Geekbrains

**Разработка бекенд-сервиса для автоматизации отчета о результатах учебного процесса с использованием фреймворка SPRING (Spring Boot)**

Программа: Разработчик

Специализация: Программист

Александренко А. В.

Москва

2024

**СОДЕРЖАНИЕ**

Введение………………………………………………………………………………………………..3

Фреймворк SPRING (SpingBoot)……………………………………………………………..4

Описание задачи……………………………………………………………………………………6

Структура Программы……………………………………………………………………………8

Структура API и документация…………………………………………………………….15

Безопасность (SpringBootSecurity)……………………………………………………….16

Приложение 1………………………………………………………………………………………19

Приложение 2………………………………………………………………………………………19

**Введение**

Трудно себе представить, что в наш век повальной автоматизации и электронной информации существуют области, в которых используется бумажный документооборот. Тем не менее они есть. Одна из таких областей, это профессиональное образование. И если сам процесс обучения в настоящее время, где то в большей, где то в меньшей степени, идет в ногу со временем, то этап отчета преподавателя (мастера, инструктора) о проведенных занятиях, зачастую требует оформления именно в бумажном виде.

Темой данной дипломной работы является автоматизация как раз такого этапа на конкретном примере учебной авиационной эскадрильи университета гражданской авиации.

Говорить о высокой коммерческой ценности данного продукта не приходится, но он имеет социальную значимость в узкопрофессиональном кругу.

**Фреймворк Spring** (**Spring Boot)**

При разработке программы использовался фреймворк Spring и его расширение Spring Boot.

История возникновения:

Язык Java появился ещё в 1995 году. Некоторое время на нём писали обычные офлайн-программы для компьютера, но постепенно он стал популярен у веб-разработчиков благодаря надёжности и стабильности. Чтобы добавить в него модульность, были созданы специальные классы JavaBeans. Они облегчили разработку компонентов, но не позволяли делать другие важные вещи: сохранять данные, управлять безопасностью или делать приложения многопользовательскими. Это было критично для тех, кто разрабатывал большие корпоративные приложения вроде бухгалтерского ПО. Чтобы решить проблему, классы JavaBeans расширили с помощью технологии Enterprise JavaBeans. Она поддерживала всё, что нужно, но слишком сильно усложняла код. Поэтому разработчики продолжали искать более простое решение. В 2003 оно нашлось — разработчик Род Джонсон представил сообществу тестовую версию нового фреймворка, Spring. С тех пор он стал стандартом разработки и обязателен к изучению практически каждым Java-программистом. Фреймворк Java Spring предоставляет разработчикам инструменты для создания сложных систем, например многопользовательских корпоративных веб-приложений со множеством функций для бизнеса. Он позволяет быстро создавать приложения, которые умеют работать с базами данных и облаками, состоят из разных модулей, обмениваются данными с пользователями через интернет по защищённым каналам. Теоретически всё это можно реализовать в Java и вручную, но Spring даёт разработчикам уже готовые инструменты, которые позволяют писать код гораздо быстрее и концентрироваться не на формальностях, а на уникальных функциях программы.

Spring Boot является расширением технологии Spring, разработанным компанией Pivotal Software. Благодаря быстродействию и простоте работы он стал популярным решением для создания развертываний в виде архива веб-приложений (WAR) и автономных Java-приложений.

Spring Boot выделяется среди других фреймворков, поскольку он предоставляет разработчикам программного обеспечения гибкую настройку, надежную пакетную обработку, эффективный рабочий процесс и большое количество инструментов, помогая разрабатывать надежные и масштабируемые приложения на базе Spring.

Данный проект Spring Boot создан с помощью Spring Initializr и среды разработки IntelliJ IDEA. В настройках проекта выбран «Web» в качестве зависимости.

**Описание задачи**

В рамках прохождения обучения в авиационном ВУЗе, студенты летного факультета должны пройти курс практического обучения на учебном самолете. Обучение осуществляют пилоты-инструкторы данного учебного заведения.

Отчет пилота-инструктора должен обязательно проводиться по окончании летной смены (не важно днем или ночью).

В настоящее время это связано с необходимостью: прибыть с аэродрома в штаб авиационной эскадрильи и заполнить ряд журналов, а именно, по инженерному направлению (наработка двигателей, расход топлива и т.д.) и по учебному (налет по студентам).

Далее, (на следующее утро) специально обученный человек:) (девушка – document logist) переносит данные за сутки в электронные таблицы и отправляет их уже в учебный и инженерный отделы университета.

Очевидно, что наличие электронного сервиса, на который пилот мог бы послать данные, скажем с мобильного устройства, плодотворно отразилось бы как на ресурсах рабочего времени инструктора, так и на скорости доступа к актуальной информации любых заинтересованных лиц.

В связи с этим от сервера, кроме ответа на стандартные REST запросы, требуется агрегация следующих данных:

- Налет (в минутах) студентов за предыдущий день (указанную дату)

- Налет (в минутах) конкретного студента за предыдущий день (указанную дату)

- Налет (в минутах) пилотов – инструкторов за предыдущий день (указанную дату)

- Налет (в минутах) конкретного пилота за предыдущий день (указанную дату)

- Налет (в минутах) конкретного пилота за предыдущую неделю; и за предыдущий месяц.

Кроме этого, необходимо возможность разграничения прав доступа к просмотру и изменению информации в базе данных, относящейся к разным направлениям.

**Структура Программы**

Данная Программа представляет собой серверную часть Приложеиия, которая взаимодействует с базой данных (БД).

Основным источником актуальной информации является Отчет (report) пилота-инструктора о вылете. Данный отчет можно представить в виде списка данных:

Дата вылета

Бортовой номер самолета

ФИО студента

ФИО инструктора

ФИО второго пилота или проверяющего (если он есть)

Количество полетов\*

Время затраченное на вылет

Наработка двигателя (по счетчику двигателя)

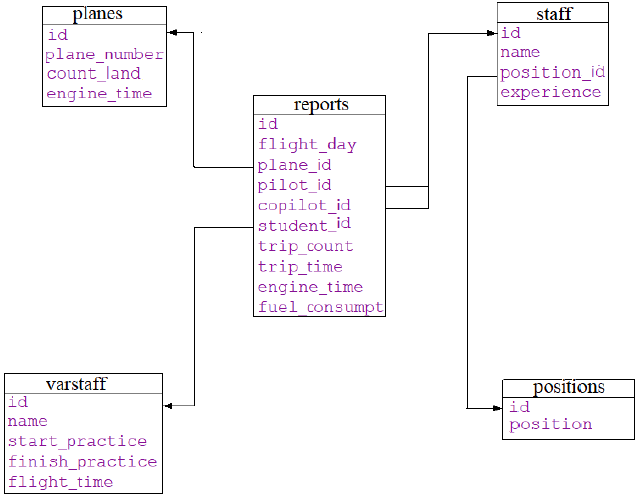
Количество израсходованного топлива на вылет

\**Часто в учебных целях, за один вылет выполняется несколько полетов методом «конвейер» (посадка – короткий пробег – последующий взлет)*

Кроме этого, должна быть возможность поддержания актуальной информации по таблицам (например, со стороны document logist возможность добавить новую группу студентов, пришедшую на практику - variable staff или изменения по составу пилотов-инструкторов - staff).

Задача сервера, на основе этой информации выполнять запросы к БД и предоставлять данные потребителю.

В рамках дипломного проекта подключена БД h2. На которой сформирована следующая структура:



Таблицы:

**planes**

|  |  |  |
| --- | --- | --- |
| №пп | Наименование поля | Пояснение |
| 1 | id | Идентификатор |
| 2 | plane\_number | Бортовой номер самолета |
| 3 | count\_land | Общее количество посадок |
| 4 | engine\_time | Наработка по счетчику двигателя |

**staff**

|  |  |  |
| --- | --- | --- |
| №пп | Наименование поля | Пояснение |
| 1 | id | Идентификатор |
| 2 | name | Имя пилота |
| 3 | position\_id | Должность |
| 4 | experience | Дата начала работы в качестве инструктора |

**positions**

|  |  |  |
| --- | --- | --- |
| №пп | Наименование поля | Пояснение |
| 1 | id | Идентификатор |
| 2 | position | Наименование должности пилота |

**varstaff**

|  |  |  |
| --- | --- | --- |
| №пп | Наименование поля | Пояснение |
| 1 | id | Идентификатор |
| 2 | name | Имя студента |
| 3 | start\_prctice | Дата начала практики |
| 4 | finish\_practice | Дата окончания практики |
| 5 | flight\_time | Общий налет (в минутах) |

**reports**

|  |  |  |
| --- | --- | --- |
| №пп | Наименование поля | Пояснение |
| 1 | id | Идентификатор |
| 2 | flight\_day | Дата вылета |
| 3 | plane\_id | Идентификатор самолета |
| 4 | pilot\_id | Идентификатор пилота |
| 5 | copilot\_id | Идентификатор второго пилота (или проверяющего) |
| 6 | student\_id | Идентификатор студента |
| 7 | trip\_count | Количество посадок за вылет |
| 8 | trip\_time | Налет за вылет (в мин.) |
| 9 | engine\_time | Наработка двигателей за вылет (в десятич. формате) |
| 10 | fuel\_consumpt | Расход топлива (в кг) |

В SPRING реализована технология JPA (Jakarta Persistence API) которая позволяет проецировать таблицы базы данных на программные сущности. Что бы использовать возможности этой технологии необходимо:

- в файле POM.xml подключить зависимость spring-boot-starter-data-jpa. По умолчанию в Spring эта технология реализуется через фреймворк Hibernate;

- создать классы с полями, которые соответствуют столбцам таблиц базы данных;

- над классом соответствующим таблице БД поставить аннотацию @Entity.

В рамках данной работы были созданы 5 классов – сущностей:

Planes, Positions, Reports, Staff, VarStaff.

**Planes**

@Entity

@Table(name = "planes")

public class Planes {

@Id

@GeneratedValue(strategy=GenerationType.*IDENTITY*)

@EqualsAndHashCode.Include

private Long id;

private String planeNumber;

private Long countLand;

private Double engineTime;

}

**Positions:**

@Entity

@Table(name = "positions")

public class Positions {

@Id

@GeneratedValue(strategy=GenerationType.*IDENTITY*)

@EqualsAndHashCode.Include

private Long id;

private String position;

}

**Reports:**

@Entity

@Table(name = "reports")

public class Reports {

@Id

@GeneratedValue(strategy=GenerationType.*IDENTITY*)

@EqualsAndHashCode.Include

private Long id;

@Column(name="flight\_day")

private LocalDate flightDay;

@Column(name="plane\_id")

private Long planeId;

@Column(name="pilot\_id")

private Long pilotId;

@Column(name = "copilot\_id")

private Long coPilotId;

@Column(name="student\_id")

private Long studentId;

private Long tripCount;

private Long tripTime;

private Double engineTime;

private Long fuelConsumpt;

}

**Staff:**

@Entity

@Table(name = "staff")

public class Staff {

@Id

@GeneratedValue(strategy=GenerationType.*IDENTITY*)

@EqualsAndHashCode.Include

private Long id;

private String name;

private Long positionId;

private LocalDate experience;

}

**VarStaff:**

@Entity

@Table(name="varstaff")

public class VarStaff {

@Id

@GeneratedValue(strategy=GenerationType.*IDENTITY*)

@EqualsAndHashCode.Include

@Column

private Long id;

private String name;

private LocalDate startPractice;

private LocalDate finishPractice;

private Long flightTime;

}

Для каждой сущности создан слой контроллеров (Например, PositionController), сервисный слой (Например, StaffService) и слой репозитория (Например, PlaneRpository). Причем слой репозитория реализован с помощью интерфейсов, которые в свою очередь являются наследниками интерфейса interface JpaRepository<T, ID>.

Над объявлениями классов-контроллеров установлена аннотация @RestController которая объединяет в себе аннотации @Controller и @ResponseBody. Это означает, что она не только помечает класс как Spring MVC Controller, но и автоматически преобразует возвращаемые контроллером данные в формат JSON или XML.

**Структура API и документация**

Базовые эндпойнты в классах контроллерах объявлены с помощью аннотации @RequestMapping(), а внутренние с помощью соответствующих аннотаций для REST методов. Например: @GetMapping, @PostMapping и т.д.

В разрабатываемом сервере, кроме основных REST запросов, реализованы сложные запросы, которые могут стать уже готовым отчетом для клиентского приложения. Например, эндпойнт **/pilots//byweek/7** вернет для пилота с идентификатором 7 список летных смен с его налетом. Подобная информация необходима руководящему составу для контроля санитарной нормы рабочего времени пилота. Полный список ресурсов (эндпойнтов) представлен в Приложении 1.

Spring Boot позволяет разработчикам описывать структуру своих API и генерировать интерактивную документацию, клиентские библиотеки и серверные модули для реализации API.

Что бы использовать эту возможность в файле POM.xml нужно подключить зависимость **springdoc-openapi-starter-webmvc-ui.**

Теперь в классах – контроллерах мы имеем возможность использовать аннотации для документирования. Например, такие как @Tag, @Operation

@Parameter.

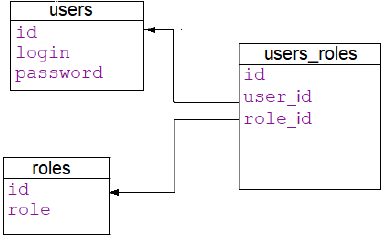
Что бы просмотреть документацию API необходимо в браузере ввести адресную строку: **http://адрес\_сервера:порт/swagger-ui.html.**

**Безопасность (SpringBootSecurity)**

В Spring Boot есть возможность автоматического подключения механизмов аутентификации и авторизации пользователей. Для этого в файле pom.xml нужно добавить зависимость **spring-boot-starter-security**.

Для реализации данного механизма необходимо создать класс который имплементирует интерфейс UserDetailsService и переопределить в нем абстрактный метод *loadUserByUsername(String username).* В данном проекте это класс MyCustomUserDetailsService. Кроме этого нужно создать класс конфигурации SecurityConfiguration в котором например, можно определить фильтры безопасности и способ кодирования пароля (используется BCryptPasswordEncoder()).

Кроме этого, в БД должны быть созданы таблицы для пользователей (users) и ролей (roles), а также связующая таблица для реализации связи «многие ко многим» с целью назначения прав доступа.



А на сервере необходимо создать классы - сущности соответствующие этим таблицам.

**Role:**

@Entity

@Table(name = "roles")

public class Role {

@Id

@GeneratedValue(strategy = GenerationType.*SEQUENCE*)

private Long id;

@Column(name = "role")

private String role;

}

**User:**

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.*SEQUENCE*)

private Long id;

@Column(name = "login")

private String login;

@Column(name = "password")

private String password;

}

**UserRole:**

@Entity

@Table(name = "users\_roles")

public class UserRole {

@Id

@GeneratedValue(strategy = GenerationType.*SEQUENCE*)

private Long id;

@Column(name = "user\_id")

private Long userId;

@Column(name = "role\_id")

private Long roleId;

}

По аналогии с основными таблицами для этих сущностей созданы серверный уровень, уровень репозитория и контроллера на которых реализованы REST-методы.

Для назначения прав доступа в рамках конкретной технической задачи создано несколько ролей:

**admin** – роль, которая позволяет выполнять все действия, связанные с таблицами users, roles users\_roles, но остальные ресурсы для этой роли закрыты;

**doclogist** – основная задача следить за актуальностью таблиц: staff (основной состав), varstaff (переменный состав - студенты), positions (таблица должностей);

**commandor** – для осуществления контролирующих функций необходим доступ к ресурсам о налете пилотов – инструкторов за неделю, месяц, и просмотру информации по учебным самолетам;

**pilot** – права на все действия с таблицей reports;

**engineer** – права на все действия с таблицей planes;

Соответствие ролей и ресурсов приведено в Приложении 1.

В рамках тестирования сервера в базе данных создано несколько пользователей: **admin** (пароль: admin), **us1pilot** (пароль: us1pilot), **us2pilot** (пароль: us2pilot), **us3pilot** (пароль: us3pilot), **us4pilot** (пароль: us4pilot), **us5pilot** (пароль: us5pilot), **engineer** (пароль: engineer), **logist** (пароль: logist)

Соответствие пользователей и ролей в Приложении 2.

Приложение 1

Таблица распределения прав пользовательских ролей

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ресурс/Роль | admin | doclogist | commandor | pilot | engineer |
| “/admin/\*\*” | да | нет | нет | нет | нет |
| “/students/\*\*” | нет | да | нет | нет | нет |
| “/pilots/\*\*” | нет | нет | да | нет | нет |
| “/positions/\*\*” | нет | нет | да | нет | нет |
| “/reports/\*\*” | нет | нет | нет | да | нет |
| “/planes/\*\*” | нет | нет | нет | нет | да |

Приложение 2

Таблица распределения пользователей по ролям

|  |  |  |
| --- | --- | --- |
| №пп | Логин пользователя | Роли |
| 1 | admin | admin |
| 2 | engineer | engineer |
| 3 | logist | doclogist |
| 4 | us1pilot | admin, pilot, commandor |
| 5 | us2pilot | pilot |
| 6 | us3pilot | pilot |
| 7 | us4pilot | pilot, logist |
| 8 | us5pilot | pilot, admin |