Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«2D ВИДЕОИГРА ПО ЛОКАЛЬНОЙ СЕТИ

В ЖАНРЕ ШУТЕРА»

БГУИР КП 1-40 01 01 114 ПЗ

Студент: гр. 151001 Матюшенко В. А.

Руководитель: асс. Шамына А. Ю.

Минск 2023

**СОДЕРЖАНИЕ**

Введение 3

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4
   1. Анализ литературных источников, существующих решений и

необходимости разработки 4

1.2 Формирование требований к проектируемому программному

средству 5

2 Анализ требований к программному средству и разработка

функциональных требований 7

3 Проектирование программного средства 9

3.1 Проектирование взаимодействия серверной и клиентской частей

системы 9

3.2 Проектирование диаграмм Use Case, Activity и схемы базы данных 11

3.3 Основные положения, принятые при проектировании программного

средства 14

4 Конструирование программного средства 16

5 Тестирование, проверка работоспособности и анализ полученных

результатов 19

6 Руководство пользователя программы 21

Заключение 25

Список использованных источников 26

Приложение А. Исходный код программы 27

Приложение Б. Исходный код юнит-тестов 31

**ВВЕДЕНИЕ**

Ведение документации в электронном виде в настоящее время распространилось на многие сферы. Среди них и такие сферы, долгое время придерживавшиеся бумажного варианта документации: государственный аппарат, здравоохранение и пр. Исключением не стали и учреждения, связанные с общественным порядком: милиция, прокуратура и другие следственные учреждения.

С учётом приведённой выше информации, было решено попытаться создать системы каталогизации преступлений, по возможности максимально приближенную к реальным условиям. Такая система, помимо связанного хранения элементов, в перспективе также должна позволять выполнять анализ выбранных данных. При этом учитывался факт конфиденциальности сведений о системе хранения информации о преступлениях и структуре единиц хранения информации. Полученная система может использоваться частными сыскными организациями.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству. Исследование области лицензирования ПО, необходимости разработки и формирование требований.
2. Анализ требований к программному средству и разработка функциональных требований. На основе поставленных требований описывается функциональность программного средства и представляется укрупнённая схема алгоритма программного средства.
3. Проектирование программного средства. Определение механизмов реализации неблокирующего пользовательского интерфейса и основных положений, принимаемых при проектировании программного средства.
4. Конструирование программного средства. Описание модулей и компонентов программного средства, а также обоснование выбранных решений относительно функциональных требований.
5. Тестирование, проверка работоспособности и анализ полученных результатов. Характеристика действий, выполненных для проведения полного тестирования программного средства.
6. Руководство пользователя программы. Включает в себя последовательности действий, выполнение которых пользователем приведёт к успешному решению определённых задач.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**
   1. **Анализ литературных источников, существующих решений и необходимости разработки**

Существование единой системы-каталога для следственных органов является объективной необходимостью. Помимо систем, действующих в пределах одного, существует международная организация «Interpol», которая занимается вопросами межгосударственного регулирования поимки, поиска и задержания преступников [1].

В Республике Беларусь существуют следующие основные базы персональных данных в Беларуси (в скобках указаны органы, имеющие к ним доступ на постоянной основе) [2]:

* Регистр населения (МВД);
* Персонифицированный учет (Фонд соцзащиты населения);
* Кредитный регистр (Нацбанк);
* База данных о гражданах, право на выезд которых временно ограничено (МВД);
* База данных абонентов сотовой связи (провайдеры услуг);
* Автоматическая информационная система (Расчет) / ЕРИП (Нацбанк);
* Единый государственный банк данных о правонарушениях (МВД);
* База данных дактилоскопической регистрации (МВД).

Как видно из приведённой выше информации, существуют отдельные база данных правонарушений и отпечатков. Получить данные из базы данных о правонарушениях МВД могут все желающие физические лица и организации при наличии согласия самого гражданина. Без согласия лица в эту базу могут обратиться органы уголовного преследования и органы, ведущие административный процесс. Схож и порядок доступа и к базе данных дактилоскопической регистрации. Сюда могут без всяких ограничений обратиться органы уголовного преследования, суды, органы, осуществляющие оперативно-разыскную деятельность, и отделы гражданства и миграции при МВД.

Поэтому необходимо заметить, что данные системы являются закрытыми, доступ к ним закрыт для лиц, не обладающими необходимыми правами доступа, кроме определённых случаев. По этой причине анализ данных систем на предмет анализа организации, поиска достоинств и недостатков не представляется возможным.

Литературные источники также не содержат детальной информации об организации данных систем, в силу описанных выше причин. Несмотря на это, научные работы в данном направлении должны проводиться для улучшения раскрываемости преступлений и качества функционирования данных систем.

Однако сам факт существования таких систем, пусть и доступных для определённого круга лиц, говорит об актуальности проблемы и активном использовании на практике. Помимо государственных сыскных организаций существуют также частные организации схожего профиля. Наряду с использованием единой государственной системы такие фирмы должны иметь собственную систему для хранения информации о своей сыскной деятельности. Наличие такой системы в будущем может позволять проводить автоматизированный анализ преступлений с заданными параметрами, что может облегчить поиск виновных в совершении правонарушения.

* 1. **Формирование требований к проектируемому программному средству**

На основании анализа вышеприведённой информации выделена главная цель программного средства – ведение каталога преступлений и связанных с ними объектов и предметов.

Ведение документации для преступлений должно быть максимально приближено к реально существующим условиям. В связи с закрытостью данной информации, придерживаться организации преступлений по делам. Необходимо хранить информацию по уликам и участникам преступлений. При этом необходимо учесть, что как человек может быть связан с несколькими преступлениями, так и улика может фигурировать в нескольких правонарушениях.

Система должна быть доступна только для авторизованных пользователей, которыми являются сотрудники фирмы. По умолчанию, все регистрируемые в базе данных сотрудники должны обладать правами детектива, т.е. могут быть указаны как лицо, расследующее дело. Каждый пользователь должен при регистрации указывать электронный адрес для возможности информировании его об определённых событиях.

Необходимо предусмотреть наличие статусов для дела («открыто» – расследуется в текущий момент; «закрыто» – расследование дела прекращено в связи с наказанием виновных; «не раскрыто» – дело перемещено в архив в связи с отсутствием продвижения в деле в течении определённого периода). Также необходимо указывать тип для преступления и улики, статус для участника преступления, а также предусмотреть возможность добавления новых статусов и типов для всех трёх сущностей.

Удаление информации в системе должно быть недоступно, для сохранности любой информации, которая может быть связана с преступлениями. Ошибочное или случайное стирание информации может привести к потере потенциально важной информации. В тоже время необходимо предусмотреть какой-то тип выделения информации, предположительно, не актуальной или не связанной с ранее указанной сущностью.

Система должна автоматически генерировать документацию по указанным параметрам. Генерация должна осуществляться в форматы pdf, xlsx, csv. Отчёты должны генерироваться: 1) по конкретному преступлению; 2) по конкретному делу; 3) по конкретной улике; 4) по конкретному человеку; 5) по конкретному детективу; 6) по делам, имеющим указанный статус; 7) по преступлениям, совершённым в указанный временной период.

В системе должна быть реализована автоматическая отправка электронных писем при совершении определённого действия или по истечении определённого периода времени. В частности, письмо должно отправляться всем детективам при регистрации в системе нового детектива; также в день рождения пользователя ему должно приходить на электронную почту письмо с поздравлением.

Программное средство должно быть Web-приложением, которое должно открываться в любом браузере, выпущенном после 2015 г., по заранее определённому адресу. Пользовательский интерфейс должен быть интуитивно понятным и реализован в спокойной цветовой гамме. Приложение должно быть доступно с любого устройства, подключённого к сети Интернет, без установки дополнительного программного обеспечения.

Система должна быть расширяемой, т.е. позволять добавлять новые компоненты без изменения существующих. К примеру, должна быть заложена возможность добавления статуса для пользователей, зарегистрированных в системе. Первоначально данная особенность не является необходимой, но необходимо предусмотреть её на случай расширения фирмы-заказчика и, соответственно, требований к используемому программному обеспечению.

Учитывая все вышеизложенные сведения, приложение должно быть простым и понятным в использовании.

**2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

Исходя из поставленной задачи и сформулированных требований, разрабатываемый программный продукт должен выполнять следующие действия:

* предоставлять различный уровень функциональных возможностей для авторизованного и неавторизованного пользователей;
* предоставлять необходимую информацию после обращения по определённому доменному имени в сети Интернет;
* вести каталог преступлений, объединённых в дела. Для каталога также были выделены следующие сущности: улика, участник преступления, детектив. Дело и участник преступления должны иметь статус, а улика и преступление – тип. Типы и статусы (кроме статуса для дела) должны браться из определённого списка, который может изменяться;
* позволять добавлять и изменять записи в каталоге;
* возвращать все записи из каталога, соответствующие определённому условию;
* не позволять выполнять удаление данных ни при каких условиях. Указание неактуальности или некорректности информации должно осуществляться путём установки специального типа или статуса там, где это указано;
* отправлять электронные письма при выполнении определённых действий или по таймеру при указанных условиях;
* автоматически генерировать документы по хранимым в каталоге данным. Генерация должна осуществляться в форматы pdf, xlsx, csv. Каждый документ должен передаваться на клиент в виде потока с корректным MIME-типом. Документы в формате PDF должны быть оформлены на "фирменном бланке" фирмы. Предусмотреть возможность создания защищенных PDF-документов, в которых заблокирована возможность копирования текста. Документы Excel должны быть готовыми для печати. Это значит, что выводимый текст не должен быть обрезан внутри колонки, а при печати таблицы должны умещаться на одной странице по ширине. Документы в формате CSV должны использовать запятую (,) или точку с запятой (;) в качестве разделителя. Выбранная для хранения кодировка текста должна позволять просмотреть сохраненный документ в приложении Microsoft Excel без дополнительных настроек;
* клиентская и серверная часть приложения должны быть разделены. Взаимодействия клиента с сервером должно осуществляться путём отправки HTTP-запросов заранее определённого формата на определённый адрес сервера. Сервер в случае успешного выполнения операции должен возвращать HTTP-код возврата, равный 200; в противном случае должен быть указан код ошибки, поясняющий невозможность совершения действия;
* сервер должен сохранять в лог выполняемые действия для возможности последующей поддержки другими разработчиками.

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Проектирование взаимодействия серверной и клиентской частей системы**

Одним из выдвинутых пользовательских требований является работа веб-приложения в любом браузере. Это накладывает определённые условия на внутреннюю организацию приложения. Веб-приложение выдвигает определённые требования к графическому интерфейсу приложения.

Совмещение разработки графического интерфейса для веб-приложения с реализацией внутренней логикой работы в настоящее время достаточно редко в силу многих причин. Одной из причин является некоторая спецификация каждого языка программирования для решения своего круга задач. Часть языков используется исключительно для создания интерфейса веб-приложения, т.е. front-end’a. Они предлагают более разнообразный и удобный набор инструментов для разработки. В тоже время, разработка логики и интерфейса веб-приложения совместно ограничивает возможности разработчика, в силу меньших возможностей. Также не следует забывать об удобстве расширения и поддержки. В случае независимости логики (back-end) и интерфейса (front-end) удобство повышается.

В силу указанных выше причин было решено разделить разработку логики и интерфейса веб-приложения. Однако тогда встаёт вопрос о взаимодействии между этими двумя частями, по сути, одного приложения. Сейчас очень популярным является принцип взаимодействия REST. Ранее сервер возвращал клиенту разметку вместе с данными, которую необходимо отобразить пользователю. Однако в данном случае серверу не нужно ничего знать о разметке, поэтому сервер может возвращать только нужные данные. Также в данном случае не нужно полного соответствия между страницей, которую надо отобразить пользователю, и запросом к серверу. Именно REST-подход и был выбран для данного приложения.

В качестве концепции клиентской части было выбрано одностраничное приложение (SPA). Самая заметная разница между обычным сайтом и SPA – это уменьшение количества обновлений страницы. В одностраничном приложении в гораздо большей степени используется AJAX – способ общения с серверными серверами без полного обновления страницы – для загрузки данных в наше приложение. В результате процесс рендеринга (построения) страниц происходит в основном на стороне клиента с помощью JavaScript [3].

Форматом данных, используемым для коммуникации при REST-подходе, был выбран формат json. Данный формат, по сравнению с также распространённым форматом xml, является более информативным (за счёт более сжатого формата записи данных), что уменьшает количество передаваемых данных.

**3.2 Проектирование диаграмм Use Case, Activity и схемы базы данных**

В рамках проектирования программного средства было выполнено создание Use Case для определения лиц, участвующих в использовании приложения, и действий, которые ими могут быть выполнены. Результат проектирования отображён на рисунке 3.1.

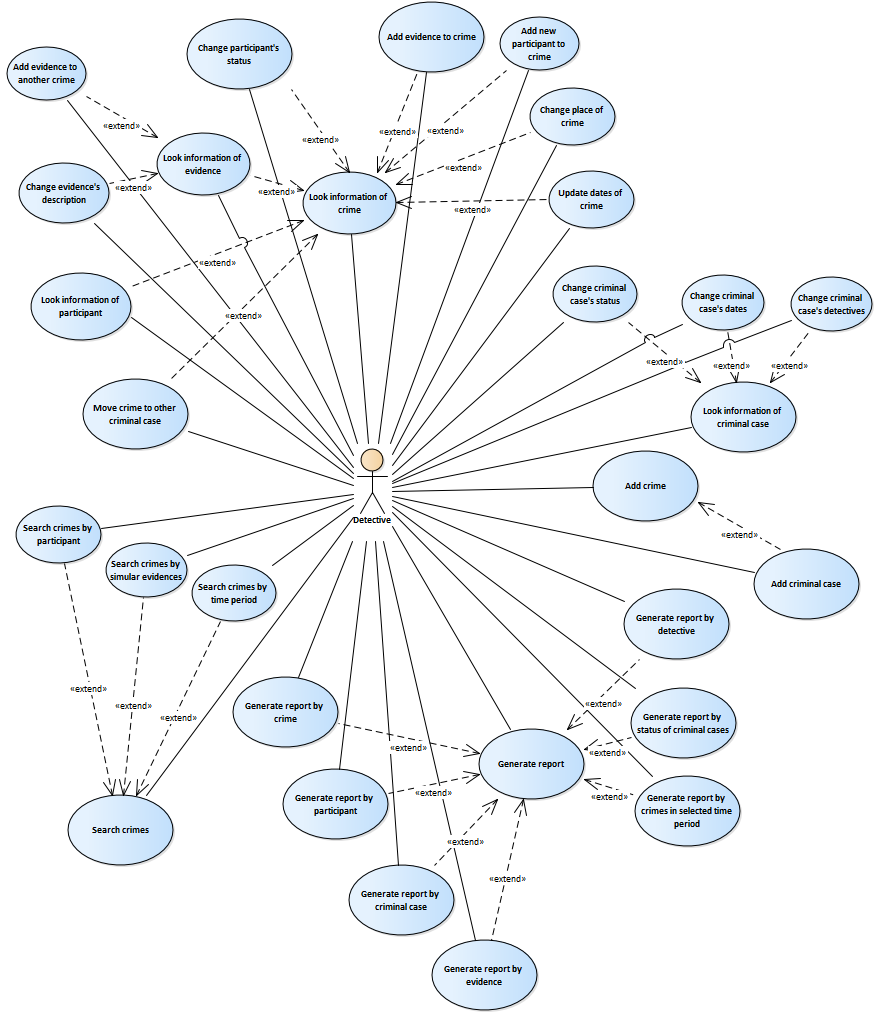


Рисунок 3.1 – Диаграмма Use Case

Для определения общего механизма действия системы была построена диаграмма Activity для всего программного средства; диаграмма представлена на рисунке 3.2.

Для конкретизации отдельных алгоритмов были построены диаграммы Activity для таких пользовательских действий как добавление нового преступления (рисунок 3.3) и добавление нового участника преступления (рисунок 3.4).

Для всех трёх диаграмм Activity характерно разделение участвующих сторон на детектива (реального человека), систему (подразумевается абстракция, объединяющая клиентскую и серверную части) и базу данных.

Данные диаграммы помогли определить направление дальнейшей разработки приложения, несмотря на последующее изменение отдельных фрагментов алгоритмов.

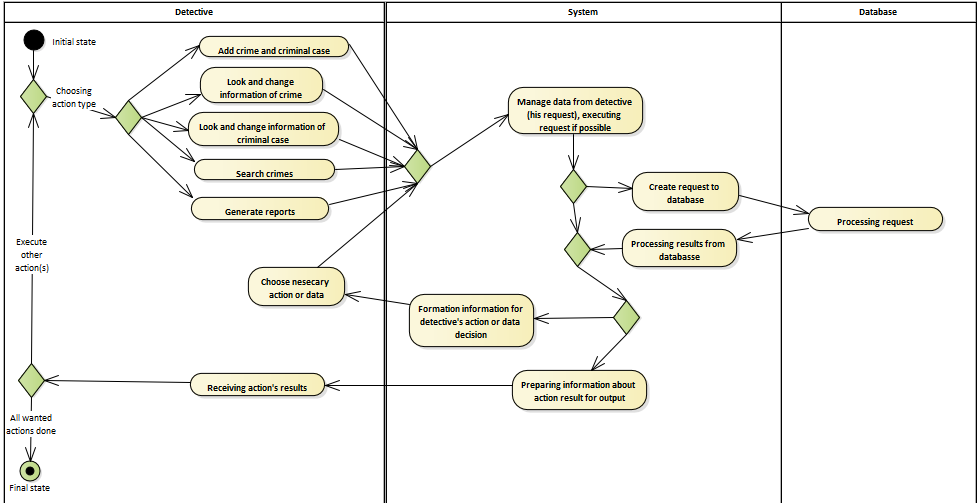


Рисунок 3.2 – Диаграмма Activity для всего приложения

Было выполнено построение структуры базы данных для данной системы, в соответствии с поставленными требованиями на заданном этапе, которая представлена на рисунке 3.5.

В качестве системы управления базой данных (СУБД) была выбраны системы «MySQL». Данная система является довольно распространённой, поддерживаемой, свободно распространяемой и обладающая достаточным набором поддерживаемых типов данных и операций, при этом демонстрирующей неплохие скоростные показатели.

Ограничение типов и статусов для преступления, улики и участника было решено хранить в виде строки, а не отдельной сущности, поскольку было решено ограничивать диапазон значений на уровне серверной части приложения, а не базы данных.

В результате более тщательной разработки возможных статусов уголовного дела было выявлено, что в принципе можно выделить не более трёх вариантов: «открыто» (расследуется в данный момент), «закрыто» (раскрыто) и «не раскрыто» (виновный не найден, но срок расследования не продлён и дело перемещено в список архивных). Поэтому было решено на уровне базы данных решить необходимость статуса с помощью с поля типа boolean и даты закрытия дела. При статусе «открыто» boolean-поле установлено в значение false; при статусе «закрыто» boolean-поле установлено в значение true и указана дата закрытия; при статусе «открыто» boolean-поле установлено в значение true, но отсутствует дата закрытия.

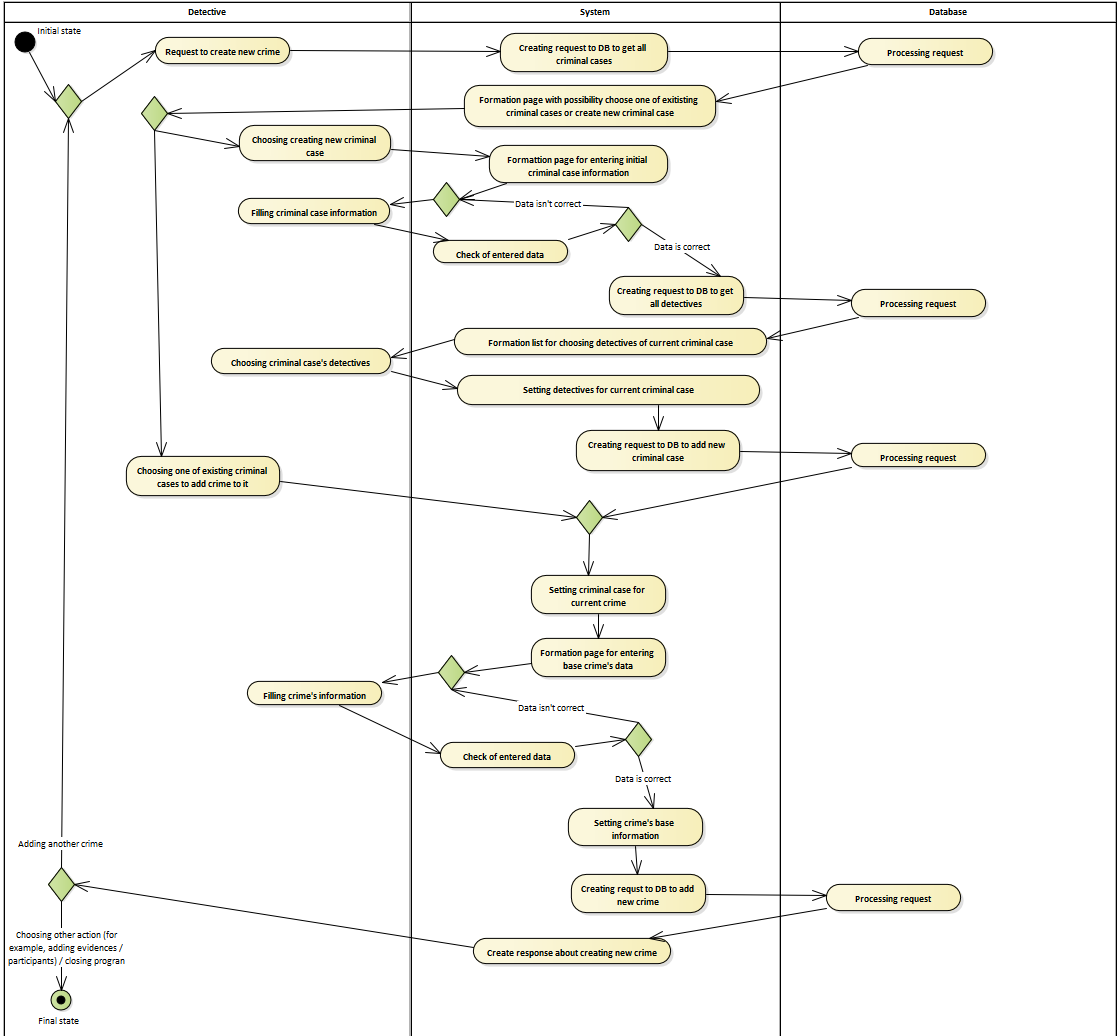


Рисунок 3.3 – Диаграмма Activity для добавления преступления

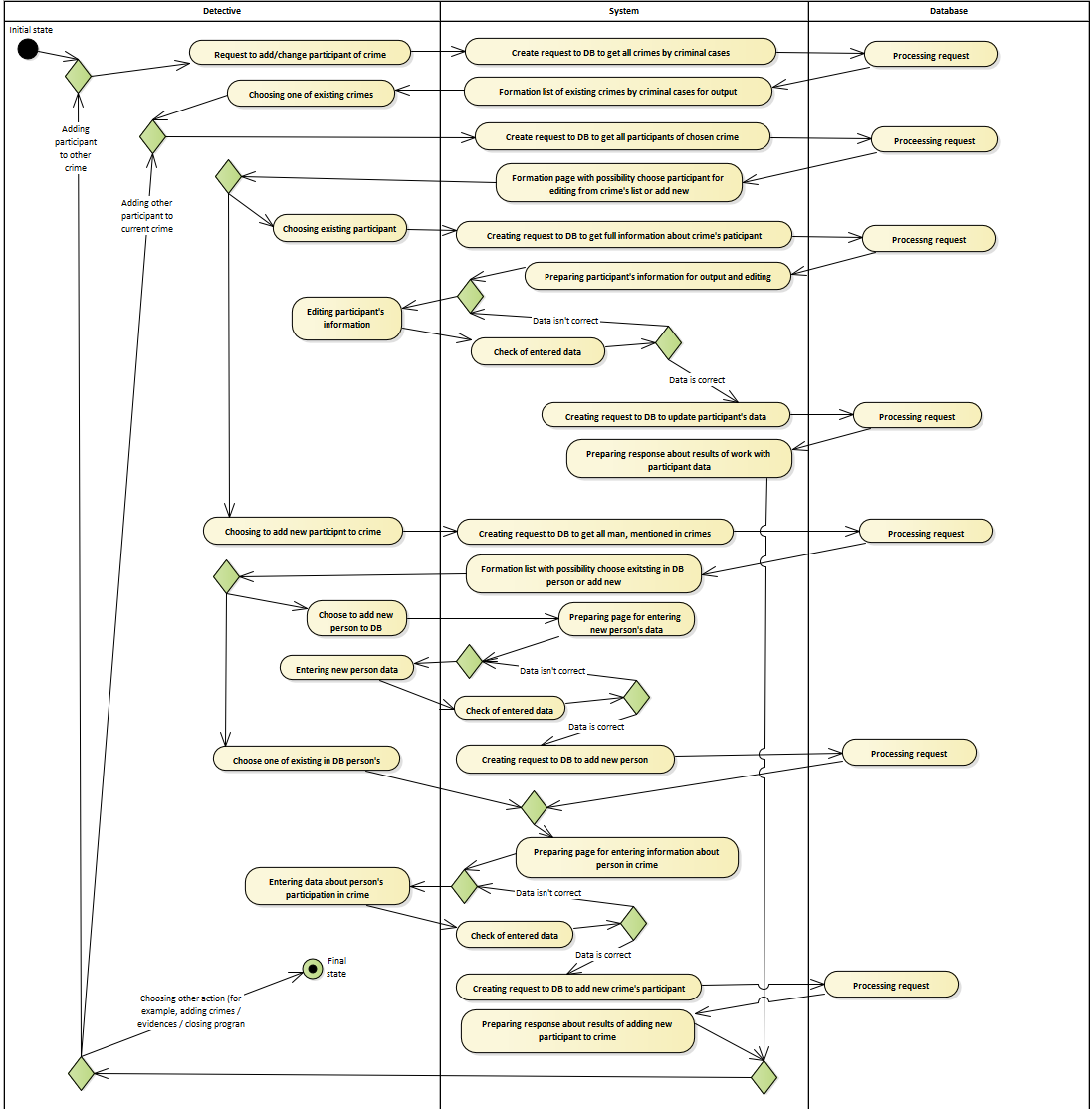


Рисунок 3.4 – Диаграмма Activity для добавления участника преступления

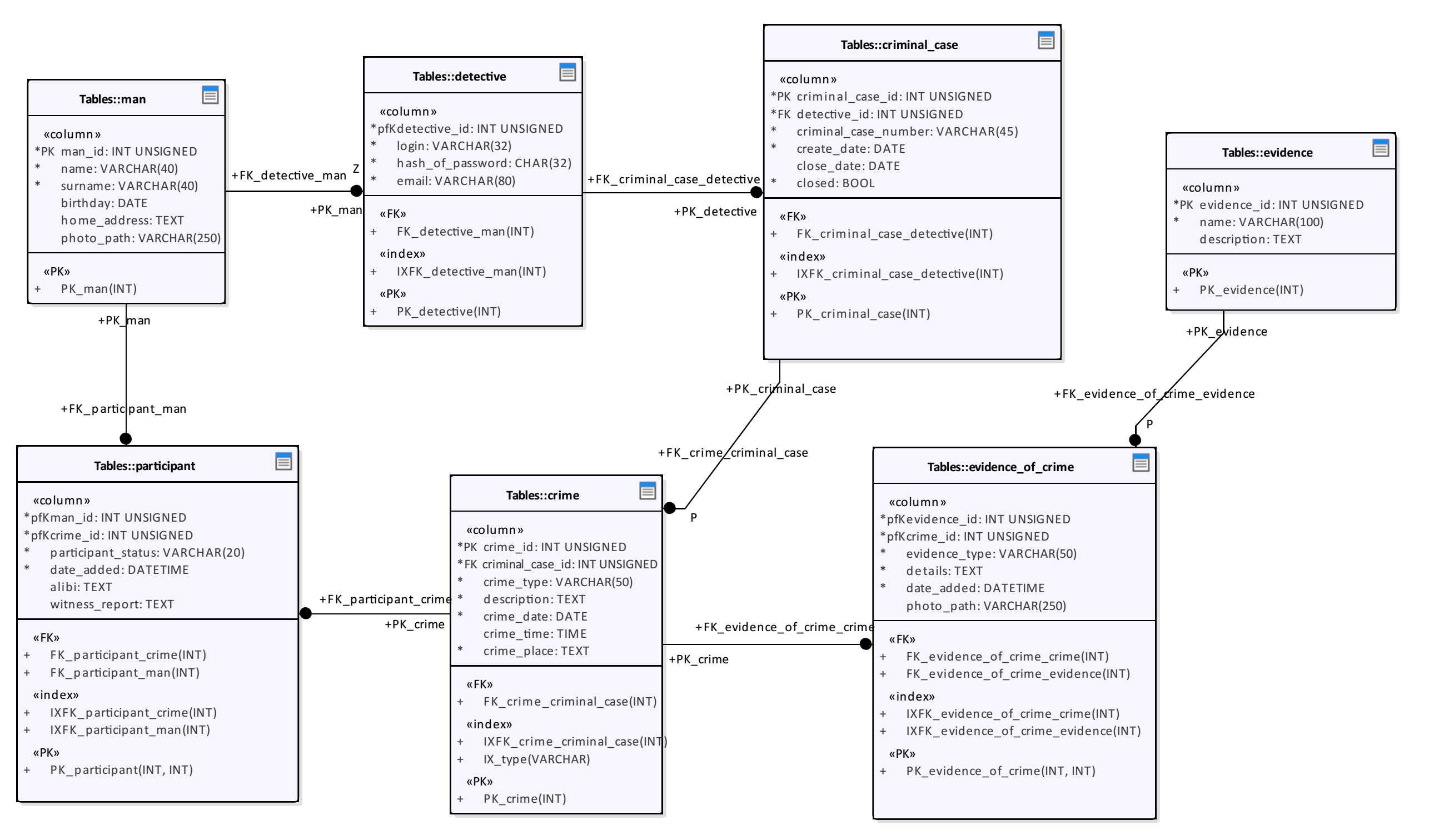


Рисунок 3.5 – Схема базы данных для СУБД «MySQL»

**3.3 Основные положения, принятые при проектировании программного средства**

При разработке программного средства важно однозначно определить ключевые моменты, которые могут играть важную роль при разработке приложения.

В связи с требованием о расширяемости и поддерживаемости приложения необходимо при разработке части, отвечающей за реализацию логики приложения, использовать разделение на независимые модули и наборы модулей схожего назначения (сервисы, контроллеры, объекты доступа к данным).

В приложении необходимо наличие системы авторизации пользователей. В связи с разделением приложения на серверную и клиентскую части, разумным вариантом является авторизация с помощью токена. При таком типе авторизации, в случае ввода пользователем корректных авторизационных данных, серверная часть должна создавать токен – строку, содержащую в зашифрованном виде данные о пользователе, дате завершения действия и другие сведения. Токен должен передаваться в серверную часть при любом REST-запросе, требующем авторизации, и анализироваться на корректность и другие параметры, такие как действительность на момент передачи. На начальной стадии разработки обновление токена при каждом запросе не планируется, но возможность для такого улучшения должна быть оставлена.

Пароли для хранения в базе данных необходимо зашифровывать; в качестве алгоритма шифрования был выбран алгоритм MD5, поскольку длина зашифрованного сообщения 32 символа является достаточной.

При генерации документов возможно временное сохранение созданного отчёта на диск, но обязательно с гарантированным последующим удалением.

Ещё одним моментом, который важно определить, это ответственность за хранение графических ресурсов. Возможно хранение изображений на сервере или на стороннем ресурсе (в последнем случае на сервер хранится адрес изображения). В нашем случае был выбран второй вариант со сторонним ресурсом.

**4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Языком программирования серверной части приложения был выбран язык *Java*. Данный язык широко используется при разработке веб-приложений, поскольку предлагает широкий набор возможностей как для разработки приложений, так и для их тестирования. Данный язык имеет возможность автоматизированного управления зависимыми библиотеками, при этом предлагает широкий набор библиотек (в том числе фреймворков) для автоматизации и упрощения разработки приложений, в том числе для сети Интернет. Одним из наиболее популярных фреймворков является *Spring* [4], который ускоряет написание приложения, предлагая готовые решения для множества задач (авторизация, реализация технологии REST, связывание компонентов). Именно данный набор инструментов и было решено использовать. Соответственно, в качестве среды разработки здесь была выбрана среда *InteliJ IDEA*.

Языком программирования клиентской части приложения был выбран язык *Angular 2*. Данный язык предназначен именно для создания одностраничных приложений и обладает широким спектром встроенных возможностей для упрощения разработки. Средой разработки была выбрана среди *Visual Studio Code*.

Рассмотрим принятые решения относительно выдвинутых к программному средству функциональных требований.

Контроль за наличием авторизационных данных принято отнести на клиентскую часть, потому что по результатам проверки данных пользователю должны быть отображены различные данные. В то же время, сама проверка возложена на серверную часть, которая осуществляет проверку введённых пользователем логина и пароля, генерацию и проверку токенов авторизации. Благодаря этому решении о наличие прав полностью возложен на серверную часть и неизвестен клиентской сторону. В связи с решением не реализовывать на начальной стадии разработки обновления токена после каждого запроса к серверной части, было принято ограничить время действия токена авторизации на срок в одну неделю. При необходимости пользователя выйти из системы, за удаление токена авторизации отвечает клиентская сторона.

В рамках системы действий для упрощения расширения и поддержки системы была выполнено разделение компонентов серверной части на следующие компоненты:

* контроллеры (выполняют принятие REST-запроса от клиентской части и дальнейшую его обработку с вызовом необходимых компонентов);
* сервисы (выполняют определённой действие, т.е. предоставляют сервис);
* объекты доступа к данным DAO (взаимодействуют с базой данных, выполняя получения, сохранение и обновление данных);
* объекты передачи данных DTO (осуществляют преобразование данных из типа, используемого в серверной части, к формату передачи данных json и обратное преобразование);
* объекты отображения (выполняют генерацию отчётов из переданных данных в требуемый формат файла);
* объекты обеспечения безопасности (проверка данных авторизации, работа с токенами);
* объекты конфигурации серверной части;
* запланированные события (по таймеру);
* непосредственно объекты данных, используемых в серверной части программного средства.

В клиентской части компоненты были разделены на:

* классы, используемые для хранения данных;
* сервисы, выполняющие операции;
* компоненты, содержащие страницы для отображения пользователю.

В серверной части контроллеры используются для обработки REST-запросов клиентской части. Для этого были использованы Spring-аннотации @RestController и @RequestMapping. В случае наличия параметров у запроса выполняется их преобразование в объект серверной стороны. После вызова необходимых сервисов выполняется создание объекта передачи данных DTO для последующего его преобразования в json, принимаемый клиентской стороной. Для автоматизированного управления созданием и удалением нужных объектов используется Spring-аннотация @Autowired.

Доступ к методам регистрации и авторизации пользователя доступен всё время, вне зависимости от наличия токена авторизации. Однако для обращения ко всем остальным входным точкам серверной части необходимо наличие HTTP-заголовка «deadpool-token», содержащего токен авторизации, полученный как содержимое json-объекта после успешной авторизации. В случае отсутствия заголовка, ошибочности токена или при истечении срока действия авторизации в доступе к серверной части будет отказано.

Ведение каталога преступлений осуществляется благодаря ранее спроектированной базе данных. Доступ к ней в серверной части осуществляется посредством объектов доступа к данным DAO. Комплекс этих объектов был написан с использованием оригинального компонента jdbc. Для сохранности от любых вмешательств используются подготовленные запросы PreparedStatement, использование которых гарантируется отсутствие повреждений в структуре базы данных.

На стадии проектирования базы данных было решено, что ограничение диапазона значений статусов и типов для преступления, улики и участника преступления будет осуществляться на серверной стороне. В связи с этим были созданы типы-перечисления, ограничивающие диапазон возможных значений. Каждому из них поставлено сообщение две строки: первая из них уникально идентифицирует значение перечисления и используется для хранения в базе данных, а вторая строка используется для отображения пользователю. Такой подход в будущем оставляет возможность для добавления поддержки других языков в интерфейс программного средства.

Серверная часть не имеет методов удаление данных, что соответствует поставленному функциональному требованию. Необходимый статусы и типы, соответствующие значению «удалено» или «неактуально», добавлены в список допустимых вариантов.

Отправка электронных писем осуществляется с использованием средств, предоставляемых фреймворком Spring; в частности, используется класс JavaMailSender. Отправка письма осуществляется в случае успешного добавления нового пользователя (детектива) в базу данных: в случае успешности действий из метода сервиса вызывается метод для отправки письма.

Для отправки писем также используется инструмент, предоставляемый набором Spring, – аннотация @Scheduled для создания события в определённое время с заданным интервалом повторений (параметр cron). Именно так и реализована отправка электронных писем с поздравлениями в честь дня рождения.

Генерация любого документа осуществляется по единой схеме: определяется тип и формат документа, затем вызывается требуемая функция, возвращающая адрес сгенерированного отчёта на диске, данный документ отправляется как ответ на REST-запрос, после чего временный файл удаляется с дискового пространства. При этом клиентская честь при выполнении запроса на документ должна указать в заголовке Accept тот MIME-тип, который соответствует желаемому типу файла; в противном случае, возникнет ошибочная ситуация, что будет отражено в коде HTTP-ответа. В свою очередь, серверная часть также устанавливает определённые заголовки в случае успешного создания отчёта. Устанавливается тип и длина документа, заголовки для отсутствия кэширования; заголовок «Content-Disposition» хранит, помимо слова «attachment», название файла через точку с запятой, а в заголовке «Deadpool\_filename» указывается только имя сгенерированного файла.

Для генерации документов в формате PDF была использована библиотека ITextPDF, для файлов в формате XLSX была использован набор инструментов Apache POI. Для создания документов в формате CSV сторонние библиотеки не использовались. Для создания защищённых документов PDF в файле отчёта устанавливались флаги ALLOW\_PRINTING и ALLOW\_SCREENREADERS, но обнулялся флаг ALLOW\_COPY.

Запись в журнал работы (логирование) на стороне сервера осуществлялась с использованием библиотеки log4j.

**5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И**

**АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Для тестирования программного средства были написаны JUnit-тесты в среде программирования InetliJ IDEA. Было решено писать тесты для ключевых компонентов серверной части: контроллеров, сервисов и объектов уровня DAO. Покрытие кода должно было составлять не менее 80 %.

В результате были написаны 195 тестов. Все тесты были выполнены успешно. Суммарный процент покрытия методов составил 96 %, этот же показатель для строк кода составил 85 %. Показатель покрытия строк кода для уровня DAO составил 81 %, для слоя сервисов – 86 %, а для контроллеров этот показатель оказался равным 90 %.

Более детальный анализ покрытия для уровня DAO представлен на рисунке 5.1, для сервисов – на рисунке 5.2, а результаты для слоя контроллеров визуализированы на рисунке 5.3.

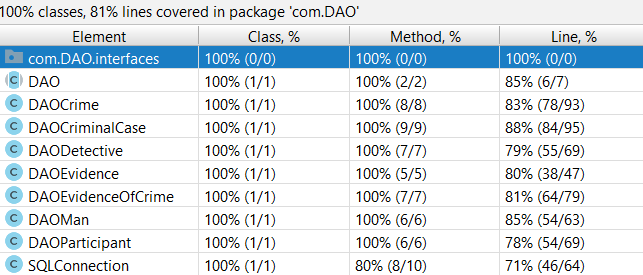


Рисунок 5.1 – Покрытие теста уровня объектов доступа к данным

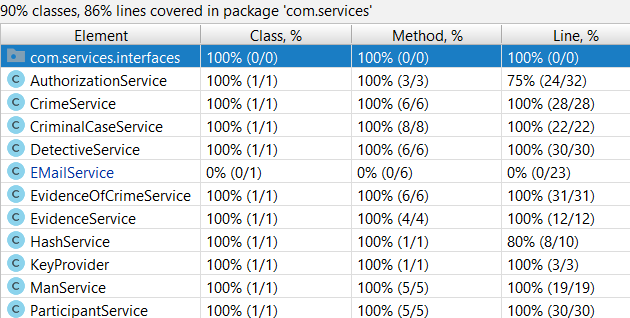


Рисунок 5.2 – Покрытие теста уровня сервисов

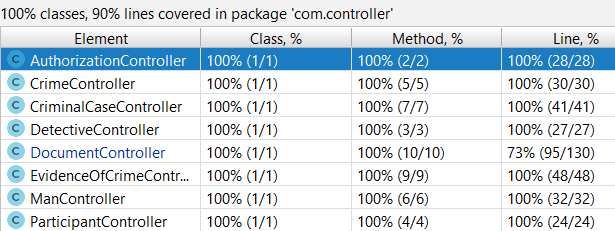


Рисунок 5.3 – Покрытие теста уровня контроллеров

Для покрытия тестами слой объектов доступа к данным были написаны дополнительные методы для добавления в базу данных готовых тестовых объектов и удаление всех тестовых объектов.

Для тестов всех уровней использовались объекты-заглушки, предоставляемые компонентом Mockito [5], позволяющие управлять поведением объектов, от которых зависит тестируемый компонент, для эмуляции различных ситуаций.

Для тестирования отправки электронной почты использовалась ручная проверка почты на предмет получения письма с указанным временем отправки, а также просмотр журнала приложения. Проведённые тесты показали корректное выполнение всех требуемых действий.

Таким образом, в ходе итогового тестирования не было выявлено каких-либо ошибок или некорректной работы приложения.

**6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ**

Для начала пользования приложением необходимо в браузере ввести адрес приложения (в случае запуска на локальном компьютере ­– http://localhost:4200/). Вам будет предложено пройти авторизацию (рис. 6.1). В случае наличия данных для авторизации, введите их в поля «Логин» и «Пароль», после чего нажмите кнопку «Войти». В противном случае, нажмите на кнопку «Регистрация», после чего вы увидите страницы для регистрации (рис. 6.2).

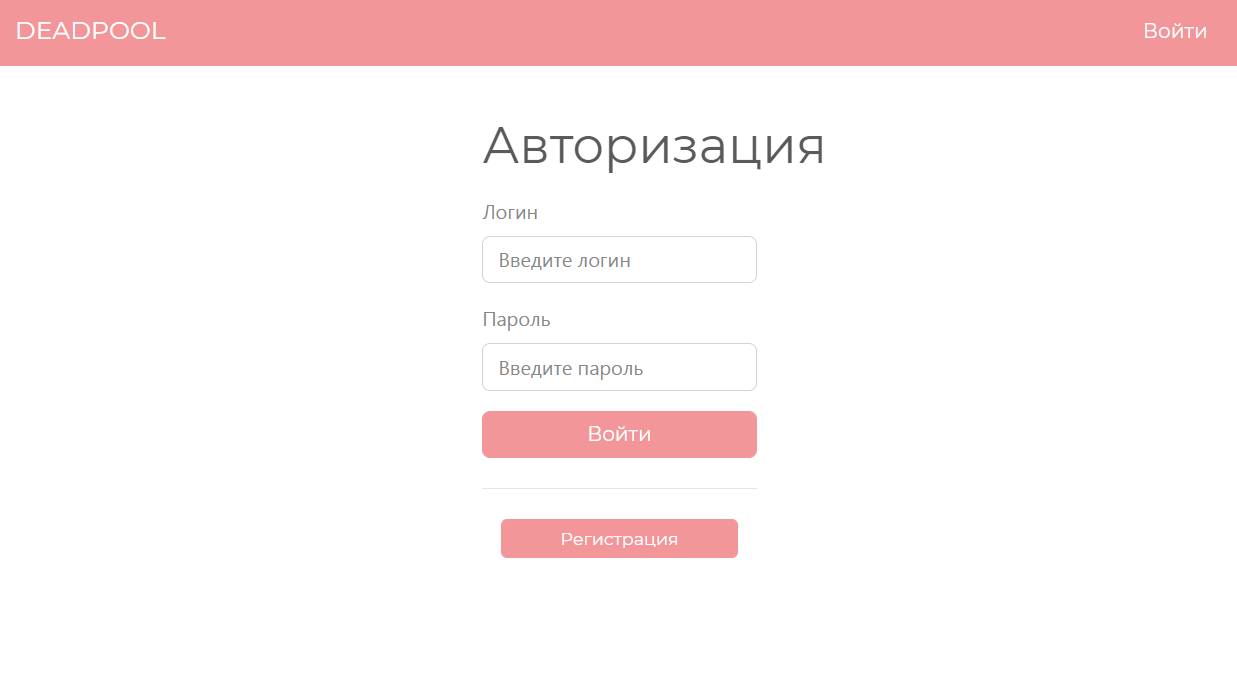


Рисунок 6.1 – Вид страницы авторизации приложения

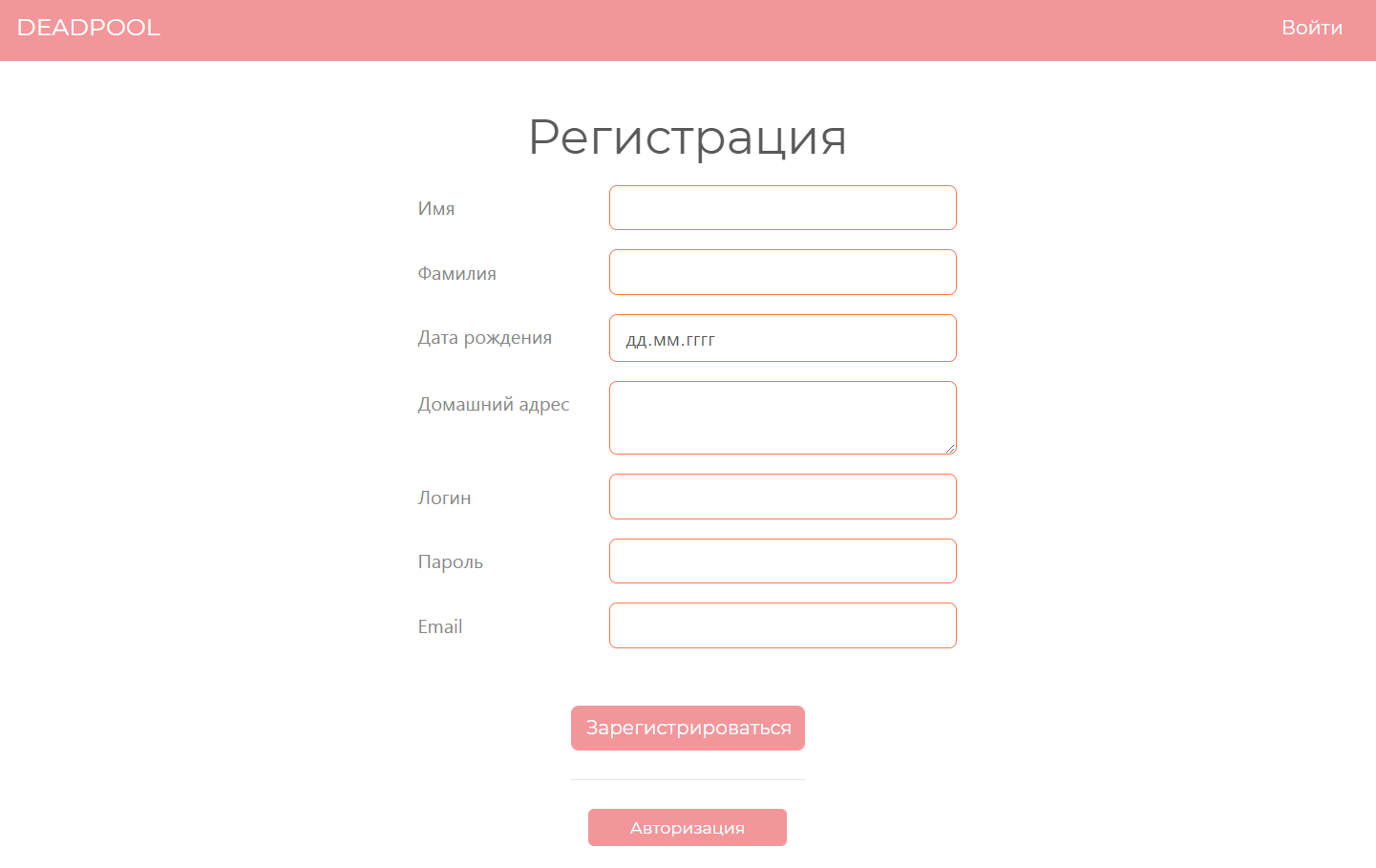


Рисунок 6.2 – Вид страницы регистрации приложения

На странице регистрации введите все необходимые данные. В случае некорректности данных в каком-то поле, контур данного поля будет выделен красным цветом. В случае успешности регистрации, вы будете перенаправлены на страницу авторизации, где можете ввести данные нового пользователя или любого другого.

После успешной авторизации будет отображена главная страница приложения, со списком всех дел (рис. 6.3). На левой боковой панели представлена возможность выбора категории. Нажатие на её элемент (например, «Улики») приводит к отображению соответствующего списка, что приведено на рисунке 6.4.



Рисунок 6.3 – Вид главной страницы приложения

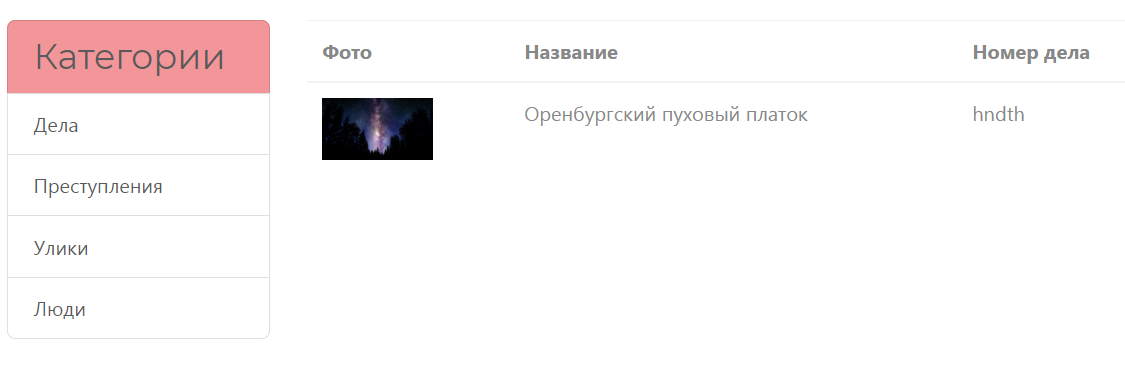


Рисунок 6.4 – Отображение списка улик

При выборе конкретного элемента (дела, преступления и т.п.) происходит переход на подробное описание элемента. Например, нажатие на дело приводит к отображение его полной информации (рис. 6.5). На боковой панели представлены действия для управления делом и его изменения.

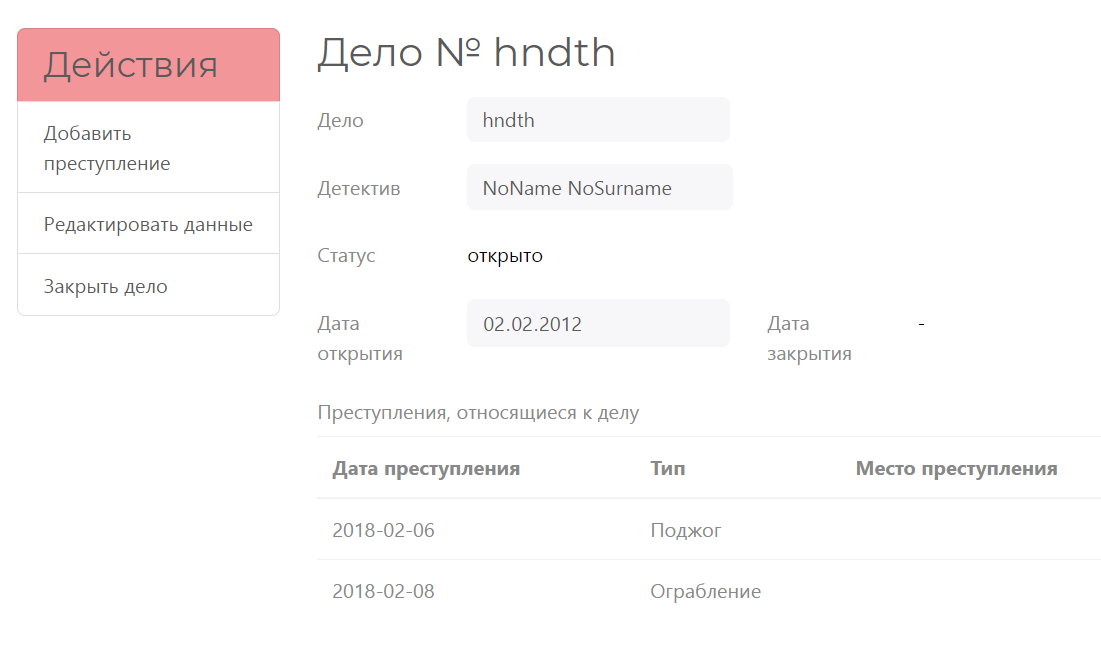


Рисунок 6.5 – Подробное описание дела

Нажатие на элемент «Редактировать данные» позволяет изменить все данные, относящиеся к элементу (рис. 6.6). Изменения фиксируются после нажатия кнопки «Сохранить изменения», возврат данных к предыдущему состоянию происходит после нажатия кнопки «Отмена». Закрытие или открытие дела осуществляется нажатием на отдельный элемент боковой панели управления.

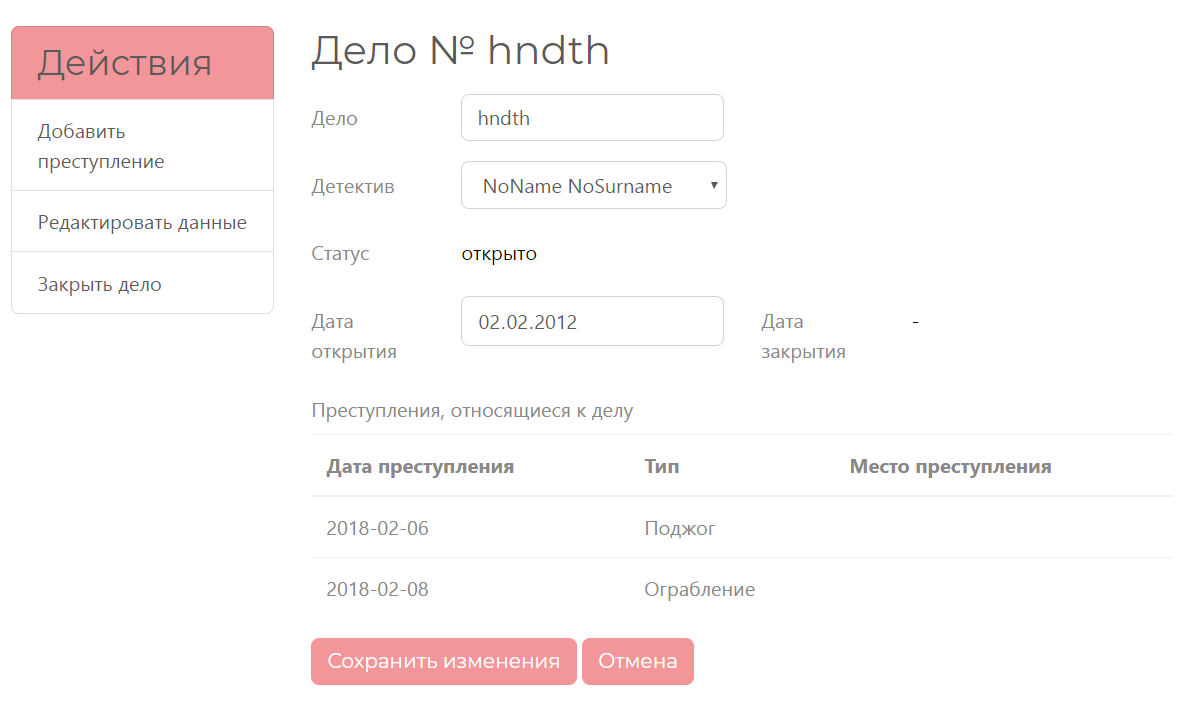


Рисунок 6.6 – Редактирование сведений о деле

Добавление нового дела осуществляется выбором соответствующего пункта на верхней панели управления.

Генерация отчётов доступна после выбора элемента «Отчёты» на верхней панели управления. На открывшейся страницы пользователь должен выбрать тип отчёта, дополнительные данные для него и формат файла (рис. 6.7). Генерация документа с последующим предложением сохранить выполняется после нажатия кнопки «Сгенерировать».

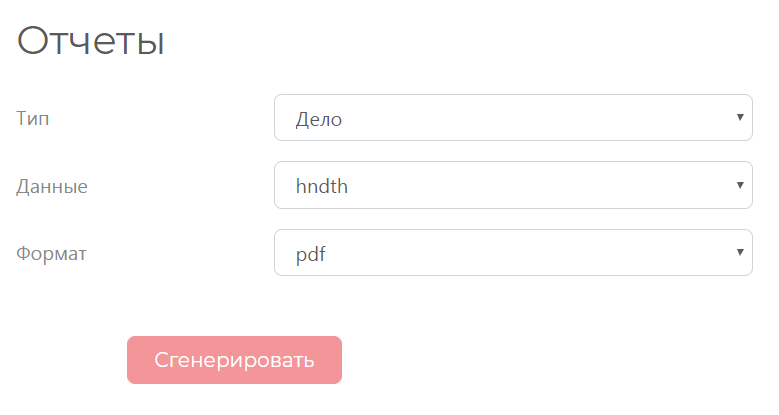


Рисунок 6.7 – Страница генерации отчётов

**ЗАКЛЮЧЕНИЕ**

В результате работы над курсовым проектом было сконструировано и разработано программное средство, выполняющее ведение каталога преступлений и связанной с ними информации. Разработанное программное средство в полной мере выполняет все операции, заданные в условии курсового проекта, работает корректно и без сбоев. Приложение доступно и отображается в любом современном браузере.

Созданные на этапе проектирования программного средства диаграммы оказали неоценимый вклад в создание приложения, внеся упорядоченность и поступательность в процесс разработки. Созданные алгоритмы позволили обратить внимание на неочевидные моменты в процесса разработки, требующие дальнейшего детального изучения.

Для создания программного средства было проведено ознакомление с различными современными технологиями для выбора наилучших решений для конкретного проекта. Были изучены возможности языка Angular 2 в рамках парадигмы одностраничных приложений. Широко использовались современные средства, предоставляемые для использования на языке Java, в частности, фреймворк Spring, позволяющий упростить и автоматизировать разработку на всех стадиях и предлагающий проверенные качественные решения для часто встречаемых задач.

Для тестирования были использованы JUnit-тесты и возможности их применения для различных частей приложения. Изучение и использование библиотеки Mockito позволило разглядеть новые возможности в тестировании сильно связанных компонентов приложения.

Работа была разделена на этапы, такие как анализ существующих решений и необходимости разработки, постановка требований к проектируемому программному средству, разработка алгоритма и его схемы, конструирование программного средства, отладка и тестирование. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее программное средство.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] http://www.interpol.ru/ [Электронный ресурс]**.** – Электронные данные. – Режим доступа: http://www.interpol.ru/Дата доступа: 09.02.18

[2] Исследования по проблеме компьютерного пиратства [Электронный ресурс]**.** – Электронные данные. – Режим доступа: https://news.tut.by/society/433248.html Дата доступа: 09.02.18

[3] SPA, Single Page Application, Одностраничное приложение [Электронный ресурс]**.** – Электронные данные. – Режим доступа: http://weblab.ua/spa/ Дата доступа: 23.02.18

[4] Spring [Электронный ресурс]**.** – Электронные данные. – Режим доступа: https://spring.io/ Дата доступа: 28.02.18

[5] Mockito framework site [Электронный ресурс]**.** – Электронные данные. – Режим доступа: http://site.mockito.org/ Дата доступа: 03.04.18

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

@RestController  
**public class** AuthorizationController {  
 **private final int tokenHoursExpire** = 24\*7;  
  
 @Autowired  
 **private** IDetectiveService **detectiveService**;  
  
 @Autowired  
 **private** IHashService **hashService**;  
  
 @Autowired  
 **private** IAuthorizationService **authorizationService**;  
  
 @CrossOrigin  
 @RequestMapping(path = **"/sign\_in"**, method = RequestMethod.***POST***)  
 **public** GenericDTO<String> signIn(@RequestBody AuthDTO authData, HttpServletResponse response) {  
 Detective detective = **detectiveService**.getDetectiveByLogin(authData.getLogin());  
 **if** (detective == **null**) {  
 response.setStatus(HttpServletResponse.***SC\_BAD\_REQUEST***);  
 **return new** GenericDTO<>(**true**,**"Нет пользователя с таким именем!"**);  
 } **else** {  
 **if** (**hashService**.getMD5Hash(authData.getPassword()).equals(detective.getHashOfPassword())) {  
 String token = **authorizationService**.getToken(detective, **tokenHoursExpire**);  
 **return new** GenericDTO<>(**false**, token);  
 } **else** {  
 response.setStatus(HttpServletResponse.***SC\_BAD\_REQUEST***);  
 **return new** GenericDTO<>(**true**, **"Неправильный пароль!"**);  
 }  
 }  
 }  
  
 @CrossOrigin  
 @RequestMapping(path = **"/sign\_up"**, method = RequestMethod.***POST***)  
 **public** GenericDTO<String> signUp(@RequestBody DetectiveWithoutManIdDTO authData, HttpServletResponse response) {  
 **boolean** exist = **detectiveService**.existDetectiveWithLogin(authData.getLogin());  
 **if** (exist) {  
 response.setStatus(HttpServletResponse.***SC\_BAD\_REQUEST***);  
 **return new** GenericDTO<>(**true**, **"Пользователь с таким именем уже существует!"**);  
 } **else** {  
 AddResult addResult = **detectiveService**.addDetective(  
 authData.getMan().getName(),  
 authData.getMan().getSurname(),  
 authData.getMan().getBirthday(),  
 authData.getMan().getHomeAddress(),  
 authData.getMan().getPhotoPath(),  
 authData.getLogin(),  
 authData.getPassword(),  
 authData.getEmail()  
 );  
 **if** (addResult.getResult()) {  
 **return new** GenericDTO<>(**false**, **"Вы успешно зарегистрированы в системе!"**);  
 } **else** {  
 response.setStatus(HttpServletResponse.***SC\_BAD\_REQUEST***);  
 **return new** GenericDTO<>(**true**, **"Не удалось добавить пользователя!"**);  
 }  
 }  
 }}

@RestController  
@RequestMapping(value = **"/participants"**)  
**public class** ParticipantController {  
 @Autowired  
 **private** IParticipantService participantService;  
  
 @IsDetective  
 @CrossOrigin  
 @RequestMapping(path = **"/add"**, method = RequestMethod.POST)  
 **public** OperationResultDTO addParticipant(@RequestBody ParticipantInputDTO participant) {  
 **boolean** result = participantService.addParticipant(  
 participant.getMan().getId(),  
 participant.getCrime().getId(),  
 participant.getStatus(),  
 participant.getDateAdded(),  
 participant.getTimeAdded(),  
 participant.getAlibi(),  
 participant.getWitnessReport()  
 );  
 **return new** OperationResultDTO(result);  
 }  
  
 @IsDetective  
 @CrossOrigin  
 @RequestMapping(path = **"/update"**, method = RequestMethod.POST)  
 **public** OperationResultDTO updateParticipant(@RequestBody ParticipantInputDTO participant) {  
 **boolean** result = participantService.updateParticipant(  
 participant.getMan().getId(),  
 participant.getCrime().getId(),  
 participant.getStatus(),  
 participant.getDateAdded(),  
 participant.getTimeAdded(),  
 participant.getAlibi(),  
 participant.getWitnessReport()  
 );  
 **return new** OperationResultDTO(result);  
 }  
@IsDetective  
 @CrossOrigin  
 @RequestMapping(path = **"/{man\_id}/{crime\_id}"**, method = RequestMethod.***GET***)  
 **public** GenericDTO<ParticipantFullInfoDTO> getParticipantByManAndCrime(  
 @PathVariable(**"man\_id"**) **long** manId,  
 @PathVariable(**"crime\_id"**) **long** crimeId  
 ) {  
 Participant participant = **participantService**.getParticipantByCrimeAndMan(manId, crimeId);  
 **return** (participant != **null**)  
 ? **new** GenericDTO<>(**false**, ParticipantParser.*parseParticipantFullInfo*(participant))  
 : **new** GenericDTO<>(**true**, **null**);  
 }  
  
 @IsDetective  
 @CrossOrigin  
 @RequestMapping(path = **"/status\_list"**, method = RequestMethod.***GET***)  
 **public** ListEnumDTO getParticipantStatuses() {  
 **return** ParticipantParser.*getParticipantStatuses*();  
 }  
}

@Service  
**public class** AuthorizationService **implements** IAuthorizationService {  
 **static** Logger *log* = Logger.*getLogger*(AuthorizationService.**class**.getName());  
  
 @Autowired  
 **private** IKeyProvider **keyProvider**;  
  
 @Override  
 **public** String getToken(Detective user, **long** hoursExpire) {  
 String token;  
 **try** {  
 Algorithm algorithm = Algorithm.*HMAC256*(**keyProvider**.getHS256Key());  
 token = JWT.*create*()  
 .withIssuer(**"auth0"**)  
 .withSubject(user.getLogin())  
 .withIssuedAt(**new** Date(System.*currentTimeMillis*()))  
 .withExpiresAt(hoursExpire > 0 ? **new** Date(System.*currentTimeMillis*() + hoursExpire\*60\*60\*1000) : **null**)  
 .sign(algorithm);  
 } **catch** (UnsupportedEncodingException exception){  
 token = **null**;  
 *log*.error(**"Error with encoding of token"**, exception);  
 } **catch** (JWTCreationException exception){  
 token = **null**;  
 *log*.error(**"Error creating JWT token"**, exception);  
 }  
 **return** token;  
 }  
  
 @Override  
 **public** TokenVerifyResult checkToken(String token) {  
 String userName = **null**;  
 **boolean** result = **false**;  
 **try** {  
 Algorithm algorithm = Algorithm.*HMAC256*(**keyProvider**.getHS256Key());  
 JWTVerifier verifier = JWT.*require*(algorithm)  
 .withIssuer(**"auth0"**)  
 .build();  
 DecodedJWT jwt = verifier.verify(token);  
 userName = jwt.getSubject();  
 result = **true**;  
 } **catch** (UnsupportedEncodingException exception){  
 *log*.error(**"Error with encoding of token"**, exception);  
 } **catch** (JWTVerificationException exception){  
 *log*.error(**"JWT verification error"**, exception);  
 }  
 **return new** TokenVerifyResult(result, userName);  
 }  
}

@Service  
**public class** ParticipantService **implements** IParticipantService {  
 @Autowired  
 **private** IDAOParticipant **daoParticipant**;  
  
 @Transactional  
 @Override  
 **public** List<Participant> getParticipantsByCrimeId(**long** id) {  
 **return daoParticipant**.getAllParticipantsByCrime(id);  
 }  
  
 @Transactional  
 @Override  
 **public** List<Participant> getParticipantsByManId(**long** id) {  
 **return daoParticipant**.getAllParticipantsByMan(id);  
 }  
  
 @Transactional  
 @Override  
 **public** Participant getParticipantByCrimeAndMan(**long** manId, **long** crimeId) {  
 **return daoParticipant**.getParticipantById(manId, crimeId);  
 }  
  
 @Transactional  
 @Override  
 **public boolean** addParticipant(**long** manId, **long** crimeId, String status, LocalDate dateAdded, LocalTime timeAdded, String alibi, String witnessReport) {  
 Participant participant = **new** Participant();  
 participant.setManId(manId);  
 participant.setCrimeId(crimeId);  
 **try** {  
 participant.setParticipantStatus(status);  
 } **catch** (IllegalArgumentException ex) {  
 **return false**;  
 }  
 **if** (dateAdded != **null** && timeAdded != **null**) {  
 participant.setDateAdded(LocalDateTime.*of*(dateAdded, timeAdded));  
 } **else** {  
 **return false**;  
 }  
 participant.setAlibi(alibi);  
 participant.setWitnessReport(witnessReport);  
 **return daoParticipant**.addParticipant(participant);  
 }  
  
 @Transactional  
 @Override  
 **public boolean** updateParticipant(**long** manId, **long** crimeId, String status, LocalDate dateAdded, LocalTime timeAdded, String alibi, String witnessReport) {  
 Participant participant = **new** Participant();  
 participant.setManId(manId);  
 participant.setCrimeId(crimeId);  
 **try** {  
 participant.setParticipantStatus(status);  
 } **catch** (IllegalArgumentException ex) {  
 **return false**;  
 }  
 **if** (dateAdded != **null** && timeAdded != **null**) {  
 participant.setDateAdded(LocalDateTime.*of*(dateAdded, timeAdded));  
 } **else** {  
 **return false**;  
 }  
 participant.setAlibi(alibi);  
 participant.setWitnessReport(witnessReport);  
 **return daoParticipant**.updateParticipant(participant);  
 }  
}

@Repository  
**public class** DAOParticipant **extends** DAOMan **implements** IDAOParticipant {  
 **static** Logger *log* = Logger.*getLogger*(DAOParticipant.**class**.getName());  
  
 @Override  
 **public** Participant getParticipantById(**long** manId, **long** crimeId) {  
  
 PreparedStatement preparedStatement = **currConnection**.prepareStatement(**"SELECT \* FROM participant JOIN man USING(man\_id) WHERE crime\_id = ? AND participant.man\_id = ? "**);  
 *//PreparedStatement preparedStatement = currConnection.prepareStatement("SELECT \* FROM `participant` WHERE `crime\_id` = ? AND `man\_id` = ?");* **try** {  
 preparedStatement.setLong(1, crimeId);  
 preparedStatement.setLong(2, manId);  
 } **catch** (SQLException e) {  
 *log*.error(e.toString());  
 **return null**;  
 }  
 List<HashMap<String, Object>> retArray = **currConnection**.queryFind(preparedStatement);  
  
 **if** (retArray.isEmpty()) **return null**;  
 Participant retParticipantRecord = **new** Participant();  
 retParticipantRecord.setManId(manId);  
 retParticipantRecord.setCrimeId(crimeId);  
  
 ProjectFunctions.*tryFillObjectByDbArray*(retParticipantRecord, retArray.get(0));  
 **return** retParticipantRecord;  
 }  
  
 @Override  
 **public boolean** updateParticipant(Participant participantToUpdate) {  
 **if** (participantToUpdate == **null**) **return false**;  
  
 PreparedStatement preparedStatement = **currConnection**.prepareStatement(**"UPDATE `participant` SET "** +  
 **"`participant\_status`=?,"** +  
 **"`alibi`=?,"** + *//nullable* **"`witness\_report`=?,"** + *//nullable* **"`date\_added`=? "** +  
 **"WHERE man\_id = ? AND crime\_id = ?"**);  
 **try** {  
 preparedStatement.setString(1, participantToUpdate.getParticipantStatus().toString());  
 preparedStatement.setString(2, participantToUpdate.getAlibi());  
 preparedStatement.setString(3, participantToUpdate.getWitnessReport());  
 preparedStatement.setTimestamp(4, Timestamp.*valueOf*(participantToUpdate.getDateAdded()));  
 preparedStatement.setLong(5, participantToUpdate.getManId());  
 preparedStatement.setLong(6, participantToUpdate.getCrimeId());  
 } **catch** (SQLException e) {  
 *log*.error(e.toString());  
 **return false**;  
 }  
  
 **return currConnection**.queryDataEdit(preparedStatement);  
 }  
  
 @Override  
 **public** List<Participant> getAllParticipantsByMan(**long** manId) {  
 PreparedStatement preparedStatement = **currConnection**.prepareStatement(**"SELECT \* FROM participant JOIN man USING(man\_id) WHERE participant.man\_id = ?"**);  
 List<Participant> participants = **new** ArrayList<Participant>();  
  
 **try** {  
 preparedStatement.setLong(1, manId);  
 } **catch** (SQLException e) {  
 *log*.error(e.toString());  
 **return null**;  
 }  
 List<HashMap<String, Object>> retArray = **currConnection**.queryFind(preparedStatement);  
  
 **for** (**int** i = 0; i < retArray.size(); i++) {  
 Participant retParticipantRecord = **new** Participant();  
 ProjectFunctions.*tryFillObjectByDbArray*(retParticipantRecord, retArray.get(i));  
 participants.add(retParticipantRecord);  
 }  
 **return** participants;  
 }

}

@Component  
**public class** ScheduledEvents {  
 **static** Logger *log* = Logger.*getLogger*(ScheduledEvents.**class**.getName());  
  
 @Autowired  
 **private** IDAODetective **daoDetective**;  
  
 @Autowired  
 **private** IEMailService **eMailService**;  
  
 @Scheduled(cron = **"0 23 15 \* \* \*"**)  
 **public void** sendBirthdayGratings() {  
 *log*.info(**"Метод sendBirthdayGratings был вызван в "** + LocalDateTime.*now*().format(*reportTitleDateTimeFormatter*));  
 List<Detective> allDetectives = **daoDetective**.getAllDetectives();  
 List<Detective> detectives = allDetectives.stream()  
 .filter(det -> det.getBirthDay() != **null** && LocalDate.*now*().getMonth() == det.getBirthDay().getMonth()  
 && LocalDate.*now*().getDayOfMonth() == det.getBirthDay().getDayOfMonth())  
 .collect(Collectors.*toList*());  
 **for**(Detective det : detectives) {  
 **eMailService**.sendTextEMail(det.getEmail(), **"Позравляем с днём рождения!"**,  
 **"Уважаемый "** + det.getName() + **" "** + det.getSurname() + **" !\n"** +  
 **"Администрация приложения \"Deadpool\" от всей души поздравляет Вас с днём рождения!"** +  
 **"Желаем крепкого здоровья и реализации всех жизненых планов!\n"** + **"\n"** +  
 **"Поздравляю\n"** +  
 **"с днём рожденья,\n"** +  
 **"желаю счастья в личной жизни.\n"** +  
 **"Пух!"**);  
 }  
 }  
}

**ПРИЛОЖЕНИЕ Б**

**Исходный код юнит-тестов**

@WebMvcTest  
**public class** ParticipantControllerTest {  
 **private** MockMvc mockMvc;  
  
 @Mock  
 **private** IParticipantService participantService;  
  
 @InjectMocks  
 **private** ParticipantController controller;  
  
 **private static** ObjectMapper objectMapper;  
  
 @Before  
 **public void** init(){  
 MockitoAnnotations.initMocks(**this**);  
 mockMvc = MockMvcBuilders  
 .standaloneSetup(controller)  
 .addFilters(**new** CORSFilter())  
 .build();  
 }  
  
 @BeforeClass  
 **public static void** getDAO() {  
 objectMapper = LogicAdditionals.objectMapper();  
 }  
  
 @Test  
 **public void** addParticipant() **throws** Exception {  
 ParticipantInputDTO inputJson = **new** ParticipantInputDTO();  
 inputJson.setMan(**new** IdOnlyDTO());  
 inputJson.setCrime(**new** IdOnlyDTO());  
 OperationResultDTO response = **new** OperationResultDTO(**true**);  
  
 when(participantService.addParticipant(inputJson.getMan().getId(), inputJson.getCrime().getId(), inputJson.getStatus(), inputJson.getDateAdded(), inputJson.getTimeAdded(), inputJson.getAlibi(), inputJson.getWitnessReport())).thenReturn(**true**);  
  
 mockMvc.perform(  
 post(**"/participants/add"**)  
 .contentType(MediaType.APPLICATION\_JSON\_UTF8)  
 .header(**"deadpool-token"**, TokensForTests.getCorrectTokenUnlimited())  
 .content(objectMapper.writeValueAsString(inputJson)))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(objectMapper.writeValueAsString(response)));  
 }  
  
 @Test  
 **public void** updateParticipant() **throws** Exception {  
 ParticipantInputDTO inputJson = **new** ParticipantInputDTO();  
 inputJson.setMan(**new** IdOnlyDTO());  
 inputJson.setCrime(**new** IdOnlyDTO());  
 OperationResultDTO response = **new** OperationResultDTO(**true**);  
  
 when(participantService.updateParticipant(inputJson.getMan().getId(), inputJson.getCrime().getId(), inputJson.getStatus(), inputJson.getDateAdded(), inputJson.getTimeAdded(), inputJson.getAlibi(), inputJson.getWitnessReport())).thenReturn(**true**);  
  
 mockMvc.perform(  
 post(**"/participants/update"**)  
 .contentType(MediaType.APPLICATION\_JSON\_UTF8)  
 .header(**"deadpool-token"**, TokensForTests.getCorrectTokenUnlimited())  
 .content(objectMapper.writeValueAsString(inputJson)))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(objectMapper.writeValueAsString(response)));  
 }  
  
 @Test  
 **public void** getParticipantByManAndCrimeCorrect() **throws** Exception {  
 Participant participant = LogicAdditionals.getCustomParticipant();  
 GenericDTO<ParticipantFullInfoDTO> response = **new** GenericDTO<>(**false**, ParticipantParser.parseParticipantFullInfo(participant));  
  
 when(participantService.getParticipantByCrimeAndMan(anyLong(), anyLong())).thenReturn(participant);  
  
 mockMvc.perform(  
 get(**"/participants/{man\_id}/{crime\_id}"**, 1, 1)  
 .header(**"deadpool-token"**, TokensForTests.getCorrectTokenUnlimited()))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(objectMapper.writeValueAsString(response)));  
 }  
  
 @Test  
 **public void** getParticipantByManAndCrimeNotCorrect() **throws** Exception {  
 GenericDTO<ParticipantFullInfoDTO> response = **new** GenericDTO<>(**true**, **null**);  
  
 when(participantService.getParticipantByCrimeAndMan(anyLong(), anyLong())).thenReturn(**null**);  
  
 mockMvc.perform(  
 get(**"/participants/{man\_id}/{crime\_id}"**, 1, 1)  
 .header(**"deadpool-token"**, TokensForTests.getCorrectTokenUnlimited()))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(objectMapper.writeValueAsString(response)));  
 }  
  
 @Test  
 **public void** getParticipantStatuses() **throws** Exception {  
 ListEnumDTO list = ParticipantParser.getParticipantStatuses();  
  
 mockMvc.perform(  
 get(**"/participants/status\_list"**)  
 .header(**"deadpool-token"**, TokensForTests.getCorrectTokenUnlimited()))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(objectMapper.writeValueAsString(list)));  
 }  
}

**public class** AuthorizationServiceTest {  
 @Mock  
 **private** IKeyProvider **keyProvider**;  
  
 @InjectMocks  
 **private** AuthorizationService **service**;  
  
 **private static** IKeyProvider *realKeyProvider*;  
  
 @BeforeClass  
 **public static void** getDAO() {  
 *realKeyProvider* = **new** KeyProvider();  
 }  
  
 @Before  
 **public void** init(){  
 MockitoAnnotations.*initMocks*(**this**);  
 }  
  
 @Test  
 **public void** getToken() **throws** Exception {  
 **long** hoursExpire = 24\*7;  
 Detective user = **new** Detective();  
 user.setLogin(**"some\_login"**);  
  
 *when*(**keyProvider**.getHS256Key()).thenReturn(*realKeyProvider*.getHS256Key());  
  
 String actualResult = **service**.getToken(user, hoursExpire);  
  
 *assertNotEquals*(**null**, actualResult);  
  
 String userName = **null**;  
 **try** {  
 Algorithm algorithm = Algorithm.*HMAC256*(*realKeyProvider*.getHS256Key());  
 JWTVerifier verifier = JWT.*require*(algorithm)  
 .withIssuer(**"auth0"**)  
 .build();  
 DecodedJWT jwt = verifier.verify(actualResult);  
 userName = jwt.getSubject();  
 } **catch** (UnsupportedEncodingException exception){  
 String t = Console.**class**.toString();  
 } **catch** (JWTVerificationException exception){  
 String t = Console.**class**.toString();  
 }  
  
 *assertEquals*(user.getLogin(), userName);  
 }  
  
 @Test  
 **public void** checkTokenCorrect() **throws** Exception {  
 **long** hoursExpire = 24\*7;  
 Detective user = **new** Detective();  
 user.setLogin(**"some\_login"**);  
 *when*(**keyProvider**.getHS256Key()).thenReturn(*realKeyProvider*.getHS256Key());  
 String token = **service**.getToken(user, hoursExpire);  
  
 TokenVerifyResult result = **service**.checkToken(token);  
 *assertEquals*(**true**, result.getIsCorrect());  
 *assertEquals*(user.getLogin(), result.getUserName());  
 }  
  
 @Test  
 **public void** checkTokenNotCorrect() **throws** Exception {  
 *when*(**keyProvider**.getHS256Key()).thenReturn(*realKeyProvider*.getHS256Key());  
  
 TokenVerifyResult result = **service**.checkToken(**"fjycgvbu.oiykcfjchghbkjniuulgjh.kjnluvkcfhg"**);  
 *assertEquals*(**false**, result.getIsCorrect());  
 *assertEquals*(**null**, result.getUserName());  
 }  
}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 010 ПЗ | | | | Пояснительная записка | | | | 34 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 551001 010 СП | | | | Программное средство для ведения каталога преступлений. Схема программы | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 010 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программное средство для ведения каталога преступлений.  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Лабоха А.К. |  | 07.05.18 | Т |  | |  | 34 | 34 |
| Пров. | | Пуздров М.Ю. |  | 07.05.18 | Кафедра ПОИТ  гр. 551001 | | | | | |
|  | |  |  |  |