МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 Программное обеспечение информационной безопасности

мобильных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Электронный документооборот»

Выполнил студент Фандо Евгений Александрович

(Ф.И.О.)

Руководитель проекта ассистент Колмаков М.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов Владимир Владиславович

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой .

Минск 2020

**Содержание**

[Введение](#_Toc42220264) 3

[1. Обзор аналогов приложения](#_Toc42220265) 4

[2. Проектирование программного продукта](#_Toc42220269) 7

[2.1 Проектирование сервера](#_Toc42220270) 7

[2.2 Проектирование мобильного приложения](#_Toc42220271) 7

[2.3 Проектирование веб-приложения](#_Toc42220272) 8

[2.4 Проектирование базы данных](#_Toc42220273) 8

[2.5 Проектирование FTP сервера 1](#_Toc42220273)0

[2.6 Выводы по главе 1](#_Toc42220273)2

[3 Реализация программного средства 1](#_Toc42220274)2

[3.1 Реализация сервера 1](#_Toc42220274)2

[3.1.1 Постановка задачи 1](#_Toc42220274)2

[3.1.2 Используемые зависимости и библиотеки 1](#_Toc42220274)2

[3.1.3 Spring Boot 1](#_Toc42220274)2

[3.1.4 Spring MVC 1](#_Toc42220274)3

[3.1.5 Spring Data 1](#_Toc42220274)4

[3.1.6 Spring Integration 1](#_Toc42220274)5

[3.1.7 Spring Security 1](#_Toc42220274)5

[3.1.8 Hibernate 1](#_Toc42220274)6

[3.1.9 JWT 1](#_Toc42220274)7

[3.1.10 MySql Driver 1](#_Toc42220274)7

[3.1.11 Lombok 1](#_Toc42220274)7

[3.1.12 Rest API 1](#_Toc42220274)8

[3.2 Реализация WEB клиента 1](#_Toc42220274)8

[3.2.1 Инструмент для сборки Webpack 2](#_Toc42220274)1

[3.3 Реализация для сборки мобильного клиента 2](#_Toc42220274)1

[3.4 Реализация FTP сервера 2](#_Toc42220274)2

[3.5 Вывод по разделу 2](#_Toc42220274)2

[4 Руководство пользователя 2](#_Toc42220274)3

[Заключение 3](#_Toc42220274)2

[Список литературы 3](#_Toc42220274)4

[Приложение А 3](#_Toc42220274)5

[Приложение Б 3](#_Toc42220274)6

# Введение

Курсовой проект посвящен разработке программного средства «FunDocker Electronic Documents» , предназначенного для работы мобильного приложения и веб ресурса с сервером и удаленной базой данных.

В данный момент в постоянно меняющемся мире, где каждый день может преподнести что-то новое самым важным становиться то, как мы можем обезопасить себя и своих близких. На сегодняшний день, из-за пандемии, проблема безопасности состоит в очень большом круге ежедневных вынужденных контактов с другими людьми, что в свою очередь повышает риски заражения и распространения вируса.

Данный курсовой проект берет на себя функции эксплуатации и подписи электронных документов, которые после могут быть использованы как юридически оформленный документ на бумаге. При активном использовании этого приложения будет снижены не только трудозатраты на собрание, оформление и подпись документов, что уже немаловажно в век, когда нужно идти в ногу со временем и не отставать ни на шаг, и так же уменьшит число вынужденных контактов и риски, связанные с ними.

Целью моего курсового проекта является создание приложения в котором будет возможность создавать организации и добавлять новые документы, которые можно будет подписать и в последствии использовать.

Задачи курсового проекта:

* Изучить аналогичные программные средства по теме курсового проекта.
* Спроектировать приложение.
* Реализовать приложение.
* Протестировать приложение.
* Разработать руководство пользователя по управлению бортовым журналом.

# 1 Обзор аналогов приложения

В приложении представлены возможности добавления своего заполненного документа pdf для того чтобы поставить свою электронную подпись или предложить поставить ее другому человеку, при этом создав нотификацию, которая придет в тот момент, когда человек поставит подпись на вашем документе. Веб-приложение представлено ниже на скриншотах на Рисунке 1.1.

|  |  |  |
| --- | --- | --- |
|  |  |  |

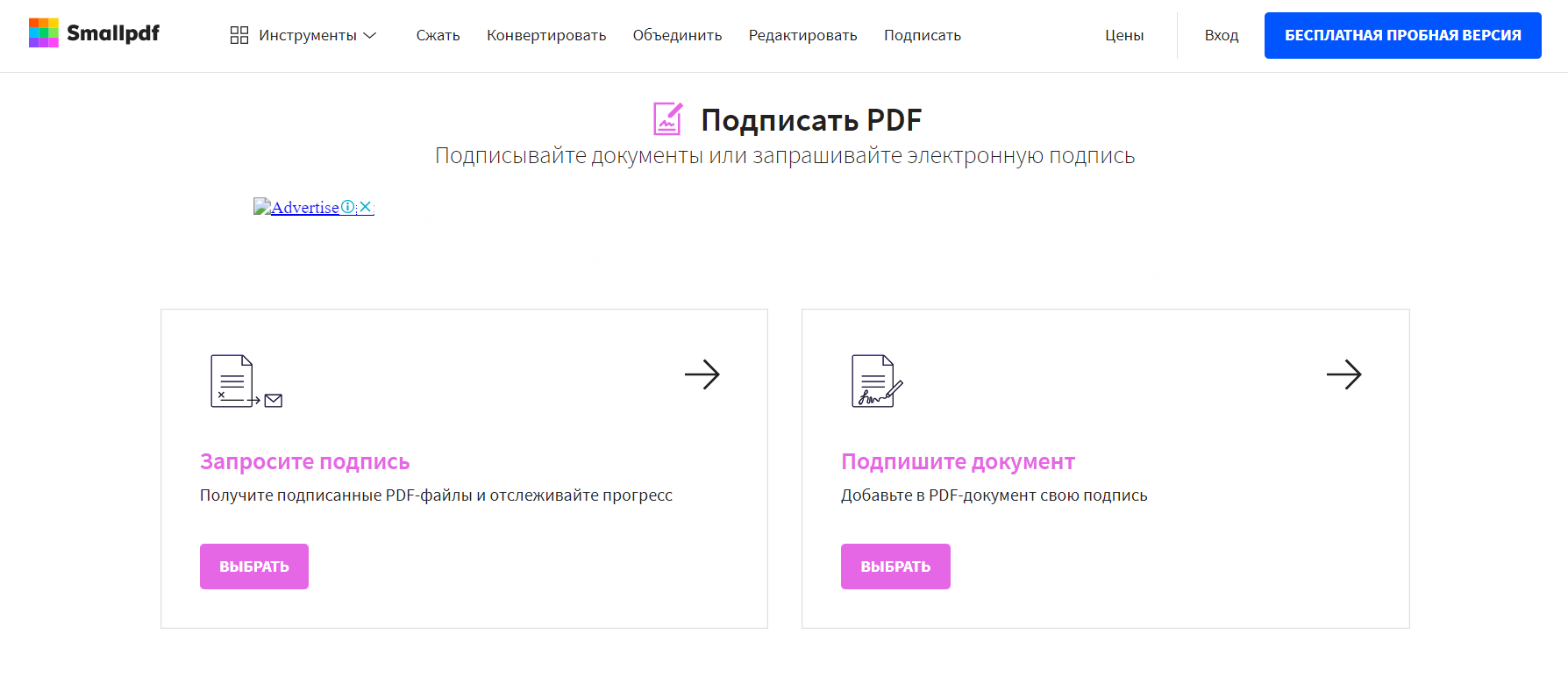


Рисунок 1.1 – Приложение «Моя машина»

Веб-приложение предоставляет удобные возможности для подписи и многочисленный выбор в вариативности сигнатуры, которая будет установлена в pdf файл.

«SignEasy»

Главной целью данного приложения является простой и быстрый способ поставить свою или получить чужую подпись на документах с помощью телефона или планшета Android. Этому приложению доверяют более 5 млн пользователей.

Мобильное приложение представлено на скриншоnt ниже на Рисуноке 1.2.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рисунок 1.2 –Приложение «SignEasy»

Главными особенностями данного приложения являются удобные возможности для подписи и многочисленный выбор в вариативности сигнатуры, которая будет установлена во множество различных типах файлов.

### 1.1 Выводы по главе

Были рассмотрены веб и мобильное приложение, выполняющие функции добавления и хранения документов, которые будут подписаны самим пользователем или любым другим пользователем, которому будет дано такое разрешение правообладателя документа. В результате по каждому приложению были выделены положительные моменты, а также недостатки.

Первое приложение называется «Smallpdf». Из положительных моментов следует выделить:

* дизайн в светлых тонах с яркими элементами
* возможность выбрать свою тему оформления
* добавление своих сигнатур
* синхронизация с облачным хранением

Из недостатков данного приложения можно выделить:

* неудобный пользовательский интерфейс
* при выборе другого цвета оформления, некоторые компоненты не сочетаются между собой
* статистика по подписям доступна только за текущий месяц
* количество документов в бесплатном варианте, которые можно добавить, ограничено

Следующее из рассмотренных приложений называется «SignEasy». Из положительных моментов можно выделить:

* интуитивно понятный пользовательский интерфейс
* возможность добавления своей сигнатуры для подписи, как картинки, так и от руки на дисплее
* большой функционал
* работа в офлайн режиме
* автоматическое сохранение изменений
* возможность добавления своих категорий, если подходящих нет

Из недостатков этого приложения можно выделить:

* сильная перегруженность интерфейса

# 2 Проектирование программного продукта

### 2.1 Проектирование сервера

Для синхронизации двух и более клиентов, а так же хранения их информации на удалённом сервере необходимо создать серверное приложение, которое предоставит удобное API для пользования. Передаваемые или получаемые данные от сервера должны передаваться в формате JSON с помощью REST API.

Серверное приложение должно быть построено на основе архитектурного стиля REST. Сервер должен иметь чистую архитектуру, для легкого масштабирования проекта и повторного использования кода. Обращение к базе данных должно быть реализовано через репозитории.

При обращении к серверу по заданному маршруту и определенному HTTP методу сервер будет определять с какой сущностью базы данных необходимо взаимодействовать и какие действия над данными необходимо будет произвести.

При помощи HTTP метода GET сервер должен возвращать запрашиваемые данные клиенту, ссылаясь на аргументы в запросе, как на условие для выборки данных. Метод POST в свою очередь должен позволять передать данные на сервер для последующей обработки. Метод PUT – обновить соответствующие данные, переданные с клиента. Метод DELETE – удалять с базы данных заданную запись.

### 2.2 Проектирование мобильного приложения

В рамках курсового проекта необходимо разработать мобильное приложение позволяющее пользователю создавать, редактировать организации, в которые можно будет вступать другим пользователям приложения, выкладывать в организацию документы на подпись другим участникам организации. Разработать оптимальный алгоритм скачивания и проверки документа на наличие его в памяти телефона пользователей, так же организовать работу в режиме офлайн. Для пользователей необходимо реализовать возможность редактирования своих профилей и профилей организации, в которых у них есть на это права.

На главной странице приложение должно показывать активные события по изменению ролей пользователя в организациях за последние 24 часа. Также на этой странице должен быть предоставлен доступ к главному меню.

Необходимо разработать страницы:

* Профиль;
* Документы;
* Организации;

Таким образом, приложение должно выполнять функции:

* Предоставлять профиль активного пользователя и так же возможность редактирования его;
* Возможность создания организации;
* Редактирование информации о организации, при наличии на это прав;
* Принимать запросы в организации на вступление, при наличии определенных прав;
* Просматривать все документы организации;
* Добавить свой документ в организацию;
* Создать запрос на подписание своих добавленных документов в организации по роли;
* Возможность подписать документ своей ЭЦП;
* Принять или сделать предложение на вступление в организацию;

### 2.3 Проектирование Веб-Приложения

В рамках курсового проекта необходимо разработать веб-приложение позволяющее пользователю создавать, редактировать организации, в которые можно будет вступать другим пользователям приложения, выкладывать в организацию документы на подпись другим участникам организации. Разработать оптимальный алгоритм скачивания. Для пользователей необходимо реализовать возможность редактирования своих профилей и профилей организации, в которых у них есть на это права.

На главной странице приложение должно показывать активные события по изменению ролей пользователя в организациях за последние 24 часа. Также на этой странице должен быть предоставлен доступ к главному меню.

Необходимо разработать страницы:

* Профиль;
* Документы;
* Организации;

Таким образом, приложение должно выполнять функции:

* Предоставлять профиль активного пользователя и так же возможность редактирования его;
* Возможность создания организации;
* Редактирование информации о организации, при наличии на это прав;
* Принимать запросы в организации на вступление, при наличии определенных прав;
* Просматривать все документы организации;
* Добавить свой документ в организацию;
* Создать запрос на подписание своих добавленных документов в организации по роли;
* Возможность подписать документ своей ЭЦП;
* Принять или сделать предложение на вступление в организацию;

### 2.4 Проектирование базы данных

В ходе выполнения курсового проекта следует создать базу данных, которая будет содержать все сущности перечисленные ниже:

Таблица 2.1 – Отношения базы данных

|  |  |
| --- | --- |
| Наименование таблицы | Назначение |
| organizations | Информация о организациях |
| organization\_roles | Информация о ролях в организации |
| documents | Информация о документах |
| users | Информация о пользователях |
| notifications | Информация о оповещениях |
| offers | Информация о предложениях |
| users\_organizations | Реализация многие ко многим users и organizations |
| users\_organization\_roles | Реализация многие ко многим users и organization\_roles |
| progress\_documents\_response | Реализация многие ко многим users и documents |

Сущность users:

* id – первичный ключ целочисленное значение;
* first\_name – имя пользователя;
* middle\_name ‒ отчество пользователя;
* last\_name – фамилия пользователя;
* username – логин для авторизации;
* email – адрес электронного почтового ящика пользователя;
* password – пароль;

Сущность organizatoins:

* id – первичный ключ целочисленное значение;
* name – название организации;
* owner\_id – идентификатор создателя орагнизции;
* type – тип организации;

Сущность organization\_roles:

* id – первичный ключ целочисленное значение;
* role\_name – название роли;
* organization\_id – идентификатор организации;

Сущность documents:

* id – первичный ключ целочисленное значение;
* organization\_id – идентификатор организации;
* owner\_id – идентификатор отправителя документа;
* path – путь в файловом хранилище;
* state – состояние документа;
* name – наименование документа;
* answer – ;

Сущность notifications:

* id – первичный ключ целочисленное значение;
* type – тип оповещения;
* message – отображаемый текст оповещения;
* user\_id – идентификатор пользователя, к которому это оповещение относится;
* link – часть ссылки для перехода к сущности оповещения;

Сущность offers:

* id – первичный ключ целочисленное значение;
* organization\_id – идентификатор организации;
* user\_id – идентификатор пользователя;
* type – тип предложения;

Сущность users\_organizations:

* user\_id –идентификатор пользователя;
* organization\_id – идентификатор организации;

Сущность users\_organization\_roles:

* user\_id – идентификатор пользователя;
* organization\_role\_id – идентификатор организации;

Сущность progress\_documents\_response:

* user\_id – идентификатор пользователя;
* document\_id – идентификатор документа;

Следует отметить, что большинство атрибутов не могут принимать нулевое значение. Каждая из таблиц имеет ограничение по первичному ключу. База данных имеет одну предопределённую роль, которая является суперпользователем.

### 2.5 Проектирование FTP сервера

Ftp сервер должен реализовывать всего несколько возможностей:

* соединение по sftp протоколу;
* загрузку и выгрузку файлов;
* удаление и переименование файлов;
* раздельное хранение документов пользователей в организациях.

### 2.6 Выводы по главе «Проектирование программного продукта»

В данной главе было рассмотрено проектирование каждого отдельного компонента приложения.

По части мобильного и веб приложения был определен основной функционал приложения.

Была также спроектирована база данных, где были определены необходимые таблицы, поля для каждой из них, определены типы данных и ограничения целостности. Также были установлены связи между таблицами, которые описаны на схеме базы данных.

При проектировании серверного приложения был определен архитектурный стиль взаимодействия между клиентом и сервером, формат передаваемых данных, и HTTP методы которые будут использованы в работе с серверной частью.

# 3 Реализация программного средства

### 3.1 Реализация сервера

### 3.1.1 Постановка задачи

Основной задачей сервера в приложении является создание единой точки доступа к удаленной реляционной базы данных, однообразия работы как веб-ресурса так и мобильного приложения и вынесения общих задач и проверки на безопасность на отдельный модуль, что в свою очередь разгружает конечное приложение пользователя. При выполнении поставленной цели в курсовом проекте на серверной стороне необходимо решить следующие задачи:

– REST API;

– авторизация пользователя и выделение при этом токена для доступа к ресурсам приложения;

– разработать возможность взаимодействия с удаленной реляционной базой данных;

– разработать возможность взаимодействия с удаленным FTP сервером через sftp протокол;

– спроектировать и разработать функционал управления;

### 3.1.2 Используемые зависимости и бибилиотеки

Сервер реализован на одном из самых популярных фреймворков – Spring. Spring разделен на множество под-проектов, которые используются для решения одной или множества задач сервера, - в этом проекте были использованы:

– Spring Boot;

– Spring MVC;

– Spring Data;

– Spring Integration;

– Spring Security;

Так же немаловажную роль выполняли такие технологии:

– Hibernate;

– JWT;

– MySql Driver;

– Lombok.

Теперь разберем в общем и в целом какую из задач решала каждая из этих зависимостей в курсовом проекте на стороне сервера.

### 3.1.3 Spring Boot

Spring Boot - это проект, построенный по принципу convention-over-configuration (соглашения по конфигурации), призванный упросить создание приложений на базе Spring Framework. Решение предварительно настроено командой разработчиков с учётом использования платформы Spring и сторонних библиотек.

Spring Boot позволяет легко создавать автономные производственные приложения Spring, которые можно запускать. Разработчики Spring Boot объективно оценивают платформу Spring и сторонние библиотеки, чтобы было возможно начать работу с минимальными усилиями. Большинству приложений Spring Boot требуется очень небольшая конфигурация Spring.

@SpringBootApplication – с помощью этой аннотации, над точкой запуска серверного приложения, - происходит установка Spring Boot.

### 3.1.4 Spring MVC

Spring MVC следует использовать для реализации серверной части приложения. Spring MVC обеспечивает архитектуру модель-представление-контроллер и готовые компоненты, которые могут быть использованы для разработки гибких и слабо связанных веб-приложений. Шаблон ModelView-Controller помогает отделять бизнеслогику, логику представления и логику навигации. Spring MVC построен вокруг центрального сервлета, который распределяет запросы по контроллерам, а также предоставляет другие широкие возможности при разработке веб приложений.

Эта библиотека играет важную роль в построении серверного приложения, за счет которой создаются слушатели http запросов и происходит направление на выполнение сервисов. Самый распространенный контроллер в листинге 3.1.

Листинг 3.1 – Код контроллера авторизации и регистрации

|  |
| --- |
| @CrossOrigin(origins = "\*", maxAge = 3600)  @RestController  @RequestMapping("auth")  @RequiredArgsConstructor  public class AuthController {  private final AuthService authService;  @PostMapping("signin")  public ResponseEntity<?> authenticateUser(@Valid @RequestBody SignInDto loginRequest) {  return authService.authenticate(loginRequest);  }  @PostMapping("signup")  public ResponseEntity<String> registerUser(@Valid @RequestBody SignUpDto signUpRequest) {  return authService.signup(signUpRequest);  }  } |

Все приведенные ниже аннотации относятся к Spring MVC.

@RestController – эта аннотация используется для того чтобы указать, что контроллер использует REST архитектуру.

@RequestMapping("auth") – эта аннотация используется для обозначения общего начала url в http запросе, что позволяет сократить повторяющийся код и облегчает понимание структуры запросов.

@PostMapping – эта аннотация обозначает какой тип метода должен принимать метод. В данном случае этот метод будет принимать только post запросы, в проекте также активно используются get, put и delete, что составляют достаточный набор для создания CRUD паттерна для моделей.

### 3.1.5 Spring Data

Spring Data — дополнительный удобный механизм для взаимодействия с сущностями базы данных, организации их в репозитории, извлечение данных, изменение, в каких то случаях для этого будет достаточно объявить интерфейс и метод в нем, без имплементации.

Основное понятие в Spring Data — это репозиторий. Это несколько интерфейсов которые используют JPA Entity для взаимодействия с ней. Так например интерфейс в листинге 3.2 обеспечивает основные операции по поиску, сохранения, удалению данных (CRUD операции). Так же этот пример примечателен тем, что тут используется одно из мощных средств Spinrg Data - механизм префиксов find…By, read…By, query…By, count…By, и get…By, далее от префикса метода начинает разбор остальной части. Вводное предложение может содержать дополнительные выражения, например, Distinct. Далее первый By действует как разделитель, чтобы указать начало фактических критериев. Можно определить условия для свойств сущностей и объединить их с помощью And и Or.

Листинг 3.2 – Код интерфейса репозитория для модели User

|  |
| --- |
| @Repository  public interface UserRepository extends JpaRepository<User, Long> {  User findByUsername(String username);  boolean existsByUsername(String username);  boolean existsByEmail(String email);  } |

Все конфигурации для подключения к базе данных прописаны в properties.yml листинг 3.3.

Листинг 3.3 – Конфигурация для подключения к базе данных

|  |
| --- |
| spring:  datasource:  url: jdbc:mysql://localhost:3306/electronic\_documents\_data  username: \*\*\*\*  password: \*\*\*\* |

### 3.1.6 Spring Integration

Spring Integration — открытый фреймворк для интеграции корпоративных приложений. Это легковесный фреймворк, который основывается на ядре Spring. Предназначен для разработки интеграционных решений с событийно-управляемой архитектурой, либо с архитектурой основанной на обмене сообщениями.

@EnableIntegration – с помощью этой аннотации, над точкой запуска серверного приложения, в проекте разрешается использование Spring Integration.

В данном случае Spring integration используется для открытия сессии между сервером и FTP сервером. Код открытия сессии приведен в листинге 3.4.

Листинг 3.4 – Конфигурационный компонент для открытия сесиии

|  |
| --- |
| @RequiredArgsConstructor @Component public class SftpSessionFactoryHandler {   private final Properties properties;   public DefaultSftpSessionFactory gimmeFactory() {  final DefaultSftpSessionFactory factory = new DeaultSftpSessionFactory();   factory.setHost(properties.getHost());  factory.setPort(properties.getPort());  factory.setAllowUnknownKeys(true);  factory.setUser(properties.getUser());  factory.setPassword(properties.getPassword());   return factory;  } } |

### 3.1.7 Spring Security

Spring Security это Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security» и был публично представлен под лицензией Apache License в марте 2004. Впоследствии был включён в Spring как официальный дочерний проект. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года, что включило официальную поддержку и подготовку от SpringSource.

Для того чтобы начать работать с Spring Security изначально нужно определить с какой моделью будет работать Spring Security и будет класифицировать как пользователя приложения. Для этого был создан класс UserPrincilpe в приложении B.

После надо создать конфигурационный класс для Security листинг 3.5.

Листинг 3.5 – Конфигурационный класс для Spring Security.

|  |
| --- |
| @Configuration  @EnableWebSecurity  @EnableGlobalMethodSecurity(  prePostEnabled = true  )  @RequiredArgsConstructor  public class WebSecurityConfig extends WebSecurityConfigurerAdapter |

В этом классе нужно создать некоторые бины, которые будут обеспечивать безопасность различных компонентов. В приложении В показаны все методы, которые нужно было переопределить.

Configure – устанавливает сервис, который будет находить пользователя по заданным параметрам для установления валидности токена, и так же устанавливается метод, которым будет зашифрован пароль пользователя. Так же устанавливает правила на каком url будет проведена проверка токена из заголовка авторизации.

### 3.1.8 Hibernate

При разработке сервера следует использовать JPA и Hibernate для доступа к базе данных. Hibernate – библиотека для языка программирования [Java](https://ru.wikipedia.org/wiki/Java), предназначенная для решения задач объектно-реляционного отображения. Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе в объектно-ориентированных средствах в реляционной базе данных.

JPA(Java Persistence API) – спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных.

В данном серверном приложении Hibernate использовался в основном для определения моделей, которые в свою очередь реализовывались в базе данных. Самый яркий пример это модель User, приведенный в приложении B .

@Entity – это аннотация, которая указывает, что данный класс будет использован в качестве модели.

@Table – это аннотация, которая описывает таблицу, которая должна соответствовать модели, например если нужно будет специфичное название таблицы, то это меняется именно через эту аннотацию, также здесь указываются индексы, которые могут быть созданы на колонки.

@Id – это одна из самых важных аннотаций, без модель не может существовать и указывает на первичный ключ, так же ключ может быть составным и для этого используется другая аннотация.

@GeneratedValue – это аннотация, которая будет генерировать дефолтное значение каждый раз при добавлении записи в таблицу. Если ее указывать без параметров, то будет инкрементация по умолчанию.

@Column – это аннотация используется для описания поля объекта как колонку в базе данных, что позволяет контролировать создание таблиц до мельчайших деталей на основе модели.

@ManyToMany – это аннотация, которая указывает, что здесь будет связь многие ко многим.

@JoinColumn – это аннотация используется для добавления колонки в таблицу модели.

@OneToMany – это аннотация, которая указывает, что здесь будет связь один ко многим.

### 3.1.9 JWT

JSON Web Token (JWT) — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

Токен JWT состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи или данных шифрования. Первые два элемента — это JSON объекты определенной структуры. Третий элемент вычисляется на основании первых и зависит от выбранного алгоритма (в случае использования не подписанного JWT может быть опущен). Токены могут быть перекодированы в компактное представление (JWS/JWE Compact Serialization): к заголовку и полезной нагрузке применяется алгоритм кодирования Base64-URL, после чего добавляется подпись и все три элемента разделяются точками («.»).

JWT токен используется для идентификации пользователя при запросах на защищенные ресурсы. Сервис для работы с токеном представлен в Приложении В.

### 3.1.10 MySql Driver

MySQL Connector / ODBC, то известный как MyODBC, компьютерное программное обеспечение от корпорации Oracle . Это ODBC интерфейс и позволяет языки программирования , которые поддерживают ODBC интерфейс для взаимодействия с MySQL базы данных. MySQL Connector / ODBC был первоначально создан MySQL AB .

### 3.1.11 Lombok

Lombok — проект по добавлению дополнительной функциональности в Java c помощью изменения исходного кода перед Java компиляцией.

По сути, проект Lombok позволяет избавиться от многословности Java в большинстве случаев и перестать писать огромные простыни кода из гетеров, сеттеров, equals, hashcode и toString (да их обычно генерит IDE, но читать и менять все равно приходится программисту), в результате Java становиться почти такой же краткой как Kotlin, Scala или C#.

@Getter – создает геттеры для поля, над которым была написана аннотация, или для всех полей, если – над классом.

@Setter – создает сеттеры для поля, над которым была написана аннотация, или для всех полей, если – над классом.

@NoArgsConstructor – создает пустой конструктор без параметров.

@RequiredArgsConstructor – создает множество конструкторов различных по количеству применяемых параметров.

@AllArgsConstructor – создает конструктор с параметрами, которые соответствуют всем полям класса.

@Builder – создает методы, которые нужны для реализации паттерна Builder в классе, что позволяет пользоваться этим паттерном.

@SneakyThrow – генерирует try – catch, что в свою очередь позволяет не писать пустых блоков, если в них нет никакой логики изначально.

### 3.1.12 REST API

Листинг 3.6 – Api для авторизации пользователя

|  |
| --- |
| @PostMapping("signin")  public ResponseEntity<?> authenticateUser(@Valid @RequestBody SignInDto loginRequest) {  return authService.authenticate(loginRequest);  }  @PostMapping("signup")  public ResponseEntity<String> registerUser(@Valid @RequestBody SignUpDto signUpRequest) {  return authService.signup(signUpRequest);  } |

Листинг 3.7 – Api для работы с нотификациями

|  |
| --- |
| @GetMapping("users/{userId}")  public UserNotificationDto getUserNotifications(@PathVariable final Long userId) {  return notificationService.getUserNotifications(userId);  } |

Остальные API можно найти в приложении В.

### 3.2 Реализация WEB клиента

В ходе проектирования данного программного обеспечения было решено использовать SPA-архитектуру для клиентской части. Она более дружелюбна для пользователя, чем стандартная реализация с помощью статических HTML-страниц, за счет смены содержимого страницы без перезагрузки. Это позволяет добавлять новый функционал в приложения, который ранее был недоступен. Кроме того, такие приложения легче поддаются тестированию.

Для реализации клиентской части приложения был использован фреймворк Angular. Он является одним из наиболее популярных для реализации пользовательского интерфейса и позволяет реализовывать большие и легко масштабируемые приложения.

Единственным статическим файлом, загружаемым с сервера, является index.html. Он содержит ссылки на подключаемые модули. Данный файл представлен в листинге 3.8.

Фреймворк Angular, несмотря на все достоинства, имеет один недостаток – не существует идеальных правил построения структуры приложения. Это значит, что для каждого программного средства необходимо разрабатывать индивидуальную архитектуру, что может затруднить поддержку приложения разными разработчиками.

Проанализировав существующие решения, была принята структура, при которой каждый компонент считается уникальной единицей. Кроме этого, выделены папки для модулей и вспомогательного функционала.

Структура клиентской части приложения представлена на рисунке 3.1.

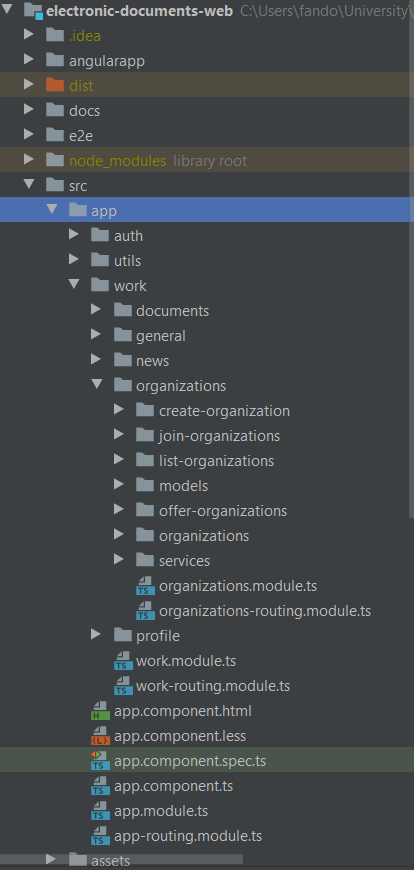


Рисунок 3.1 – Структура клиентской части приложения

Как видно из рисунка выше, модули представляют собой директорию со вспомогательными файлами скриптов и разметки. В листинге 3.1 приведен код стартового файла приложения.

Листинг 3.8 – Код стартового файла приложения

|  |
| --- |
| <!doctype html>  <html lang="en">  <head>  <meta charset="utf-8">  <title>ElectronicDocumentsWeb</title>  <base href="/">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="icon" type="image/x-icon" href="favicon.ico">  </head>  <body>  <app-root></app-root>  </body>  </html> |

Данный подход рекомендуется при написании переиспользуемых модулей, так как в этом случае, папка модуля содержит все необходимое: исходный код и стили.

Angular пропагандирует компонентный подход. Они позволят разделить интерфейс на самостоятельные повторно использующиеся части и анализировать каждую из них в отдельности. Компонент – это функция JavaScript, которая хранит свое состояние и возвращает простую HTML-разметку. Компоненты можно наследовать друг от друга, комбинировать, либо создавать hoc-компоненты (high order components), которые увеличивают переиспользуемость кода и позволяют инкапсулировать в себе некоторую логику, оставляя компонент чистым.

В таблице 3.1 приведена информация о компонентах приложения: название и описание.

Таблица 3.1 – Описание компонентов приложения

|  |  |
| --- | --- |
| Название | Описание |
| Login | Страница авторизации |
| SignUp | Страница регистрации |
| App | Главная страница приложения |
| DocumentsView | Страница отображения документа |
| Documents | Страница отображения всех пользовательских документов |
| General | Страница отображения оповещений и главного меню |
| News | Компонент, содержащий оповещения |
| CreateOrganizations | Страница создания организации |
| JoinOrganizations | Страница отображения всех публичных организаций |
| ListOrganizations | Страница отображения всех организаций пользователя |
| OffersOrganizations | Страница отображения всех запросов пользоватлея |
| OrganizationDocuments | Страница отображения всех документов организации |
| OrganizationGeneral | Страница отображения краткой информации организации |
| OrganizationMembers | Страница отображения всех пользователей организации |
| OrganizationRequests | Страница отображения всех запросов организации |
| OrganizationSettings | Страница отображения настроек организации |
| OrganizationUserDocuments | Станица отображения документов пользователя в организации |
| Profile | Страница отображения профиля пользователя |

### 3.2.1 Инструмент для сборки Webpack

Инструмент для сборки Webpack можно назвать «сборщиком модулей». Он берет модули с зависимостями и генерирует статические наборы ресурсы, представляющие эти модули. Он строит граф зависимостей, позволяя разработчикам использовать модульный подход для построения своих веб-приложений. Особенностью Webpack является то, что он не «склеивает» все файлы в один большой, как это делают другие сборщики. Webpack, строит граф начиная с главного файла, называемого «точкой входа». Таким образом, сборщик оптимизирует выходной файл, включая лишь необходимые исходные.

На листинге 3.9 представлен пример файла конфигурации Webpack – webpack.config.js.

Листинг 3.9 – Пример файла конфигурации Webpack

|  |
| --- |
| const path = require("path");  module.exports = (env, argv) => {  return [{ entry: { 'main': "./Scripts/app/bootstrap.js" },  resolve: { extensions: [".js"] },  output: { path: path.resolve(\_\_dirname, outputFolder),  filename: "components.bundle.js"}}]}; |

### 3.3 Реализация мобильного клиента

В качестве фреймворка для реализации мобильной версии был выбран Ionic. Ionic framework — один из самых широко обсуждаемых фреймворков. Как гласит официальный сайт, Ionic — это SDK для создания гибридных мобильных приложений, набор CSS и JS компонент, созданный на основе AngularJS, SASS, Apache Cordova.

Выбор в пользу Ionic фреймворка был сделан благодаря из-за того, что можно было легко использовать компоненты с веб приложения с некоторыми переработками. Поэтому все что соответствует веб-приложению также соответствует и мобильному. На мобильном приложении есть дополнительно офлайн режим и работа с локальной базой данных.

Работа с локальной базой данных происходит при скачивании документа организации. Сохраняется неизменяемые данные, которые будут использованы для проверки на наличие документа в файловой системе мобильного телефона.

В офлайновый режим можно перейти на двух вкладках в авторизации и регистрации. В нем можно просмотреть все скаченные файлы организаций.

### 3.4 Реализация FTP сервера

В результате FTP сервер был реализован с помощью образа Docker, из которого был создан контейнер. Выбор в пользу Docker был сделан в основном из-за того, что данная технология позволяет абстрагироваться от системы хранения и хранить сервер на любом компьютере, с любой операционной системой, на котором установлено программное обеспечение для работы с Docker образами и контейнерами.

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Изначально использовал возможности LXC, с 2015 года применял собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux — libcontainer. С появлением ​Open Container Initiative начался переход от монолитной к модульной архитектуре.

В качестве image был использован ftp сервер с возможностью подключения через sftp протокол.

### 3.5 Вывод по разделу

В данном разделе был описаны элементы, из которых состоит серверная, клиентская, мобильная, базы данных и FTP сервера части разрабатываемого приложения.

Для серверной части была описана архитектура приложения. Была описана схема маршрутизации, применяемая в приложении. Кроме того, были описаны модули по работе с базой данных, логика по хранению и подписи документов, регистрации и авторизации пользователей, обеспечение безопасности с помощью jwt тщкенов, редактирование профилей, генерация сигнатуры подписи и сертификата с паролем для каждого пользователя.

Для клиентской части были описаны элементы, из которых она состоит: компоненты, действия, обработчики действий. Так же, была приведена и описана структура клиентской части приложения. Были описаны возможности, которые предоставляются клиентом.

В реализации FTP сервера были описаны причина выбора технологии Docker.

# 4 Руководство пользователя

Для того чтобы установить приложения на телефон необходимо иметь Android минимум 7.0 и версию API не ниже 26. Версию Android и API вашего телефона можно узнать в настройках телефона. Так же приложение подходит для смартфонов iOS версии не ниже 10.0.

Если версия подходит, тогда можно устанавливать приложение на мобильный телефон. В меню ваших приложений или на главном экране смартфона появится новая иконка с именем «FunDocker». Запустим приложение нажатием на иконку. Первым делом откроется страница авторизации приложения. Форма для регистрации пользователя можно открыть при нажатии sign up на Рис. 4.1.

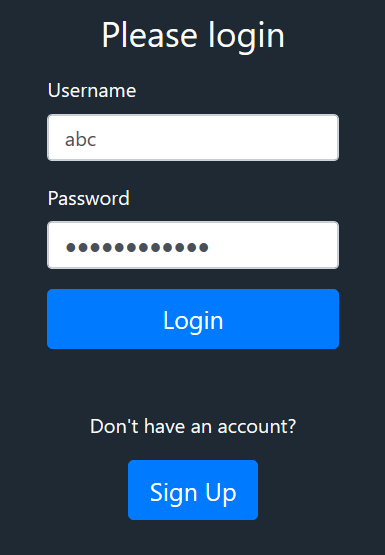


Рисунок 4.1 – страница авторизации

Для начала работы с приложением, новым пользователям необходимо зарегистрироваться, нажав на кнопку «Sign Up» и введя все необходимые поля - зарегистрироваться на рис. 4.2.

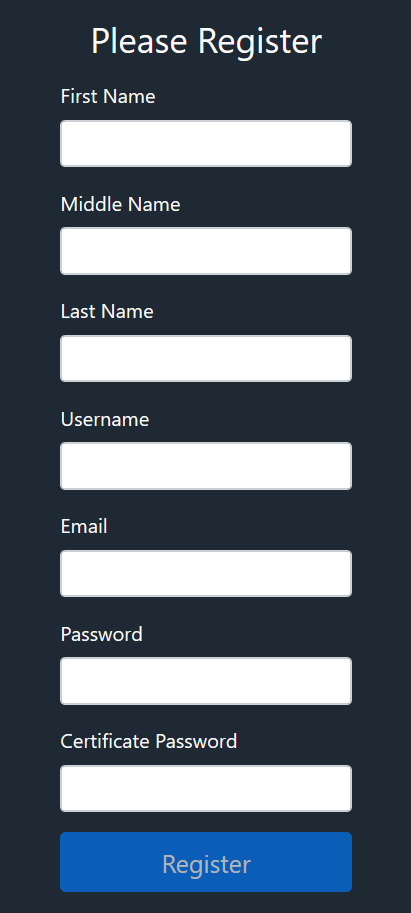


Рисунок 4.2 – страница регистрации

После регистрации и авторизации – пользователь увидит главную страницу, где расположены нотификации пользователя и главное меню, в котором можно перейти в профиль, список организаций и списки файлов, которые относятся к пользователю на рис. 4.3.

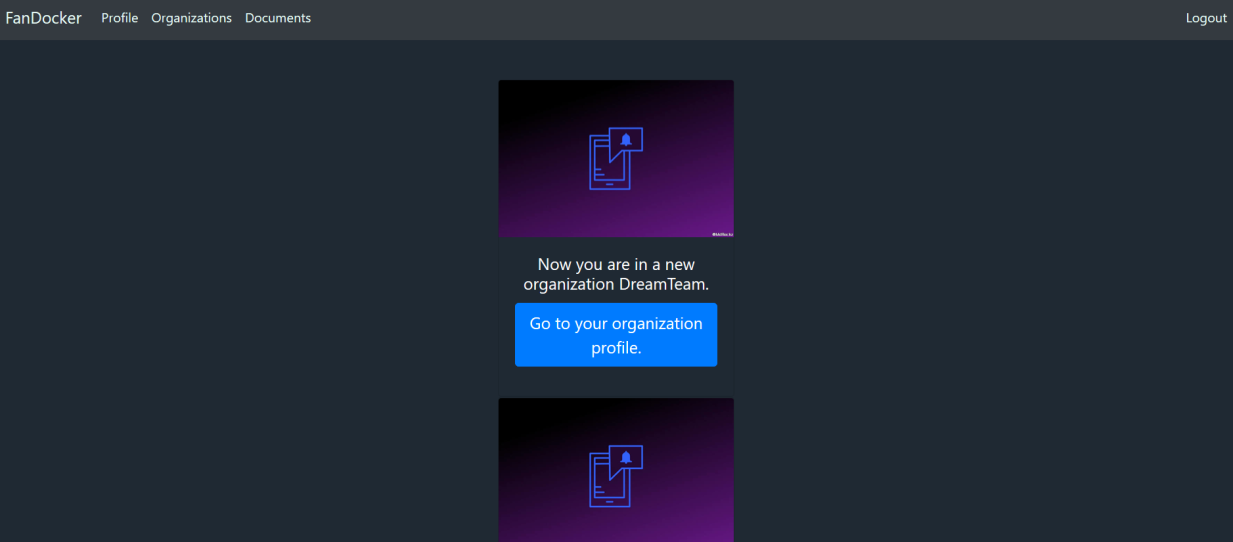


Рисунок 4.3 – главная страница с нотификациями

При нажатии на вкладку профиль, пользователь перейдет на свой профиль, где он сможет обновить информацию о себе на рис 4.4.

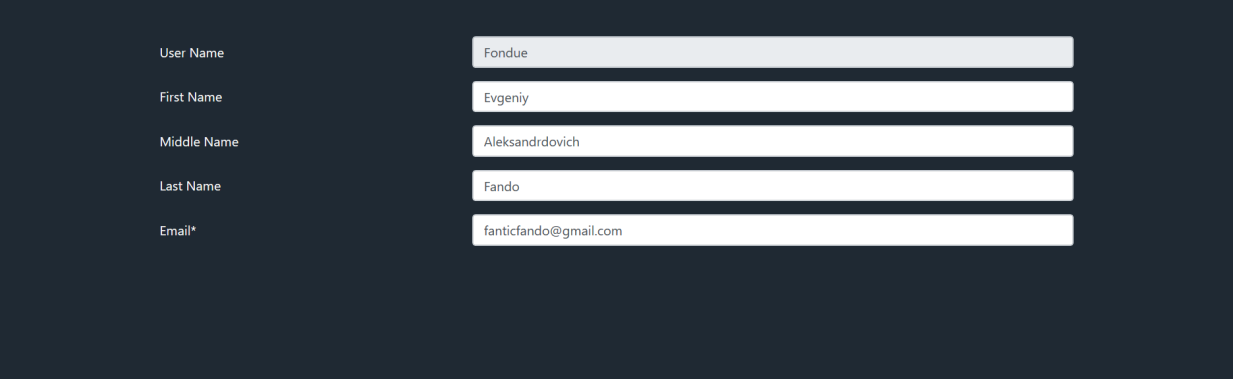


Рисунок 4.4– профиль

При нажатии на вкладку документы – пользователь попадает в активность, где отображаются списки, которые разделены на категории, но все относятся к одному пользователю на рис. 4.5.

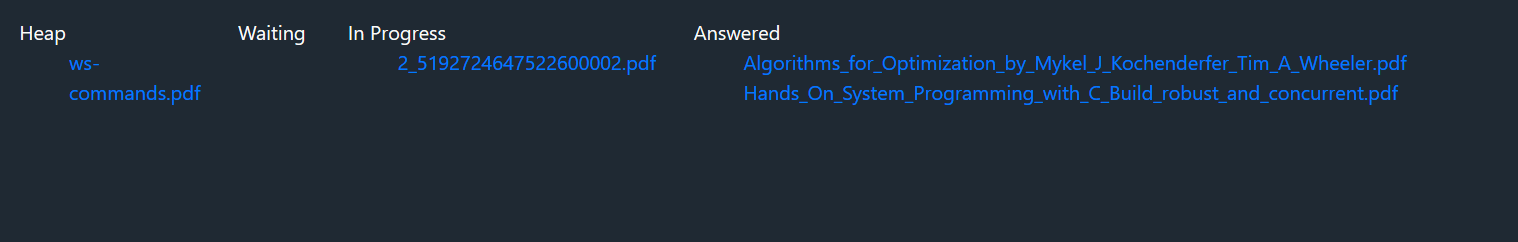


Рисунок 4.5 – документы пользователя

После нажатия на вкладку организации пользователь увидит все манипуляции, которые он может совершить в отношении организации на рис 4.6.

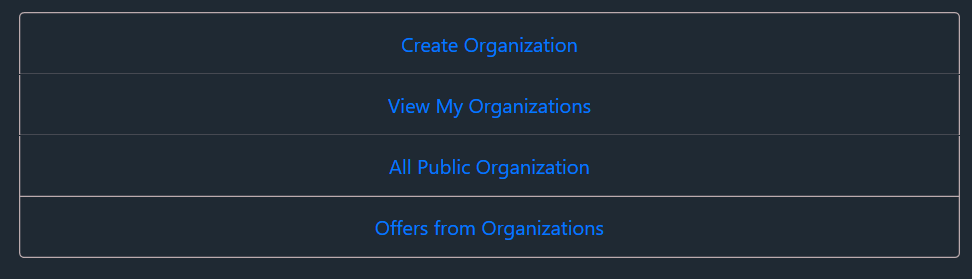


Рисунок 4.6 – вкладки на действия с организациями

Пользователь может создать новую организацию, указав имя и тип на рис 4.7.

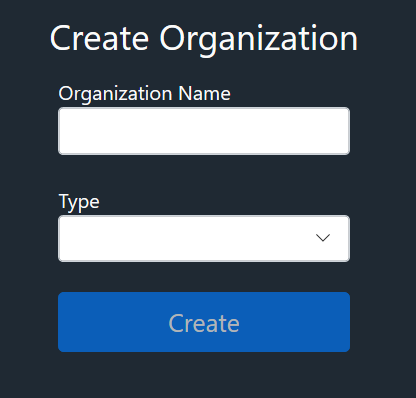


Рисунок 4.7 – страница с созданием организации

Пользователь может отправить запрос на добавление в организацию, открыв активность со всеми публичными организациями. Если пользователь состоит или уже отправил запрос, то это указывается на рис 4.8.



Рисунок 4.8– список всех публичных организаций

Так же есть вкладка, где показаны все запросы, которые отправил пользователь и от организаций на рис 4.9.



Рисунок 4.9 – запросы, которые относятся к пользователю

Когда пользователь перейдет на вкладку где, показаны все его организации, то он увидит список всех изображений с краткой информацией об организации на рис 4.10.



Рисунок 4.10 – список всех организаций пользователя

После перехода на конкретную организацию, то в зависимости от роли пользователь будет видеть различное количество вкладок на рис. 4.11 и на рис 4.12.

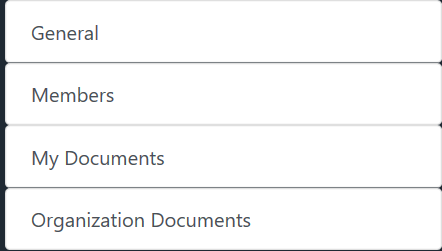


Рисунок 4.11 – список вкладок для обычного пользователя

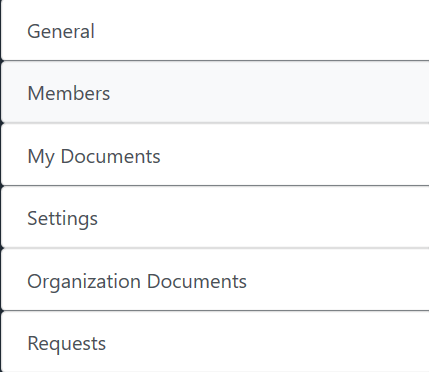


Рисунок 4.12 – список вкладок для создателя организации

На главной вкладке пользователь может увидеть краткую информацию об организации и свои роли в ней на рис 4.13.

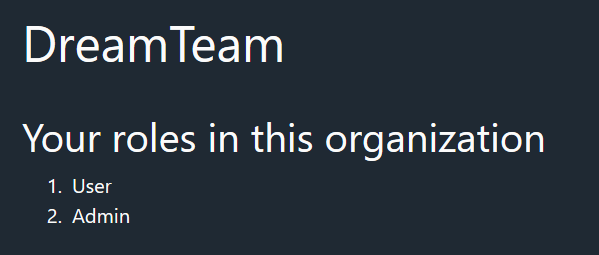


Рисунок 4.13 – главная вкладка организации

Когда пользователь перейдет на вкладку участники, то он увидит всех пользователей, которые находятся в организации на рис 4.14.



Рисунок 4.14 – отображение участника организации

Только создатель организации может перейти на изменение ролей у конкретного пользователя на рис. 4.15.

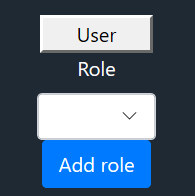


Рисунок 4.15 – страница с выделением ролей

В настройках создатель организации может изменить информацию об организации на рис 4.16 и еще перейти на вкладку по добавлению или изменению ролей в организации на рис 4.17.

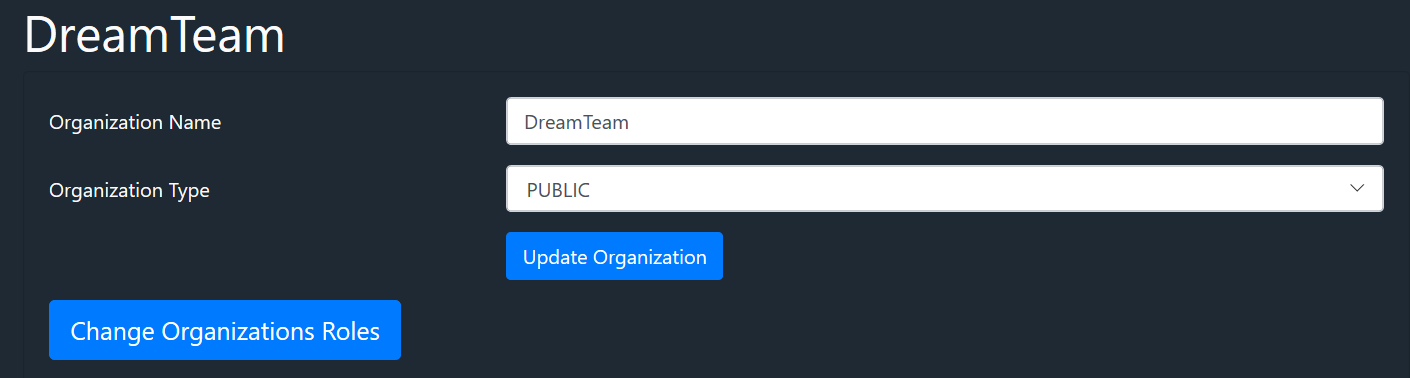


Рисунок 4.16 – страница настроек организации

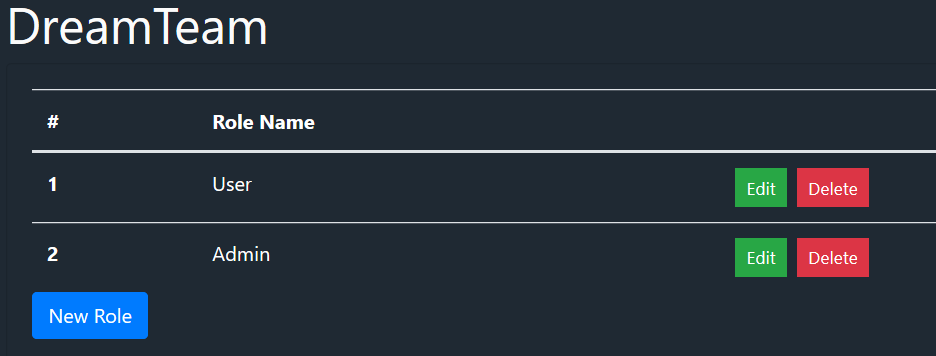


Рисунок 4.17 – страница по изменению ролей в организации

Во вкладке запросы создатель организации сможет увидеть все, что прислали пользователи и он сможет отклонить или приять их на рис 4.18.

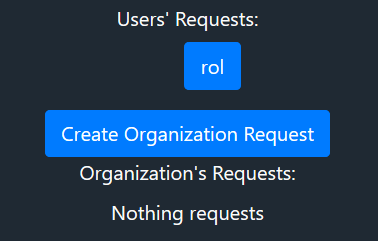


Рисунок 4.18 – страница с запросами в организации

Во вкладке мои документы пользователь может добавить свой документ на рис 4.19. Здесь показаны все документы, которые пользователь добавил, поделенные на состояния.

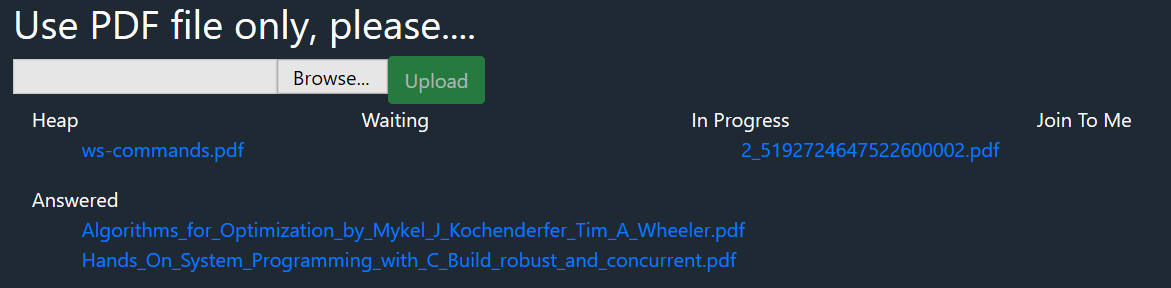


Рисунок 4.19 – страница с документами пользователя в организации

На вкладке документы пользователь сможет увидеть все документы организации, которые прошли через все подписи на рис 4.20.

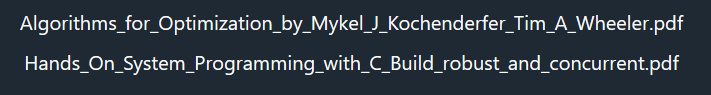


Рисунок 4.20 – страница с документами организации

По поводу подписи документов, есть 5 состояний документа: heap, waiting, in progress, join to me, answered.

Как только пользователь загружает документ ‑ он получает статус heap и чтобы перейти в состояние in waiting пользователь должен подтвердить его своей подписью и поставить роль пользователей, которая должна присутствовать у пользователей подписывающих этот документ.

Как только документ перешел в состояние in waiting все пользователи, которые должны подписать этот документ смогут найти информацию о нем во вкладке документы с главной страницы или в документах пользователя в организации.

После того как пользователь скачает документ, то документ перейдет в состояние in progress и можно будет увидеть состояние подписей других пользователей.

Когда все пользователи поставят свои подписи на документ, то он перейдет в состояние answered и его можно будет увидеть в документах организации.

# ЗАКЛЮЧЕНИЕ

Целью моего курсового проекта является создание приложения в котором будет возможность создавать организации и добавлять новые документы, которые можно будет подписать и в последствии использовать.

Были решены следующие поставленные задачи:

* изучение аналогов;

В результате анализа аналогов было рассмотрено 2 приложения. Были выявлены их достоинства и недостатки. Был сделан вывод о том, что очень важно не только правильно подобрать нужный функционал, но и грамотно поместить его, чтобы не перегрузить приложение. Также важна логичная и простая навигация, большое количество средств взаимодействия с пользователем, лёгкий и ненавязчивый дизайн.

* проектирование приложения;

В проекте электронный документооборот был определен основной функционал, а также были разработаны необходимые диаграммы. При проектировании базы данных были определены необходимые таблицы и разработана схема базы данных. В серверном приложении был определен архитектурный стиль взаимодействия между клиентом и сервером (REST), формат передаваемых данных и HTTP методы для работы с серверной частью.

* реализация приложения;

Данная задача была реализована с помощью следующих технических средств: MySQL, sqlite, сред разработки: Intellij Idea, языков программирования typescript и Java, мобильной платформы ionic, фреймворка angular и библиотек Spring.

При решении данной задачи была достигнута поставленная цель, и был разработан проект «FunDocker», который включает в себя мобильное приложение, сервер, веб-приложение, FTP сервер и удалённую базу данных.

* разработка руководства пользователя.

Разработано руководство, предоставляющее пользователям ознакомление со всеми возможностями программы, а также инструкцию по использованию функций данного приложения, к которой можно обращаться в процессе работы с приложением.

Мобильное и веб приложение имеет простой, интуитивно понятный пользователю интерфейс и обладает следующими особенностями:

* Предоставлять профиль активного пользователя и так же возможность редактирования его;
* Возможность создания организации;
* Редактирование информации о организации, при наличии на это прав;
* Принимать запросы в организации на вступление, при наличии определенных прав;
* Просматривать все документы организации;
* Добавить свой документ в организацию;
* Создать запрос на подписание своих добавленных документов в организации по роли;
* Возможность подписать документ своей ЭЦП;
* Принять или сделать предложение на вступление в организацию;

В соответствии с поставленными задачами и полученным результатом можно сделать вывод, что курсовой проект выполнен в полном объеме и поставленные задачи успешно выполнены.

# Список литературы

1. mysql.com [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.mysql.com/docs/>.

2. angular.io [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://angular.io/](https://www.mysql.com/docs/).

3. docker.com [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docker.com /.

4. spring.io [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://spring.io/>.

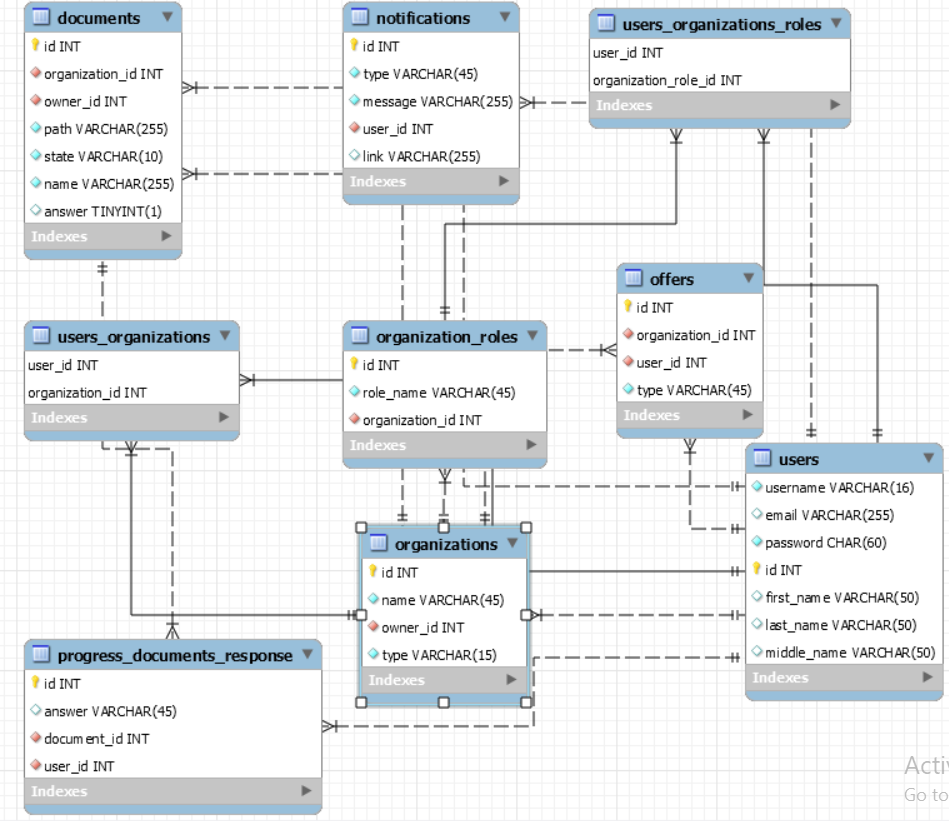
5. wikipedia.org [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://wikipedia.org/>.

6. ionicframework.com [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ionicframework.com/>.

7. cryptopro.app [Электронный ресурс]. – Электронные данные. – Режим доступа: https://cryptopro.app /.

# Приложение А

Структура базы данных проектируемого программного средства



# Приложение Б

create table documents  
(  
 id int auto\_increment  
 primary key,  
 organization\_id int not null,  
 owner\_id int not null,  
 path varchar(255) not null,  
 state varchar(10) not null,  
 name varchar(255) not null,  
 answer tinyint(1) null,  
 constraint const\_fk\_org\_id  
 foreign key (organization\_id) references organizations (id),  
 constraint const\_fk\_user\_id  
 foreign key (owner\_id) references users (id)  
);  
  
create index const\_fk\_org\_id\_idx  
 on documents (organization\_id);  
  
create index const\_fk\_user\_id\_idx  
 on documents (owner\_id);

create table notifications  
(  
 id int auto\_increment  
 primary key,  
 type varchar(45) not null,  
 message varchar(255) not null,  
 user\_id int not null,  
 link varchar(255) null,  
 constraint asasdfergebgsdfgv  
 foreign key (user\_id) references users (id)  
);  
  
create index asasdfergebgsdfgv\_idx  
 on notifications (user\_id);

create table offers  
(  
 id int auto\_increment  
 primary key,  
 organization\_id int not null,  
 user\_id int not null,  
 type varchar(45) not null,  
 constraint fk\_organization\_id  
 foreign key (organization\_id) references organizations (id)  
 on update cascade on delete cascade,  
 constraint fk\_user\_id  
 foreign key (user\_id) references users (id)  
);  
  
create index fk\_organization\_id\_idx  
 on offers (organization\_id);  
  
create index fk\_user\_id\_idx  
 on offers (user\_id);

create table organization\_roles  
(  
 id int auto\_increment  
 primary key,  
 role\_name varchar(45) not null,  
 organization\_id int not null,  
 constraint sdfasdfasdfadfg  
 foreign key (organization\_id) references organizations (id)  
);  
  
create index sdfasdfasdfadfg\_idx  
 on organization\_roles (organization\_id);

create table organizations  
(  
 id int auto\_increment  
 primary key,  
 name varchar(45) not null,  
 owner\_id int not null,  
 type varchar(15) default 'PUBLIC' not null,  
 constraint UK\_mlqcjckw7552m3frcbs09j055  
 unique (name),  
 constraint name\_UNIQUE  
 unique (name),  
 constraint FK525ovcw3fy6440s4o0tj8xr95  
 foreign key (owner\_id) references users (id)  
);  
  
create index fk\_owner\_id\_idx  
 on organizations (owner\_id);

create table progress\_documents\_response  
(  
 id int auto\_increment  
 primary key,  
 answer varchar(45) null,  
 document\_id int not null,  
 user\_id int not null,  
 constraint rtyjnrtyjrbr  
 foreign key (user\_id) references users (id),  
 constraint werqwerqwe  
 foreign key (document\_id) references documents (id)  
);  
  
create index rtyjnrtyjrbr\_idx  
 on progress\_documents\_response (user\_id);  
  
create index werqwerqwe\_idx  
 on progress\_documents\_response (document\_id);

create table users  
(  
 username varchar(16) not null,  
 email varchar(255) null,  
 password char(60) not null,  
 id int auto\_increment  
 primary key,  
 first\_name varchar(50) null,  
 last\_name varchar(50) null,  
 middle\_name varchar(50) null,  
 constraint UK6dotkott2kjsp8vw4d0m25fb7  
 unique (email),  
 constraint UKr43af9ap4edm43mmtq01oddj6  
 unique (username),  
 constraint email\_UNIQUE  
 unique (email),  
 constraint username\_UNIQUE  
 unique (username)  
);

create table users\_organizations  
(  
 user\_id int not null,  
 organization\_id int not null,  
 primary key (user\_id, organization\_id),  
 constraint FKcx7c5yo5xkeunti8qv07f6gjk  
 foreign key (user\_id) references users (id),  
 constraint FKmr2o7bfoblu76nlj9qk23r1en  
 foreign key (organization\_id) references organizations (id)  
);  
  
create index FKmr2o7bfoblu76nlj9qk23r1en\_idx  
 on users\_organizations (organization\_id);  
  
create definer = root@localhost trigger users\_organizations\_AFTER\_INSERT  
 after insert  
 on users\_organizations  
 for each row  
BEGIN  
 INSERT INTO notifications(type, message, user\_id, link)  
 select 'ADD\_ORGANIZATION' as type,  
 CONCAT('Now you are in a new organization ', o.name, '.') as message,  
 NEW.user\_id as user\_id,  
 CONCAT('/organizations/list/', NEW.organization\_id, '/general') as link  
 from organizations as o  
 where o.id = NEW.organization\_id;  
END;

create table users\_organizations\_roles  
(  
 user\_id int not null,  
 organization\_role\_id int not null,  
 primary key (user\_id, organization\_role\_id),  
 constraint asdfasdfas  
 foreign key (user\_id) references users (id),  
 constraint baergthgfbdf  
 foreign key (organization\_role\_id) references organization\_roles (id)  
);  
  
create index asdfasdfas\_idx  
 on users\_organizations\_roles (user\_id);  
  
create index baergthgfbdf\_idx  
 on users\_organizations\_roles (organization\_role\_id);  
  
create definer = root@localhost trigger after\_added\_user\_role  
 after insert  
 on users\_organizations\_roles  
 for each row  
BEGIN  
 INSERT INTO notifications(type, message, user\_id, link)  
 select 'ADD\_ROLE' as type,  
 CONCAT('You have a new ', o\_r.role\_name, ' role in the ', o.name, ' organization.') as message,  
 NEW.user\_id as user\_id,  
 CONCAT('/organizations/list/', o\_r.organization\_id, '/general') as link  
 from organization\_roles as o\_r  
 inner join organizations as o ON o.id = o\_r.organization\_id  
 where o\_r.id = NEW.organization\_role\_id;  
END;  
  
create definer = root@localhost trigger after\_delete\_user\_role  
 after delete  
 on users\_organizations\_roles  
 for each row  
BEGIN  
 INSERT INTO notifications(type, message, user\_id, link)  
 select 'DELETE\_ROLE' as type,  
 CONCAT('In your profile, the ', o\_r.role\_name, ' role in the ', o.name,  
 ' organization was removed.') as message,  
 OLD.user\_id as user\_id,  
 CONCAT('/organizations/list/', o\_r.organization\_id, '/general') as link  
 from organization\_roles as o\_r  
 inner join organizations as o ON o.id = o\_r.organization\_id  
 where o\_r.id = OLD.organization\_role\_id;  
END;

# Приложение B

Сервис по управлению токеном

@Component

public class JwtProvider {

private static final Logger logger = LoggerFactory.getLogger(JwtProvider.class);

@Value("${t.app.jwtSecret}")

private String jwtSecret;

@Value("${t.app.jwtExpiration}")

private int jwtExpiration;

public String generateJwtToken(final Authentication authentication) {

final UserPrinciple userPrinciple = (UserPrinciple)authentication.getPrincipal();

return Jwts.builder()

.setSubject(userPrinciple.getUsername())

.setIssuedAt(new Date())

.setExpiration(new Date(new Date().getTime() + jwtExpiration))

.signWith(SignatureAlgorithm.HS512, jwtSecret)

.compact();

}

public String getUserNameFromJwtToken(final String token) {

return Jwts.parser()

.setSigningKey(jwtSecret)

.parseClaimsJws(token)

.getBody().getSubject();

}

public boolean validateJwtToken(final String authToken) {

try {

Jwts.parser()

.setSigningKey(jwtSecret)

.parseClaimsJws(authToken);

return true;

} catch (MalformedJwtException e) {

logger.error("Invalid JWT token -> Message: {}", e);

} catch (ExpiredJwtException e) {

logger.error("Expired JWT token -> Message: {}", e);

} catch (UnsupportedJwtException e) {

logger.error("Unsupported JWT token -> Message: {}", e);

} catch (IllegalArgumentException e) {

logger.error("JWT claims string is empty -> Message: {}", e);

}

return false;

}

public PrivateJoinTokenDto generatePrivateJoinJwtToken(final GenerateOrganizationJoinJwtDto jwtDto) {

final String subject = new JSONObject()

.put("username", jwtDto.getUsername())

.put("organizationId", jwtDto.getOrganizationId()).toString();

final String token = Jwts.builder()

.setSubject(subject)

.setIssuedAt(new Date())

.setExpiration(new Date(new Date().getTime() + jwtExpiration))

.signWith(SignatureAlgorithm.HS512, jwtSecret)

.compact();

return PrivateJoinTokenDto.builder()

.token(token).build();

}

public String getSubject(final String token) {

return Jwts.parser()

.setSigningKey(jwtSecret)

.parseClaimsJws(token)

.getBody().getSubject();

}

}

Код класса UserPrinciple

@Getter

@Setter

@Builder

public class UserPrinciple implements UserDetails {

private static final long serialVersionUid = 1L;

private Long id;

private String username;

@JsonIgnore

private String password

private Collection<? extends GrantedAuthority> authorities;

public static UserPrinciple build(final User user) {

final List<GrantedAuthority> authorities = user.getOrganizationRoles().stream()

.map(OrganizationRole::getRoleName)

.map(SimpleGrantedAuthority::new)

.collect(toList());

return UserPrinciple.builder()

.id(user.getId())

.username(user.getUsername())

.password(user.getPassword())

.authorities(authorities).build();

}

}

Методы для обеспечения защиты в конфигурационном классе

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService)

.passwordEncoder(passwordEncoder());

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(final HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeRequests()

.antMatchers("/resources/\*\*").permitAll()

.antMatchers("/auth/\*\*").permitAll()

.anyRequest().authenticated().and()

.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and() .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

Модель User

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue

@Column(name = "id")

private Long id;

@NotBlank

@Encrypted

@Column(name = "first\_name")

private String firstName;

@NotBlank

@Encrypted

@Column(name = "middle\_name")

private String middleName;

@NotBlank

@Encrypted

@Column(name = "last\_name")

private String lastName;

@NotBlank

@Column(name = "username")

private String username;

@NotBlank

@Encrypted

@Size(max = 255)

@Email

@Column(name = "email")

private String email;

@NotBlank

@Column(name = "password", columnDefinition = "CHAR(60)")

private String password;

@ManyToMany

@JoinTable(name = "users\_organizations",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "organization\_id"))

private List<Organization> organizations;

@OneToMany(mappedBy = "owner")

private List<Organization> ownerOrganization;

@OneToMany(mappedBy = "user")

private List<Offer> offers;

@ManyToMany

@JoinTable(name = "users\_organizations\_roles",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "organization\_role\_id"))

private List<OrganizationRole> organizationRoles;

}

Api для работы с организацией

@PostMapping("organization/{organizationId}/user/{userId}")

public void uploadOrganizationFile(@PathVariable final Long organizationId,@PathVariable final Long userId,@RequestBody final MultipartFile file) {

service.uploadOrganizationFile(organizationId, userId, file);

}

@GetMapping("organization/{organizationId}/user/{userId}")

public MyOrganizationDocumentsInfoDto getUserOrganizationDocuments(@PathVariable final Long organizationId,

@PathVariable final Long userId) {

return service.getUserOrganizationDocuments(organizationId, userId);

}

@GetMapping("{id}")

public OrganizationDocumentViewDto getOrganizationDocumentView(@PathVariable final Long id) {

return service.getOrganizationDocumentView(id);

}

@GetMapping("user/{userId}")

public UserDocumentsInfoDto getUserDocumentsInfo(@PathVariable final Long userId) {

return service.getUserDocumentsInfo(userId);

}

@PutMapping("{documentId}")

public void changeDocumentState(@PathVariable final Long documentId,

@RequestBody final ChangeDocumentStateDto dto) {

service.changeDocumentState(documentId, dto);

}

@PutMapping("{documentId}/approveDeny")

public void changeDocumentState(@PathVariable final Long documentId,

@RequestBody final DocumentAnswerDto answer) {

service.approveDenyDocument(documentId, answer);

}

@PostMapping("{documentId}/download")

public byte[] renameOrganizationRole(@PathVariable Long documentId) {

return service.download(documentId);

}

@GetMapping("{documentId}/heap")

public HeapDocumentViewDto getHeapDocument(@PathVariable Long documentId) {

return service.getHeapDocument(documentId);

}

@PostMapping("{documentId}/heap")

public void approveHeapDocument(@PathVariable final Long documentId,

@RequestBody final HeapDocumentViewDto heapDocumentViewDto) {

service.approveHeapDocument(heapDocumentViewDto);

}

@GetMapping("{documentId}/waiting")

public WaitingDocumentViewDto getWaitingDocument(@PathVariable Long documentId) {

return service.getWaitingDocument(documentId);

}

@GetMapping("{documentId}/join-to-me")

public JoinToMeDocumentViewDto getJoinToMeDocument(@PathVariable Long documentId) {

return service.getJoinToMeDocument(documentId);

}

@SneakyThrows

@PostMapping("{documentId}/join-to-me/download")

public ResponseEntity<byte[]> downloadDocumentForCheck(@PathVariable Long documentId) {

final byte[] blob = service.download(documentId);

return ResponseEntity.ok().contentType(MediaType.APPLICATION\_PDF).contentLength(blob.length).body(blob);

}

@PostMapping("{documentId}/join-to-me/answer")

public void answerDocument(@PathVariable final Long documentId,

@RequestBody final DocumentAnswerDto answer) {

service.answerDocument(documentId, answer);

}

@GetMapping("{documentId}/progress")

public PendingDocumentViewDto getPendingDocument(@PathVariable Long documentId) {

return service.getPendingDocument(documentId);

}

@GetMapping("{documentId}/answered")

public AnsweredDocumentViewDto getAnsweredDocument(@PathVariable Long documentId) {

return service.getAnsweredDocument(documentId);

}

@GetMapping("organization/{organizationId}")

public OrganizationDocumentsInfoDto getOrganizationDocumentsInfo(@PathVariable final Long organizationId) {

return service.getOrganizationDocumentsInfo(organizationId);

}

@GetMapping("{documentId}/state")

public String getDocumentState(@PathVariable final Long documentId) {

return service.getDocumentState(documentId);

}

Api для работы с организациями

@GetMapping("{organizationId}")

public OrganizationViewDto get(@PathVariable final Long organizationId) {

return service.getOrganizationView(organizationId);

}

@PostMapping

public Long create(@RequestBody OrganizationCreateDto organizationCreateDto) {

return service.createOrganization(organizationCreateDto);

}

@GetMapping("{organizationId}/members")

public List<OrganizationMemberDto> getOrganizationMembers(@PathVariable final Long organizationId) {

return service.getOrganizationMembers(organizationId);

}

@GetMapping("{organizationId}/members/{userId}")

public OrganizationMemberViewDto getOrganizationMember(@PathVariable final Long organizationId,

@PathVariable final Long userId) {

return service.getOrganizationMember(organizationId, userId);

}

@GetMapping("{organizationId}/settings")

public OrganizationSettingsDto getOrganizationSettings(@PathVariable final Long organizationId) {

return service.getOrganizationSettings(organizationId);

}

@PutMapping("{organizationId}/settings")

public void updateOrganizationSettings(

@RequestBody final OrganizationSettingsDto organizationSettingsDto) {

service.updateOrganizationSettings(organizationSettingsDto);

}

@GetMapping("{organizationId}/offers")

public OrganizationRequestsView getOffers(@PathVariable final Long organizationId) {

return service.getOffers(organizationId);

}

@PutMapping("offers")

public void answeredOffer(@RequestBody final OrganizationAnswerOfferDto organizationAnswerOfferDto) {

service.answeredOffer(organizationAnswerOfferDto);

}

@GetMapping("public")

public List<OrganizationInfoDto> getPublicOrganizations() {

return service.getPublicOrganizations();

}

@PostMapping("generate/private-token")

public PrivateJoinTokenDto generatePrivateJoinToken(@RequestBody final GenerateOrganizationJoinJwtDto jwtDto) {

return service.generatePrivateJoinToken(jwtDto);

}

@PostMapping("join/private")

public void privateJoin(@RequestBody final PrivateJoinTokenDto privateJoinTokenDto) {

service.privateJoin(privateJoinTokenDto);

}

@PostMapping("{organizationId}/user/{userId}/request")

public void createRequest(@PathVariable final Long organizationId, @PathVariable final Long userId) {

service.createRequest(organizationId, userId);

}

@GetMapping("{organizationId}/roles")

public List<OrganizationRoleInfoDto> getOrganizationRoles(@PathVariable final Long organizationId) {

return service.getOrganizationRoles(organizationId);

}

@PostMapping("roles/create")

public void creteOrganizationRole(@RequestBody final CreateOrganizationRoleDto createOrganizationRoleDto) {

service.createOrganizationRole(createOrganizationRoleDto);

}

@DeleteMapping("roles/{organizationRoleId}")

public void deleteOrganizationRole(@PathVariable final Long organizationRoleId) {

service.deleteOrganizationRole(organizationRoleId);

}

@PutMapping("roles/rename")

public void renameOrganizationRole(@RequestBody final RenameOrganizationRoleDto renameOrganizationRoleDto) {

service.renameOrganizationRole(renameOrganizationRoleDto);

}

@PostMapping("organization-request/create")

public void createOrganizationRequest(@RequestBody CreateOrganizationRequest createOrganizationRequest) {

service.createOrganizationRequest(createOrganizationRequest);

}

@PostMapping("{organizationId}/members/{memberId}/roles")

public void addRole(@PathVariable final Long organizationId,

@PathVariable final Long memberId,

@RequestBody final AddRoleDto addRoleDto) {

service.addRole(organizationId, memberId, addRoleDto);

}

@DeleteMapping("{organizationId}/members/{memberId}/roles/{id}")

public void deleteRole(@PathVariable final Long organizationId,

@PathVariable final Long memberId,

@PathVariable final Long id) {

service.deleteRole(organizationId, memberId, id);

}

@GetMapping("{organizationId}/general")

public OrganizationGeneral getOrganizationGeneral(@PathVariable final Long organizationId) {

return service.getOrganizationGeneral(organizationId);

}

@GetMapping("{organizationId}/name")

public String getOrganizationName(@PathVariable final Long organizationId) {

return service.getOrganizationNameById(organizationId);

}

Api для работы с пользователем

@GetMapping("{username}/dashboard")

public UserDashboardDto get(@PathVariable final String username) {

return service.getUserDashboard(username);

}

@PutMapping("{username}/dashboard")

public void put(@PathVariable final String username, @RequestBody final UserDashboardDto userDashboardDto) {

service.updateUserDashboard(username, userDashboardDto);

}

@GetMapping("{username}/organizations")

public List<OrganizationInfoDto> getList(@PathVariable final String username) {

return service.getOrganizations(username);

}

@GetMapping("{username}/offers")

public UserRequestsViewDto getRequests(@PathVariable final String username) {

return service.getRequests(username);

}

@PostMapping("permissions/{organizationId}")

public void checkPermissions(@PathVariable Long organizationId) throws Exception {

service.checkPermissions(organizationId);

}