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:

«ПРОГРАММНОЕ СРЕДСТВО

РАСПРЕДЕЛЁННОГО УДАЛЁННОГО УПРАВЛЕНИЯ»

БГУИР КП 1-40 01 01 010 ПЗ

Студент: гр. 351002 Яхновец В.А.

Руководитель: Шамына А.Ю.

Минск 2025

**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 Разработка 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**

В условиях стремительного развития цифровых технологий и широкого распространения удалённого взаимодействия между пользователями и компьютерными системами возрастает потребность в эффективных средствах удалённого мониторинга и управления. Такие программные решения находят применение в различных сферах — от системного администрирования до обеспечения информационной безопасности.

Одним из направлений в данной области является создание программных средств, способных фиксировать и анализировать действия пользователя. Одним из таких инструментов выступает кейлоггер — программа, регистрирующая нажатия клавиш на клавиатуре. Несмотря на то, что кейлоггеры часто ассоциируются с вредоносной деятельностью, их изучение и моделирование позволяют глубже понять потенциальные угрозы, которые могут возникать в корпоративной или личной информационной среде. Разработка собственного безопасного кейлоггера позволяет осознанно изучить механизмы работы подобного рода программ и тем самым повысить уровень защищённости систем от внешнего вмешательства.

В рамках данной работы кейлоггер выступает в качестве одного из модулей распределённого программного средства удалённого управления, демонстрируя возможность централизованного сбора и анализа пользовательской активности на удалённых устройствах.

Кроме исследовательских целей, подобные решения могут использоваться в корпоративной среде для мониторинга действий сотрудников, в образовательных целях — для анализа поведения пользователей, а также в технических экспериментах, направленных на изучение принципов удалённого взаимодействия между компонентами информационной системы.

С учётом вышеизложенного, целью данной курсовой работы является создание программного средства распределённого удалённого управления, способного фиксировать пользовательские действия и передавать их для последующего анализа. Особое внимание при этом уделяется архитектуре системы, обеспечению стабильности её работы и возможностям масштабирования.

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. Разработка 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**

Современные корпоративные и информационные системы всё чаще переходят к распределённой архитектуре с активным использованием средств удалённого мониторинга и управления. Это особенно актуально в условиях масштабирования инфраструктур, перехода к гибридным и облачным средам, а также удалённой работы сотрудников. Программные средства удалённого управления позволяют централизованно отслеживать состояние клиентских устройств, проводить диагностику, оказывать техническую поддержку и при необходимости собирать данные о действиях пользователей.

На рынке существует множество коммерческих решений в этой области — таких как TeamViewer, AnyDesk, Radmin, Ammyy Admin и другие. Однако большинство из них нацелено на визуальное удалённое подключение и поддержку, а не на автоматический сбор информации о действиях пользователя. Кроме того, открытые или бесплатные решения редко предоставляют гибкость и расширяемость, необходимую для встроенного мониторинга, передачи данных по расписанию и глубокого взаимодействия между клиентом и сервером.

Кейлоггеры, несмотря на негативную репутацию, являются важным объектом изучения в области информационной безопасности. Понимание принципов работы таких программ позволяет выявлять уязвимости, улучшать системы защиты и разрабатывать методы противодействия вредоносным инструментам. Анализ деятельности кейлоггеров особенно важен для специалистов, работающих в области кибербезопасности, администрирования корпоративных сетей и разработки защитного программного обеспечения.

Поскольку большинство кейлоггеров и аналогичных программ относятся к категории вредоносного ПО, их архитектура и принципы работы не освещаются в официальных источниках. Это затрудняет исследование существующих решений. Научные публикации в данной области, как правило, ограничены теоретическим описанием угроз, без углублённого анализа реализации средств мониторинга.

Таким образом, разработка и моделирование безопасного распределённого программного средства удалённого управления, включающего модуль кейлоггера, актуальны как с исследовательской, так и с практической точки зрения. Это позволяет с одной стороны — повысить осведомлённость о возможных угрозах, а с другой — продемонстрировать, как подобные решения могут быть реализованы в рамках правомерных задач, например, для изучения пользовательского поведения, технической диагностики или обеспечения внутренней безопасности корпоративных систем.

Разработка собственного решения также позволяет детально проанализировать механизмы взаимодействия между клиентом и сервером, оценить особенности работы в сетевой среде, рассмотреть архитектуру распределённых систем и протестировать различные подходы к передаче, хранению и визуализации данных.

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

Целью разрабатываемого программного средства является создание распределённой системы удалённого контроля, предназначенной для централизованного сбора, хранения и анализа данных, поступающих с пользовательских устройств, на которых установлен агент-наблюдатель — кейлоггер. Система должна обеспечивать автоматизированный сбор информации о действиях пользователей, её безопасную передачу на центральный сервер, а также предоставлять веб-интерфейс для мониторинга и управления полученными данными.

Ключевым элементом системы является программный модуль-кейлоггер. Он устанавливается на целевых пользовательских устройствах и выполняет непрерывную фиксацию всех нажатий клавиш. При этом он должен сохранять не только последовательность нажатых клавиш, но и сопутствующую контекстную информацию: название и идентификатор активного приложения, в котором производился ввод, а также точное время и дату каждого действия. Данные должны накапливаться в виде текстовых файлов и сохраняться локально на устройстве до момента передачи на сервер.

Передача информации от кейлоггера на сервер должна происходить автоматически по заданному расписанию (например, каждые N минут), а также по специальному событию — например, при значительном накоплении данных или по прямому запросу со стороны сервера. Для реализации более гибкого обмена данными необходимо предусмотреть двустороннюю синхронизацию: сервер должен иметь возможность инициировать досрочную отправку логов, не дожидаясь следующего планового запроса от клиента.

Серверная часть программного комплекса должна представлять собой REST API-приложение, обрабатывающее входящие запросы, сохраняющее переданную информацию и предоставляющее интерфейс для доступа к этим данным. В целях простоты и отказоустойчивости данные на сервере должны храниться в виде файловой системы: каждому устройству соответствует отдельный файл или каталог, содержащий переданные с него логи.

Для взаимодействия с системой пользователей предоставляется фронтенд-интерфейс. Интерфейс должен обеспечивать удобный просмотр собранных логов. Также должна быть предусмотрена реализация базовых операций управления: запрос обновлённой информации от устройства, обновление или пересоздание списка логов, удаление конкретного лога. Кроме того, интерфейс должен отображать структуру и статус всех подключённых устройств, чтобы администратор мог отслеживать их активность и своевременно обнаруживать неполадки или отключения.

Интерфейс должен быть кроссбраузерным и адаптированным под все современные устройства, включая ПК, планшеты и смартфоны.

Кроме того, программное средство должно обладать высокой степенью расширяемости: архитектура приложения должна позволять легко добавлять новые модули и функции, такие как поддержка различных типов агентов (например, скриншотеров или мониторинга сети), а также внедрение баз данных на следующем этапе развития проекта.

Таким образом, разрабатываемая система представляет собой многоуровневое распределённое решение, ориентированное на безопасный сбор, централизованное хранение и удобный анализ пользовательской активности в рамках удалённого мониторинга.

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

Исходя из поставленной задачи и сформулированных требований, разрабатываемое программное средство должно обеспечивать выполнение следующих функций:

* осуществлять установку и запуск клиентского агента (кейлоггера) на целевом устройстве с минимальным вмешательством пользователя;
* производить непрерывный сбор данных о вводе с клавиатуры, включая символы, время и дату нажатий, а также имя и идентификатор активного окна, в котором происходил ввод;
* сохранять зафиксированные данные в локальных текстовых файлах на клиентском устройстве в структурированном виде;
* выполнять передачу логов на серверную часть по заданному расписанию (например, каждые N минут);
* обеспечивать возможность серверной части запрашивать у кейлоггера актуальные данные в реальном времени без ожидания плановой отправки;
* сохранять полученные от клиентов данные на сервере в виде иерархической файловой структуры, где каждой машине соответствует отдельный файл или каталог;
* предоставлять REST API-интерфейс для обработки входящих запросов от клиента, а также для взаимодействия с фронтенд-интерфейсом;
* реализовать защищённый и адаптивный веб-интерфейс для администратора системы, позволяющий:
* просматривать структуру всех подключённых устройств;
* анализировать логи за выбранный период;
* инициировать отправку новых данных с устройства;
* удалять отдельные логи;
* отслеживать статус подключения клиентов и общее состояние системы;
* обеспечить кроссбраузерную совместимость веб-интерфейса, а также его корректную работу на ПК, планшетах и мобильных устройствах;
* обеспечить возможность расширения системы в будущем — за счёт внедрения дополнительных клиентских модулей, а также подключения СУБД для хранения и анализа данных.

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Проектирование программного средства начинается с определения его общей архитектуры, исходя из требований надёжности, расширяемости, удобства сопровождения и обеспечения кроссплатформенности. Разрабатываемое программное средство предназначено для удалённого контроля пользовательских действий и должно включать в себя как средства сбора информации, так и средства её отображения и администрирования.

Система условно разделена на две основные части: клиентскую и серверную. Такая архитектура обеспечивает модульность, что позволяет независимо обновлять или масштабировать компоненты, не затрагивая всю систему в целом. Это решение также повышает надёжность и гибкость при внедрении изменений и новых функций.

Серверная часть отвечает за хранение, обработку и предоставление данных, поступающих от агентов, установленных на пользовательских устройствах. Также сервер обрабатывает запросы от клиентского интерфейса и реализует логику взаимодействия с файловой системой, включая работу с логами. Для реализации серверной логики применяется REST-подход, при котором каждая операция доступна через отдельный HTTP-метод (GET, POST, DELETE). Такой подход обеспечивает простоту и универсальность взаимодействия между компонентами.

Также серверная часть развёрнута на устройстве с постоянным (статическим) IP-адресом. Для обеспечения доступа к серверу из внешней сети настроен проброс портов через маршрутизатор. Это позволяет агенту (кейлоггеру) и административной панели взаимодействовать с сервером, независимо от их физического местоположения, что критически важно для системы удалённого контроля.

Клиентская часть, выполняющая роль административной панели, реализуется в виде одностраничного веб-приложения (SPA). Выбор такого подхода обусловлен необходимостью быстрой и динамичной работы интерфейса, особенно при регулярной загрузке новых данных. Благодаря использованию технологий AJAX, клиент может асинхронно обмениваться данными с сервером без перезагрузки страницы, что повышает скорость работы.

Обмен данными между клиентом и сервером осуществляется в формате JSON, поскольку этот формат легко интегрируется в веб-технологии, экономит трафик и обладает хорошей читаемостью. В отличие от XML, JSON содержит меньше избыточной информации и обрабатывается быстрее, что особенно важно при регулярной передаче больших объёмов логов.

Архитектура программного средства также предполагает наличие внутреннего агента (кейлоггера), который работает на пользовательском устройстве и передаёт информацию на сервер. В рамках проектирования определены форматы логов, механизмы их хранения и расписание передачи данных, включая возможность принудительной отправки через систему лонг-пуллинга. Для повышения устойчивости к обнаружению со стороны антивирусного ПО кейлоггер перед отправкой данных случайным образом подбирает HTTP-заголовки, имитируя обычную сетевую активность, а также применяет кодирование содержимого логов в формате Base64. Это позволяет маскировать передаваемую информацию под легитимный трафик и минимизирует риск блокировки со стороны систем защиты.

Сам агент кейлоггера разделён на два логически независимых компонента: модуль регистрации нажатий клавиш и модуль взаимодействия с сервером. Первый компонент занимается непосредственным отслеживанием пользовательской активности и сохранением информации в локальный файл. Второй компонент, выступающий в роли клиента, отслеживает состояние этого файла, осуществляет чтение накопленных данных и их отправку на сервер. Такое разделение позволяет изолировать процесс логирования от сетевого взаимодействия, повышая надёжность и устойчивость системы к сбоям в соединении, а также упрощает последующую модификацию и сопровождение отдельных компонентов.

В итоге, архитектура программного средства представляет собой распределённую систему, в которой каждый компонент выполняет строго определённую роль. Это упрощает поддержку, тестирование и модификацию системы в дальнейшем.

**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.

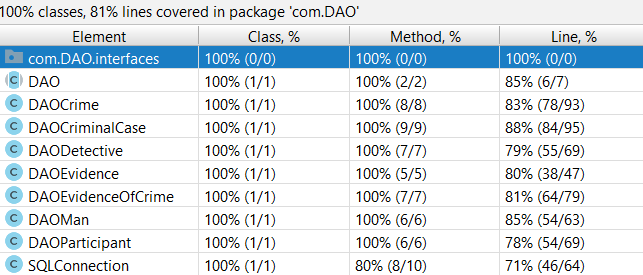


Рисунок 5.1 – Покрытие теста уровня объектов доступа к данным

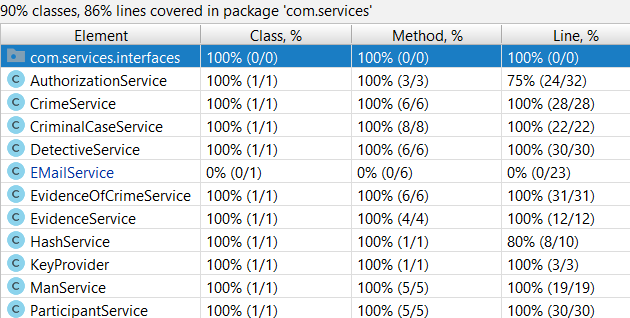


Рисунок 5.2 – Покрытие теста уровня сервисов

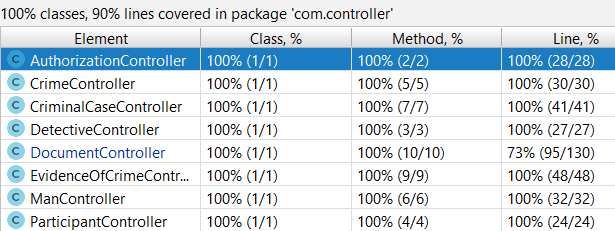


Рисунок 5.3 – Покрытие теста уровня контроллеров

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).

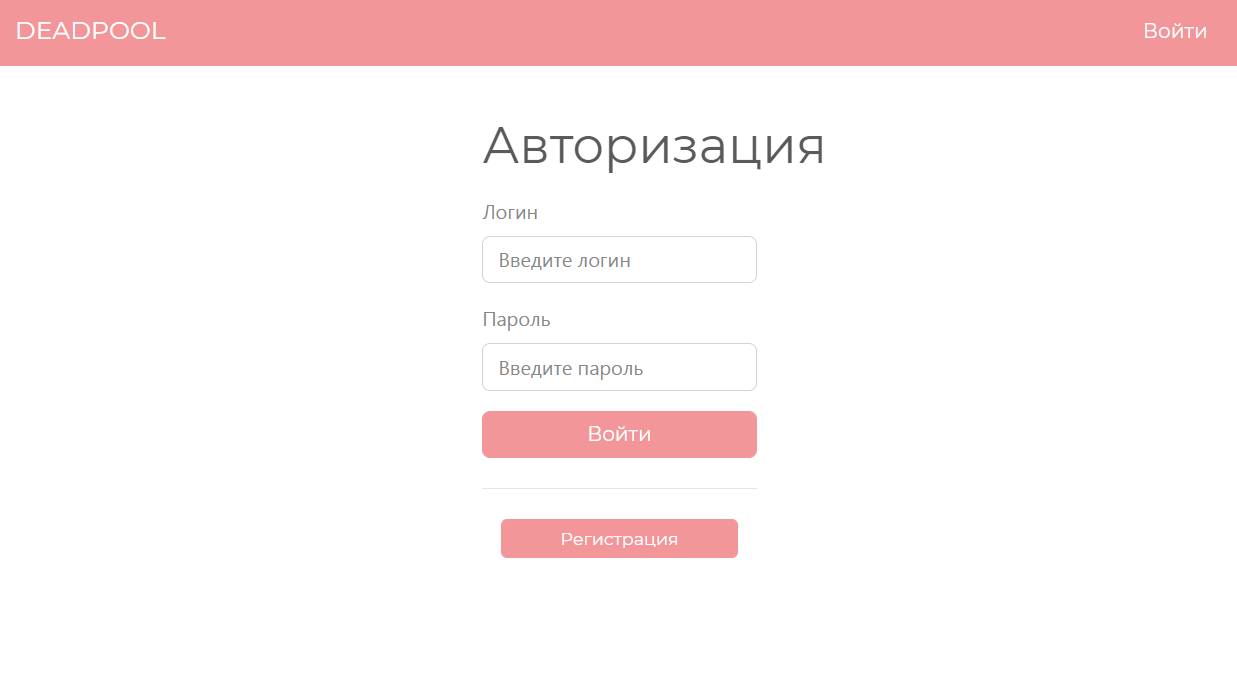


Рисунок 6.1 – Вид страницы авторизации приложения

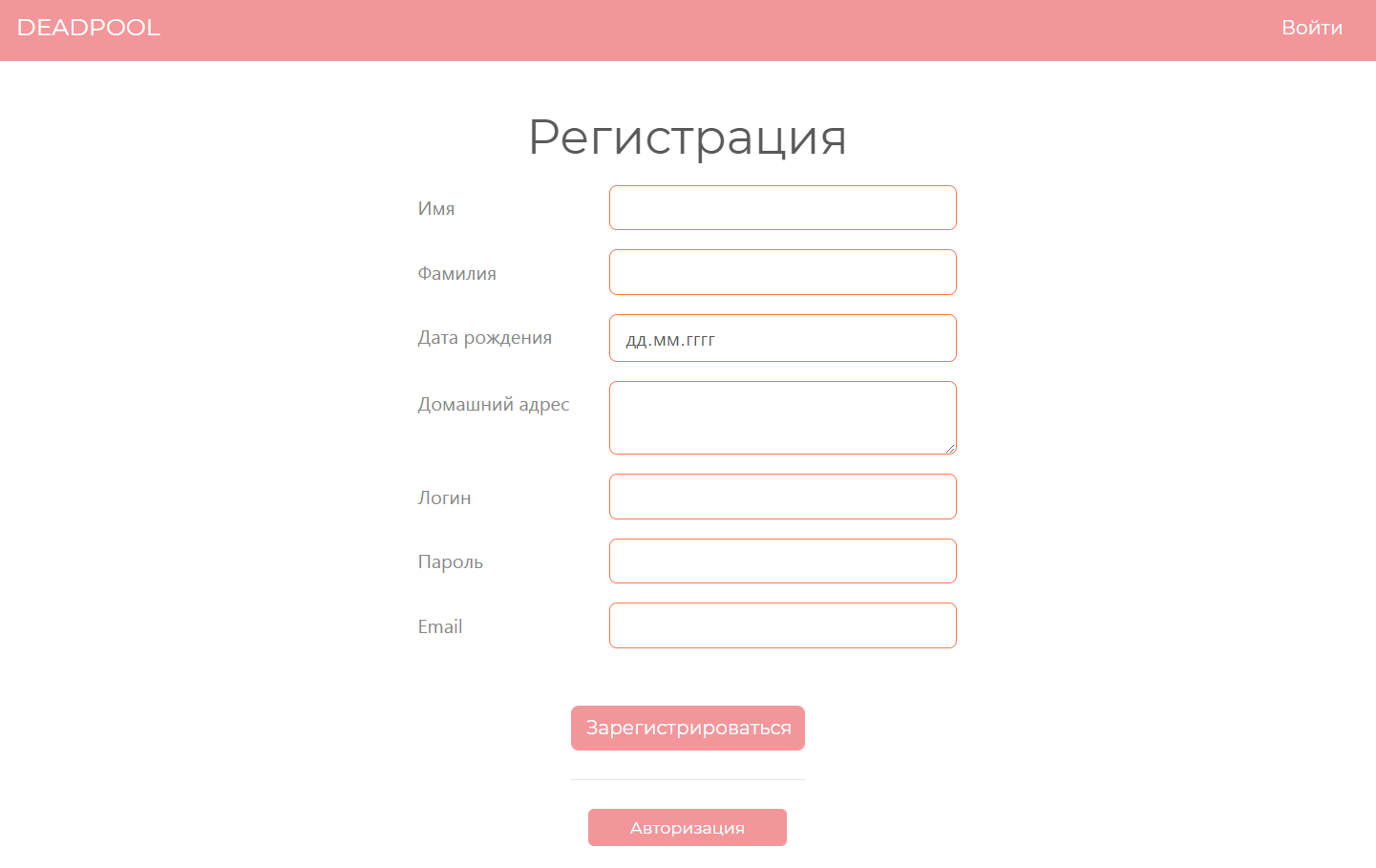


Рисунок 6.2 – Вид страницы регистрации приложения

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

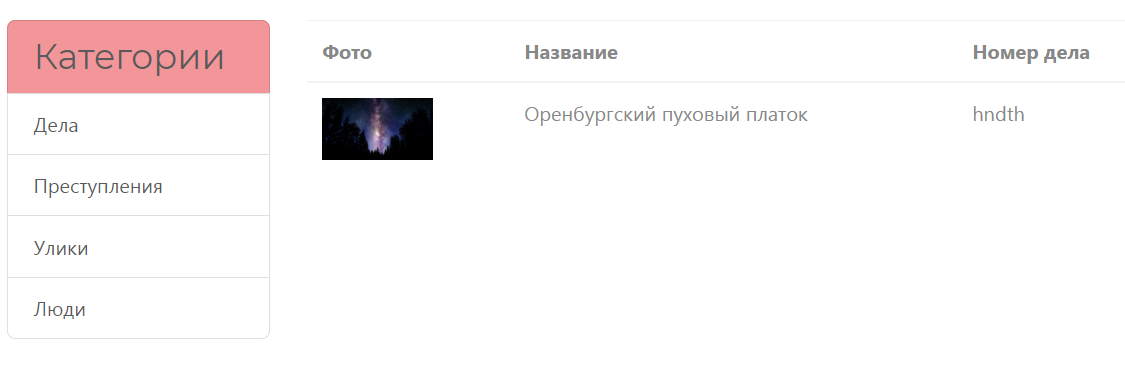


Рисунок 6.4 – Отображение списка улик

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.

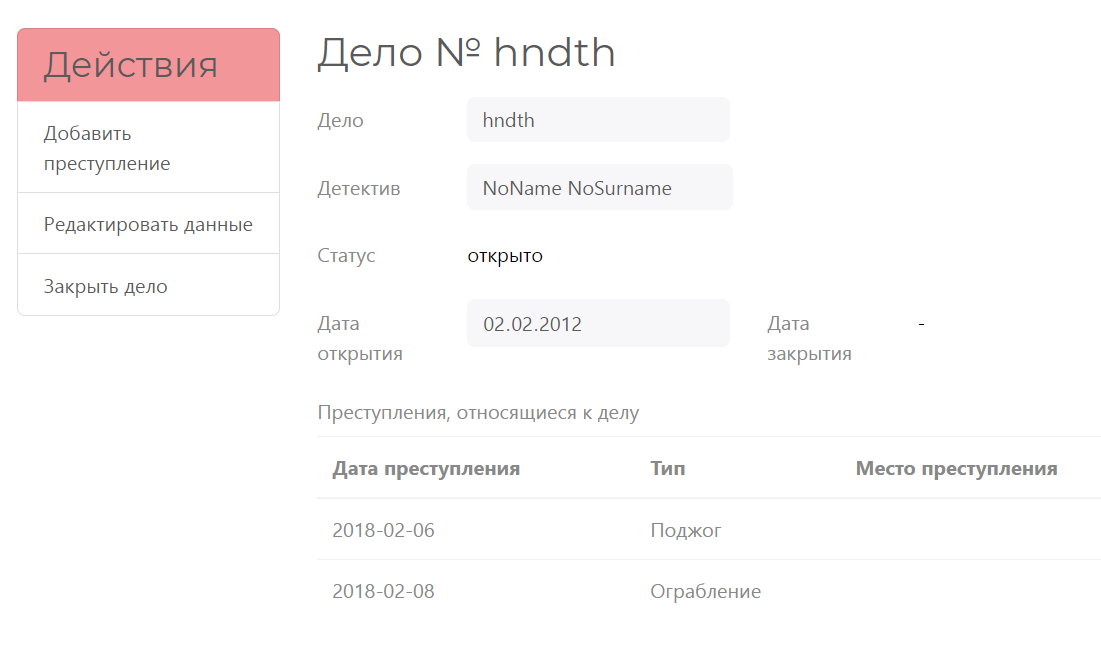


Рисунок 6.5 – Подробное описание дела

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.

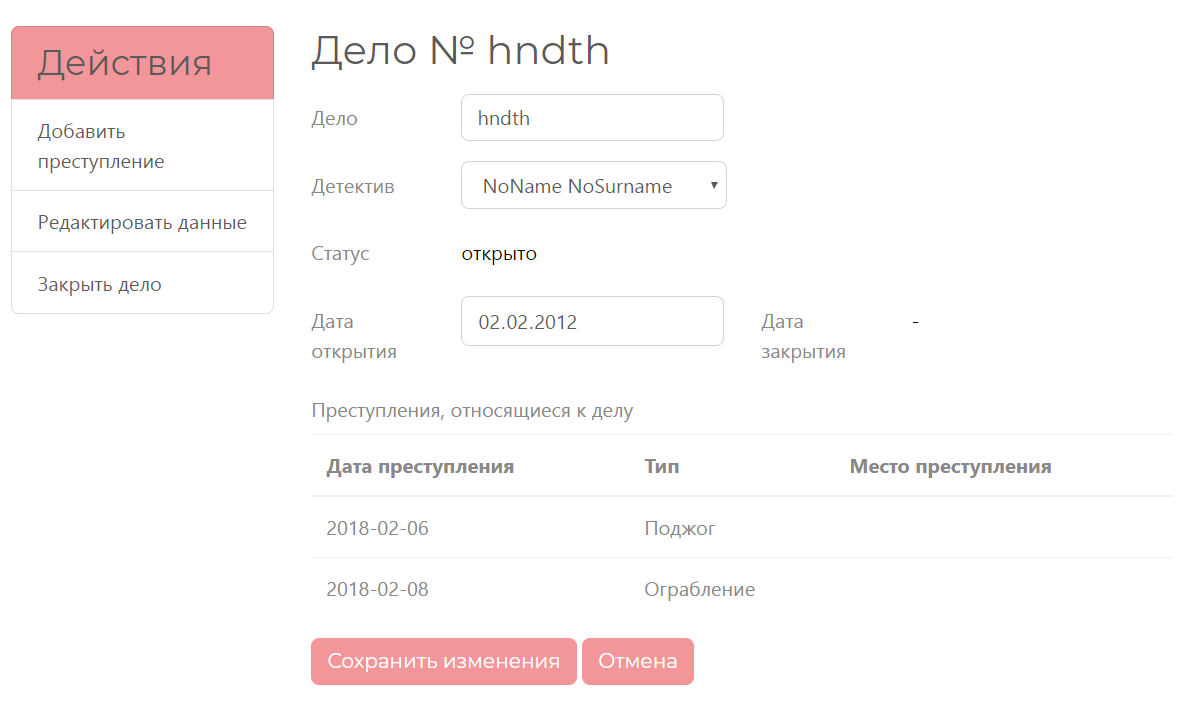


Рисунок 6.6 – Редактирование сведений о деле

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».

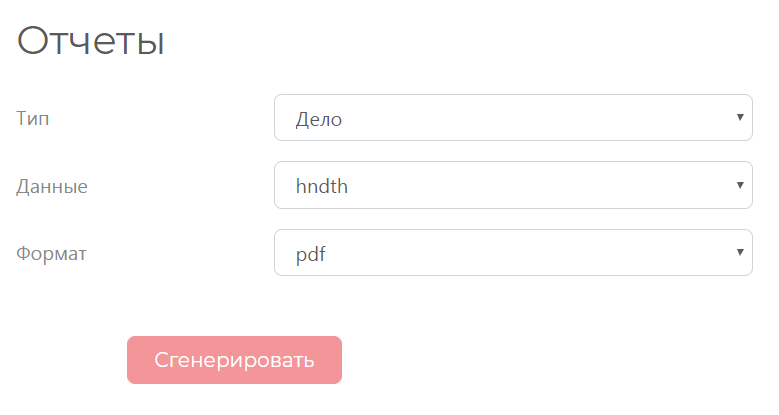


Рисунок 6.7 – Страница генерации отчётов

**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 | | | | | |
|  | |  |  |  |