает необходимость работать с коллекциями. Говоря о коллекциях, подразумевается разновидность серверного ресурса, позволяющая работать с перечнем однородных объектов, т.е. добавлять, удалять, изменять объекты и осуществлять выборку из них. Помимо этого, коллекция теоретически может обладать собственными свойствами (например, максимальное число элементов на страницу) и функциями.

Рано или поздно любая действующая система начинает эволюционировать: развиваться, усложняться, масштабироваться, усовремениваться. Для разработчиков *REST API* это чревато в первую очередь тем, что необходимо запускать новые версии API при работающих старых. Основные архитектурные изменения подразумевают, что изменяется сам формат данных и набор операций с ними. В любом случае версионность нужно предусмотреть как в изначальной организации исходного кода, так и в принципе построения *URL*. Что касается *URL*, здесь существует два наиболее популярных способа указания версии *API*, которой адресован запрос. Префиксация пути *example-api.com/v1/* и разведение версий на уровне субдомена *v1.example-api.com*. Использовать можно любой из них, в зависимости от потребности и необходимости [2].

Сервис *Web API* сложных систем, поддерживающих несколько пользовательских ролей, зачастую требует разделения на части, каждая из которых обслуживает свой спектр задач. По сути, каждая часть может быть самостоятельным приложением, работать на разных физических машинах и платформах. В контексте описания API совершенно не важно, как сервер обрабатывает запрос и какие технологии в этом замешаны. Для клиента *API* – система инкапсулирована. Тем не менее разные части системы могут обладать совершенно разной функциональностью, например, административная и пользовательская часть. И методология работы с одними и теми же, казалось бы, ресурсами может существенно отличаться. Поэтому такие части необходимо разделять на уровне домена *admin.v1.example-api.com* или префикса пути *example-api.com/v1/admin/*. Это требование не является обязательным, и многое зависит от сложности системы и её назначения.

В современном сообществе разработчиков установилось мнение, что самым удобным и функциональным, форматом обмена данными является *JSON*, но также популярны *XML*, *YAML* другие форматы, позволяющее хранить сериализованные объекты без потери типа данных. При желании можно сделать в *API* поддержку нескольких форматов ввода/вывода. Достаточно задействовать *HTTP* заголовок запроса для указания желаемого формата ответа *Accept* и *Content-Type* для указания формата переданных в запросе данных. Другим популярным способом является добавление расширения к *URL* ресурса, например, GET /users.xml, но такой способ кажется менее гибким, хотя бы потому, что утяжеляет URL и верен скорее для *GET*-запросов, нежели для всех возможных операций.

Помимо собственно данных, в ответах, возвращающих набор элементов коллекции, обязательно должна присутствовать информация о постраничной навигации (пагинации) по результатам запроса.

Минимальный набор значений для пагинации состоит из:

* общего числа записей;
* числа страниц;
* номера текущей страницы;
* числа записей на странице;
* максимального числа записей на странице, поддерживаемого серверной стороной.

Некоторые разработчики *Web API* также включают в пагинацию набор готовых ссылок на соседние страницы, а также первую, последнюю и текущую.

Результаты выборки требуется упорядочить по возрастанию или убыванию значений определенных полей, которые поддерживают сравнительную (для числовых полей) или алфавитную (для строковых полей) сортировку. Например, упорядочить список пользователей по имени или товары по цене. Помимо этого, можно задать направление сортировки от A до Я или в обратном направлении, причём разное для разных полей.

* 1. Обзор аналогов разрабатываемого программного средства

Поскольку банковская сфера развита на всем земном шаре, то и необходимость в удобной форме кредитования. К сожалению, аналоги поставляются исключительно на платной основе, и судить об их функционале приходится исходя из описания в официальных источниках, а также опираясь на отзывы непосредственных пользователей. На текущий момент разработано небольшое количество схожих систем. Так как дипломный проект ориентирован на интернет приложение, то поиск аналогичных программных средств также ориентирован именно на web-приложения.

1.2.1 Сервис Директ Кредит

Сервис Директ Кредит является крупнейшим сервисом для ПОС-кредитования в России для интернет-магазинов, для торговых точек и сетей с небольшим кредитным оборотом. Сервис позволяет создавать заявки, использовать кредитный калькулятор. Данный сервис так же предоставляет возможность просматривать свои заявки и загружать необходимые документы после подписания их клиентом.

С помощью «Директ Кредит» вы предоставляете условия *online* кредитования покупателям. «Директ Кредит» объединяет необходимые вам банки.

Минусы процесса оформления заявки через платформу «Директ Кредит»:

* Ориентирован только на рынок России;
* в случае работы с брокером, клиент узнает условия только после получения одобрения;
* онлайн заявка оставляет клиента на сайте до конечного результата (в случае с брокером клиент ожидает звонка, за это время он может найти товар у конкурентов);
* по завершению оформления заявки на кредит клиент получает решение от нескольких банков, которые затем фильтруются брокером и только потом предоставляется выбор клиенту.

Интерфейс приложения можно видеть на рисунке 1.1.

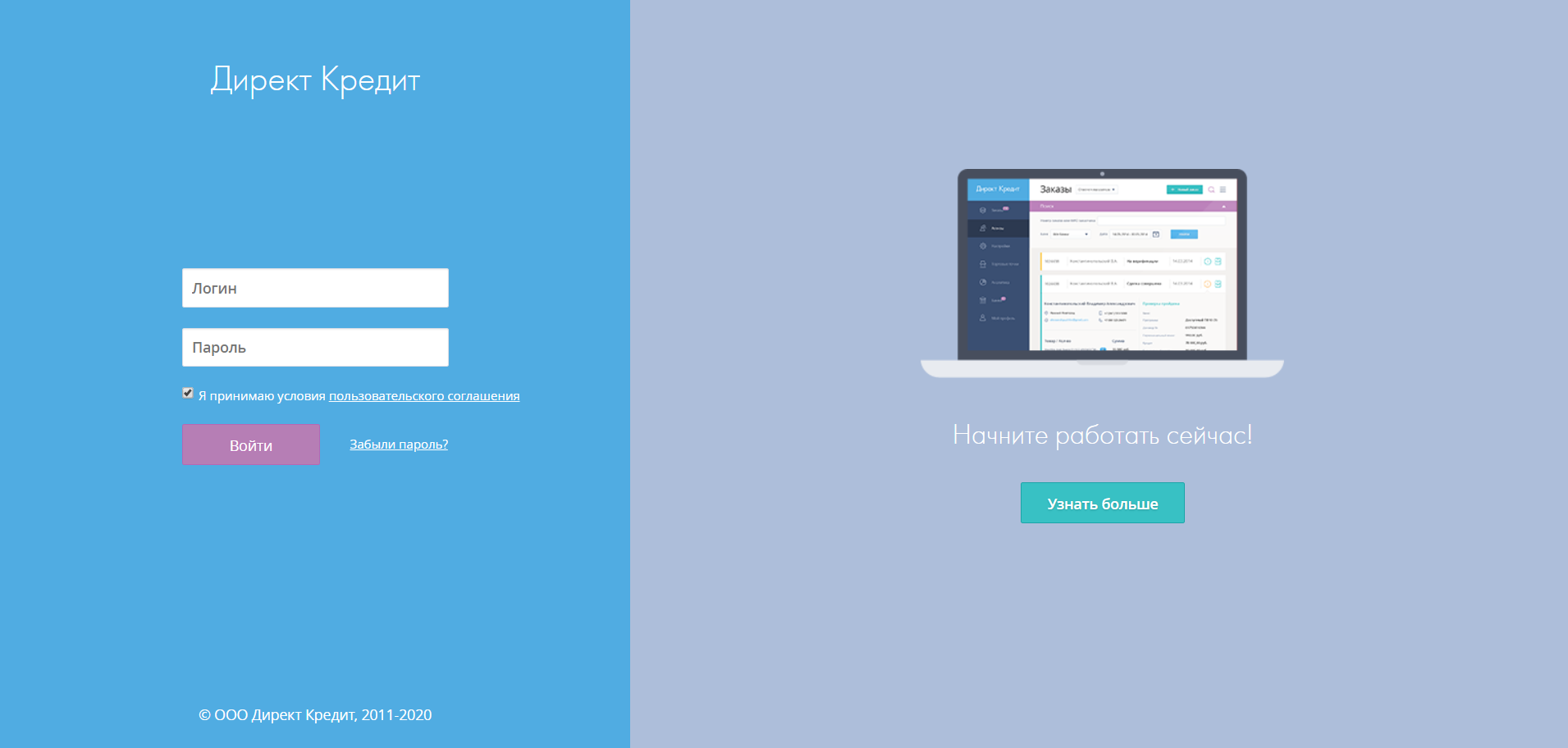


Рисунок 1.1 – Интерфейс приложения Директ Кредит

Возможности, предоставляемые Директ Кредит:

* прямой выбор программ кредитования;
* доставка документов;
* сопровождение и консультирование клиентов.

1.2.2 Онлайн оформление кредитов в системе банков

Каждый банк предоставляет возможность оформить кредит у себя в интернет-банкинге или мобильном приложении.

Основным минусом является ограниченность выбора и доступность только клиентам.

Вы можете оформить кредит в течении пяти минут имея доступ к необходимым ресурсам. Весь процесс кредитования происходит онлайн: от оформления заявки до выдачи кредита.

Чтобы получить кредит не требуется никуда ехать, необходим только интернет. Получайте дополнительные деньги, когда это надо, и там, где удобно: в магазине, дома, за границей.

Чтобы взять кредит, вам не нужно предоставлять банку дополнительное обеспечение: ни залог, ни поручителей.

Вся необходимая информация расписана грамотно и красиво.

Так же в таких системах существует возрастной порог, минимальный среднемесячный доход и непрерывный стаж работы.

Интерфейс интернет-банкинга «Приорбанк» представлен на рисунке 1.2.

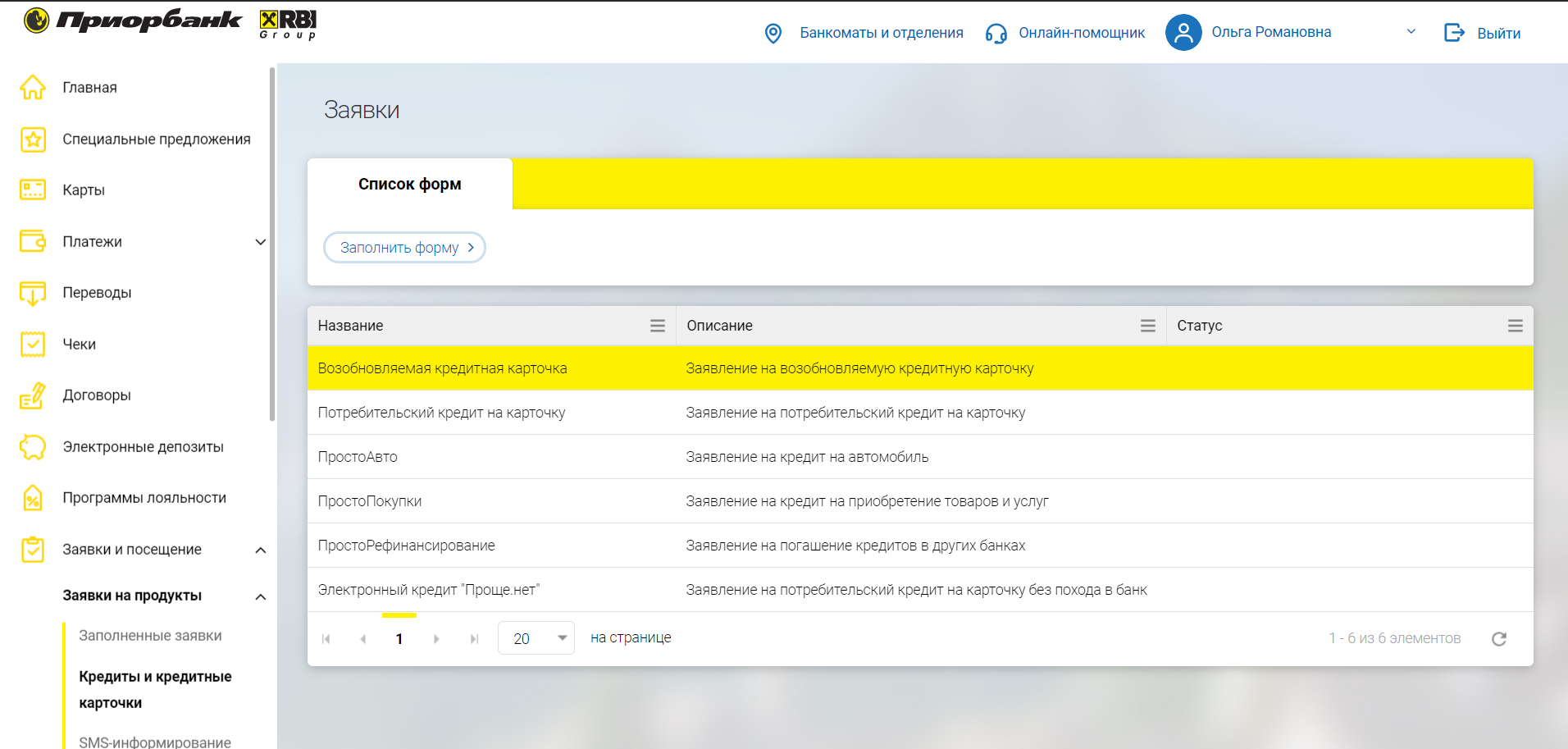


Рисунок 1.2 – Интерфейс интернет-банкинга «Приорбанка»

Возможности, предоставляемые интернет-банкингом «Приорбанк»:

* быстро и доступно в любое время суток;
* удобное погашение;
* без залогов и поручителей.

1.2.3 Онлайн оформление лизингов

Оформление лизингов онлайн доступно на некоторых сайтах лизинговых компаний. Но для подписания соответствующих бумаг, пользователю все равно необходимо обратиться к поставщикам услуг.

Одним из таких примеров является Маркетплейс. На сайте вы можете заказать консультацию по лизинговому продукту или сразу оформить заявку.

Суть покупки в лизинг достаточно проста — лизинговая компания заключает с клиентом договор лизинга, беря на себя обязательство приобрести определенное имущество, которое необходимо клиенту, и передать это имущество клиенту в пользование на определенный срок. При этом клиент сам выбирает имущество и продавца этого имущества. При заключении договора лизинга клиент, как правило, часть стоимости имущества оплачивает сразу (авансовый платеж), а остальное вносит равными долями в течение определенного срока. По окончании срока лизинга и выплаты платежей по договору клиент получает имущество в собственность.

Одним из главных плюсов этой компании то, что лизинговые платежи содержат в себе налог на добавочную стоимость и могут быть зачтены и относятся к расходам компании.

Так же компания обещает быстрое оформление и быстрое получение денег. Так же клиент сам определяет, что он хочет приобрести, его комплектность, продавца имущества и условия поставки. После принятия положительного решения о возможности заключения лизинговой сделки подготавливается договор лизинга.

Интерфейс сервиса «Маркетплейс» представлен на рисунке 1.3.

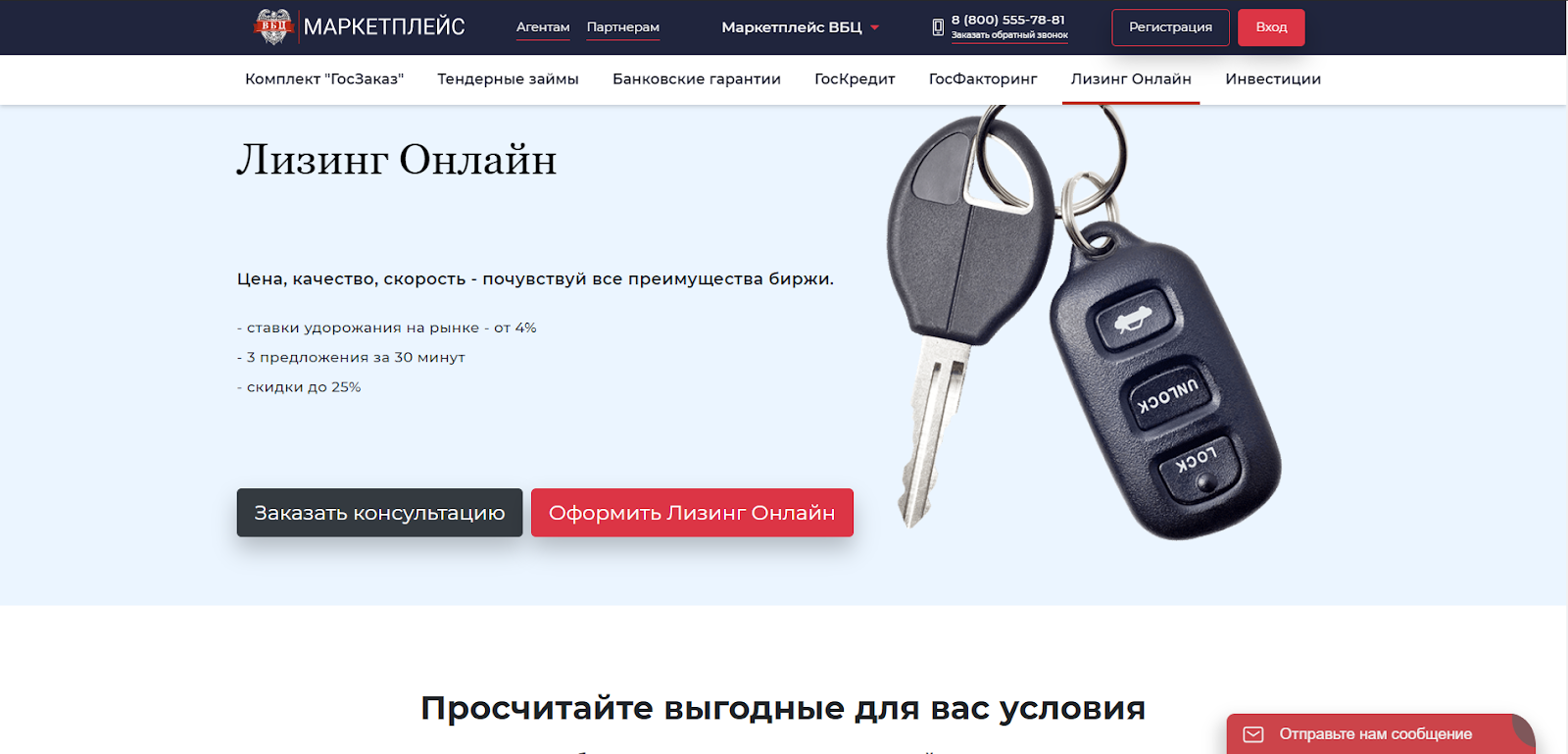


Рисунок 1.3 — Сервис Маркетплейс

Возможности, предоставляемые сервисом «Маркетплейс»:

* быстро и доступно;
* небольшой пакет предоставляемых документов;
* денежные средства получается сразу после одобрения.

## **1.3 Обзор используемых технологий**

## **1.3.1** **Язык программирования JavaScript**

Язык программирования *JavaScript* добавляет интерактивность на ваш веб-сайт: обработчики событий, динамические стили, анимация. Это полноценный динамический язык программирования, который применяется к *HTML* документу, и может обеспечить динамическую интерактивность на веб-сайтах.

Язык *JavaScript* исполняется в браузере, имеет множество готовых модулей и обширную документацию. Разрабатывается с 1995 года, имеет множество стандартов.

Популярные современные браузеры обеспечивают поддержку последних стандартов *ECMAScript*, а это значит, что и приложение, в основе которого лежит *JavaScript*, может быть выполнено в любом из этих браузеров.

Обеспечивает кроссплатформенность, поскольку, фактически, веб-приложение не зависит от операционной системы, в которой оно исполняется. Ещё одной положительной чертой является независимость от устройства, которое использует конечный пользователь: будь то персональный компьютер, планшет или мобильный телефон — всё это не имеет никакого значения, если на данном устройстве используется браузер с поддержкой *JavaScript*.

К минусам можно отнести лишь то, что разные браузеры обеспечивают разную поддержку *JavaScript*. Например, *Safari* до сих пор не поддерживает функцию «*Object.values*». К счастью, такие случаи возникают редко.

На момент написания актуальным стандартом является *ECMAScript* 2018.

На момент написания последней редакцией *ECMAScript* является десятая версия. Выпущена в июле 2019 года [3].

1.3.2 Стандарт EcmaScript

*EcmaScript* – это просто сокращение для *ECMAScript*. Каждое издание ECMAScript получает аббревиатуру *ES* с последующим его номером. Всего существует 8 версий *ECMAScript*. *ES1* была выпущена в июне 1997 года, *ES2* – в июне 1998 года, *ES3* – в декабре 1999 года, а версия *ES4* – так и не была принята. Не будем углубляться в эти версии, так как они морально устарели, а рассмотрим только последние четыре.

*ES5* был выпущен в декабре 2009 года, спустя 10 лет после выхода третьего издания. Среди изменений можно отметить:

* поддержку строгого режима (*strict mode*);
* аксессоры *getters* и *setters*;
* возможность использовать зарезервированные слова в качестве ключей свойств и ставить запятые в конце массива;
* многострочные строковые литералы;
* новую функциональность в стандартной библиотеке;
* поддержку *JSON*.

Версия *ES6*/*ES2015* вышла в июне 2015 года. С выходом этого пакета обновлений комитет принял решение перейти к ежегодным обновлениям. Поэтому издание было переименовано в *ES2015*, чтобы отражать год релиза. Последующие версии также называются в соответствии с годом их выпуска. В этом обновлении были сделаны следующие изменения:

* добавлено деструктурирующее присваивание;
* добавлены стрелочные функции;
* в шаблонных строках можно объявлять строки с помощью обратных кавычек. Шаблонные строки могут быть многострочными, также могут интерполироваться;
* *let* и *const* – альтернативы *var* для объявления переменных. Добавлена «временная мертвая зона»;
* итератор и протокол итерации теперь определяют способ перебора любого объекта, а не только массивов. *Symbol* используется для присвоения итератора к любому объекту;
* добавлены функции-генераторы. Они используют *yield* для создания последовательности элементов. Функции-генераторы могут использовать *yield*\* для делегирования в другую функцию генератора, кроме этого они могут возвращать объект генератора, который реализует оба протокола;
* добавлены промисы.

В соответствии с вышеперечисленным можно сделать вывод, что данный стандарт упрощает жизнь разработчикам и ускоряет время написания кода.

*ES2016* (*ES7*) вышла в июне 2016 года. Среди изменений в этой версии *ECMAScript* можно отметить:

* оператор возведения в степень \*\*;
* метод *Array*.*prototype*.*includes*, который проверяет, содержится ли переданный аргумент в массиве.

Спустя еще год выходит версия *ES2017* (*ES8*). Данный стандарт получил следующие изменения:

* асинхронность теперь официально поддерживается (*async*/*await*);
* «висячие» запятые в параметрах функций. Добавлена возможность ставить запятые в конце списка аргументов функций;
* добавлено два новых метода для работы со строками: *padStart*() и *padEnd*(). Метод *padStart*() подставляет дополнительные символы слева, перед началом строки. А *padEnd*(), в свою очередь, справа, после конца строки;
* добавлена функция *Object.getOwnPropertyDescriptors*(), которая возвращает массив с дескрипторами всех собственных свойств объекта;
* добавлено разделение памяти и объект *Atomics* [4].

## **1.3.3 Библиотека ReactJS**

Библиотека *ReactJS* – это одна из самых популярных библиотек для создания сложных клиентских приложений. Представляет собой *JavaScript* библиотеку для построения пользовательских интерфейсов. Данная библиотека работает с виртуальным *DOM*, самостоятельно отслеживает изменения в состоянии приложения и выполняет перерисовку лишь той тех элементов страницы, которые были изменена, что является несомненным плюсом для производительности [5].

*ReactJS* основывается на так называемых компонентах – это классы, позволяющие описывать элементы *DOM*, производить различные операции над ними в ходе жизненного цикла компонента.

Компоненты *ReactJS* являются переиспользуемыми, поскольку представляют собой обычные классы, унаследованные от *React.Component*. При грамотном проектировании это позволяет исключить дублирование кода, обеспечить гибкость и далее легкую поддержку готового приложения. Для *ReactJS* существует множество обёрток, библиотек и готовых решений, которые позволяют просто и быстро начать реализацию нового проекта. Библиотека является самой популярной из рассматриваемых.

Причины использовать *React*:

* отлично подходит для командной разработки, строгое соблюдение *UI*, и шаблона рабочего процесса;
* код читабельный и прост в сопровождении;
* разработка *UI* на основе отдельных компонентов – это будущее веб-разработки и чем раньше это поймет разработчик – тем лучше для него.

Одна из главных особенностей *ReactJS* – это свобода действий, существует огромное количество подходов к построению приложений с его помощью (*react-redux, mobx, react-flux* и другие) [6].

К минусам ReactJS можно отнести сложность в освоении, поскольку для создания действительно хороших приложения с его использованием необходимо хорошее понимание его концепций.

1.3.4 Библиотека React-Redux

*Redux* – это менеджер состояний. Чаще всего его используют с *React*, но его возможности не ограничиваются одной этой библиотекой. Хотя в *React* есть собственный метод управления состояниями, он плохо масштабируется. Перемещение состояния вверх по дереву работает для простых приложений, но в более сложных архитектурах изменение состояния производится через свойства (*props*). Ещё лучше делать это через внешнее глобальное хранилище.

Схема работы *react-redux* представлена на рисунке 1.4.

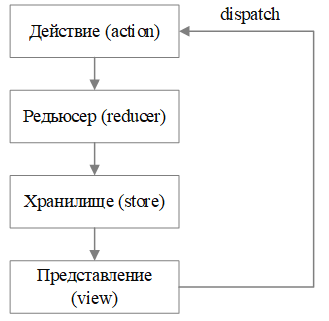


Рисунок 1.4 — Схема работы *react-redux* приложения

Как можно видеть из схемы, представленной на рисунке выше, жизненный цикл обработки данных является однонаправленным, что упрощает понимание работы приложения.

Схема содержит следующие компоненты:

* действие (*action*);
* редьюсер (*reducer*);
* хранилище данных приложения (*store*);
* представление (*view*) – результат рендера;
* диспатчер (*dispatch*) – позволяет распространять действия для изменения состояния [7].

В *Redux* общее состояние приложения представлено одним объектом *JavaScript* – *state* (состояние) или *state tree* (дерево состояний). Неизменяемое дерево состояний доступно только для чтения, изменить ничего напрямую нельзя. Изменения возможны только при отправке *action* (действия).

При запуске действия обязательно что-то происходит и состояние приложения изменяется. Это работа редукторов. Редуктор (*reducer*) – это чистая функция, которая вычисляет следующее состояние дерева на основании его предыдущего состояния и применяемого действия. Чистая функция работает независимо от состояния программы и выдаёт выходное значение, принимая входное и не меняя ничего в нём и в остальной программе. Получается, что редуктор возвращает совершенно новый объект дерева состояний, которым заменяется предыдущий.

Хранилище (*store*) – это объект, который содержит состояние приложения, отображает состояние через *getState()*, может обновлять состояние через *dispatch*(), позволяет регистрироваться (или удаляться) в качестве слушателя изменения состояния через *subscribe()*.

1.3.5 Среда выполнения Node.js

Среда выполнения *Node.js* построена на *JavaScript*-движке *V8* из *Chrome*. В основе *Node*.*js* лежит событийно-управляемая модель с неблокирующими операциями *I/O*, что делает её легкой и эффективной [8].

Другими словами, *Node*.js предлагает вам возможность писать невероятно производительный серверный код с использованием *JavaScript*. Как говорится в официальном описании: *Node*.*js* –  это среда выполнения, использующая тот же *JavaScript*-движок *V8*, который вы можете найти в браузере *Google* *Chrome*. Но этого недостаточно для успеха *Node*.*js*. В *Node*.*js* используется *libuv* – кроссплатформенная библиотека поддержки с акцентом на асинхронный ввод-вывод [9].

С точки зрения разработчика, *Node*.*js* однопоточен, но под капотом *libuv* использует потоки, события файловой системы, реализует цикл событий и так далее. В большинстве случаев разработчик не взаимодействует с *libuv* напрямую.

Плюсы *Node*.*js*:

* способность держать очень много «сквозных» запросов. В самом распространённом случае сервер не ждёт ответа базы данных, а обрабатывает остальные запросы, аналогичная картина с *I/O* операциями, работой с файлами и так далее;
* масштабируемость;
* пакетные менеджеры *npm* и *yarn* со своей базой модулей, позволяющий управлять модулями. Количество реализованных библиотек огромно, а их назначение крайне разнообразно [10];
* разделение кода. Выгода очевидна: серверная часть не содержит клиентского кода, как, например, с *ASP.NET MVC*, где серверная часть содержит представления;
* легковесность;
* язык *JavaScript*. Все его плюсы можно отнести к плюсам *Node*.*js*.

Минусы:

* порог входа. Например, разработка с использованием *ASP.NET MVC* проще, потому что он описан четкими соглашениями по написанию кода и является строго типизированным языком.
* совершенно не подходит для математических операций, тяжёлого рендеринга, и вообще любых «не сквозных» операций;
* необходимость очень тщательно следить за кодом и обрабатывать ошибки.

1.3.6 Базы данных NoSQL

Базы данных *NoSQL* – это термин, использующийся для обозначения высокопроизводительных нереляционных баз данных. В базах данных *NoSQL* используются различные модели данных, включая документную, графовую и столбчатую модели, а также модель «ключ-значение». Базы данных *NoSQL* получили широкое распространение в связи с простотой эксплуатации, возможностью масштабирования, высокой доступностью и гибкостью. Все это достигается путем смягчения жестких требований к непротиворечивости данных, характерных для других типов БД.

Базы данных *NoSQL* хорошо подходят для многих современных приложений, например, мобильных, игровых, интернет‑приложений, когда требуются гибкие масштабируемые базы данных с высокой производительностью и широкими функциональными возможностями, способные обеспечивать максимальное удобство использования.

Как правило, базы данных *NoSQL* предлагают гибкие схемы, что позволяет осуществлять разработку быстрее и обеспечивает возможность поэтапной реализации. Благодаря использованию гибких моделей данных БД *NoSQL* хорошо подходят для частично структурированных и неструктурированных данных.

Базы данных *NoSQL* рассчитаны на масштабирование с использованием распределенных кластеров аппаратного обеспечения, а не путем добавления дорогих надежных серверов. Некоторые поставщики облачных услуг проводят эти операции в фоновом режиме, обеспечивая полностью управляемый сервис [11].

Документно-ориентированная базы данных позволяют разработчикам хранить и запрашивать данные в БД с помощью той же документной модели, которую они используют в коде приложения. Гибкий, полуструктурированный, иерархический характер документов и документных баз данных позволяет им развиваться в соответствии с потребностями приложений.

1.3.7 Микросервисная архитектура

Микросервисная архитектура – вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых.

Структура микроcервисной архитектуры представлен на рисунке 1.5.

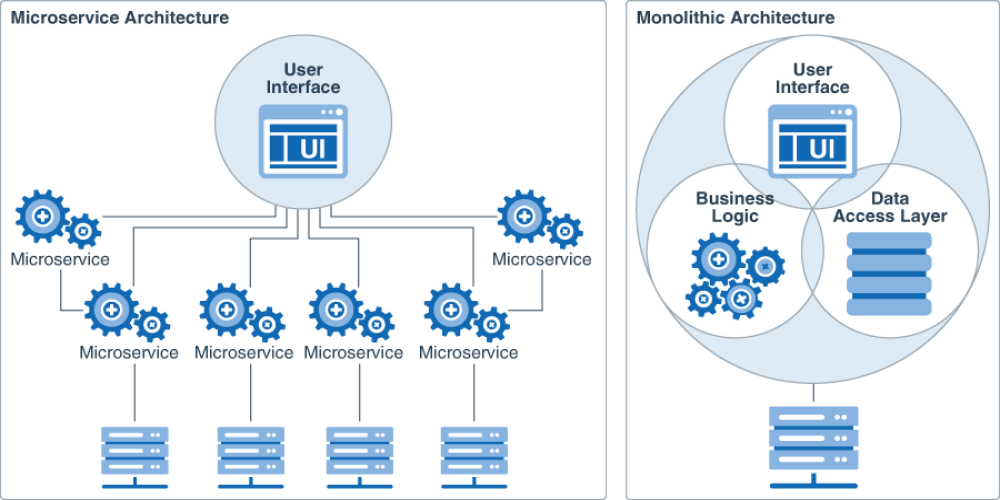


Рисунок 1.5 — Структура микросервиной архитектуры

Свойства, характерные для микросервисной архитектуры:

* модули можно легко заменить в любое время: акцент на простоту, независимость развёртывания и обновления каждого из микросервисов;
* модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;
* модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах: приоритет отдаётся в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;
* архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые [12].

Архитектура микросервисов использует библиотеки, но их основной способ разбиения приложения – путем деления его на сервисы. Мы определяем библиотеки как компоненты, которые подключаются к программе и вызываются ею в том же процессе, в то время как сервисы — это компоненты, выполняемые в отдельном процессе и коммуницирующие между собой через веб-запросы или *remote* *procedure* *call* (*RPC*).

Главная причина использования сервисов вместо библиотек – это независимое развертывание. Если вы разрабатываете приложение, состоящее из нескольких библиотек, работающих в одном процессе, любое изменение в этих библиотеках приводит к переразвертыванию всего приложения. Но если ваше приложение разбито на несколько сервисов, то изменения, затрагивающие какой-либо из них, потребуют переразвертывания только изменившегося сервиса. Конечно, какие-то изменения будут затрагивать интерфейсы, что, в свою очередь, потребует некоторой координации между разными сервисами, но цель хорошей архитектуры микросервисов — минимизировать необходимость в такой координации путем установки правильных границ между микросервисами, а также механизма эволюции контрактов сервисов [13].

1.4 Постановка задачи

Необходимо разработать веб-платформу для оформления кредитных и лизинговых продуктов и заявок с применением стека технологий:

* *Java Script*;
* *Node.js*;
* *React.js*;
* *Express*;
* *Redux*;
* *Redux-From*;
* *Redis*;
* *GIT*;
* *Pug*;
* *Swagger*;
* *Mongo DB*;
* *ODM* *Mongoose*.
* *Docker*;
* *Ngnix*.

Основные требования к проекту:

* возможность создания новых пользователей в роли администраторов, менеджеров, брокеров и банковских работников;
* возможность создания новых владельцев продуктов, как банков, так и лизинговых компаний;
* возможность создания новых точек продаж и назначение менеджеров для их управления;
* возможность блокировки и удаления существующих точек продаж;
* возможность создания кредитных и лизинговых продуктов;
* возможность блокировки и удаления продуктов при необходимости;
* возможность блокировки и удаления пользователей ролей администратор, менеджеров, брокеров и банковских работников;
* возможность создания и оформление кредитной и лизинговой заявки;
* возможность одобрения или отклонения кредитной или лизинговой заявки работником банка или лизинговой компании;
* возможность использования кредитного и лизингового калькулятора;
* возможность экспорта данных в формате *csv*;
* возможность экспорта данных по лизинговой заявке в формате *pdf*.

1.5 Вывод по разделу

Проанализировав существующие аналоги, можно сказать, что на сегодняшний день не существует системы, которая позволит просто и удобно создавать кредитные и лизинговые продукты, а также оформлять заявки на получение кредита или лизинга.

Необходимость экспорта данных в *CSV* формате, чтобы можно было без особых проблем и усилий импортировать их в систему управления финансами.

В разделе составлена постановка задачи, включающая обязательный набор функций и предполагаемый технологический стек разработки. По результатам исследования у создаваемого программного средства был обнаружено несколько аналогов, но нигде нет полного комплекса всех предполагаемых возможностей.

Суммируя все вышеперечисленное можно сказать, что работа, проделанная по аналитическому обзору литературы, является крепким фундаментом для разрабатываемого программного средства. На его основе можно преступать к проектированию и разработке модели базы данных, построению микросервиснной архитектуры, разработке дизайна клиентского приложения.

Для реализации серверной части приложения будут использоваться следующие технологии:

* платформа *Node*.*js* *Web* *API* с *REST*-подходам и микросервисной архитектурой, поскольку разработка клиентской части приложения и серверной на одном языке упрощают процесс разработки;
* в качестве базы данных будет использоваться *NoSQL*, база данных будет храниться в MongoDB;
* в качестве архитектуры серверной части приложения будет использоваться микросервисная, что позволит удобно организовать код приложения;
* форматом передаваемых данных станет *JSON*.

Для разработки клиентской части приложения существует множество различных библиотек, подходов, технологий. Некоторые из них были рассмотрены выше.

*React.js* предоставляет инкапсулированную и гибкую работу с *DOM*, самостоятельно отслеживает изменения в состоянии приложения и выполняет перерисовку лишь той части страницы, которая была изменена, что является несомненным плюсом для производительности

Клиентский код исполняется в браузере на стороне клиента. Для создания динамических страниц требуется *JavaScript*. Собственно, при разработке *React*-приложения так же не обойтись без *JavaScript*.

1. Проектирование программного средства

ФИО

Подпись

Дата

Лист

1

ДП 02.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С.*

Утверд.

*Пацей Н.В.*

2 Проектирование веб-приложения

Лит.

Листов

10

*БГТУ 74417058, 2020*

Программное средство, разрабатываемое в рамках данного дипломного проекта, в первую очередь, предназначено для помощи в оформлении кредитных и лизинговых заявок.

Приложение ориентировано для коммерческих предприятий, которые хотят просто и эффективно внедрить оформление кредитных и лизинговых продуктов.

Разработка подобного веб-приложения затрагивает достаточно много аспектов разработки программного обеспечения в целом.

Прежде чем приступить к проектированию веб-приложения, необходимо провести формализацию и стандартизацию данных и задач, являющихся необходимыми для всех этапов разработки.

Реализация проекта выполняется в несколько этапов:

* предварительные исследования;
* разработка технического задания;
* создание концепции дизайна сайта;
* программирование создание сайта;
* тестирование;
* устранение недостатков;
* сдача проекта заказчику.

Предварительные исследования – это одна из самых важных частей проектирования, поскольку ошибки на этом этапе приводят к значительному снижению ценности выполненной работы. В предварительные исследования входят:

* уточнение целей и задач;
* изучение бизнеса клиента, определение и анализ целевой аудитории;
* анализ решений, использованных конкурентами;
* составление календарного плана работ;
* формирование бюджета и рабочей группы.

Далее следует разработка и утверждение окончательного технического задания, выбор площадки размещения и соответственно формирование требований к аппаратной части проекта.

Разработка и согласование дизайна и разработка структуры сайта оказываются неразделимыми на этапе создания концепции дизайна сайта. Дизайн и структура очень взаимозависимы, поскольку изменения в объеме информации требуют изменения её представления.

## **2.1 Основные технические требования к разработке**

Целью данного дипломного проекта является разработка программного средства, представляющего собой веб-приложение. Для создания приложения, обеспечивающего создание кредитных и лизинговых продуктов и заявок, а также экспорт данных в формате *csv*.

Клиентская часть приложения должна:

* обеспечивать кроссплатформенность;
* обладать удобным интерфейсом;
* предоставлять *UI* для создания сертификатов;
* предоставлять *UI* для прохождения сертификации;
* быть легко поддерживаемым.

Серверная часть приложения должна:

* обеспечивать хранение данных;
* основываться на *REST*-архитектуре;
* выполнять генерацию сертификата в формате *csv*;
* обеспечивать легкую поддержку.

2.2 Проектирование базы данных

**2.2.1 Проектирование логической структуры базы данных**

Для реализации необходимо спроектировать базу данных для каждого микросервиса. Для их проектирования мы будем использовать *ORM Mongoose*. Так же будет использоваться система управления базами данных (СУБД) *Robo3Mongo*.

База данных – это совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами.

2.2.2 Контекст данных приложения

Для взаимодействия с базой данных приложению необходимо создать подключение к базе данных.

Использование *ODM* (*Object Document Mapping*) позволит ускорить процесс разработки, придает гибкость функциям, улучшает читаемость кода. Так же использование *ODM* исключает возможность внедрения *NoSQL*-инъекций. Под *NoSQL*-инъекцией понимается возможность получения сервером регулярного выражения или нативного кода *NoSQL*-запроса, который, например, могут удалить все данные из коллекций.

В качестве *ODM* используется *Mongoose* – документо-ориентированная технология доступа к данным, является *object*-*document* *mapping* (*ODM*) решением для *Node*.*js*.

В каждом разрабатываемом микросервисе должен содержаться контекст данных, который определит следующий перечень таблиц:

* пользователи (*users*);
* профили (*profiles*);
* роли (*roles*);
* кредитные заявки (*credit*\_*applications*);
* лизинговые заявки (*leasing*\_*applications*);
* путь к файлам (*file\_paths*);
* банки (*banks*);
* продукты (*products*);
* последний идентификатор для лизингового продукта (*leasing\_product\_last\_id*);
* точки продаж (*sale\_points*);
* связь точек продаж и продуктов (*point\_products*);
* последний идентификатор для точек продаж (*last\_id*).

При первичном запуске базы данных будут пустыми, что не позволит пользователям начать работу незамедлительно. Чтобы этого избежать, требуется после запуска Docker-контейнеров необходимо произвести первоначальную инициализацию. Для инициализации базы данных необходимо запустить скрипт *backup\_mongo.sh*. Данный скрипт запускает миграции, который заполняет некоторые таблицы предзаполненной информаций и создает суперадминистратора, который в дальнейшем создает остальных пользователей, точки продаж, владельцев продутков и сами продукты.

2.2.3 Модели базы данных

Для взаимодействия с базой данных (создание/получение/изменение/удаление) для *ODM* требуется описать сущность базы данных. Каждая сущность представляет собой одну коллекцию из базы данных, является ее отражением на приложение, тем самым позволяет взаимодействовать с хранимыми данными в базе данных.

На рисунке 2.4 представлен перечень моделей микроверсиса *auth*, хранимые в папке *models*.

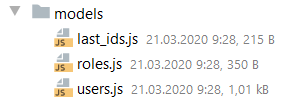


Рисунок 2.4 — Модели микросервиса auth

Важным условием является правильное формирование id (для наших документов), именно для этого необходимо было создать документ, который хранит в себе последний сгенерированный id нашим приложением. Это было сделано для быстрого распознания роли пользователя или типа заявки. Например, пользователи роли администратора будут иметь вот такую структуру id – 11xxxxx, где xxxxx – номер, сгенерированный системой.

На рисунке 2.5 представлены модели микросервиса *application.*

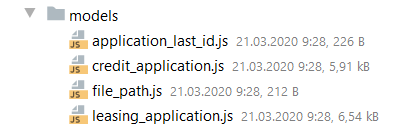


Рисунок 2.5 — Модели микросервиса application

На рисунке 2.6 представлены модели микросервиса banks.

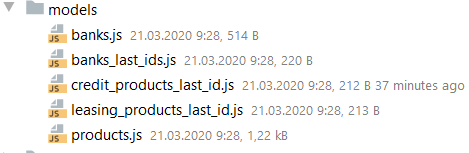


Рисунок 2.6 — Модели микросервиса application

На рисунке 2.7 представлены модели микросервиса profiles.



Рисунок 2.7 — Модели микросервиса profiles

На рисунке 2.8 представлены модели микросервиса sale\_points.

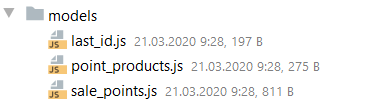


Рисунок 2.8 — Модели микросервиса sale\_points

Все модели представленные выше будет задействованы в приложении.

2.3 Проектирование веб-сервисов

Планируется, что разрабатываемое веб-приложение будет иметь пять ролей: суперадминистратор, администратор, менеджер, брокер, банковский сотрудник, пользовтель. Предусмотрена форма для создания пользователей и их входа в систему. Доступ к функционалу должен выдаваться в соответствии с ролью пользователя.

2.3.1 Разработка диаграммы вариантов использования

Приложение будет позволять создавать пользователей четырёх ролей:

* роль «Администратор»;
* роль «Менеджер»;
* роль «Брокер»;
* роль «Банковский сотрудник»;

Каждая из ролей имеет свои разрешения на доступ к той или иной функции.

Пользователь – данным актёром являются все пользователи системы вне зависимости от их ролей (кроме сотрудника банка). Он может выполнять следующие обобщённые функции:

– авторизация в системе;

– просмотр профилей остальных пользователей.

Суперадминистратор − данный актер имеет больше всего привилегий в системе. Он может выполнять следующие функции:

– управление абсолютно всеми пользователями программного средства;

– управление точками продаж;

– управление владельцами продуктов;

– управление кредитными и лизинговыми продуктами;

– управление поданными заявками на получение кредита/лизинга;

– прикрепление и скачивание документов, поданных вместе с заявками на получение кредита/лизинга;

– просмотр списка точек продаж;

– просмотр списка кредитных и лизинговых продуктов;

– просмотр списка владельцев продуктов;

– просмотр списка поданных заявок на получение кредита/лизинга.

Во всех функциях управления общими основными составляющими функциями являются:

– создание новой позиции в соответствующем списке;

– редактирование существующей в списке позиции;

– удаление позиции из списка;

– восстановление ранее удаленной позиции.

Администратор − данный актер может выполнять следующие функции:

– управление всеми пользователями (кроме суперадминистраторов)

– управление точками продаж;

– управление владельцами продуктов;

– управление кредитными и лизинговыми продуктами;

– управление поданными заявками на получение кредита/лизинга;

– прикрепление и скачивание документов, поданных вместе с заявками на получение кредита/лизинга;

– просмотр списка точек продаж;

– просмотр списка кредитных и лизинговых продуктов;

– просмотр списка владельцев продуктов;

– просмотр списка поданных заявок на получение кредита/лизинга.

Менеджер − данный актер имеет возможность всевозможного управления заявками на получение кредита или лизинга в системе. Это включает в себя следующие функции:

– управление поданными заявками на получение кредита/лизинга;

– прикрепление и скачивание документов, поданных вместе с заявками на получение кредита/лизинга;

– просмотр списка заявок на получение кредита/лизинга, созданных брокерами в подчинении определённого менеджера;

– изменение статусов заявок на получение кредита/лизинга;

– просмотр точек продаж;

– просмотр списка кредитных и лизинговых продуктов;

– просмотр списка владельцев продуктов.

Брокер − данный актер имеет возможность всевозможного управления заявками на получение кредита или лизинга, созданными им самим в системе. Это включает в себя следующие функции:

– управление поданными заявками на получение кредита/лизинга;

– прикрепление и скачивание документов, поданных вместе с заявками на получение кредита/лизинга;

– просмотр списка заявок на получение кредита/лизинга, созданных определённым брокером;

– изменение статусов заявок на получение кредита/лизинга;

– просмотр точек продаж;

– просмотр списка кредитных и лизинговых продуктов;

– просмотр списка владельцев продуктов.

Сотрудник владельца продукта − данный актер имеет возможность всевозможного управления продуктом в системе. Это включает в себя следующие функции:

– управление кредитными и лизинговыми продуктами;

– просмотр списка кредитных и лизинговых продуктов;

– изменение статусов заявок на получение кредита/лизинга;

– прикрепление документов, поданных вместе с заявками на получение кредита/лизинга;

– просмотр точек продаж;

– просмотр поданных заявок на получение кредита/лизинга.

Сотрудник банка − данный актер имеет возможность одобрения и отклонения заявок на получение кредита или лизинга. Это включает в себя следующие функции:

– одобрение заявок на получение кредита/лизинга;

– отклонение заявок на получение кредита/лизинга;

– авторизация в системе;

– просмотр поданных на получение кредита/лизинга заявок;

– скачивание прикрепленных к заявке на получение кредита/лизинга документов.

2.3.2 Структура приложения

Приложение разделяется на две части: серверную и клиентскую. Каждая из них представлена своей архитектурой и своими технологиями, своими библиотеками и фреймворками.

Серверная часть будет реализована с использованием *Node.js*, и будет представлять собой API, при помощи которого клиентская часть может получить необходимые данные. В основе серверной части лежит микросервисная архитектура.

Каждый микросервис содержит в себе:

* папку *server*;
* *dockerfile* (для запуска *docker*-контейнера);
* *.env.development* файлы, которые содержит в себе переменные среды (для корректной работы *docker*-контейнера);
* .*dockerignore*;
* *ecosystem*.*config*.*js* (используемый для старта микросервиса внутри *docker*-контейнера);
* вспомогательные скрипты, которые могут быть использованы по назначению.

Модуль *server* содержит в себе:

* модели данных (для базы данных);
* данные для первоначальной инициализации базы данных;
* папка *migrations* содержит в себе все миграции, которые выполнялись над схемой базы данных;
* папка *queries* содержит в себе абстрактные методы для заботы с моделями;
* папка *node\_modules*, в которой содержатся зависимости клиентской части приложения, устанавливаемые при помощи пакетного менеджера npm;
* таблица маршрутов представлена в файле *routes.js*;
* папка *schemas*, содержит в себе валидационные правила принимаемых данных и данных отдаваемых клиенту;
* папка *features*, в которой предоставлен вспомогательный узконаправленный функционал, содержат в себе основную логику по работе с данными;
* точка входа в приложение, которая содержится в файле *index.js*. Под точкой входа понимается функция, с которой начинается выполнение программы, а в случае рассматриваемого приложения — производится первоначальная настройка сервера;
* в папке *app*\_*init* подключается вся необходимая информация, для запуска микросервиса.
* в папках *utils* и *lib* представлен вспомогательные функции:
* папка *configs*, содержащая конфигурационные файлы для различных окружений (*development* и *production*);
* файл *package*.*json* содержит в себе перечень модулей, которые используются приложением. *Npm* при установке модулей использует этот файл;
* папка *services* содержит в себе методы для общения с другими микросервисами.

Структура микросерсиса *applications* представлена на рисунке 2.9.

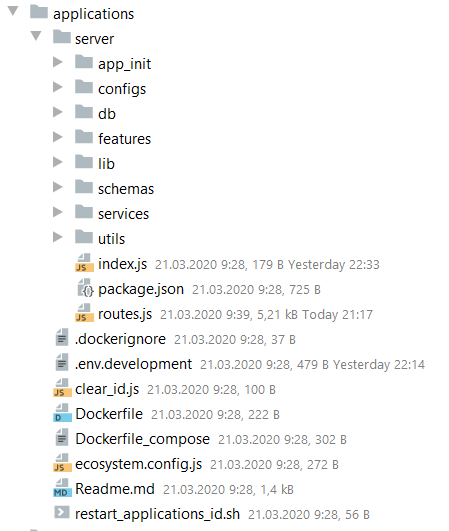


Рисунок 2.9 — Структура микросерсиса applications

Клиентская часть состоит из папок *src* и *public*.

В папке *public* содержит в себе иконку клиентской части и базовый html, который содержит себе все приложение.

Папка *src* содержит в свою очередь:

* папка *actions* содержит в себе описание действий нашего приложения;
* папка *components* описывает компоненты приложения. Под компонентами понимается класс, унаследованный от *React.Component*, который производит непосредственную отрисовку элементов. Например, это может быть какая-либо форма (регистрации или входа), или самый обычный *input*. Так же компоненты могут содержать в себе логику для валидации и прочее;
* папка *node\_modules*, в которой содержатся зависимости клиентской части приложения, устанавливаемые при помощи пакетного менеджера *npm*;
* папка *constant* содержит в себе перечень переменных, которые используются для различных целей;
* папка *container* содержит в себе фрагменты отрисовываемые на странице;
* папка *images* содержит изображения системы;
* в папке *reducer* описаны редьюсеры приложения;
* в папке *style* представлены стили отрисовки компонентов;
* в папке *utils* содержаться вспомогательные функции;
* файл *package*.*json* содержит в себе перечень модулей, которые используются приложением. *Npm* при установке модулей использует этот файл;
* в файле *Api*.*js* описана прослойка для обработки ответов;
* файл *index*.*js* содержит инициализацию всего проекта.

Структура клиентской части представлена на рисунке 2.9.

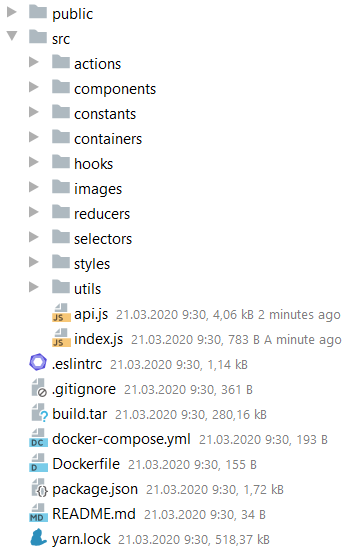


Рисунок 2.8 — Структура клиентской части веб-приложения

Так же в проекте есть файл .eslintrc, который описывает правила оформления и форматирования кода.

2.4 Вывод

В данном разделе были подробно описаны шаги проектирования веб-приложения для оформления кредитных и лизинговых продуктов и заявок.

После выбора базы данных и основных технологий для разработки серверной и клиентской части приложения.

В данном разделе была определена логическая структура базы данных приложения, представлена общая схема таблиц и связей между ними. Были рассмотрены серверная и клиентская части приложения. Подводя итог, можно сказать, что в данном разделе была произведена подготовка к реализации веб-приложения для оформления кредитных и лизинговых заявок и продуктов.

3 Реализация программного средства

Изм.

№ докум.

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

Куванова О.Р.

Пров.

Шиман Д.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

3 Реализация программного средства

Лит.

Листов

15

74417058, 2020

У

Разрабатываемое приложение включает в себя две части: серверную и клиентскую. Для разработки каждой из них будет использован свой набор технологий, включающий различные библиотеки, языки программирования.

3.1 Разработка клиентской части приложения

Для реализации клиентской части приложения используется библиотека React. Данная библиотека позволяет реализовывать большие и легко масштабируемые приложения. В данном разделе будут рассмотрены элементы, расположенные в корневой папке приложения.

Одним из основных элементов приложения является шаблон, описанный с помощью *HTML*-разметки в файле *index.html* представлен в листинге 3.1. В нём могут быть задействованы различные ссылки модули. Данный файл первым отправляется на сторону клиента, после чего скачивается содержимое связанных файлов.

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Dietify</title>

<base href="/">

<meta name="viewport" content=" maximum-scale=1.0, user-scalable=no">

<link rel="icon" type="image/ico" href="favicon.ico?v=1.0">

<link rel="stylesheet" href="https://maxcdnpcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"

integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<script type="text/javascript" src="https://js.stripe.com/v3/" async></script>

</head>

<body>

<scouted>Loading...</scouted>

</body>

</html>

Листинг 3.1 — Код стартовой страницы приложения

Библиотека *React* построена вокруг концепции компонентов. Компоненты – это небольшие части пользовательского интерфейса, которые можно использовать повторно. React решает задачи уровня представления (*view*), а из этого следует, что он придерживается шаблона *MVVM* (*Model*-*View*-*ViewModel*). Так же для построение правильного взаимодействия компонентов в проекте используется паттерн проектирования *Flux*.

Паттерн *MVVM* (*Model*-*View*-*ViewModel*) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения.

*MVVM* состоит из трех компонентов: модели (*Model*), модели представления (*ViewModel*) и представления (*View*) [14].

*Flux*, в свою очередь, состоит из четырёх компонентов: создатели действий (*Actions*), диспетчер (*Dispatcher*), хранилище (*Store*) и представления (*View*) [15]. Основной особенность данного паттерна является односторонняя направленность передачи данных между компонентами архитектуры. Что в свою очередь накладывает ограничения на поток данных, в частности, исключает возможность обновления состояния компонентов самими собой. Такой подход делает поток данных предсказуемым и позволяет легче проследить возможные исключения в процессе разработки веб-клиента. Так же основная концепция Flux состоит в том, что хранилище – это единственная сущность в приложении, которая знает, как изменить данные.

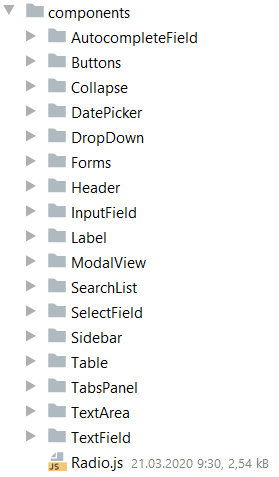


Рисунок 3.1 — Компоненты приложения

Таким образом, практически каждая часть страницы представляет из себя независимый компонент, что позволяет их переиспользовать в дальнейшем.

3.1.1 Маршрутизация, используемая в приложении

Под маршрутизацией (роутингом) понимается сопоставление компонентов приложения определенным маршрутам. Под маршрутом понимается значение, отображаемое в адресной строке браузера.

В разрабатываемом приложении для удобства перехлда между сущностями предусмотрены различные маршруты. На рисунке 3.2 представлен код роутера приложени.



Рисунок 3.2 — Компонент роутера приложения

Роутеры приложения, представленные на рисунке 3.2, показывает перечень доступных роутеров для неавторизованных пользователей и показан компонент, который вызывается при его авторизации. В случае, если пользователь пытается перейти по маршруту, который не существует или к которому у него нет доступа, то он будет перенаправлен в компонент, соответствующий корневому маршруту (<*Redirect* *to*='/' />).

Маршрут представляет собой соответствие ссылки определенному компоненту. Пример маршрутов представлен на рисунки 3.3.



Рисунок 3.3 — Код пример маршрутов

Из рисунка 3.3 видно, что использую маршрут /*proposals* можно получить доступ к компоненту *Proposals*. Такой маршрут строится исходя из значения атрибута path. Каждый маршрут (<*Route* … />) допускает использовать вложенную структуру, что позволяет строить сложные маршрутизаторы.

В данном приложении существуют следующие роутеры:

* вход (*login*);
* выход (*logout*);
* просмотр списка пользователей (*users*);
* просмотр информации о пользователях (*user*/:*id*);
* создание пользователя (*user*);
* просмотр списка продуктов (*products*);
* просмотр информации о продукте (*product*/:*id*);
* создание продукта (*product*);
* просмотр списка заявок (*proposals*);
* просмотр информации о заявке (*proposal*/:*id*/:*type*);
* создание заявки (*proposals*/*create*);
* просмотр списка точек продаж (*salepoints*);
* просмотр информации о точке продаж (*salepoints*/:*id*);
* создание точки продаж (*salepoint*/*create*);
* просмотр списка владельцев продуктов (*products*);
* просмотр информации о владельце продукта (*products*/:*id*);
* создание владельца продуктов (*products*/*create*).

3.1.2 Компоненты приложения

Под компонентами приложения понимаются классы, в которых происходит непосредственная отрисовка (*render*) элементов приложения. На рисунке 3.4 представлен фрагмент кода класса контейнера *Users*.



Рисунок 3.4 — Код пример компонента

Рассмотри подробнее содержимое класса контейнера, представленного на рисунке 3.4. При помощи метода *setState*, задается текущие состояние компонента. При помощи *this.props* задаются значения для компонента, которые были переданы из родительского компонента.

В приложение реализован следующий перечень контейнеров:

* администратор (*Admins*);
* вход (*Auth*);
* брокер (*Brokers*);
* калькулятор (*Calculator*);
* менеджер (*Managers*);
* меню (*Menu*);
* владельцы продуктов (*Owners*);
* продукты (*Products*);
* заявки (*Proposals*);
* точки продаж (*SalePoints*);
* пользователи (*Users*).

3.1.3 Действия (actions) для манипуляции данными приложения

Под действиями *React.js* понимаются объекты, с помощью которых становится возможным манипулировать состоянием приложения.

Действие представляет собой объект, содержащий в себе три поля: *endpoint*, *method*, *types*. В поле *endpoint* указывается маршрут к которому веб-клиент обращается к веб-серверу. В поле *method* указывается *HTTP*-метод для запроса. В качестве type используются уникальные строковые значения, позволяющие однозначно задать создаваемое действие, а также однозначно определить его при получении.

Пример распространения действия представлен на рисунке 3.5.

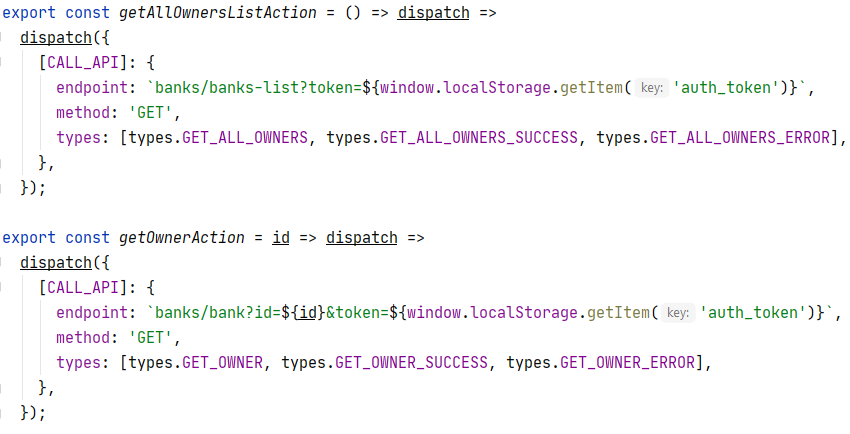


Рисунок 3.5 — Код пример действия

Для каждой сущности приложения описан свой перечень функций, позволяющих выполнять запросы к серверному *API*.

3.1.4 Сервисы и утилиты, реализованные в приложении

Приложение располагает определенным перечнем сервисов и утилит, назначение которых различно. Рассмотрим некоторые из них.

Утилиты располагаются в папке *utils*. Данная папка насчитывает 7 групп утилит. Рассмотрим их подробнее:

* утилиты для работы с датой. Содержится в файле *parseDate*.*js*;
* утилиты для работы с числами. Содержится в файле *parseNumber*.*js*;
* утилиты для вычислений в кредитной и лизинговой заявках. Содержится в файле *creditCalculation*.*js*;
* утилиты для работы с адресами. Содержится в файле *addressHandler*.*js*;
* утилиты для работы с информацией о точках продаж. Содержится в файле *getSalePointInfo*.*js*;
* утилиты для форматирования данных с форм. Содержится в файле *reduxFormHelper*.*js*;
* утилиты для форматирования данных пред отправкой запроса. Содержится в файле *utils*.*js*;

3.1.5 Зависимости клиентской части приложения

Под зависимостями приложения понимаются модули, инсталлируемые из сторонних источников. В *JavaScript* для установки зависимостей используется yarn, а для их описания используется файл *package*.*json*.

Файл *package*.*json* описывает не только зависимости приложения, а также название, версии, описание и прочее. Как можно догадаться, форматом представления для данного файла является формат *JSON*. На рисунке 3.6 представлен фрагмент кода данного файла.

Фрагмент кода файла *package*.*json*, изображенный на рисунке 3.6, представлен в формате JSON. Из данного листинга видно следующее:

* описание данного проекта (поле *description*);
* имя проекта (поле *name*);
* версия проекта (поле *version*);
* поле, которое указывает на приватность репозитория (поле *private*);
* зависимости (раздел *dependencies*) – список модулей, используемых при работе приложения;
* данные по удаленному репозиторию (поле *repository*);
* объект с конфигурационными данными (поле *config*);
* зависимости, используемые при разработке (раздел *devDependencies*).

Также данный файл может содержать команды для запуска приложения (поле *scripts*).

После каждого модуля следует его версия. Так, например, в примере выше для модуля *react*-*redux* версия 7.1.0 или выше.



Рисунок 3.6 — Фрагмент кода файла package.json

3.2 Реализация серверной части

Для реализации серверной части приложения будет использоваться *Node*.*js* *API*. Это программная платформа, позволяющая создавать различного рода веб-приложения. Постфикс *API* означает, что серверная часть будет использоваться исключительно для оперирования данными, и не будет иметь отношения к отрисовке клиентской части.

В данном разделе будут рассмотрены элементы, зависимости и принцип работы приложения.

3.2.1 Маршрутизаторы для обработки запросов, авторизация

Под маршрутизаторами понимаются объект класса, предназначенный для перехвата запросов. Фрагмент кода маршрутизаторов для работы с заявками представлены на рисунке 3.7.

Рассмотрим рисунок 3.7 подробнее. Маршрутизатор использует функционал из сервисов для работы с заявками. В рассматриваемом примере методы являются асинхронными.



Рисунок 3.7 — Фрагмент кода маршрутизаторов

Маршрутизатор имеет корневой *URI*, в рассматриваемом примере это *api*/*user*. Доступ к дочерним элементам может быть по *URI* маршрутизатора, либо же дополнен чем-либо. Так, например, метод *getApplication* доступен по URI *api*/*application*/{*id*}. Постфикс {*id*} добавляется вследствие аргумента, переданного в *HttpGet*.

Всего в приложении реализовано 58 маршрутизаторов. Вот перечень некоторых из них: *application*, *applications*-*list*, *file*, *address*, *get*-*education*, *get*-*form*-*ownership*, *get*-*industries*, *get*-*nationality*, *get*-*count*-*employees*, get-*period*-*activity*, *get*-*social*-*status*, *login*, *broker*-*token*, *logout*, *change*-*password*, *password*-*form*, *new*-*password*, *bank*, *banks*-*list*, *product*, *products*-*list*, *user*, *users*-*list*, *point*, *points*-*list*.

3.2.2 Сервисы, реализованные в приложении

Под сервисами понимается такой слой приложения, назначением которого является выполнение какой-либо логики. Маршрутизаторы лишь должны принять запрос, а затем передать его дальше по цепочке соответствующему контроллеру.

Каждый из реализованных контроллеров содержит методы, которые выполняют свое узконаправленное назначение.

Фрагмент кода сервиса для авторизации представлен на рисунке 3.8.



Рисунок 3.8 — Фрагмент кода сервиса

Рассмотрим рисунок 4.12 подробнее. Сервис представленный на рисунке представлен метод Post, который позволяет пользователю авторизоваться. Данный сервис генерирует токен. И выкидывает ошибку, если пользователь не найден в базе данных.

Всего в приложении реализовано 58 сервисов. Вот перечень некоторых из них: *handleAuth*, *checkToken*, *user*, *getBrokerToken*, *logout*, *changePassword*, *getForm*, *setPassword*, *addApplication*, *updateApplication*, *getApplication*, *deleteApplication*, *getApplicationsList*, *getFile*, *getList*, *addBank*, *getBank*, *updateBank*, *deleteBank*, *getBanksList*, *getProductBanks*, *getProductsBanksInfo*, *getBankAvailability*, *addProduct*, *getProduct*, *updateProduct*, *deleteProduct*, *getProductsList*, *getActiveProducts*, *getPointProducts*, *addPointProducts*, *deletePointProducts*, *addPoint*, *getPoint*, *updatePoint*, *deletePoint*, *getPointsList*, *updatePoints*, *pointsByCurator*, *managersByPoints*, *contractorByPoints*, *getUser*, *addUser*, *updateUser*, *deleteUser*, *getUsersList*, *getBanksUsers*, getCurator, *getBrokersByManagers*, *getApplication*, *getConsent*, *changePassword*, *newUser*, *usersList*, *pointsList*, *productsList*, *applicationsList*, *banksList*, employeesList, *getAddress*.

3.2.3 Модели данных приложения

Под моделью данных приложения понимается класс, описывающий одну таблицу базы данных. Фрагмент модели данных представлен на рисунке.

На рисунке 3.9 представлена схема для коллекции users.

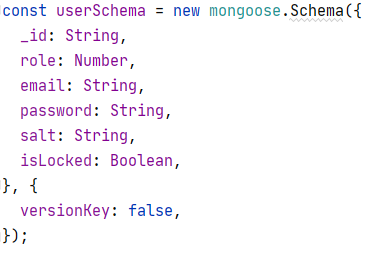


Рисунок 3.9 — Фрагмент кода схемы

Всего в приложении реализовано 15 моделей: *users*, *roles*, *last*\_*id*, *application*\_*last*\_*id*, *credit*\_*applications*, *file*\_*paths*, *leasing*\_*applications*, *banks*, *bank*\_*last*\_*ids*, *leasing*\_*product*\_*last*\_*id*, *credit*\_*product*\_*last*\_*id*, *products*, *profile*, *last*\_*id*, *point*\_*products*, *sale*\_*points*.

3.2.3 Модели данных приложения

Под моделями приложения понимается файл, который содержит методы-обёртки для контекста данных приложения. С каждой моделью может быть связана лишь одна сущность. Таким образом, модель представляет собой API для работы с одной конкретной таблицей базы данных. Под сущностью понимается файл, описывающий одну таблицу базы данных.

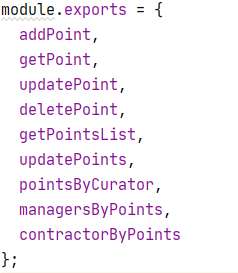


Рисунок 3.10 — Фрагмент кода модели

Всего в приложении реализовано 15 моделей: *users*, *roles*, *last*\_*id*, *application*\_*last*\_*id*, *credit*\_*applications*, *file*\_*paths*, *leasing*\_*applications*, *banks*, *bank*\_*last*\_*ids*, *leasing*\_*product*\_*last*\_*id*, *credit*\_*product*\_*last*\_*id*, *products*, *profile*, *last*\_*id*, *point*\_*products*, *sale\_points.*

3.2.5 Поставщики данных (провайдеры) приложения

Под провайдерами данных понимаются такие классы приложения, которые, например, предоставляют некоторый функционал по конвертации единиц измерения, содержат некоторый перечень констант, либо выполняют некоторую часть работы при генерации данных.

Некоторые разработанных провайдеров: провайдеры по генерированию документов в формте *CSV*, провайдеры для генерации документов в формате *PDF*, провайдет для отправки *email*-сообщений.

3.3 Вывод по разделу

В данной главе были описаны элементы, из которых состоит серверная и клиентская часть разработанного приложения.

Для серверной части был описана архитектура приложения. Были описаны элементы, из которых она состоит:

* модели данных;
* сервисы;
* маршрутизаторы.

Были даны примеры и определения данным составляющим серверной части приложения. Описан принцип их работы. Также были определены прочие составные элементы серверной части приложения, например, провайдеры данных.

В данной главе были описаны модели данных, что позволяет сопоставить результаты проектирования базы данных приложения (глава 2) с реализацией в серверной части приложения.

Благодаря рассмотренным элементам реализованного приложения можно представить общий процесс работы. Созданные схемы призваны упростить понимание данного процесса.

Для клиентской части показан визуальный интерфейс приложения, показана локализация, реализованная в приложении, на примере формы входа. Описаны возможности клиентской части приложения. Описаны элементы, из которых состоит клиентская часть приложения:

* контейнеры;
* компоненты;
* хранилище.

Даны определения данным элементам. Был описан принцип работы с *API*. Определены используемые зависимости.

Была представлена общая схема работы приложения. Был сгенерирован и отображен документ, созданный разработанным приложением.

4 Тестирование приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

ШиманюД.В

Н. контр.

*Рвжанкова А.С*.

Утверд.

*Пацей Н.В.*

4 Тестирование и анализ полученных результатов

Лит.

Листов

7

*74417058, 2020*

Тестирование программного обеспечения является важным этапом разработки программного средства, позволяющий обнаружить, исправить или предотвратить возможные ошибки в работе приложения. Представляет собой проверку на соответствие между реальным и ожидаемыми поведением веб-сервиса.

Необходимо протестировать корректность функционирования базовых операций, доступных пользователям. Также стоит удостовериться, что при заведомо неверных действиях система предотвратит их и оповестит об ошибке. Для исключения возможности возникновения ошибок необходимо протестировать каждый из компонентов системы в отдельности.

4.1 Тестирование серверной части

Для тестирования сервера будет использоваться программное средство Postman. Данное программное средство позволяет управлять заголовками и телом запроса, эмулируя запросы клиента. В качестве формата передачи тела запроса будет использоваться формат JSON.

4.1.1 Аутентификация пользователя

Одним из основных методов серверной части является метод создания токена аутентификации. Для получения токена необходимо отправить в теле запроса emil и пароль.



Рисунок 4.1 — Получение токена

При успешном выполнении в ответ мы получаем токен, id пользователя и роль.

4.1.1 Создание нового аккаунта пользователя

Для *Web API* сервера важно предоставлять информацию о недостающих полях в теле запроса, чтобы разработчик мог быстро добавить их в запрос. Пример такой ошибки продемонстрирован на рисунке 4.2.

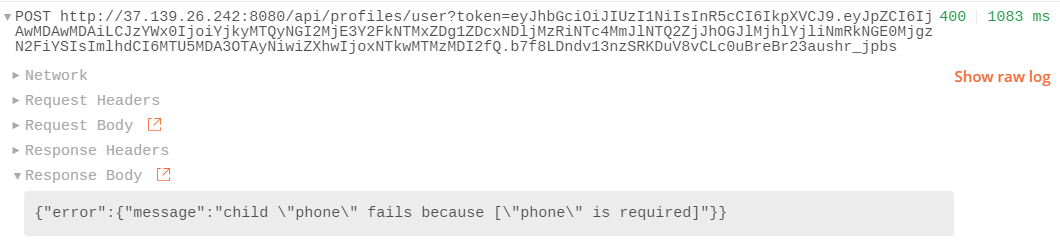


Рисунок 4.2 — Валидация недостающих полей запроса

Так же в запросе поле *e-mail* должно иметь валидацию на корректный формат *e-mail* адреса во избежание регистрации в системе пользователя с неверным адресом. Это может привести к невозможности восстановления пароля по *e-mail*.

Так же следует убедиться, что для всех ошибок валидации возвращается статус код 400, что свидетельствует об ошибках на стороне клиента, а в случае, если сервер возвращает статус код 500, это значит, что это ошибка не обработана.

4.1.2 Доступ к запрещенным ресурсам

Разработанный веб-сервер должен проверять расширение на доступ к ресурсам. Для проверки разрешений используется токен, который хранит в себе информацию об роли пользователя, его идентификатор. Токен выдается при входе пользователем в систему.

При отсутствии токена сервер должен вернуть соответствующую валидациооную ошибку и статус 401.

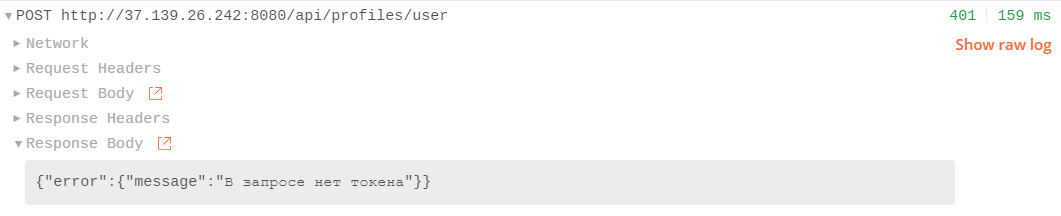


Рисунок 4.3 — Валидация отсутствия токена в запросе

Так же при попытке доступа с невалидным токеном веб-сервер вернет соответствующие валидационное сообщение и статус код 401.

4.2 Тестирование клиентской части

Тестирование клиентской части будет произведено с помощтю тест кейсов. Тест-кейс – набор тестовых входных данных, условий выполнения и ожидаемых результатов, разработанных с целью проверки некоторого пути выполнения программы или проверки соответствия некоторому требованию.

Тест-кейс может включать поля:

* идентификатор тест-кейса;
* связанные с тест-кейсом требования;
* приоритет тест-кейса;
* модуль и подмодуль приложения, к которым относится тест-кейс;
* шаги для выполнения тест-кейса;
* исходные данные, необходимые для тест-кейса;
* описание ожидаемых результатов;
* поле статуса выполнения теста;
* автор тест-кейса;
* время выполнения последнего тест-кейса;
* последний полученный результат;
* связанный с тест-кейсом дефект.

Достоинствами тест-кейсов являются:

* предоставление тестировщикам структурированного системного подхода, что снижает вероятность пропуска ошибок;
* один из способов тестирования проектной документации;
* ускорение процесса регрессионного тестирования;
* возможность отслеживания статистики проводимого тестирования.

По ожидаемому результату тест-кейсы делятся на позитивные и негативные. Позитивный тест-кейс использует только корректные данные и проверяет, что программное средство правильно выполнило вызываемую функцию. Негативный тест-кейс оперирует некорректными данными и проверяет программное средство на исключительные ситуации.

Таблица 4.1 — Тест-кейсы

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Результат |
| **Наличие страницы и формы для авторизации**  1. Открыть страницу для авторизации http://creditbroker.com  2. Проверить элементы на странице | 1. Открывается страница авторизации.  2. Отображаются следующие элементы:  − форма авторизации (заголовок, дополнительный текст, поле для ввода email’a, поле для ввода пароля, кнопка, отправляющая форму, кнопка, восстанавливающая пароль) | Успех |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Результат |
| **Ввод имени пользователя**  1. Открыть страницу для авторизации http://creditbroker.com  2. Проверить элементы на странице  3. Кликнуть на поле «Password»  4. Ввести любую информацию в поле ввода | 1. Открывается страница авторизации.  2. Отображаются следующие элементы:  − форма авторизации (заголовок, дополнительный текст, поле для ввода email’a, поле для ввода пароля, кнопка, отправляющая форму, кнопка, восстанавливающая пароль).  3. Поле становится активным:  − текст-подсказка «Password» заменяется зашифрованным введённым символом;  − курсор – активный.  4. В поле ввода отображается введенная информация в виде точек. | Успех |
| **Неверный ввод пароля**  1. Открыть страницу для авторизации http://creditbroker.com  2. Проверить элементы на странице  3. Кликнуть на поле «Password»  4. Ввести данные, несоответствующие Email’у, в поле ввода  5. Нажать кнопку «Войти» | 1. Открывается страница авторизации.  2. Отображаются следующие элементы:  − форма авторизации (заголовок, дополнительный текст, поле для ввода email’a, поле для ввода пароля, кнопка, отправляющая форму, кнопка, восстанавливающая пароль).  3. Поле становится активным:  − текст-подсказка «Password» заменяется зашифрованными введёнными символами;  − курсор – активный.  4. В поле ввода отображается введенная информация в виде точек.  5. Под полем появляется фраза «Something bad happened» красного цвета. | Успех |
| **Авторизация**  1. Открыть страницу для авторизации http://creditbroker.com  2. Проверить элементы на странице  3. Ввести валидную существующую информацию в поля «Email» и «Password» | 1. Открывается страница авторизации.  2. Отображаются следующие элементы:  − форма авторизации (заголовок, дополнительный текст, поле для ввода email’a, поле для ввода пароля, кнопка, отправляющая форму, кнопка, восстанавливающая пароль).  3. В полях «Email» и «Password» отображается введенная информация. | Успех |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| 4. Кликнуть кнопку «Войти» | 4. Отображается страница со списком заявок. |  |
| **Создание нового администратора**  1. Открыть страницу для просмотра пользователей http://creditbroker.com  2. Проверить элементы на странице  3. Ввести валидную информацию в обязательные поля  4. Кликнуть кнопку «Создать» | 1. Открывается страница создания нового пользователя.  2. Отображаются следующие элементы:  − навигационное меню;  − форма создания нового пользователя (заголовок, обязательные для заполнения поля, поле с заполненной датой создания пользователя, поле для заполнения примечания, кнопка добавления нового пользователя, кнопка, отменяющая создание пользователя, кнопка, очищающая введённые данные).  3. Все обязательные поля заполнены.  4. Отображается нотификация с текстом «Новый пользователь создан! Email с логином и паролем был отправлен пользователю.» | Успех |
| **Экспорт списка пользователей в .CSV**  1. Открыть страницу для просмотра пользователей http://creditbroker.com  2. Проверить элементы на странице  3. Кликнуть кнопку «Экспорт в .CSV» | 1. Открывается страница создания нового пользователя.  2. Отображаются следующие элементы:  − навигационное меню;  − форма отображения списка пользователей (вкладки с разными ролями пользователей, заголовок, кнопка создания нового пользователя, кнопка экспорта в .CSV, список пользователей).  3. Отображается сообщение о загрузке документа в файловое хранилище. | Успех |
| **Создание нового кредитного продукта**  1. Открыть страницу для просмотра пользователей [http://creditbroker.com /products](http://creditbroker.com /products )  2. Проверить элементы на странице | 1. Открывается страница создания нового пользователя.  2. Отображаются следующие элементы:  − навигационное меню;  − форма создания нового кредитного продукта (заголовок, обязательные для заполнения поля и флажки, необязательные поля для | Успех |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Результат |
| 3. Ввести валидную информацию в обязательные поля  4. Кликнуть кнопку «Создать» | заполнения, кнопка добавления нового кредитного продукта, кнопка, отменяющая создание нового кредитного продукта).  3. Все обязательные поля заполнены.  4. Отображается нотификация с текстом «Новый кредитный продукт создан в системе» |  |
| **Печать таблицы с графиками и суммами платежей**  1. Открыть страницу для просмотра пользователей  [http://creditbroker.com /proposals](http://localhost:3000/proposals%20)  2. Проверить элементы на странице | 1. Открывается страница администрирования заявок.  2. Отображаются следующие элементы:  − навигационное меню;  − форма со списком кредитных заявок (вкладки с разными ролями пользователей, заголовок, создающая заявку кнопка, скачивающая данные кнопка, применяющая фильтры кнопка, отменяющая фильтры кнопка, выпадающие списки для настройки фильтров, поисковая строка, календарь, таблица с данными о заявках и их статусами, элемент пагинации). | Успех |

4.4 Вывод по разделу

Приложение было протестировано с помощью юнит-тестирования, а также мануально. Исходя из результатов тестирования, которое было проведено выше, можно сделать выводы, что приложение показало высокую отказоустойчивость и масштабируемость, что позволит ему без проблем выполнять свои функции в реальной среде. Каждый тип тестирования имеет как плюсы, так и минусы. Приложение на протяжении всей разработки приложения необходимо производить поддержку тестовых сценариев. Приложение допускается к использованию, только в том случае если все тестовые сценарии важного по приоритету функционала, пройдены успешно.

5 Руководство по установке и использованию программного обеспечения

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С.*

Утверд.

*Пацей Н.В.*

5 Руководство по установке и использованию программного обеспечения

Лит.

Листов

14

*74417058, 2020*

Каждое разработанное приложение должно иметь инструкцию пользователя, для того чтобы клиент приложения мог начать работу с приложением сразу без каких-либо сложностей. Инструкция должна быть написана простым языком в сопровождении графических материалов, которые будут показывать, что именно должно происходить, выполнив то или иное действие.

Так же каждое приложение имеет минимальные системные требования, которыми должен обладать сервер, на котором будет развертываться данный сервер. Также сервер должен содержать некоторый набор предустановленных библиотек и программ, для начала работы с приложением.

5.1 Системные требования оборудования

Данное приложение написано посредством Node.js для серверной части, и с использованием библиотеки React.js для клиентской. Таким образом сервер на котором будет разворачиваться приложение не имеет каких-либо ограничений по установленной операционной системе, так как данные технологии являются кроссплатформенными.

Для быстрой работы приложения, сервер должен содержать не менее четырех гигабайт оперативной памяти. Количество ядер сервера должно быть не менее четырех.

Таким образом можно заметить, что для полноценной работы приложения, сервер на который будет производится развертывание не должен обладать высокими системными требованиями, что позволит использовать данное приложение и на слабых компьютерах.

5.2 Необходимое программное обеспечение и библиотеки

После указания минимальных системных требования приложения, следующим этапом является определения списка необходимого для успешного запуска приложения и его дальнейшей работы. Серверная часть приложения реализована при помощи Node.js, а это значит, что для установки сервера, на компьютере должен быть установлен Node.js. Клиентская часть реализована при помощи библиотеки React.js . Для установки клиентской части так же необходимо произвести установку Node.js с менеджером пакетов NPM. Он необходим для установки зависимостей приложения перед его запуском.

Для сервера базы данных, на котором будет храниться информация необходимая для работы приложения, был выбран MongoDB.

Запуск сервера и базы данных происходит при помощи запуска Docker-контейнеров. Таким образом необходимо установить docker и docker-compose. Такой запуск приложение обеспечивает полную кроссплатформенность и избавляет нас от установки некоторых программных модулей.

Таким образом необходимый перечень программного обеспечения, который должен быть установлен перед началом развертывания приложения на сервер, состоит из следующих программ:

* Node JS;
* React.js;
* Docker;
* Docker-compose.

Как можно заметить список предустановленных программ не является большим, и подготовка к началу развертывания приложения не займет много времени.

5.3 Подготовка инфраструктуры и развертывание приложения

После определения минимальных системных требований и необходимого программного обеспечения, следующим этапом является подготовка необходимой инфраструктуры перед запуском приложения.

Первое, что необходимо сделать, это настроить операционную систему для развёртывания серверной части приложения. Для этого необходимо скачать с официального сайта Node.js последнюю LTS версию.

Последующим этапом является подготовка запуск docker-контейнеров. Для этого необходимо выполнить команду в терминале для запуска контейнеров.

Если развертывание docker-контейнеров было успешно, тозапуск сервера является успешным.

Для развертывания клиентской части необходимо поочередно перейти из командной строки в папку с пользовательским интерфейсом и выполнить установку необходимых пакетов посредством менеджера пакетов NPM.

NPM (аббр. node package manager) – это стандартный менеджер пакетов, автоматически устанавливающийся вместе с Node.js. Он используется для скачивания пакетов из облачного сервера npm, либо для загрузки пакетов на локальный сервер.

В NPM используются следующие понятия:

* Пакет (Package) – базовая единица, которой управляет NPM как единым целым. Может содержать любое количество файлов и кода. NPM позволяет устанавливать пакеты, обновлять или удалять.
* Реестр (Registry) – хранилище пакетов NPM. Каждый желающий может опубликовать пакет в npm registry, потратив буквально минуту, а остальные смогут его использовать. В хранилище на текущий момент сотни тысяч пакетов и их количество стремительно растет. Исходный код пакетов, как правило, хранится на гитхабе. Несмотря на это, пакеты в npm никак не связаны с git и github.

Файл package.json содержит в себе информацию о вашем приложении: название, версия, зависимости и тому подобное. Любая директория, в которой есть этот файл, интерпретируется как Node.js-пакет, даже если вы не собираетесь публиковать его.  
Способ использования файла package.json зависит от того, собираетесь ли вы скачивать пакет или публиковать его.

Если вы хотите скачать пакет вручную, вам необязательно использовать для этого package.json. Вы можете выполнить в терминале команду npm install с названием нужного пакета в качестве аргумента команды, и пакет будет автоматически скачан в текущую директорию.

Результат выполнения установки пакетов, представлен на рисунке 5.1.

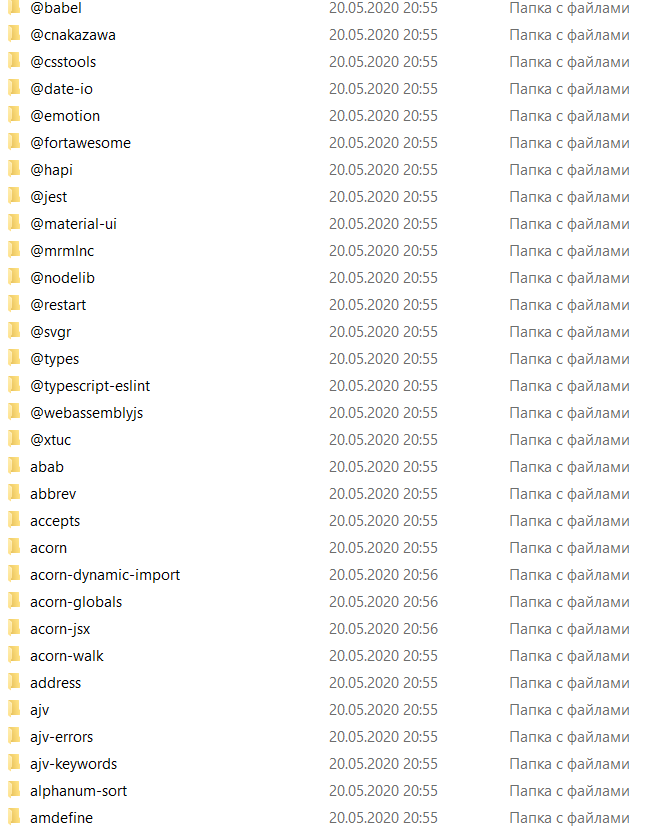


Рисунок 5.1 — Результат выполнения установки пакетов

Как видим результат установки пакетов, является набор скаченных библиотек на сервер в папку node modules.

После подготовки инфраструктуры, следующим этап – это установка и запуск приложения на сервере.

Для этого необходимо перейти из командной строки в папку с клиентской частью и выполнить запуск приложения посредством менеджера пакетов NPM.

5.4 Инструкция пользователя

В данном разделе будет представлена основная информация по использованию данного программного средства.

Для работы с программным средством необходим браузер, так как веб-приложение разворачивается на сервере и доступ к ее функционалу осуществляется через пользовательский веб-интерфейс из браузера любого цифрового устройства с доступом к сети, через которую имеется доступ к серверу приложения. Запуск приложения осуществляется по url-адресу http://creditbroker.com/, где находится страница авторизации, представлено 5.2. На данную страницу попадает любой неавторизованный пользователь при попытке перейти по любому url-адресу принадлежащему данному веб-приложению, так как это закрытая веб-система и доступ к функционалу приложения имеется только у авторизованных пользователей.

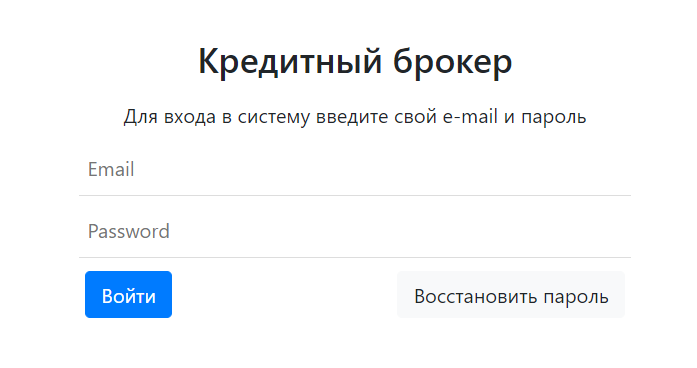


Рисунок 5.2 — Страница авторизации пользователя

На странице авторизации находится форма для подтверждения своих прав доступа. Пользователю требуется ввести свой логин (*e-mail*) и пароль, после чего нажать кнопку «Войти», чтобы войти в систему. В ситуации ввода некорректных данных пользователь будет уведомлён об этом с помощью нотификаций об ошибке. Существует возможность восстановления пароля, которая отображена кнопкой «Восстановить пароль».

В веб-приложении существует три уровня прав доступа:

* суперадминистратор;
* администратор;
* менеджер;
* брокер;
* сотрудник банка/лизинговой компании.

На каждой веб-странице данного веб-приложения находится панель навигации в левой части экрана, представленная на рисунке 5.2, на которой находятся доступные пользователю разделы.

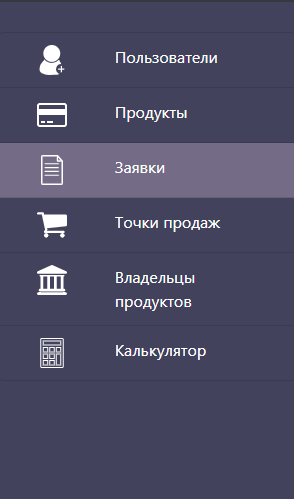


Рисунок 5.3 — Панель навигации

Также на каждой веб-странице данного веб-приложения расположен заголовок сайта представленный на рисунке 5.4, на котором отображены следующие элементы:

* кнопка сворачивания панели навигации до иконок;
* фотография пользователя;
* имя пользователя;
* роль пользователя;
* кнопка выхода из аккаунта.

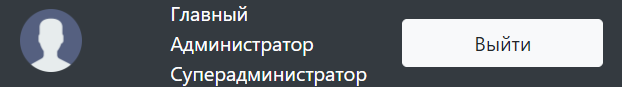


Рисунок 5.4 — Заголовок сайта

Стоит рассмотреть возможности пользователя с правами доступа суперадминистратора. После авторизации пользователь попадает на страницу со списком кредитных заявок.

На странице администратора находится таблица, отображающая данные о всех заявках определённого вида:

* ID заявки;
* клиент;
* кредитный продукт, на покупку которого оформляется кредитная заявка;
* банк, в который отправляется заявка на получение кредита;
* брокер, оформивший заявки;
* процентная ставка;
* сумма кредита;
* дата создания кредитной заявки;
* точка продаж, где оформляется кредитная заявка;
* статус заявки.

Снизу, под таблицей, имеется возможность выбора страницы и постраничного перехода.

Таблица представлена на рисунке 5.5.

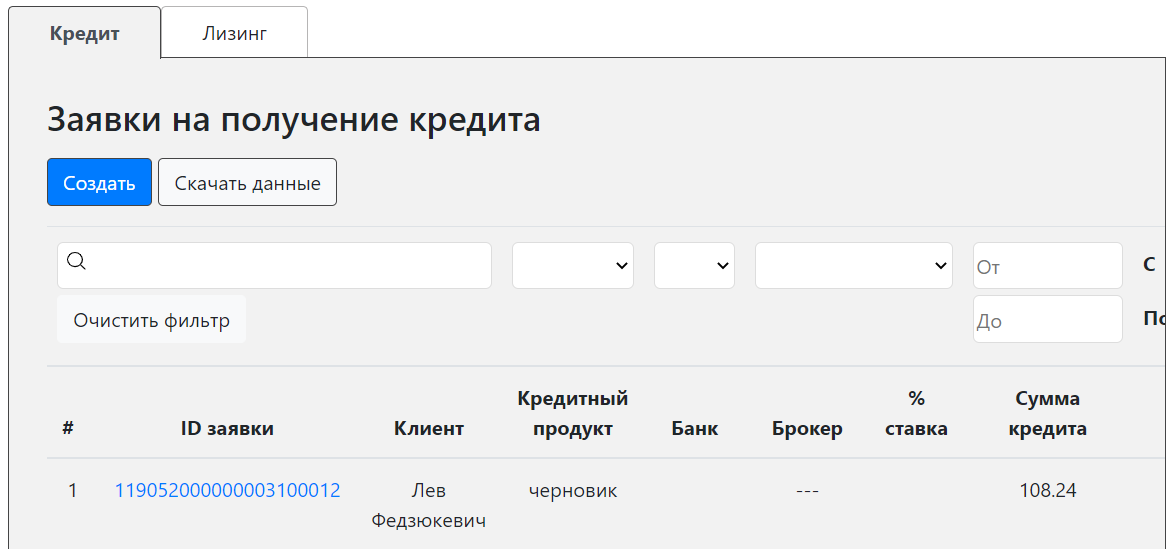


Рисунок 5.5 — Страница со списком кредитных заявок

Таблица поддерживает сортировку данных по всем категориям, которые представлены в таблице, что позволяет находить различные заявки с определённой суммой кредита на разные продукты из всех возможных точек продаж, составленные в определённый календарный промежуток, с всеми доступными статусами, а также поддерживается фильтрация данных на основе установленных фильтров к колонкам клиента, брокера и процентной ставки кредита. Все фильтры представляют собой выпадающие списки. Также имеется строка для быстрого поиска по таблице. Данный поиск будет реагировать на любые возможные совпадения по всем колонкам таблицы. Следует отметить, что сортировка, фильтрация и поиск по таблице могут использоваться одновременно и давать корректную выборку данных о студентах.

В самом верху страницы, находятся вкладки для переключения списков лизинговых и кредитных продуктов.

Также, в верху обеих страниц, находятся кнопки скачивания таблиц со списком созданных заявок и создания новой заявки. При нажатии на последнюю кнопку открывается страница с формой для создания новой кредитной либо лизинговой заявки.

Страница содержит несколько разделов:

* точка продаж;
* данные по кредиту;
* клиент;
* адрес клиента;
* контактное лицо;
* семейное положение;
* место работы;
* прочая информация.

Форма создание заявки изображена на рисунке 5.6.

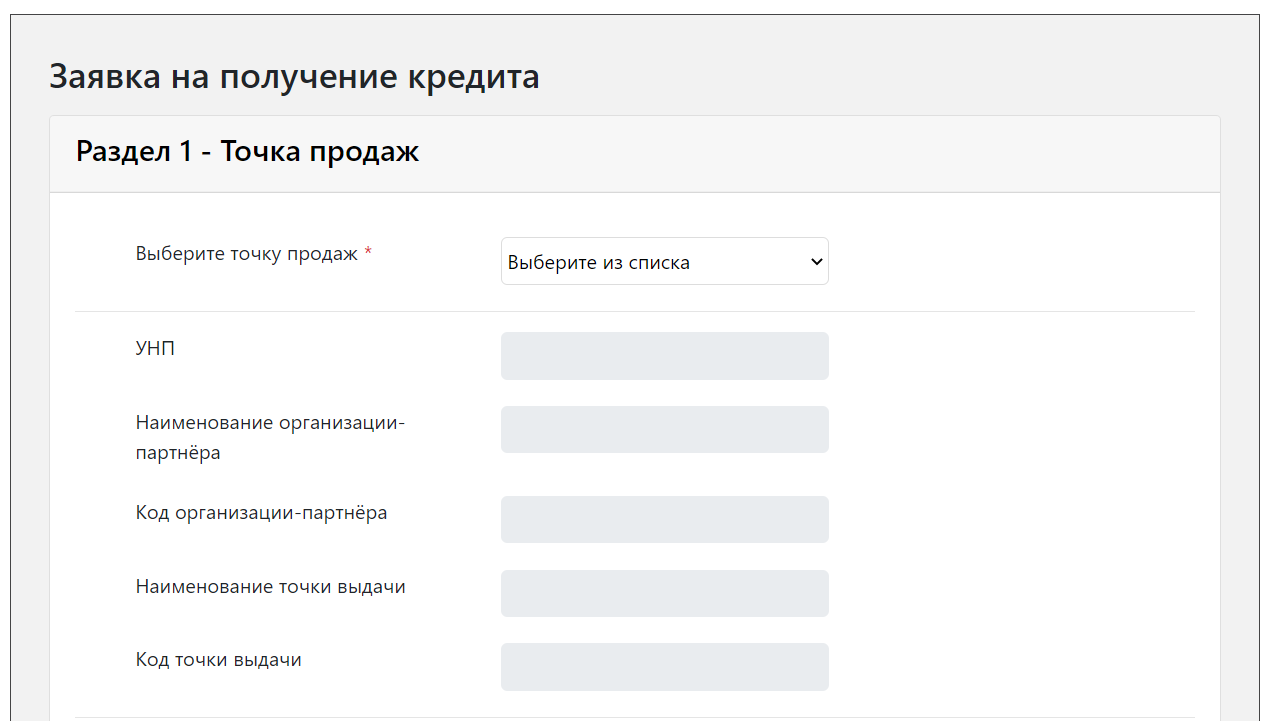


Рисунок 5.6 — Страница формой создания заявки

Также страница содержит поле примечаний, кнопку загрузки файлов, предоставленных клиентом для получения одобрения со стороны банка, а также поисковую строку для выбора продукта. В самом низу страницы находятся три кнопки:

* отмены;
* сохранения заявки как черновика;
* отправления заявки в банк.

Кнопка, отвечающая за отправление заявки в банк, будет доступна только если будут выполняться необходимые правила – заполнение всех обязательных полей. Кнопка отмены и кнопка сохранения заявки как черновика доступны на всех стадиях создания новой заявки.

После нажатия любой из этих кнопок, пользователь вернётся на страницу с таблицей, состоящей из внесённых в систему заявок, с отличием лишь в статусе заявки.

Редактирование как кредитных, так и лизинговых заявок доступно по нажатию на их ID в таблице. После этого открывается форма редактирования заявки на получение кредита, которая выглядит аналогично странице создания заявки.

Отличие заключается лишь в том, что поля, связанные с изменением информации о выбранном продукте, недоступны. Внешний вид страницы продемонстрирован на рисунке 5.6.

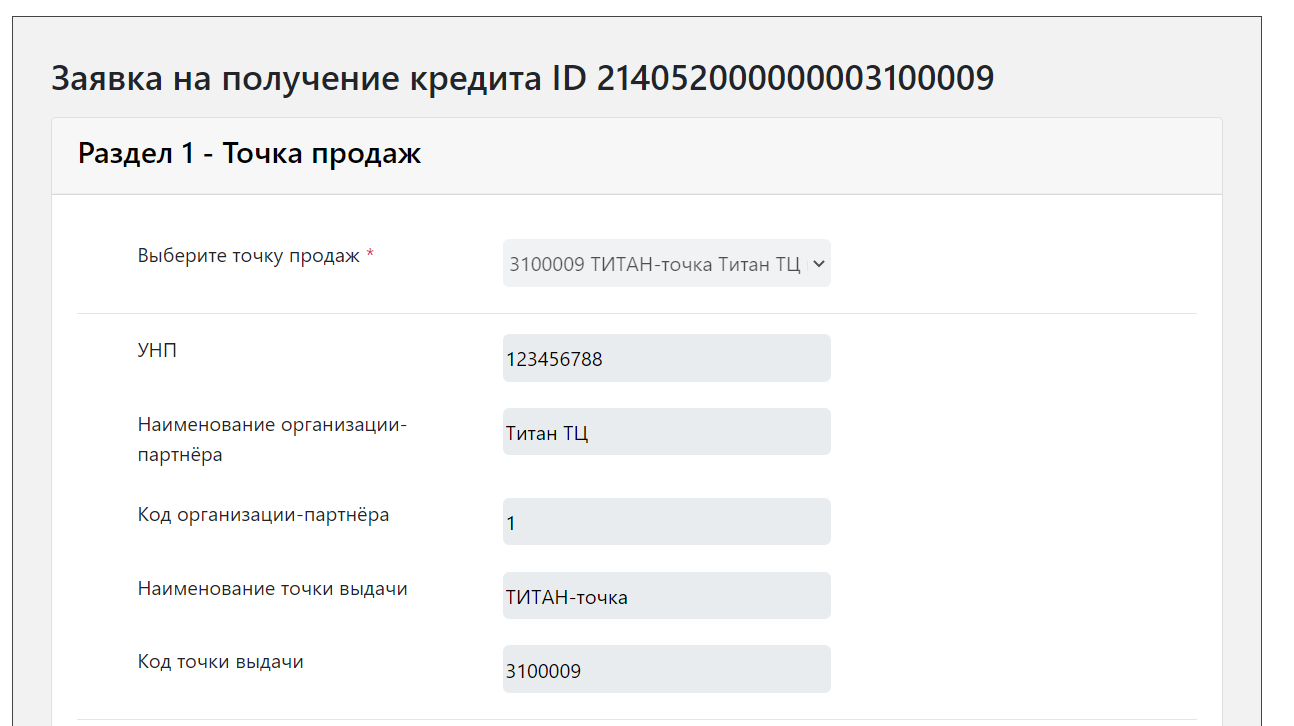


Рисунок 5.7 — Страница редактирования существующей заявки

Если какие-то правила будут нарушены пользователь увидит сообщение непосредственно под полем, где ошибка была допущена. Пока ошибка не будет исправлена программное средство не даст сохранить или отправить в банк заявку.

При переходе на страницу продуктов по нажатию на вкладку «Продукты» на навигационной панели пользователю будет показана таблица кредитных продуктов, представлено на рисунок 5.8.

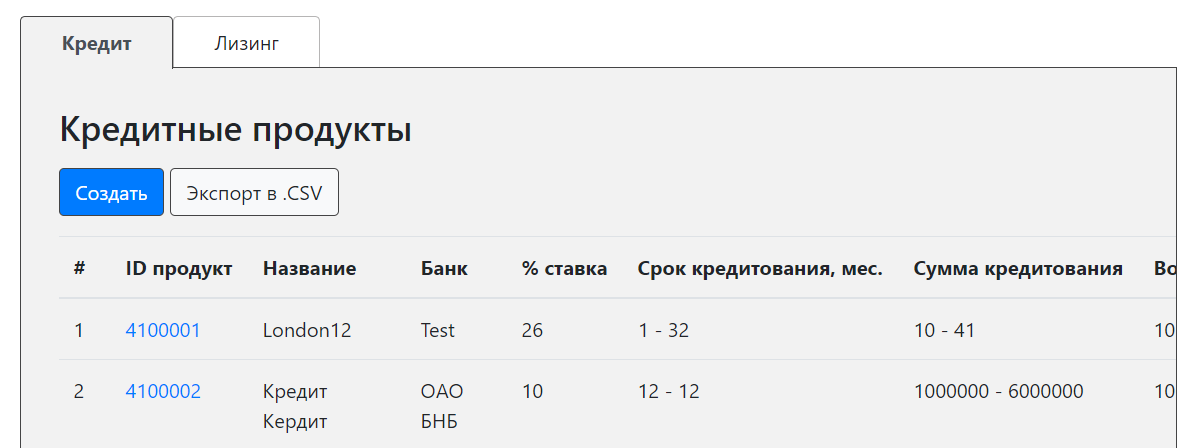


Рисунок 5.8 – Страница со списком кредитных продуктов

На этой странице расположены вкладки для перехода между таблицами кредитных и лизинговых продуктов. Также на каждой странице со списками находятся кнопки экспорта таблиц в .CSV и создания нового кредитного или лизингового продукта. При нажатии на кнопку создания пользователь попадёт на страницу, на которой находится панель с формой для заполнения информации о продукте.

Для регистрации продукта суперадминистратору необходимо ввести все необходимые данные в обязательные поля. После заполнения формы пользователю следует нажать кнопку создания и товар появится в таблице продуктов.

Пользователю также доступна функция редактирования продукта по нажатию на его ID в таблице. Так, перед пользователем появляется форма с информацией о продукте. Также на ней доступны кнопки удаления продукта, его сокрытия в списке и распечатывания в бумажном виде, по нажатию на которую появляется панель печати. Форма редактирования продукта представлена на рисунке 5.9.

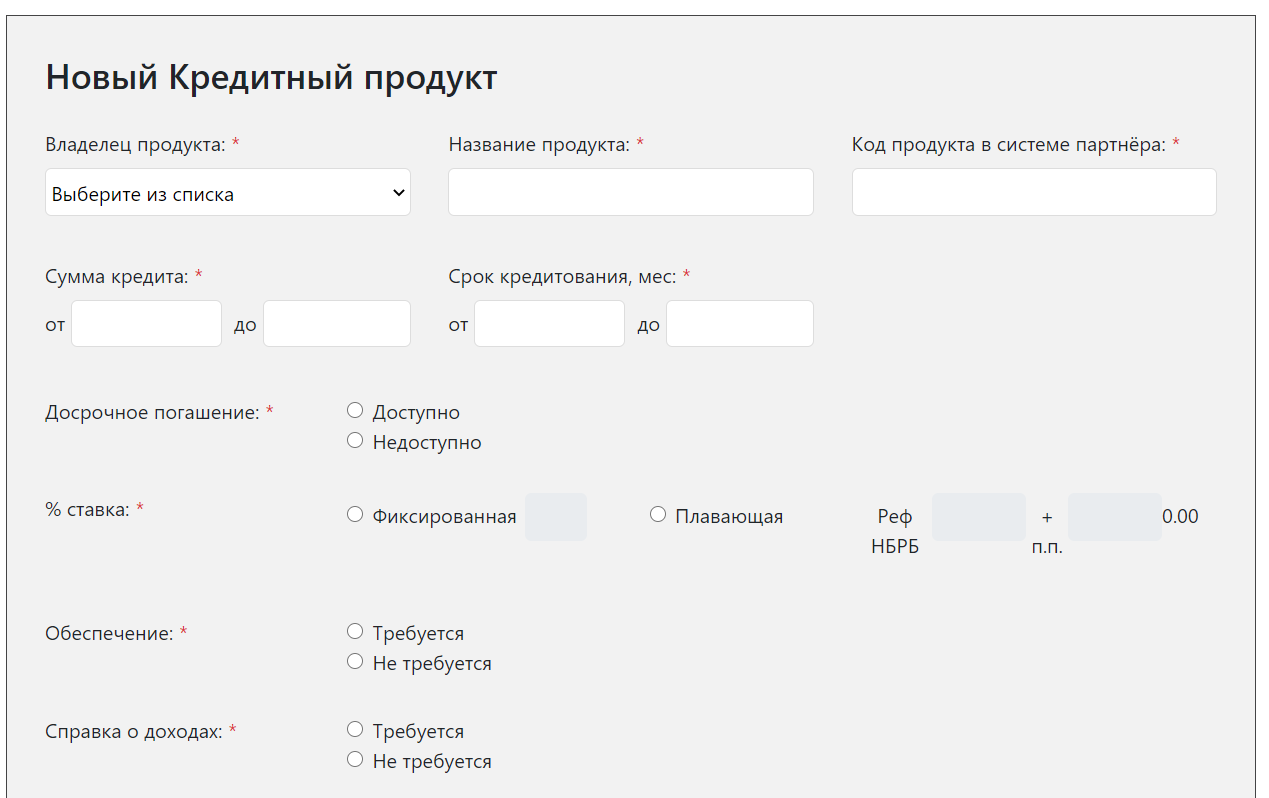


Рисунок 5.9 — Форма редактирования информации о продукте

При переходе на страницу пользователей по нажатию на вкладку «Пользователи» на навигационной панели пользователю будет показана таблица пользователей. Над таблицей есть вкладки, которые позволяют выбрать роль отображаемых пользователей.

На странице находится таблица, отображающая данные обо всех администраторах. Также на ней доступна кнопка создания нового пользователя с такой же ролью. При нажатии на неё откроется форма создания нового.

Также у суперадминистратора есть возможность попасть на страницу с формой редактирования профилей пользователей. Форма редактирования аналогична форме создания нового пользователя с различием лишь в том, что в форме редактирования все поля заполнены, изображено на рисунке 5.10.

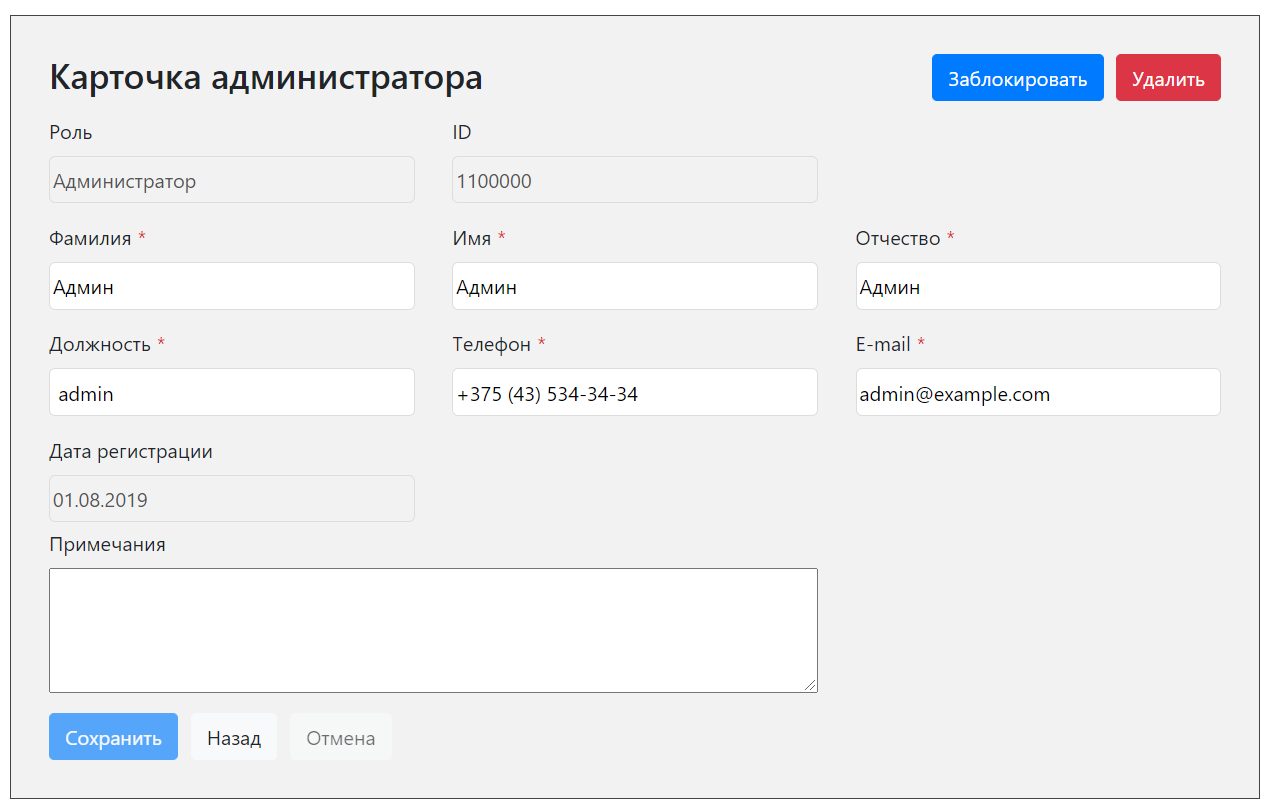


Рисунок 5.10 — Форма редактирования администратора

Главному администратору доступна функция блокировки и удаления пользователя, которые отсутствуют у других ролей пользователей. Также кнопка очистки полей заменяется на этой форме кнопкой отмены, которая убирает все изменения и возвращает первоначальную информацию обратно.

Раздел «Пользователи» содержит в себе таблицы с тремя типами ролей пользователей:

* администраторы;
* менеджеры;
* брокеры.

Редактирование всех трёх таблиц доступно только суперадминистратору. Администратор имеет возможность работать с двумя таблицами: менеджеры и брокеры.

При переходе на страницу с точками продаж по нажатию на вкладку «Точки продаж» на навигационной панели пользователю будет показана таблица всех занесённых в систему магазинов и торговых точек.

Таблица содержит следующие поля:

* ID;
* название;
* юридическое лицо;
* адрес;
* сотрудник;
* номер телефона;

Таблица приведена на рисунке 5.11.

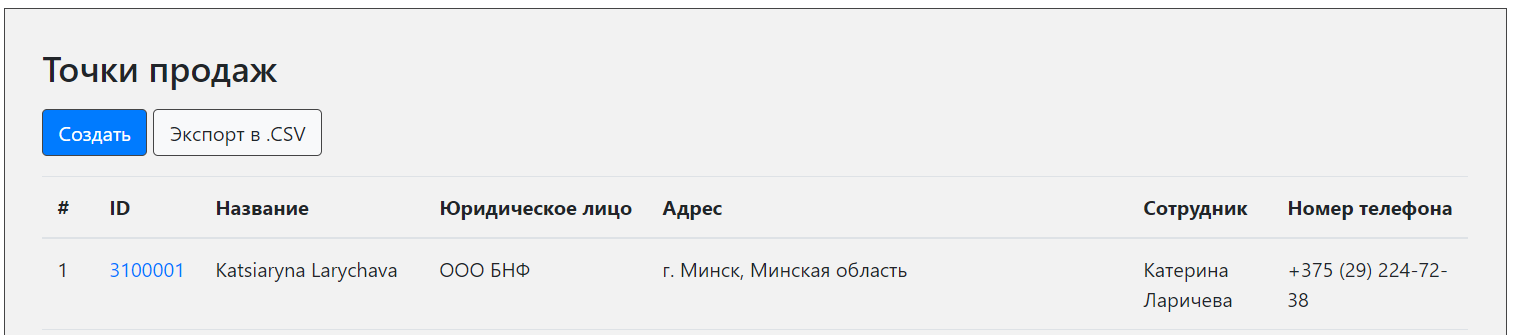


Рисунок 5.11 — Таблица точек продаж

На странице кроме таблицы точек продаж присутствует две кнопки: «Скачать данные» и «Создать». При нажатии на первую кнопку появляется панель печати с помещенный на лист А4 таблицей. Вторая кнопка открывает форму создания новой торговой точки, изображенную на рисунке 5.12.

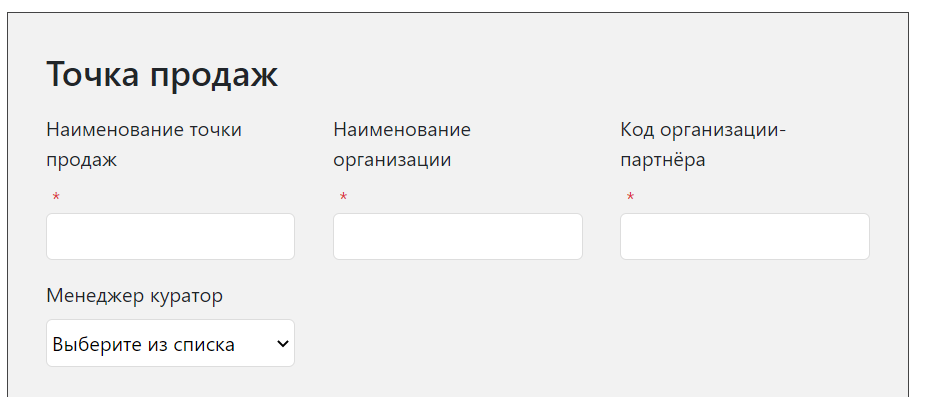


Рисунок 5.12 — Страница с формой создания новой точки продаж

Форма создания новой точки продаж содержит в себе поля для заполнения, кнопку отмены создания торговой точки и кнопку сохранения нововведённых данных.

Пользователь в этом разделе имеет возможность также изменить информацию о уже существующей точке продаж. Логика изменения информации о существующей торговой точке такая же, как и в случаях с заявками, продуктами и пользователями.

При нажатии на вкладку «Владельцы продуктов» на навигационной панели пользователь попадает на веб-страницу с информацией о владельцах продуктов. На странице в двух вкладках представлены таблицы с данными банков и владельцев продуктов, поставляющих лизинговые продукты в разные точки продаж. В первой вкладке «Банки» расположена таблица с данными о банках, а во второй вкладке «Лизинг» находится таблица с информацией о лизинговых компаниях. Также на странице присутствуют кнопки, дающие возможности скачать данные в .CSV формате и создать новые позиции в таблицах.

Страница со списком банков представлена на рисунке 5.13.

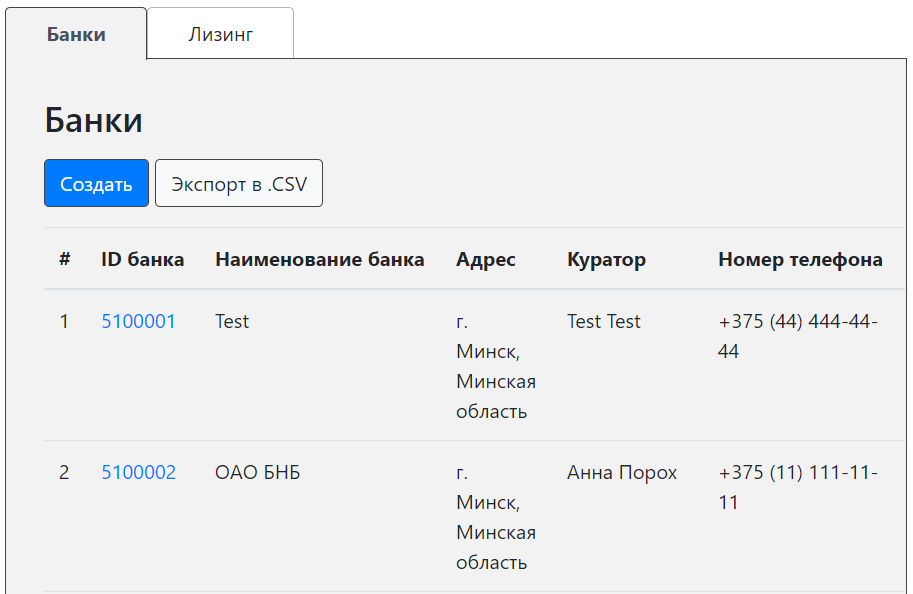


Рисунок 5.13 — Страница со списком банков

Отличительной особенностью продукта является калькулятор, который позволяет быстро рассчитать суммы платежей, а также получить полноценный график платежей. Пользователю следует нажать на вкладку «Калькулятор» на навигационной панели для получения доступа к калькулятору.

На странице находятся две вкладки «Кредит» и «Лизинг». В каждой из них находится калькулятор, состоящий из следующих компонентов:

* стоимости продукта;
* страхового кредита;
* первоначального взноса;
* кредитного продукта.

При заполнении всех обязательных полей станет доступна кнопка расчёта кредита либо лизинга. После нажатия на кнопку появляется дополнительная секция с аннуитетным и дифференцированным графиками платежей для кредитного калькулятора и с таблицей, содержащей данные о возмещении стоимости и вознаграждении лизингодателя, для лизингового калькулятора. В таблице подробно расписаны суммы в погашение долга и в погашение процентов. Также на странице вместе с таблицей появляется кнопка «Распечатать», которая вызывает панель печати и даёт возможность пользователю получить копию расчётов. Калькулятор представлен на рисунке 5.14.

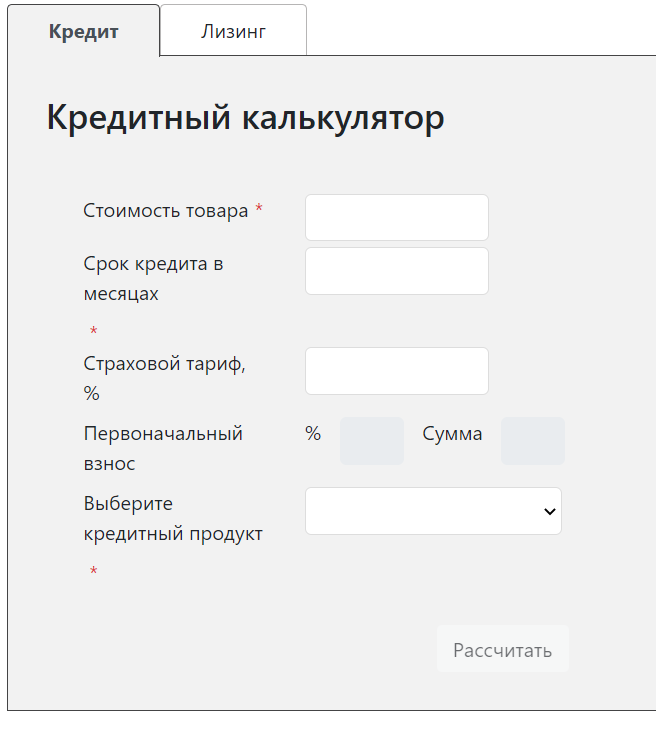


Рисунок 5.14 — Страница с кредитным калькулятором

5.5 Вывод по разделу

В данной главе продемонстрирована работа программы. Она работает стабильно. Интерфейс удобный и понятный. Предусмотрены различные проверки во избежание критических ситуаций и ошибок пользователя. Была учтена логика администрирования.

Для корректной работы приложения необходим сервер с установленной операционной системой Windows, Mac или Linux с объемом оперативной памяти не менее четырех гигабайт оперативной памяти.

**6 Экономический раздел**

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С.*

Утверд.

*Пацей Н.В.*

6 Экономический раздел

Лит.

Листов

8

*БГТУ 74417058, 2020*

Евлаш А.И.

**6.1 Общая характеристика разрабатываемого программного средства**

При выполнении данного дипломного проекта были разработаны клиентская и серверная части приложения.

Во время разработки дипломного проекта для серверной части использовалась технология Node.js для клиентской части в свою очередь использовалась библиотека React совместно с React-Redux. Разработанное веб-приложение представляет собой социальную платформу для создания заявок на предоставления кредита или лизинга, их просмотр, а также добавление самих продуктов, если их ещё нет в приложении. Также необходимо выполнить расчет экономии основных видов ресурсов в связи с использованием разработанного программного средства. Данное веб-приложение внедрено в компании ООО «Софттеко».

Данный раздел служит для определения затрат, произведенных на всех стадиях разработки программного средства.

**6.2 Исходные данные и маркетинговый анализ**

Источниками исходных данных для данных расчетов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Численность разработчиков | чел. | Чр | 1 |
| Норматив дополнительной заработной платы | % | Ндз | 15 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нфсзн | 34 |
| Ставка отчислений в Белгосстрах | % | Нбгс | 0,4 |
| Цена одного машино-часа | руб. | Смч | 0,6 |
| Норматив прочих затрат | % | Нпз | 34 |
| Норматив накладных расходов | % | Нобп*,* обх | 216 |
| Норматив расходов на сопровождение и адаптацию | % | Нрса | 17 |
| Ставка НДС | % | Нндс | 20 |

Конс.

Конс..

В ходе проведения маркетингового анализа, была выявлена стоимость разработки программного продукта по отслеживанию и добавлению отзывов на продукты питания. Средняя цена разработки составляет 40000 рублей. Также стоит учитывать модуль уведомлений и модуль экспорта данных, разработка которых отдельно стоит 4500 рублей и 6000, соответственно. Таким образом, общая стоимость разработки данного программного средства составляет 50500 рублей.

**6.3 Методика обоснования цены**

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

* снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
* сокращения расходов на оплату машинного времени и других ресурсов на отладку программных средств;
* снижения расходов на материалы;
* оптимизации программных средств;
* ускорения ввода в эксплуатацию новых систем;
* улучшения показателей основной деятельности в результате использования передовых программных средств.
* Стоимостная оценка программных средств у разработчиков предполагает определение затрат, что включает следующие статьи:
* заработная плата исполнителей – основная и дополнительная;
* отчисления в фонд социальной защиты населения;
* отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* расходы на оплату машинного времени;
* прочие прямые затраты;
* накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена конечного программного средства.

### **6.3.1 Объём программного средства**

Для оценки объема программного средства, все его функции классифицируются с использованием специального каталога функций, который определяет их объем. Общий объем программного средства Vo, вычисляется как сумма объемов Vi каждой из n его функций (формула 6.1).

|  |  |  |
| --- | --- | --- |
|  |  | (6.1) |

В ходе рассмотрения классификационной таблицы были выбраны функции, присутствующие в итоговом программном средстве. В таблице 6.2 представлены функции в условных машино-командах.

Таблица 6.2 – Содержание и объем функций в программном средстве

|  |  |  |
| --- | --- | --- |
| № функции | Содержание функции | Объем, условных машино-команд |
| 101 | Организация ввода информации | 520 |
| 102 | Контроль, предварительная обработка | 320 |
| 111 | Управление вводом/выводом | 1 000 |
| 201 | Обработка ошибочных и сбойных ситуаций | 400 |
| 202 | Взаимодействие между компонентами системы | 1600 |
| 401 | Взаимодействие с базой данных | 1200 |
| 402 | Вспомогательные методы | 210 |
| 601 | Генерация документов | 1500 |
| 707 | Графический вывод результатов | 550 |

Исходя из данных таблицы 6.2, можно рассчитать объем программного средства, разработанного в процессе дипломного проектирования:

*Vo* = 520 + 320 + 1000 + 400 +1600 + 1200 + 210 + 1500 + 550 = 7 300 (условных машино-команд).

Уточненный объем программного средства *Vo/* равен произведению объема программного средства *Vo* на коэффициент изменения скорости обработки информации *К*ск .

|  |  |  |
| --- | --- | --- |
|  | *Vo/ = Vo ·*Кск | (6.2) |

Исходя из вычисленного объема программного средства, можно определить уточненный объем программного средства:

*Vo/* = 7300 ⋅ 0,6 = 4380 (условных машино-команд).

**6.3.2 Основная заработная плата**

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в сфере веб-программирования на Node.js, React.js. В итоге было установлено, что средняя месячная заработная плата для позиций Junior/Middle составляет 1000 рублей [57-58]. Проект разрабатывался одним человеком на протяжении трёх месяцев. Основная заработная плата рассчитывается по формуле 6.3:

|  |  |
| --- | --- |
|  | (6.3) |

где Соз – основная заработная плата, руб.;

Траз – время разработки, месяцев;

Краз – количество разработчиков, человек;

Сзп – средняя месячная заработная плата.

руб.

Основная заработная плата, рассчитана исходя из данных представленных в таблице 6.1.

**6.3.3 Дополнительная заработная плата**

Законодательство о труде предусматривает наличие выплат, которые определяются по нормативу в процентах к основной заработной плате по формуле 6.4:

|  |  |
| --- | --- |
|  | (6.4) |

где Соз– основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %. Исходя из основной заработной платы, а также норматива дополнительной заработной платы, можно рассчитать сумму дополнительной заработной платы:

руб.

**6.3.4 Отчисления в Фонд социальной защиты населения**

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле 6.5:

|  |  |
| --- | --- |
|  | (6.5) |

где Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата на конкретное ПС, тыс. руб.;

Нфсзн– норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.6:

|  |  |
| --- | --- |
|  | (6.6) |

руб.

руб.

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 24,84 руб., а в фонд социальной защиты населения – руб.

**6.3.5 Расходы на материалы**

Сумма расходов на материалы СМ определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода НМ на уточненный объем программного средства *Vo/* .

|  |  |  |
| --- | --- | --- |
|  |  | (6.7) |

Учитывая, что норма расхода материалов в расчете на сто строк исходного кода равен 0,46 руб., можно определить сумму расходов на материалы:

СМ = 0,46 ⋅ 4380 / 100 = 20,15 руб.

Сумма расходов на материалы была вычислена на основе данных приведенных в таблице 6.1 данного дипломного проектирования.

**6.3.6 Расходы на оплату машинного времени**

Сумма расходов на оплату машинного времени Смв определяется как произведение стоимости одного машино-часа Смч на уточненный объем программного средства *Vo/* и на норматив расхода машинного времени на отладку ста строк исходного кода НМВ.

|  |  |  |
| --- | --- | --- |
|  | . | (6.8) |

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен можно определить сумму расходов на оплату машинного времени:

Смв = 0,06 ⋅ 4380 ⋅ 15 / 100 = 39,42 руб.

Сумма расходов на материалы была вычислена на основе данных приведенных в таблице 6.1 данного дипломного проектирования.

**6.3.7 Прочие прямые затраты**

Сумма прочих затрат Спз определяется как произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив прочих затрат в целом по организации Нпз.

|  |  |  |
| --- | --- | --- |
|  | . | (6.9) |

Спз = 5400 ⋅ 34 / 100 = 1836 руб.

**6.3.8 Накладные расходы**

Сумма накладныхрасходовСобп,обх – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх.

|  |  |  |
| --- | --- | --- |
|  | . | (6.10) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

Собп,обх = 5400 ⋅ 216 / 100 = 11664 руб.

**6.3.9 Сумма расходов на разработку программного средства**

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов.

|  |  |  |
| --- | --- | --- |
|  | Ср = Соз + Сдз + Сфсзн + Сбгс + См + Смв + Спз + Собп,обх | (6.11) |

Ср = 5400 + 810 + 2111,4 + 24,84 + 20,15 + 39,42 + 1836 + 11664 = 21905,81 руб.

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

**6.3.10 Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса.

|  |  |  |
| --- | --- | --- |
|  | . | (6.12) |

Срса = 21905,81 ⋅ 17 / 100 = 3723,99 руб.

Сумма расходов на сопровождение и адаптацию была вычислена на основе данных, рассчитанных ранее в раном разделе.

Все проведённые выше расчёты необходимы для вычисления полной себестоимости проекта.

**6.3.11 Полная себестоимость**

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов на сопровождение и адаптацию программного средства Срса.

|  |  |  |
| --- | --- | --- |
|  | . | (6.13) |

Сп = 21905,81 + 3723,99 = 25629,8 руб.

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

**6.3.12 Определение цены, оценка эффективности**

При расчете прибыли будем опираться на среднюю рыночную цену, по которой можно разработать данное программное обеспечение. Средняя рыночная цена составляет 23500 руб.

Прибыль рассчитывается по формуле 6.14:

|  |  |  |
| --- | --- | --- |
|  | . | (6.14) |

где Ппс – прибыль от реализации программного средства, руб.;

Цр – средняя рыночная цена продукта, руб.;

Сп – полная себестоимость программного средства, руб.;

Ппс = (50500/1,2) – 25629,8 = 16453,53 (руб.).

Уровень рентабельности разработанного программного средства определяется по формуле 6.15:

|  |  |
| --- | --- |
| , | (6.15) |

где Урент – уровень рентабельности программного средства, %

Сп – полная себестоимость программного средства, руб.;

Ппс – прибыль от реализации программного средства, руб.

Урент = (16453,53 / 25629,8) ⋅ 100 = 64,20 %

Рентабельность разработки программного средства была вычислена на основе данных, расчитанных ранее в данном разделе.

**6.4 Выводы по разделу**

В таблице 6.3 представлены результаты расчетов для основных показателей данного раздела.

Таблица 6.3 – Результаты расчетов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, мес. | 3 |
| Количество программистов, чел. | 1 |
| Зарплата с отчислениями, руб. | 8346,24 |
| Расходы на материалы, оплату машинного времени, прочие, руб. | 1895,57 |
| Накладные расходы, руб. | 11664 |
| Себестоимость разработки программного средства, руб. | 16341,06 |
| Расходы на сопровождение и адаптацию, руб. | 3723,99 |
| Полная себестоимость, руб. | 25629,8 |
| Цена аналога, руб. | 25629,8 |
| Прибыль от реализации, руб. | 16453,53 |
| Рентабельность разработки, % | 64,20 |

Необходимость разработки программного средства, обусловлена банков и ков и лизинговых компаний, что создаёт проблему выбора кредитного или лизингового продукта для рядового клиента. Чтобы решить данную проблему был разработано веб-приложение, которые будет помогать точкам продаж сотрудничать с различными банками и лизинговыми компаниями.

Разработка программного средства, осуществляемая одним программистом в течении трёх месяцев, при заданных условиях обойдется компании в 25629,8руб. Реализации данного программного средства по среднерыночной цене в 50500 руб, принесет прибыль компании в сумме 16453,53 рублей. Уровень рентабельности составит 64,20 %.

**Заключение**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С.*

Утверд.

*Пацей Н.В.*

Заключение

Лит.

Листов

1

*74417058, 2020*

В ходе дипломного проектирования проанализированы существующие программные средства, применяемые для оформления кредитных и лизинговых продуктов и заявок, выявлены их достоинства и недостатки. Это позволило определить существующие проблемы и пути их решения. Рассмотрены основные термины и определения в рассматриваемой сфере, рассмотрена структура сертификата. Рассмотрены различные библиотеки и фреймворки, которые могут быть использованы при разработке программного средства, описаны их достоинства и недостатки. На основе рассмотренных технологий был сделан и обоснован выбор тех библиотек и фреймворков, которые использовались в разработке приложения.

Спроектирована база данных, а также клиентская и серверная части приложения. Описан принцип взаимодействия между компонентами приложения, а также архитектура частей приложения. Были описаны принципы и подходы, применяемые для взаимодействия серверной части приложения с базой данных.

Разработано программное средство, представляющее собой веб-приложение, которое включает клиентскую и серверную части. Назначением разработанного приложения является создание кредитных и лизинговых продуктов и заявок.

Рассмотрен процесс разработки программного средства: даны примеры элементов клиентского и серверного приложений, подробно описано их назначение, а также принцип взаимодействия. Представлена UML-диаграмма вариантов использования, которая отражает уровни доступа для каждой из ролей приложения. Приложение позволяет назначать пользователям различные роли: Суперадминистратор, Администратор, Менеджер, Брокер, Сотрудник банка. Были разработаны инструкции пользователя.

**Список использованных источников**

ФИО

Подпись

Дата

Лист

1

ДП 00.00.ПЗ

Разраб.

Куванова О.Р.

Провер.

Шиман Д.В.

Н. контр.

*Рыжанкова А.С*.

Утверд.

*Пацей Н.В.*

Список использованных источников

Лит.

Листов

1

*БГТУ 74417012, 2019*

1. Habr: Философия REST [Электронный ресурс] / habr.com – Дата доступа: https://habr.com/post/312360 /. Дата доступа: 21.04.2020.
2. REST API: Коды состояний [Электронный ресурс] / www.restapitutorial.ru – Режим доступа: http://www.restapitutorial.ru/httpstatuscodes.html. Дата доступа: 21.04.2020.
3. Лекция: Назначение и применение JavaScript, общие сведения [Электронный ресурс] – Режим доступа: https://studfiles.net/preview/6854757/. – Дата доступа: 12.05.2020.
4. ECMA International [Электронный ресурс] / ECMAScript 2017 Language Specification – Режим доступа: http://www.ecma-international.org/ecma-262/ – Дата доступа: 22.02.2018.
5. Введение в React [Электронный ресурс] – Дата доступа: https://metanit.com/web/react/1.1.php. – Дата доступа: 12.05.2020.
6. Эффекты и саги. React + Redux-saga [Электронный ресурс]: https://ru.redux-saga.js.org/. – Дата доступа: 12.05.2019.
7. Tproger: Краткое руководство по Redux для начинающих [Электронный ресурс]: <https://tproger.ru/translations/redux-for-beginners/> – Дата доступа: 12.05.2020.
8. Цыганенко, Н.П. Курс лекций по NodeJS / Н.П. Цыганенко. – Минск: БГТУ, 2017. – 175 с.
9. Medium: Node.js [Электронный ресурс]: medium.com/ – Режим доступа: https://medium.com/devschacht/node-hero-chapter-1-239f7afeb1d1/. Дата доступа: 24.04.2020.
10. npm [Электронный ресурс] / Главная страница – Режим доступа: https://www.npmjs.com/ – Дата доступа: 10.04.2018.
11. Amazon: Что такое базы данных NoSQL? [Электронный ресурс]: <https://aws.amazon.com/ru/nosql/>. Дата доступа: 24.04.2020.
12. Wikipedia: Микросервисна архитектура [Электронный ресурс]: <https://ru.wikipedia.org/wiki/микросервисная_архитектура>. Дата доступа: 24.04.2020.\
13. Habr: Микросервисы (Microservices) [Электронный ресурс]: <https://habr.com/ru/post/249183/>. Дата доступа12.05.2020.
14. Паттерн MVVM. Определение паттерна MVVM [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/wpf/22.1.php. – Дата доступа: 21.05.2019.
15. Habr: Flux для глупых людей [Электронный ресурс]: <https://habr.com/ru/post/249183/>. Дата доступа12.05.2020.

ПРИЛОЖЕНИЕ А

ПРИЛОЖЕНИЕ Б

ПРИЛОЖЕНИЕ В

ПРИЛОЖЕНИЕ Г

ПРИЛОЖЕНИЕ Д

ПРИЛОЖЕНИЕ Е