# **Введение**

В современном динамичном обществе, пронизанном постоянными вызовами и угрозами, вопросы обеспечения безопасности становятся неотъемлемой составляющей жизни. В этом контексте ключевую роль играют частные охранные предприятия (ЧОП), которые несут на себе ответственность за обеспечение безопасности как для отдельных граждан, так и для различных объектов, включая предприятия, офисные здания, торговые центры, склады и многое другое. Подъем технологий и усиление конкуренции на рынке заставляют ЧОП постоянно совершенствовать свою деятельность и внедрять новые инструменты и подходы для эффективного решения возникающих задач безопасности.

Частное Охранное Предприятие (ЧОП) представляет собой организацию, специализирующуюся на предоставлении услуг в области безопасности и охраны. Такие предприятия осуществляют контроль и обеспечение безопасности на различных объектах, включая предприятия, офисы, торговые центры, а также мероприятия различного масштаба. Они обладают персоналом, состоящим из профессиональных сотрудников, ответственных за обеспечение порядка и безопасности клиентов.

Эффективность работы ЧОП во многом зависит от профессиональной подготовки сотрудников, их знаний, навыков и умений. В условиях постоянно меняющихся требований к безопасности и повышенных стандартов профессионализма, ЧОП сталкиваются с необходимостью эффективного мониторинга квалификации своего персонала и оценки их соответствия установленным стандартам. Одним из важнейших инструментов, обеспечивающих профессиональную подготовку сотрудников ЧОП, является система учета аттестации.

# **Актуальность**

На сегодняшний день по данным лицензионно-разрешительного управления МВД, в России в 26 тысячах ЧОП трудятся почти 2 миллиона зарегистрированных сотрудников. Вследствие чего возникает нужда во ведении электронного документооборота внутри частных охранных предприятий.

Система учета аттестаций сотрудников ЧОП становится все более актуальной в контексте растущих требований к безопасности и профессионализму в области охраны. Регулярная аттестация персонала не только позволяет контролировать и подтверждать их квалификацию, но и способствует повышению уровня безопасности предприятий и общественных мест. Кроме того, в условиях ужесточения законодательства в области безопасности, поддержание актуальной документации об аттестациях становится необходимым требованием для соблюдения правовых норм и предотвращения возможных штрафных санкций.

Информационная система учета аттестаций позволит автоматизировать рутинные задачи, такие как сбор и обработка данных, составление отчетов, уведомление сотрудников о сроках аттестации. Это освободит время сотрудников для более важных задач, повысит скорость и качество обработки информации, а также обеспечит ее достоверность. Кроме того, руководство будет иметь доступ к актуальной информации о состоянии аттестаций, что позволит принимать более обоснованные решения.

# **Описание**

Предметной областью данной курсовой работы является система учета аттестаций сотрудников, которая позволяет перевести бумажный документооборот по аттестациям сотрудников в электронный вид.

На регулярной основе через определенные промежутки времени каждому сотруднику возникает необходимость пере пройти аттестацию на ношение оружия, ношение спецсредств и прочее. Информационная система учета аттестаций ЧОП позволяет автоматизировать и цифровизировать документооборот учета аттестаций, проходить их в более удобном электронном формате, а также получать уведомления о необходимости перепрохождения аттестации.

Система учета аттестаций должна обладать следующими возможностями:

* Добавить сотрудника
* Добавить аттестацию
* Перепройти аттестацию
* Удалить сотрудника
* Удалить аттестацию

В данной системе существуют 2 роли:

* Оператор – он добавляет сотрудников, управляет информацией об аттестациях, подтверждает прохождение аттестации.
* Сотрудник – он проходит аттестации.

База данных располагается на сервере предприятия и хранит информацию о сотрудниках и о пройденных ими аттестациях.

Объектами автоматизации в данной системе являются процессы, связанные с аттестацией сотрудников ЧОП и управлением данными о их профессиональной подготовке. В первую очередь, это включает в себя информацию о каждом сотруднике, такую как его биографические данные, квалификация и иные характеристики, влияющие на процесс аттестации. Система также автоматизирует процессы планирования и проведения аттестаций, включая уведомления, распределение экспертов, и формирование отчетов. Кроме того, объектами автоматизации являются базы данных, хранящие результаты аттестаций, а также системы обеспечения безопасности данных, гарантирующие конфиденциальность информации. В целом, автоматизация данных процессов направлена на обеспечение эффективного управления кадрами и повышение уровня безопасности предоставляемых услуг ЧОП.

# **Диаграммы**

**Диаграмма классов**

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования. На моей диаграмме изображен класс "Сотрудник", у него есть методы: Пройти аттестацию и перепройти аттестацию, атрибуты ФИО, должность и квалификация. Также на диаграмме присутствует класс "База данных" с методами: Добавить запись, Удалить запись, Получить информацию. Класс "Аттестация" имеет атрибуты: Информация об аттестации, Дата аттестации. Класс "Оператор" содержит методы: Добавить и удалить сотрудника, добавить и удалить аттестацию, редактировать аттестацию, сформировать запрос на проведение аттестации, а также атрибут личные данные. И класс "Уведомление" обладает методом: Отправить уведомление, а также атрибутами Текст уведомления, Дата отправки и получатель. Класс "Десктоп клиент" имеет методы: Управление списком сотрудников, Добавление/редактирование информации об аттестациях, Загрузка/выгрузка документов, Добавление/редактирование информации о сотрудниках, Просмотр списка аттестаций. Класс "Мобильное приложение " имеет методы: Просмотр списка аттестаций и Просмотр информации об аттестациях.

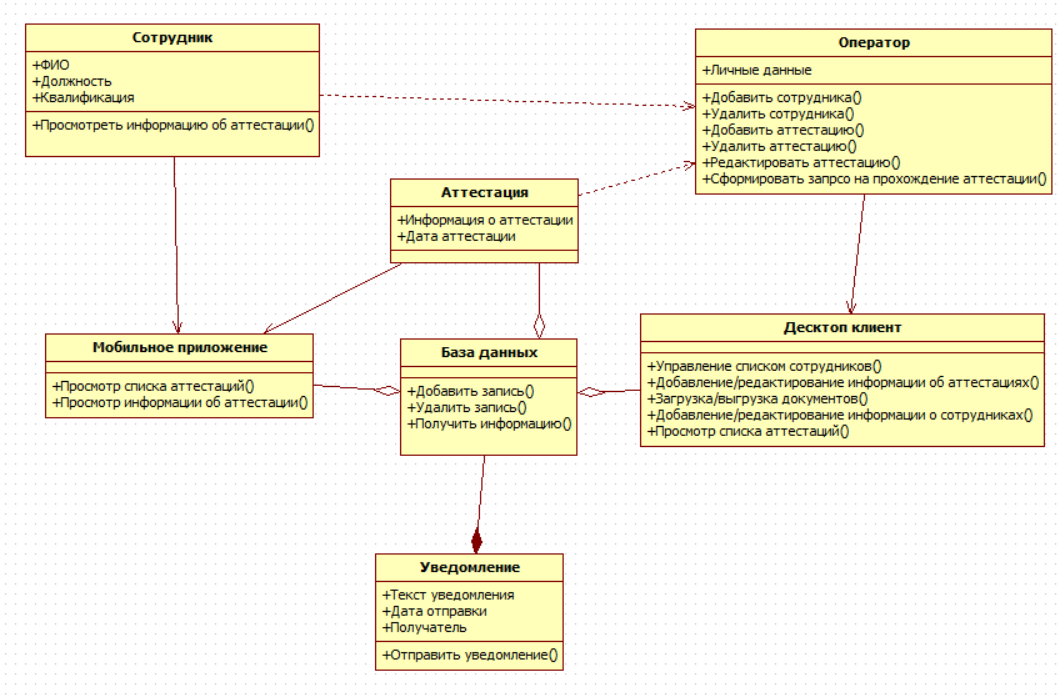
****

Рис.1 Диаграмма классов

**Диаграмма прецедентов**

Диаграмма прецедентов или диаграмма вариантов использования — диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

На моей диаграмме представлены 2 актора : оператор и сотрудник, сотрудник непосредственно проходит аттестацию, а оператор подает запрос на проведение аттестации, кроме этого он может добавить/удалить сотрудника или аттестацию.

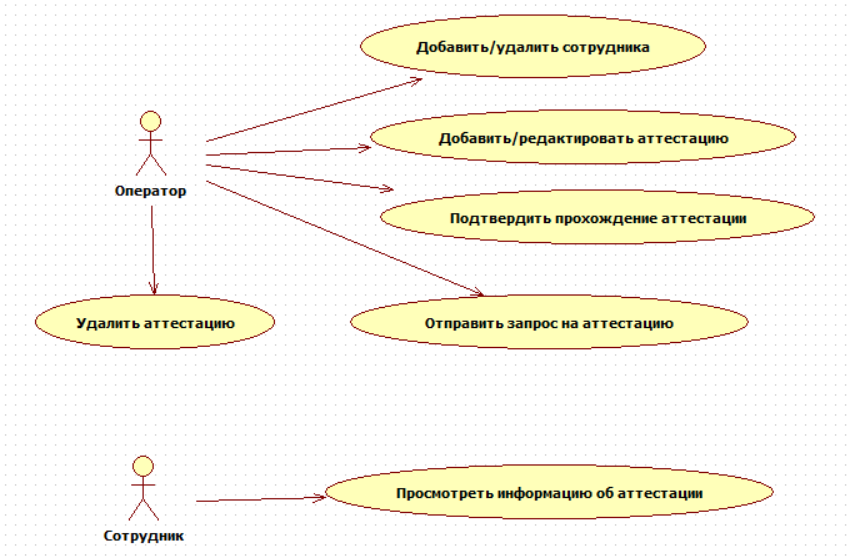
****

Рис.2 Диаграмма прецедентов

**Диаграмма состояний**

Она показывает, как объект переходит из одного состояния в другое. Диаграммы состояний служат для моделирования динамических аспектов системы. Данная диаграмма полезна при моделировании жизненного цикла объекта. От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события. На моей диаграмме представлено состояние сотрудника ЧОП, которые ожидает прохождения аттестации. После того как он прошел успешно аттестации процесс завершается, если же аттестация не подтверждена то он снова ожидает аттестацию.

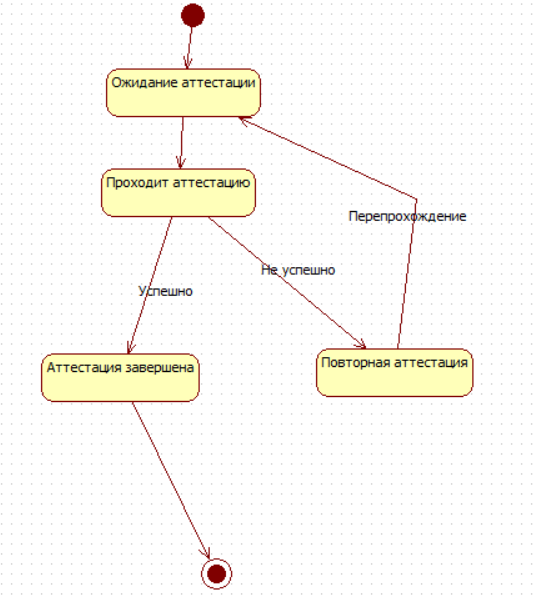
****

Рис.3 Диаграмма состояний

**Диаграмма последовательностей**

Диаграммы последовательностей используются для уточнения диаграмм прецедентов, более детального описания логики сценариев использования.

Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста.

Объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов).

Сообщения (вызовы методов) - линиями со стрелками.

Возвращаемые результаты - пунктирными линиями со стрелками.

Прямоугольники на вертикальных линиях под каждым из объектов показывают “время жизни” (фокус) объектов. Впрочем, довольно часто их не изображают на диаграмме, все это зависит от индивидуального стиля проектирования.

На моей диаграмме представлены 3 объектами: Оператором, Сотрудником и Системой учета аттестаций. Поток управления начинается с того что оператор добавляет сотрудника и аттестацию, затем в какой то момент сотрудник проходит аттестацию, затем данные в системе отправляются обновляются и затем формируется уведомление о том сколько действительна аттестация .

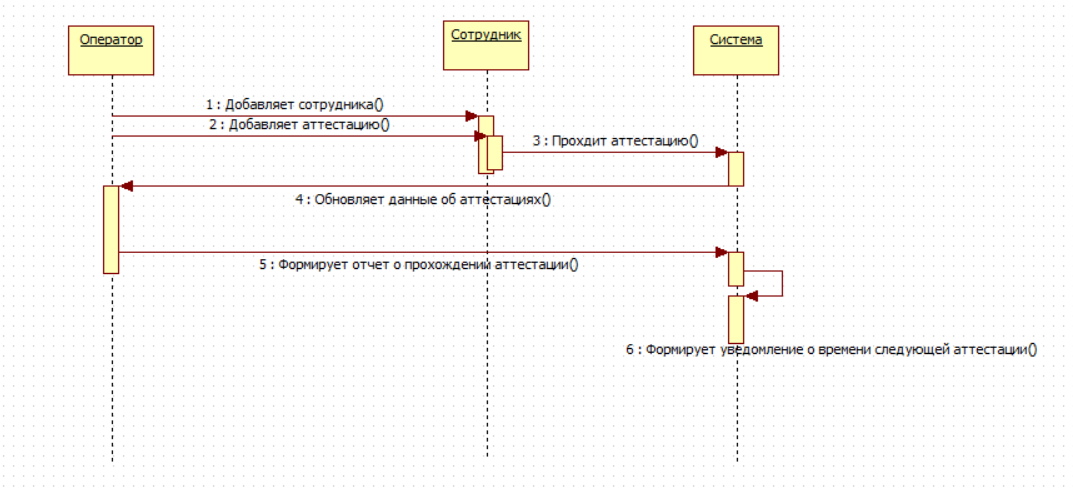
****

Рис.4 Диаграмма последовательностей

**Диаграмма активности**

Диаграмма активности UML позволяет более детально визуализировать конкретный случай использования. Это поведенческая диаграмма, которая иллюстрирует поток деятельности через систему.

Диаграммы активности UML также могут быть использованы для отображения потока событий в бизнес-процессе. Они могут быть использованы для изучения бизнес-процессов с целью определения их потока и требований.

На моей диаграмме представлены 3 инициатора процесса: Оператор, Сотрудник, и Система учета аттестаций. Поток управления начинается с того что оператор добавляет сотрудника, определяет какие типы аттестаций будет проходить данный сотрудник, в определенный момент когда подходит время формирует запрос на проведение аттестации, затем сотрудник проходит аттестацию, затем обновляет данные об аттестациях и формирует уведомление о том сколько действительна аттестация.

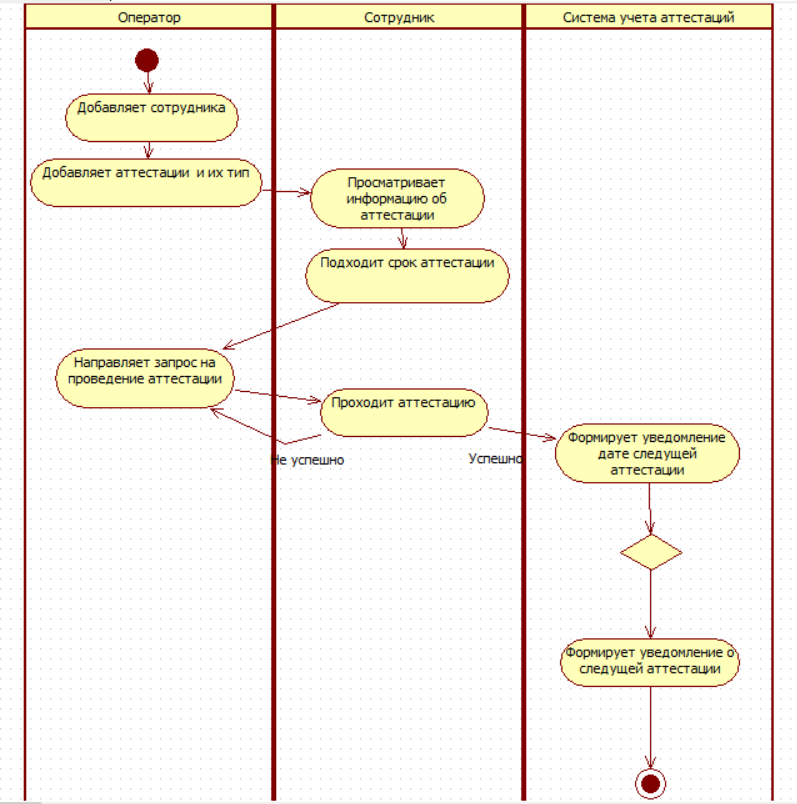
****

Рис.5 Диаграмма активности

**Диаграмма развертывания**

Диаграмма развертывания в UML используется для отображения физической структуры системы, показывая, как программное обеспечение, аппаратное обеспечение и сети взаимодействуют друг с другом. Она отображает размещение компонентов системы на физических устройствах и их взаимодействие через сети связи.

На диаграмме представлены: Сервер БД, где находятся СУБД, БД всех аттестаций, а также БД всех сотрудников и их данных. Управление осуществляется с ПК оператора, который связан с БД посредством сети Интернет. Сотрудники взаимодействуют с системой путем мобильного приложения, которое также сообщается с БД посредством сети Интернет.

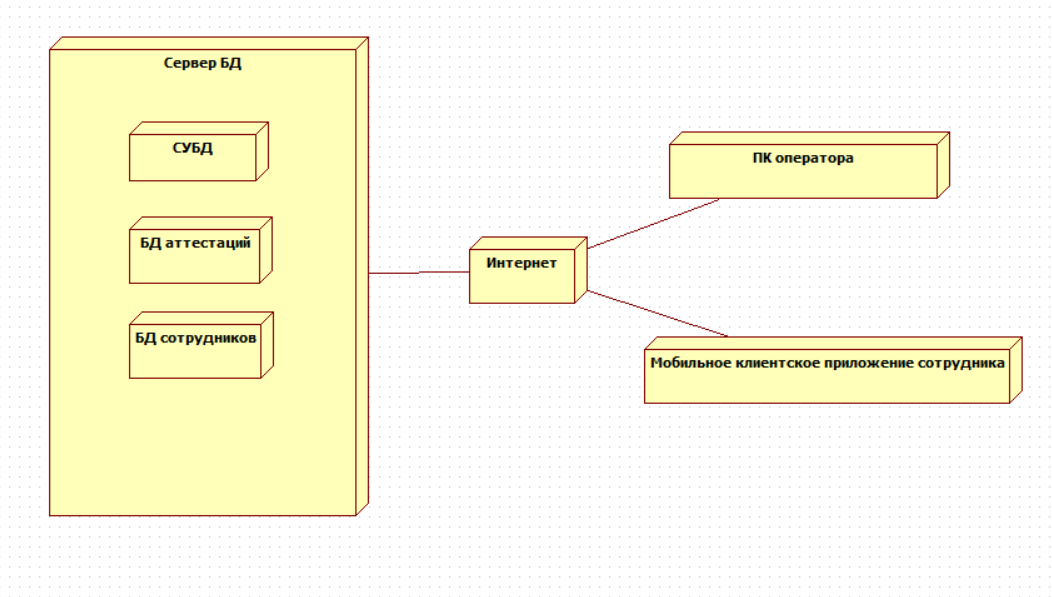
****

Рис.6 Диаграмма развертывания

**Диаграмма компонентов**

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы.

Компонентами могут быть программные компоненты, такие как база данных или пользовательский интерфейс; или аппаратные компоненты, такие как схема, микросхема или устройство; или бизнес-подразделение, такое как поставщик, платежная ведомость или доставка.

На моей диаграмме представлены: Клиентское приложение, хранящее в себе GUI- графический интерфейс сотрудника, а также уведомления об аттестациях, веб сервер, занимающийся обработкой запросов, сервер приложения осуществляющий логику управления сотрудниками и аттестациями, а также несущий интерфейс оператора, а также база данных, что аккумулирует себе средства управления базой данных и хранит все данные.

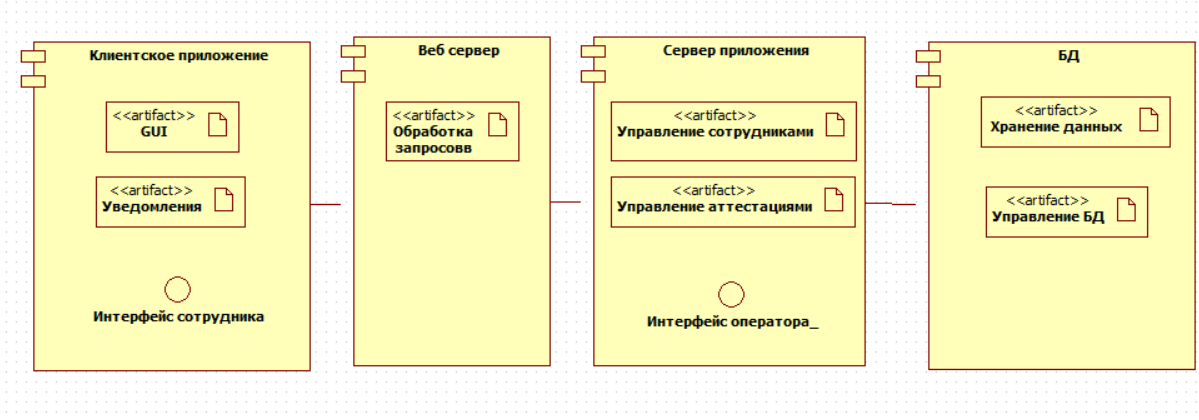
****

Рис.7 Диаграмма компонентов

## **Описание разрабатываемого продукта**

Презентационный слой мобильного приложения включает в себя все элементы пользовательского интерфейса, которые видит и с которыми взаимодействует пользователь. Это включает в себя различные экраны приложения, элементы управления, такие как кнопки и поля ввода, графические компоненты, такие как изображения и иконки, стилизацию текста и другие визуальные атрибуты, а также анимации и переходы между экранами.

В нашем мобильном приложении имеется 4 экрана:

1. Экран авторизации (LoginScreen) – он содержит изображение и текстовое приветствие, поля для ввода логина и пароля, а также кнопку «Войти» при нажатии на которую происходит попытка авторизации пользователя. При вводе правильного логина и пароля пользователь перенаправляется на главный экран (MainScreen). При неправильном вводе отображается диалоговое окно с сообщением об ошибке.

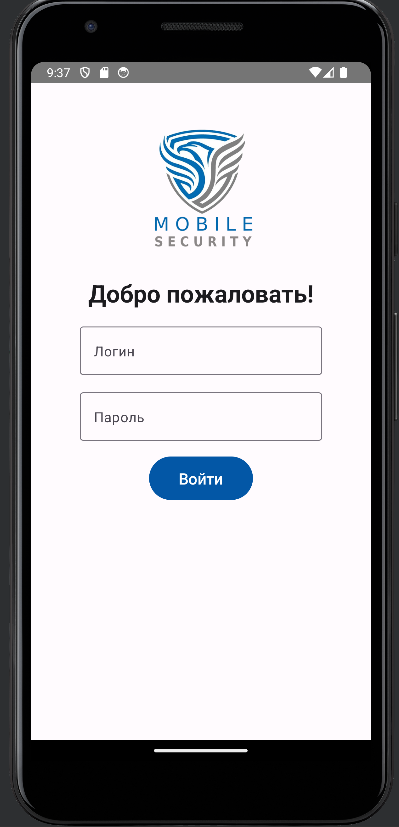


Рис.8 «LoginScreen»

1. Главный экран (MainScreen) – он содержит две кнопки – «Аттестация» и «Смена». При нажатии на кнопку «Аттестация» совершается переход на экран аттестации(AttestationScreen), при нажатии на кнопку «Смена» осуществляется переход на экран смена(ShiftScreen).

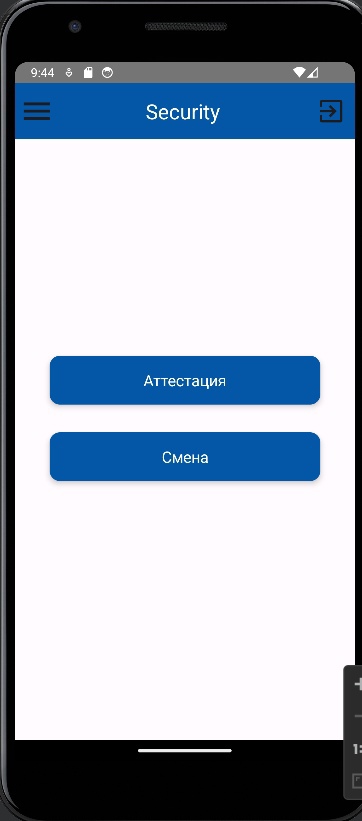


Рис.9 «MainScreen»

1. Экран аттестации (AttestationScreen) – он отображает список аттестаций пользователя в виде карточек с названием, датой прохождения и статусом. Пользователь может просматривать список аттестаций. По завершении просмотра можно вернуться на предыдущий экран с помощью кнопки "Назад".

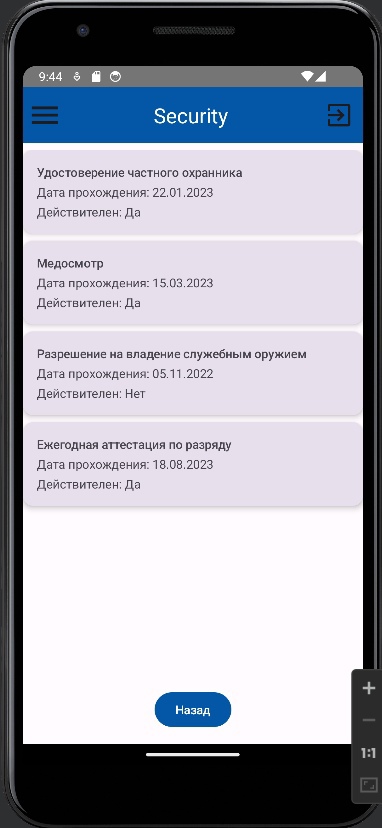


Рис.10 «AttestationScreen»

1. Экран смена (ShiftScreen) - Экран показывает информацию о текущей смене: объект, адрес, дата, время начала и окончания, а также предоставляет возможность начать или завершить смену. Также есть кнопка для возврата на главный экран.

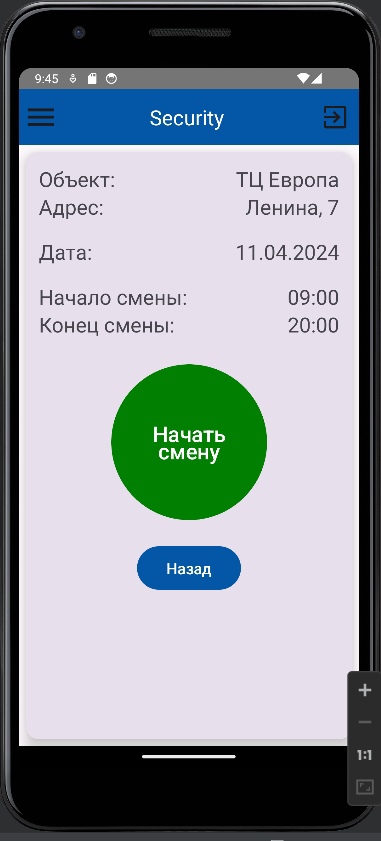


Рис.11 «ShiftScreen»

Кроме этого используются дополнительные компоненты в виде рисунков, надписей, иконок, элементов Material Design, а также элемент Header – он содержит кнопку перехода на главный экран, название приложения, а также кнопку выхода из системы.

Переход между экранами осуществляется с помощью компонента NavController. NavController - это компонент архитектуры навигации, предоставляемый Android Jetpack, который облегчает управление фрагментами и активностями в приложении. Он представляет собой централизованный объект, который обеспечивает связь между различными точками назначения в приложении и позволяет осуществлять переходы между ними.

Взаимодействие с NavController осуществляется через методы, такие как navigate() для перехода на новый фрагмент и popBackStack() для возврата к предыдущему состоянию приложения. В данной работе этот компонент является основным методом для перехода между экранами. Осуществляется это с помощью нажатий на кнопки и метода navigate. К примеру Button() {onClick = {navController.navigate(main screen)}}, после чего происходит переключение на главный экран. Пример использования popBackStack - Button() {onClick = {navController. popBackStack ()}}

Кроме этого используются дополнительные компоненты в виде рисунков, надписей, иконок, элементов Material Design, а также элемент Header – он содержит кнопку перехода на главный экран, название приложения, а также кнопку выхода из системы.

Для разработки приложения была выбрана платформа Android, как популярная и самая распространенная платформа и с учетом целевой аудитории. Язык программирования использовался Kotlin, как современный, безопасный и поддерживаемый Google язык.

Kotlin – высокоуровневый язык программирования общего применения, который является надстройкой над Java и разработан JetBrains – русской компанией разработчиков. Язык поддерживается компанией Google и используется в качестве основного языка для разработки приложений на Android в официальной SDK – Android Studio. Подобно Java, Kotlin – язык с сильной статической типизацией. Kotlin – объектно-ориентированный язык, при этом язык поддерживает широкий выбор возможностей для функционального программирования.

Язык обратно совместим с Java и выполняется в виртуальной среде Java – JVM. Также Kotlin поддерживает пакеты расширений Java. При этом, Kotlin обладает своими пакетами расширений, некоторая часть которых – доработанные пакеты Java. Kotlin и Java интероперабельны – их код может сосуществовать в одном приложении. При этом, Kotlin обладает более компактным и читаемым синтаксисом, а также рядом дополнительных функций, который нет в Java. Kotlin позволяет избежать некоторых ошибок, которые возникают в Java – например, NullPointerException.

Для повышения эффективности и удобства программирования на Kotlin, была использована технология Jetpack Compose. Фреймворк для UI, для интерактивного и реактивного интерфейса.

Jetpack Compose - это современный инструментарий для создания пользовательских интерфейсов в приложениях Android, разработанный компанией Google. Этот декларативный фреймворк позволяет разработчикам создавать динамические и интерактивные пользовательские интерфейсы, используя набор компонентов и функций Kotlin.

Основная идея Jetpack Compose заключается в том, чтобы позволить разработчикам описывать интерфейс приложения как функцию состояния, которая реагирует на изменения данных и автоматически обновляет пользовательский интерфейс при необходимости. Это обеспечивает простоту и четкость кода, а также повышает производительность приложения за счет улучшенного управления рендерингом и ресурсами.

Благодаря своей декларативной природе, Jetpack Compose сокращает объем кода, улучшает читаемость и поддерживает более эффективное тестирование пользовательского интерфейса. Это делает его мощным инструментом для разработки современных и привлекательных приложений Android.

Android Studio — интегрированная среда разработки производства Google, представляет собой мощный инструмент, специально адаптированный для работы с мобильными приложениями. Android Studio можно установить на Windows, Mac и Linux. Android Studio создавалась на базе IntelliJ IDEA. IDE можно загрузить и пользоваться бесплатно. В ней присутствуют макеты для создания UI, с чего обычно начинается работа над приложением. В Studio содержатся инструменты для разработки решений для смартфонов и планшетов.

**База данных**

База данных написана на программном обеспечении IntelliJ IDEA. IntelliJ IDEA (также известная как IntelliJ) — это интегрированная среда разработки (IDE) для программирования на различных языках, таких как Java, Kotlin, Groovy, Scala и других. Это продукт компании JetBrains, предназначенный для повышения производительности разработчиков и обеспечения удобной среды для создания программного обеспечения. IntelliJ IDEA предоставляет множество функций и инструментов для разработки приложений, включая подсветку синтаксиса, автодополнение кода, интегрированные системы сборки, отладчик, рефакторинг кода, управление зависимостями, поддержку контроля версий и многое другое. Одной из ключевых особенностей IntelliJ IDEA является его интеллектуальная поддержка, которая предлагает различные средства анализа кода, помогающие разработчикам создавать качественное программное обеспечение с меньшими ошибками. IDEA поддерживает различные плагины и расширения, что позволяет разработчикам интегрировать инструменты и языковые фреймворки, которые им необходимы для своего проекта. Это делает IntelliJ IDEA популярным выбором среди разработчиков Java и других языков, и она широко используется в индустрии разработки программного обеспечения.

Также для разработки используется фреймворк SpringBoot. Spring Boot - это фреймворк для создания приложений на языке Java. Он предоставляет удобные средства для быстрой разработки и запуска самодостаточных, упакованных по умолчанию приложений Spring. Spring Boot упрощает настройку и развертывание приложений, обеспечивая автоматическую конфигурацию большинства компонентов, что позволяет разработчикам фокусироваться на бизнес-логике приложения. Spring Boot также интегрируется с другими проектами Spring, такими как Spring Framework, Spring Data и Spring Security, обеспечивая широкие возможности для создания различных типов приложений, включая веб-приложения, микросервисы, приложения для обработки данных и другие.

Для решения проблемы миграции базы данных используется библиотека Flyway. Flyway - это инструмент для управления миграциями базы данных. Он позволяет автоматизировать и контролировать процесс изменения структуры базы данных в рамках разработки приложений. Flyway позволяет создавать и применять миграции базы данных, что обеспечивает управление версиями базы данных и автоматизацию процесса миграции при запуске приложения. Он также совместим с различными системами управления базами данных и может быть интегрирован с процессом непрерывной интеграции и поставки (CI/CD), что делает его полезным инструментом для разработчиков, работающих с базами данных в контексте разработки приложений.

**Сущности БД**

Сущность в базе данных является основным понятием, используемым для описания и хранения данных. Она представляет собой отдельный объект или предмет, о котором содержится информация и который имеет свои характеристики и атрибуты. В нашей базе данных присутствуют 2 сущности – UserEntity и AttestationEntity.

Сущность UserEntity представляет собой отображение объекта "пользователь" в базе данных в контексте Java Persistence API (JPA). Аннотация @Entity указывает, что класс является сущностью, которая будет отображаться в таблицу базы данных. Аннотация @Table (name = "user") указывает имя таблицы, к которой будет относиться данная сущность. Аннотация @Id указывает, что поле id является первичным ключом, а @GeneratedValue(strategy = GenerationType.IDENTITY) определяет способ последовательной генерации значений для этого ключа. Аннотация @OneToMany указывает на отношение "один ко многим" между сущностью UserEntity и сущностью AttestationEntity. Это означает, что одному пользователю может соответствовать несколько аттестаций. Геттеры и сеттеры используются для доступа к полям сущности из других частей приложения, а конструктор по умолчанию позволяет создавать объекты данной сущности без аргументов.

Поля класса:

* id: Первичный ключ сущности пользователя.
* login: Строковое поле, предназначенное для хранения логина пользователя.
* password: Строковое поле, используемое для хранения пароля пользователя.
* FIO: Строковое поле, представляющее ФИО (Фамилия, Имя, Отчество) пользователя.
* attestations: Список аттестаций, связанных с данным пользователем. Отношение "один ко многим" (One-to-Many) между пользователем и его аттестациями.

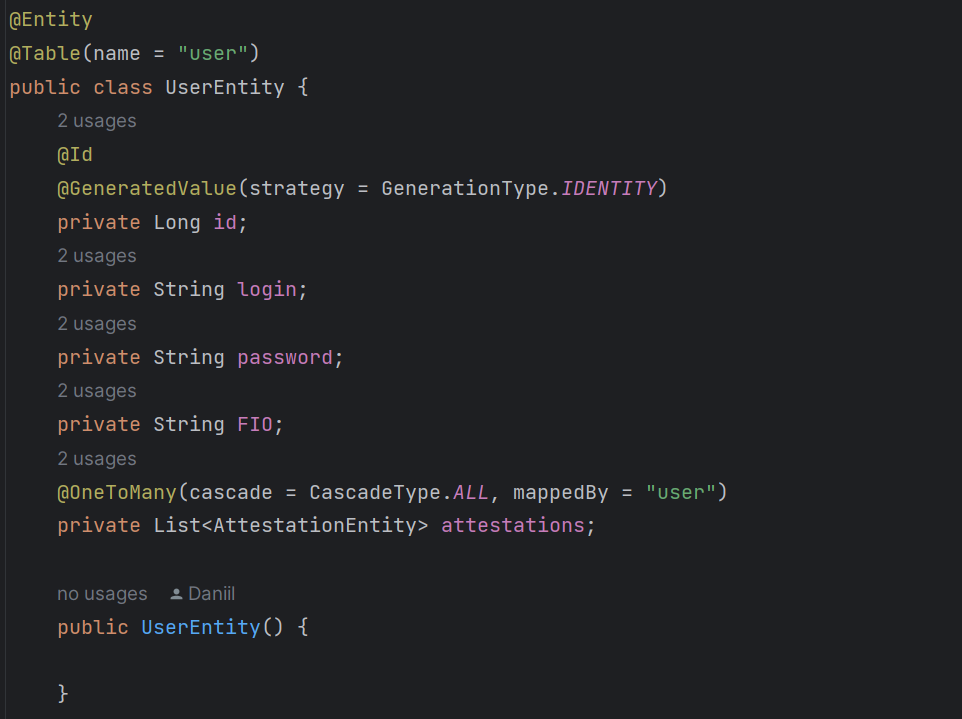


Рис.12 «Сущность UserEntity»

Сущность AttestationEntity представляет объект "аттестация" в базе данных в контексте Java Persistence API (JPA). Аннотация @Entity указывает, что класс является сущностью, которая будет отображаться в таблицу базы данных. Аннотация @Table (name = "attestation") указывает имя таблицы, к которой будет относиться данная сущность. Аннотация @Id указывает, что поле id является первичным ключом, а @GeneratedValue (strategy = GenerationType.IDENTITY) определяет способ генерации значений для этого ключа. Поле user является связью с сущностью UserEntity, которая представляет отношение "многие к одному" (Many-to-One) между AttestationEntity и UserEntity. Это означает, что нескольким аттестациям может соответствовать один пользователь. Геттеры и сеттеры используются для доступа к полям сущности из других частей приложения, а конструктор по умолчанию позволяет создавать объекты данной сущности без аргументов.

Поля класса:

* id: Поле id представляет собой первичный ключ сущности аттестации. Аннотация @Id указывает, что это поле является первичным ключом, а @GeneratedValue(strategy = GenerationType.IDENTITY) определяет способ генерации значений для этого ключа. В данном случае, используется генерация значений с помощью автоинкремента.
* title: Поле title представляет название аттестации. Оно отображается в столбце "title" в таблице базы данных.
* isValid: Поле isValid представляет состояние аттестации (действительна или нет). Тип данных Boolean указывает на возможные значения true или false.
* dataPassed: Поле dataPassed представляет дату прохождения аттестации. Оно отображается в соответствующем столбце в таблице базы данных.
* user: Поле user представляет связь с сущностью UserEntity и определяет отношение "многие к одному" (Many-to-One) между AttestationEntity и UserEntity. Оно связывается с полем "id" в сущности UserEntity посредством аннотации @JoinColumn(name = "user\_id").

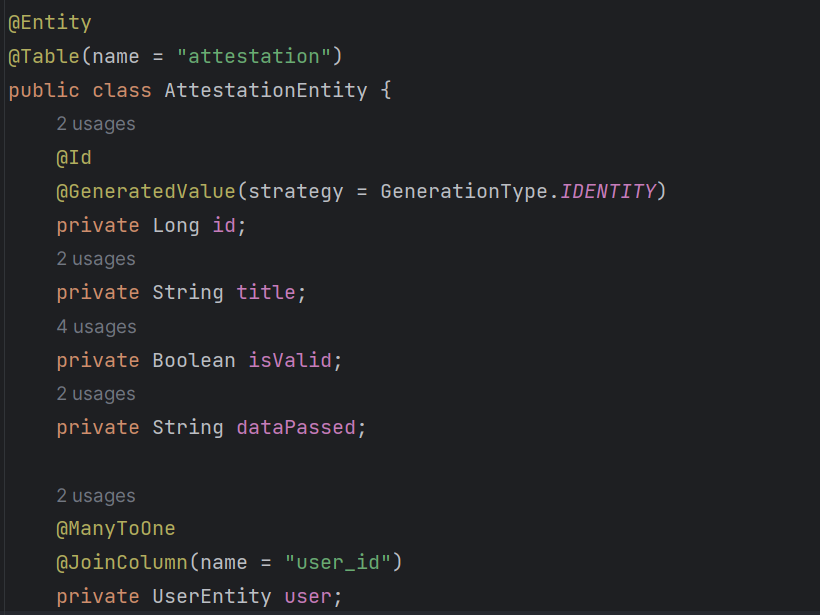


Рис.13 «Сущность AttestationEntity»

Для управления базой данных написаны методы REST API. В классе UserController описаны методы registration для добавления пользователя, getOneUser для получения пользователя по его id номеру, а также deleteUser для удаления пользователя по его id номеру.



Рис.14 «Методы класса UserController»

Для разделения бизнес логики и реализации все необходимые операции реализованы в классе UserService



Рис.15 «Методы класса UserService»

В классе AttestationController описаны методы createAttestation для создания аттестации и привязки к пользователю по его id, completeAttestation для обновления статуса аттестации, а также deleteAttestation для удаления аттестации.

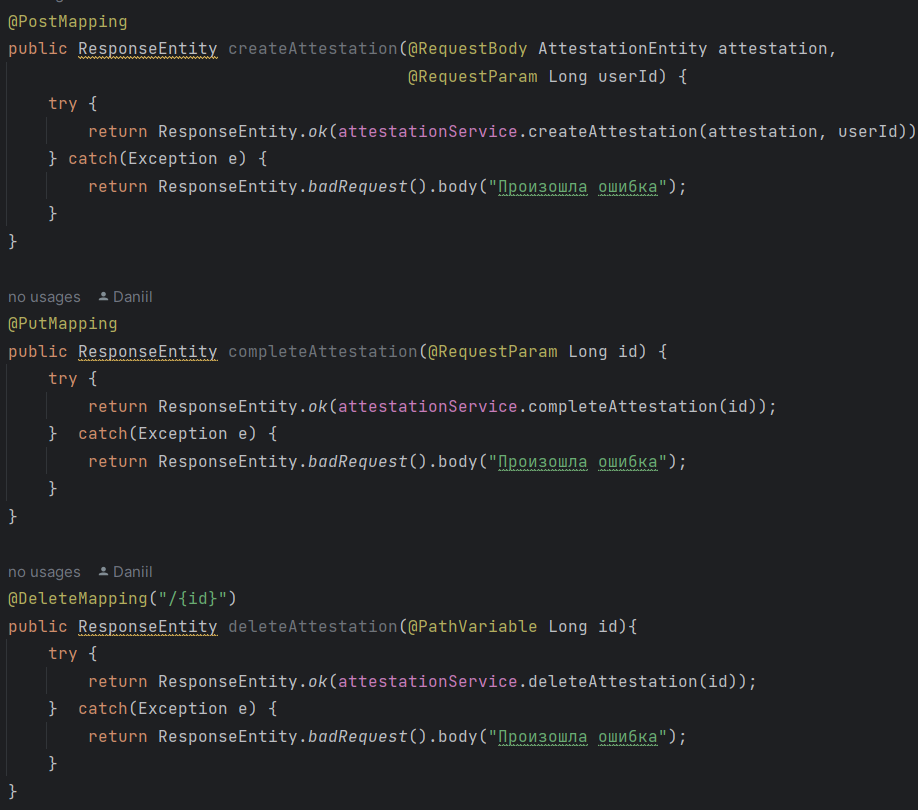


Рис.16 «Методы класса AttestationController»

Для разделения бизнес логики и реализации все необходимые операции реализованы в классе AttestationService

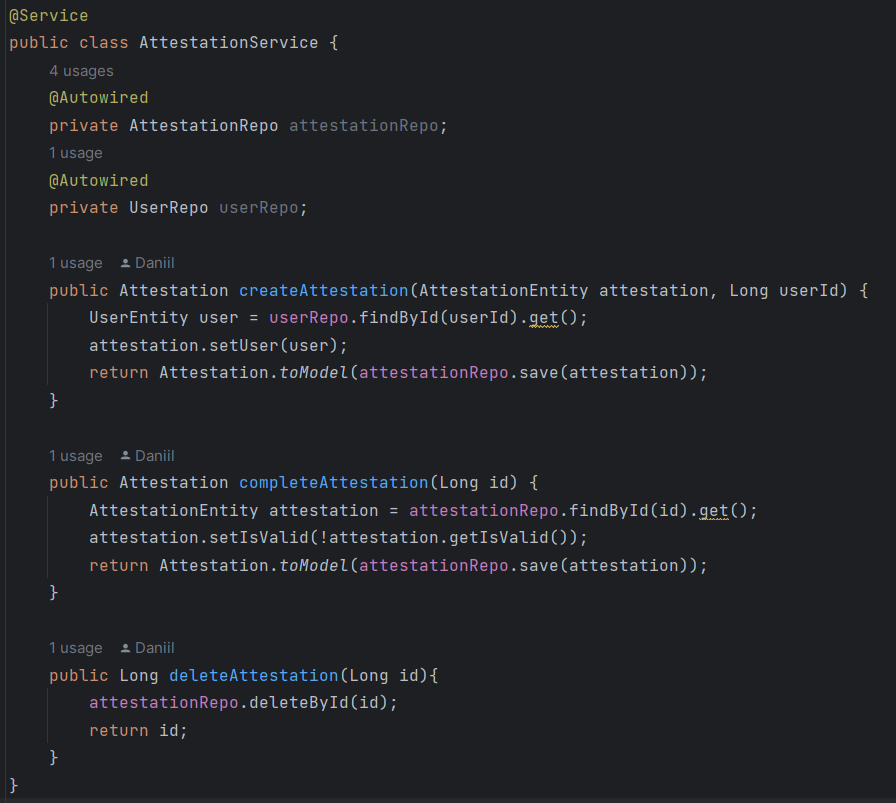


Рис.17 «Методы класса AttestationService»

Для решения проблемы миграции баз данных написан файл V1\_\_CREATE\_TABLES.sql

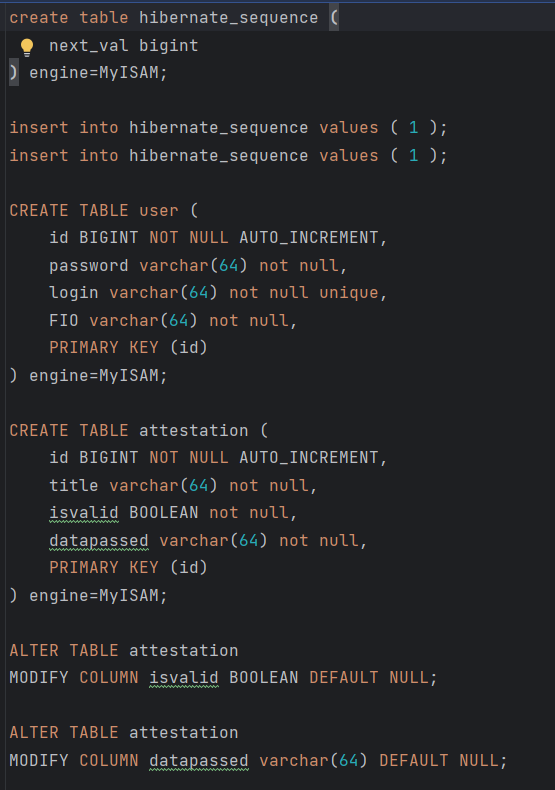


Рис.18 «Файл V1\_\_CREATE\_TABLES.sql»

**Тест-Кейсы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Название | Требование | Валидные данные | Результат ожидаемый | Не валидные данные | Результат не ожидаемый |
| 1 | логин | От 4 символов, латинские символы, без цифр | Ivanov  Smirnov | Пройдено | Иванов  Смирнов228  «» | Выбрасывается ошибка: неверный логин или пароль |
| 2 | пароль | От 4 символов, латинские символы | kmuinmd | Пройдено | \_\_\_\_\_\_\_\_  ------------  «» | Выбрасывается ошибка: неверный логин или пароль |
| 3 | авторизация | Введен верный логин и пароль |  | переход на экран Mainscreen |  | Переход на другое окно / ничего не произошло |
| 4 | Запрос на вывод 1 пользователя | Осуществлен GETзапрос на localhost:8080/users?id=1 |  | status code: 200 и в теле запроса  {      "id": 1,      "login": "test",      "fio": "Иванов Иван Иванович"  } |  | Вернулась ошибка 400 Bad request или любая другая ошибка |
| 5 | Запрос на добавление пользователя | Осуществлен POST запрос на localhost:8080/users | {      "login": "test2",      "password": "test2",      "fio": "Петров Иван Иванович"  } | status code: 200 и в теле запроса «Пользователь сохранен» |  | Вернулась ошибка 400 Bad request или любая другая ошибка |
| 6 | Запрос на удаление пользователя по id | Осуществлен запрос DELETE localhost:8080/users/2 для удаления пользователя с id = 2 |  | status code: 200 и id удаленного пользователя (2) |  | Вернулась ошибка 400 Bad request или любая другая ошибка |
| 7 | Запрос на создание аттестации | Осуществлен запрос POST localhost:8080/attestations?userId=3 | {      "title": "Удостоверение частного охранника",      "dataPassed": "22.01.2023",      "isValid": **true**  } | status code: 200 и в теле запроса  {      "id": 7,      "title": "Удостоверение частного охранника",      "isValid": **true**,      "dataPassed": "22.01.2023"  } |  | Вернулась ошибка 400 Bad request или любая другая ошибка |
| 8 | Запрос на обновление аттестации | Осуществлен запрос PUT localhost:8080/attestations?id=8 |  | status code: 200 и в теле запроса  {      "id": 8,      "title": "Удостоверение частного охранника",      "isValid": **false**,      "dataPassed": "22.01.2023"  } |  | Вернулась ошибка 400 Bad request или любая другая ошибка |
| 9 | Запрос на удаление аттестации | Осуществлен запрос DELETE localhost:8080/attestations/7 |  | status code: 200 и в теле запроса  7 – id удаленной аттестации |  | Вернулась ошибка 400 Bad request или любая другая ошибка |

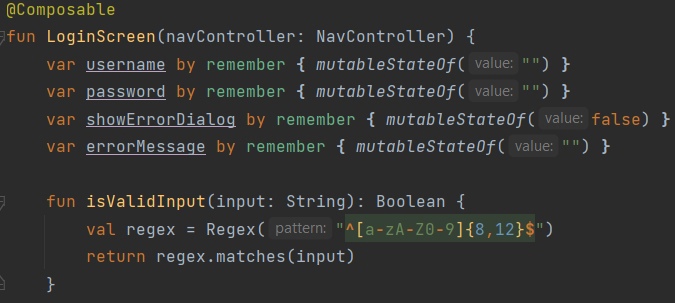


Рис.19 «Тест-кейс 1»

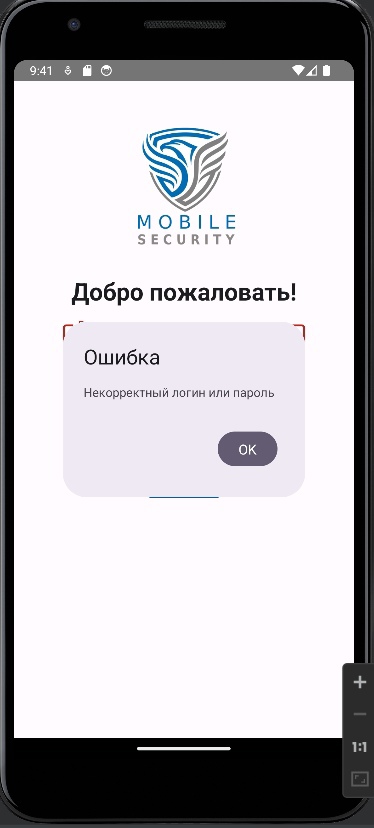


Рис.20 «Тест-кейс 2»

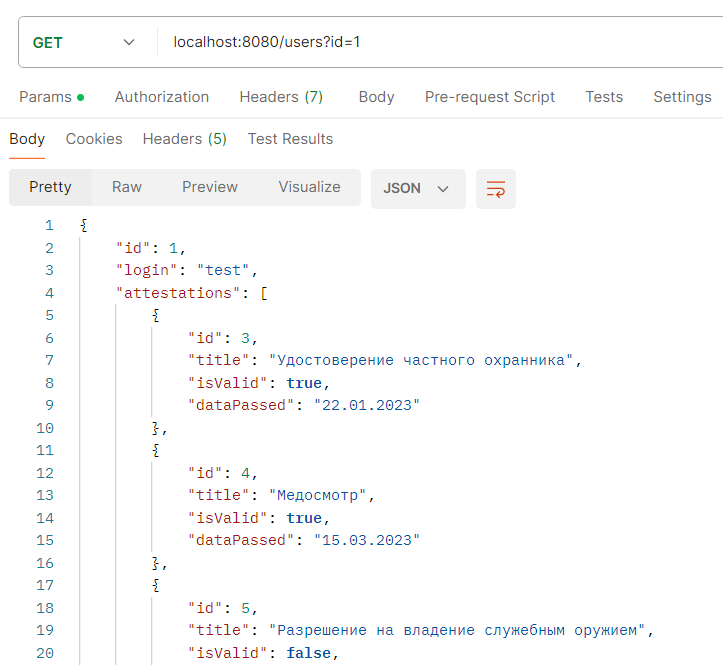


Рис.21 «Тест-кейс 4»

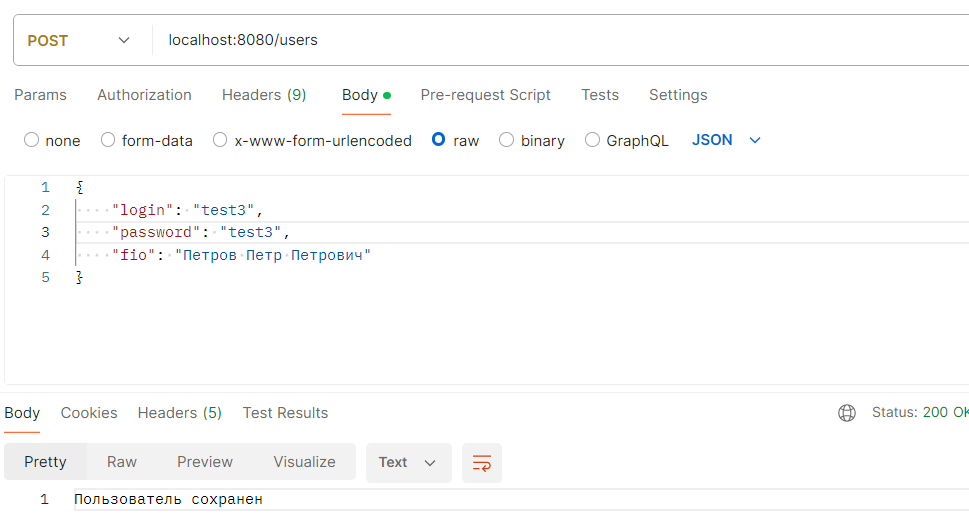


Рис.22 «Тест-кейс 5»

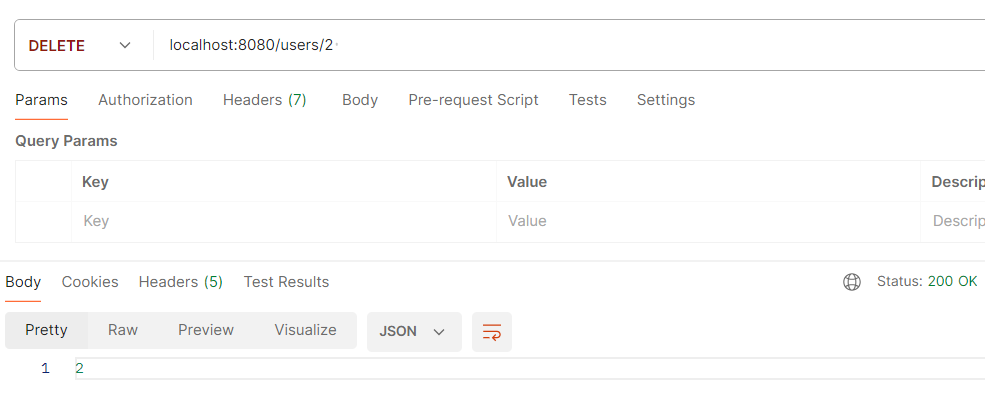


Рис.23 «Тест-кейс 6»

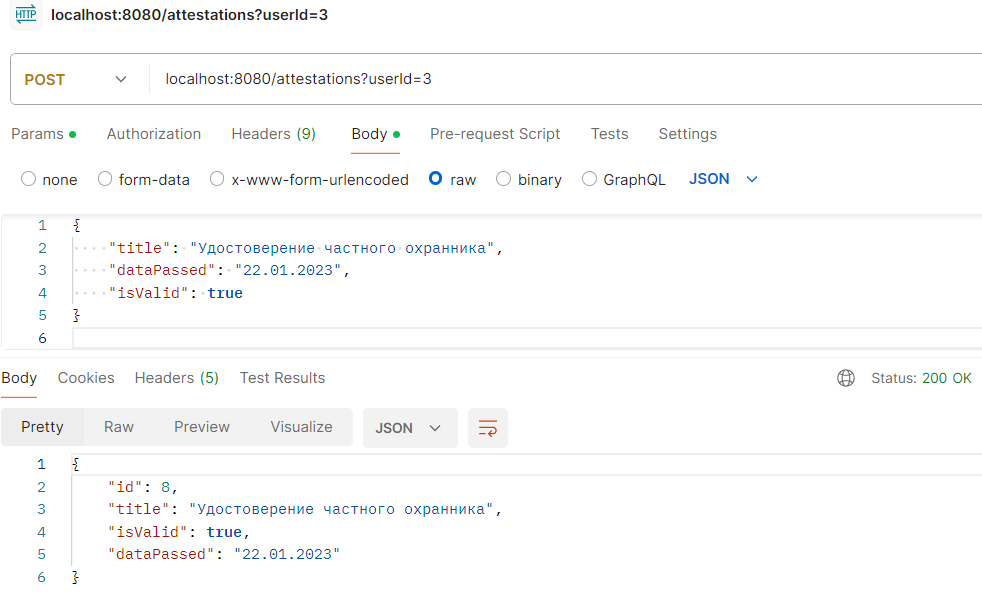


Рис.24 «Тест-кейс 7»

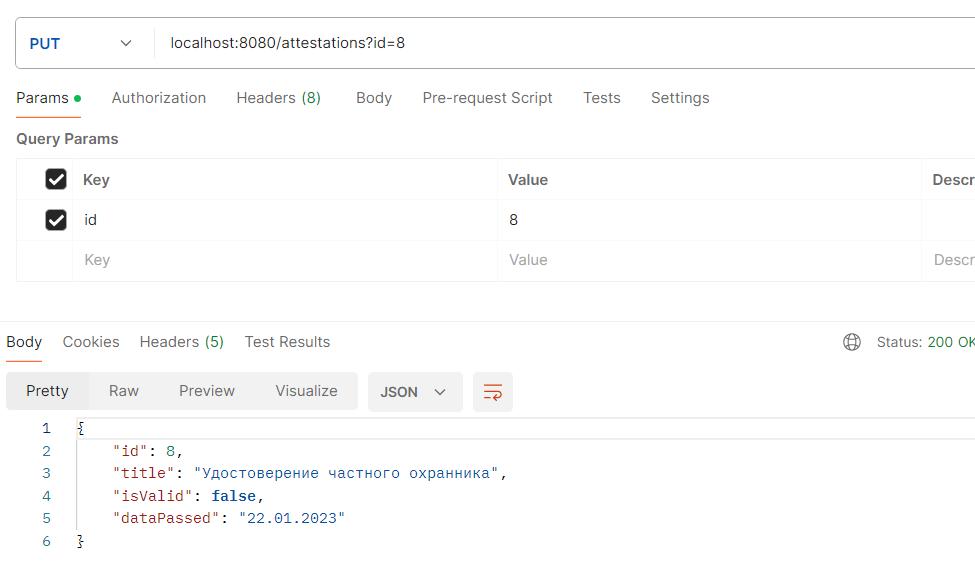


Рис.25 «Тест-кейс 8»

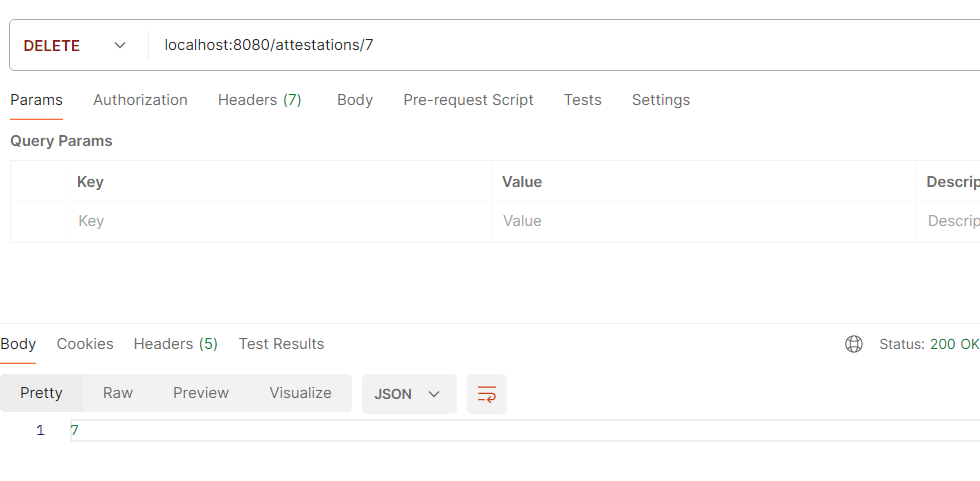


Рис.26 «Тест-кейс 9»

# **ЗАКЛЮЧЕНИЕ**

Таким образом, была разработана информационная система оперативного сопровождения ЧОП с учетом аттестаций. Кроме этого, был проведен анализ предмета автоматизации и учет проблем. Также была написана техническая документация по разрабатываемой системе.