

Министерство науки и высшего образования РФ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
Национальный исследовательский технологический университет «МИСиС»

Институт компьютерных наук  
Кафедра автоматизированных систем управления

Отчёт по производственной практике на тему:  
«Разработка базового функционала мобильного клиент-серверного приложения, включающего создание системы авторизации, управления очередями и уведомления пользователей, для оптимизации процесса отработки учебных задолженностей в образовательных организациях»

Выполнил:  
ст. гр. БИВТ-21-1 Витязев Е. Д.

Москва 2024

# Содержание

1. Введение.....	3
1.1 Актуальность темы.....	3
1.2 Основная часть.....	4
1.2.1 Обзор предметной области.....	4
1.2.2 Объект исследования.....	5
1.2.3 Основная проблематика.....	5
1.3 Заключение.....	6
2. Цели и задачи практики.....	6
3. Разработка (Основная часть).....	7
3.1 Настройка серверной инфраструктуры.....	7
3.2 Разработка серверной части (backend).....	9
3.2.1 Создание проекта с использованием фреймворка Spring Boot и библиотек Thymeleaf и Hibernate.....	9
3.2.2 Реализации системы авторизации.....	11
3.2.3 Разработка фильтров для ограничения доступа неавторизованным пользователям .....	13
3.2.4 Разработка контролеров и веб-интерфейса (представлений) для управления очередями на отработки (добавление, удаление, просмотр отработок).....	15
Модель DebtRepayment.....	15
Репозиторий DebtRepaymentRepo.....	15
Контролер RepaymentController.....	15
Веб-интерфейс.....	16
3.2.5 Создание модели очереди студентов и системы управления очередями (приём студентов, определение позиций в очереди). Добавление возможности импортировать списки записавшихся студентов из Google форм.....	17
Модель QStudent.....	17
Репозиторий QStudentRepo.....	18
Методы контроллера RepaymentController.....	18
Представления.....	19
Проверка работоспособности.....	20
3.2.6 Разработка API для передачи информации между сервером и мобильным приложением.....	22
3.3 Разработка мобильного приложения (Android).....	24
3.3.1 Реализация возможности идентификации студента по электронной почте с сохранением информации.....	24
3.3.2 Реализация возможности просмотра информации о отработках, на которые записан студент.....	27
3.3.3 Создание формы для просмотра информации о текущем положении студента в очереди в приложении.....	31
Разметка activity_repayment_view.xml.....	31
Класс RepaymentViewActivity.....	32
Класс PositionUpdater.....	32
4. Выводы о проделанной работе.....	33
5. Приложения.....	34

# 1. Введение

## 1.1 Актуальность темы

Выбор темы для моей практической работы — "Разработка базового функционала мобильного клиент-серверного приложения, включающего создание системы авторизации, управления очередями и уведомления пользователей, для оптимизации процесса отработки учебных задолженностей в образовательных организациях" — обусловлен несколькими ключевыми факторами, связанными с текущими потребностями и вызовами в образовательной сфере.

Во-первых, в условиях цифровой трансформации образовательных процессов автоматизация управления очередями на отработки учебных долгов становится всё более необходимой. В настоящее время существующая система организации отработок сопряжена с рядом неудобств, таких как хаотичное ожидание студентов у кабинета преподавателя, что создаёт стрессовую и шумную атмосферу, мешающую как самим студентам, так и преподавателям.

Во-вторых, преподаватели часто сталкиваются с проблемой неявки студентов на отработки, что ведет к неэффективному управлению очередями, потере времени и снижению общей продуктивности. Автоматизация этого процесса позволит создать более чёткую и прозрачную систему оповещения студентов о времени их отработки, их текущем положении в очереди и необходимости явки. Это обеспечит более удобные условия для студентов и облегчит работу преподавателей, позволяя им сосредоточиться на образовательном процессе.

Наконец, внедрение таких технологий является необходимым шагом в современном образовательном процессе, направленным на повышение качества обучения и управления временем. Автоматизация управления очередями на отработки учебных долгов представляет собой важный шаг для повышения эффективности и удобства как для преподавателей, так и для студентов.

## 1.2 Основная часть

### 1.2.1 Обзор предметной области

Проект направлен на решение задач, возникающих в образовательных учреждениях, где необходимо эффективное управление очередями студентов, имеющих академические задолженности и нуждающихся в отработке пропущенных занятий или выполнении дополнительных заданий. Проблема академических задолженностей актуальна для многих учебных заведений, и традиционные методы управления такими процессами часто оказываются неэффективными, создавая дополнительные нагрузки для студентов и преподавателей.

### 1.2.2 Объект исследования

В рамках моей практической работы я исследую возможность реализации базового функционала мобильного клиент-серверного приложения. Объект исследования включает в себя несколько ключевых компонентов:

1. **Мобильное приложение для Android:** основная задача которого — предоставить студентам возможность регистрироваться в системе, просматривать свою очередь и получать информация об отработках. Приложение обеспечит удобный интерфейс для взаимодействия с системой и будет доступно в любое время и из любого места.
2. **Серверная часть с использованием веб-фреймворка:** сервер будет обрабатывать запросы от мобильного приложения и веб-интерфейса, управлять данными, аутентификацией пользователей и логистикой очередей и отработок. Использование современного веб-фреймворка обеспечит масштабируемость и надежность системы.
3. **База данных:** центральное хранилище информации о студентах, очередях и отработках. Надежная система управления базами данных (СУБД) обеспечит целостность данных и быстрый доступ к ним.
4. **Веб-интерфейс для преподавателей:** позволит преподавателям управлять очередями, назначать время отработок и вести приём студентов. Веб-интерфейс будет разработан с учетом удобства использования и доступен из любого веб-браузера.

### 1.2.3 Основная проблематика

Основная проблема, на решение которой направлен данный проект, заключается в отсутствии автоматизации процесса управления очередями на отработки. В текущей системе организации отработок отсутствует автоматизация, что приводит к следующим негативным последствиям:

1. **Неэффективность:** процесс управления очередями вручную занимает много времени и усилий, что снижает общую продуктивность образовательного процесса.
2. **Ошибки:** человеческий фактор в управлении очередями часто приводит к ошибкам, что может вызвать путаницу и недовольство среди студентов и преподавателей.
3. **Задержки:** отсутствие оперативного информирования студентов о времени их отработки и их положении в очереди приводит к задержкам и дезорганизации процесса.
4. **Стресс и неудобства:** студенты вынуждены ожидать своей очереди в неудобных условиях, что создает дополнительный стресс и ухудшает общую атмосферу в учебных заведениях.

В рамках данной практической работы я постараюсь создать основу для автоматизированной системы управления очередями на отработки учебных долгов. Это позволит в дальнейшем доработать проект до полноценного решения, которое обеспечит более эффективное, удобное и прозрачное управление процессом отработки для всех участников образовательного процесса.

### 1.3 Заключение

В результате выполнения практики ожидается, что я смогу реализовать основной функционал как мобильного приложения, так и серверной части проекта, а также углубить свои знания и навыки в области разработки на платформе Android.

## 2. Цели и задачи практики

Основной целью моей практической работы является разработка базового функционала мобильного клиент-серверного приложения, включающего создание системы авторизации, управления очередями и уведомления пользователей, для оптимизации процесса отработки учебных задолженностей в образовательных организациях. Данная задача включает в себя следующие пункты:

1. **Настройка серверной инфраструктуры:**
  - Установка Docker и Docker-compose.
  - Настройка Docker для запуска MySQL и PHPMyAdmin для управления базой данных и удобства администрирования.
2. **Разработка серверной части (backend):**
  - Создание проекта с использованием фреймворка Spring Boot и библиотек Thymeleaf и Hibernate.

- Создание контролеров и представлений с использованием Thymeleaf для реализации системы авторизации.
- Разработка фильтров для ограничения доступа неавторизованным пользователям.
- Разработка контролеров и веб-интерфейса (представлений) для управления очередями на отработки (добавление, удаление, просмотр отработок).
- Создание модели очереди студентов и системы управления очередями (приём студентов, определение позиций в очереди).
- Добавление возможности импортировать списки записавшихся студентов из Google форм с автоматическим определением электронных адресов и ФИО в CSV файле.
- Разработка API для передачи информации между сервером и мобильным приложением.

### 3. Разработка мобильного приложения (Android):

- Реализация возможности идентификации студента по электронной почте с сохранением информации.
- Реализация возможности просмотра информации о отработках, на которые записан студент.
- Создание формы для просмотра информации о текущем положении студента в очереди в приложении.

## 3. Разработка (Основная часть)

### 3.1 Настройка серверной инфраструктуры

Для обеспечения функциональности и надёжности сервера было необходимо развернуть базу данных и систему для её управления. В качестве основы использовались технологии Docker и Docker-compose.

Для обеспечения функциональности и надёжности сервера было необходимо выполнить ряд задач. Сначала я проверил наличие Docker на сервере Ubuntu 22, используя команду `docker ps`, которая показала запущенные контейнеры и подтвердила корректную установку. Затем я установил Docker-compose, так как он не был предварительно установлен на сервере, с помощью команды `sudo apt install docker-compose`.

Для развёртывания базы данных и управления ею я создал файл конфигурации `docker-compose.yml`. В этом файле были определены два сервиса:

- **db:** контейнер с образом MySQL версии 8.4, настроенный с переменными окружения для задания паролей и имён баз данных.
- **phpmyadmin:** контейнер с образом PHPMyAdmin, предоставляющий веб-интерфейс для администрирования MySQL.

```
J AppMainController.java 1 application.properties M docker-compose.yml M X
D: > MyProjects > DebtClearFlow > docker-compose.yml
1 version: "3"
2 services:
3   db:
4     image: mysql:8.4
5     restart: always
6     environment:
7       MYSQL_ROOT_PASSWORD: debtClearTest!Passw0rd
8       MYSQL_DATABASE: debtClearFlow
9       MYSQL_USER: egor
10      MYSQL_PASSWORD: debtClearTest!Passw0rd
11   ports:
12     - "3307:3306"
13   phpmyadmin:
14     image: phpmyadmin/phpmyadmin:latest
15     restart: always
16     ports:
17       - "8081:80"
18     environment:
19       PMA_HOST: db
20       MYSQL_ROOT_PASSWORD: debtClearTest!Passw0rd
21
```

Рис 3.1.1 «Файл конфигурации docker-compose»

Затем я запустил созданные сервисы с помощью команды `docker-compose up -d` и проверил их статус снова при помощи `docker ps`, чтобы убедиться в успешном запуске контейнеров MySQL и PHPMyAdmin.

```
root@cold-rock:~/Documents/debtClearFlow# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
9a422ddc75f5   mysql:8.4          "docker-entrypoint.s..." About a minute ago Up 4 seconds   33060/tcp, 0.0.0.0:3307->3306/tcp, :::3307->3306/tcp debtclearflow_db_1
737ce7a0f2d8   phpmyadmin/phpmyadmin:latest "docker-entrypoint.s..." About a minute ago Up 4 seconds   0.0.0.0:8081->80/tcp, :::8081->80/tcp debtclearflow_phpmyadmin_1
50ea02633411   quay.io/outline/shadowbox:stable "docker-entrypoint.s..." 8 weeks ago    Up 5 minutes                                     shadowbox
9438a2250763   containrrr/watchtower "/watchtower --clean..." 2 months ago   Up 5 minutes (healthy) 8080/tcp      watchtower
```

Рис 3.1.2 «Результат выполнения команды docker ps»

## 3.2 Разработка серверной части (backend)

### 3.2.1 Создание проекта с использованием фреймворка Spring Boot и библиотек Thymeleaf и Hibernate

Для реализации серверной части приложения был создан проект Spring Boot с использованием OpenJDK 17. В проект были добавлены следующие ключевые зависимости, которые обеспечивают необходимую функциональность:

#### Зависимости в файле pom.xml:

- **spring-boot-starter-data-jpa:** Данная зависимость позволяет использовать JPA (Java Persistence API) для работы с базой данных. Включает в себя Hibernate для ORM (Object-Relational Mapping).
- **spring-boot-starter-thymeleaf:** Зависимость, предоставляющая поддержку Thymeleaf, шаблонизатора для создания веб-интерфейсов в Spring приложениях.
- **spring-boot-starter-web:** Данная зависимость обеспечивает базовую поддержку веб-приложений в Spring Boot, включая Spring MVC и встроенный сервер Tomcat.
- **spring-session-core:** Зависимость для управления сессиями в Spring Framework, обеспечивающая поддержку работы с сессиями пользователей.
- **mysql-connector-java:** JDBC драйвер для подключения к MySQL базе данных. Используется для взаимодействия с MySQL сервером.
- **spring-boot-starter-test:** Зависимость для тестирования Spring Boot приложений, включающая JUnit и другие необходимые библиотеки.



```

<dependencies> Add Spring Boot Starters...
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

```

Рис 3.2.1.1 «Зависимости в pom.xml»

Настройка файла application.properties для Spring Boot:

```

J AppMainController.java 1 application.properties M X docker-compose.yml M pom.xml
src > main > resources > application.properties
1  spring.application.name=debt-clear-flow-server
2  spring.jpa.hibernate.ddl-auto=update
3  spring.datasource.url=jdbc:mysql://5.182.86.164:3307/debtClearFlow
4  spring.datasource.username=egor
5  spring.datasource.password=debtClearTest!Passw0rd
6  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

```

Рис 3.2.1.2 «Конфигурация проекта Spring Boot»

В данной конфигурации указаны параметры для подключения к базе данных MySQL (url, username, password, driver-class-name) и настройки Hibernate (ddl-auto для автоматического обновления структуры БД). Эти настройки необходимы для правильной работы приложения с базой данных и оркестрации персистентности данных.

### 3.2.2 Реализации системы авторизации

Для реализации системы авторизации были выполнены следующие шаги:

В файле Teacher.java была определена сущность Teacher с использованием аннотаций JPA (@Entity, @Id, @GeneratedValue). Эта сущность представляет собой модель преподавателя с полями для идентификации, имени, логина, пароля и названия файла с аватаркой.

В интерфейсе TeacherRepo.java, расширяющем JpaRepository, определены методы для работы с данными преподавателей, включая поиск по логину (findByLogin).

Класс Init.java, реализующий CommandLineRunner, содержит логику для инициализации начальных данных. Если в базе данных отсутствует учетная запись с логином "test", то создается новый объект Teacher с данными "test" для имени, логина, пароля и названия файла с аватаркой.

В классе AuthController.java, аннотированном как @Controller, определены методы для отображения страницы входа (/login), выхода (/logout) и обработки попыток входа (/login). В методе tryToLogin производится попытка входа с проверкой логина и пароля в базе данных. В случае успеха данные сохраняются в сессии, и пользователь перенаправляется на главную страницу, иначе он остается на странице входа.

```
16 @Controller
17 public class AuthController {
18     @Autowired
19     private TeacherRepo teacherRepo;
20
21     @GetMapping("/")
22     public ModelAndView getGatePage(){
23         return new ModelAndView("gate");
24     }
25
26     @GetMapping("/login")
27     public ModelAndView getLoginView(){
28         return new ModelAndView("login");
29     }
30
31     @GetMapping("/logout")
32     public ModelAndView getLogoutView(HttpSession session){
33         session.invalidate();
34         return new ModelAndView("redirect:/");
35     }
36
37     @PostMapping("/login")
38     public ModelAndView tryToLogin(@RequestParam(required = true) String username, @RequestParam(required = true) String password, HttpSession session){
39         List<Teacher> check = teacherRepo.findByLoginAndPassword(username, password);
40         if (check.size()>=1){
41             session.setAttribute("login", username);
42             session.setAttribute("id", check.get(index:0).id);
43             return new ModelAndView("redirect:/");
44         }
45         else{
46             return new ModelAndView("login");
47         }
48     }
49 }
50 }
```

Рис 3.2.2.1 «Код контроллера авторизации»

На HTML странице login.html была разработана форма для входа, интегрированная с системой авторизации приложения. Форма предоставляет поля для ввода логина и пароля, которые передаются на серверный контроллер при отправке формы методом POST. При успешной аутентификации пользователь перенаправляется на главную страницу приложения.

На HTML странице layout.html было создано меню навигации, интегрированное с основными функциональными возможностями приложения. Меню включает ссылки на страницы "Мои отработки" и "Создать отработку", доступ к которым зависит от статуса авторизации пользователя. Для неаутентифицированных пользователей доступна ссылка на страницу входа, а для авторизованных — возможность выхода из системы.

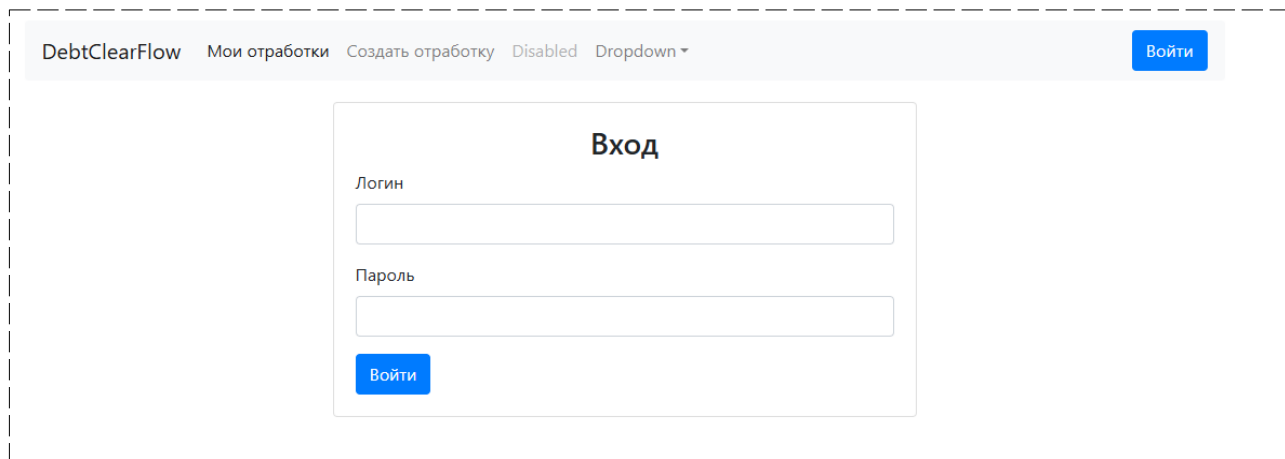
The screenshot shows the login form of the DebtClearFlow application. At the top, there is a navigation bar with the application name "DebtClearFlow" and several menu items: "Мои отработки", "Создать отработку", "Disabled", and "Dropdown" with a downward arrow. On the right side of the navigation bar is a blue button labeled "Войти". Below the navigation bar is a central white box with the title "Вход" (Login). Inside this box, there are two input fields: "Логин" (Login) and "Пароль" (Password). Below the password field is a blue button labeled "Войти" (Login).

Рис 3.2.2.2 «Форма авторизации и меню»

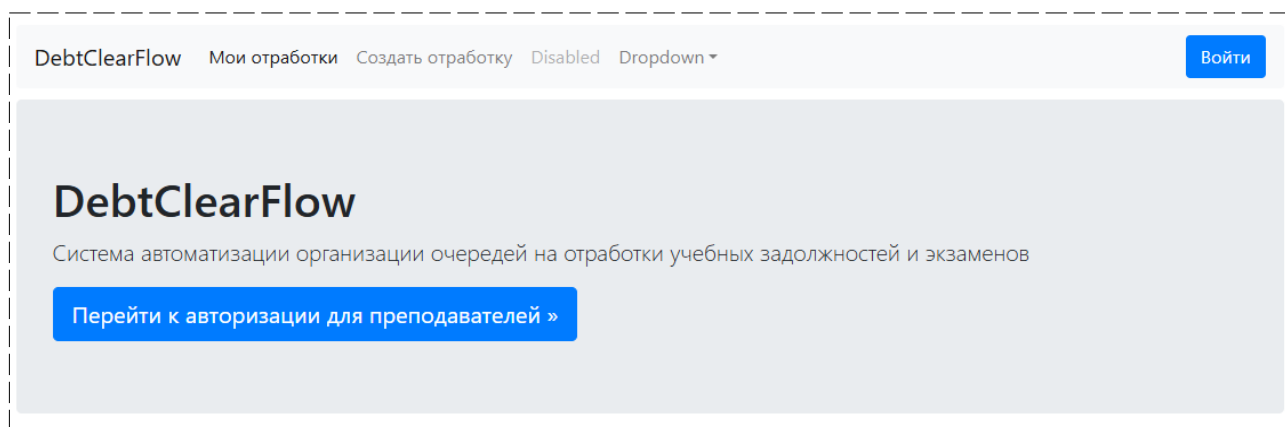
The screenshot shows the main page for non-authenticated users. At the top, there is a navigation bar with the application name "DebtClearFlow" and several menu items: "Мои отработки", "Создать отработку", "Disabled", and "Dropdown" with a downward arrow. On the right side of the navigation bar is a blue button labeled "Войти". Below the navigation bar is a large light blue box with the application name "DebtClearFlow" in a large font. Below the name is a subtitle: "Система автоматизации организации очередей на отработки учебных задолжностей и экзаменов". At the bottom of this box is a blue button labeled "Перейти к авторизации для преподавателей »" (Go to authentication for teachers »).

Рис 3.2.2.3 «Главная страница неавторизованного пользователя»

Для проверки функциональности системы авторизации был создан тестовый аккаунт, который автоматически создается при запуске сервера. Этот аккаунт используется для проверки процесса входа в систему и его работоспособности.

На странице `gate.html` была разработана главная страница приложения DebtClearFlow. Эта страница представляет собой точку входа для пользователей и содержит основные элементы интерфейса. Страница включает в себя заголовок "DebtClearFlow" с описанием системы, которая автоматизирует управление очередями на отработки учебных задолженностей и экзаменов. Для удобства пользователей на странице размещены кнопки, позволяющие перейти к авторизации для преподавателей или к панели управления в зависимости от их текущей сессии. Неаутентифицированным пользователям доступна кнопка "Перейти к авторизации для преподавателей", а авторизованным — кнопка "Перейти к панели для преподавателей".

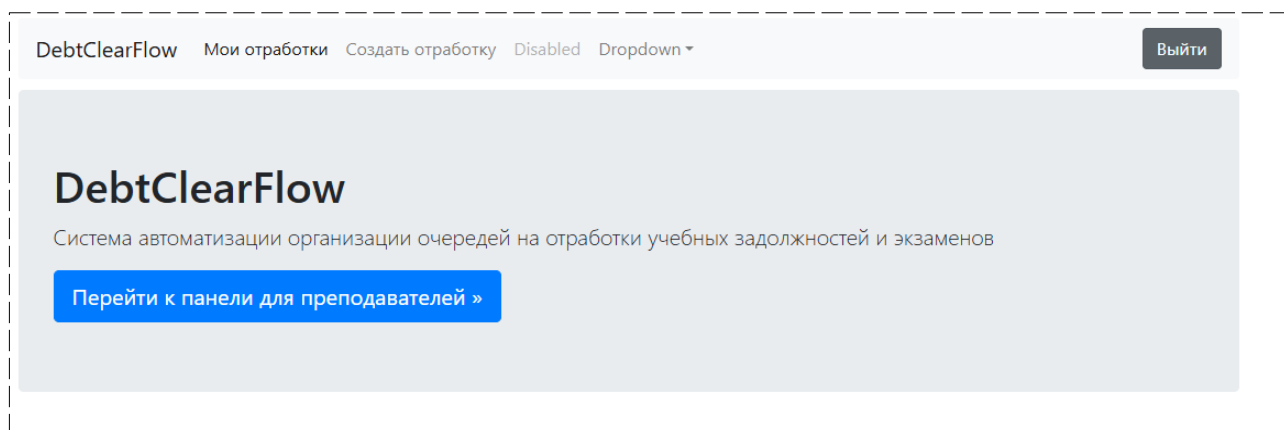


Рис 3.2.2.4 «Главная страница авторизованного пользователя»

### 3.2.3 Разработка фильтров для ограничения доступа неавторизованным пользователям

Для обеспечения безопасности и контроля доступа в приложении DebtClearFlow был разработан фильтр `AuthFilter`. Этот фильтр реализован в классе `AuthFilter.java` и интегрирован в конфигурацию приложения с помощью класса `WebConfig.java`.

#### Описание функциональности:

Фильтр `AuthFilter` реализует интерфейс `OncePerRequestFilter` и обрабатывает каждый HTTP-запрос.

1. **Проверка сессии:** В методе `doFilterInternal`, фильтр извлекает текущую сессию пользователя из запроса. Проверяется наличие атрибутов сессии `login` и `id`, что указывает на авторизованного пользователя.
2. **Доступ к ресурсам:** Если пользователь авторизован (атрибуты `login` и `id` не равны `null`), фильтр передает запрос дальше по цепочке фильтров с помощью `filterChain.doFilter(request, response)`.
3. **Перенаправление на страницу входа:** Если пользователь не авторизован, фильтр перенаправляет запрос на страницу `/login` с помощью `response.sendRedirect("/login")`. Это означает, что неавторизованным пользователям ограничен доступ к защищенным ресурсам приложения.

```

GigaCode: explain | explain step by step | doc | test
@Component
public class AuthFilter extends OncePerRequestFilter{

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

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull FilterChain filterChain) throws ServletException {
        HttpSession session = request.getSession();

        if (session != null && session.getAttribute("login") != null && session.getAttribute("id") != null){
            logger.info("User is logged in with name = " + session.getAttribute("login").toString());
            filterChain.doFilter(request, response);
        }
        else{
            logger.info("User is not logged in");
            response.sendRedirect("/login");
        }
    }
}

```

Рис 3.2.3.1 «Код фильтра»

## Интеграция фильтра в приложение:

Класс WebConfig.java выполняет конфигурацию фильтра с помощью FilterRegistrationBean. Фильтр AuthFilter применяется к URL-шаблону /panel/\*, что означает, что он активируется для всех запросов, начинающихся с /panel/.

```

@Configuration
public class WebConfig {

    // Устанавливает режим отслеживания сессии на COOKIE.
    @Bean
    public ServletContextInitializer servletContextInitializer() {
        HashSet<SessionTrackingMode> set = new HashSet<SessionTrackingMode>();
        set.add(SessionTrackingMode.COOKIE);
        return servletContext -> {
            servletContext.setSessionTrackingModes(set);
        };
    }

    @Bean
    @Autowired
    public FilterRegistrationBean<AuthFilter> regAuthFilter(AuthFilter authFilter) {
        FilterRegistrationBean<AuthFilter> registrationBean = new FilterRegistrationBean<>();
        registrationBean.setFilter(authFilter);
        registrationBean.addUrlPatterns("/panel/*");
        registrationBean.setOrder(0);
        return registrationBean;
    }
}

```

Рис 3.2.3.2 «Код регистрации фильтра»

## Проверка работоспособности:

Для проверки работоспособности фильтра AuthFilter, в коде были добавлены логи для вывода информации в консоль о его действиях. Это позволяет отслеживать, какие пользователи проходят аутентификацию, а также какие запросы блокируются в случае отсутствия аутентификации.

```

2024-07-13T16:59:20.977+03:00 INFO 4508 --- [debt-clear-flow-server] [
to enable JTA platform integration)
2024-07-13T16:59:21.146+03:00 INFO 4508 --- [debt-clear-flow-server] [
main] o.h.e.t.j.p.i.JtaPlatformInitiator : H#000489: No JTA platform available (set 'hibernate.transaction.jta.platform'
2024-07-13T16:59:21.392+03:00 INFO 4508 --- [debt-clear-flow-server] [
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-07-13T16:59:21.392+03:00 INFO 4508 --- [debt-clear-flow-server] [
main] o.s.d.j.r.query.QueryEnhancerFactory : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-07-13T16:59:21.766+03:00 WARN 4508 --- [debt-clear-flow-server] [
main] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
be performed during view rendering. Explicitly configure spring.jpa.open-in-view
2024-07-13T16:59:22.380+03:00 INFO 4508 --- [debt-clear-flow-server] [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-07-13T16:59:22.389+03:00 INFO 4508 --- [debt-clear-flow-server] [
main] c.v.d.DebtClearFlowServerApplication : Started DebtClearFlowServerApplication in 5.06 seconds (process running for 5.
296)
2024-07-13T16:59:22.586+03:00 INFO 4508 --- [debt-clear-flow-server] [
main] c.v.d.Misc.InitMisc : Test account already exists
2024-07-13T16:59:26.574+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] o.a.c.c.c.[Tomcat].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-07-13T16:59:26.574+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-07-13T16:59:26.573+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2024-07-13T16:59:26.598+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.Controllers.Filters.AuthFilter : User is not logged in

```

Рис 3.2.3.2 «Логи попытки доступа неавторизованного пользователя»

```

to enable JPA platform integration)
2024-07-13T16:59:21.146+03:00 INFO 4508 --- [debt-clear-flow-server] [ main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-07-13T16:59:21.302+03:00 INFO 4508 --- [debt-clear-flow-server] [ main] o.s.d.j.r.query.QueryEnhancerFactory : Hibernate is in classpath; if applicable, HQL parser will be used.
2024-07-13T16:59:21.766+03:00 WARN 4508 --- [debt-clear-flow-server] [ main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-07-13T16:59:22.380+03:00 INFO 4508 --- [debt-clear-flow-server] [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-07-13T16:59:22.389+03:00 INFO 4508 --- [debt-clear-flow-server] [ main] c.v.d.DebtClearFlowServerApplication : Started DebtClearFlowServerApplication in 5.06 seconds (process running for 5.
296)
2024-07-13T16:59:22.586+03:00 INFO 4508 --- [debt-clear-flow-server] [ main] c.v.d.Misc.InitMisc : Test account already exists
2024-07-13T16:59:26.572+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-07-13T16:59:26.572+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-07-13T16:59:26.573+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2024-07-13T16:59:26.598+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.Controllers.Filters.AuthFilter : User is not logged in
2024-07-13T17:00:27.719+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-6] c.v.d.Controllers.Filters.AuthFilter : User is logged in with name = test

```

Рис 3.2.3.2 «Логи попытки доступа авторизованного пользователя»

### 3.2.4 Разработка контролеров и веб-интерфейса (представлений) для управления очередями на отработки (добавление, удаление, просмотр отработок)

Для управления очередями на отработки в приложении DebtClearFlow были разработаны контролеры и веб-интерфейс. Эти компоненты обеспечивают функциональность для добавления, удаления и просмотра отработок.

#### Модель DebtRepayment

Модель DebtRepayment представляет собой сущность отработки задолженности. В ней содержатся поля для идентификатора отработки, названия, кабинета, времени начала и окончания, логинов преподавателей, участвующих в отработке, и статуса доступности отработки. Класс также включает аннотации для валидации полей, такие как `@NotBlank` и `@Size`.

#### Репозиторий DebtRepaymentRepo

Интерфейс DebtRepaymentRepo расширяет JpaRepository и предоставляет методы для работы с сущностью DebtRepayment. Дополнительно, он содержит метод `findByTeachersLoginsContaining`, который позволяет находить отработки, к которым имеет доступ определенный преподаватель.

#### Контролер RepaymentController

Контролер RepaymentController предоставляет методы для обработки HTTP-запросов, связанных с управлением отработками.

- **Создание новой отработки:** Метод `getCreateRepayment` возвращает представление для создания новой отработки. Метод `postCreateRepayment` обрабатывает POST-запросы для создания новой отработки. В этом методе происходит валидация данных, и если данные проходят валидацию, то отработка сохраняется в базе данных. Логин преподавателя, создавшего отработку, добавляется к списку логинов в поле `teachersLogins`.
- **Просмотр отработок:** Метод `getmyRepayments` возвращает представление со списком всех отработок, созданных преподавателем или к которым он имеет доступ. Этот метод использует метод `findByTeachersLoginsContaining` из репозитория DebtRepaymentRepo для получения соответствующих записей.

- **Удаление отработки:** Метод deleteRepayment обрабатывает GET-запросы для удаления отработки по ее идентификатору. После удаления отработки происходит перенаправление на страницу со списком отработок.

```
private ModelAndView createRepayment(){
    var mv = new ModelAndView("createRepayment");
    mv.addObject("repayment", new DebtRepayment());
    return mv;
}

@GetMapping("createRepayment")
public ModelAndView getCreateRepayment(){
    return createRepayment();
}

// создание новой отработки
@PostMapping(path = "createRepayment", method=RequestMethod.POST, consumes = MediaType.APPLICATION_FORM_URLENCODED_VALUE)
public ModelAndView postCreateRepayment(@Valid @ModelAttribute DebtRepayment repayment, BindingResult bindingResult, HttpSession session){
    logger.info("Got model in 'postCreateRepayment': " + repayment.toString());
    if (bindingResult.hasErrors()){
        logger.error("Data didn't pass validation in 'createRepayment'");
        var errors = bindingResult.getAllErrors();
        for (var error : errors){
            logger.error(error.getDefaultMessage() + ":" + error.getObjectName());
        }
    }
    else{
        logger.info("Data passed validation in 'createRepayment'");
        // добавляем логин преподавателя, который создал эту отработку
        repayment.setTeachersLogins(repayment.getTeachersLogins()+" , " + session.getAttribute("login").toString());
        debtRepaymentRepo.save(repayment);
        logger.info("Saved model!");
    }
    return createRepayment();
}

// просмотр всех отработок созданных преподавателем или к которым он имеет доступ
@GetMapping("myRepayments")
public ModelAndView getmyRepayments(HttpSession session){
    List<DebtRepayment> debtRepayments = debtRepaymentRepo.findByTeachersLoginsContaining(session.getAttribute("login").toString());
    return new ModelAndView("myRepayments", "debtRepayments", debtRepayments);
}

@RequestMapping(value = "/delete/{rpid}", method = RequestMethod.GET)
public ModelAndView deleteRepayment(@PathVariable("rpid") Integer id){
    logger.info("Got model in 'deleteRepayment': " + id.toString());
    debtRepaymentRepo.deleteById(id);
    logger.info("Deleted model!");
    return new ModelAndView("redirect:/panel/myRepayments");
}
```

Рис 3.2.4.1 «Код методов в контролере „RepaymentController“»

## Веб-интерфейс

Для взаимодействия с пользователями был разработан веб-интерфейс с использованием шаблонизатора Thymeleaf. Основные представления включают страницы для создания новой отработки и просмотра списка отработок. Эти страницы содержат формы и таблицы для отображения и ввода данных, обеспечивая удобство работы преподавателей с системой управления отработками.



DebtClearFlow   Мои отработки   Создать отработку   Disabled   Dropdown ▾								Выйти
ID	Название	Кабинет	Время начала	Время окончания	Логины преподавателей	Открыто	Действия	
1	Физика3434	Л207	2024-07-13T17:01	2024-07-23T17:01	test2, test	false	Начать приём Удалить	Списки
2	Ещё физика	Л507	2024-07-13T18:57	2024-07-13T21:00	test4, test	false	Начать приём Удалить	Списки

Рис 3.2.4.2 «Страница со списком отработок к которым преподаватель имеет доступ»



DebtClearFlow   Мои отработки   Создать отработку   Disabled   Dropdown ▾		Выйти
Название		<input type="text" value="Введите название отработки"/>
Кабинет		<input type="text" value="Введите номер кабинета"/>
Время начала		<input type="text" value="дд.мм.гггг --:--"/> 
Время окончания		<input type="text" value="дд.мм.гггг --:--"/> 
Логины преподавателей		<input type="text" value="Введите логины преподавателей, разделенные запятой"/>
Создать отработку		

Рис 3.2.4.3 «Форма для создания новой отработки»

### 3.2.5 Создание модели очереди студентов и системы управления очередями (приём студентов, определение позиций в очереди). Добавление возможности импортировать списки записавшихся студентов из Google форм

Для реализации системы управления очередями студентов в проекте DebtClearFlow были разработаны соответствующие модели, репозитории и методы контроллеров. Эти компоненты обеспечивают функциональность по приему студентов в очередь, определению их позиций и импортированию списков из Google форм.

#### Модель QStudent

Модель QStudent представляет собой сущность студента, добавленного в очередь на отработку. В ней содержатся поля для идентификатора студента, его полного имени, электронной почты, описания работы, идентификатора очереди на отработку, статусов принятия и нахождения в процессе приема, а также логина преподавателя, который принимает данного студента. В классе реализован метод checkEmailAndFullName, который проверяет наличие полного имени и электронной почты у студента.



## Репозиторий QStudentRepo

Интерфейс QStudentRepo расширяет JpaRepository и предоставляет методы для работы с сущностью QStudent. Среди методов репозитория есть такие, как:

- findByDebtRepaymentIdOrderByIdAsc - поиск студентов, записанных на определенную очередь, с упорядочиванием по идентификатору.
- findByDebtRepaymentIdAndIsAcceptedFalseOrderByIdAsc - поиск студентов, которые еще не были приняты.
- findByEmail - поиск студентов по электронной почте.
- findFirstByDebtRepaymentIdAndIsAcceptedFalseAndIsInProcessFalseOrderByIdAsc - поиск первого студента в очереди, который еще не был принят и не находится на приеме.
- findByDebtRepaymentIdAndTeacherLoginAndIsInProcessTrueAndIsAcceptedFalse - поиск студентов, которые в данный момент находятся на приеме у конкретного преподавателя.

```
src > main > java > com > vityazev_igor > debt_clear_flow_server > Models > J QStudentRepo.java > ...
1 package com.vityazev_igor.debt_clear_flow_server.Models;
2 import org.springframework.data.jpa.repository.JpaRepository;
3 import org.springframework.data.jpa.repository.Query;
4 import org.springframework.data.repository.query.Param;
5
6 import java.util.List;
7 public interface QStudentRepo extends JpaRepository<QStudent, Integer>{
8     // ищем студентов, который записаны на определенную очередь
9     public List<QStudent> findByDebtRepaymentIdOrderByIdAsc(Integer debtRepaymentId);
10
11     // ищем студентов которые сейчас находятся в очереди
12     public List<QStudent> findByDebtRepaymentIdAndIsAcceptedFalseOrderByIdAsc(Integer debtRepaymentId);
13
14     // найти всех студентов с определённой почтой
15     public List<QStudent> findByEmail(String email);
16
17     // найти следующего студента в очереди который ещё не был принят и не находится на приёме
18     public QStudent findFirstByDebtRepaymentIdAndIsAcceptedFalseAndIsInProcessFalseOrderByIdAsc(Integer debtRepaymentId);
19
20     // ищем студента который в данный момент находится на приём
21     public List<QStudent> findByDebtRepaymentIdAndTeacherLoginAndIsInProcessTrueAndIsAcceptedFalse(Integer debtRepaymentId, String teacherLogin);
22
23     @Query("SELECT s FROM QStudent s WHERE s.debtRepaymentId = :debtRepaymentId AND s.teacherLogin = :teacherLogin AND s.isInProcess = true AND s.isAccepted = false")
24     public QStudent findCurrentStudent(@Param("debtRepaymentId") Integer debtRepaymentId, @Param("teacherLogin") String teacherLogin);
25
26 }
```

Рис 3.2.5.1 «Код репозитория „QStudentRepo“»

## Методы контроллера RepaymentController

Для работы с очередями студентов в RepaymentController были добавлены следующие методы:

- **Просмотр очереди:** Метод getViewRepaymentModelById возвращает модель представления для просмотра конкретной очереди, включая информацию об отработке и список студентов. Этот метод используется в методе viewRepayment, который обрабатывает GET-запросы для отображения очереди по ее идентификатору.
- **Добавление студентов из CSV:** Метод addStudents обрабатывает POST-запросы для добавления студентов в очередь из CSV-файла. В этом методе используется библиотека Apache Commons CSV для чтения данных из файла. Студенты добавляются в очередь, если у них указаны корректные полное имя и электронная почта. Также метод поддерживает добавление описания работы из указанного столбца CSV-файла.

- **Очистка очереди:** Метод `clearAll` обрабатывает GET-запросы для очистки списка студентов в очереди. Он удаляет всех студентов, записанных на конкретную обработку, и перенаправляет пользователя на страницу просмотра очереди.

```

public ModelAndView addStudents(@RequestPart MultipartFile csvFile, @RequestPart String descriptionColumnName, @PathVariable("rpid") Integer Id){
    if (csvFile.getSize() > 0){
        logger.info("Got file! File size = " + csvFile.getSize());
        String fileName = java.util.UUID.randomUUID().toString() + ".csv";
        File file = Paths.get(Shared.csvFilesDirectory.toString(), fileName).toFile();
        try {
            csvFile.transferTo(file);
            logger.info("Saved csv file as "+fileName);

            FileReader in = new FileReader(file, StandardCharsets.UTF_8);
            Iterable<CSVRecord> records = CSVFormat.DEFAULT.builder().setHeader().setSkipHeaderRecord(true).setTrim(true).build().parse(in);
            for (CSVRecord record : records) {
                List<String> columnsData = record.toList();
                var qStudent = new QStudent();
                qStudent.setDebtRepaymentId(Id);

                for (String data : columnsData){
                    if (Shared.isEmail(data)){
                        qStudent.setEmail(data);
                    }
                    if (Shared.isFullName(data)){
                        qStudent.setFullName(data);
                    }
                }

                if (!descriptionColumnName.isBlank()){
                    qStudent.setWorkDescription(record.get(descriptionColumnName));
                }

                if (qStudent.checkEmailAndFullName()){
                    qStudentRepo.save(qStudent);
                    logger.info("Saved qStudent!");
                    System.out.print(qStudent.toString());
                }
            }
        } catch (IllegalStateException e) {
            logger.error("Can't transfer csv file", e);
        } catch (IOException e) {
            logger.error("Can't write csv file", e);
        }
    }
    else{
        logger.error("Can't get file");
    }
    return getViewRepaymentModelById(Id);
}

```

Рис 3.2.5.2 «Код метода для добавление студентов из CSV файла»

## Представления

Для всех методов контроллера были разработаны соответствующие HTML-страницы с использованием Thymeleaf. Эти представления обеспечивают пользователю удобный интерфейс для взаимодействия с системой управления очередями. Среди них страницы для просмотра очередей, добавления студентов из CSV-файлов и очистки очередей.

DebtClearFlow
Мои отработки
Создать отработку
Disabled
Dropdown
Выйти

Добавить новых студентов на отработку Физика3434

Выберите CSV файл

Выбор файла
Не выбран ни один файл

Название столбца с описанием работы

Добавить студентов
Очистить список студентов

ID	Фамилия	Описание работы
1	Витязев Егор Денисович	ЛР-1-07
2	Егор Федюлин Иванов	ЛР1-07, ЛР2-12

Рис 3.2.5.3 «Просмотр списка студентов»

## Проверка работоспособности

Работоспособность системы управления очередями была проверена путем вывода сообщений в консоль о выполнении действий. Это позволило убедиться в корректной работе всех операций, таких как добавление, удаление и просмотр студентов в очереди, а также импорт данных из CSV-файлов.

```

2024-07-13T19:21:31.480+03:00 INFO 4508 --- [debt-clear-flow-server] [io-8080-exec-10] c.v.d.Controllers.Filters.AuthFilter : User is logged in with name = test
2024-07-13T19:21:31.483+03:00 INFO 4508 --- [debt-clear-flow-server] [io-8080-exec-10] c.v.d.C.W.RepaymentController : Got model in 'viewRepayment': 1
2024-07-13T19:21:54.448+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.Controllers.Filters.AuthFilter : User is logged in with name = test
2024-07-13T19:21:54.478+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.C.W.RepaymentController : Got file! File size = 336
2024-07-13T19:21:54.481+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.C.W.RepaymentController : Saved csv file as 22177061-566d-4b22-98af-7030de73b00f.csv
2024-07-13T19:21:54.985+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.C.W.RepaymentController : Saved qStudent!
id: 1
fullName: Витязев Егор Денисович
email: m2103087@edu.misis.ru
workDeskription: ЛР-1-07
debtRepaymentId: 1
isAccepted: false
isInProcess: false
teacherLogin: null
2024-07-13T19:21:55.406+03:00 INFO 4508 --- [debt-clear-flow-server] [nio-8080-exec-1] c.v.d.C.W.RepaymentController : Saved qStudent!
id: 2
fullName: Егор Федюлин Иванов
email: e.vityazev.e@gmail.com
workDeskription: ЛР1-07, ЛР2-12
debtRepaymentId: 1
isAccepted: false
isInProcess: false
teacherLogin: null

```

Рис 3.2.5.4 «Логи показывающие успешное добавление студентов из CSV файла»

Для реализации функциональности приема студентов был создан отдельный контроллер ReceptionController и соответствующее представление. Этот компонент системы позволяет преподавателю управлять очередью студентов и принимать их на отработку в определенном порядке.

Контроллер ReceptionController управляет процессом приема студентов и взаимодействует с моделью и репозиторием очереди студентов (QStudent) и отработок (DebtRepaument). В контроллере есть два основных метода.

- Метод getReception:** обрабатывает GET-запросы по пути /panel/reception/{rpid} и возвращает представление для приема студентов. Сначала он получает логин преподавателя из сессии, затем ищет текущего студента на приеме, используя метод

findCurrentStudent репозитория QStudentRepo, и добавляет информацию о текущем студенте и отработке в модель представления.

- **Метод getReceptionNext:** обрабатывает GET-запросы по пути /panel/reception/{rpid}/next и переключает текущего студента на следующего в очереди. Сначала он получает логин преподавателя из сессии, затем обновляет статус текущего студента, помечая его как принятого и больше не находящегося на приеме. Далее он ищет следующего студента, который еще не был принят и не находится на приеме, используя метод findFirstByDebtRepaymentIdAndIsAcceptedFalseAndIsInProcessFalseOrderByIdAsc, и обновляет его статус, помечая его как находящегося на приеме и сохраняя логин преподавателя, который его принимает. После этого происходит перенаправление на страницу приема студентов.

```
@Controller
@RequestMapping(path = "/panel/")
public class ReceptionController {

    @Autowired
    DebtRepaymentRepo debtRepaymentRepo;

    @Autowired
    QStudentRepo qStudentRepo;

    @RequestMapping(value = "/reception/{rpid}", method = RequestMethod.GET)
    public ModelAndView getReception(@PathVariable Integer rpid, HttpSession session){
        String teacherLogin = (String)session.getAttribute("login");
        var mav = new ModelAndView("reception");

        // в начале мы берём текущего студента на приеме и добавляем его к представлению
        QStudent currentStudent = qStudentRepo.findCurrentStudent(rpid, teacherLogin);
        mav.addObject("currentStudent", currentStudent);

        // ну и инфу о самой отработке лишней не будет
        mav.addObject("repayment", debtRepaymentRepo.findById(rpid).get());
        return mav;
    }

    @RequestMapping(value = "/reception/{rpid}/next", method = RequestMethod.GET)
    public ModelAndView getReceptionNext(@PathVariable Integer rpid, HttpSession session){
        String teacherLogin = (String)session.getAttribute("login");
        // если текущий студент был то мы обновляем инфу о том что он был принят
        QStudent currentStudent = qStudentRepo.findCurrentStudent(rpid, teacherLogin);
        if (currentStudent != null){
            currentStudent.setIsAccepted(isAccepted:true);
            currentStudent.setIsInProcess(isInProcess:false);
            qStudentRepo.save(currentStudent);
        }

        QStudent nextStudent = qStudentRepo.findFirstByDebtRepaymentIdAndIsAcceptedFalseAndIsInProcessFalseOrderByIdAsc(rpid);
        if (nextStudent != null){
            nextStudent.setIsInProcess(isInProcess:true);
            nextStudent.setTeacherLogin(teacherLogin);
            qStudentRepo.save(nextStudent);
        }

        return new ModelAndView("redirect:/panel/reception/" + rpid.toString());
    }
}
```

Рис 3.2.5.5 «Код контролера „ReceptionController“»

Для метода `getReception` было разработано соответствующее HTML-представление с использованием Thymeleaf. В этом представлении отображается заголовок с названием отработки, форма для отображения информации о текущем студенте (ФИО и описание задолженности), если такой студент существует, сообщение об ошибке, если текущий студент отсутствует, и кнопка для перехода к следующему студенту в очереди.

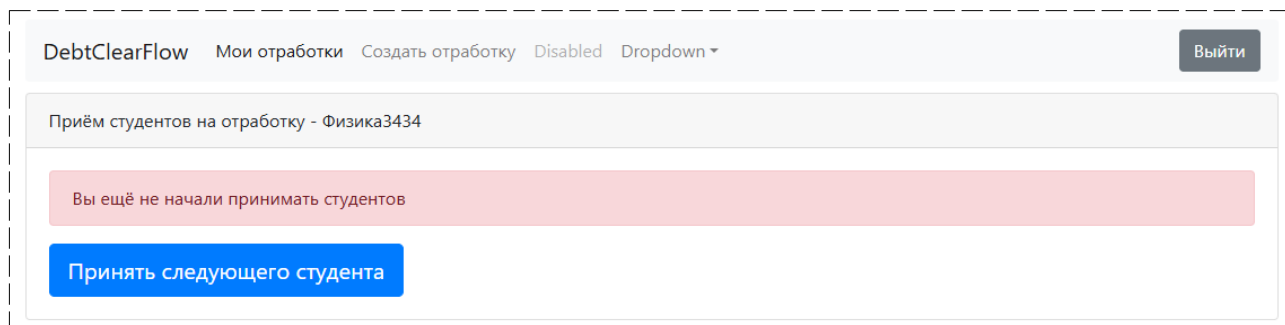


Рис 3.2.5.6 «Страница с контролем очереди (приём ещё не начался)»

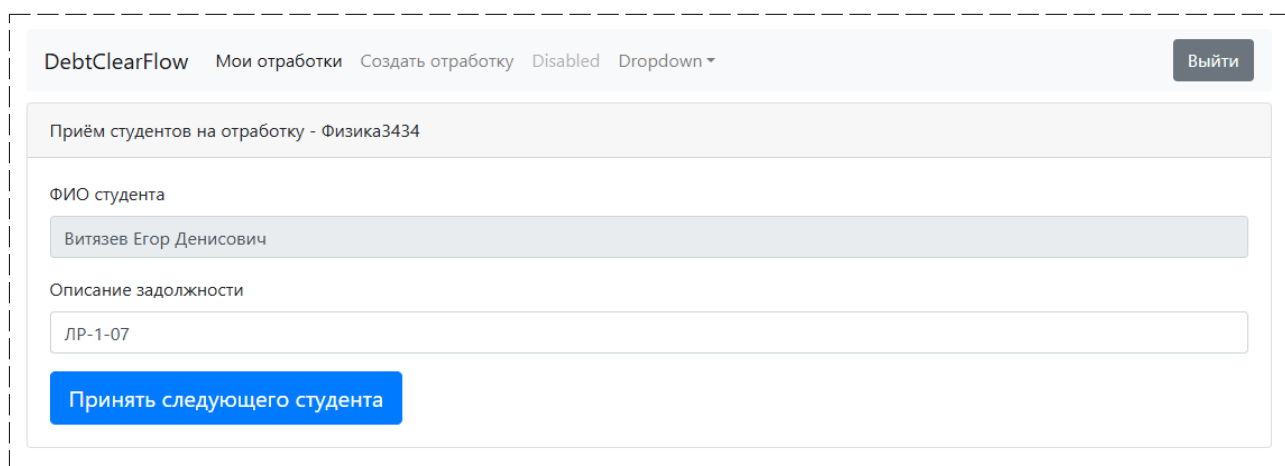


Рис 3.2.5.7 «Страница с контролем очереди (принимается студент)»

### 3.2.6 Разработка API для передачи информации между сервером и мобильным приложением

Для интеграции мобильного приложения с сервером был разработан контроллер `AppMainController`, который предоставляет RESTful API для передачи данных. Этот контроллер содержит методы для получения информации о записанных на отработку студентах и определения их позиции в очереди.

Контроллер `AppMainController` аннотирован с использованием `@RestController` и `@RequestMapping`, что позволяет обрабатывать HTTP-запросы по указанным путям.

Первый метод, `findReceptions`, обрабатывает POST-запросы по пути `/api/findReceptions`. Этот метод принимает параметр `email` через `@RequestPart` и возвращает список отработок, на которые записан студент с указанным `email`. В методе осуществляется поиск всех студентов с данным `email` в репозитории `QStudentRepo`. Для каждого найденного студента извлекается информация о соответствующей отработке из репозитория `DebtRepaymentRepo`, и если такая отработка существует, она добавляется в результирующий список. Этот список и возвращается в ответе.

Второй метод, `findPosition`, обрабатывает POST-запросы по пути `/api/findPosition`. Этот метод принимает параметры `email` и `repaymentId` через `@RequestPart` и возвращает позицию студента в очереди на обработку. Сначала метод ищет текущую очередь студентов для указанной обработки, используя метод `findByDebtRepaymentIdAndIsAcceptedFalseOrderByIdAsc` репозитория `QStudentRepo`. Затем среди этой очереди ищется студент с указанным `email`. Если такой студент найден, возвращается его позиция в очереди (индекс плюс один), иначе возвращается `-1`, что означает, что студент прошел очередь.

Таким образом, разработанное API предоставляет мобильному приложению возможность взаимодействовать с сервером для получения информации об отработках и текущей позиции студента в очереди.

API было проверено на работоспособность с использованием инструмента POSTMAN. Все запросы выполнялись корректно, и ответы сервера соответствовали ожиданиям, что подтверждает правильность реализации и надежность разработанного функционала.

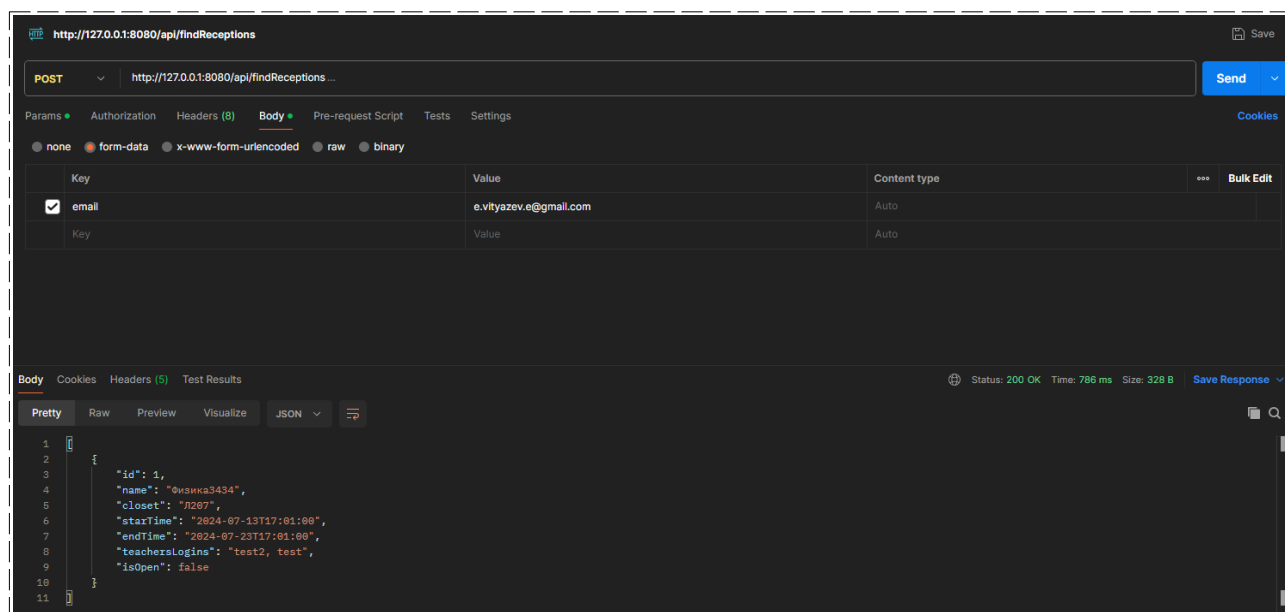


Рис 3.2.6.1 «Проверка работоспособности API»

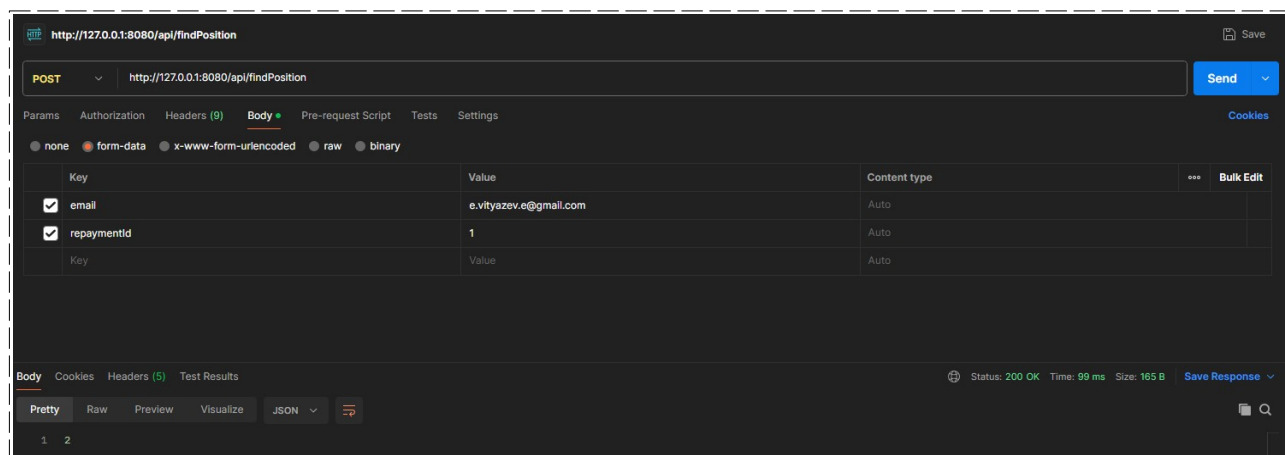


Рис 3.2.6.2 «Проверка получения положение в очереди через API»

### 3.3 Разработка мобильного приложения (Android)

#### 3.3.1 Реализация возможности идентификации студента по электронной почте с сохранением информации

В рамках разработки мобильного приложения для платформы Android была реализована функциональность, позволяющая студентам идентифицироваться по электронной почте и сохранять эту информацию для последующего использования. Этот процесс был выполнен в главной активности приложения MainActivity.

В разметке activity\_main.xml была создана основная структура пользовательского интерфейса. В верхней части экрана располагается Toolbar с заголовком "DebtClearFlow". Под ним размещено текстовое поле EditText для ввода учебной электронной почты и кнопка Button для подтверждения и сохранения введенной информации. Компоненты интерфейса расположены с использованием ConstraintLayout, что обеспечивает гибкую и адаптивную верстку.

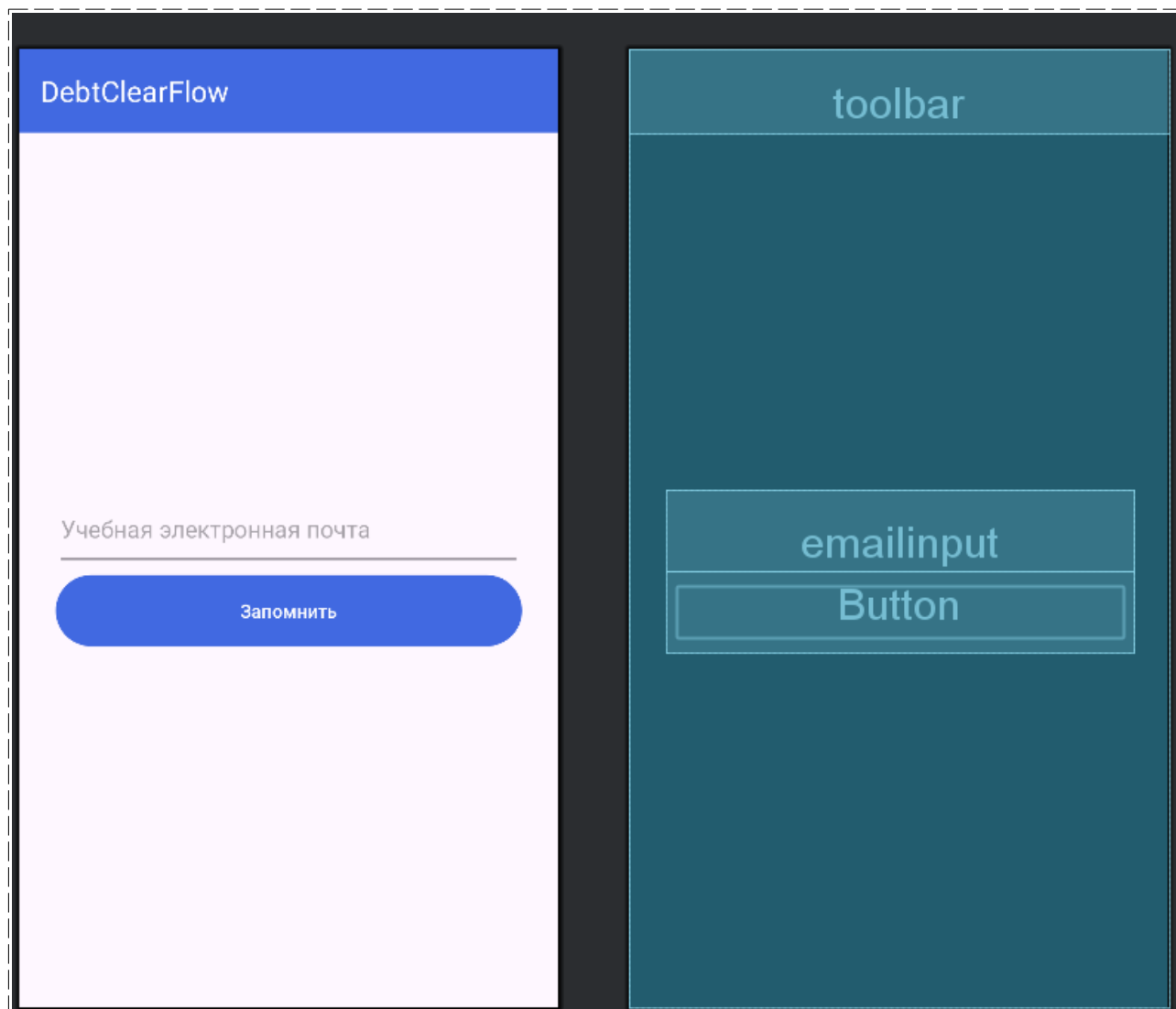


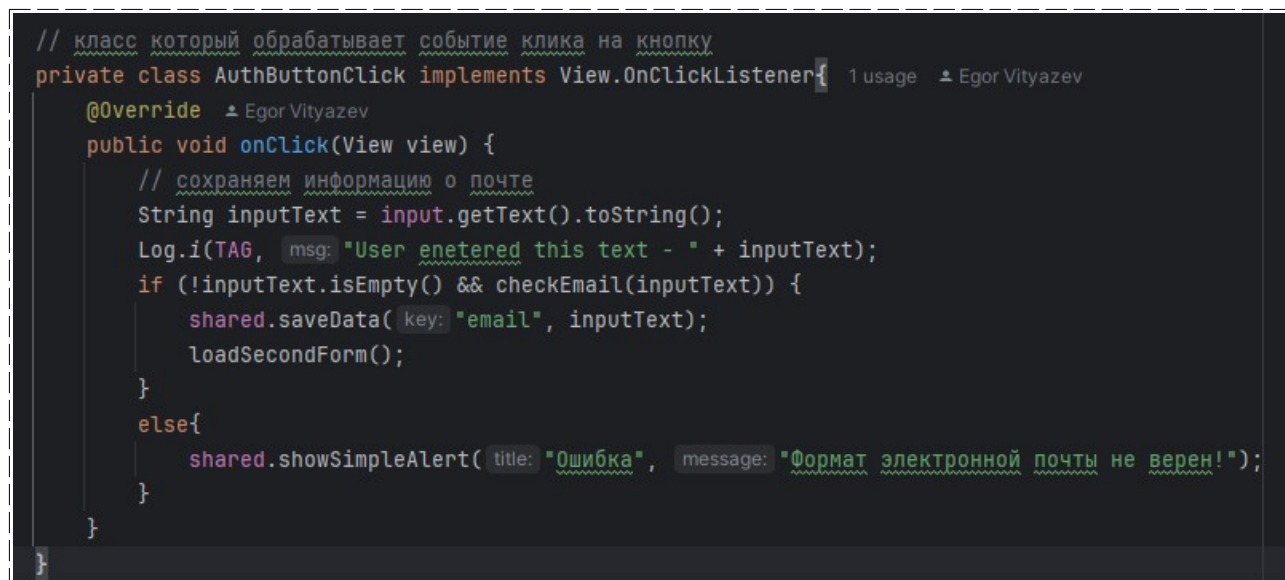
Рис 3.3.1.1 «Интерфейс формы идентификации студента в приложении»



Класс MainActivity отвечает за обработку логики, связанной с вводом и сохранением электронной почты студента. В методе onCreate происходит инициализация активности и настройка пользовательского интерфейса. Использование библиотеки EdgeToEdge обеспечивает корректное отображение элементов интерфейса с учетом системных отступов.

В начале метода onCreate происходит инициализация объекта Shared, который используется для работы с сохраненными данными. Также выполняется проверка наличия сохраненного email. Если email уже сохранен, происходит переход к следующей активности с помощью метода loadSecondForm.

Основная логика обработки ввода электронной почты реализована в классе AuthButtonClick, который является обработчиком кликов для кнопки. При нажатии на кнопку считывается введенный текст из поля EditText, и если он не пустой и соответствует формату электронной почты (содержит символы "@" и "."), то информация сохраняется с использованием метода saveData объекта Shared. В случае успешного сохранения происходит переход на следующую активность.

A screenshot of a code editor showing the implementation of the AuthButtonClick class. The code is in Java and implements the View.OnClickListener interface. It contains an onClick method that checks if the input text is empty and if it matches the email format. If it does, it saves the email to Shared Preferences and loads the next form. If not, it shows an alert message.

```
// класс который обрабатывает событие клика на кнопку
private class AuthButtonClick implements View.OnClickListener { 1 usage  ▲ Egor Vityazev
    @Override  ▲ Egor Vityazev
    public void onClick(View view) {
        // сохраняем информацию о почте
        String inputText = input.getText().toString();
        Log.i(TAG, msg: "User entered this text - " + inputText);
        if (!inputText.isEmpty() && checkEmail(inputText)) {
            shared.saveData( key: "email", inputText);
            loadSecondForm();
        }
        else{
            shared.showSimpleAlert( title: "Ошибка", message: "Формат электронной почты не верен!");
        }
    }
}
```

Рис 3.3.1.2 «Код класса AuthButtonClick»

Метод checkEmail проверяет, соответствует ли введенная строка формату электронной почты. Он проверяет наличие символа "@" и точки после него. Если проверка не пройдена, пользователю отображается предупреждающее сообщение об ошибке с использованием метода showSimpleAlert объекта Shared.

Объект Shared представляет собой вспомогательный класс для работы с локальным хранилищем данных приложения с использованием SharedPreferences. Он обеспечивает сохранение, получение и очистку данных, а также отображение простых диалоговых окон.

Основные методы класса Shared включают:

- saveData для сохранения данных по заданному ключу.
- clearData для очистки всех сохраненных данных.
- getData для получения данных по заданному ключу.
- showSimpleAlert для отображения диалогового окна с сообщением.



```

public class Shared { 8 usages  ⚡ Egor Vityazev
    private final Context context; 2 usages
    private final SharedPreferences sharedPreferences; 4 usages
    public final static String serverUrl = "http://192.168.1.65:8080/api/"; 2 usages

    public Shared(Context context) { 4 usages  ⚡ Egor Vityazev
        String APP_NAME = "DebtClearFlow";
        sharedPreferences = context.getSharedPreferences(APP_NAME, Context.MODE_PRIVATE);
        this.context = context;
    }

    public void saveData(String key, String data) { 1 usage  ⚡ Egor Vityazev
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString(key, data);
        editor.apply();
    }

    public void clearData(){ 1 usage  ⚡ Egor Vityazev
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.clear();
        editor.apply();
    }

    public String getData(String key) { return sharedPreferences.getString(key, ""); }

    public void showSimpleAlert(String title, String message){ 2 usages  ⚡ Egor Vityazev
        new AlertDialog.Builder(context).setTitle(title).setMessage(message)
            .setPositiveButton( text: "OK", new DialogInterface.OnClickListener() { ⚡ Egor Vityazev
                @Override ⚡ Egor Vityazev
                public void onClick(DialogInterface dialogInterface, int i) {}
            }).show();
    }
}

```

Рис 3.3.1.3 «Код класса Shared»

### 3.3.2 Реализация возможности просмотра информации о отработках, на которые записан студент

В рамках разработки мобильного приложения для платформы Android была реализована функциональность, позволяющая студентам просматривать информацию о всех отработках, на которые они записаны. Эта функциональность была реализована в активности SecondActivity.

Разметка activity\_second.xml представляет собой интерфейс, включающий в себя Toolbar с заголовком "Мои отработки", ListView для отображения списка отработок и кнопку для выхода из учетной записи. Все элементы размещены внутри ConstraintLayout, что позволяет гибко управлять их положением на экране.

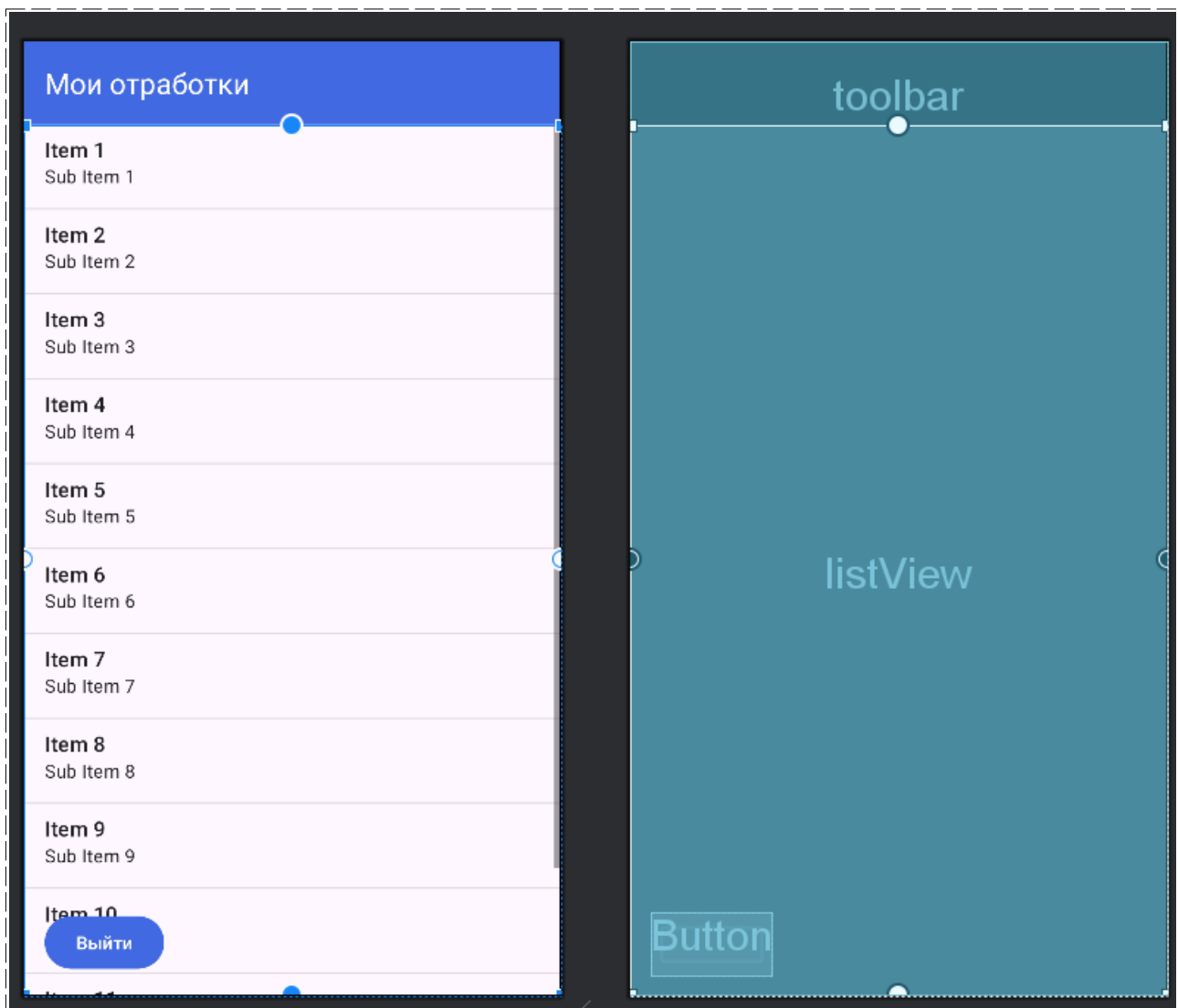


Рис 3.3.2.1 «Интерфейс формы со списком отработок студента»

Адаптер CustomListAdapter был создан для работы с ListView и предоставления данных для каждого элемента списка. Этот адаптер используется для отображения информации об отработках, предоставленной через массив объектов CustomListModel. Каждый элемент списка представляет собой отдельный макет, определенный в файле activity\_custom\_list\_view.xml. Он состоит из двух текстовых полей для отображения названия

и даты отработки, а также кнопки "Перейти", которая позволяет пользователю перейти к просмотру состояния очереди.

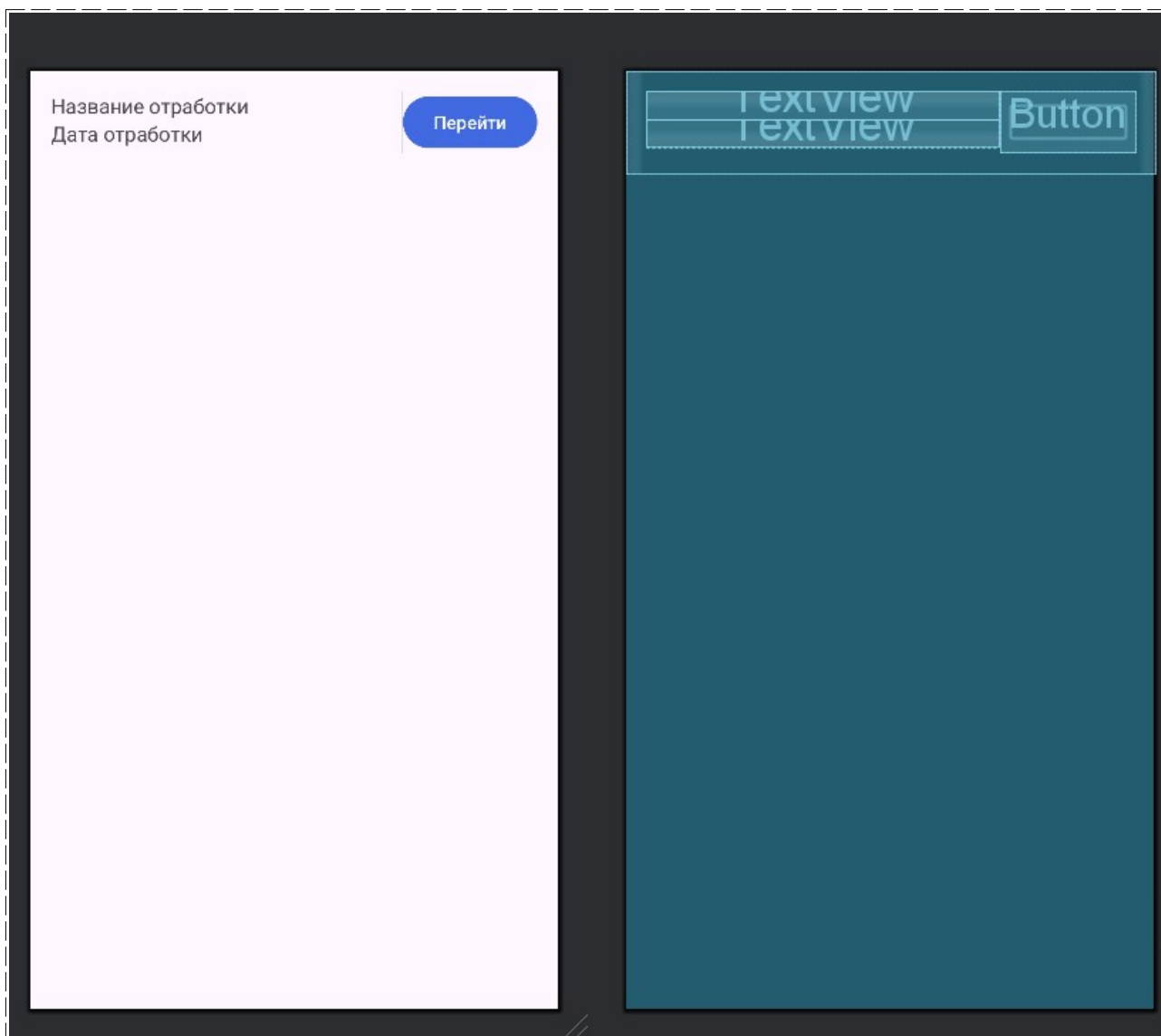


Рис 3.3.2.2 «Интерфейс элемента списка»

Класс CustomListAdapter содержит методы для управления данными и создания представлений элементов списка. Основные методы включают:

- getCount возвращает количество элементов в списке.
- getItem возвращает элемент данных по заданной позиции.
- getItemId возвращает идентификатор элемента данных по заданной позиции.
- getView создает и возвращает представление для элемента списка.

При создании представления в методе getView устанавливаются текстовые значения для названия и даты отработки, а также задается обработчик клика для кнопки "Перейти". При нажатии на кнопку создается Intent, который передает идентификатор и название отработки в активность RepaymentViewActivity, где отображается информация об состоянии очереди.

```

@Override  ▲ Egor Vityazev
public View getView(int i, View view, ViewGroup viewGroup) {
    view = inflater.inflate(R.layout.activity_custom_list_view, root: null);
    TextView receptionName = view.findViewById(R.id.reception_name);
    TextView receptionDate = view.findViewById(R.id.reception_date);
    Button button = view.findViewById(R.id.go_button);
    receptionName.setText(data.get(i).name);
    receptionDate.setText(data.get(i).time);

    button.setOnClickListener(new View.OnClickListener() {  ▲ Egor Vityazev
        @Override  ▲ Egor Vityazev
        public void onClick(View view) {
            //Shared shared = new Shared(context);
            //shared.showSimpleAlert("Test", data.get(i).id.toString());

            Intent intent = new Intent(context, RepaymentViewActivity.class);
            intent.putExtra(name: "id", data.get(i).id); //для передачи id
            intent.putExtra(name: "name", data.get(i).name);
            context.startActivity(intent);
        }
    });
    return view;
}

```

Рис 3.3.2.3 «Код метода getView»

Класс SecondActivity отвечает за управление событиями и взаимодействие с пользователем на экране списка отработок. Он содержит методы жизненного цикла активности, такие как onCreate, onStop и onPause, обеспечивая корректное поведение приложения при смене состояний.

При создании активности (onCreate) инициализируются интерфейсные элементы и устанавливаются обработчики событий. Кнопка "Выйти" позволяет пользователю выйти из текущей учетной записи, очищая сохраненные данные и возвращаясь к MainActivity.

Фоновый поток ReceptionsGetter отвечает за получение данных об отработках с сервера. Он использует библиотеку OkHttp для выполнения HTTP-запросов и библиотеку Jackson для обработки полученных данных в формате JSON. В фоновом потоке отправляется запрос на сервер для получения списка отработок, и полученные данные обновляют адаптер, отображающий их в ListView.

Основные задачи ReceptionsGetter включают:

- Отправку HTTP-запроса на сервер для получения списка отработок по указанному адресу и электронной почте.
- Десериализация ответа сервера в массив объектов DebtRepayment.

- Добавление новых отработок в адаптер и обновление интерфейса через хендлер Handler.

```
public class ReceptionsGetter implements Runnable{ 1 usage  ± Egor Vityazev
    private final OkHttpClient client = new OkHttpClient().newBuilder().build(); 1 usage

    private final Request request; 2 usages
    private final ObjectMapper mapper = new ObjectMapper(); 2 usages

    public ReceptionsGetter(String email, String serverUrl){ 1 usage  ± Egor Vityazev
        RequestBody body = new MultipartBody.Builder().setType(MultipartBody.FORM)
            .addFormDataPart( name: "email",email)
            .build();
        this.request = new Request.Builder()
            .url(serverUrl + "findReceptions")
            .method( method: "POST", body)
            .build();
        mapper.registerModule(new JavaTimeModule());
    }

    @Override  ± Egor Vityazev
    public void run() {
        while (isActive) {
            try {
                Response response = client.newCall(request).execute();
                assert response.body() != null;
                String responseString = response.body().string();
                Log.i(TAG, responseString);
                DebtRepayment[] receptions = mapper.readValue(responseString, DebtRepayment[].class);
                for (DebtRepayment reception : receptions){
                    if (adapter.findModelById(reception.getId()) == null) {
                        adapter.addModel(new CustomListModel(reception.getId(), reception.getName(), reception.getStarTime().toString()));
                    }
                }
                handler.post() -> adapter.notifyDataSetChanged();
            } catch (Exception e) {
                Log.e(TAG, msg: "I could not get receptions", e);
            }

            try {
                Thread.sleep( millis: 5000);
            } catch (InterruptedException ignored) {}
        }
    }
}
```

Рис 3.3.2.4 «Код класса ReceptionsGetter»

Такое разделение логики на адаптер, активность и фоновый поток позволяет легко управлять данными и отображением списка, а также улучшает читаемость и поддерживаемость кода.

### 3.3.3 Создание формы для просмотра информации о текущем положении студента в очереди в приложении

В рамках разработки мобильного приложения для платформы Android была создана форма для отображения информации о текущем положении студента в очереди на обработку. Данная форма реализована в активности RepaymentViewActivity и ее разметке activity\_repayment\_view.xml.

#### Разметка activity\_repayment\_view.xml

XML-файл activity\_repayment\_view.xml определяет интерфейс, который включает следующие элементы:

- Toolbar с заголовком "Название отработки", который отображает название выбранной отработки.
- Два TextView: один для отображения метки "Ваша позиция в очереди" и другой для отображения текущей позиции студента.
- Кнопка "Назад" для возврата на предыдущий экран.

Все элементы расположены внутри ConstraintLayout, что обеспечивает гибкую компоновку элементов интерфейса.

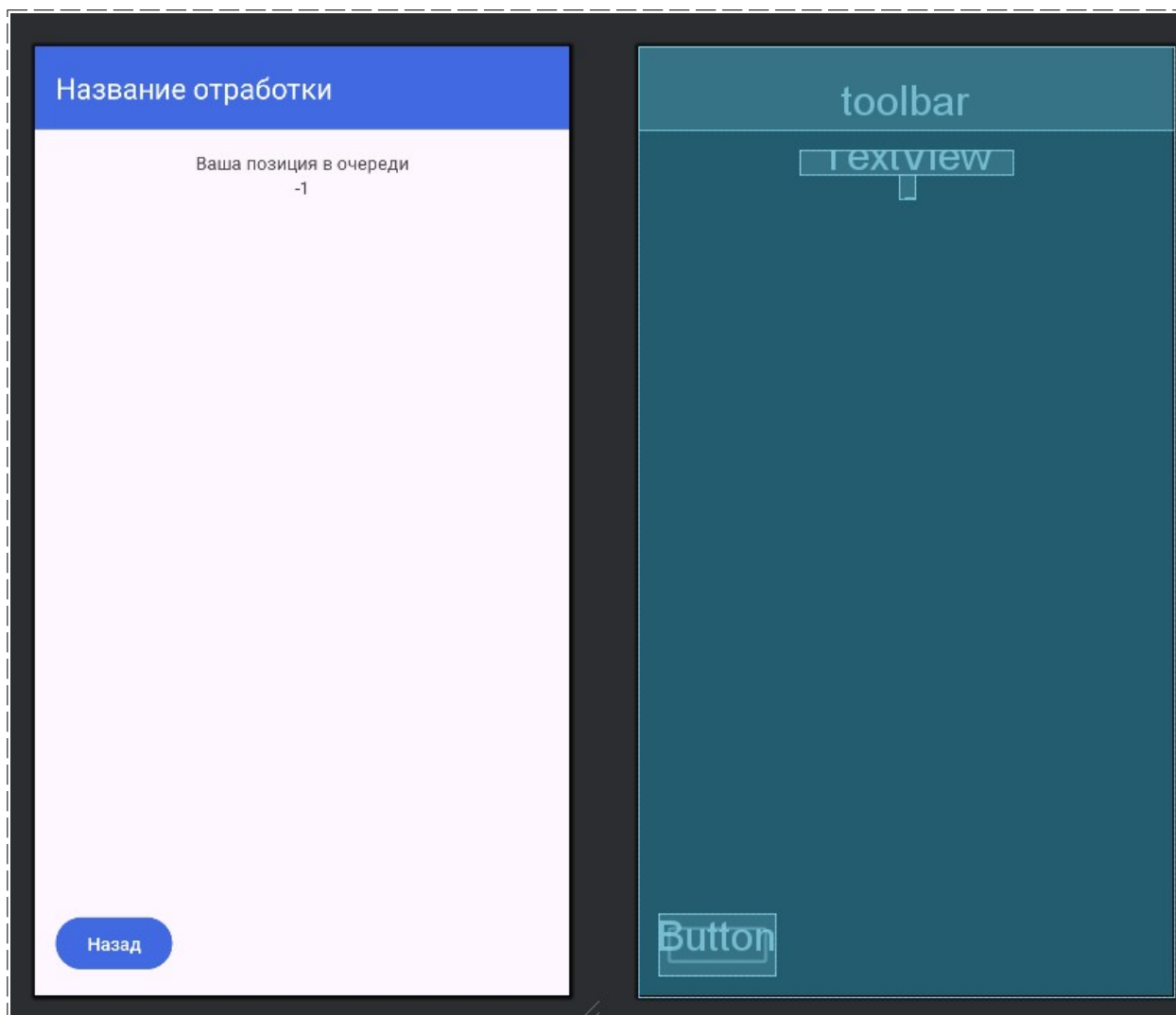


Рис 3.3.3.1 «Интерфейс просмотра текущего положения в очереди»

## **Класс RepaymentViewActivity**

Класс RepaymentViewActivity управляет событиями и логикой отображения информации о текущей позиции студента в очереди. Основные этапы и функции включают:

### **1. Инициализация Активности:**

- В методе onCreate происходит настройка интерфейса с использованием setContentView(R.layout.activity\_repayment\_view).
- Настраивается обработка системных отступов с помощью ViewCompat.setOnApplyWindowInsetsListener.

### **2. Получение Данных:**

- Из Intent извлекаются данные, переданные из предыдущей активности: идентификатор отработки (repaymentId) и название отработки (name).
- Также извлекается электронная почта пользователя из локального хранилища с помощью класса Shared.

### **3. Отображение Данных:**

- Устанавливаются значения для заголовка Toolbar и текстового поля, отображающего идентификатор отработки.
- Настраивается обработчик нажатия на кнопку "Назад", который запускает SecondActivity.

### **4. Фоновый Поток для Обновления Позиции в Очереди:**

- Создается и запускается фоновый поток PositionUpdater, который отправляет запросы на сервер для получения текущей позиции студента в очереди.
- Поток отправляет HTTP-запрос с использованием библиотеки OkHttp и получает ответ с сервера.
- Ответ сервера обрабатывается и обновляется текстовое поле queuePositionNumber с текущей позицией студента в очереди.

## **Класс PositionUpdater**

Этот внутренний класс реализует интерфейс Runnable и выполняет следующие задачи:

### **• Отправка Запроса:**

- Формируется HTTP-запрос с параметрами электронной почты пользователя и идентификатора отработки.
- Запрос отправляется на сервер по указанному URL.

### **• Обработка Ответа:**

- В цикле проверяется активность активности, чтобы продолжать обновление данных.
- Полученный ответ сервера обрабатывается и обновляет текстовое поле queuePositionNumber.
- Обновление происходит каждые 4 секунды для обеспечения актуальности данных.

### **• Остановка Обновления:**



- Обновление позиции прекращается при остановке или приостановке активности.

```
// класс который получает информацию о текущем положении в очереди студента
private class PositionUpdater implements Runnable { 1 usage  ± Egor Vityazev
    private final OkHttpClient client = new OkHttpClient().newBuilder().build(); 1 usage
    private final Request request; 2 usages

    public PositionUpdater(String email, int repaymentId, String serverUrl) { 1 usage  ± Egor Vityazev
        RequestBody body = new MultipartBody.Builder().setType(MultipartBody.FORM)
            .addFormDataPart( name: "email", email)
            .addFormDataPart( name: "repaymentId", String.valueOf(repaymentId))
            .build();

        request = new Request.Builder()
            .url(serverUrl + "findPosition")
            .method( method: "POST", body)
            .build();
    }

    @Override  ± Egor Vityazev
    public void run() {
        while (isActive) {
            try{
                Response response = client.newCall(request).execute();
                if (!response.isSuccessful() || response.body() == null) throw new Exception("Bad Request or body is null");
                String textResponse = response.body().string();
                handler.post() -> {
                    queuePositionNumber.setText(textResponse);
                };
                Thread.sleep( millis: 4000);
            } catch (Exception e) {
                Log.e(TAG, msg: "Can't get current position from server", e);
            }
        }
    }
}
```

Рис 3.3.3.2 «Код класса PositionUpdater»

## 4. Выводы о проделанной работе

В ходе практической работы была выполнена значительная часть задач, направленных на создание автоматизированной системы управления очередями на отработку учебных долгов. Основные выводы о проделанной работе можно сформулировать следующим образом:

### 1. Настройка серверной инфраструктуры:

- Успешно развернута серверная инфраструктура с использованием Docker и Docker-compose. Настроены контейнеры для MySQL и PHPMyAdmin, что обеспечило надежное управление базой данных и упростило администрирование системы.

### 2. Разработка серверной части:

- Создан проект на основе фреймворка Spring Boot, что обеспечило быстрый и структурированный подход к разработке серверной части.
- Реализована система авторизации и аутентификации пользователей с использованием Thymeleaf для создания удобных веб-интерфейсов.



- Введены фильтры для ограничения доступа неавторизованным пользователям, что повысило безопасность системы.
- Разработаны контроллеры и представления для управления очередями на отработки, включая функции добавления, удаления и просмотра отработок.
- Создана модель очереди студентов и система управления очередями, что позволило автоматизировать прием студентов и определение их позиций в очереди.
- Внедрена возможность импорта списков студентов из Google форм, что значительно упростило процесс добавления новых записей в систему.
- Разработано API для передачи информации между сервером и мобильным приложением, обеспечившее надежное взаимодействие между компонентами системы.

### 3. Разработка мобильного приложения (Android):

- Реализована возможность идентификации студентов по электронной почте с сохранением информации, что упростило процесс авторизации в мобильном приложении.
- Добавлена функция просмотра информации о отработках, на которые записан студент, что позволило пользователям получать актуальные данные о своих учебных задолженностях.
- Создана форма для просмотра текущего положения студента в очереди, что повысило прозрачность и удобство использования системы для студентов.

В результате проделанной работы была создана основа для автоматизированной системы управления очередями на отработки учебных долгов. Это позволит в дальнейшем доработать проект до полноценного решения, обеспечивающего более эффективное, удобное и прозрачное управление процессом отработки для всех участников образовательного процесса

## 5. Приложения

1. Полный исходный код проекта - <https://github.com/VityazevEgor/DebtClearFlow>