

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

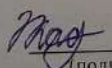
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

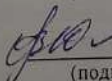
на тему

**БИЗНЕС-ПРОЕКТ «КРЕДИТНЫЙ БАНК:
ОНЛАЙН-ПЛАТФОРМА ДЛЯ
АВТОМАТИЗАЦИИ КРЕДИТНЫХ
ПРОЦЕССОВ»**

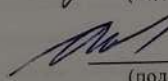
Студент

 30.05.2025 Кабачек Никита Андреевич
(подпись, дата) (ФИО полностью)

Руководитель

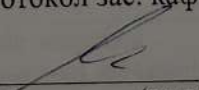
 30.05.2025 Орова О.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 30.05.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой выпущено (протокол заседания ГЭК от

09.06.2025 г. № 20)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«06» мая 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кабачек Никита Андреевич гр. 21ВВП1 факультета ФВТ

2. Тема работы Бизнес - проект «Кредитный банк: онлайн-платформа для автоматизации кредитных процессов»

Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.

3. Руководитель бакалаврской работы Юрова О.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать онлайн платформу, позволяющую автоматизировать кредитные процессы. Составить бизнес-модель и финансовый план. Язык программирования Java и TypeScript, среда разработки IntelliJ IDEA. Программа должна обладать интуитивно понятным интерфейсом в браузере и использовать следующие технологии:

1. REST API
2. Java
3. TypeScript
4. Микросервисная архитектура
5. SMTP

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие онлайн платформы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для разработки приложения на Java и Typescript.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	4%	06.05.2025	<i>фиа</i>
2	Анализ требований	4%	07.05.2025	<i>фиа</i>
3	Выбор технологий, применяемых при проектировании и разработке	2%	08.05.2025	<i>фиа</i>
4	Проектирование архитектуры	10%	09.05.2025 - 11.05.2025	<i>фиа</i>
5	Проектирование и Разработка ПО	50%	12.05.2025 - 24.05.2025	<i>фиа</i>
6	Тестирование и отладка	20%	25.05.2025 - 29.05.2025	<i>фиа</i>
7	Оформление выпускной работы	10%	30.05.2025	<i>фиа</i>

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы *фиа 06.05.25* Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент *Маг 06.05.25* Кабачек Н.А.
(подпись, дата)

Работу к защите допустить
Декан факультета *Л 06.06.2025* Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 94 листа, 39 рисунков, 3 таблицы, 4 источника, 3 приложения.

ОНЛАЙН ПЛАТФОРМА, ЛИЧНЫЙ КАБИНЕТ, КРЕДИТНАЯ КАРТА, ПОДАЧА ЗАЯВКИ, ВАЛИДАЦИЯ ДАННЫХ, ТЕСТИРОВАНИЕ, БАЗА ДАННЫХ, POSTGRESQL, JAVA, SPRING BOOT, TYPESCRIPT, REACT, INTELLIJ IDEA, LIQUIBASE, REST

Цель работы – разработать инструмент, позволяющий автоматизировать кредитные процессы.

Одним из путей достижения поставленной задачи является онлайн платформа, обеспечивающая выполнение следующих функций: авторизация пользователей, отслеживание активных и завершенных заявок в личном кабинете, отображение текущего курса валют банка, актуальных новостей из мира финансов, пятиэтапное оформление кредитной карты, информирование пользователя письмами на электронную почту.

В конечном итоге разработана онлайн платформа, которая обеспечивает автоматизированное оформление кредитных карт. Онлайн платформа полностью соответствует цели данной выпускной квалификационной работы. В ходе достижения результата, были использованы следующие технологии: Java, Spring Boot, TypeScript, React, PostgreSQL, Liquibase, Google SMTP, Lombok, bcrypt, FeignClient, Swagger. Онлайн платформа протестирована и готова к эксплуатации.

					ПГУ1.090301.09.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Бизнес - проект «Кредитный банк: онлайн-платформа для автоматизации кредитных процессов»	Лит.	Лист	Листов
Разраб.		Кабачек Н.А.		30.05.25				
Провер.		Юрова О.В.		30.05.25			2	94
Н. Контр		Бычков А.С.		30.05.25		ФВТ, каф. ВТ гр. 21ВВП1		
				30.05.25				
Утверд.		Митрохин М.А.		30.05.25				

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Анализ предметной области.....	7
1.1 Описание потребности пользователей.....	7
1.1.1 Аналогичные решения.....	8
1.1.2 Прямые аналоги.....	8
1.1.3 Косвенные аналоги.....	8
1.2 Постановка задачи.....	10
1.2.1 Назначение	10
1.2.2 Требования к аппаратной части	11
2 Архитектура приложения и выбор инструментов разработки	12
2.1 Архитектура приложения	12
2.2 Выбор инструментов разработки.....	14
3 Реализация.....	15
3.1 Работа с БД.....	15
3.2 Описание доступных функций	16
3.3 Описание программы.....	26
3.4 Тестирование	32
4 Безопасность жизнедеятельности.....	34
4.1 Правила безопасности при использовании ПК.....	34
4.2 Правила безопасности при использовании смартфона	35
5 Стартап проект.....	37
5.1 Аннотация проекта.....	37
5.2 Базовая бизнес-идея	37
5.3 Характеристика будущего продукта или услуги	38
5.4 Организационные, производственные и финансовые параметры	39
5.4.1 Внутренние процессы организации бизнеса.....	39
5.4.2 Ключевые характеристики производства	40

5.4.3 Объемы и разнообразие требуемых ресурсов	40
5.5 Основные конкурентные преимущества.....	40
5.6 Научно-техническое решение и/или результаты, необходимые для создания продукции	41
5.7 Финансовые параметры.....	42
5.8 Взаимоотношения с клиентами	42
5.9 Оценка затрат и доходов	43
Заключение.....	45
Список использованных источников.....	47
Приложение А - UML – диаграммы программы.....	48
Приложение Б - Листинг программы	53
Приложение В - Презентация	90

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

МС – микросервис

ПСК (полная стоимость кредита) – совокупность всех расходов заёмщика, связанных с получением и обслуживанием кредита

СУБД – системы управления базами данных

API – (англ. Application programming interface) — это набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением для использования во внешних программных продуктах

CPU (англ. central processing unit) – центральный процессор

DTO (англ. Data Transfer Object) – класс данных, содержащий только необходимые поля данных без какого-либо поведения или бизнес-логики

IDE (англ. Integrated Development Environment) – интегрированная среда разработки

JPA (англ. Java Persistence API) — спецификация API Jakarta Enterprise Edition, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных

Prescoring – предварительное рассмотрение заявки клиента при минимальном предоставлении пакета документов

REST (англ. REpresentational State Transfer – «передача репрезентативного состояния» или «передача «самоописываемого» состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети

Scoring – это метод анализа, который банки и другие финансовые организации используют для оценки рисков при выдаче кредитов

SMTP (англ. Simple Mail Transfer Protocol) – протокол передачи электронной почты

UML (англ. Unified Modeling Language) – унифицированный язык моделирования

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

Введение

В настоящее время банковская деятельность переживает период бурных изменений, которые вызваны внедрением новых информационных технологий и глобализацией финансовых рынков. На волне радикальных рыночных реформ банковская система коренным образом изменилась: увеличилось количество банковских организаций, при этом все они основывают свою деятельность на рыночных принципах, что создает условия для развития конкуренции на рынке банковских услуг [1].

Как правило, внедрение современных банковских технологий снижает уровень рисков, сопутствующих проведению расчетов.

Целью выпускной работы является реализация системы банковского сервиса.

Актуальность выпускной квалификационной работы обусловлена следующим фактором: внедрение информационных технологий открыло перед банками новые возможности по управлению рисками, развитию прогрессивных форм обслуживания клиентов, дальнейшей диверсификации их деятельности. Информационные технологии стали основой многих финансовых инноваций, привели к созданию различных финансовых инструментов, которые сократили степень неполноты и несовершенства финансовых рынков. К тому же в соответствие со статьей 4 Федерального закона "О Центральном банке Российской Федерации", Банк России уполномочен вводить для банков особые правила бухгалтерского учета и отчетности, для которых не подходят обычные бухгалтерские системы, не имеющих средств автоматизации получения отчетности [2,3]. Кроме того, постоянные изменения законодательства означают необходимость постоянных изменений программного обеспечения.

Таким образом, можно сделать вывод, что тема выпускной квалификационной работы является актуальной.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

1 Анализ предметной области

1.1 Описание потребности пользователей

Кредитное приложение предназначено для широкого круга пользователей, которым важно быстро и удобно оформить кредитный продукт без необходимости посещения банковского отделения. В первую очередь, это люди, ценящие свое время и предпочитающие цифровые сервисы традиционным методам взаимодействия с финансовыми организациями.

Основная аудитория включает в себя работающих граждан, фрилансеров, малый бизнес и молодых специалистов, которым могут потребоваться дополнительные средства для решения текущих финансовых задач. Например, это могут быть внезапные расходы (ремонт, лечение), крупные покупки (техника, путешествия) или необходимость в гибком кредитном лимите для повседневных трат.

Современный ритм жизни требует мгновенного доступа к финансовым услугам. Пользователи хотят избежать длительного ожидания в очередях, бумажной волокиты и сложных процедур верификации. Им важно получить предварительное решение по заявке в течение нескольких минут, сравнить условия кредитования и выбрать наиболее подходящий вариант.

Ключевые потребности пользователей основаны, в первую очередь, на простоте и скорости – пользователи хотят интуитивно понятного интерфейса и минимальных действий для ожидаемого результата, в нашем случае – оформления заявки. Также важно видеть все условия кредита до подписания договора, такие как: ставку, срок, переплату. Пользователи, в том числе, нуждаются в защите персональных данных и подтверждении операций через email или SMS. К потребностям также относится возможность оформления нескольких заявок, изменение условий или отказ от кредита на любом этапе.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

1.1.1 Аналогичные решения

В современном банкинге существует множество решений для онлайн-кредитования, каждое из которых предлагает свой подход к оформлению заявок, проверке клиентов и выдаче кредитных продуктов. Рассмотрим несколько ключевых примеров, чтобы понять, какие функции востребованы на рынке и какие недостатки стоит учесть при разработке.

1.1.2 Прямые аналоги

Т-Банк – один из лидеров в сфере цифрового кредитования. Их система позволяет оформить кредитную карту полностью онлайн, начиная с минимального prescoring'a и заканчивая электронной подписью документов. Процесс занимает считанные минуты, а клиенту сразу предлагается несколько вариантов условий (например, со страховкой или без). Однако у Т-Банка довольно строгие требования к кредитной истории, и, если заявка отклонена, пользователь не может оперативно скорректировать данные – приходится ждать или обращаться в поддержку.

Альфа-Банк также предоставляет возможность взять кредит онлайн, но здесь больше внимания уделяется безопасности. Например, для подписания документов используется биометрическая аутентификация (Face ID/Touch ID), что добавляет надежности, но может замедлить процесс для пользователей без современных смартфонов. В отличие от Тинькофф, проверка заявки в Альфа иногда занимает до суток, что не всегда удобно для тех, кому деньги нужны срочно.

1.1.3 Косвенные аналоги

Если говорить о дополнительных функциях, таких как курсы валют и финансовые новости, то здесь хороший пример – Revolut. Это приложение показывает актуальные обменные курсы в реальном времени и дополняет кратким дайджестом экономических событий. Это полезно для клиентов, которые хотят не только оформить кредит, но и следить за финансовой ситуацией в целом. Однако Revolut не занимается кредитованием напрямую, так что этот

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

функционал можно рассматривать как вдохновение для главной страницы. Мобильное приложение Revolut показано на рисунке 1.

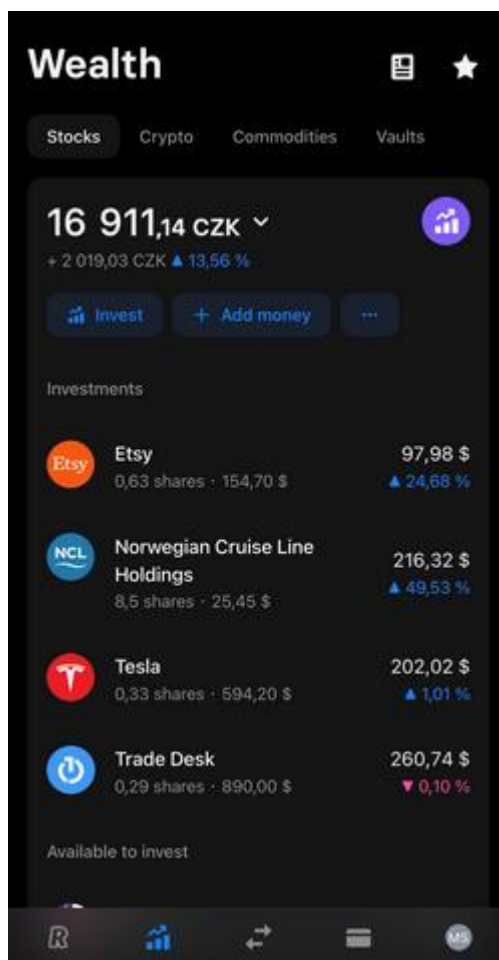


Рисунок 1 – Интерфейс мобильного приложения банка

Что касается работы с документами и электронной подписью, то здесь безусловный лидер – DocuSign. Их система позволяет удобно подписывать договоры, устанавливать таймеры на подтверждение и даже интегрироваться с другими сервисами. Но DocuSign – это отдельный продукт, а не часть банковского приложения, поэтому в нашем случае важно сделать процесс подписания таким же удобным, но без лишних шагов. Интерфейс web-приложения DocuSign показан на рисунке 2.

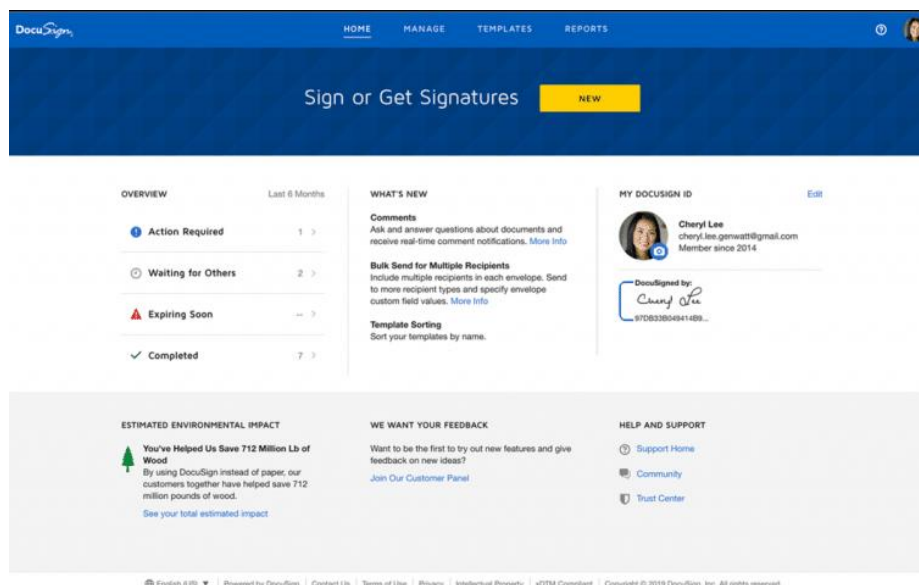


Рисунок 2 – Интерфейс web-приложения DocuSign

Сравнение аналогов и решения, предлагаемого в этой работе представлено в таблице 1.

Таблица 1 – Сравнение аналогов

Критерий	Тинькофф	Альфа-Банк	Revolut	DocuSign	Наше решение
Prescoring	+	+	-	-	+
Варианты условий	+	-	-	-	+
Email-поток	+	+	-	+	+
Таймеры	-	-	-	+	+
Курсы валют	-	-	+	-	+

1.2 Постановка задачи

1.2.1 Назначение

Разрабатываемое приложение представляет собой онлайн-сервис для оформления кредитных карт, где пользователь может быстро подать заявку, выбрать подходящие условия и завершить процесс полностью дистанционно –

без посещения банка. Основная цель – сделать процедуру максимально удобной, прозрачной и безопасной, сократив время от подачи заявки до получения решения до минимума.

1.2.2 Требования к аппаратной части

Аппаратные требования стандартны для веб-приложений: серверная часть разворачивается на виртуальной машине или в облаке с минимум 2 ядрами CPU и 4 ГБ оперативной памяти. Для базы данных рекомендуется выделенный сервер с SSD-диском для оптимальной скорости работы. Клиентская часть должна корректно работать на любых современных устройствах – от смартфонов до десктопов, с поддержкой браузеров Chrome, Firefox, Safari и Edge (последние две версии).

					ПГУ1.090301.09.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

2 Архитектура приложения и выбор инструментов разработки

2.1 Архитектура приложения

Для разработки приложения выбрана многослойная микросервисная архитектура, поскольку она предлагает разделить систему на множество небольших, слабо связанных сервисов. Каждый сервис отвечает за конкретную бизнес-функцию и может разрабатываться, развертываться и масштабироваться независимо от остальных. Это обеспечивает высокую гибкость и возможность быстрого внесения изменений, но также усложняет процесс разработки и управления системой в целом [4].

В отличие от микросервисной архитектуры, монолитная архитектура подразумевает создание единого, неделимого приложения. Все функциональные модули тесно связаны между собой, образуя сплоченный организм. Такой подход имеет ряд преимуществ: простота разработки, отладки и развертывания, высокая производительность при выполнении сложных операций, затрагивающих несколько модулей системы [1].

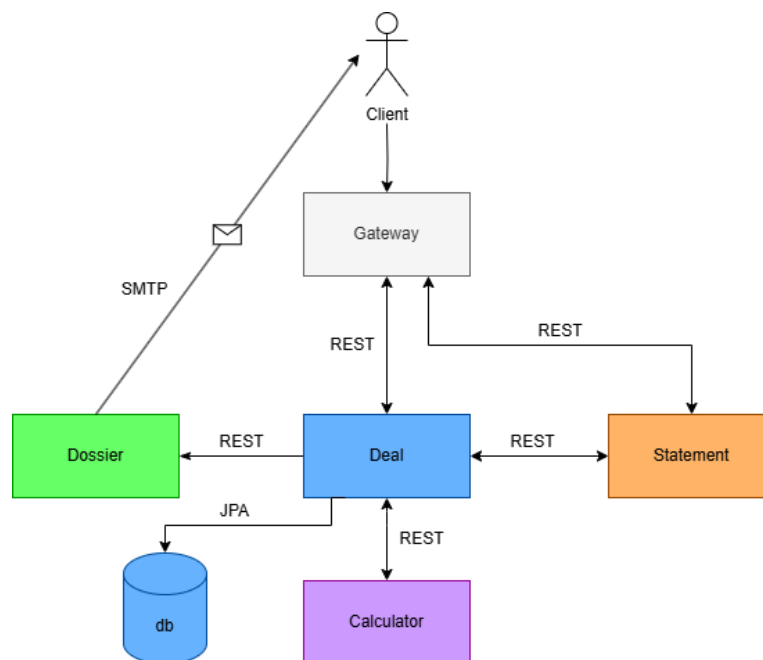


Рисунок 3 – Архитектура приложения

На рисунке 3 изображены компоненты системы микросервисной архитектуры. Работу системы можно описать следующим образом:

- пользователь отправляет заявку на кредит;
- MC Statement осуществляет prescoring заявки и если он проходит, то заявка сохраняется в MC Deal и отправляется в MC Calculator;
- MC Calculator возвращает через MC Заявку пользователю 4 предложения по кредиту с разными условиями (например, без страховки, со страховкой, с зарплатным клиентом, со страховкой и зарплатным клиентом) или отказ;
- пользователь выбирает одно из предложений, отправляется запрос в MC Заявка, а оттуда в MC Сделка, где заявка на кредит и сам кредит сохраняются в базу;
- MC Dossier отправляет клиенту письмо с текстом "Ваша заявка предварительно одобрена, завершите оформление";
- клиент отправляет запрос в MC Сделка со всеми своими полными данными о работодателе и прописке;
- происходит scoring данных в MC Calculator, MC Calculator рассчитывает все данные по кредиту (ПСК, график платежей и тд), MC Deal сохраняет обновленную заявку;
- после валидации MC Dossier отправляет письмо на почту клиенту с одобрением или отказом;
- если кредит одобрен, то в письме присутствует ссылка на запрос "Сформировать документы";
- клиент отправляет запрос на формирование документов в MC Dossier, MC Dossier отправляет клиенту на почту документы для подписания и ссылку на запрос на согласие с условиями;
- клиент может отказаться от условий или согласиться. Если согласился - MC Dossier на почту отправляет код и ссылку на подписание документов, куда клиент должен отправить полученный код в MC Сделка;

– если полученный код совпадает с отправленным, MC Deal выдает кредит.

2.2 Выбор инструментов разработки

Требования к бэкенд-части включают использование современных технологий: языка Java 17+ и фреймворк Spring Boot 3.3 для обеспечения стабильности и производительности, PostgreSQL в качестве надежной СУБД, Spring Data JPA для удобной работы с данными и Liquibase для реализации миграции БД. Документирование API будет вестись через набор инструментов Swagger, тестирование реализуется через фреймворк для написания и выполнения модульных тестов JUnit, а библиотека Lombok поможет сократить объем шаблонного кода. Для отправки писем на электронную почту был выбран SMTP-сервер Google Workspace.

Фронтенд-часть строится на основе библиотеки React версии 17.2.0 и выше с TypeScript для строгой типизации и меньшего количества ошибок. Стилизация – на выбор разработчика: языки разметки CSS, Sass/SCSS или Less, а для запросов к API используется библиотека Fetch или Axios. Тестирование интерфейсов проводится с помощью testing-library-react.

В качестве среды для разработки был выбран IntelliJ IDEA, поскольку она предоставляет мощную поддержку Java и Spring Boot, включая автодополнение кода, встроенные инструменты для работы с базами данных и REST API, а также удобную интеграцию с Maven/Gradle. IDE также предлагает advanced-отладку, анализ производительности и встроенные шаблоны для микросервисной архитектуры, что ускоряет разработку. Дополнительные преимущества — умный рефакторинг и поддержка плагинов для работы с PostgreSQL, что критично для данного проекта.

3 Реализация

3.1 Работа с БД

Для хранения информации необходимо реализовать базу данных со следующими таблицами:

- account - таблица, содержащая информацию о данных авторизации клиента;
- client - таблица, содержащая информацию о клиенте;
- statement - таблица, содержащая информацию о заявках;
- credit - таблица, содержащая информацию о кредитных предложениях.

Для наглядного понимания работы БД, на рисунке 1.13 представлена схема физической модели базы данных.

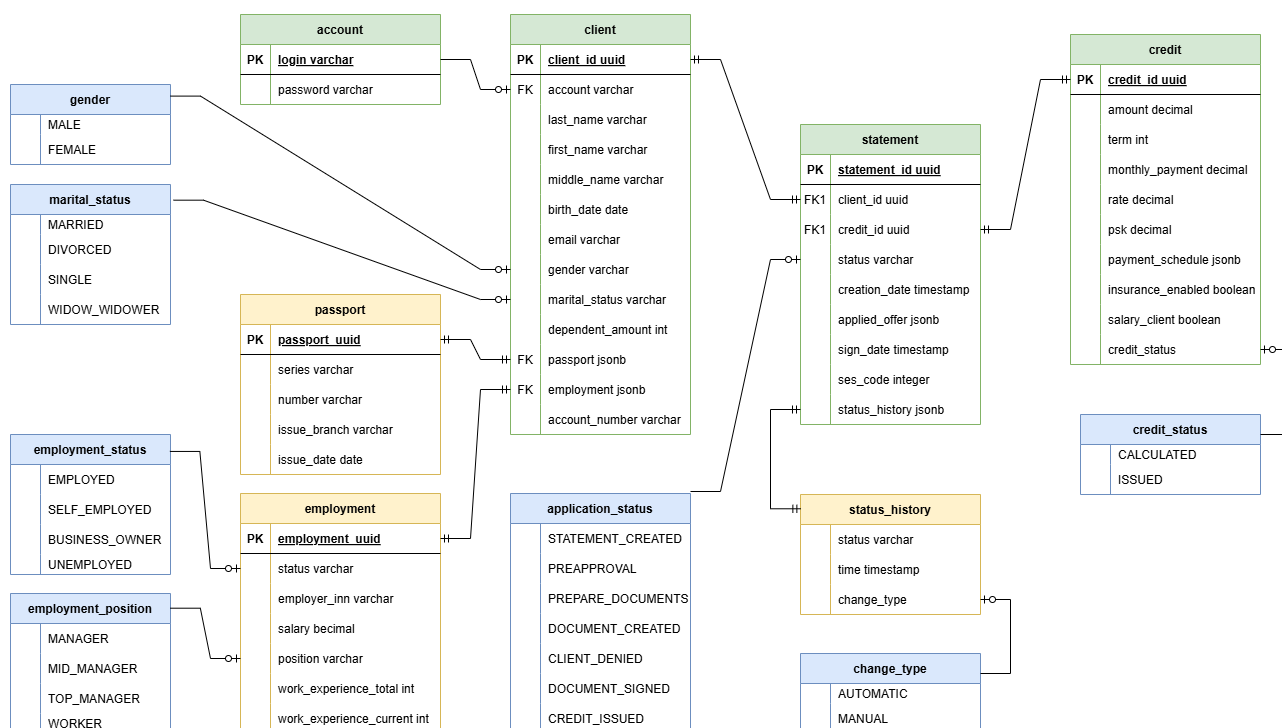


Рисунок 4 – Физическая модель базы данных

3.2 Описание доступных функций

Онлайн платформа предполагает реализацию таких функций, как:

- авторизация;
- отображение текущего курса валют;
- отображение актуальных новостей из сферы финансов;
- оформление кредитной карты;
- отправка писем на электронную почту.

Открыть онлайн платформу можно двумя способами:

- по ссылке домена, если приложение располагается на сервере и принимает запросы от зарегистрированного домена;
- через ссылку «<http://localhost:3000>», если приложение запущено локально, на том же устройстве.

После перехода по ссылке, пользователю открывается главная страница онлайн платформы, представленная на рисунках 5, 6, 7, 8.

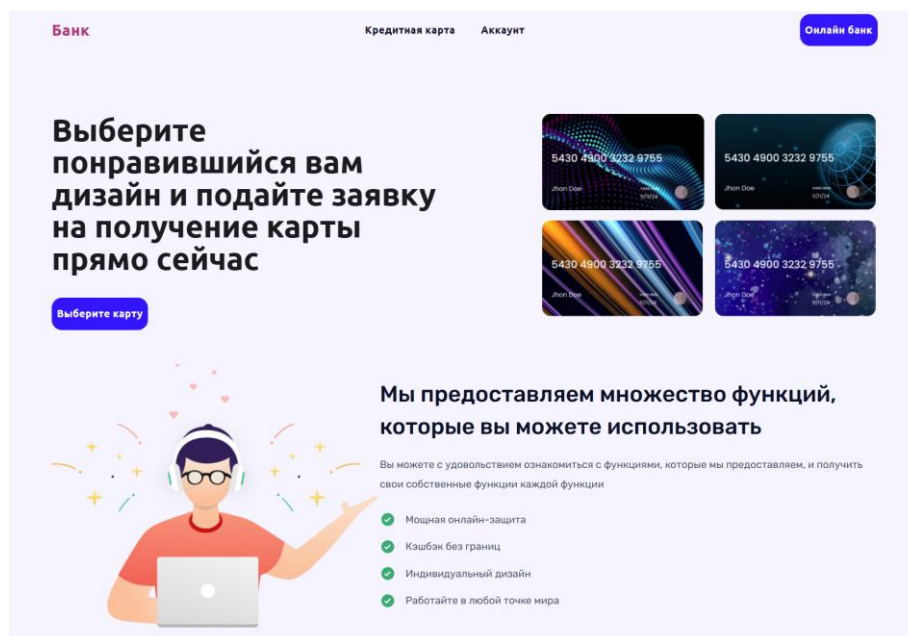


Рисунок 5 – Главная страница

Изм.	Лист	№ докум.	Подпись	Дата

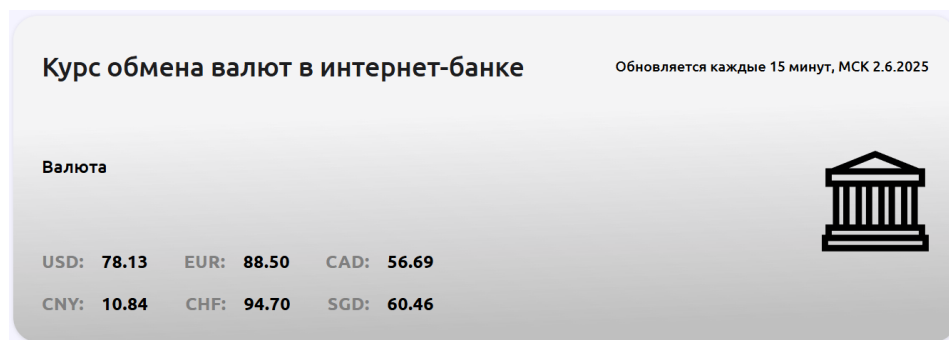


Рисунок 6 – Отображение курса валют банка

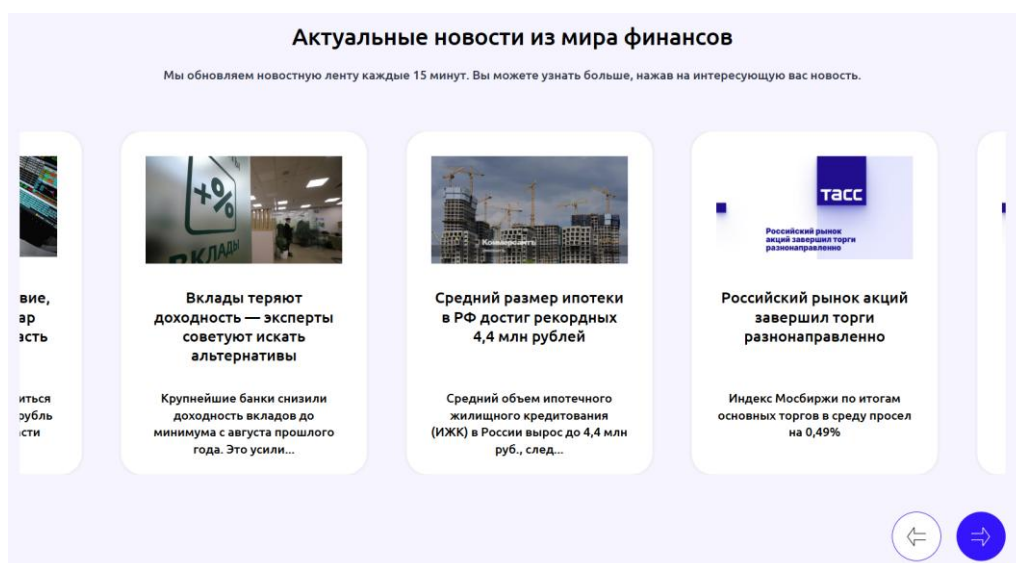


Рисунок 7 – Отображение актуальных новостей

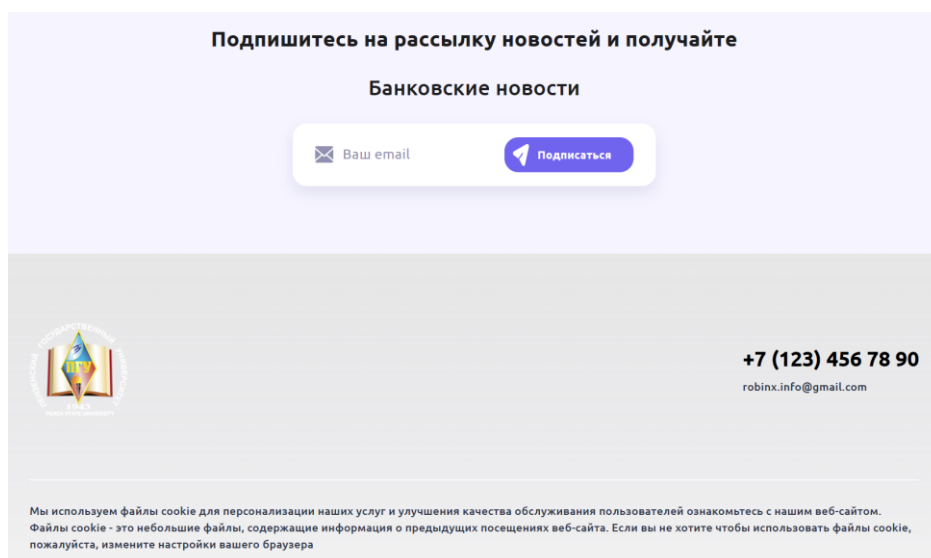


Рисунок 8 – Контактные данные и подписка на рассылку

Перед тем, как приступить к оформлению кредитной карты, необходимо зарегистрироваться, если пользователь впервые пользуется платформой, или авторизоваться, если пользователь ранее был зарегистрирован. Страница авторизации представлена на рисунке 9, а страница регистрации на рисунке 10;

Рисунок 9 – Страница авторизации

Рисунок 10 – Страница регистрации

После авторизации пользователю будет доступен личный кабинет. Если пользователь ранее оставлял заявки в сервисе, то в личном кабинете будут отображены оставленные заявки с визуализацией этапа, на котором клиент

Изм.	Лист	№ докум.	Подпись	Дата

остановился при её оформлении. Пример личного кабинета с заявками показан на рисунке 11.

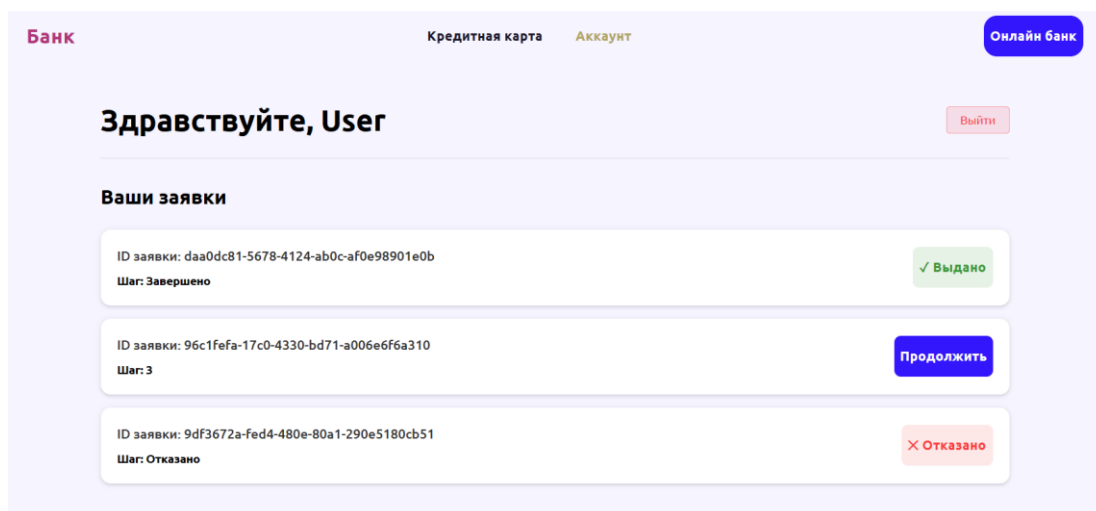


Рисунок 11 – Личный кабинет

Для того, чтобы приступить к оформлению кредитной карты, необходимо нажать на кнопку «Кредитная карта» в оглавлении любой страницы либо перейти на главную страницу и нажать на кнопку «Выберите карту». Страница первого этапа оформления кредитной карты показана на рисунках 12, 13.

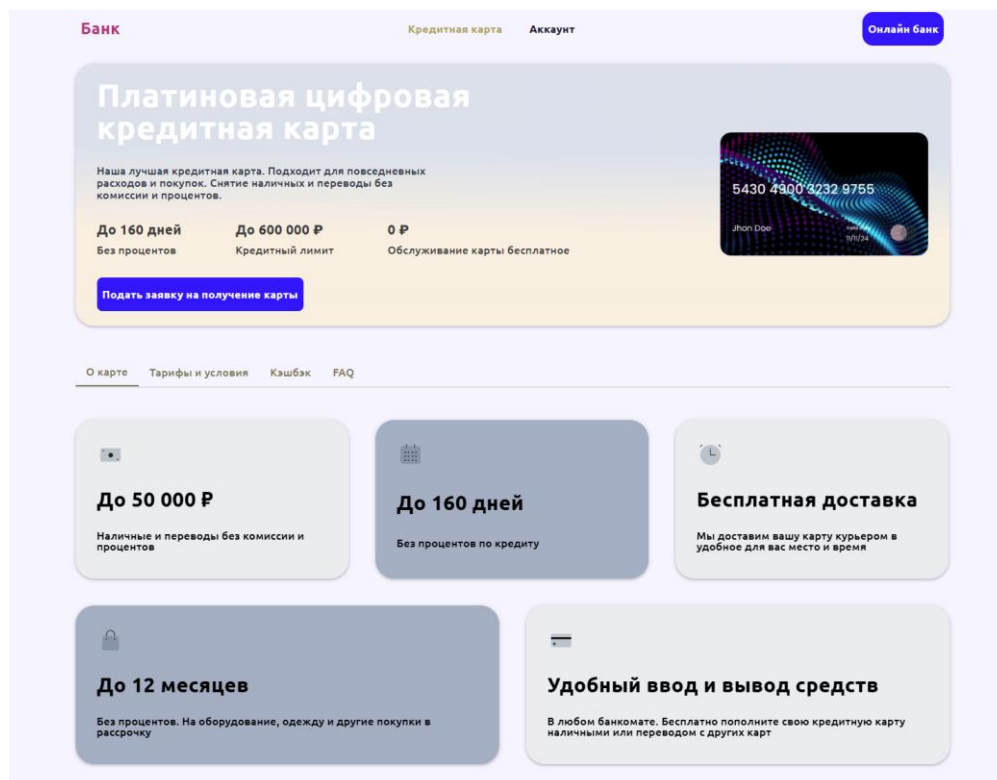


Рисунок 12 – Страница первого этапа оформления кредитной карты

Как получить карту

1

Заполните онлайн-заявку - вам не нужно посещать банк

2

Узнайте о решении банка сразу после заполнения заявления

3

Банк бесплатно доставит карту в любое удобное для вас место в вашем городе

Настройте свою карту

Шаг 1 из 5

Вы выбрали сумму

150 000 Р

Выберите сумму

150 000

15 000

600 000

Контактная информация

Имя *

Фамилия *

Отчество (при наличии)

Срок *

Почта *

Дата рождения *

Серия паспорта *

Номер паспорта *

Продолжить

Рисунок 13 – Первый этап оформления кредитной карты

Первый этап состоит из выбора суммы займа, его срока и ввода паспортных данных и электронной почты. После заполнения этих данных и нажатия кнопка «Продолжить» на той же странице отображается 4 предложения, различающиеся между собой процентной ставкой и суммой ежемесячных платежей, которые зависят от наличия страховки и зарплатного клиента. Пример таких предложений показан на рисунке 14.

Как получить карту

1

Заполните онлайн-заявку - вам не нужно посещать банк

2

Узнайте о решении банка сразу после заполнения заявления

3

Банк бесплатно доставит карту в любое удобное для вас место в вашем городе



Запрашивается: 150 000 Р
Общая сумма: 157 500 Р
На 6 месяцев
Ежемес. платёж: 27 488.52 Р
Ставка: 16%
Страхование включено ✓
Зарплатный клиент ✓

Выбрать



Запрашивается: 150 000 Р
Общая сумма: 157 500 Р
На 6 месяцев
Ежемес. платёж: 27 566.82 Р
Ставка: 17%
Страхование включено ✓
Зарплатный клиент ✗

Выбрать



Запрашивается: 150 000 Р
Общая сумма: 150 000 Р
На 6 месяцев
Ежемес. платёж: 26 403.55 Р
Ставка: 19%
Страхование включено ✗
Зарплатный клиент ✓

Выбрать



Запрашивается: 150 000 Р
Общая сумма: 150 000 Р
На 6 месяцев
Ежемес. платёж: 26 478.42 Р
Ставка: 20%
Страхование включено ✗
Зарплатный клиент ✗

Выбрать

Рисунок 14 – Выбор предложения на первом этапе

После выбора предложения клиенту на указанную почту будет отправлено письмо с уведомлением о завершении регистрации и ссылкой на следующий этап оформления кредитной карты. Пример сообщения показан на рисунке 15.

Регистрация завершена

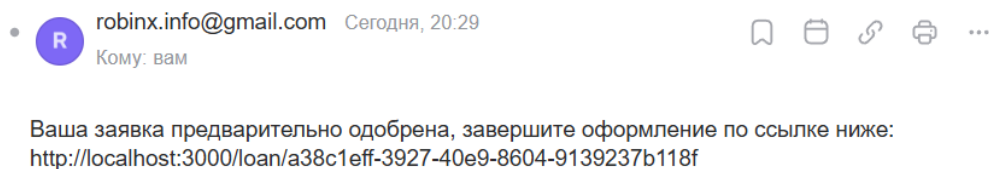


Рисунок 15 – Электронное письмо первого этапа

Для того, чтобы перейти к следующему этапу оформления кредитной карты есть три способа: остаться на той же странице после выбора предложения, перейти по ссылке в электронном письме или через личный кабинет, выбрав соответствующую заявку. При этом важно, чтобы клиент был авторизован в профиль, через который была подана эта заявка, поскольку другим пользователям будет недоступна ссылка этой заявки. Страница второго этапа представлена на рисунке 16.

Рисунок 16 – Второй этап подачи заявки

Второй этап состоит из заполнения оставшихся паспортных данных и данных о трудоустройстве клиента. На этом этапе возможны два исхода: отклонение или одобрение заявки. Заявка может быть отклонена, если:

- общий стаж менее восемнадцати месяцев;
- стаж на последней работе менее трёх месяцев;
- сумма займа больше, чем 10 зарплат;
- рабочий статус указан как «Безработный».

Пример электронных писем после второго этапа представлены на рисунках 17, 18.

Заявка отклонена

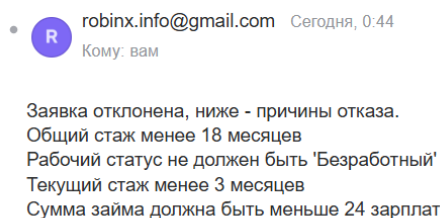


Рисунок 17 – Письмо с информированием об отклонённой заявке

Документы созданы

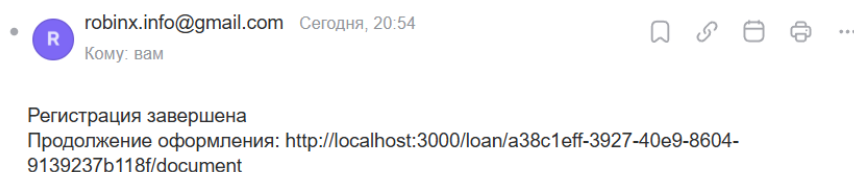


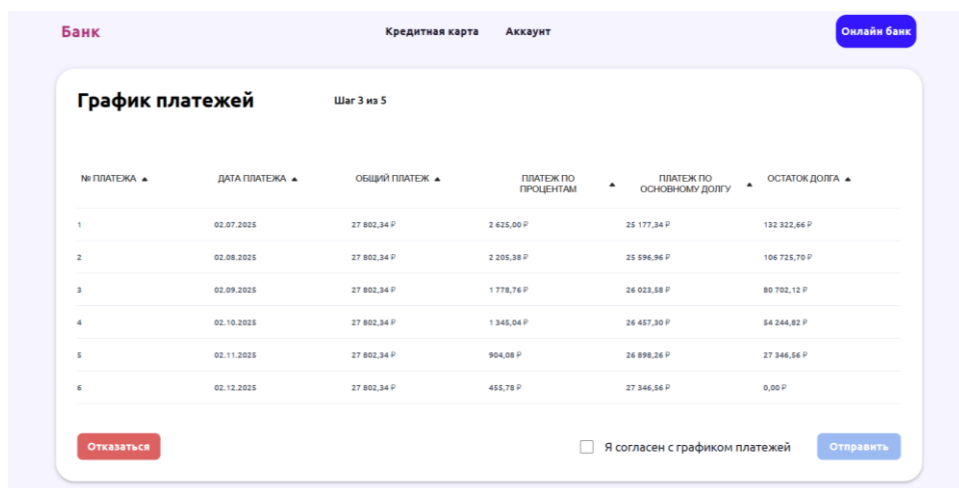
Рисунок 18 – Письмо с информированием об одобрении заявки

В случае, если был получен отказ, клиента перенаправит на главную страницу, где он сможет перейти в личный кабинет и увидеть статус своей заявки.

Переход к следующему этапу аналогичен предыдущему при условии, что заявка была одобрена. На третьем этапе будет показан подробный график платежей в течение всего периода, на котором отображены следующие поля: номер платежа, дата платежа, общий платёж, платёж по процентам, платёж по основному долгу, остаток долга. Таблицу можно отсортировать в порядке

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

возрастания или убывания по одному из полей. Третий этап представлен на рисунке 19.



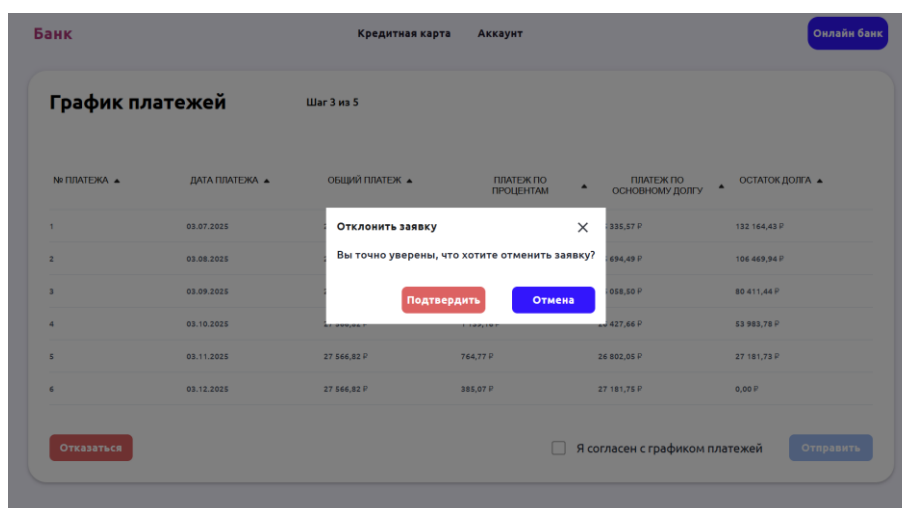
№ ПЛАТЕЖА	ДАТА ПЛАТЕЖА	ОБЩИЙ ПЛАТЕЖ	ПЛАТЕЖ ПО ПРОЦЕНТАМ	ПЛАТЕЖ ПО ОСНОВНОМУ ДОЛГУ	ОСТАТОК ДОЛГА
1	02.07.2025	27 802,34 P	2 635,00 P	25 177,34 P	132 322,66 P
2	02.08.2025	27 802,34 P	2 205,38 P	25 596,96 P	106 725,70 P
3	02.09.2025	27 802,34 P	1 778,76 P	26 023,58 P	80 702,12 P
4	02.10.2025	27 802,34 P	1 345,04 P	26 457,30 P	54 244,82 P
5	02.11.2025	27 802,34 P	904,08 P	26 898,26 P	27 346,56 P
6	02.12.2025	27 802,34 P	455,78 P	27 346,56 P	0,00 P

☐ Я согласен с графиком платежей

Отказаться Отправить

Рисунок 19 – Третий шаг подачи заявки

На этом этапе клиент может согласиться с графиком платежей, щёлкнув на соответствующую флаговую кнопку, либо отказаться от него, нажав на кнопку «Отказаться». После нажатия кнопки отказа появится модальное окно с подтверждением отмены заявки. После подтверждения пользователь будет возвращён на главную страницу, а заявка будет удалена из личного кабинета. Окно отказа показано на рисунке 20. Пример письма завершения третьего этапа показано на рисунке 21.



№ ПЛАТЕЖА	ДАТА ПЛАТЕЖА	ОБЩИЙ ПЛАТЕЖ	ПЛАТЕЖ ПО ПРОЦЕНТАМ	ПЛАТЕЖ ПО ОСНОВНОМУ ДОЛГУ	ОСТАТОК ДОЛГА
1	03.07.2025			335,57 P	132 164,43 P
2	03.08.2025			694,49 P	106 469,94 P
3	03.09.2025			938,50 P	80 411,44 P
4	03.10.2025			1 427,66 P	53 983,78 P
5	03.11.2025	27 566,82 P	764,77 P	26 802,05 P	27 181,73 P
6	03.12.2025	27 566,82 P	385,07 P	27 181,75 P	0,00 P

☐ Я согласен с графиком платежей

Отказаться Отправить

Рисунок 20 – Отказ от предложенного графика платежей

Документы отправлены

•  robinx.info@gmail.com Сегодня, 0:41
Кому: вам

Документы отправлены.
Подписать документы: <http://localhost:3000/loan/a38c1eff-3927-40e9-8604-9139237b118f/document/sign>

Рисунок 21 – Письмо с информированием об отправке документов

Способ перехода к четвёртому этапу аналогичен предыдущим этапам в случае согласия с графиком платежей. Предпоследний этап подачи заявки состоит из прочтения информации по выдаваемой кредитной карте и подписания согласия с условиями обслуживания этой карты. Страница подписания документов показана на рисунке 22.

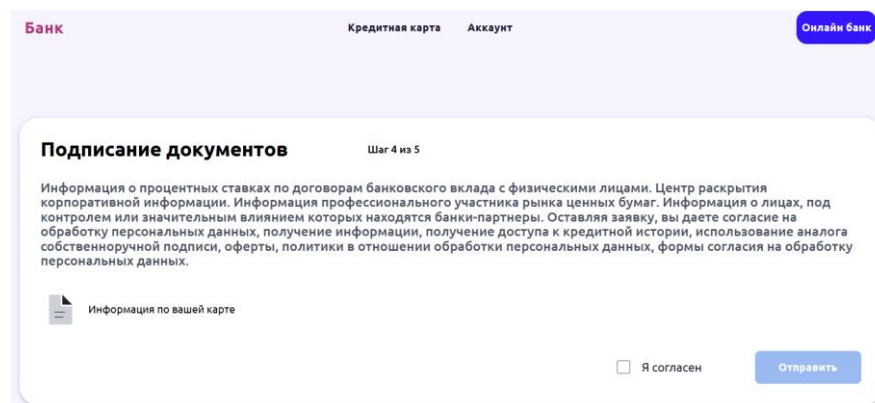


Рисунок 22 – Четвёртый шаг подачи заявки

Файл с условиями обслуживания можно загрузить по нажатию по полю с текстом «Информация по вашей карте». После этого будет загружен документ, пример которого изображен на рисунке 23.

До 160 дней без %			Использование кредита для выдачи наличных		Определяется банком партнёром после одобрения
Беспроцентный период	Пользуйтесь деньгами и не платите процент	До 160 дней (точные условия определяются тарифным планом банка-партнёра, который одобрил карту)	Минимальный платеж		3-10% от суммы задолженности, минимум 300 Р (точные условия определяются тарифным планом банка-партнёра, который одобрил карту)
	Возобновляется на следующий день	После полного погашения задолженности	Штрафы и неустойки		Определяется банком партнёром после одобрения
	Распространяется	На все типы операций (точные условия определяются тарифным планом банка-партнёра, который одобрил карту)	Неустойка при допущении просроченной задолженности по оплате минимального платежа		
Выдача наличных		До 300 000 Р в календарный месяц (точные условия определяются тарифным планом банка-партнёра, который одобрил карту)			
Пополнение счета карты		Бесплатно в Онлайн Банке			
Выпуск карты	Выпуск карты, в том числе дополнительной	Бесплатно			
	Перевыпуск карты по сроку действия	Бесплатно			
Процентная ставка	Использование кредита для всех типов операций в рамках беспроцентного периода	Определяется банком партнёром после одобрения			
	Использование кредита для оплаты товаров или услуг	Определяется банком партнёром после одобрения			

Рисунок 23 – Условия обслуживания карты

После отметки флага поля «Я согласен» и нажатия кнопки «Отправить», клиент получит письмо с информированием о подписании документов, с ссылкой на следующий этап и кодом подтверждения, необходимым для завершения подачи заявки. Пример такого письма показан на рисунке 24.

Код подтверждения

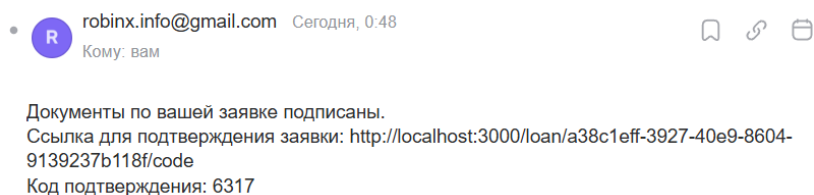


Рисунок 24 – Письмо с кодом подтверждения заявки

Способ перехода к завершающему этапу аналогичен предыдущим. Здесь необходимо ввести четырёхзначный код подтверждения, отправленный ранее на электронную почту. Страница пятого этапа показана на рисунке 25.

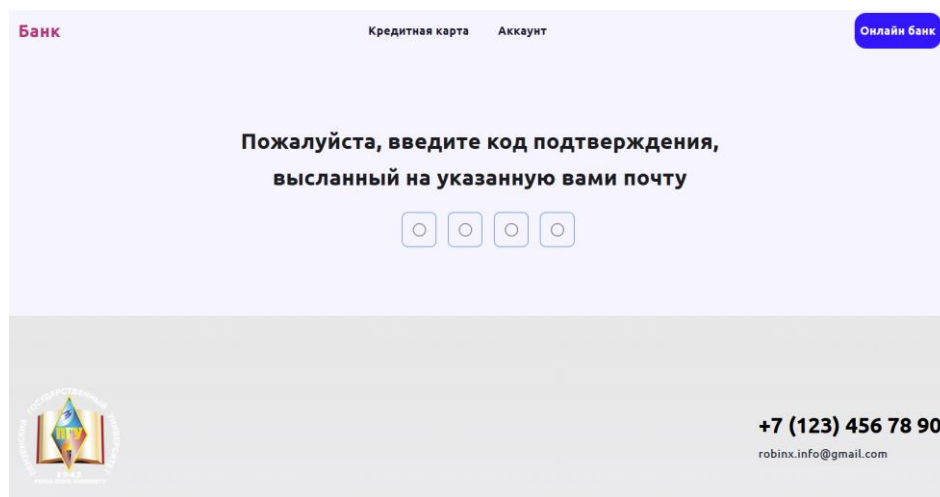


Рисунок 25 – Пятый этап получения кредитной карты

После ввода кода подтверждения откроется страница с поздравлением о завершении оформления кредитной карты, после чего будет открыта главная страница и процесс подачи заявки можно будет повторить снова. Страница завершения оформления кредитной карты показана на рисунке 26.

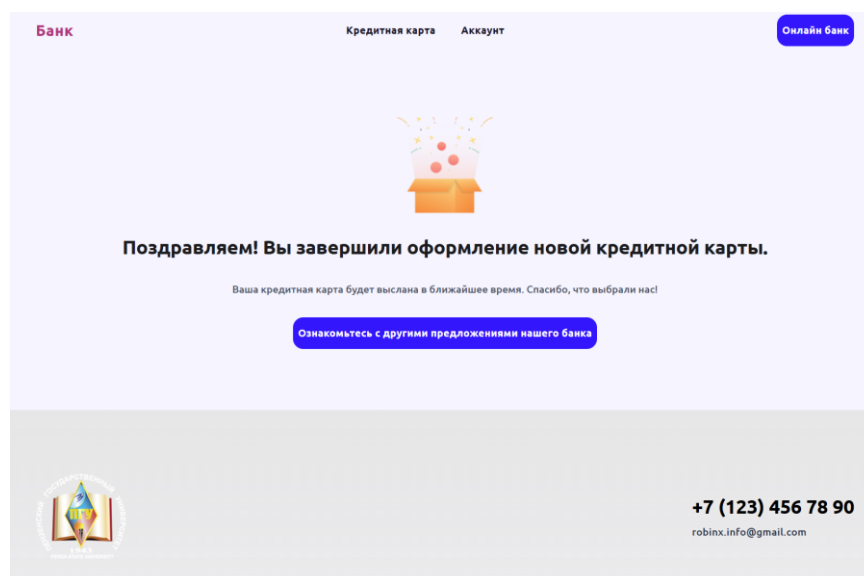


Рисунок 26 – Завершение оформления кредитной карты

3.3 Описание программы

Для работы системы кредитного банка были созданы пять микросервисов:

- Calculator: необходим для расчёта возможных условий кредита и полного расчета параметров кредита;

- Deal: необходим для взаимодействия с БД;
- Dossier: микросервис, отвечающий за отправку сообщений на электронную почту клиенту;
- Statement: необходим для валидации данных, полученных после оформления заявки на первом этапе;
- Gateway: микросервис, главная задача которого – перенаправлять запросы другим микросервисам, взаимодействуя с фронтендом.

Миграция БД – процесс управления изменениями в структуре БД при разработке приложения. Он позволяет автоматизировать применение изменений, отслеживать их историю и обеспечивать согласованность в разных окружениях. Миграция необходима для жёсткого контроля структуры БД, допускающая её изменение только через специальные changelog`и, в которых описываются изменения, вносимые в БД.

Перед началом разработки необходимо создать базу данных PostgreSQL с именем «Credit_bank» любым удобным способом, например, через инструмент для администрирования баз данных pgAdmin 4, представленном на рисунке 27.

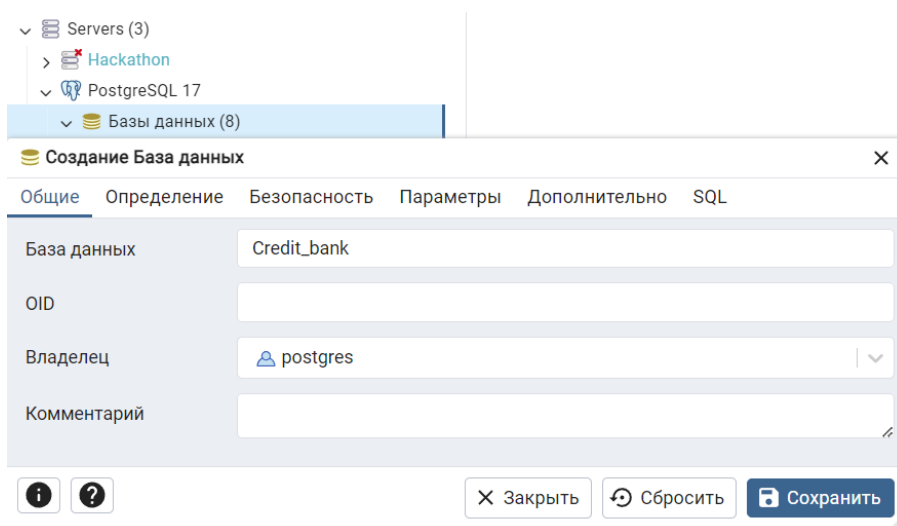


Рисунок 27 – Создание БД

Следующим шагом будет написание changelog`ов для инициализации таблиц и создания расширения для автоматического создания уникальных

идентификаторов при добавлении записей в таблицы. Пример такого changelog`а на рисунке 28.

```

1 <databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog" xmlns:xsi="http://www.
2   <changeSet id="import_uuid_generator" author="robinx51">
3     <sql>
4       CREATE EXTENSION IF NOT EXISTS "uuid-oss";
5     </sql>
6   </changeSet>
7   <changeSet id="create_table_client" author="robinx51"...>
25  <changeSet id="create_table_credit" author="robinx51">
26    <createTable tableName="credit">
27      <column name="credit_id" type="uuid" defaultValueComputed="uuid_generate_v4()">
28        <constraints primaryKey="true" primaryKeyName="pk_credit" nullable="false"/>
29      </column>
30      <column name="amount" type="numeric(38, 2)"/>
31      <column name="term" type="integer"/>
32      <column name="monthly_payment" type="numeric(38, 2)"/>
33      <column name="rate" type="numeric(38, 2)"/>
34      <column name="psk" type="numeric(38, 2)"/>
35      <column name="payment_schedule" type="jsonb"/>
36      <column name="insurance_enabled" type="boolean"/>
37      <column name="salary_client" type="boolean"/>
38      <column name="credit_status" type="character varying(255)"/>
39    </createTable>
40  </changeSet>
41  <changeSet id="create_table_statement" author="robinx51"...>
68 </databaseChangeLog>

```

Рисунок 28 – Changelog инициализации БД

Для того, чтобы внести изменения в существующую БД, необходимо в файле конфигурации указать источник подключения и данные авторизации пользователя PostgreSQL. На рисунке 29 представлены параметры подключения к БД.

```

spring.datasource.url=jdbc:postgresql://localhost:5432/Credit_Bank
spring.datasource.username=${db.username}
spring.datasource.password=${db.password}

```

Рисунок 29 – Параметры подключения к БД

Перед тем, как запустить процесс миграции, необходимо написать сущности БД, которые будут отражать структуру каждой таблицы. Пример такой сущности показан на рисунке 30.

```

11 > @ {...}
17 public class Account implements Serializable {
18     @Id
19     @Column(name = "login", nullable = false)
20     private String login;
21
22     @Column(name = "password", nullable = false)
23     private String password;
24 }

```

Рисунок 30 – Сущность таблицы account

После написания всех сущностей можно запускать приложение, после чего начнётся миграция базы данных и создание таблиц на основании changelog'ов, показанные на рисунке 31.

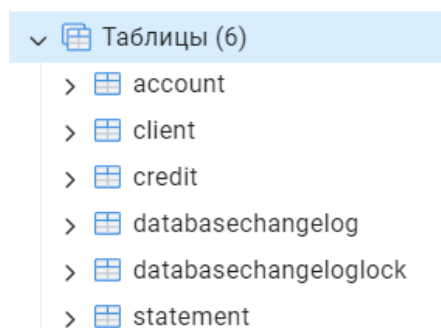


Рисунок 31 – Таблицы БД, созданные в результате миграции

Поскольку БД готова к работе, можно приступить к написанию контроллера для принятия запросов к бэкенду. Пример контроллера одного из микросервисов на рисунке 32.

```

20 @RestController  @ robinx51 *
21 @RequestMapping("/calculator")
22 public class CalculatorController {
23     private final OfferService offerService; 2 usages
24     private final CalcService calcService; 2 usages
25
26     @Autowired  @ robinx51
27     public CalculatorController(ScoringService scoringService) {...}
31
32     @PostMapping("/offers")  @ robinx51
33     public List<LoanOfferDto> createOffers(@RequestBody @Validated LoanStatementRequestDto request) {...}
36
37     @PostMapping("/calc")  @ robinx51
38     public CreditDto calculateCredit(@RequestBody @Validated ScoringDataDto request) {...}
41 }

```

Рисунок 32 – Контроллер микросервиса Calculator

Контроллер создан, обработчики запросов написаны, предварительно создав DTO с необходимыми для обмена полями. В таких классах можно

использовать различные аннотации, как стандартные, так и ручные, например, для валидации данных. Пример реализации такой аннотации на рисунках 33, 34.

```

12 @Constraint(validatedBy = LoanAmountValidator.class) 4 usages  ⚡ robinx51
13 @Target({ElementType.TYPE})
14 @Retention(RetentionPolicy.RUNTIME)
15 public @interface LoanAmountMoreThanSalaries {
16     String message() default "Сумма займа должна быть больше, чем 24 зарплат";
17
18     Class<?>[] groups() default {};  ⚡ robinx51
19
20     Class<? extends Payload>[] payload() default {};  ⚡ robinx51
21 }

```

Рисунок 33 – Аннотация проверки платёжеспособности клиента

```

10 public class LoanAmountValidator implements 2 usages  ⚡ robinx51 *
11     ConstraintValidator<LoanAmountMoreThanSalaries, ScoringDataDto> {
12     @Override  ⚡ robinx51 *
13     @@@ public boolean isValid(ScoringDataDto loanRequestDto, ConstraintValidatorContext context) {
14         BigDecimal loanAmount = loanRequestDto.getAmount();
15         BigDecimal salary = loanRequestDto.getEmployment().getSalary();
16
17         if (loanAmount == null || salary == null) {
18             return true;
19         }
20
21         BigDecimal requiredAmount = salary.multiply(new BigDecimal( val: 24));
22
23         if (loanAmount.compareTo(requiredAmount) >= 0) {
24             context.disableDefaultConstraintViolation();
25             context.buildConstraintViolationWithTemplate(
26                 messageTemplate: "Сумма займа должна быть меньше 24 зарплат") ConstraintViolationBuilder
27                 .addPropertyNode( name: "amount") NodeBuilderCustomizableContext
28                 .addConstraintViolation();
29             return false;
30         }
31
32         return true;
33     }
34 }

```

Рисунок 34 – Класс проверки платёжеспособности клиента

После создания DTO необходимо написать сервисы, которые будут обрабатывать данные, которые получил контроллер. Для работы микросервиса Calculator необходимы три сервиса:

- CalcService: необходим для формирования ответа для запроса по полному расчету параметров кредита;
- ScoringService: необходим для расчёта полей кредитного предложения в зависимости от базовой ставки, установленной в файле конфигураций, и оценки платёжеспособности клиента;

– OfferService: необходим для формирования ответа для запроса по расчёту возможных условий кредита.

Также необходимо обратить внимание на безопасную передачу данных авторизации и их шифрование перед сохранением в БД.

Процесс регистрации устроен таким образом: клиент вводит логин и пароль дважды, для каждого из этих полей написаны регулярные выражения таким образом, что логин может быть только буквами латинского алфавита, цифрами и символами точки или нижнего подчёркивания, а пароль обязательно должен иметь буквы и цифры и может иметь специальные символы. После этого создаётся запрос на бэкэнд с телом запроса регистрационных данных. Микросервис Gateway принимает запрос и отправляет запрос в микросервис Deal для проверки уникальности логина в БД. В случае, если такой логин найден, пользователь увидит соответствующую надпись на странице регистрации. В противном случае, пароль будет зашифрован через библиотеку bcrypt и отправлен в тот же микросервис для сохранения профиля в БД, а также будут сгенерированы и сохранены в текущей сессии уникальные токены авторизации, которые будут проверяться при авторизации и при других запросах от фронтенда. Этот токен авторизации и логин будут сохранены в куках ответа с такими настройками, что токен будет действовать в течение суток, а после этого потребуется повторная авторизация.

Авторизация работает так: клиент вводит данные на странице авторизации и отправляет запрос к бэкенду. Далее следует запрос к БД по логину с ответом в виде того же логина и зашифрованного пароля, и также через библиотеку bcrypt проверяется валидность пароля. В случае, если данные верны, в куки устанавливается сгенерированный токен и логин, а в тело ответа помещаются заявки, оставленные клиентом ранее, если таковы имеются. Если же данные окажутся неверными, то будет возвращён ответ со статусом 401 (Unauthorized). Фронтенд это обработает и сообщит клиенту о неправильности введённых данных.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		31

3.4 Тестирование

Были проведены ручные и автоматизированные тесты. Ручные тесты показаны на рисунках 35, 36, 37, 38. Автоматизированные тесты показаны на рисунке 39.

Контактная информация

Имя *	Фамилия *	Отчество (при наличии)	Срок *
<input type="text" value="Например: Иван"/>	<input type="text" value="Например: Иванов"/>	<input type="text" value="Например: Иванович"/>	<input type="text" value="6 месяцев"/>
<small>Значение не должно быть короче 2-х символов</small>	<small>Значение не должно быть короче 2-х символов</small>		
Почта *	Дата рождения *	Серия паспорта *	Номер паспорта *
<input type="text" value="test@gmail.com"/>	<input type="text" value="01/01/2000"/>	<input type="text" value="0000"/>	<input type="text" value="000000"/>
<small>Проверьте корректность email-адреса</small>	<small>Введите корректную дату</small>	<small>Проверьте корректность серии паспорта</small>	<small>Проверьте корректность номера паспорта</small>

Рисунок 35 – Тестирование пустых значений первого шага

Контактная информация

Имя *	Фамилия *	Отчество (при наличии)	Срок *
<input type="text" value="1."/>	<input type="text" value="1."/>	<input type="text" value="1."/>	<input type="text" value="6 месяцев"/>
<small>Значение должно содержать только буквы</small>	<small>Значение должно содержать только буквы</small>	<small>Значение должно содержать только буквы</small>	
Почта *	Дата рождения *	Серия паспорта *	Номер паспорта *
<input type="text" value="admin@admin.com"/>	<input type="text" value="19/04/2030"/>	<input type="text" value="111"/>	<input type="text" value="22222"/>
<small>Проверьте корректность email-адреса</small>	<small>Дата не может быть больше сегодняшней</small>	<small>Проверьте корректность серии паспорта</small>	<small>Проверьте корректность номера паспорта</small>

Рисунок 36 – Тестирование валидации первого шага

Продолжение подачи заявки

Шаг 2 из 5

Укажите ваш пол *	Семейное положение *	Количество иждивенцев *
<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value="Например: 1"/>
<small>Выберите один из вариантов</small>	<small>Выберите один из вариантов</small>	<small>Обязательное поле</small>
Дата выдачи паспорта *	Код подразделения *	
<input type="text" value="Например: 01/01/2024"/>	<input type="text" value="000-000"/>	
<small>Обязательное поле</small>	<small>Обязательное поле</small>	
Трудоустройство		
Трудовой статус *	ИНН *	Зарплата *
<input type="text" value=""/>	<input type="text" value="000000000000"/>	<input type="text" value="Например: 100 000"/>
<small>Выберите один из вариантов</small>	<small>Обязательное поле</small>	<small>Введите свою зарплату</small>
Должность *	Общий стаж работы в месяцах *	Текущий опыт работы в месяцах *
<input type="text" value=""/>	<input type="text" value="Например: 18"/>	<input type="text" value="Например: 3"/>
<small>Выберите один из вариантов</small>	<small>Обязательное поле</small>	<small>Обязательное поле</small>

Рисунок 37 – Тестирование пустых значений второго шага

Дата выдачи паспорта *	Код подразделения *
<input type="text" value="11/11/202"/>	<input type="text" value="111-11"/>
<small>Неверная дата выдачи паспорта</small>	<small>Серия должна состоять из 6 цифр</small>
Трудоустройство	
Трудовой статус *	ИНН *
<input type="text" value=""/>	<input type="text" value="11111111111"/>
<small>Выберите один из вариантов</small>	<small>ИНН должен состоять из 12 цифр</small>
Зарплата *	
<input type="text" value="Например: 100 000"/>	
<small>Введите свою зарплату</small>	

Рисунок 38 – Тестирование валидации второго шага

✓ Test Results	5 sec 228 ms
✓ CalculatorExceptionHandlerTest	1 sec 387 ms
✓ testHandleValidationErrors()	1 sec 387 ms
✓ OfferServiceTest	254 ms
✓ generateOffers()	254 ms
> ✓ ScoringServiceTest	35 ms
> ✓ CalculatorControllerOffersTest	56 ms
✓ CalculatorControllerCalcTest	218 ms
✓ testValidScoringDataDto()	188 ms
✓ testResponseCreditDto()	8 ms
✓ testInvalidScoringDataDto()	22 ms
✓ DealServiceTest	1 sec 530 ms
✓ selectOffer()	1 sec 512 ms
✓ processClient()	8 ms
✓ finishRegistration()	10 ms
✓ MsGatewayApplicationTests	1 sec 183 ms
✓ contextLoads()	1 sec 183 ms
✓ StatementControllerTest	565 ms
✓ testSelectOffer()	362 ms
✓ testInitialRegistration_InvalidRequest()	96 ms
✓ testInitialRegistration()	107 ms

Рисунок 39 – Тестирование бэкенда

Результат тестирования представлен в таблице 2.

Таблица 2 – Результаты тестирования

№ теста	Что проверяется	Ожидаемый результат	Фактический результат
1	Поведение формы первого этапа при пустых обязательных полях	Рядом с такими полями появится предупреждение	Совпадает с ожидаемым результатом
2	Поведение полей формы первого этапа при вводе недопустимых символов	Рядом с такими полями появится предупреждение	Совпадает с ожидаемым результатом
3	Поведение формы второго этапа при пустых обязательных полях	Рядом с такими полями появится предупреждение	Совпадает с ожидаемым результатом
4	Поведение полей формы второго этапа при вводе недопустимых символов	Рядом с такими полями появится предупреждение	Совпадает с ожидаемым результатом
5	Работа основных функций бэкенда	Все тесты будут завершены без ошибок	Совпадает с ожидаемым результатом

4 Безопасность жизнедеятельности

4.1 Правила безопасности при использовании ПК

В данном разделе необходимо учесть риски, перечисленные в ГОСТ 12.0.003-2015 "Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности". Ниже приведен перечень рисков, которым следует придать особое значение:

- риск повреждения зрения и утомления глаз;
- длительное использование ПК без перерывов и отдыха глаз;
- плохая эргономика рабочего места, включая неправильное расположение монитора, отражение света на экране и недостаточное освещение;
- риск травм и заболеваний опорно-двигательной системы: неправильное положение тела при работе с ПК, например, неправильная осанка, неправильная поза или неправильное расположение рук и ног;
- отсутствие эргономической мебели и принадлежностей, таких как стулья с подлокотниками и подставки для рук;
- риск воздействия электромагнитных полей: использование устаревшего или неисправного оборудования, которое может создавать сильные электромагнитные поля;
- неправильное расположение оборудования, например, слишком близко к рабочему месту или другим электронным устройствам;
- риск пожара и электрического удара: неправильное подключение и использование электрооборудования, включая перегрузку розеток и использование поврежденных кабелей;
- нарушение правил пожарной безопасности, включая использование запрещенных электроприборов вблизи ПК и наличие горючих материалов рядом с оборудованием;

– риск нарушения информационной безопасности: небезопасное использование паролей и учетных записей, включая слабые пароли или их совместное использование;

– несанкционированный доступ к компьютерной информации, включая утечку конфиденциальных данных;

– риск воздействия шума: использование шумных компьютерных компонентов или работа в шумной среде.

4.2 Правила безопасности при использовании смартфона

В этом разделе следует уделить внимание рискам, связанным с безопасностью при использовании данного устройства. А также разработать свод правил, чтобы гарантировать предотвратить следующие риски:

– риск зависимости и негативного воздействия на психологическое и физическое здоровье: чрезмерное использование смартфона, включая зависимость от социальных сетей, игр и постоянного онлайн-присутствия, что может негативно сказываться на здоровье и социальной активности.

– риск отрицательного влияния на сон и здоровье: использование смартфона перед сном, из-за светового излучения экрана, может нарушать естественный сон и вызывать бессонницу;

– чрезмерное использование смартфона в темное время суток может снижать уровень мелатонина, гормона сна;

– риск потери или кражи смартфона: несанкционированное получение доступа к устройству, возможность потери или кражи смартфона, что может привести к утечке личных данных или использованию их в недобросовестных целях;

– нарушение конфиденциальности и безопасности данных в случае потери или кражи смартфона;

– риск воздействия радиочастотного излучения: возможные воздействия радиочастотного излучения на организм при длительном использовании

смартфона, хотя точные последствия такого воздействия все еще находятся в стадии исследования;

- при работе в плохо покрытых районах мобильной связи смартфон может выделять больше энергии для подключения к сети, что может привести к повышенному радиочастотному излучению;

- риск отвлечения водителя: использование смартфона во время вождения может отвлекать водителя и повышать риск дорожно-транспортных происшествий. Это особенно актуально в случае чтения сообщений, отправки сообщений или проведения телефонных разговоров во время движения.

С учетом указанных рисков, следует разработать и рекомендовать следующие правила безопасности при использовании смартфона:

- ограничьте время использования смартфона, особенно перед сном;
- соблюдайте эргономику и правильное положение тела при работе со смартфоном;
- используйте оригинальные зарядные устройства и кабели, избегайте перегрева и короткого замыкания;
- устанавливайте надежные пароли и используйте функции блокировки экрана для защиты данных;
- храните смартфон в безопасном месте и будьте бдительны в общественных местах;
- регулярно делайте резервные копии данных и используйте антивирусное программное обеспечение для защиты от вредоносных программ;
- ограничьте использование смартфона во время вождения и соблюдайте правила безопасности на дороге.

Правильное соблюдение этих правил поможет минимизировать риски и обеспечить безопасное использование смартфона.

5 Стартап проект

5.1 Аннотация проекта

В современном мире финансовые услуги становятся всё более цифровыми, и клиенты ожидают удобства, скорости и прозрачности при оформлении кредитов.

Целью проекта является создание онлайн-платформы для автоматизации кредитных процессов с использованием микросервисной архитектуры, REST API, базы данных PostgreSQL и механизма отправки уведомлений на электронную почту для информирования клиентов о статусе их заявок.

Основные задачи проекта направлены на разработку программного обеспечения в виде онлайн платформы для выдачи кредитных карт.

Предлагаемый продукт предназначен для персональных устройств Laptop/Desktop под управлением Windows, Linux и macOS, в том числе мобильных устройств под управлением iOS/Android. Реализация продукта организуется по модели «Software as a Service» - пользование программным обеспечением как услугой.

Потребителями являются физические (b2c) и юридические лица (частные (b2b) и государственные организации (b2g)), работающие в сферах малого и среднего бизнеса, банков и финансовых организациях.

5.2 Базовая бизнес-идея

Разрабатываемое приложение представляет собой онлайн-сервис для оформления кредитных карт, где пользователь может быстро подать заявку, выбрать подходящие условия и завершить процесс полностью дистанционно. Программа предназначена для персональных устройств Laptop / Desktop под управлением Windows, Linux или macOS, в том числе мобильных устройств под управлением iOS или Android.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		37

Современный ритм жизни требует мгновенного доступа к финансовым услугам. Пользователи хотят избежать длительного ожидания в очередях, бумажной волокиты и сложных процедур верификации. Им важно получить предварительное решение по заявке в течение нескольких минут, сравнить условия кредитования и выбрать наиболее подходящий вариант.

Кредитное приложение предназначено для широкого круга пользователей, которым важно быстро и удобно оформить кредитный продукт без необходимости посещения банковского отделения. В первую очередь, это люди, ценящие свое время и предпочитающие цифровые сервисы традиционным методам взаимодействия с финансовыми организациями.

Предлагаемый продукт решает проблему в следующих областях:

- частные клиенты (люди, которые хотят быстро и удобно оформить кредит без необходимости посещения банковского отделения);
- малый и средний бизнес (предприниматели, которым необходимы оперативные кредитные решения для развития своего бизнеса);
- банки и финансовые организации (кредитные учреждения, заинтересованные в автоматизации своих процессов и снижении операционных издержек);
- финтех-компании: организации, которые могут использовать платформу как часть своих услуг для клиентов.

5.3 Характеристика будущего продукта или услуги

Разрабатываемое программное обеспечение для онлайн-оформления кредитных карт будет обладать следующими ключевыми характеристиками:

- полностью цифровой процесс оформления кредитной карты, пятиэтапная процедуры подачи заявки, возможность сохранения и продолжения заявки на любом этапе, поддержка работы на различных устройствах и браузерах (Chrome, Firefox, Safari, Edge);

– многоуровневая система оценки заявок, встроенный прескоринг для предварительной оценки кредитоспособности, скоринговая система для детальной оценки рисков, автоматическая генерация четырех вариантов кредитных предложений;

– личный кабинет пользователя, отслеживание статуса заявки, история поданных заявок, возможность просмотра и сравнения кредитных условий, отображение текущего этапа оформления;

– многофакторная система безопасности, шифрование персональных данных, безопасное хранение учетных записей в базе данных, криптографическая защита паролей;

– сопровождение заявок email-письмами, моментальные уведомления о статусе заявки, встроенные информационные виджеты (курсы валют, новости).

5.4 Организационные, производственные и финансовые параметры

5.4.1 Внутренние процессы организации бизнеса

Исследование и разработка (R&D) включают глубокий анализ рынка банковских услуг, изучение потребностей пользователей в цифровом кредитовании, а также оценку конкурентных решений, таких как Т-Банк и Альфа-Банк. На этапе проектирования разрабатывается архитектура приложения на основе микросервисов, что обеспечивает гибкость и масштабируемость. Прототипирование позволяет создать первые рабочие версии ключевых модулей, таких как prescoring и электронная подпись документов, для тестирования и валидации концепции.

Производственный процесс начинается с подбора необходимого оборудования, включая аренду облачных серверов для развертывания серверной части и организации рабочих мест для команды разработчиков. Производственные циклы строятся по методологии Agile, что позволяет оперативно вносить изменения и адаптировать продукт под требования

пользователей. Контроль качества включает проведение модульных, интеграционных и нагрузочных тестов, а также проверку безопасности передачи данных.

Логистика и поставки ограничиваются закупкой лицензий на программное обеспечение, таких как PostgreSQL и инструменты для CI/CD, а также организацией SMTP-сервиса для рассылки уведомлений. Управление запасами не требуется, так как продукт является цифровым.

5.4.2 Ключевые характеристики производства

Технологическая гибкость обеспечивается за счет использования современных фреймворков (Spring Boot, React) и поддержки различных устройств и браузеров. Высокая точность достигается благодаря строгой типизации (TypeScript) и автоматизированному тестированию. Скорость вывода на рынок ускоряется за счет модульной архитектуры и готовых решений, таких как Swagger для документирования API.

5.4.3 Объемы и разнообразие требуемых ресурсов

Человеческие ресурсы включают команду backend- и frontend-разработчиков, тестировщиков, DevOps-инженера и аналитика. Финансовые ресурсы направлены на начальные инвестиции в облачную инфраструктуру, лицензии и заработную плату сотрудников, а также операционные расходы на маркетинг и поддержку пользователей.

5.5 Основные конкурентные преимущества

Разрабатываемая онлайн-платформа для автоматизации кредитных процессов обладает рядом ключевых преимуществ. Во-первых, это скорость обработки заявок — благодаря prescoring'у и автоматизированному скорингу пользователь получает предварительное решение в течение нескольких минут, что выгодно отличает решение от аналогов, где проверка может занимать до суток.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		40

Во-вторых, прозрачность условий кредитования. Платформа предоставляет четыре варианта условий (например, со страховкой или без), что позволяет клиенту выбрать оптимальный продукт. Это отсутствует у многих конкурентов, где условия часто стандартизированы.

В-третьих, интеграция дополнительных функций, таких как отображение курсов валют и финансовых новостей, повышает ценность продукта для пользователей, которые хотят оставаться в курсе экономических событий.

В-четвертых, безопасность данных обеспечивается шифрованием паролей, двухфакторной аутентификацией и использованием биометрических методов подтверждения, что соответствует лучшим практикам рынка.

Наконец, удобство продолжения оформления кредита. Пользователь может в любой момент вернуться к незавершенной заявке через личный кабинет, что сокращает время на повторное заполнение данных и повышает конверсию.

5.6 Научно-техническое решение и/или результаты, необходимые для создания продукции

Для разработки онлайн-платформы автоматизации кредитных процессов применяются следующие ключевые решения:

- микросервисная архитектура на базе REST API. Взаимодействие между модулями (prescoring, скоринг, работа с документами) реализовано через HTTP-запросы, что обеспечивает гибкость и простоту масштабирования;

- оптимизированная обработка данных:

- 1) prescoring выполняется на основе минимального набора данных (ФИО, доход), что ускоряет предварительное одобрение заявки;

- 2) скоринг использует алгоритмы оценки кредитоспособности, интегрированные с базой данных PostgreSQL для хранения истории заявок;

- 3) работа с документами включает генерацию PDF-договоров и их отправку через SMTP (Google Workspace).

- безопасность и управление доступом:

1) аутентификация пользователей через логин и пароль с хешированием bcrypt;

2) защита персональных данных на уровне базы (шифрование чувствительных полей).

– интеграция внешних сервисов:

1) курсы валют загружаются из внешнего API и кэшируются для снижения нагрузки;

2) финансовые новости отображаются на главной странице, повышая вовлеченность пользователей.

– фронтенд на React + TypeScript:

1) динамические формы с валидацией (React Hook Form + Zod);

2) состояние приложения управляется через Redux Toolkit с сохранением сессии (Redux Persist);

5.7 Финансовые параметры

Основные источники дохода:

– комиссия банков-партнеров за каждую одобренную заявку (10–15% от суммы кредита);

– платные интеграции с другими fintech-сервисами (например, страхование).

5.8 Взаимоотношения с клиентами

Для обеспечения долгосрочного взаимодействия с пользователями и партнерами (банками) предусмотрены следующие меры:

– техническая поддержка и консультации:

1) круглосуточная поддержка через онлайн-чат и email для оперативного решения проблем пользователей;

2) отдельная линия для банков-партнеров по вопросам интеграции и отчетности;

3) база знаний с FAQ и видеоинструкциями по оформлению заявок.

– обратная связь и обновления:

1) регулярный сбор отзывов через встроенную анкету в личном кабинете;

2) ежеквартальные обновления функционала (например, добавление новых способов верификации);

3) публичная дорожная карта развития платформы для повышения прозрачности.

– программы лояльности:

1) бонусы для частых клиентов (снижение процентной ставки при повторных заявках);

2) партнерские вознаграждения банкам за привлечение пользователей.

5.9 Оценка затрат и доходов

Примерные затраты в первый год показаны в таблице 3.

Таблица 3 – Затраты

Категория	Сумма (руб)	Обоснование
Заработная плата	2 250 000	Команда из 5 человек: 2 backend-разработчика, 1 frontend-разработчик, 1 тестировщик, 1 менеджер.
Лицензии и оборудование	1 100 000	Подписки на ПО (JetBrains, Figma), аренда серверов (Docker-контейнеры), домены.
Аренда офиса	1 250 000	Коворкинг для команды на 6 месяцев (после — удаленная работа).
Коммунальные услуги	250 000	Интернет, электроэнергия, облачные сервисы (Google Workspace).
Маркетинг	1 000 000	Контекстная реклама, SEO-продвижение, участие в fintech-конференциях.
Итого	5 850 (первый год)	

Ожидаемые доходы после интеграции проекта:

- комиссионные от банков: 9 000 тыс. руб. (3000 одобренных заявок со средней комиссией 3 тыс. руб.);
- дополнительные услуги: 500 тыс. руб. (премиум-поддержка для банков);
- чистая прибыль: $9\,500 - 5\,850 = 3\,650$ тыс. руб.

Срок окупаемости:

- первоначальные затраты: 5 850 тыс. руб;
- среднемесячная прибыль: $3\,650 / 12 \approx 304$ тыс. руб;
- окупаемость: $5\,850 / 304 \approx 19$ месяцев (1 год и 7 месяцев).

					ПГУ1.090301.09.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

Результат выполнения данной выпускной квалификационной работы можно считать успешно реализованным в соответствии с заданием «Бизнес-проект: «Кредитный банк: онлайн-платформа для автоматизации кредитных процессов»

Были выполнены следующие задачи:

– анализ процессов и потребностей кредитного банка. Были исследованы требования и ожидания клиентов, что позволило определить функциональность и основные модули системы;

– выбор языка программирования. Был выбран Java в качестве языка программирования для разработки онлайн платформы. Java обладает надежной экосистемой, высокой производительностью и широкими возможностями для создания масштабируемых enterprise-решений, что критично для банковской сферы;

– выбор фреймворков и технологий. Для backend-разработки был использован Spring Boot, обеспечивающий быструю разработку микросервисов, безопасность и интеграцию с базами данных. Frontend реализован на React с TypeScript, что позволило создать интуитивно понятный интерфейс с строгой типизацией;

– выбор системы управления базами данных. В качестве СУБД выбрана PostgreSQL, которая обеспечивает надежное хранение данных, поддержку транзакций и высокую производительность при работе с большими объемами информации;

– проектирование архитектуры. Была разработана микросервисная архитектура с REST API, что обеспечило гибкость, масштабируемость и простоту поддержки системы;

– реализация функционала. Созданы ключевые модули: авторизация, личный кабинет, prescoring и скоринг заявок, интеграция с внешними API (курсы валют, новости), а также механизм отправки уведомлений через SMTP.

В результате разработана онлайн-платформа, которая автоматизирует процесс оформления кредитных продуктов, сокращает время обработки заявок и повышает удобство для клиентов. Платформа готова к интеграции с банками и масштабированию.

Перспективы развития включают: внедрение двухфакторной аутентификации для повышения безопасности, интеграцию с дополнительными финансовыми сервисами (например, страхование), использование машинного обучения для более точного скоринга, добавление панели администратора для ручного управления и проверки оформляемых заявок.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		46

Список использованных источников

1. Электронные деньги в коммерческом банке. Практическое пособие / А.В. Пухов и др. – М.: КноРус, ЦИПСИР, 2015. – 208с.
2. Молотников, А. Е. Правовое регулирование рынка ценных бумаг: Моногр. – М.: Стартап, 2013. – 552 с.
3. Павлов, П. В. Законодательство в области финансов, банков и бухгалтерского учета. – М.: Магистр, 2008. – 240 с.
4. Вернон В. Реализация методов предметно-ориентированного проектирования. — М.: Вильямс, 2020. — 688 с.

Приложение А

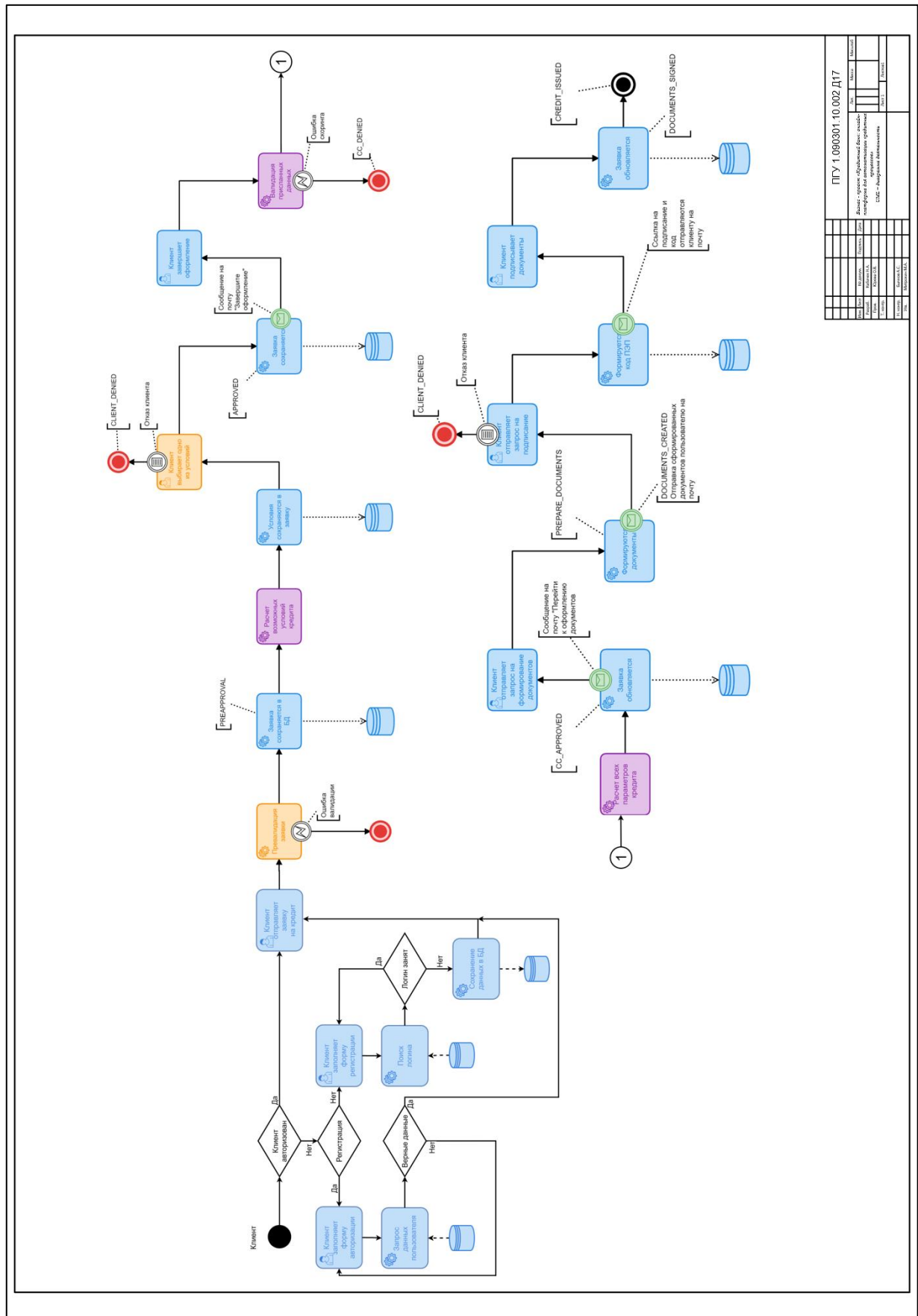
(обязательное)

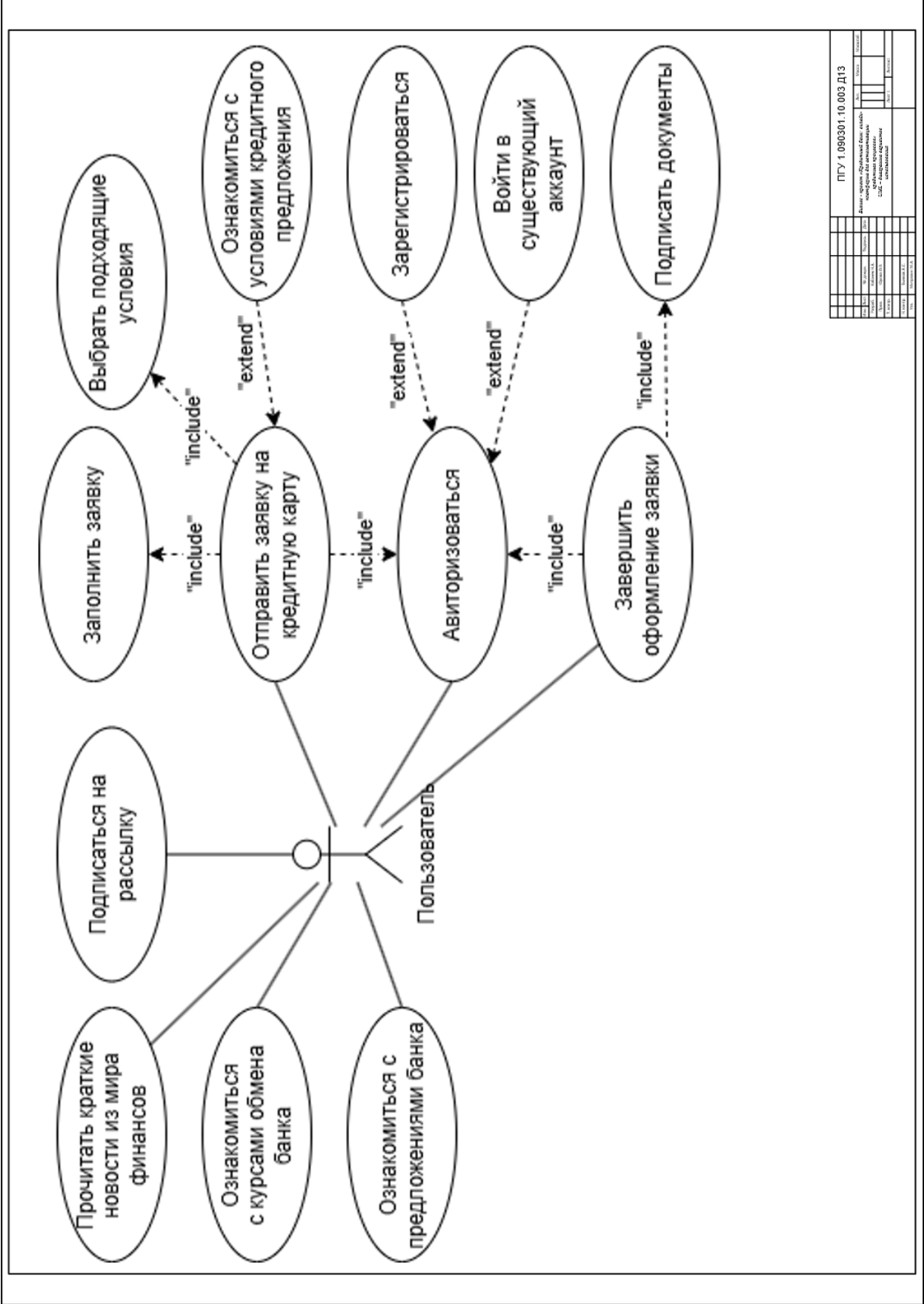
Бизнес - проект «Кредитный банк: онлайн-платформа для автоматизации
кредитных процессов»

UML диаграммы

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		48

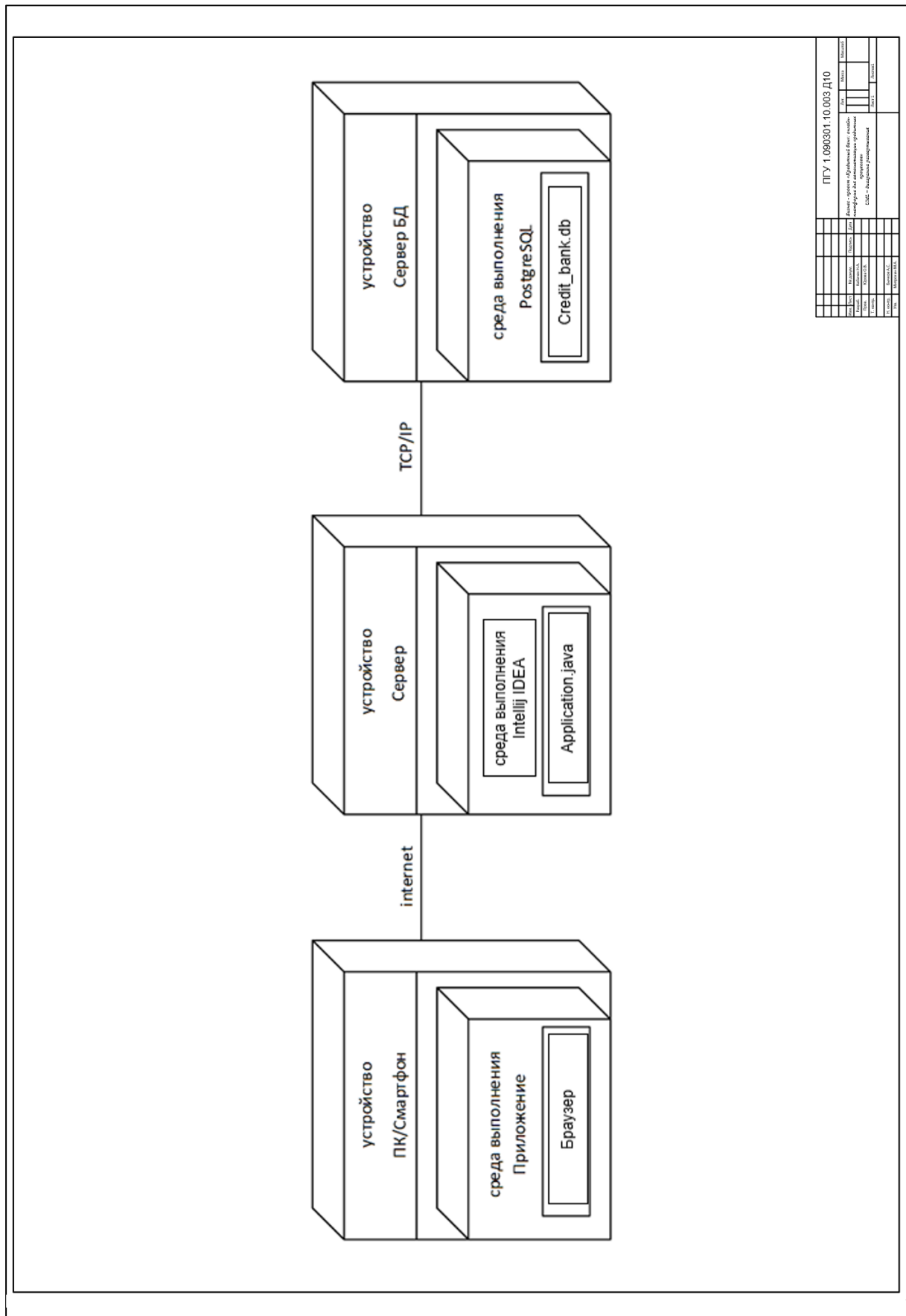
Приложение А.1 – UML – диаграмма деятельности





ПГУ 1 090301.10.003 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

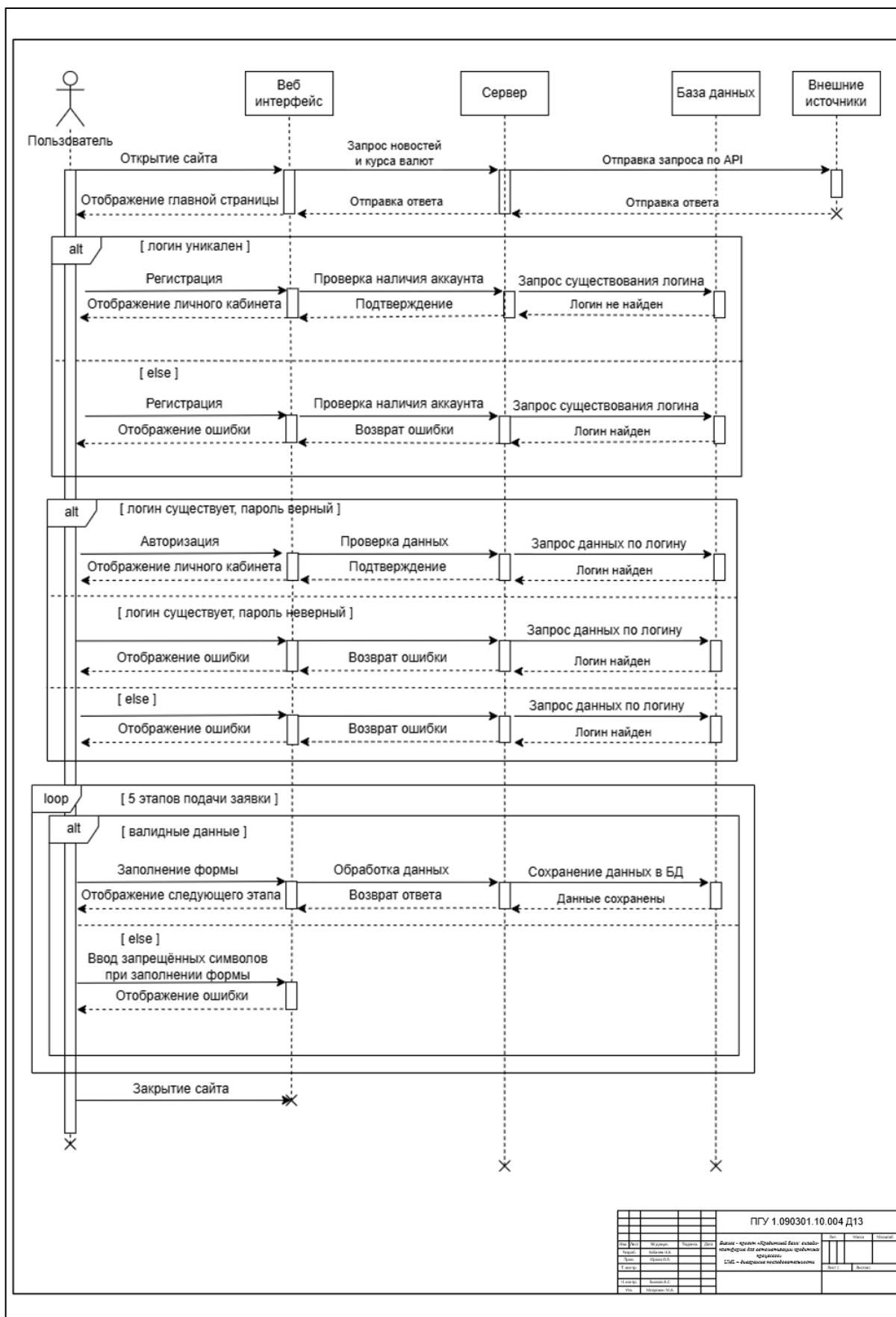
Приложение А.3 – UML – диаграмма развертывания



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Приложение А.4 – UML – диаграмма последовательности



ПГУ 1.090301.10.004 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Приложение Б

(обязательное)

Бизнес - проект «Кредитный банк: онлайн-платформа для автоматизации
кредитных процессов»

Листинг программы

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		53

Файл «AuthController.java»

```
@RestController @RequiredArgsConstructor
@RequestMapping("/gateway/auth")
public class AuthController {
    private final AuthService authService;
    @PostMapping("/registry")
    public ResponseEntity<?> registry(@RequestBody @Validated
LoginRequest request, HttpServletResponse response) {
        return authService.registry(request, response);
    }
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest
request, HttpServletResponse response) {
        return authService.login(request, response);
    }
    @PostMapping("/logout")
    public ResponseEntity<?> logout(HttpServletResponse response,
HttpServletRequest request) {
        return authService.logout(response, request);
    }
}
```

Файл «GatewayController.java»

```
@RestController @RequestMapping("/gateway")
@RequiredArgsConstructor @Slf4j
public class GatewayController {
    private final GatewayService gatewayService;
    private final AuthService authService;
    private final NewsService newsService;
    @PostMapping("/statement")
    public List<LoanOfferDto> initialRegistration (@RequestBody
@Validated LoanStatementRequestDto request, HttpServletRequest
httpRequest) {
        String username = authService.getUsername(httpRequest);
        return gatewayService.initialRegistration(request,
username);
    }
    @PostMapping("/statement/offer")
    public void selectOffer(@RequestBody @Validated LoanOfferDto
request) { gatewayService.selectOffer(request); }
    @PostMapping("/calculate/{statementId}")
    public void finishRegistration(@PathVariable String
statementId, @RequestBody FinishRegistrationRequestDto request)
throws FeignValidationException {
        gatewayService.finishRegistration(statementId, request);
    }
    @PostMapping("/document/{statementId}/send")
    public void createDocuments(@PathVariable String statementId) {
        gatewayService.sendDocuments(statementId);
    }
}
```

```

        @PostMapping("/document/{statementId}/sign")
        public void signRequestDocuments(@PathVariable String statementId) {
            gatewayService.signRequestDocuments(statementId);
        }
        @PostMapping("/document/{statementId}/code")
        public void signDocuments(@PathVariable String statementId,
        @RequestParam Integer sesCode) throws ValidationException {
            gatewayService.signDocuments(statementId, sesCode);
        }
        @GetMapping("/admin/statement/{statementId}")
        public StatementEntityDto getStatement(@PathVariable String statementId) {
            return gatewayService.getStatementById(statementId);
        }
        @PostMapping("/email")
        public void emailSubscribe (@RequestBody String email) {
            log.info("Получен запрос подписки на рассылку, но не
        обработан");
        }
        @GetMapping("/news")
        public NewsData getNews() {
            return newsService.getNews();
        }
    }

```

Файл «AuthService.java»

```

@Service
@RefreshScope
@RequiredArgsConstructor
public class AuthService {
    private final PasswordService passwordService;
    private final DealFeignClient dealFeignClient;
    private final List<UUID> tokens;
    private final String authCookieName = "authTokenBank";
    private final String usernameCookieName = "usernameTokenBank";

    public ResponseEntity<?> registry (LoginRequest request,
    HttpServletResponse response)
    {request.setPassword(passwordService.encryptPassword(request.getPa
    ssword()));
        boolean success = dealFeignClient.registry(request);
        if (success) {
            response.addCookie(getAuthCookie());
            response.addCookie(getUsernameCookie(request.getUsername()));
        }
        return (success ? ResponseEntity.ok().build() :
        ResponseEntity.badRequest().build());
    }
    public ResponseEntity<?> login(LoginRequest request,
    HttpServletResponse response) {

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

55

```

        try {
            if (isValidCredentials(request)) {
                response.addCookie(getAuthCookie());
                response.addCookie(getUsernameCookie(request.getUsername()));
                Map<UUID, Integer> loginResponse =
                dealFeignClient.getStatementMap(request.getUsername());
                return ResponseEntity.ok().body(loginResponse); }
            return ResponseEntity.status(401).build(); } catch
            (Exception e) { return ResponseEntity.status(500)
                .body(Map.of("message", e.getMessage())); } }

        public ResponseEntity<?> logout(HttpServletResponse response,
        HttpServletRequest request) {
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if (authCookieName.equals(cookie.getName())) {
                        tokens.remove(UUID.fromString(cookie.getValue()));
                    } } }
            response.addCookie(deleteCookie(authCookieName));
            response.addCookie(deleteCookie(usernameCookieName));
            return ResponseEntity.ok().build(); }

        public ResponseEntity<?> validateToken(HttpServletRequest
        request) {
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if (authCookieName.equals(cookie.getName()) &&
                        validateToken(cookie.getValue())) {
                        return ResponseEntity.ok().build(); } } }
            return ResponseEntity.status(401).build(); }

        public String getUsername(HttpServletRequest request) {
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if (usernameCookieName.equals(cookie.getName())) {
                        return cookie.getValue(); } } }
            return null; }

        private Cookie getAuthCookie() {
            Cookie authCookie = new Cookie(authCookieName,
            generateToken());
            authCookie.setHttpOnly(true);
            authCookie.setPath("/");
            authCookie.setMaxAge(24 * 60 * 60); // 1 день
            authCookie.setSecure(false);
            return authCookie; }

        private Cookie getUsernameCookie(String username) {
            Cookie authCookie = new Cookie(usernameCookieName,
            username);
            authCookie.setHttpOnly(true);
            authCookie.setPath("/");
            authCookie.setMaxAge(24 * 60 * 60);

```

```

        authCookie.setSecure(false);
        return authCookie;
    }
    private Cookie deleteCookie(String cookieName) {
        Cookie cookie = new Cookie(cookieName, null);
        cookie.setMaxAge(0);
        cookie.setHttpOnly(true);
        cookie.setPath("/");
        return cookie; }
    private boolean isValidCredentials(LoginRequest request) {
        try {
            LoginRequest data =
dealFeignClient.getData(request.getUsername());
            return (
request.getUsername().equals(data.getUsername()) &&
passwordService.matches(request.getPassword(), data.getPassword())
                ); } catch (ValidationException e) {
                return false; } }
    private String generateToken() {
        String token = UUID.randomUUID().toString();
        tokens.add(UUID.fromString(token));
        return token;
    }
    private boolean validateToken(String token) {
        if (token != null && !token.isEmpty()) {
            return tokens.contains(UUID.fromString(token));
        }
        return false;
    }
}

```

Файл «PasswordService.java»

```

@Service
public class PasswordService {
    private final PasswordEncoder passwordEncoder;
    public PasswordService()
        this.passwordEncoder = new BCryptPasswordEncoder();
    public String encryptPassword(String rawPassword)
        return passwordEncoder.encode(rawPassword);
    public boolean matches(String rawPassword, String
encryptedPassword)
        return passwordEncoder.matches(rawPassword,
encryptedPassword);
}

```

Файл «NewsService.java»

```

@Service @RequiredArgsConstructor
public class NewsService {
    private final RestTemplate restTemplate;

```

					ПГУ1.090301.09.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @Value("${newsapi.url}")
        private String apiUrl;
        @Cacheable(value = "worldFinanceNews", unless =
"#result.articles.empty")
        public NewsData getNews() {
            NewsApiResponse response =
restTemplate.getForObject(apiUrl, NewsApiResponse.class);
            if (response == null || response.getArticles() == null)
                return new NewsData(Collections.emptyList());
            return new NewsData(
                response.getArticles().stream()
                    .map(article -> new NewsItem(
                        article.getTitle(),
                        article.getDescription(),
                        article.getUrl(),
                        article.getUrlToImage() != null ? article.getUrlToImage() : ""))
                    .filter(article ->
!article.getUrl().startsWith("https://habr.com/"))
                    .collect(Collectors.toList())
            );
        }
    }
}

```

Файл «GatewayService.java»

```

@Service @RequiredArgsConstructor
public class GatewayService {
    private final DealFeignClient dealFeignClient;
    private final StatementFeignClient statementFeignClient;
    public List<LoanOfferDto>
initialRegistration(LoanStatementRequestDto request, String
username)
        return statementFeignClient.initialRegistration(request,
username);
    public void selectOffer(LoanOfferDto request)
        statementFeignClient.selectOffer(request);
    public void finishRegistration(String statementId,
FinishRegistrationRequestDto request) throws
FeignValidationException
        dealFeignClient.finishRegistration(statementId, request);
    public void sendDocuments(String statementId)
        dealFeignClient.createDocuments(statementId);
    public void signRequestDocuments(String statementId)
        dealFeignClient.signRequestDocuments(statementId);
    public void signDocuments(String statementId, Integer sesCode)
        dealFeignClient.signDocuments(statementId, sesCode);
    public StatementEntityDto getStatementById(String statementId)
        return dealFeignClient.getStatement(statementId);
}

```

Файл «DealFeignClient.java»

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58


```

        @NotNull
        @Pattern(regexp = "[а-яА-ЯёЁа-zA-Z]{2,30}$", message =
"Фамилия - от 2 до 30 латинских букв")
        private String lastName;
        @Pattern(regexp = "[а-яА-ЯёЁа-zA-Z]{2,30}$", message =
"Отчество, при наличии - от 2 до 30 латинских букв")
        private String middleName;
        @NotNull @Email
        private String email;
        @Adult
        private LocalDate birthdate;
        @NotNull
        @Pattern(regexp = "\\d{4}$", message = "Серия паспорта - 4
цифры")
        private String passportSeries;
        @NotNull
        @Pattern(regexp = "\\d{6}$", message = "Номер паспорта - 6
цифр")
        private String passportNumber;
    }

```

Файл «Adult.java»

```

@Constraint(validatedBy = AdultValidator.class)
@Target({ElementType.FIELD, ElementType.METHOD,
ElementType.ANNOTATION_TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface Adult {
    int over() default 18;
    int under() default 65;
    String message() default "Возраст должен быть более {over} и
менее {under} лет";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

```

Файл «AdultValidator.java»

```

public class AdultValidator implements ConstraintValidator<Adult,
LocalDate> {
    private int over;
    private int under;

    @Override
    public void initialize(Adult constraintAnnotation) {
        over = constraintAnnotation.over();
        under = constraintAnnotation.under();
    }

    @Override
    public boolean isValid(LocalDate birthdate,
ConstraintValidatorContext context) {

```

```

        if (birthdate == null)
            return false;
        int age = Period.between(birthdate,
LocalDate.now()).getYears();
        return age >= over && age <= under;
    }
}

```

Файл «LoanOfferDto.java»

```

@Data @Builder @AllArgsConstructor @NoArgsConstructor
public class LoanOfferDto implements Comparable<LoanOfferDto> {
    @NotNull
    private UUID statementId;
    @NotNull
    @Min(value = 20000, message = "Сумма кредита - действительно
число, большее или равное 20000")
    private BigDecimal requestedAmount;
    @NotNull
    private BigDecimal totalAmount;
    @NotNull
    @Min(value = 6, message = "Срок кредита - целое число, большее
или равное 6")
    private Integer term;
    @NotNull
    private BigDecimal monthlyPayment;
    @NotNull
    private BigDecimal rate;
    @NotNull
    private Boolean isInsuranceEnabled;
    @NotNull
    private Boolean isSalaryClient;
    @Override
    public int compareTo(LoanOfferDto other)
        return this.rate.compareTo(other.rate);
}

```

Файл «FinishRegistrationRequestDto.java»

```

@Data @Builder @AllArgsConstructor @NoArgsConstructor
public class FinishRegistrationRequestDto {
    @NotNull
    private ScoringDataDto.Gender gender;
    @NotNull
    private ScoringDataDto.MaritalStatus maritalStatus;
    @NotNull
    private Integer dependentAmount;
    @NotNull @DateBeforeToday
    private LocalDate passportIssueDate;
    @NotNull
    private String passportIssueBranch;
    @NotNull
    private EmploymentDto employment;
}

```

```

        @NotNull
        private String accountNumber;
    }

```

Файл «EmploymentDto.java»

```

@Data @Builder @AllArgsConstructor @NoArgsConstructor
public class EmploymentDto {
    public enum EmploymentStatus {
        EMPLOYED, SELF_EMPLOYED, BUSINESS_OWNER, UNEMPLOYED
    }
    public enum Position {
        MANAGER, MID_MANAGER, TOP_MANAGER, WORKER
    }
    @NotNull
    @Employer
    private EmploymentStatus employmentStatus;
    @NotNull
    @Pattern(regexp = "^\\d{12}$", message = "ИНН - 12 цифр")
    private String employerINN;
    @NotNull
    private BigDecimal salary;
    @NotNull
    private Position position;
    @NotNull
    @Min(value = 18, message = "Общий стаж менее 18 месяцев")
    private Integer workExperienceTotal;
    @NotNull
    @Min(value = 3, message = "Текущий стаж менее 3 месяцев")
    private Integer workExperienceCurrent;
}

```

Файл «DateBeforeToday.java»

```

@Constraint(validatedBy = DateBeforeTodayValidator.class)
@Target({ElementType.FIELD, ElementType.METHOD,
        ElementType.ANNOTATION_TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface DateBeforeToday {
    String message() default "Дата должна быть до сегодняшнего дня";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

```

Файл «DateBeforeToday.java»

```

public class DateBeforeTodayValidator implements
ConstraintValidator<DateBeforeToday, LocalDate> {
    @Override
    public boolean isValid(LocalDate birthDate,
        ConstraintValidatorContext context) {
        if (birthDate == null) return false;
    }
}

```

```

        return birthDate.isBefore(LocalDate.now());
    }
}

```

Файл «LoginRequest.java»

```

@Data @Builder @AllArgsConstructor @NoArgsConstructor
public class LoginRequest {
    @Pattern(regexp = "^(?=.*[a-zA-Z])[a-zA-Z0-9._]{4,30}+$",
            message = "Логин должен содержать буквы (обязательно),
цифры, точки или подчёркивания, от 4 до 30 символов")
    private String username;
    @Pattern(regexp = "^(?=.*[a-zA-Z])(?=.*\\d)[a-zA-Z\\d\\W_]{6,30}+$",
            message = "Пароль должен содержать хотя бы одну букву
и одну цифру (допускаются специальные символы), от 6 до 30
символов")
    private String password;
}

```

Файл «changelog.xml»

```

<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-
latest.xsd">
    <include file="changeset-init-database.xml"
relativeToChangelogFile="true"/>
</databaseChangeLog>

```

Файл «changeset-init-database.xml»

```

<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-
latest.xsd">
    <changeSet id="import_uuid_generator" author="robinx51">
        <sql>CREATE EXTENSION IF NOT EXISTS "uuid-oss";</sql>
    </changeSet>
    <changeSet id="create_table_client" author="robinx51">
        <createTable tableName="client">
            <column name="client_id" type="uuid"
defaultValueComputed="uuid_generate_v4()">
                <constraints primaryKey="true" primaryKeyName="pk_client"
nullable="false"/></column>

```

```

<column name="last_name" type="varchar(255)"/>
<column name="first_name" type="varchar(255)"/>
<column name="middle_name" type="varchar(255)"/>
<column name="birth_date" type="date"/>
<column name="email" type="varchar(255)"/>
<column name="gender" type="varchar(255)"/>
<column name="marital_status" type="varchar(255)"/>
<column name="dependent_amount" type="integer"/>
<column name="passport" type="jsonb"/>
<column name="employment" type="jsonb"/>
<column name="account_number" type="varchar(255)"/>
<column name="account" type="varchar(255)"/>
</createTable>
</changeSet>
<changeSet id="create_table_credit" author="robinx51">
  <createTable tableName="credit">
    <column name="credit_id" type="uuid"
defaultValueComputed="uuid_generate_v4()">
      <constraints primaryKey="true" primaryKeyName="pk_credit"
nullable="false"/>
    </column>
    <column name="amount" type="numeric(38, 2)"/>
    <column name="term" type="integer"/>
    <column name="monthly_payment" type="numeric(38, 2)"/>
    <column name="rate" type="numeric(38, 2)"/>
    <column name="psk" type="numeric(38, 2)"/>
    <column name="payment_schedule" type="jsonb"/>
    <column name="insurance_enabled" type="boolean"/>
    <column name="salary_client" type="boolean"/>
    <column name="credit_status" type="character varying(255)"/>
  </createTable>
</changeSet>
<changeSet id="create_table_statement" author="robinx51">
  <createTable tableName="statement">
    <column name="statement_id" type="uuid"
defaultValueComputed="uuid_generate_v4()">
      <constraints primaryKey="true"
primaryKeyName="pk_statement" nullable="false"/>
    </column><column name="client_id" type="uuid">
      <constraints foreignKeyName="fk_client_id"
references="client(client_id)"/></column>
    <column name="credit_id" type="uuid"> <constraints
foreignKeyName="fk_credit_id" references="credit(credit_id)"/>
    </column>
    <column name="status" type="character varying(255)"/>
    <column name="creation_date" type="timestamp without time
zone"/>
    <column name="applied_offer" type="jsonb"/>
    <column name="sign_date" type="timestamp without time zone"/>
    <column name="ses_code" type="integer"/>
  </createTable>
</changeSet>

```

```

        <column name="status_history" type="jsonb"/>
    </createTable>
    <createTable tableName="account">
        <column name="login" type="varchar(255)">
            constraints primaryKey="true" primaryKeyName="pk_account"
nullable="false"/> column>
        <column name="password" type="varchar(255)"/>
    </createTable>
</changeSet>
</databaseChangeLog>

```

Файл «CorsConfig.java»

```

@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Value("${frontend.url}")
    private String frontendUrl;
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/*")
            .allowedOrigins(frontendUrl)
            .allowedMethods("*");
    }
}

```

Файл «SecurityConfig.java»

```

@Configuration @EnableWebSecurity @Slf4j
public class SecurityConfig {
    @Value("${frontend.url}")
    private String frontendUrl;
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
        http
            .cors(cors ->
cors.configurationSource(corsConfigurationSource()))
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
            .requestMatchers("/gateway/**").permitAll()
            .requestMatchers("/gateway/auth/**").permitAll()
            .anyRequest().authenticated());
        return http.build();
    }
    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(List.of(
            "http://frontend:3000",
            "http://frontend-image:3000",
            frontendUrl
        ));
        config.setAllowedMethods(List.of("*"));
    }
}

```

```

        config.setAllowedHeaders(List.of("*"));
        config.setExposedHeaders(List.of("Authorization", "Set-
Cookie"));
        config.setAllowCredentials(true);
        config.setMaxAge(3600L);
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/*", config);
        return source;
    }
}

```

Файл «ScoringService.java»

```

@Setter @Service
public class ScoringService {
    private static final Logger logger =
LoggerFactory.getLogger(ScoringService.class);
    @Value("${calculator.baseRate}")
    private int baseRate;
    private static final double ratioInsurance = 1.05;
    public BigDecimal evaluateTotalAmountByServices(BigDecimal amount,
boolean isInsuranceEnabled) {
        if (isInsuranceEnabled)
            return amount.multiply(new
BigDecimal(ratioInsurance)).setScale(2, RoundingMode.HALF_UP);
        else
            return amount;
    }
    public BigDecimal calculateRate(boolean isInsuranceEnabled,
boolean isSalaryClient) {
        int rate = baseRate;
        if (isInsuranceEnabled)
            rate = rate - 3;
        if (isSalaryClient)
            rate = rate - 1;
        return new BigDecimal(rate);
    }
    public BigDecimal getMonthlyPayment(final BigDecimal totalAmount,
final BigDecimal rate, final int term) {
        double monthlyRate = (rate.doubleValue() / 100) / 12;
        //  $K = (M * (1 + M)^S) / ((1 + M)^S - 1)$ 
        // где М – месячная процентная ставка по кредиту, S – срок
кредита в месяцах.
        double ratioPayment = (monthlyRate * Math.pow((1 +
monthlyRate), term)) / (Math.pow((1 + monthlyRate), term) - 1);
        logger.debug("Коэффициент аннуитета: {}", ratioPayment);
        //  $X = C * K$ 
        // где X – аннуитетный платеж, C – сумма кредита, K –
коэффициент аннуитета.
        return new BigDecimal(totalAmount.doubleValue() *
ratioPayment).setScale(2, RoundingMode.HALF_UP);
    }
}

```

```

}
public BigDecimal calculateFinalRate(ScoringDataDto request) {
    BigDecimal rate = new BigDecimal(baseRate);
    switch (request.getEmployment().getEmploymentStatus()) {
        case SELF_EMPLOYED -> rate =
rate.add(BigDecimal.valueOf(2));
        case BUSINESS_OWNER -> rate =
rate.add(BigDecimal.valueOf(1));
    } switch (request.getEmployment().getPosition()) {
        case MID_MANAGER -> rate =
rate.subtract(BigDecimal.valueOf(2));
        case TOP_MANAGER -> rate =
rate.subtract(BigDecimal.valueOf(3));
    } switch (request.getMaritalStatus()) {
        case MARRIED -> rate =
rate.subtract(BigDecimal.valueOf(3));
        case DIVORCED -> rate =
rate.add(BigDecimal.valueOf(1));
    } switch (request.getGender()) {
        case MALE -> {
            if (getYears(request.getBirthdate()) >= 30 &&
getYears(request.getBirthdate()) <= 50)
                rate =
rate.subtract(BigDecimal.valueOf(3));
        } case FEMALE -> {
            if (getYears(request.getBirthdate()) >= 32 &&
getYears(request.getBirthdate()) <= 60)
                rate =
rate.subtract(BigDecimal.valueOf(3));
        }
    }
    return rate;
}

public List<PaymentScheduleElementDto> calculatePaymentSchedule
(BigDecimal totalAmount, int term, BigDecimal rate, BigDecimal
monthlyPayment) {
    BigDecimal interestPayment; // Выплата процентов
    BigDecimal debtPayment; // Выплата долга
    BigDecimal remainingDebt = totalAmount; // Оставшийся долг
    double monthlyRate = (rate.doubleValue() / 100) / 12;
    List<PaymentScheduleElementDto> list = new ArrayList<>();
    for (int i = 1; i <= term; i++) {
        interestPayment =
remainingDebt.multiply(BigDecimal.valueOf(monthlyRate)).setScale(2
, RoundingMode.HALF_UP);
        debtPayment =
monthlyPayment.subtract(interestPayment).setScale(2,
RoundingMode.HALF_UP);
        remainingDebt =
remainingDebt.subtract(debtPayment).setScale(2,
RoundingMode.HALF_UP);
    }
}

```

```

        list.add(PaymentScheduleElementDto.builder()
            .number(i)
            .date(LocalDate.now().plusMonths(i))
            .totalPayment(monthlyPayment)
            .interestPayment(interestPayment)
            .debtPayment(debtPayment)
            .remainingDebt(remainingDebt).build());
    }
    list.get(list.size() - 1).setRemainingDebt(BigDecimal.ZERO);
    return list;
}

public BigDecimal calculatePsk(BigDecimal monthlyPayment, Integer
term)
    return monthlyPayment.multiply(BigDecimal.valueOf(term));
private int getYears(LocalDate birthdate) {
    return Period.between(birthdate,
        LocalDate.now()).getYears();
}
}

```

Файл «Client.java»

```

@Entity @Data @Builder @NoArgsConstructor @AllArgsConstructor
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name = "client")
public class Client implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "client_id", columnDefinition = "uuid",
        updatable = false, nullable = false)
    private UUID clientId;
    @Column(name = "first_name", nullable = false)
    private String firstName;
    @Column(name = "last_name", nullable = false)
    private String lastName;
    @Column(name = "middle_name")
    private String middleName;
    @Column(name = "birth_date")
    private Date birthDate;
    @Column(name = "email")
    private String email;
    @Column(name = "gender")
    @Enumerated(EnumType.STRING)
    private ScoringDataDto.Gender gender;
    @Column(name = "marital_status")
    @Enumerated(EnumType.STRING)
    private ScoringDataDto.MaritalStatus maritalStatus;
    @Column(name = "dependent_amount")
    private Integer dependentAmount;
    @Column(name = "passport")
    @JdbcTypeCode(SqlTypes.JSON)
    private PassportDto passportDto;
}

```

```

@Column(name = "employment")
@JdbcTypeCode(SqlTypes.JSON)
private EmploymentDto employment;
@Column(name = "account_number")
private String accountNumber;
@Column(name = "account")
private String account;
}

```

Файл «Credit.java»

```

@Entity @Data @Builder @NoArgsConstructor @AllArgsConstructor
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name = "credit")
public class Credit {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID creditId;
    @Column(name = "amount")
    private BigDecimal amount;
    @Column(name = "term")
    private Integer term;
    @Column(name = "monthly_payment")
    private BigDecimal monthlyPayment;
    @Column(name = "rate")
    private BigDecimal rate;
    @Column(name = "psk")
    private BigDecimal psk;
    @JdbcTypeCode(SqlTypes.JSON)
    @Column(name = "payment_schedule")
    private List<PaymentScheduleElementDto> paymentSchedule;
    @Column(name = "insurance_enabled")
    private Boolean insuranceEnabled;
    @Column(name = "salary_client")
    private Boolean salaryClient;
    @Column(name = "credit_status")
    @Enumerated(EnumType.STRING)
    private CreditStatus creditStatus;
}

```

Файл «Statement.java»

```

@Entity @Data @Builder @NoArgsConstructor @AllArgsConstructor
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name = "statement")
public class Statement implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "statement_id", columnDefinition = "uuid",
    updatable = false, nullable = false)
    private UUID statementId;
    @OneToOne(fetch = FetchType.LAZY, optional = false)

```

```

@JoinColumn(name = "client_id", nullable = false)
private Client client;
@OneToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "credit_id", nullable = false)
private Credit credit;
@Column(name = "status")
@Enumerated(EnumType.STRING)
private ApplicationStatus status;
@Column(name = "creation_date")
private Timestamp creationDate;
@Column(name = "applied_offer")
@JdbcTypeCode(SqlTypes.JSON)
private LoanOfferDto appliedOffer;
@Column(name = "sign_date")
private Timestamp signDate;
@Column(name = "ses_code")
private Integer sesCode;
@Column(name = "status_history")
@JdbcTypeCode(SqlTypes.JSON)
private List<StatementStatusHistoryDto> statusHistory;
}

```

Файл «Statement.java»

```

@Entity @Data @Builder @NoArgsConstructor @AllArgsConstructor
@Table(name = "account")
public class Account implements Serializable {
    @Id
    @Column(name = "login", nullable = false)
    private String login;
    @Column(name = "password", nullable = false)
    private String password;
}

```

Файл «ClientRepository.java»

```

@Repository
public interface ClientRepository extends JpaRepository<Client,
UUID> {}

```

Файл «CreditRepository .java»

```

@Repository
public interface CreditRepository extends JpaRepository<Credit,
UUID> {}

```

Файл «StatementRepository.java»

```

@Repository
public interface StatementRepository extends
JpaRepository<Statement, UUID> {
    @Query("SELECT s.statementId as statementId, s.status as

```

```

status FROM Statement s " +
        "RIGHT JOIN Client c ON s.client.clientId = c.clientId
" +
        "WHERE c.account = :username")
    List<Tuple> findAllStatementMap(@Param("username") String
username);
}

```

Файл «AccountRepository.java»

```

@Repository
public interface AccountRepository extends JpaRepository<Account,
String> {}

```

Файл «ClientServiceDB.java»

```

@Service
public class ClientServiceDB {
    @Autowired
    private ClientRepository clientRepository;
    private static final Logger logger =
    LoggerFactory.getLogger(ClientServiceDB.class);
    public Client createClient(LoanStatementRequestDto request, String
username) {
        logger.info("Добавление client в БД");
        Client client = Client.builder()
            .lastName(request.getLastName())
            .firstName(request.getFirstName())
            .middleName(request.getMiddleName())
            .birthDate(Date.valueOf(request.getBirthdate()))
            .email(request.getEmail())
            .passportDto(
                PassportDto.builder().passportId(UUID.randomUUID()).passportSeries
                (request.getPassportSeries()).passportNumber(request.getPassportNu
                mber()).build()
            )
            .account(username)
            .build();
        return clientRepository.save(client);
    }
    public void updateClient(Client client) {
        logger.info("Обновление client с id: {}",
client.getClientId());
        if (clientRepository.existsById(client.getClientId())) {
            clientRepository.save(client);
            logger.info("Client обновлён успешно");
        } else {

```

```

        logger.error("Client с id: {} не найден",
client.getClientId());
        throw new EntityNotFoundException("Client с id: " +
client.getClientId() + " не найден");
    }
}
public Client getClientById(UUID clientId)
    return clientRepository.findById(clientId)
        .orElseThrow(() -> new EntityNotFoundException("Client
с id: " + clientId + " не найден"));
}

```

Файл «CreditServiceDB.java»

```

@Service
public class CreditServiceDB {
    @Autowired
    private CreditRepository creditRepository;
    private static final Logger logger =
LoggerFactory.getLogger(CreditServiceDB.class);
    public Credit createCredit(boolean insuranceEnabled, boolean
salaryClient) {
        logger.info("Добавление credit в БД");
        Credit credit = new Credit();
        credit.setInsuranceEnabled(insuranceEnabled);
        credit.setSalaryClient(salaryClient);
        credit.setCreditStatus(CreditStatus.PREPARED);
        return creditRepository.save(credit);
    }
    public Credit getCreditById(UUID creditId)
        return creditRepository.findById(creditId)
            .orElseThrow(() -> new
EntityNotFoundException("Credit с id: " + creditId + " не
найден"));
    public void updateCredit(Credit credit) {
        logger.info("Обновление credit с id: {}",
credit.getCreditId());
        if (creditRepository.existsById(credit.getCreditId())) {
            creditRepository.save(credit);
            logger.info("Credit обновлён успешно");
        } else {
            logger.error("Credit с id: {} не найден",
credit.getCreditId());
            throw new EntityNotFoundException("Credit с id: " +
credit.getCreditId() + " не найден");
        }
    }
}

```

Файл «StatementServiceDB.java»

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

```

@Service @RequiredArgsConstructor
public class StatementServiceDB {
    @Autowired
    public StatementRepository statementRepository;
    private static final Logger logger =
LoggerFactory.getLogger(StatementServiceDB.class);
    public Statement createStatement(Client client) {
        logger.info("Добавление statement в БД");
        Statement statement = new Statement();
        statement.setClient(client);
        return statementRepository.save(statement);
    }
    public void updateStatement(Statement statement) {
        logger.info("Обновление statement с id: {}",
statement.getId());
        if
(statementRepository.existsById(statement.getId())) {
            statementRepository.save(statement);
            logger.info("Statement обновлён успешно");
        } else {
            logger.error("Statement с id: {} не найден",
statement.getId());
            throw new EntityNotFoundException("Statement с id: " +
statement.getId() + " не найден");
        }
    }
    public Statement getStatementById(UUID statementId)
        return statementRepository.findById(statementId)
            .orElseThrow(() -> new
EntityNotFoundException("Statement с id: " + statementId + " не
найден"));
    public List<Statement> getStatements()
        return statementRepository.findAll();
    public List<Tuple> getStatementMap(String username)
        return statementRepository.findAllStatementMap(username);
}

```

Файл «AccountServiceDB.java»

```

@Service @RequiredArgsConstructor
public class AccountServiceDB {
    private final AccountRepository accountRepository;
    private static final Logger logger =
LoggerFactory.getLogger(ClientServiceDB.class);
    public boolean createAccount(LoginRequest request) {
        logger.info("Добавление account в БД");
        String username = request.getUsername();
        String password = request.getPassword();
        if (accountRepository.existsById(username))
            return false;

        Account account = new Account(username, password);
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

73

```

        accountRepository.save(account);
        return true;
    }
    public LoginRequest getAccount(String username) {
        logger.info("Получение account из БД");
        try {
            Account response =
accountRepository.getReferenceById(username);
            return new LoginRequest(response.getLogin(),
response.getPassword());
        } catch (Exception e)
            throw new ValidationException("usernameNotFound");
    }
}

```

Файл «HomePage.tsx»

```

const HomePage: FC = () => {
    const [data, setData] = useState<NewsProps[]>([]);
    const intervalId = useRef<NodeJS.Timeout>();
    const getData = async () => {
        try {
            const response = await getFinanceNews();
            if (response?.articles?.length)
                setData(response.articles);
        } catch (error)
            console.error("Error fetching news:", error);
    };
    useEffect(() => {
        getData();
        intervalId.current = setInterval(
            () => { getData(); }, 15 * 60 * 1000, );
        return () => {
            clearInterval(intervalId.current);
        };
    }, []);
    return (
        <main className={styles.homePage}>
            <CardDesign />
            <UsedFunctions />
            <ExchangeRate />
            <section className={styles.ourServices}>
                <h3>Вы можете воспользоваться нашими услугами в любой
точке мира</h3>
                <p>Выводите и переводите деньги онлайн через наше
приложение</p>
                <img src={Map} alt="map" width="1060" height="538" />
            </section>
            <News articles={data} />
            <SubscribeNewsletter />
        </main>
    );
}

```

```
};
export default HomePage;
```

Файл «CardDesign.tsx»

```
const CardDesign: FC = () => {
  return (
    <section className={styles.cardDesign}>
      <div className={styles.cardDesign__choose}>
        <h1>Выберите понравившийся вам дизайн и подайте заявку на
получение карты прямо сейчас</h1>
        <DefaultButton
          buttonType={ButtonType.link}
          className={styles.radius}
          link={CREDIT_CARD_ROUTE}
        >
          Выберите карту
        </DefaultButton>
      </div>
      <div className={styles.cardDesign__design}>
        {card.map(({ img, alt }, index) => (
          <img data-testid={alt + index} src={img} alt={alt}
key={index} />
        ))}
      </div>
    </section>
  );
};
export default CardDesign;
```

Файл «UsedFunctions.tsx»

```
const UsedFunctions: FC = () => {
  return (
    <section className={styles.usedFunctions}>
      <img src={Man} alt="man" />
      <div className={styles.usedFunctions__info}>
        <h2>Мы предоставляем множество функций, которые вы можете
использовать</h2>
        <p>
          Вы можете с удовольствием ознакомиться с функциями,
которые мы предоставляем, и получить свои
собственные функции каждой функции
        </p>
        <div className={styles.info__list}>
          {features.map((title, index) => (
            <div
              data-testid={`_${title}`}
              className={styles.list__items}
              key={index} >
              <img src={Check} alt="check" />
              <p>{title}</p>
            </div>
          ))}
        </div>
      </div>
    </section>
  );
};
```

```

        </div>
      )})
    </div>
  </div>
</section>
);
};
export default UsedFunctions;

```

Файл «ExchangeRate.tsx»

```

const ExchangeRate: FC = () => {
  const [data, setData] = useState<Record<string, number> |
undefined>();
  const intervalId = useRef<NodeJS.Timeout>();
  const getData = async () => {
    const response = await getCurrency();
    setData(response);
  };
  useEffect(() => {
    getData();
    intervalId.current = setInterval(
      () => {
        getData();
      },
      15 * 60 * 1000,
    );
    return () => {
      clearInterval(intervalId.current);
    };
  }, []);
  const getDate = () => {
    const date = new Date();
    const currentYear = date.getFullYear();
    const currentMonth = date.getMonth() + 1;
    const currentDay = date.getDate();
    return [currentYear, currentMonth,
currentDay].reverse().join(".");
  };
  return (
    <section className={styles.exchangeRate}>
      <div className={styles.exchangeRate__currency}>
        <h2>Курс обмена валют в интернет-банке</h2>
        <p className={styles.exchangeRate__text}>Валюта</p>
        <div className={styles.currency__block}>
          {data ? (
            currency.map((item, index) => (
              <div key={index + item}
className={styles.block__item}>
                <p className={styles.item__name}>{item}:</p>
                <p className={styles.item__rate}>
                  {(1 / data[item]).toFixed(2)}

```

```

        </p>
      </div>
    ))
  ) : (
    <p className={styles.exchangeRate__text}>
      Не удалось получить данные
    </p>
  )}
</div>
</div>
<div className={styles.exchangeRate__update}>
  <p>`Обновляется каждые 15 минут, МСК ${getDate()}`</p>
  <img src={Bank} alt="bank" />
</div>
</section>
);
};
export default ExchangeRate;

```

Файл «News.tsx»

```

const News: FC<NewsData> = ({ articles }) => {
  const [activeIndex, setActiveIndex] = useState(0);
  const [isSmallScreen, setIsSmallScreen] =
    useState(window.innerWidth <= 600);
  const ref = useRef<number | null>(null);
  const lastIndex = isSmallScreen ? articles.length - 1 :
    articles.length - 3;
  const startSlide = activeIndex === 0;
  const endSlide = activeIndex === lastIndex;
  const isEdge = startSlide || endSlide;
  const prevSlide = () => {
    if (activeIndex > 0) {
      setActiveIndex((prevState) => prevState - 1);
    }
  };
  const nextSlide = () =>
    setActiveIndex((prevState) => prevState + 1);
  const handleTouchStart: TouchEventHandler<HTMLDivElement> = (e)
=> {
    ref.current = e.touches[0].clientX;
  };
  const handleTouchMove: TouchEventHandler<HTMLDivElement> = (e)
=> {
    if (isEdge)
      return null;
    const touchDown = ref.current;
    if (touchDown === null)
      return;
    const currentTouch = e.touches[0].clientX;
    const diff = touchDown - currentTouch;
    if (diff > 5)

```

```

        nextSlide();
    else if (diff < -5)
        prevSlide();
    ref.current = null;
};
const resizeListener = useCallback(
    (event: UIEvent) => {
        const checkIsSmallScreen = (event.view?.innerWidth ?? 0) <=
600;
        if (checkIsSmallScreen !== isSmallScreen) {
            setIsSmallScreen(checkIsSmallScreen);
        }
    }, [isSmallScreen],
);
useEffect(() => {
    document.addEventListener("resize", resizeListener);
    return () => {
        document.removeEventListener("resize", resizeListener);
    };
}, [resizeListener]);
return (
    <section className={styles.news__section}>
        <div>
            <h2>Актуальные новости из мира финансов</h2>
            <p className={styles.section__description}>
                Мы обновляем новостную ленту каждые 15 минут. Вы можете
узнать больше,
                нажав на интересующую вас новость.
            </p>
        </div>
        <div
            onTouchStart={handleTouchStart}
            onTouchMove={handleTouchMove}
            className={styles.news__items}
        >
            {articles.length ? (
                articles.map((item, index) => (
                    <NewsCard
                        key={index + item.title}
                        {...item}
                        activeIndex={activeIndex}
                    />
                ))
            ) : (
                <p className={styles.section__description}>Пока новостей
нет</p>
            )}
        </div>
        <div className={styles.items__buttons}>
            <DefaultButton
                buttonType={ButtonType.button}

```

```

        onClick={prevSlide}
        className={` ${startSlide ? styles.disabled :
styles.buttonLeft} ${styles.radius}`}
        disabled={startSlide}
        data-testid={"prev-slide"}
      >
        <img src={ArrowLeft} alt="left" />
      </DefaultButton>
      <DefaultButton
        buttonType={ButtonType.button}
        onClick={nextSlide}
        className={` ${endSlide ? styles.disabled : ""}
${styles.radius}`}
        disabled={endSlide}
        data-testid={"next-slide"}
      >
        <img src={ArrowRight} alt="right" />
      </DefaultButton>
    </div>
  </section>
);
};
export default News;

```

Файл «NewsCard.tsx»

```

interface NewsCardProps extends NewsProps {
  activeIndex: number;
}
const NewsCard: FC<NewsCardProps> = ({
  url,
  urlToImage,
  title,
  description,
  activeIndex,
}) => {
  const getSubstring = (string: string) => {
    return string.length > 90 ? string.substring(0, 90) + "... " :
string;
  };
  const handleError: ReactEventHandler<HTMLImageElement> |
undefined = ({
    currentTarget,
  }: SyntheticEvent<HTMLImageElement>) => {
    currentTarget.onerror = null;
    currentTarget.src = NewsImg;
  };
  const slideImageStyle = `translateX(calc(-${activeIndex * 100}%
- ${80 * activeIndex}px))`;
  return (

```

```

<Link
  to={url}
  target="_blank"
  rel="noopener noreferrer"
  className={styles.newsCard}
  style={{
    transform: slideImageStyle,
  }}
  data-testid={`news${title}`}
>
  <div className={styles.newsCard__item}>
    <div className={styles.item__img}>
      <img src={urlToImage} alt="image" onError={handleError}
    />
    </div>
    <p
      className={styles.newsCard__title}>{getSubstring(title)}</p>
    <p className={styles.newsCard__description}>
      {getSubstring(description)}
    </p>
    </div>
  </Link>
);
};
export default NewsCard;

```

Файл «auth.ts»

```

const cookieOptions = {
  path: '/',
  expires: 1, // 1 день
  sameSite: 'strict' as const,
};
export const registry = async (username: string, password:
string): Promise<boolean> => {
  const response = await postRegistry(username, password);
  if (response) {
    Cookies.set(IS_AUTH_COOKIE, 'true', cookieOptions);
    Cookies.set(USERNAME_COOKIE, username, cookieOptions);
    return true;
  }
  clearCookies();
  return false;
};
export const login = async (username: string, password: string):
Promise<Record<string, number> | null> => {
  const response = await checkLogin(username, password);
  if (response !== null) {
    Cookies.set(IS_AUTH_COOKIE, 'true', cookieOptions);
    Cookies.set(USERNAME_COOKIE, username, cookieOptions);
    return response;
  }

```

```

    }
    clearCookies();
    return null;
};
export const logout = async () => {
    try
        await postLogout();
    finally
        clearCookies();
};
export const clearCookies = () => {
    Cookies.remove(IS_AUTH_COOKIE);
    Cookies.remove('authTokenSmartStore', { path: '/' });
};
export const checkAuth = (): boolean =>
    return Cookies.get(IS_AUTH_COOKIE) === 'true';
export const getUsernameCookie = (): string | undefined =>
    return Cookies.get(USERNAME_COOKIE)?.toString();

```

Файл «requests.ts»

```

// Info
const getData = async <T>(url: string): Promise<T | undefined> =>
{
    try {
        const result = await axios.get<T>(url);
        return result.data;
    } catch (error) {
        console.log(error);
    }
};
export const getCurrency = async () => {
    const data = await getData<GetCurrencyType>(
        CURRENCY_URL,
    );
    return data?.conversion_rates;
};
export const getNews = async () : Promise<NewsData> => {
    const response = await instance.get("/news");
    return response.data;
};
// Auth
export const postRegistry = async (username: string, password:
string): Promise<boolean> => {
    const response = await instance.post('/auth/registry', {
username, password }, {
        withCredentials: true,
        headers: {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Credentials' : 'true',
        }
    });
};

```

```

    return response.status === 200;
}
export const checkLogin = async (username: string, password:
string): Promise<Record<string, number> | null> => {
    try {
        const response = await instance.post('/auth/login', {
username, password }, {
            withCredentials: true,
            headers: {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Credentials' : 'true',
            }
        });
        return response.data;
    } catch {
        return null;
    }
};
export const postLogout = async (): Promise<void> => {
    try
await instance.post('auth/logout', {}, { withCredentials: true });
    catch (err)
        console.error('Error logout: ' + err);
}
// Bank
export const sendEmail = async (email: string) =>
    await instance.post("email", { email });
export const sendPrescoringForm = async (data:
LoanStatementRequestDto) => {
    return await instance.post<LoanOfferDto[]>("statement", data, {
        withCredentials: true,
        headers: {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Credentials' : 'true',
        }
    });
};
export const selectOffer = async (offer: LoanOfferDto) =>
    await instance.post("statement/offer", offer);
export const finishRegistration = async (
    id: string,
    data: FinishRegistrationBody,
) => await instance.post(`calculate/${id}`, data);
export const createDocument = async (id: string) =>
    await instance.post(`document/${id}/send`);
export const signDocument = async (id: string) =>
    await instance.post(`document/${id}/sign`);
export const sendSignCode = async (id: string, sesCode: number) =>
    await instance.post(`document/${id}/code`, null, {
        params: { sesCode },
        headers: { "Content-Type": "application/json", },
    });
};

```

```
export const getCreditDataById = async (id: string) => await
instance.get<StatementEntityDto>(`admin/statement/${id}`);
```

Файл «main.tsx»

```
createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <Router>
      <App />
    </Router>
  </StrictMode>,
);
```

Файл «App.tsx»

```
function App() {
  return (
    <Provider store={store}>
      <PersistGate loading={null} persistor={persistor}>
        <Header />
        <AppRouterProvider />
        <Footer />
      </PersistGate>
    </Provider>
  );
}
export default App;
```

Файл «index.tsx»

```
const AppRouterProvider: FC = () => {
  const location = useLocation();
  useEffect(() => {
    window.scrollTo(0, 0);
  }, [location.pathname]);
  return (
    <Routes location={location}>
      <Route path={HOME_PAGE_ROUTE} element={<HomePage />} />
      <Route path={LOGIN_ROUTE} element={<AuthPage />} />
      <Route path={ACCOUNT_ROUTE} element={<AccountPage />} />
      <Route path={CREDIT_CARD_ROUTE} element={<CreditCardPage />} />
      <Route path={LOAN_ROUTE} element={<LoanPage />} />
      <Route path={DOCUMENT_ROUTE} element={<DocumentPage />} />
      <Route path={DOCUMENT_SIGN_ROUTE}
element={<DocumentSignPage />} />
      <Route path={CONFIRMATION_CODE_ROUTE}
element={<ConfirmationCodePage />} />
      <Route path={GENERAL_ROUTE} element={<NotFoundPage />} />
    </Routes>
  );
};
```

```
};
export default AppRouterProvider;
```

Файл «Header.tsx»

```
const Header: FC = () => {
  return (
    <header className={styles.header}>
      <Link to={HOME_PAGE_ROUTE} className={styles.header__logo}>
        Банк
      </Link>
      <nav className={styles.header__menu}>
        {headerLinks.map(({ link, title }, index) => (
          <NavLink
            data-testid={title}
            to={link}
            key={index}
            className={({ isActive }) => (isActive ? styles.active
: "")}>
            {title}
          </NavLink>
        ))}
      </nav>
      <DefaultButton buttonType={ButtonType.link}
className={styles.radius}>
        <p>Онлайн банк</p>
      </DefaultButton>
    </header>
  );
};
export default Header;
```

Файл «Footer.tsx»

```
export default const Footer: FC = () => {
  return (
    <footer>
      <div className={styles.footer}>
        <section className={styles.footer__top}>
          <div className={styles.footer__social}>
            <img src={Logo} alt="logo" />
            <address className={styles.footer__info}>
              <a data-testid="footerTel" href="tel:+71234567890">
                +7 (123) 456 78 90
              </a>
              <a data-testid="footerMail"
href="mailto:robinx.info@gmail.com">
                robinx.info@gmail.com
              </a>
            </address>
          </div>
        </section>
        <section className={styles.footer__bottom}>
```

<p>
 Мы используем файлы cookie для персонализации наших
 услуг и улучшения качества обслуживания пользователей
 ознакомьтесь с нашим веб-сайтом. Файлы cookie – это
 небольшие файлы, содержащие
 информация о предыдущих посещениях веб-сайта. Если вы
 не хотите
 чтобы использовать файлы cookie, пожалуйста, измените
 настройки вашего браузера

</p>
 </section>
 </div>
 </footer>
);
 };

Файл «Schemas.ts»

```

import { z } from "zod";
const maxDate = new Date(new Date().setFullYear(new
Date().getFullYear() - 18));
export const textSchema = z
  .string()
  .min(2, "Значение не должно быть короче 2-х символов")
  .max(30, "Значение не должно быть длиннее 30 символов")
  .regex(/^[-a-яА-ЯёЁa-zA-Z\s]+$/, "Значение должно содержать
только буквы")
  .trim();
const loginSchema = z
  .string()
  .min(4, "Логин должен содержать минимум 4 символа")
  .max(30, "Логин не должен превышать 30 символов")
  .regex(
    /^(?=.*[a-zA-Z])[a-zA-Z0-9._]+$/,
    "Логин должен содержать буквы (обязательно), цифры, точки или
подчёркивания"
  )
  .trim();
const passwordSchema = z
  .string()
  .min(6, "Пароль должен содержать минимум 6 символов")
  .max(30, "Пароль не должен превышать 30 символов")
  .regex(
    /^(?=.*[a-zA-Z])(?=.*\d)[a-zA-Z\d\W_]+$/,
    "Пароль должен содержать хотя бы одну букву и одну цифру
(допускаются специальные символы)"
  )
  .trim();
export const loginFormSchema = z
  .object({
    login: loginSchema,
    password: passwordSchema,
  })

```

```

});
export const registryFormSchema = z
  .object({
    login: loginSchema,
    password: passwordSchema,
    confirmPassword: z.string(),
  })
  .refine(data => data.password === data.confirmPassword, {
    message: "Пароли не совпадают",
    path: ["confirmPassword"]
  });
export const customizeCardFormSchema = z
  .object({
    amount: z.coerce
      .number({ message: "Обязательное поле" })
      .min(1, "Обязательное поле")
      .min(15000, "Минимальная сумма 15 000")
      .max(600000, "Максимальная сумма 600 000"),
    term: z.string().or(z.number()),
    firstName: textSchema,
    lastName: textSchema,
    middleName: textSchema
      .or(z.string().max(0).nullable())
      .transform(v => v === "" ? null : v),
    email: z.string().email("Проверьте корректность email-адреса").trim(),
    birthdate: z
      .string({ message: "Обязательное поле" })
      .regex(
        /^(0[1-9]|1[0-9]|2[0-9]|3[0-1])\/(0[1-9]|1[0-2])\/(\d{4})$/ ,
        "Введите корректную дату",
      ),
    passportSeries: z
      .string()
      .min(4, "Проверьте корректность серии паспорта")
      .trim(),
    passportNumber: z
      .string()
      .min(6, "Проверьте корректность номера паспорта")
      .trim(),
  })
  .superRefine((arg, ctx) => {
    const [day, month, year] = arg.birthdate.split("/");
    const date = new Date(+year, +month - 1, +day);
    if (date.getTime() > new Date().getTime()) {
      ctx.addIssue({
        message: "Дата не может быть больше сегодняшней",
        path: ["birthdate"],
        code: z.ZodIssueCode.custom,
      });
    }
  });

```

```

    }
    if (date.getTime() > maxDate.getTime()) {
      ctx.addIssue({
        message: "Допускаются пользователи старше 18-ти лет",
        path: ["birthdate"],
        code: z.ZodIssueCode.custom,
      });
    }
  });
export const selectSchema = z.string({ message: "Выберите один из вариантов" });
export const applicationIdFormSchema = z
  .object({
    gender: selectSchema,
    maritalStatus: selectSchema,
    dependentAmount: z
      .string({ message: "Обязательное поле" })
      .min(1, "Обязательное поле"),
    passportIssueDate: z
      .string({ message: "Обязательное поле" })
      .regex(
        /^(0[1-9]|1[0-9]|2[0-9]|3[0-1])\/(0[1-9]|1[0-2])\/(\d{4})$/,
        "Неверная дата выдачи паспорта",
      ),
    passportIssueBranch: z
      .string({ message: "Обязательное поле" })
      .min(7, "Серия должна состоять из 6 цифр"),
    employmentStatus: selectSchema,
    employerINN: z
      .string({ message: "Обязательное поле" })
      .min(12, "ИНН должен состоять из 12 цифр"),
    salary: z
      .string({ message: "Введите свою зарплату" })
      .min(1, "Введите свою зарплату")
      .max(9, "Слишком большая зарплата"),
    position: selectSchema,
    workExperienceTotal: z
      .string({ message: "Обязательное поле" })
      .min(1, "Укажите общий стаж вашей работы")
      .max(3, "Стаж должен быть до трёх цифр."),
    workExperienceCurrent: z
      .string({ message: "Обязательное поле" })
      .min(1, "Укажите свой текущий опыт работы")
      .max(3, "Стаж должен быть до трёх цифр."),
  })
  .superRefine((arg, ctx) => {
    const [day, month, year] =
      arg.passportIssueDate.split("/");
    const date = new Date(+year, +month - 1, +day);

```

```

        if (date.getTime() > new Date().getTime()) {
            ctx.addIssue({
                message: "Дата не может быть больше сегодняшней",
                path: ["passportIssueDate"],
                code: z.ZodIssueCode.custom,
            });
        }
    });
export type CustomizeCardFormFields = z.infer<typeof
customizeCardFormSchema>;
export type ApplicationIdFormFields = z.infer<typeof
applicationIdFormSchema>;

```

Файл «routes.ts»

```

export const HOME_PAGE_ROUTE: string = "/";
export const GENERAL_ROUTE: string = "/*";
export const ACCOUNT_ROUTE: string = "/account";
export const LOGIN_ROUTE: string = "/login";
export const CREDIT_CARD_ROUTE: string = "/credit-card";
export const NOT_FOUND_PAGE_ROUTE: string = "/not-found";
export const LOAN_ROUTE: string = "/loan/:id";
export const DOCUMENT_ROUTE: string = "/loan/:id/document";
export const DOCUMENT_SIGN_ROUTE: string =
"/loan/:id/document/sign";
export const CONFIRMATION_CODE_ROUTE: string = "/loan/:id/code";

```

Файл «useStep.ts»

```

// Навигация к следующему шагу
const navigateToStep = (step: number, id: string, navigate:
NavigateFunction) => {
    switch (step) {
        case 2:
            navigate(LOAN_ROUTE.replace(':id', id));
            break;
        case 3:
            navigate(DOCUMENT_ROUTE.replace(':id', id));
            break;
        case 4:
            navigate(DOCUMENT_SIGN_ROUTE.replace(':id', id));
            break;
        case 5:
            navigate(CONFIRMATION_CODE_ROUTE.replace(':id', id));
            break;
        case 6: // выдан
        case -1: // отклонено
            navigate(HOME_PAGE_ROUTE);
            break;
    }
}

```

```

}
const useStep = (step: number, id?: string) => {
  const appIdsStep = useSelector(appIdStepSelector);
  const dispatch = useAppDispatch();
  const navigate = useNavigate();
  useEffect(() => {
    if (!checkAuth()) {
      navigate(LOGIN_ROUTE);
    } else {
      if (id) {
        const currentStep = appIdsStep[id];
        if (currentStep !== step) {
          navigateToStep(currentStep, id, navigate);
        }
      }
    }
  }, [id, step, appIdsStep, navigate, dispatch]);
};
export default useStep;

```

Приложение В

(обязательное)

Бизнес - проект «Кредитный банк: онлайн-платформа для автоматизации
кредитных процессов»

Презентация

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		90

Бизнес-проект: «Кредитный банк: онлайн-платформа для автоматизации кредитных процессов»

Докладчик: Кабачек Никита Андреевич

Проблемы оформления кредитных карт



Длительное ожидание

Долгий процесс рассмотрения заявок



Бумажная волокита

Множество документов



Высокий риск ошибок

Человеческий фактор



Низкая прозрачность

Неясные условия

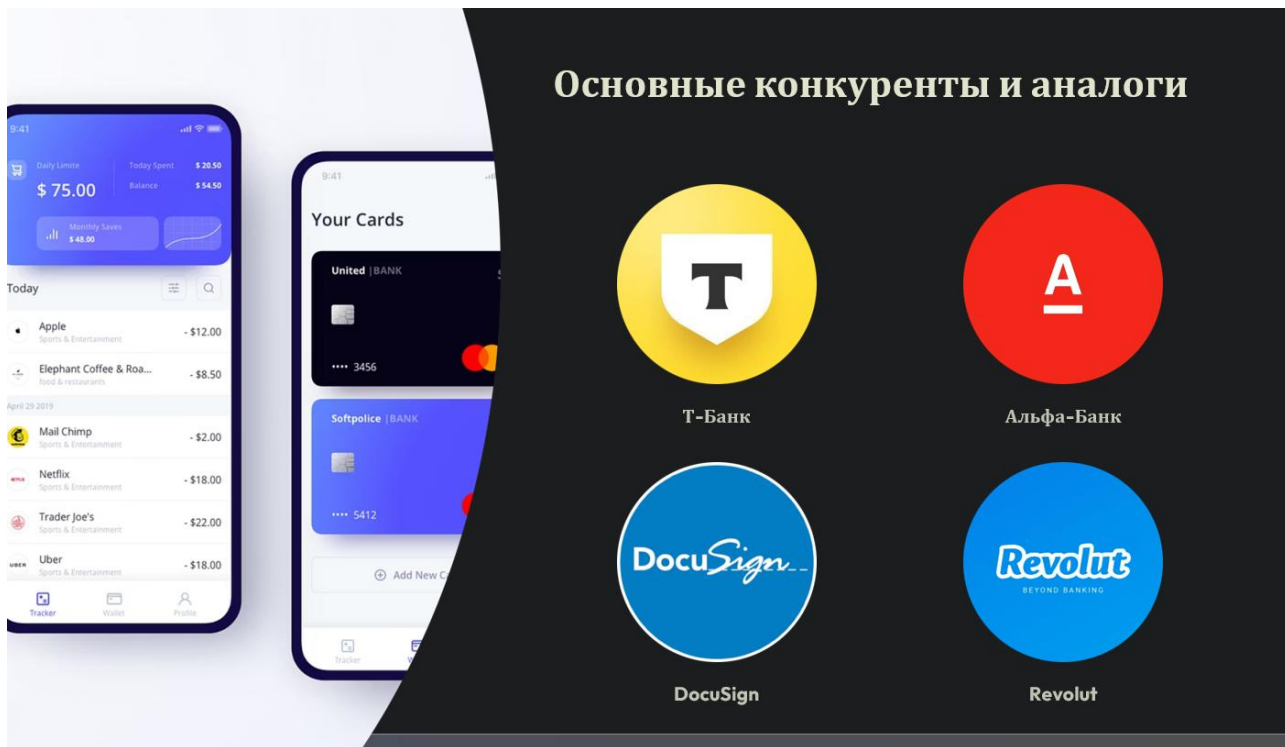


Изм.	Лист	№ докум.	Подпись	Дата

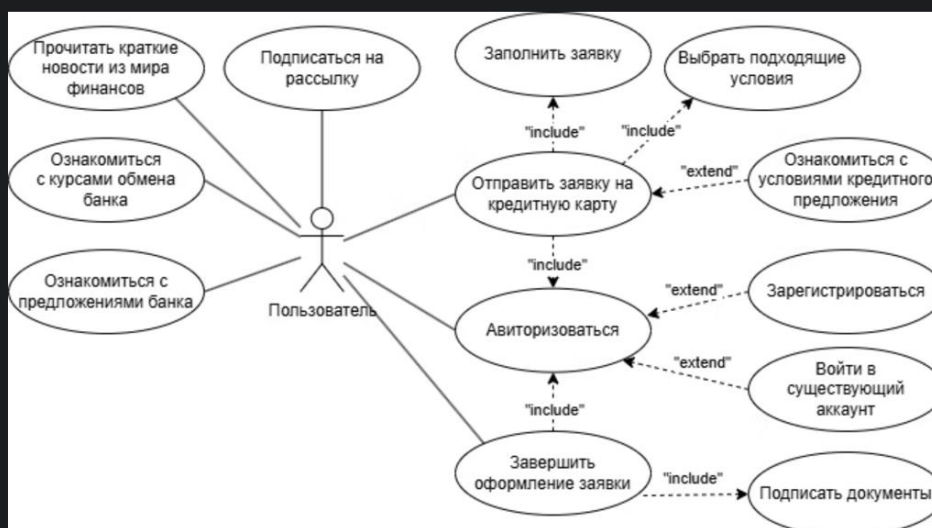
ПГУ1.090301.09.001 ПЗ

Лист

91



UML: Диаграмма вариантов использования



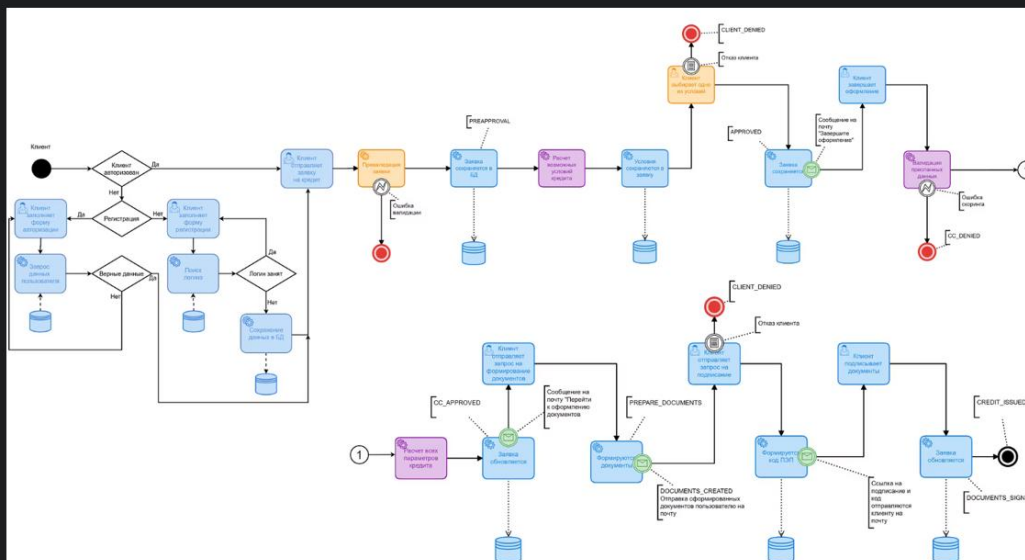
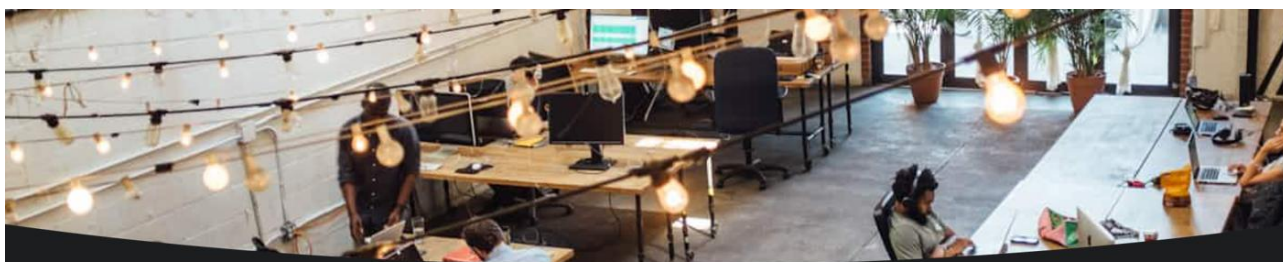
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

92

UML: Диаграмма деятельности



Стартап: Решение для разных сегментов

Частные клиенты

Быстрое одобрение, удобное оформление

Малый и средний бизнес

Доступ к финансированию

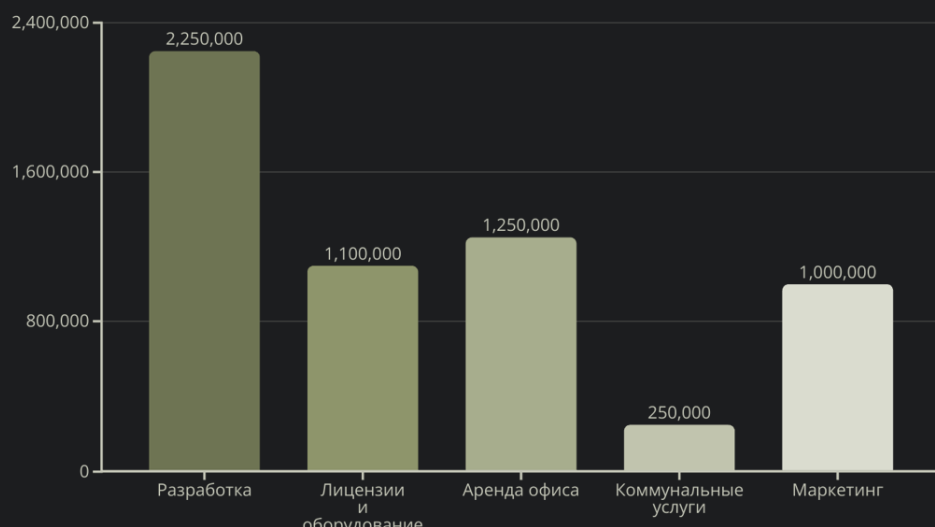
Банки и финансовые организации

Оптимизация процессов

Финтех-компании

Интеграция с сервисами

Планируемые затраты на первый год, Р



Спасибо за внимание!

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

94