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

СОДЕРЖАНИЕ 4

ВВЕДЕНИЕ 5

1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ СТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ 6

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 8

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9

3.1 Проектирование взаимодействия серверной и клиентской частей системы 9

3.2 Проектирование серверной части 9

3.3 Проектирование клиентской части 10

3.4 Основные положения, принятые при проектировании программного средства 10

4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 11

5. ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 13

ЗАКЛЮЧЕНИЕ 19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20

# ВВЕДЕНИЕ

На сегодняшний день одним из наиболее востребованных направлением в программировании является разработка компьютерных игр. Несмотря на то, что сейчас более популярными играми являются 3D шутеры, ролевые и экшен, для получения опыта в этой области в качестве темы моего курсового проекта я выбрал 2D видеоигру, так как именно с таких игр началась эра компьютерных игр. Из всех жанров я выбрал шутер, потому что был заинтересован в разработке динамической игры, где персонаж должен постоянно бегать, прыгать, а также взаимодействовать с игровым миром: лазать по лестницам и стрелять во врагов. Но главной особенностью игры является то, что в неё может играть сразу несколько людей, находящихся в одной локальной сети и подключивщихся к одному серверу.

В процессе написания курсового проекта я планирую освоить некоторые методы программного построения двумерного мира с движущимися по привычным нам физическим законам объектами. Помимо этого я бы хотел, чтобы программа взаимодействовала с файлами, в которых будет вся информация о картах, анимациях, а также картинки, звуки и музыка. Так как логика игры делится на две стороны (клиент и сервер), то мне необходимо построить логику на стороне клиента так, чтобы у игрока была возможность подключаться к серверам с разными правилами игры.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта:

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

2) Анализ требований к программному средству и разработка функциональных требований.

3) Проектирование программного средства.

4) Конструирование программного средства.

5) Тестирование, проверка работоспособности и анализ полученных результатов.

6) Руководство пользователя программы.

# АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ СТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

За идею я взял известную компьютерную игру в жанре шутера от первого лица «Call Of Duty 4: Modern Warfare», которая была разработана [американской](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%A8%D1%82%D0%B0%D1%82%D1%8B_%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B8" \o "Соединённые Штаты Америки) компанией [Infinity Ward](https://ru.wikipedia.org/wiki/Infinity_Ward" \o "Infinity Ward) и издана [Activision](https://ru.wikipedia.org/wiki/Activision" \o "Activision). Помимо прохождения сюжета, игрок может также принять участие в многопользовательской игре по локальной сети или интернету. До сих пор люди проводят временя на некоторых серверах, играя в разных режимах.



Рисунок 11 - Список серверов

Помимо лекционного материала, я глубже изучил принцип работы многопользовательской игры [1]. Также я поиграл на нескольких серверах и проанализировал игровой мир, чтобы сформировать список требований для моего программного средства.

При подключении к серверу в случае, когда у клиента нет данных о карте, на которой идёт игра на сервере, клиент автоматически скачивает данные с сервера. После обмена информации о карте игрок выбирает команду. Во всех режимах (за исключением свободной игры, там каждый играет сам за себя) команды две. Если количество игроков в командах различается больше, чем на один, то происходит автобаланс - последнего подключившегося игрока из команды с большим числом игроков переводят в другую команду, что позволяет уровнять бой. Бой ведётся, пока не истечёт время или не наберётся определённое количество очков. Очки начисляются за убийство врага. После отключения от сервера есть возможность выбрать другой, не выходя из игры.



Рисунок 12 - Игровой мир

После проведения анализа, я решил, что моё программное средство будет работать по схожему принципу. Единственное принципиальное отличие заключается в том, что я буду проектировать 2D игру от третьего лица вместо 3D от первого лица, так как в этом проекте основной целью является взаимодействие игроков по сети. Поэтому я планирую опустить графику на второй план, чтобы сосредоточить силы на написание логики сервера и клиента.

На стороне сервера необходимо иметь файлы с данными о карте, включая текстуру (она понадобится исключительно для передачи клиенту). После инициализации сервер должен принимать новых игроков, не прерывая обновление уже играющих. При подключении к серверу клиент должен удостоверится в наличии данных о карте, и при необходимости сервер должен разрешить скачивание недостающих данных. После этого сервер даёт возможность игроку ввести своё имя и выбрать команду. Если игрок ввёл уже существующее имя, к строке с его именем прибавится «\_1» в конце, а сам игрок уведомится об этом. При подключении или отключении какого-нибудь игрока сервер обязан уравнять команды по игрокам.

На стороне клиента должна быть возможность выбрать сервер путём ввода его локального IP адреса и порта. После скачивания недостающих файлов, ввода имени и выбора команды игрок может приступить к игре. Помимо передвижения влево и вправо игрок может прыгать и лазать по лестницам, а так же стрелять по врагам. После смерти игрок будет возрождаться не сразу, а после нажатия определённой клавиши. Кроме игрового мира на экране будут отображатся игроки двух команд, пули, счёт и здоровье игрока. Игрок будет иметь возможность в любой момент отключиться от сервера и подключится к другому, не закрывая приложение.

# АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

Исходя из поставленной задачи и сформулированных требований, сервер должен выполнять следующие функции:

- подключение новых игроков без прерывания обновления других;

- проверить при подключении наличие всех файлов на клиенте;

- проверить на уникальность введённое имя игрока и при обратном прибавить к строке с именем «\_1»;

- автораспределение при подключении или отключении игрока и разнице игроков в командах более чем один;

- приём данных о нажатых клавишах, обновление позиции и отправка полученной игровой ситуации всем игрокам;

- реализация передвижения игрока влево и вправо, прыжка, лазанья по лестницам, стрельбы, получения урона, возрождения по готовности;

- реалистичное обновление положений объектов.

На стороне клиента обязаны быть реализованными следующие функции:

- возможность выбора сервера;

- поддерживать несколько режимов игры, любые карты и любые текстуры;

- отображать всех игроков, пули, счёт и шкалу здоровья;

- отправка данных о нажатых клавишах серверу, приём игровой ситуации и её отображение.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 3.1 Проектирование взаимодействия серверной и клиентской частей системы

Так как режимов игры или карт может быть несколько, то сервера могут быть разные, на каждом из которых крутится определённый режим, тем самым давая игрокам выбор. Но клиент должен быть реализован в одном экземпляре, чтобы пользователю было достаточно скачать одно приложение. Для этого подключение к разным серверам должно включать одинаковые этапы.

Сначала проверяет наличие на клиенте файла с информацией о карте, текстуры и заднего фона. Если у клиента отсутствует какой-либо файл, то производится скачивание недостающего файла с сервера. После этого клиент вводит своё имя, а сервер проверяет его на уникальность. Если это имя уже использовано, то к строке с именем прибавляется «\_1», а клиент об этом уведомляется. Далее сервер показывает текущее количество игроков в командах и даёт возможность клиенту выбрать за какую команду играть. В случае, когда разница между количествами игроков в командах больше одного, происходит автобаланс. После этого клиент становится новым игроком - его данные о нажатых клавишах принимаются сервером, обрабатываются и высылаются всем игрокам.

## 3.2 Проектирование серверной части

Инициализация сервера заключается в следующем:

- считывание данных о карте с XML файла и получение из них двумерной матрицы объектов;

- создание массива клиентов и массива пуль;

- создание двух потоков: один для подключения новых клиентов, другой для обновления уже подключившихся.

Для обновления положения персонажей сервер должен знать исключительно объектную характеристику карты, тоесть где какой объект находится. Чтобы при подключении нового клиента или появлении новой пули не создавать объект, а при отключении клиента или взрыва пули не удалять его, можно использовать массив созданных объектов, переинициализируя его вместо создания. Подключение новых клиентов может затормозить обновление играющих, особенно если высылаются файлы. По этой причине для каждой задачи создан отдельный поток.

КАРТИНКУ СЮДА НУЖНО

## 3.3 Проектирование клиентской части

Инициализация клиента содержит следующие этапы:

- считывание данных об анимациях персонажа и пули с XML файла;

- загрузка картинки для шкалы здоровья;

- создание массива игроков и массива пуль.

На стороне клиента также как и на сервере необходимо заранее подготовить два массива для хранения информации об игроках и пулях. После подключения к серверу загружается карта.

## 3.4 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.

4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Языком программирования программного средства был выбран язык C++. Данный язык широко используется при разработке игр, поскольку позволяет повысить производительость, сравнительно с другими языками. Дополнительно я использовал стороннюю библиотеку SFML-2.5.1 [2], чтобы упростить работу с созданием окна, прорисовкой карты и аминаций, а также проигрыванием музыки и звуков. В этой библиотеки также есть модуль для работы с сетями. Для считывания XML файлов я подключил модуль с парсером TinyXML-2 [3]. В качестве среды разработки была выбрана среда Visual Studio 2022.

Чтобы структурировать код серверная часть была поделена на следующие модули:

- Consts. Здесь объявлены необходимые константы, такие как имена требуемых файлов, размеры массивов клиентов и пуль, размеры игрока и пули как объектов, коды состояний игрока и коды используемых в игре клавиш. Переменные, определяющие скорость передвижения, коэффициент g и прочие переменные, которые характеризуют движение объекта, было решено хранить как поле класса, так как в будущем могут быть добывлены бонусы, позволяющие ускорять передвижение объекта или изменять гравитацию;

- Level. Здесь объявлен класс, содержащий информацию о карте. Конструктор с помощью парсера TinyXML-2 считывает информацию о карте и создаёт двумерную матрицу объектов;

- Bullet. Здесь объявлен класс, который характеризует пулю. В этом классе реализован метод, обрабатывающий коллизию пули с объектами карты и игроками;

- Player. Здесь объявлен класс, который характеризует игрока. В этом классе реализованы методы переинициализации, обновления состояния игрока и обработки коллизии с картой;

- Client. Здесь объявлен класс, который характеризует клиента. Одним из его полей является класс Player. В этом классе реализован метод, который формирует пакеты с текущим состоянием игрока для передачи другим, и метод для распаковки принятого пакета;

- main. Здесь происходит инициализация и создаются два потока (для подключения новых клиентов и для обновления уже подключившихся). Для каждого из потоков реализована своя функция.

Для структурирования кода клиентская часть была также поделена на модули:

- Consts. Здесь объявлены те же константы, что и на сервере;

- Level. Здесь объявлен класс, содержащий информацию о карте. Конструктор с помощью парсера TinyXML-2 считывает информацию о карте и создаёт вектор спрайтов, которой используется для отображения карты;

- Animation. Здесь объявлены классы Animation и AnimationManager. Первый содержит в себе векторы с координатами анимаций движения в одну сторону и другую. В этом классе реализован метод, обновляющий анимацию. Второй класс с помощью парсера TinyXML-2 считывает информацию об анимациях и загружает спрайты. Этот класс содержит словарь со значениеми типа Animation. AnimationManager отвечает за проигрывание нужной анимации;

- Bullet. Здесь объявлен класс, который характеризует анимацию пулю. Одним из его полей является класс AnimationManager. В этом классе реализован метод выбора соответствующей состоянию пули анимации;

- Player. Здесь объявлен класс, который характеризует анимацию игрока. Одним из его полей является класс AnimationManager. В этом классе реализован метод выбора соответствующей состоянию игроку анимации;

- HealthBar. Здесь объявлен класс, который визуализирует количество здоровья игрока;

- main. Здесь происходит инициализация и осуществляется подключение к серверу с последующим обновлением состояния игры.

# 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.

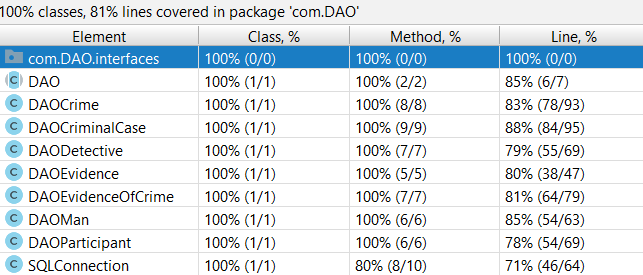


Рисунок 5.1 – Покрытие теста уровня объектов доступа к данным

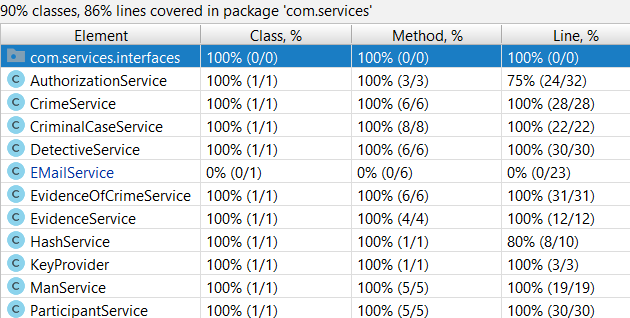


Рисунок 5.2 – Покрытие теста уровня сервисов

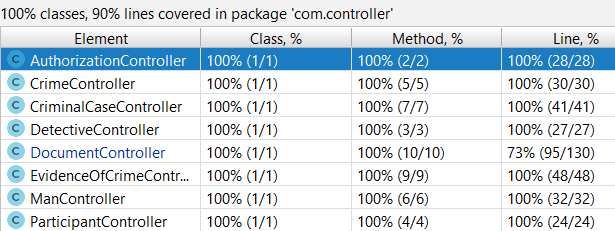


Рисунок 5.3 – Покрытие теста уровня контроллеров

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

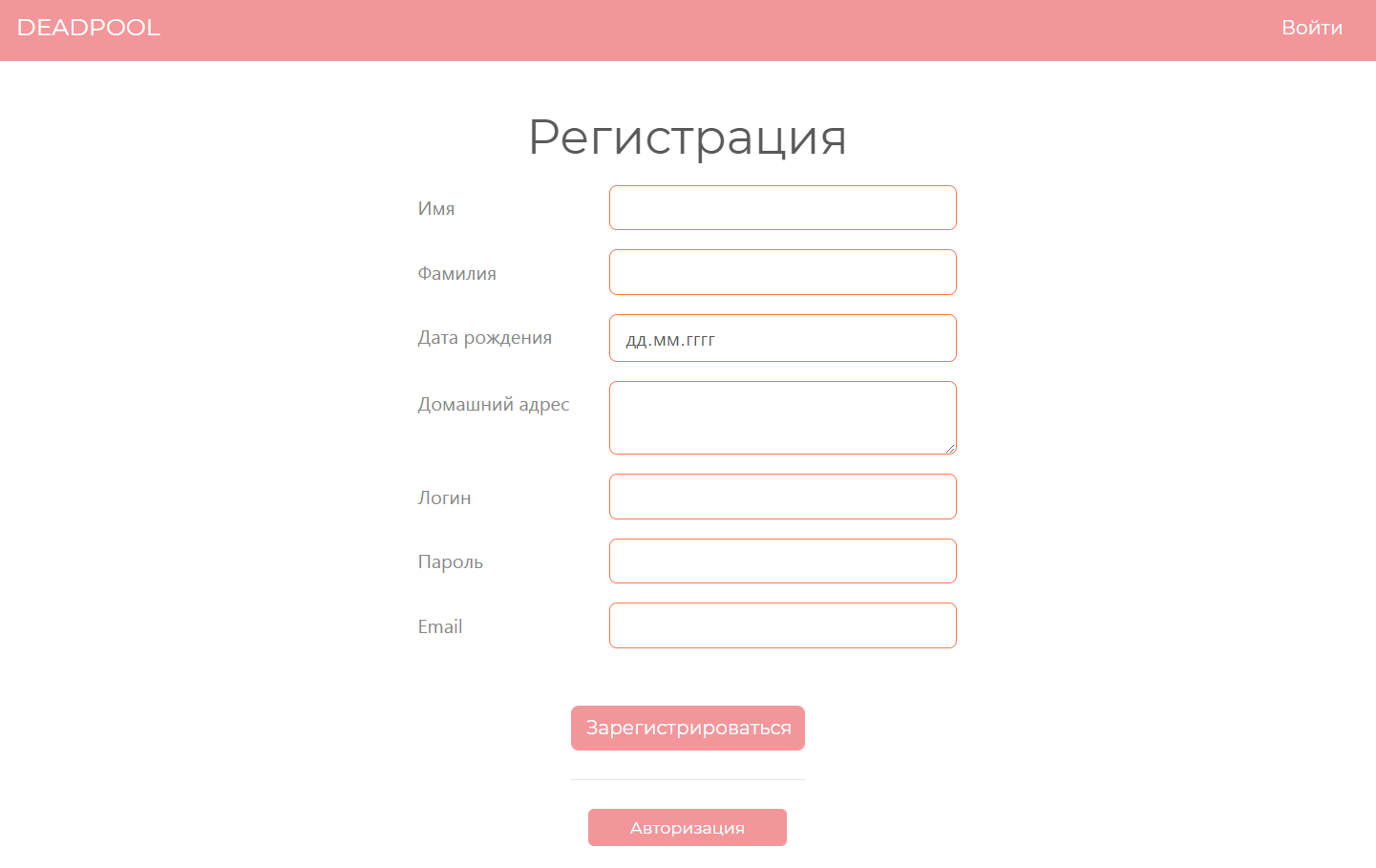


Рисунок 6.2 – Вид страницы регистрации приложения

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

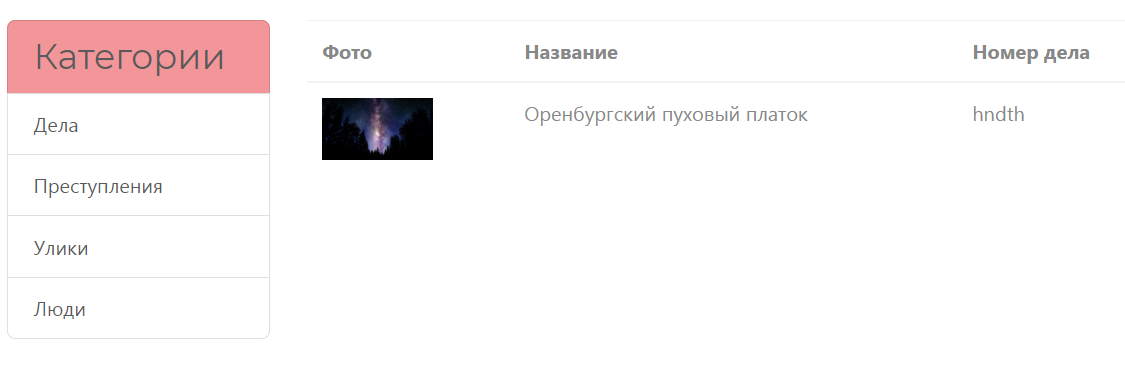


Рисунок 6.4 – Отображение списка улик

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.

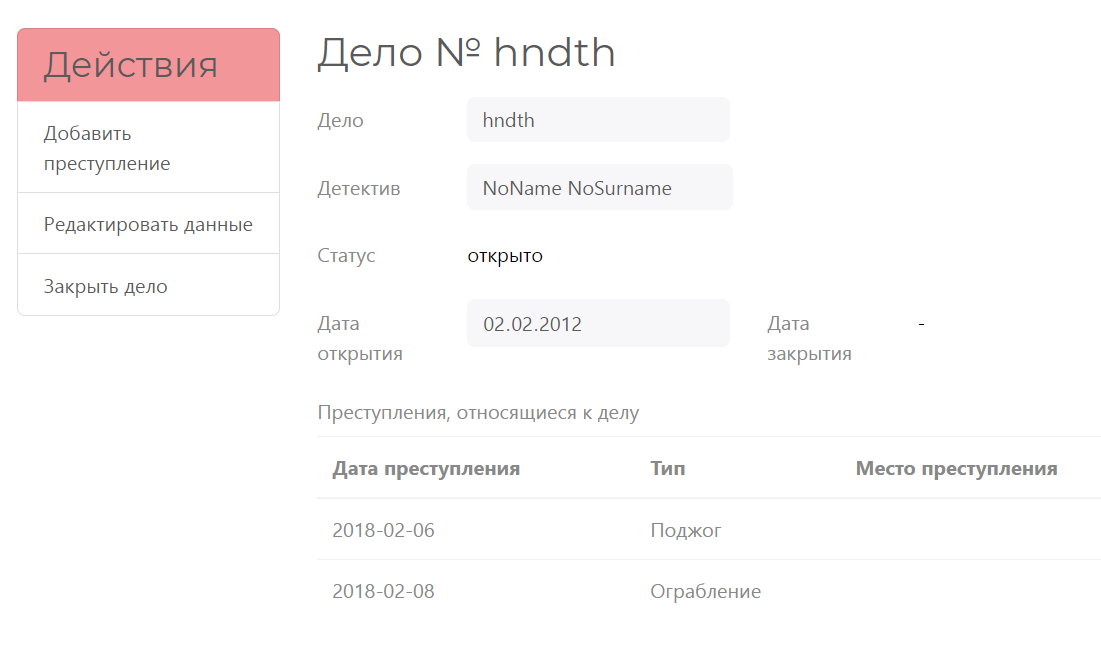


Рисунок 6.5 – Подробное описание дела

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.

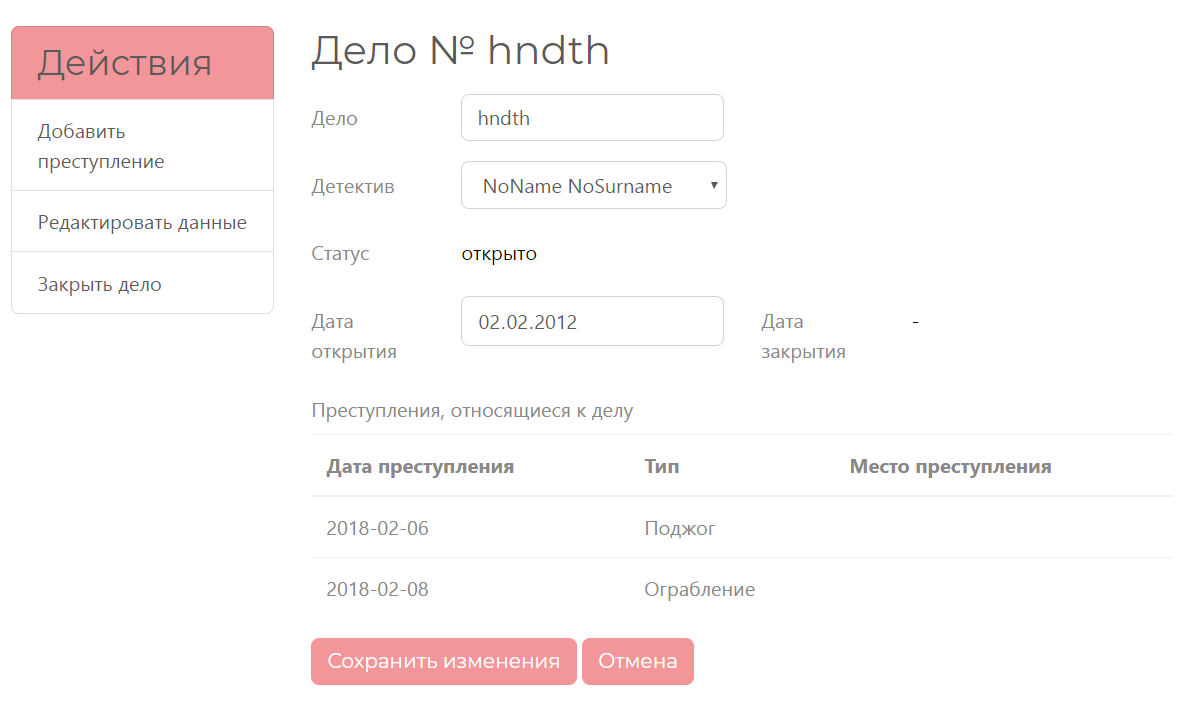


Рисунок 6.6 – Редактирование сведений о деле

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».

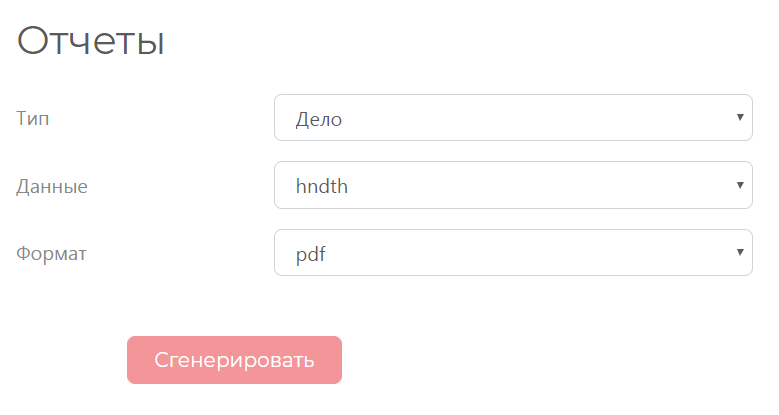


Рисунок 6.7 – Страница генерации отчётов

# 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. <https://youtu.be/0LxyqYV86Fk> [Электронный ресурс] – Как устроен мультиплеер - подробно | Networking.
2. [https://www.sfml-dev.org/documentation/2.5.1](https://www.sfml-dev.org/documentation/2.5.1/) [Электронный ресурс] –Документация по библиотеке SFML-2.5.1.
3. <https://github.com/leethomason/tinyxml2> [Электронный ресурс] – репозиторий с модулями и документацией TinyXML-2.

**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());  
 }  
}