

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования
Кафедра проектирования информационно-компьютерных систем
Рефакторинг и оптимизация программного кода

Отчет
по результатам выполнения лабораторных работ
и заданий к практическим занятиям

Проверил

(подпись)

А.В. Шелест

зачтено

(дата защиты)

Выполнил



(подпись)

М.А. Потапчик
гр. 214371

СОДЕРЖАНИЕ

Ссылки на репозитории.....	3
Задание №1.....	4
Задание №2.....	8
Задание №3.....	9
Задание №4.....	11
Задание №5.....	16
Задание №6.....	22
Задание №7.....	48
Задание №8.....	50
Задание №9.....	52
Задание №10.....	56

ССЫЛКИ НА РЕПОЗИТОРИИ

<https://github.com/MaryPotapchik1/Client>

<https://github.com/MaryPotapchik1/AuthService>

<https://github.com/MaryPotapchik1/BusinessService>

Задание №1

Цель: спроектировать архитектуру ПС, следуя требованиям архитектурного шаблона *Clean Architecture*, и разработать систему дизайна (*UI-kit*) пользовательского интерфейса.

Диаграмма вариантов использования. Диаграмма вариантов использования – это наиболее общее представление функционального назначения системы с точки зрения получения значимого результата для пользователя.

Пользовательские требования посредством диаграммы вариантов использования, представленные на рисунке 1.

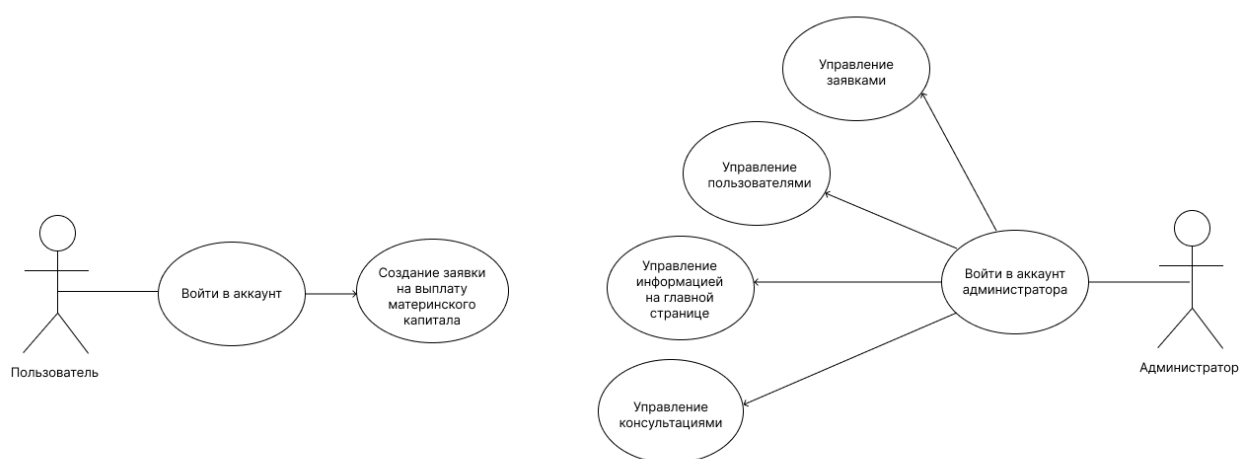


Рисунок 1.1 – Диаграмма вариантов использования

Таблица 6 – Описание действующих лиц

Действующее лицо	Вариант использования
Клиент	Авторизоваться, подать заявку на получение материнского капитала
Менеджер	Одобрять или отклонять заявки на получение материнского капитала, управлять пользователями, управлять информацией на странице, управлять поступающими консультациями

Архитектура программного средства представляет собой структуру системы, включающую программные компоненты, их внешние

характеристики и взаимосвязи между ними. Архитектурное решение определяет организацию программного средства, способы взаимодействия его частей, используемые технологии и методики для реализации, а также подходы к развитию и поддержке системы.

Контекстная диаграмма – это визуальное представление системы, показывающее её взаимодействие с внешним окружением. В ней отображаются основные входы и выходы системы, а также внешние сущности, с которыми она взаимодействует, такие как пользователи, другие системы или внешние сервисы. Эта диаграмма помогает определить границы системы, выделить её основные функции и установить контекст её использования. Она является важным инструментом для обсуждения и понимания требований к системе, а также для определения возможностей интеграции с другими системами или сервисами.

Программным средством пользуется клиент и администратор.

Клиент оставляет заявку для получения материнского капитала.

Администратор же одобряет или отклоняет заявку на получение материнского капитала.

Контекстная диаграмма модели для программного средства представлена на рисунке 1.2.

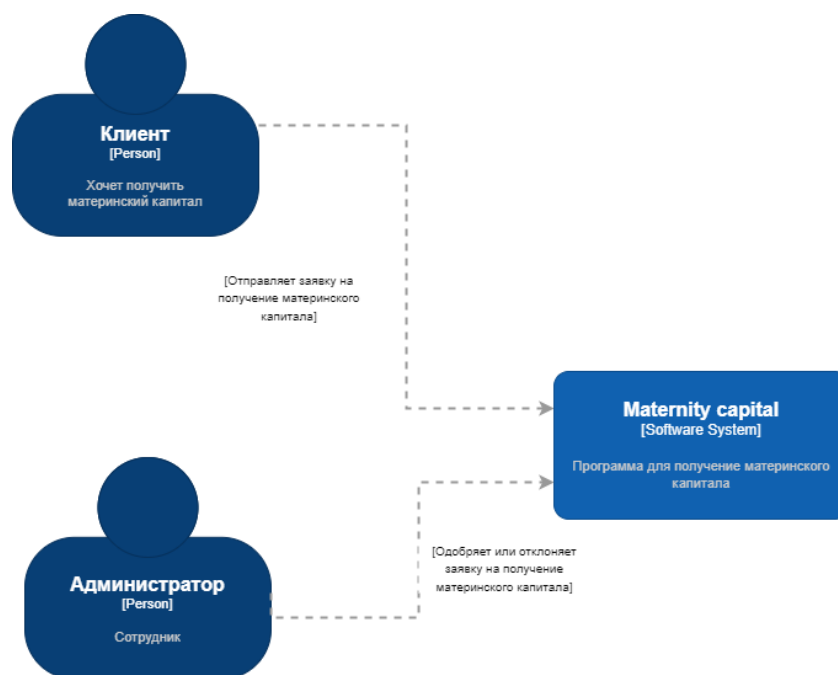


Рисунок 1.2 – Контекстная диаграмма

Контейнерная диаграмма – это диаграмма, используемая в архитектуре программного обеспечения для визуализации высокоуровневой структуры

системы. Она отображает компоненты или сервисы системы в виде контейнеров, которые группируют логически связанные элементы. Каждый контейнер представляет собой набор связанных компонентов, обеспечивающих определенную функциональность или сервис.

На контейнерной диаграмме показываются взаимосвязи и зависимости между контейнерами, а также их взаимодействие с внешними системами или средами. Это позволяет разработчикам и архитекторам лучше понимать структуру системы, определять границы компонентов и выявлять ключевые зависимости. Контейнерные диаграммы часто используются для документирования архитектуры системы, обеспечивая ясное представление её компонентов и их взаимосвязей.

Контейнерная диаграмма модели для программного средства представлена на рисунке 1.3.

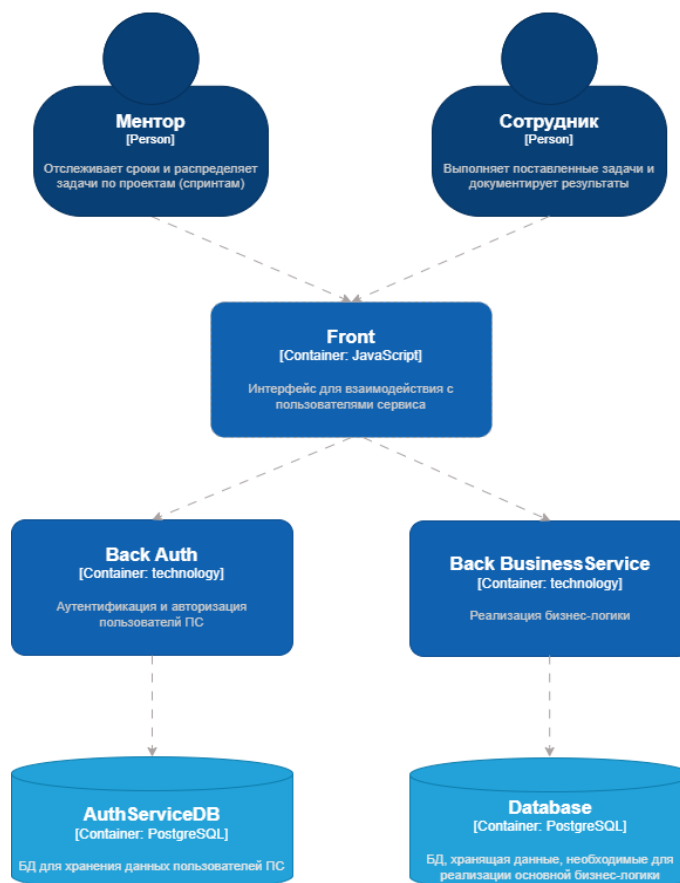


Рисунок 1.3 – Контейнерная диаграмма

Компонентная диаграмма – это вид диаграммы, используемый в инженерии программного обеспечения для визуализации и описания компонентов, из которых состоит система, и их взаимосвязей. Она позволяет

представить архитектурное решение системы на более высоком уровне абстракции, выделяя отдельные модули или компоненты, которые выполняют определенные функции.

На компонентной диаграмме компоненты представлены в виде прямоугольников с указанием их имени и основных атрибутов. Взаимодействие между компонентами обычно показывается с помощью стрелок, указывающих направление потока данных или управления.

Основная цель компонентной диаграммы - обеспечить понимание структуры и организации системы, выделить ключевые компоненты и их взаимосвязи. Она может использоваться для документирования архитектуры системы, обеспечения общего понимания участников проекта, а также в качестве основы для проектирования и разработки ПО.

Компонентная диаграмма модели для приложения представлена на рисунке 1.4.

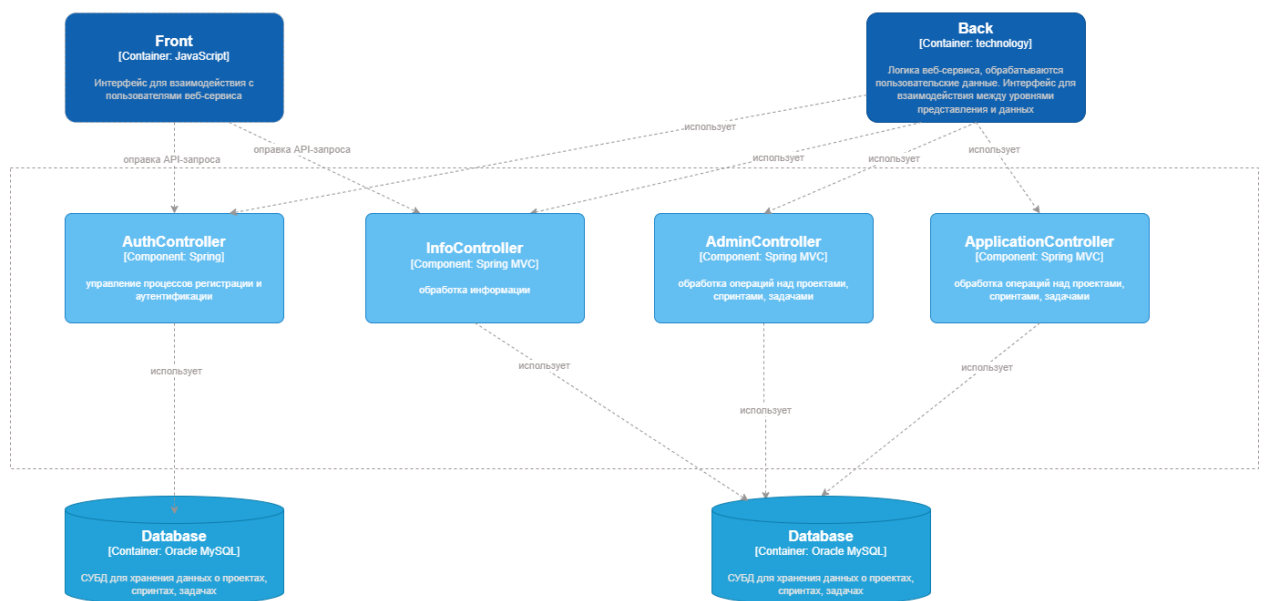


Рисунок 1.4 – Компонентная диаграмма

Диаграмма кодового уровня – это графическое представление исходного кода программы или ее отдельных модулей на уровне исходного текста. Она позволяет визуализировать структуру программы, включая классы, методы, переменные и другие элементы, а также их взаимосвязи и зависимости.

На диаграмме кодового уровня каждый компонент программы представлен в виде блока или прямоугольника, внутри которого указывается его имя и сопутствующая информация. Взаимосвязи между компонентами

могут быть показаны с использованием стрелок или линий, указывающих на вызовы методов, использование переменных и другие типы связей.

Основная цель диаграммы кодового уровня - обеспечить наглядное представление структуры программы для программистов и других участников проекта. Она помогает лучше понять организацию и взаимодействие компонентов программы, улучшить ее архитектуру, а также облегчить понимание исходного кода при его анализе, отладке и сопровождении.

Диаграмма кодового уровня *C4 Model* для приложения представлена на рисунке 1.5.

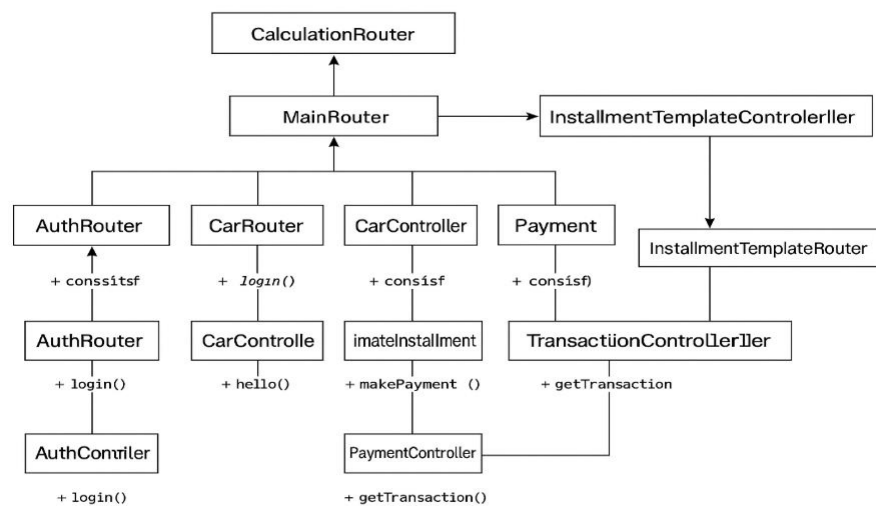


Рисунок 1.5 – Диаграмма кодового уровня *C4 Model*

На данной диаграмме представлены элементы, а также взаимодействие между ними.

Задание №2

Цель: проектирование пользовательского интерфейса ПС.

Диаграмма пользовательского потока (*User-Flow*) – это визуализация последовательности действий, которые пользователь выполняет в приложении или на веб-сайте для достижения конкретной цели. Обычно на диаграмме показываются различные экраны или страницы, которые пользователь видит, и связи между ними.

User-flow диаграмма для пользователей в роли администратора представлена на рисунке 2.1.

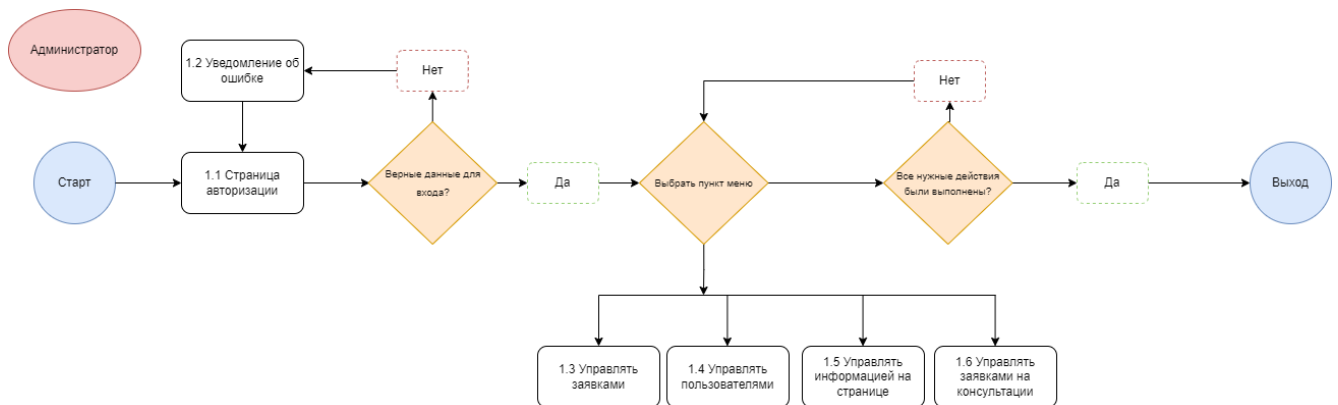


Рисунок 2.1 – *User-flow* диаграмма для Администратора

User-flow диаграмма для пользователей в роли зарегистрированного клиента представлена на рисунке 2.2.

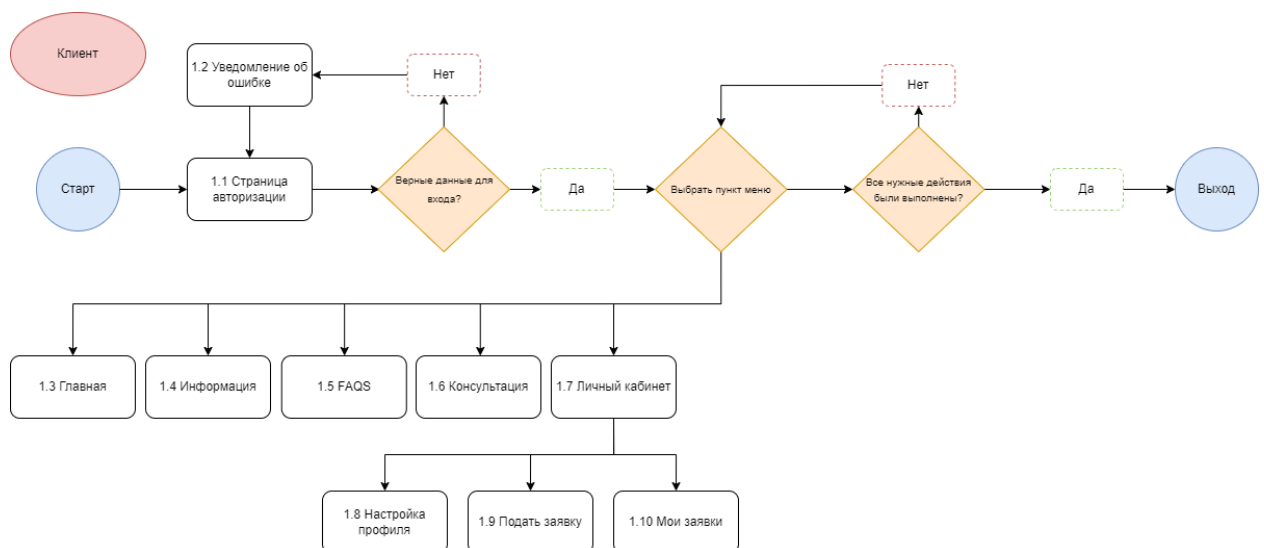


Рисунок 2.2 – *User-flow* диаграмма для Клиента

В итоге проектирования дизайна программного приложения был создан современный и инновационный дизайн, учитывающий все необходимые факторы и данные. Этот дизайн полностью соответствует современным требованиям и подходит для эффективного использования программного продукта.

Задание №3

Цель: реализация клиентской части ПС.

Для реализации клиентской части программной системы были выбраны следующие технологии: *JavaScript*, *React*, *Redux Toolkit*, *TypeScript*, *Bootstrap*. Такой стек технологий обеспечивает современный, масштабируемый и поддерживаемый пользовательский интерфейс.

Выбор данных инструментов обусловлен следующими преимуществами:

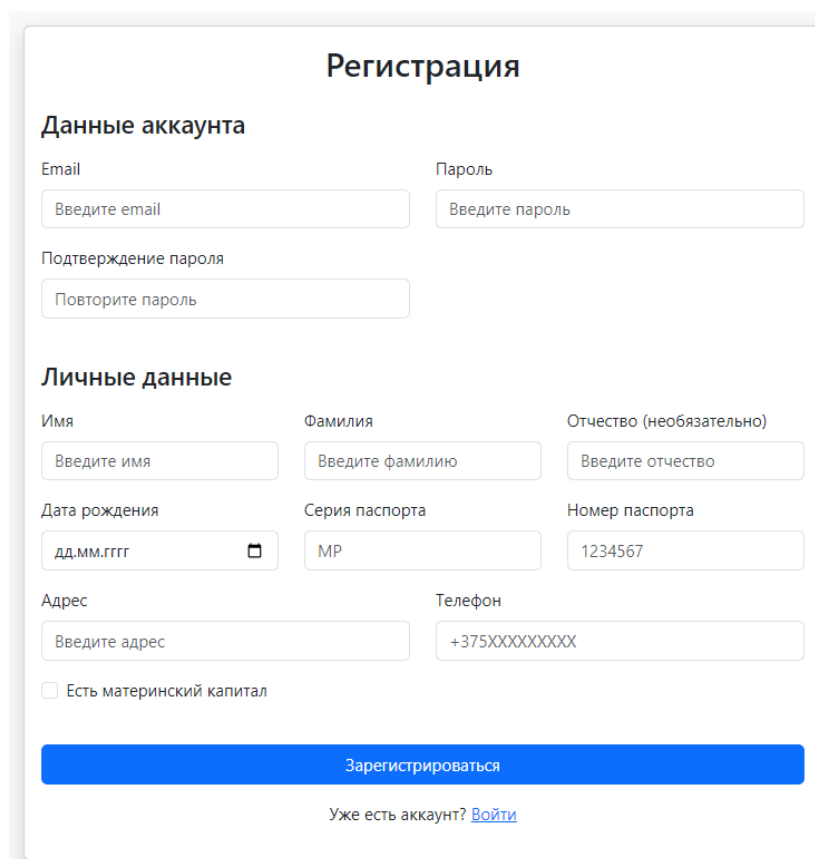
- *React* – одна из самых популярных библиотек для построения пользовательских интерфейсов, основанная на компонентном подходе, что позволяет создавать переиспользуемые и легко тестируемые элементы интерфейса.
- *Redux Toolkit* – упрощённая и рекомендуемая практика управления состоянием в *React*-приложениях. Он предоставляет удобный способ организации хранилища данных и логики их обработки, повышая читаемость и модульность кода.
- *TypeScript* – надмножество *JavaScript*, добавляющее строгую типизацию. Это снижает количество ошибок на этапе разработки, улучшает автодополнение в редакторах кода и способствует лучшей документации.
- *Bootstrap* – фреймворк для быстрой и адаптивной вёрстки, обеспечивающий кроссбраузерную совместимость и ускоряющий процесс создания современного и отзывчивого интерфейса.

Дополнительные преимущества выбранного подхода:

- Лёгкость интеграции с *REST API* и другими внешними сервисами.
- Широкое сообщество и обилие документации, что снижает порог вхождения для новых разработчиков.
- Возможность масштабирования проекта без переработки архитектуры.
- Простота автоматизированного тестирования компонентов и логики приложения.

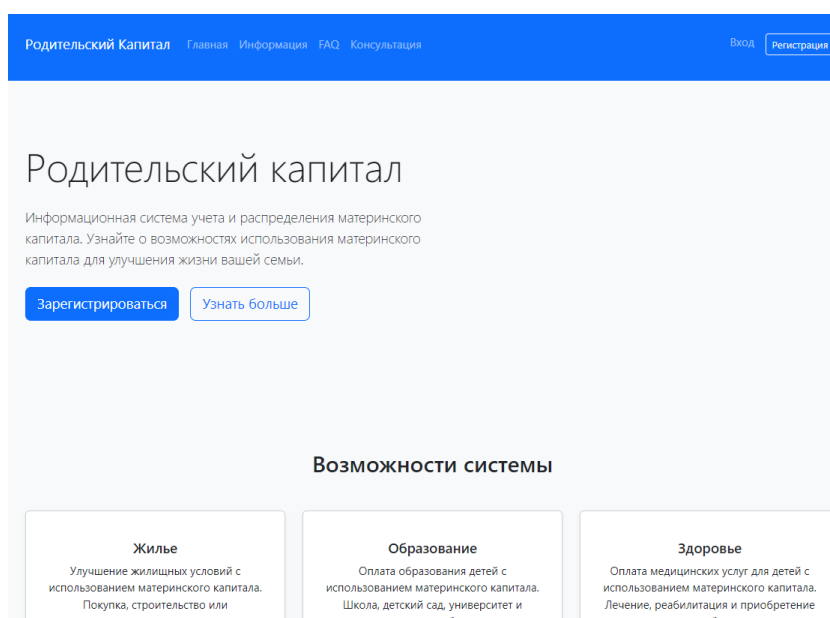
Такой выбор технологий позволяет создать удобный, стабильный и расширяемый пользовательский интерфейс, соответствующий современным требованиям веб-разработки.

Ниже представлен дизайн программного средства на рисунках 3.1, 3.2, 3.3.



The registration form is titled "Регистрация" (Registration). It is divided into two main sections: "Данные аккаунта" (Account Data) and "Личные данные" (Personal Data). The "Данные аккаунта" section includes fields for "Email" (with placeholder "Введите email"), "Пароль" (with placeholder "Введите пароль"), and "Подтверждение пароля" (with placeholder "Повторите пароль"). The "Личные данные" section includes fields for "Имя" (with placeholder "Введите имя"), "Фамилия" (with placeholder "Введите фамилию"), "Отчество (необязательно)" (with placeholder "Введите отчество"), "Дата рождения" (with placeholder "дд.мм.гггг" and a calendar icon), "Серия паспорта" (with placeholder "MP"), "Номер паспорта" (with placeholder "1234567"), "Адрес" (with placeholder "Введите адрес"), and "Телефон" (with placeholder "+375XXXXXXXXX"). There is a checkbox labeled "Есть материнский капитал" (Have maternity capital). At the bottom, there is a blue button labeled "Зарегистрироваться" (Register) and a link "Уже есть аккаунт? Войти" (Already have an account? Log in).

Рисунок 3.1 – Регистрация



The main screen features a blue header with the text "Родительский Капитал" and navigation links: "Главная", "Информация", "FAQ", "Консультация". On the right side of the header are links for "Вход" (Login) and "Регистрация" (Registration). The main content area has the title "Родительский капитал" and a description: "Информационная система учета и распределения материнского капитала. Узнайте о возможностях использования материнского капитала для улучшения жизни вашей семьи." Below this are two buttons: "Зарегистрироваться" (Register) and "Узнать больше" (Learn more). The section "Возможности системы" (System Capabilities) is divided into three columns: "Жилье" (Housing) with text "Улучшение жилищных условий с использованием материнского капитала. Покупка, строительство или реконструкция жилья.", "Образование" (Education) with text "Оплата образования детей с использованием материнского капитала. Школа, детский сад, университет и дополнительное образование.", and "Здоровье" (Health) with text "Оплата медицинских услуг для детей с использованием материнского капитала. Лечение, реабилитация и приобретение средств реабилитации."

Рисунок 3.2 – Главный экран клиента

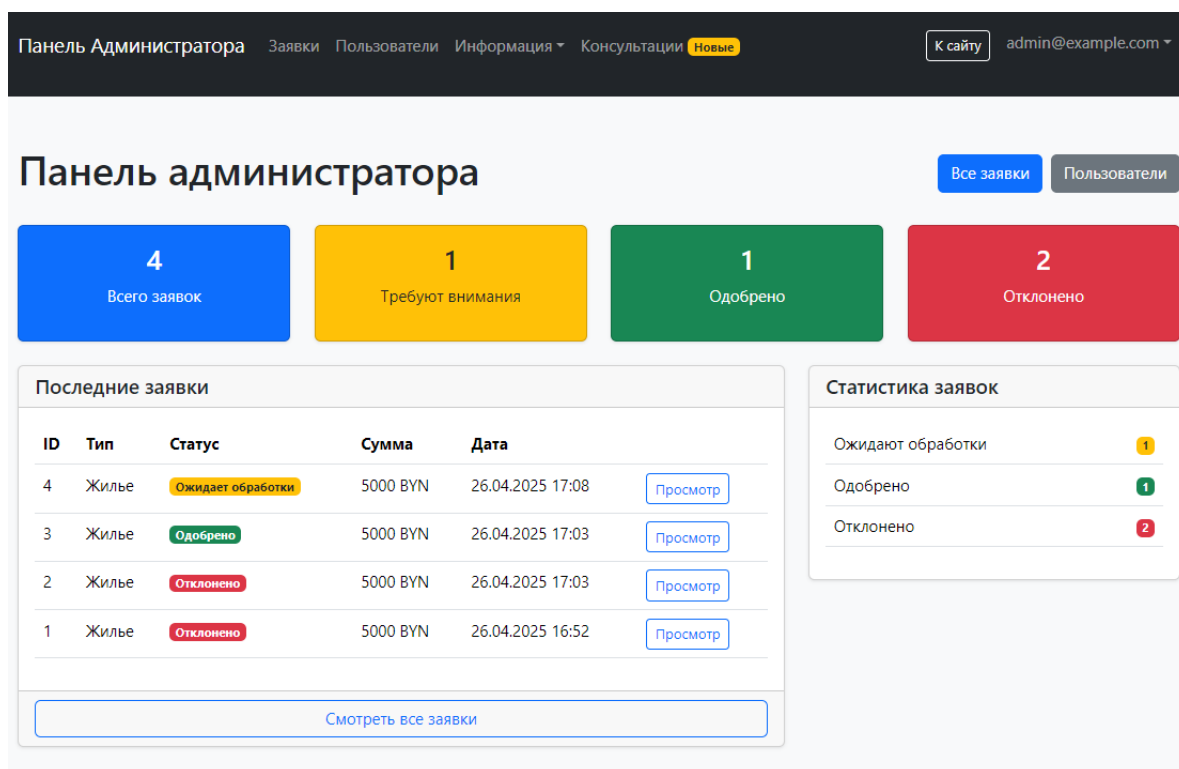


Рисунок 3.3. – Главный экран администратора

Таким образом была представлена реализация клиентской части ПС.

Задание №4

Цель: спроектировать схему БД и представить описание ее сущностей и их атрибутов.

В соответствии с требованиями предметной области «Веб-ресурс *WordPress* на примере интернет-магазина строительных материалов», была спроектирована реляционная база данных, обеспечивающая хранение информации о пользователях, рассрочках, шаблонах рассрочек и платежах. Схема приведена к третьей нормальной форме (3НФ). В таблице 4.1 расписаны сущности и их атрибуты.

Таблица 4.1 – Описание сущностей и их атрибутов

Поле	Тип данных	Описание
<i>User</i>		
<i>id</i>	<i>number</i>	Первичный ключ
<i>email</i>	<i>string</i>	<i>Email</i> пользователя
<i>password</i>	<i>string</i>	Пароль
<i>role</i>	<i>string</i>	Роль пользователя
<i>created_at</i>	<i>Date</i>	Дата создания
<i>UserProfile</i>		
<i>user_id</i>	<i>number</i>	Внешний ключ
<i>first_name</i>	<i>string</i>	Имя
<i>last_name</i>	<i>string</i>	Фамилия
<i>middle_name</i>	<i>string</i>	Отчество
<i>birth_date</i>	<i>Date</i>	Дата рождения
<i>passport_series</i>	<i>string</i>	Серия паспорта
<i>passport_number</i>	<i>string</i>	Номер паспорта
<i>address</i>	<i>string</i>	Адрес
<i>phone</i>	<i>string</i>	Телефон
<i>has_maternal_capital</i>	<i>boolean</i>	Наличие материнского капитала
<i>maternal_capital_amount</i>	<i>number</i>	Сумма материнского капитала
<i>housing_type</i>	<i>string</i>	Тип жилья
<i>living_area</i>	<i>number</i>	Жилая площадь
<i>ownership_status</i>	<i>enum</i>	Статус собственности
<i>Application</i>		
<i>id</i>	<i>number</i>	Первичный ключ
<i>user_id</i>	<i>number</i>	Внешний ключ
<i>application_type</i>	<i>enum</i>	Тип заявки
<i>status</i>	<i>enum</i>	Статус
<i>requested_amount</i>	<i>number</i>	Запрошенная сумма
<i>approved_amount</i>	<i>number</i>	Одобренная сумма
<i>rejection_reason</i>	<i>string</i>	Причина отказа

<i>purpose</i>	<i>string</i>	Цель заявки
<i>description</i>	<i>string</i>	Описание
<i>created_at</i>	<i>Date</i>	Дата создания
<i>updated_at</i>	<i>Date</i>	Дата обновления
<i>ApplicationComment</i>		
<i>id</i>	<i>number</i>	Первичный ключ
<i>application_id</i>	<i>number</i>	Внешний ключ
<i>user_id</i>	<i>number</i>	Внешний ключ
<i>is_admin</i>	<i>boolean</i>	Комментарий от админа
<i>comment</i>	<i>string</i>	Текст комментария
<i>created_at</i>	<i>Date</i>	Дата создания
<i>FamilyMember</i>		
<i>id</i>	<i>number</i>	Первичный ключ
<i>user_id</i>	<i>number</i>	Внешний ключ
<i>relation_type</i>	<i>enum</i>	Родственная связь
<i>first_name</i>	<i>string</i>	Имя
<i>last_name</i>	<i>string</i>	Фамилия
<i>middle_name</i>	<i>string?</i>	Отчество
<i>birth_date</i>	<i>Date</i>	Дата рождения
<i>document_type</i>	<i>enum</i>	Тип документа
<i>document_number</i>	<i>string</i>	Номер документа

На рисунке 4.1 отображена схема базы данных.

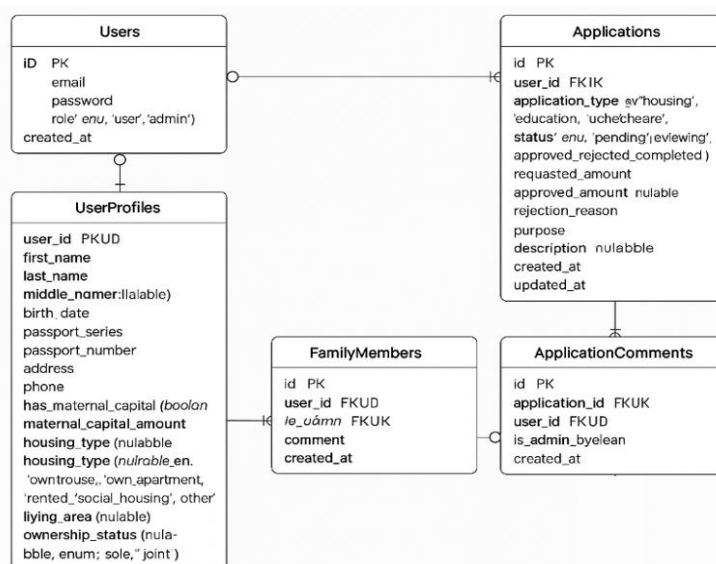


Рисунок 4.1 – Схема базы данных

На данной схеме представлены элементы, а также взаимодействие между ними.

Задание №5

Цель: представить детали реализации ПС через *UML*-диаграммы.

Диаграмма деятельности для программного средства представлена на рисунке 5.1.

Диаграмма деятельности представлена на рисунке 5.1.

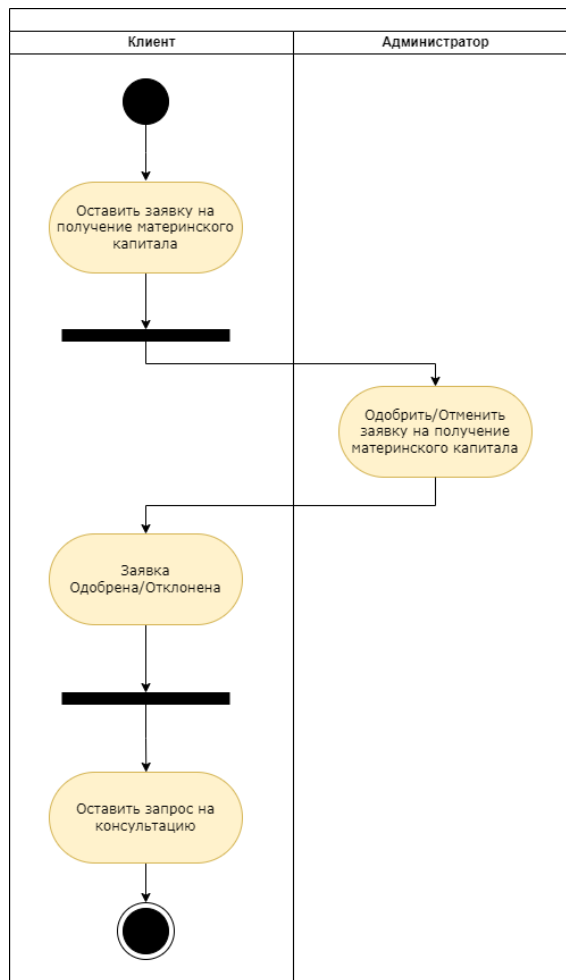


Рисунок 5.1 – Диаграмма деятельности

Диаграмма последовательности представлена на рисунке 5.2.

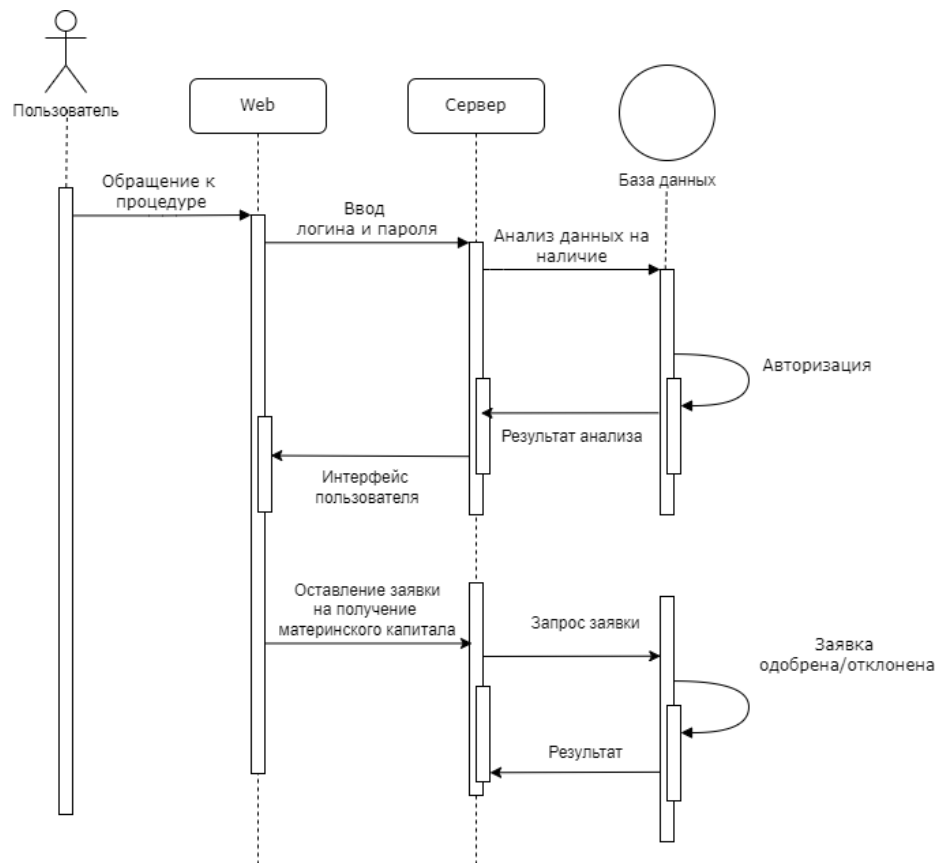


Рисунок 5.2 – Диаграмма последовательности

Диаграмма состояний представлена на рисунке 5.3.



Рисунок 5.3 – Диаграмма состояний

Диаграмма размещений представлена на рисунке 5.4.

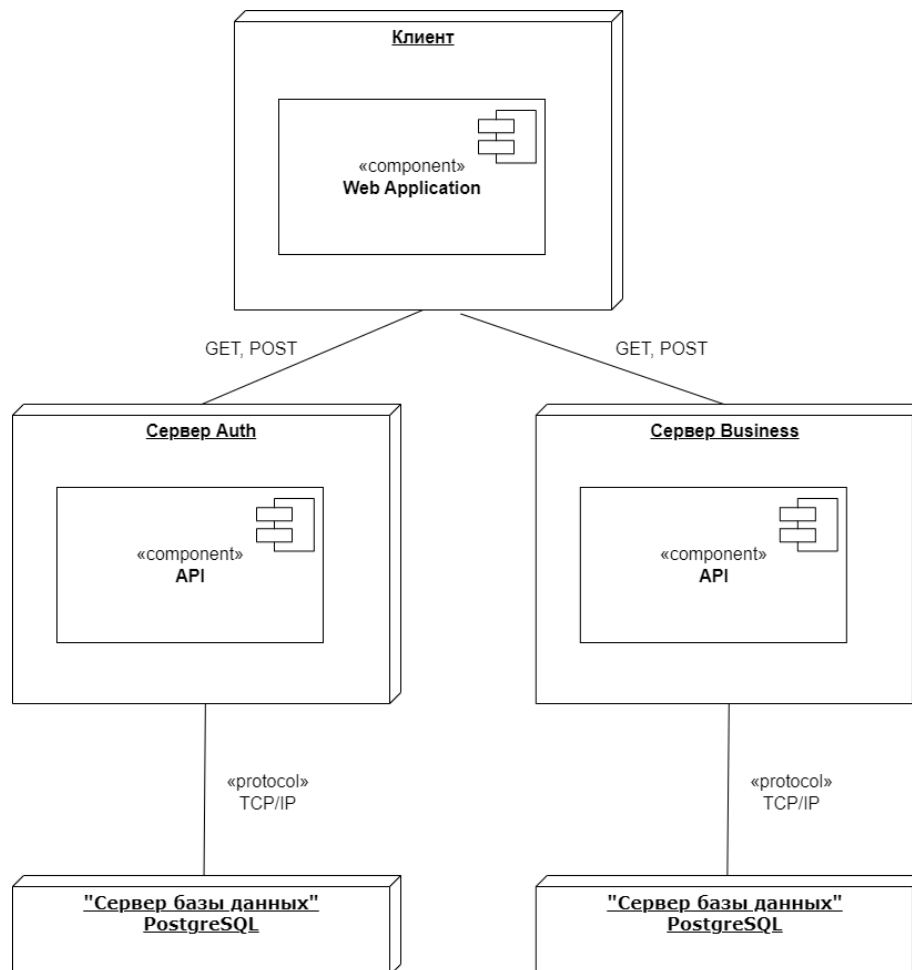


Рисунок 5.4 – Диаграмма размещений

В заключение, использование *UML*-диаграмм, таких как диаграмма состояний, диаграмма последовательности, диаграмма размещений, а также диаграмма деятельности обеспечивает систематизированный подход к проектированию и анализу программного средства. Эти модели не только упрощают понимание ключевых процессов, но и позволяют выявить потенциальные проблемы на этапе проектирования, что снижает риски и способствует успешной реализации системы.

Задание №6

Цель: разработать документацию к ПС с *Open API* и оценить качество программного кода

В процессе разработки серверной части программного средства была создана документация к *REST API* с применением спецификации *OpenAPI 3.0*. Документация описывает все доступные конечные точки: авторизацию и аутентификацию пользователей, управление профилем и членами семьи, а также административные функции. В описании каждой конечной точки указаны:

- *HTTP*-методы и маршруты;
- заголовки авторизации (включая работу с *JWT*);
- структура тела запроса и ответа;
- возможные коды ответов и сообщения об ошибках
- права доступа (например, только для администратора).

Документация обеспечивает прозрачность взаимодействия клиентской и серверной частей и может быть использована для генерации клиентских библиотек или автоматического тестирования.

Для оценки качества кода были проанализированы следующие аспекты:

- читаемость. Соблюдение соглашений по именованию, модульность и логическая организация кода;
- надёжность. Реализована проверка авторизации и ролей, обработка ошибок сторонних сервисов (через *axios*), и возврат корректных *HTTP*-статусов;
- повторное использование. Общие *middleware*-функции, такие как *authenticateToken* и *authorizeRole*, переиспользуются в различных маршрутах;
- обработка исключений. В контроллерах предусмотрена обработка исключений с логированием ошибок и возвратом информативных сообщений;
- безопасность. Реализована проверка *JWT*-токенов, смена пароля требует авторизации, конфиденциальные данные не передаются в открытом виде.

Таким образом, разработанная документация и структура кода обеспечивают хорошую расширяемость, надёжность и сопровождение ПС.

Задание №7

Цель: реализация системы аутентификации и авторизации пользователей ПС и механизмов обеспечения безопасности данных.

Для реализации аутентификации используется пара «электронная почта + пароль». При регистрации пользователь передаёт свой пароль, который перед сохранением в базу данных проходит хэширование с помощью криптографической библиотеки *bcrypt*. Это означает, что даже при утечке данных невозможно получить исходные пароли пользователей, так как хранимые значения представляют собой стойкие односторонние хэши.

Хэширование производится с использованием 10 раундов соли, что обеспечивает дополнительную стойкость к атакам перебором (*brute force*) и к атаке по радужным таблицам. Пароль никогда не хранится и не передаётся в открытом виде.

После успешной регистрации или входа в систему пользователю выдается *JWT*-токен (*JSON Web Token*), в котором зашифрована информация о пользователе и его роли. Этот токен используется для дальнейшей аутентификации в каждом запросе, что избавляет от необходимости постоянного хранения сессий на сервере.

Система поддерживает разграничение доступа на основе ролей. В рамках проекта реализовано два уровня прав:

- Обычный пользователь (*user*) – имеет доступ к функционалу управления собственным профилем и информацией о членах семьи;
- Администратор (*admin*) – обладает расширенными правами, включая просмотр всех пользователей, редактирование профилей, сбор статистики и прочее.

Для защиты маршрутов используется *middleware*-функция *authorizeRole*, проверяющая роль пользователя на основании переданного *JWT*. Это обеспечивает строгую проверку доступа к защищённым маршрутам и предотвращает несанкционированное выполнение операций.

Также реализована функция *authenticateToken*, которая декодирует токен и проверяет его подлинность. В случае недействительного или отсутствующего токена, доступ к ресурсу блокируется, а пользователю возвращается сообщение об ошибке.

Система обеспечивает хранение полной информации о пользователях в реляционной базе данных *PostgreSQL*. Для каждого пользователя создаётся расширенный профиль, содержащий:

- фамилию, имя и отчество;
- дату рождения;

- паспортные данные;
- адрес проживания;
- номер телефон;
- данные о наличии материнского капитала и его размере;
- информацию о жилищных условиях: тип жилья, площадь, статус владения.

Кроме того, пользователь может добавить сведения о членах своей семьи: указать степень родства, ФИО, дату рождения и данные документа. Это реализуется через отдельную таблицу *family_members*, связанную с основной таблицей пользователей.

Для защиты этих данных применяются следующие меры:

Все обращения к чувствительной информации проходят через аутентификацию и авторизацию.

Сервер не возвращает пароли в ответах на запросы (поле *password* при выборке пользователей обнуляется).

Используются параметризованные *SQL*-запросы через метод *query*, что предотвращает *SQL*-инъекции.

Во всех функциях обработки данных реализована система перехвата и логирования ошибок. В случае возникновения исключения система фиксирует его и возвращает безопасное сообщение, не раскрывающее детали реализации или содержимого базы данных. Это важно с точки зрения безопасности, чтобы злоумышленник не мог получить информацию о внутреннем устройстве системы или структуре запросов.

Архитектура системы проектировалась с учётом масштабируемости. При необходимости можно добавить новые роли, расширить формат профиля, внедрить двухфакторную аутентификацию или интеграцию с внешними сервисами.

Задание №8

Цель: разработать *unit*- и интеграционные тесты для оценки работоспособности реализованной функциональности ПС.

Тест-кейсы для проверки уровня базовых пользовательских требований приведены в таблице 8.1.

Таблица 8.1 – Тест-кейсы для проверки уровня базовых пользовательских требований

Заглавие тест-кейса	Шаги тест-кейса	Ожидаемый результат	Фактический результат
<i>УС 1</i> Проверка функциональности входа в личный кабинет пользователя с корректными и некорректными данными.	1 Открыть программное средство 2 Нажать на кнопку «Вход» 3 Ввести корректные логин и пароль 4 Проверить успешность входа и доступ к личному кабинету 5 Выйти из системы. 6 Повторить шаги 2-5 с некорректными данными (неверный логин и/или пароль) 7 Проверить сообщение об ошибке и отказ в доступе	1 Программное средство открывается без ошибок. 2 Переход на страницу авторизации проходит успешно 3 Корректные логин и пароль вводятся без ошибок 4 Пользователь успешно авторизуется и попадает на главную страницу 5 Личный кабинет отображает персональные данные пользователя и предоставляет доступ к функциям кабинета 6 При повторном входе с некорректными данными появляется сообщение об ошибке авторизации и пользователь не авторизуется	Фактический результат совпал с ожидаемым
<i>УС 2</i> Проверка функциональности входа в аккаунт администратора с правильным логином и паролем.	1 Открыть программное средство 2 Перейти на главной странице к кнопке «Панель администратора» 3 Ввести корректный логин и пароль администратора 4 Нажать кнопку «Вход»	1 Программное средство открывается без ошибок. 2 Переход на страницу авторизации проходит успешно 3 Корректные логин и пароль вводятся без ошибок 4 Администратор успешно авторизуется и	Фактический результат совпал с ожидаемым

	5 Проверить успешность входа и доступ к административной панели	попадает на главную страницу административной панели 5 Административная панель отображает все доступные функции и данные	
УС 3 Проверка функциональности подачи заявки для выплаты материальной выплаты	1 Открыть программное средство 2 Если клиент ранее зарегистрирован, нужно нажать на кнопку «Вход». Если клиент не зарегистрирован, то нажать на кнопку «Регистрация» и пройти регистрацию. После этого нажать на кнопку «Вход» 3 Ввести корректные логин и пароль 4 Нажать кнопку «Вход» 5 На главной странице нажать на кнопку «Подать заявку» 6 Ввести все необходимые данные 7 Нажать на кнопку «Создать заявку»	1 Программное средство открывается без ошибок 2 Переход на страницу авторизации проходит успешно 3 Корректные логин и пароль вводятся без ошибок 4 Пользователь успешно авторизуется и попадает на главную страницу 5 Пользователь нажимает на кнопку «Подать заявку» 6 Данные вводятся без ошибок 7 Пользователь успешно оставляет заявку на получение выплаты материнского капитала 8 В личном кабинете отображается информация о подтвержденной или отмененной заявки на получение материнского капитала	Фактический результат совпал с ожидаемым
УС 4 Проверка функциональности администратором подтверждать или отменять заявку на выплату материнского капитала	1 Открыть программное средство 2 Нажать на кнопку «Вход» 3 Ввести корректные логин и пароль 4 Нажать кнопку «Вход» 5 Перейти к разделу «Завки»	1 Программное средство успешно открывается без ошибок 2 После нажатия на кнопку «Панель администратора», система перенаправляет пользователя на страницу	Фактический результат совпал с ожидаемым

	6 Нажать на заявку 7 Нажать на кнопку «Одобрить/Отклонить» заявку 8 Убедиться, что статус данной заявки поменялся у администратора 9 Убедиться, что у клиента заявка поменяла свой статус	аутентификации администратора 3 Корректные логин и пароль вводятся без ошибок 4 Администратор успешно авторизуется и попадает на главную страницу панели управления 5 Переход к разделу «Заявки» проходит успешно 6 Статус заявки успешно подтверждается или отменяется, и статус бронирования меняется на «Подтверждено» или «Отменено» соответственно 7 Заявка у клиента поменяла свой статус на Одобрено или Отклонено	
УС 5 Проверка функциональности оставления заявки на консультацию при вводе всех необходимых и корректных данных.	1 Открыть программное средство 2 Нажать на кнопку «Вход/Регистрация» 3 Ввести корректные логин и пароль 4 Нажать кнопку «Вход» 5. Перейти к разделу «Консультации» 6 Оставить заявку на консультацию 7 Нажать «Отправить запрос» 8 Проверить отображение данной заявки на консультацию в разделе «Управлять отзывами» со страницы администратора	1 Программное средство открывается без ошибок 2 Переход на страницу авторизации проходит успешно 3 Корректные логин и пароль вводятся без ошибок 4 Пользователь успешно авторизуется и попадает на главную страницу 5 Переход в раздел «Консультации» проходит успешно 6 Текст вводится без ошибок 8 Заявка успешно сохраняется 9 Заявка отображается в разделе «Консультации»	Фактический результат совпал с ожидаемым

		со страницы администратора	
--	--	-------------------------------	--

В результате выполнения тестирования разработанного программного приложения, предназначенного для автоматизации процесса получения заявки на получение материнского капитала, было подтверждено его соответствие его функциональной полноты и базовым требованиям.

userService.test.js

```
// userService.test.js
const userService = require('../services/userService');
const { User, UserProfile, FamilyMember } = require('../models');

jest.mock('../models'); // Мокаем Sequelize-модели

describe('userService', () => {
  describe('createUser', () => {
    it('создает пользователя с хешированным паролем', async () => {
      const mockUser = { id: 1, email: 'test@example.com', role: 'user' };
      User.create.mockResolvedValue(mockUser);

      const result = await userService.createUser('test@example.com',
'password123');

      expect(User.create).toHaveBeenCalledWith(expect.objectContaining({
        email: 'test@example.com',
        password: expect.any(String), // хешированный
        role: 'user',
      }));
      expect(result).toEqual(mockUser);
    });

    it('возвращает null при ошибке', async () => {
      User.create.mockRejectedValue(new Error('DB Error'));

      const result = await userService.createUser('test@example.com', 'pass');

      expect(result).toBeNull();
    });
  });

  describe('findUserByEmail', () => {
    it('находит пользователя по email', async () => {
      const mockUser = { id: 1, email: 'findme@example.com' };
      User.findOne.mockResolvedValue(mockUser);

      const result = await userService.findUserByEmail('findme@example.com');

      expect(User.findOne).toHaveBeenCalledWith({      where:      {      email:
'findme@example.com' } }));
    });
  });
});
```

```

        expect(result).toEqual(mockUser);
    });
});

describe('createUserProfile', () => {
    it('создает профиль пользователя', async () => {
        const profileData = { name: 'John', user_id: 1 };
        const mockProfile = { id: 1, ...profileData };
        UserProfile.create.mockResolvedValue(mockProfile);

        const result = await userService.createUserProfile(profileData);

        expect(UserProfile.create).toHaveBeenCalledWith(profileData);
        expect(result).toEqual(mockProfile);
    });
});

describe('addFamilyMember', () => {
    it('добавляет члена семьи', async () => {
        const member = { name: 'Son', user_id: 1 };
        const mockMember = { id: 5, ...member };
        FamilyMember.create.mockResolvedValue(mockMember);

        const result = await userService.addFamilyMember(member);

        expect(FamilyMember.create).toHaveBeenCalledWith(member);
        expect(result).toEqual(mockMember);
    });
});

describe('deleteFamilyMember', () => {
    it('удаляет члена семьи и возвращает true', async () => {
        const mockDestroy = jest.fn().mockResolvedValue(1);
        FamilyMember.destroy = mockDestroy;

        const result = await userService.deleteFamilyMember(5);

        expect(mockDestroy).toHaveBeenCalledWith({ where: { id: 5 } });
        expect(result).toBe(true);
    });
});
});

```

userController.test

```

import { register, login, changePassword } from
'../controllers/userController';
import * as UserModel from '../models/User';
import httpMocks from 'node-mocks-http';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';

jest.mock('../models/User');
jest.mock('bcrypt');
jest.mock('jsonwebtoken');

```

```

describe('User Controller', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  describe('register', () => {
    it('should register a new user', async () => {
      const req = httpMocks.createRequest({
        method: 'POST',
        body: {
          email: 'test@example.com',
          password: 'password123',
          profile: { name: 'Test User' },
          familyMembers: [{ name: 'Child', age: 5 }],
        },
      });

      const res = httpMocks.createResponse();

      (UserModel.findUserByEmail as jest.Mock).mockResolvedValue(null);
      (UserModel.createUser as jest.Mock).mockResolvedValue({ id: 1, email:
'test@example.com', role: 'user' });
      (UserModel.createUserProfile as jest.Mock).mockResolvedValue(true);
      (UserModel.addFamilyMember as jest.Mock).mockResolvedValue(true);
      (jwt.sign as jest.Mock).mockReturnValue('fake-token');

      await register(req, res);

      expect(res.statusCode).toBe(201);
      const json = res._getJSONData();
      expect(json.token).toBe('fake-token');
      expect(json.message).toBe('Пользователь успешно зарегистрирован');
    });

    it('should not register user with existing email', async () => {
      const req = httpMocks.createRequest({ method: 'POST', body: { email:
'test@example.com' } });
      const res = httpMocks.createResponse();

      (UserModel.findUserByEmail as jest.Mock).mockResolvedValue({ id: 1 });

      await register(req, res);

      expect(res.statusCode).toBe(400);
      expect(res._getJSONData().message).toBe('Пользователь с таким email уже
существует');
    });
  });
});

describe('login', () => {
  it('should login user with correct credentials', async () => {
    const req = httpMocks.createRequest({
      method: 'POST',
      body: { email: 'test@example.com', password: 'password123' },
    });
  });
});

```

```

const res = httpMocks.createResponse();

(UserModel.findUserByEmail as jest.Mock).mockResolvedValue({ id: 1,
email: 'test@example.com', password: 'hashed', role: 'user' });
(bcrypt.compare as jest.Mock).mockResolvedValue(true);
(jwt.sign as jest.Mock).mockReturnValue('jwt-token');

await login(req, res);

expect(res.statusCode).toBe(200);
expect(res._getJSONData().token).toBe('jwt-token');
});

it('should reject invalid login', async () => {
const req = httpMocks.createRequest({
method: 'POST',
body: { email: 'test@example.com', password: 'wrongpass' },
});
const res = httpMocks.createResponse();

(UserModel.findUserByEmail as jest.Mock).mockResolvedValue({ id: 1,
password: 'hashed' });
(bcrypt.compare as jest.Mock).mockResolvedValue(false);

await login(req, res);

expect(res.statusCode).toBe(401);
expect(res._getJSONData().message).toBe('Неверный email или пароль');
});
});

describe('changePassword', () => {
it('should change password if current one is correct', async () => {
const req = httpMocks.createRequest({
method: 'POST',
body: {
email: 'test@example.com',
currentPassword: 'oldPass',
newPassword: 'newPass',
},
});
req.user = { id: 1 };

const res = httpMocks.createResponse();

(UserModel.findUserByEmail as jest.Mock).mockResolvedValue({ id: 1,
email: 'test@example.com', password: 'hashed' });
(bcrypt.compare as jest.Mock).mockResolvedValue(true);
(bcrypt.hash as jest.Mock).mockResolvedValue('newHashed');
const db = require('../config/db').default;
db.query = jest.fn().mockResolvedValue(true);

await changePassword(req, res);

```

```
expect(res.statusCode).toBe(200);  
expect(res._getJSONData().message).toBe('Пароль успешно изменен');  
});  
});  
});
```

Анализ результатов тестирования, включающего проверку различных сценариев использования, позволил убедиться в успешном завершении всех тестов.

Задание №9

Цель: описание процесса развертывания ПС.

Для корректной установки и работы веб-приложения необходимо наличие ряда компонентов. Система должна быть развернута на операционной системе семейства *MacOS* или *Windows*. В качестве веб-сервера используется *Express*, для функционирования серверной части требуются соответствующие зависимости и библиотеки. Клиентская часть приложения реализована с использованием *React.js* и также требует установки зависимостей. В качестве базы данных используется *PostgreSQL*, сервер которой должен быть установлен и настроен.

Процесс установки программного обеспечения состоит из нескольких этапов. В первую очередь необходимо предоставить соответствующие права пользователю, под которым будет производиться установка. Далее следует установка и настройка *Node.js*, после чего необходимо установить зависимости как для серверной, так и для клиентской части приложения. Затем устанавливается и настраивается сервер базы данных *PostgreSQL*. На заключительном этапе производится установка и конфигурация самого веб-приложения.

В комплект поставки конечному пользователю входят скрипт для создания пустой базы данных, а также ярлык или ссылка для доступа к приложению через веб-браузер.

Для проверки работоспособности системы необходимо убедиться в том, что запущен сервер базы данных *PostgreSQL*, а также функционируют серверная и клиентская части веб-приложения. После этого можно открыть любой современный браузер и перейти по адресу <http://localhost:3000>. В случае успешной настройки на экране отобразится главная страница веб-приложения.

Задание 10

Цель: разработка руководства пользователя.

Руководство пользователя предназначено для описания функциональных возможностей веб-приложения, доступных различным категориям пользователей, а также для обеспечения их эффективного взаимодействия с системой. Веб-приложение поддерживает два уровня доступа: пользователь (клиент) и администратор. Ниже представлены возможности каждого из них.

Функциональные возможности для клиента:

Пользователь (клиент) осуществляет вход в приложение через веб-интерфейс по предоставленной ссылке. После авторизации клиенту доступны следующие функции:

- Подача заявки на получение материнского капитала. Пользователь заполняет форму с необходимыми данными и отправляет заявку на рассмотрение.
- Просмотр своих заявок. Клиент может просматривать список всех поданных заявок, а также отслеживать статус их обработки (на рассмотрении, одобрена, отклонена).
- Просмотр и редактирование профиля. В личном кабинете отображается основная информация о пользователе, доступна возможность её изменения.
- Раздел «Информация». Содержит актуальные сведения, касающиеся условий и порядка получения материнского капитала. Информация структурирована и регулярно обновляется.
- Раздел «FAQ» (Часто задаваемые вопросы). Здесь собраны ответы на наиболее популярные вопросы, возникающие у пользователей. Также предоставлена возможность задать новый вопрос через соответствующую форму.
- Раздел «Консультации». В данном блоке клиент может оставить заявку на консультацию, указав тему и контактную информацию. После обработки запроса с ним свяжется специалист.

Все разделы интуитивно понятны, оформлены в едином стиле и направлены на обеспечение максимально удобного взаимодействия клиента с сервисом.

Функциональные возможности для администратора:

Администратор осуществляет вход в административную панель приложения с расширенными правами доступа. Ему доступны следующие функции:

- Управление заявками клиентов. Администратор может просматривать полный список всех поступивших заявок на получение материнского капитала, а также принимать решение об их одобрении или отклонении.
- Управление пользователями. В админ-панели реализованы функции просмотра, редактирования и удаления учетных записей клиентов.
- Редактирование информационных материалов. Администратор может добавлять, изменять и удалять тексты, размещаемые в разделе «Информация».
- Модерация блока *FAQ*. Доступна возможность добавления новых вопросов и ответов, а также редактирования или удаления существующих записей.
- Управление заявками на консультации. Администратор обрабатывает входящие запросы на консультацию, назначает ответственных специалистов и обеспечивает обратную связь с клиентами.

Таким образом, руководство пользователя предоставляет подробное описание всех ключевых действий, которые могут быть выполнены в системе, и служит основой для быстрого освоения функционала как для конечных пользователей, так и для администраторов.

Выводы

В ходе разработки программного средства для реализации онлайн-сервиса по использованию материнского капитала была создана система, которая значительно облегчает процесс подачи заявок на использование этого капитала, а также обеспечивает высокий уровень безопасности и надежности работы. Разработанная система включает в себя различные функции, такие как регистрация пользователей, авторизация, управление личными профилями и семейными данными, а также обработка заявок на использование материнского капитала.

Особое внимание было уделено безопасности, и для этого использовались технологии аутентификации и авторизации на основе *JWT* (*JSON Web Token*) и *bcrypt* для хеширования паролей. Это позволяет обеспечить надежную защиту данных пользователей и гарантирует, что доступ к системе получают только авторизованные лица.

Для реализации взаимодействия с пользователями был разработан API с использованием спецификации *OpenAPI* 3.0. Это позволило создать детальную и понятную документацию, обеспечивающую удобство в использовании и поддержке системы. API включает все необходимые эндпоинты для работы с профилями пользователей, их семейными данными, а также для подачи и обработки заявок.

В процессе разработки были проведены *unit*- и интеграционные тесты, которые подтверждают корректность работы всех ключевых функций системы. Тестирование охватило различные сценарии использования, включая регистрацию, вход в систему, управление данными пользователя, а также обработку заявок на материнский капитал. Все тесты прошли успешно, что подтвердило надежность системы.

Развертывание системы включает установку и настройку необходимых компонентов, таких как *Node.js*, *PostgreSQL*, а также соответствующих зависимостей, которые подробно описаны в документации. Пользователь может легко следовать инструкциям для корректной установки на различных операционных системах, что делает систему доступной для широкого круга пользователей.

Таким образом, результатом работы является функциональный и безопасный онлайн-сервис, который упрощает и автоматизирует процесс использования материнского капитала, а также предоставляет пользователям удобный и надежный интерфейс для работы с их личными данными и заявками.