**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**"РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА" (МИИТ)**

**Институт управления и цифровых технологий**

**Кафедра «Цифровые технологии управления транспортными процессами»**

**КУРСОВАЯ РАБОТА**

**По дисциплине:**

**«Ознакомительная практика»**

**На тему:**

**«Почтовые отправления»**

Выполнил:

Студент группы УИС-211

Чаругин А. М.

Принял:

Преподаватель кафедры «ЦТУТП»

Толкунов Д. В.

Москва

2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc147876368)

[**Требования** 5](#_Toc147876369)

[**Подготовка** 5](#_Toc147876370)

[**Реализация** 6](#_Toc147876371)

[**Структура проекта** 6](#_Toc147876372)

[**Результат работы приложения** 18](#_Toc147876373)

[ВЫВОД 20](#_Toc147876374)

[ПРИЛОЖЕНИЕ 21](#_Toc147876375)

# ВВЕДЕНИЕ

Целью данной курсовой работы является разработка информационной системы для управления и мониторинга почтовыми отправлениями с использованием Java Spring и формата хранения данных YAML. Мы будем исследовать способы хранения, обработки и визуализации данных о почтовых отправлениях, а также создадим пример приложения, демонстрирующий эту концепцию.

Отправление почтовых грузов и документов — это важная и распространенная задача в современном мире. Каждое отправление характеризуется рядом признаков, которые необходимо учитывать для эффективного управления и мониторинга. Эти признаки включают в себя:

Уникальный номер отслеживания (trackingNumber): Каждое отправление имеет уникальный идентификатор, который позволяет однозначно идентифицировать и отслеживать его перемещение.

Адрес отправителя (senderAddress): Информация о месте, откуда идет отправление.

Адрес назначения (destinationAddress): Информация о месте, куда должно быть доставлено отправление.

Статус (status): Текущее состояние отправления, которое может изменяться по времени и этапам его передвижения.

Дата поступления отправления в службу (dateReceived): Дата, когда отправление было передано в службу доставки.

Дата присвоения последнего статуса(lastStatusDate): Дата, когда был установлен последний статус отправления.

Статусы отправления могут меняться в соответствии с определенной схемой. Основными этапами являются "Принято", "В пути", "Поступило в пункт назначения" и "Доставлено". Однако также предусмотрен статус "Ошибка", который может возникнуть на любом этапе обработки. После статуса "Ошибка" отправление переводится в специальное состояние, и дальнейшие изменения статуса становятся невозможными  
 В этой курсовой работе, должно использоваться Java Spring и формат хранения данных YAML, важно понимать, как эти два элемента сочетаются в контексте разработки приложений.

Java Spring — это мощный фреймворк для разработки Java-приложений, который предоставляет ряд инструментов и возможностей для упрощения создания приложений с использованием Java. В контексте курсовой работы, Java Spring используется для разработки бэкенда, приложения, управления бизнес-логикой и взаимодействия с YAML. Ниже перечислены некоторые ключевые аспекты Java Spring:

Внедрение зависимостей (DI): Spring позволяет управлять жизненным циклом компонентов приложения и внедрять их зависимости, что делает приложения более гибкими и облегчает их конфигурирование.

Spring Boot — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

Spring MVC: Spring MVC — это модуль, который помогает создавать веб-приложения с использованием шаблона MVC (Model-View-Controller).

MVC позволяет разделять логику приложения на более мелкие и управляемые части, что облегчает сопровождение, расширение и повторное использование кода. Этот паттерн также способствует разделению обязанностей между разработчиками фронтенда и бекенда, что делает разработку более эффективной и организованной. MVC часто используется во многих веб-фреймворках и технологиях для создания веб-приложений.

YAML (YAML Ain't Markup Language) - это удобный и читаемый формат для хранения данных, который часто используется для конфигурации приложений. Он использоваться в моем приложении для хранения данных о почтовых отправлениях и конфигурации вашего приложения. Но сам по себе YAML может многое. Примеры данных, которые можно хранить в YAML:

* Настройки приложения, такие как порт, на котором запущен веб-сервер.
* Сведения о почтовых отправлениях, такие как адрес отправителя, адрес назначения, статус, даты и другие характеристики.

## **Требования**

Формата хранения данных на диске – YAML.

Предметная область: почтовые отправления.

Каждое отправление характеризуется следующими признаками: уникальный номер отслеживания, адрес отправителя, адрес назначения, статус, дата поступления отправления в службу, дата присвоения последнего статуса.

Отправление может менять свой статус по схеме («Принято» – «В пути» – «Поступило в пункт назначения» – «Доставлено», на любом этапе возможен статус «Ошибка», после которого перевод в другие статусы невозможен).

## **Подготовка**

1. Установить IntelliJ IDEA 2023.1.3
2. Открыть проект
3. Для запуска необходимо ввести команды:

**java –version** - используется для вывода информации о версии Java, установленной на вашем компьютере. Вы можете выполнить эту команду в командной строке или терминале.

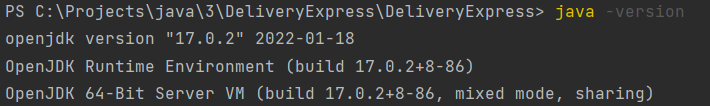


Рис 1 – java которая используется в проекте

**mvn install** - Компиляция исходного кода: Maven анализирует исходный код вашего проекта (обычно на Java) и компилирует его в байт-код Java. Это включает в себя компиляцию всех классов и ресурсов, необходимых для вашего проекта.

**mvn clean install -U** - используется для очистки проекта от предыдущих сборок, сборки проекта и его установки в локальный репозиторий Maven с обновлением снапшотов зависимостей из удаленных репозиториев.

**npm run** - выполнения пользовательских скриптов, определенных в файле package.json

**java -jar target/DeliveryExpress-0.0.1-SNAPSHOT.jar** – запуск java spring проекта

## **Реализация**

## **Структура проекта**

Рис 2 – структура проекта

**MainController.java:**

Этот контроллер обеспечивает обработку различных HTTP-запросов и управление перенаправлениями между страницами в зависимости от действий пользователя.

showHomePage : Этот метод обрабатывает GET-запросы на корневой URL ("/"). Он возвращает "index.html", что, вероятно, является страницей главной страницы вашего веб-приложения.

search(@RequestParam String trackingNumber, Model model): Этот метод обрабатывает GET-запросы на URL "/search" и ожидает параметр `trackingNumber` в запросе. Он вызывает сервис `shipmentService` для поиска почтовой отправки по номеру отслеживания `trackingNumber`. Если отправка найдена, она добавляется в модель `model`, и метод выполняет перенаправление на страницу "result.html". Если отправка не найдена, происходит перенаправление на главную страницу "index.html".

showResultPage(@RequestParam String trackingNumber, Model model): Этот метод обрабатывает GET-запросы на URL "/result" и также ожидает параметр `trackingNumber` в запросе. Он вызывает сервис `shipmentService` для поиска почтовой отправки по номеру отслеживания `trackingNumber`. Если отправка найдена, она добавляется в модель `model`, и метод возвращает страницу "result.html". Если отправка не найдена, происходит перенаправление на главную страницу.

showCreatePage(): Этот метод обрабатывает GET-запросы на URL "/create" и возвращает страницу "create.html". Вероятно, это страница для создания новых почтовых отправлений.

createShipment(@ModelAttribute ShipmentForm shipmentForm): Этот метод обрабатывает POST-запросы на URL "/create" и ожидает объект `ShipmentForm` в теле запроса. Он вызывает метод `createShipmentFromForm` у сервиса `shipmentService` для создания новой почтовой отправки на основе данных из формы. После успешного создания отправки метод выполняет перенаправление на главную страницу "index.html".

**ShipmentController.java:**

Этот контроллер обеспечивает обработку различных HTTP-запросов и переход между страницами в зависимости от действий пользователя.

showHomePage(Model model): Этот метод обрабатывает GET-запросы на корневой URL ("/"). Он на главную страницу веб-приложения.

search(@RequestParam String trackingNumber, Model model): Этот метод обрабатывает GET-запросы на URL "/search" и ожидает параметр trackingNumber в запросе. Он вызывает сервис shipmentService для поиска почтовой отправки по номеру отслеживания trackingNumber. Если отправка найдена, она добавляется в модель model, и метод выполняет перенаправление на страницу "result.html". Если отправка не найдена, происходит перенаправление на главную страницу "index.html".

showResultPage(@RequestParam String trackingNumber, Model model): Этот метод обрабатывает GET-запросы на URL "/result" и также ожидает параметр trackingNumber в запросе. Он вызывает сервис shipmentService для поиска почтовой отправки по номеру отслеживания trackingNumber. Если отправка найдена, она добавляется в модель model, и метод возвращает страницу "result.html". Если отправка не найдена, происходит перенаправление на главную страницу "index.html".

showCreatePage(): Этот метод обрабатывает GET-запросы на URL "/create" и возвращает страницу "create.html". Вероятно, это страница для создания новых почтовых отправлений.

createShipment(@ModelAttribute ShipmentForm shipmentForm): Этот метод обрабатывает POST-запросы на URL "/create" и ожидает объект ShipmentForm в теле запроса. Он вызывает метод createShipmentFromForm у сервиса shipmentService для создания новой почтовой отправки на основе данных из формы. После успешного создания отправки метод выполняет перенаправление на главную страницу "index.html".

**ShipmentForm.java**

Этот класс предназначен для облегчения передачи данных о почтовых отправлениях между клиентской стороной (например, веб-страницей или мобильным приложением) и серверной стороной вашего приложения. Когда пользователь вводит информацию о почтовой отправке (например, при создании новой отправки), данные могут быть собраны в объект ShipmentForm, который затем передается в контроллер или сервис для обработки и сохранения в базе данных или других целях.

Поля класса ShipmentForm:

trackingNumber: Строковое поле, предназначенное для хранения уникального номера отслеживания почтовой отправки.

senderAddress: Строковое поле, предназначенное для хранения адреса отправителя почтовой отправки.

destinationAddress: Строковое поле, предназначенное для хранения адреса назначения почтовой отправки.

status: Поле типа ShipmentStatus, которое предназначено для хранения текущего статуса почтовой отправки.

Геттеры и сеттеры: Класс ShipmentForm содержит геттеры и сеттеры для каждого поля, что позволяет получать и устанавливать значения полей этого объекта. Геттеры используются для получения значений полей, а сеттеры - для установки значений полей.

**Shipment.java:**

Этот класс Shipment представляет собой модель данных для отслеживания почтовых отправлений в вашем приложении и обеспечивает маппинг данных между объектами Java и записями в базе данных.

@Entity: Эта аннотация указывает, что класс Shipment является сущностью, которая будет отображена в базе данных. Это позволяет использовать объекты этого класса для взаимодействия с базой данных.

@Id и @GeneratedValue(strategy = GenerationType.IDENTITY): Эти аннотации указывают, что поле id является первичным ключом и будет автоматически генерироваться базой данных при создании новой записи.

Поля класса:

trackingNumber: Строковое поле, предназначенное для хранения уникального номера отслеживания почтовой отправки.

senderAddress: Строковое поле, предназначенное для хранения адреса отправителя почтовой отправки.

destinationAddress: Строковое поле, предназначенное для хранения адреса назначения почтовой отправки.

status: Поле типа ShipmentStatus, которое предназначено для хранения текущего статуса почтовой отправки.

dateReceived: Поле типа Date, предназначенное для хранения даты поступления почтовой отправки в службу.

lastStatusDate: Поле типа Date, предназначенное для хранения даты присвоения последнего статуса почтовой отправке.

Геттеры и сеттеры: Класс Shipment содержит геттеры и сеттеры для каждого поля, что позволяет получать и устанавливать значения полей этого объекта.

@Enumerated(EnumType.STRING): Эта аннотация указывает, что поле status использует перечисление ShipmentStatus и хранит его в виде строки.

@Temporal(TemporalType.TIMESTAMP): Эти аннотации указывают, что поля dateReceived и lastStatusDate используют тип Date и хранят дату и время.

@Column: Эти аннотации указывают дополнительные настройки для столбцов в базе данных, такие как nullable (может ли поле быть пустым) и unique (является ли поле уникальным).

**ShipmentsData.java:**

Используется для управления списком почтовых отправлений в приложении, добавлять, получать и удалять отправки из списка.

Поле shipments: Это поле типа List<Shipment>, которое представляет собой список почтовых отправлений. Исходно оно инициализируется пустым списком.

Конструктор ShipmentsData(List<Shipment> shipments): Этот конструктор принимает список почтовых отправлений в качестве параметра, но внутри конструктора список shipments инициализируется пустым списком. Таким образом, входной список не используется для инициализации.

addShipment(Shipment shipment): Этот метод позволяет добавить новую почтовую отправку в список shipments.

getShipment(int index): Этот метод позволяет получить почтовую отправку из списка по ее индексу. Если индекс находится в допустимом диапазоне, метод возвращает соответствующую отправку, в противном случае возвращает null.

removeShipment(int index): Этот метод позволяет удалить почтовую отправку из списка по индексу, если индекс находится в допустимом диапазоне.

**ShipmentStatus.java:**

Определяет возможные статусы для почтовых отправлений в приложении:  
ACCEPTED, IN\_TRANSIT, FINAL\_DESTINATION, SUCCESSFULLY\_DELIVERY, ERROR\_DELIVERY

**ShipmentRepository.java:**

Этот интерфейс существует для выполнения операций с данными о почтовых отправлениях, таких как поиск, сохранение и получение информации о них.

findByTrackingNumber(String trackingNumber): Метод, который позволяет найти почтовую отправку по ее уникальному номеру отслеживания.

save(Shipment shipment): Метод, который позволяет сохранить информацию о почтовой отправке. Он может использоваться для создания новых отправлений или обновления существующих.

findAll(): Метод, который возвращает список всех доступных почтовых отправлений.

findById(Long id): Метод, который позволяет найти почтовую отправку по ее уникальному идентификатору.

**YamlShipmentRepository.java**

Это реализация интерфейса ShipmentRepository для работы с данными о почтовых отправлениях, используя формат YAML для хранения и чтения данных.

findByTrackingNumber(String trackingNumber): Этот метод выполняет поиск почтовой отправки по ее уникальному номеру отслеживания trackingNumber. Для этого он загружает список всех отправлений из файла YAML, а затем ищет отправку с указанным номером.

save(Shipment shipment): Этот метод сохраняет новую почтовую отправку, добавляя ее к списку всех отправлений и сохраняя обновленный список обратно в файл YAML.

findAll(): Этот метод загружает список всех почтовых отправлений из файла YAML и возвращает его. Если файл YAML не может быть прочитан, он возвращает пустой список.

findById(Long id): Этот метод выполняет поиск почтовой отправки по ее уникальному идентификатору id среди всех отправлений и возвращает найденную отправку в виде Optional.

saveAll(List<Shipment> shipments): Этот метод сохраняет список почтовых отправлений обратно в файл YAML после внесения изменений.

loadShipmentsFromYamlFile(): Этот метод загружает список отправлений из файла YAML с использованием библиотеки SnakeYAML.

**ShipmentService.java:**

Определяет методы для обработки и управления почтовыми отправлениями в вашем приложении.

createShipment(Shipment shipment): Этот метод предназначен для создания новой почтовой отправки. Он принимает объект Shipment в качестве параметра и возвращает созданный объект Shipment.

updateShipmentStatus(Long shipmentId, ShipmentStatus newStatus): Этот метод обновляет статус существующей почтовой отправки. Он принимает уникальный идентификатор отправки shipmentId и новый статус newStatus. Обновленная почтовая отправка возвращается как результат.

getAllShipments(): Этот метод возвращает список всех почтовых отправлений, хранящихся в вашем приложении.

getShipmentById(Long shipmentId): Этот метод позволяет получить почтовую отправку по ее уникальному идентификатору shipmentId. Возвращается найденная почтовая отправка.

findShipmentByTrackingNumber(String trackingNumber): Этот метод выполняет поиск почтовой отправки по ее уникальному номеру отслеживания trackingNumber и возвращает найденную отправку.

createShipmentFromForm(ShipmentForm shipmentForm): Этот метод используется для создания почтовой отправки на основе данных, полученных из формы ShipmentForm. Он принимает объект ShipmentForm, извлекает из него необходимые данные и создает новую почтовую отправку.

**ShipmentServiceImpl.java:**

Этот файл содержит класс ShipmentServiceImpl, который предоставляет реализацию методов, определенных в ShipmentService.

Роль: ShipmentServiceImpl содержит конкретную бизнес-логику для выполнения операций с отправлениями, используя ShipmentRepository для доступа к данным.

**YamlShipmentService.java:**

Этот файл содержит класс YamlShipmentService, который предоставляет реализацию ShipmentService для работы с данными, предоставленными в формате YAML.

Роль: YamlShipmentService обеспечивает чтение и обработку данных об отправлениях из YAML-файла, позволяя вашему приложению использовать такие данные для операций с отправлениями.

**Конфигурационный pom.xml**

XML-файл pom.xml описывает конфигурацию проекта и его зависимостей для сборки с использованием Apache Maven. Давайте рассмотрим его содержимое от вашего лица:

<project>: Элемент, определяющий корневой элемент проекта. Внутри этого элемента, вы определяете настройки проекта и его зависимости.

<modelVersion>: Указывает версию модели POM. В данном случае, используется версия 4.0.0.

<parent>: Определяет родительский проект, который наследует настройки сборки. В данном случае, это spring-boot-starter-parent, что означает, что проект наследует настройки сборки для Spring Boot.

<groupId>: Уникальный идентификатор группы для вашего проекта.

<artifactId>: Идентификатор артефакта (проекта).

<version>: Версия вашего проекта.

<name> и <description>: Имя и описание вашего проекта.

<properties>: Здесь вы можете определить свойства и переменные, которые используются в вашем проекте. В данном случае, указана версия Java.

<dependencies>: Список зависимостей вашего проекта. Зависимости указывают на библиотеки и фреймворки, которые ваш проект использует. В вашем случае, вы используете зависимости для Spring Boot, Spring Security, Spring Data JPA, Spring DevTools и другие.

<build>: Этот раздел определяет настройки сборки проекта. Внутри <plugins> вы указываете плагины, которые используются для сборки и управления проектом. В данном случае, используется spring-boot-maven-plugin, который интегрирует Spring Boot в процесс сборки Maven.

Этот файл pom.xml является важной частью проекта, так как он определяет все зависимости, конфигурации и настройки для сборки и запуска вашего Java-приложения с использованием Apache Maven.

Элемент <dependencies> в файле pom.xml используется для определения зависимостей вашего проекта. Зависимости представляют собой библиотеки и фреймворки, которые ваш проект использует для выполнения определенных функций. Maven использует информацию из этого раздела для автоматической загрузки и управления библиотеками и их версиями.

В определены следующие зависимости:

org.springframework.boot:spring-boot-starter-security: Зависимость Spring Boot Starter Security, которая предоставляет интеграцию Spring Security для обеспечения безопасности вашего приложения.

org.springframework.boot:spring-boot-starter-web: Зависимость Spring Boot Starter Web, которая предоставляет инструменты для разработки веб-приложений с использованием Spring MVC.

org.springframework.boot:spring-boot-starter-data-jpa: Зависимость Spring Boot Starter Data JPA, предоставляющая интеграцию Spring Data JPA для работы с базой данных.

org.springframework.boot:spring-boot-devtools: Зависимость Spring Boot DevTools, которая предоставляет инструменты для удобства разработки, такие как автоматическая перезагрузка при изменении кода.

org.springframework.boot:spring-boot-starter-test: Зависимость Spring Boot Starter Test, предназначенная для тестирования приложения, включая JUnit и другие библиотеки.

javax.persistence:javax.persistence-api: Зависимость, предоставляющая API для работы с Java Persistence (JPA), что полезно при использовании Spring Data JPA.

org.springframework.security:spring-security-test: Зависимость для тестирования Spring Security.

org.yaml:snakeyaml: Зависимость для работы с форматом YAML, который может использоваться, например, для конфигурации вашего приложения.

shipments.yaml – база данных в yaml формате

В фрагменте YAML представлена информация о почтовом отправлении с уникальным номером отслеживания например "ABC123". Отправление было отправлено с адреса "123 Main St, City A" и должно быть доставлено по адресу "456 Elm St, City B". На момент записи данных, статус отправления был "In transit" (в процессе доставки). Дата поступления отправления в службу и дата последнего изменения статуса указывают на "2023-09-30T10:00:00"

index.html – главная страница c поиском по трек - номером

result.html – подробнее о отправлении

create.html – создание отправления

.

## **Результат работы приложения**

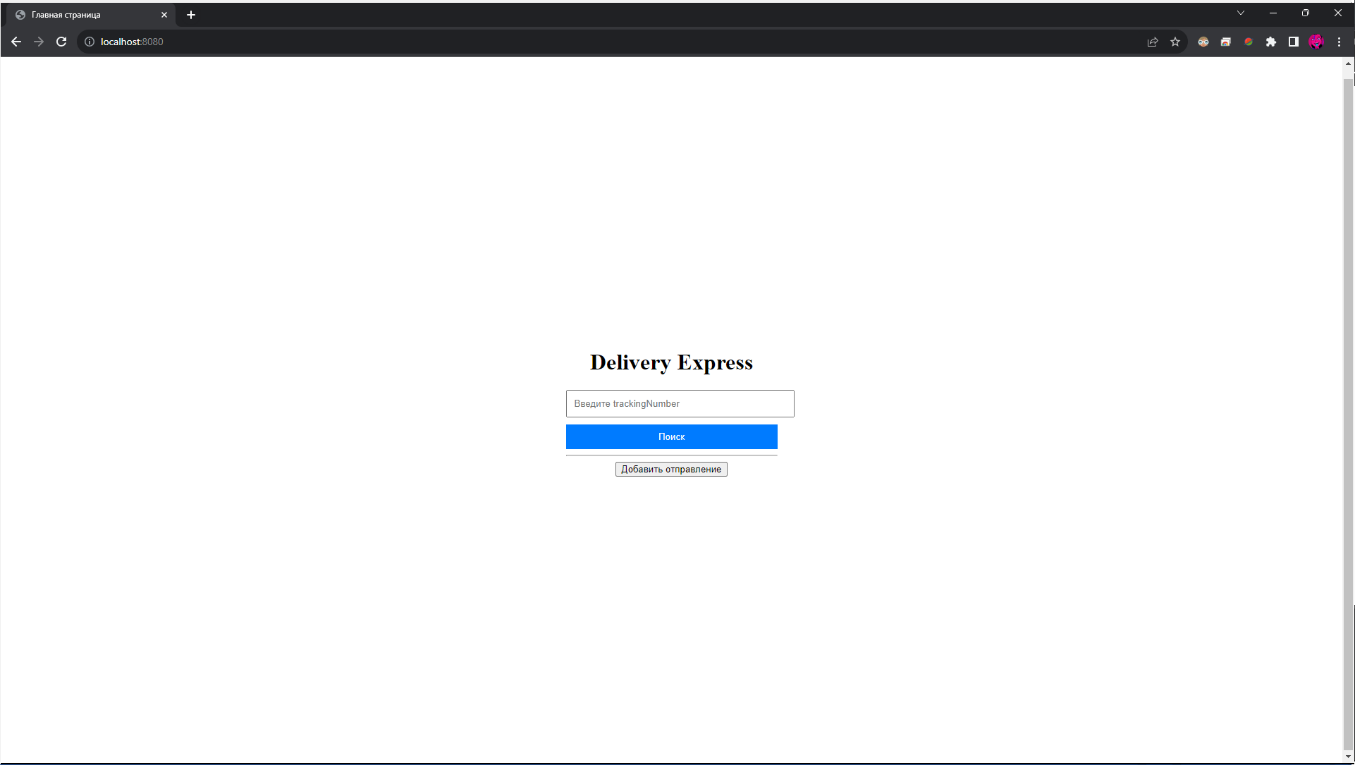
 На главной странице необходимо ввести правильный **trackingNumber** в случае, если введенного трек номера не существует программа ничего не выдаст.

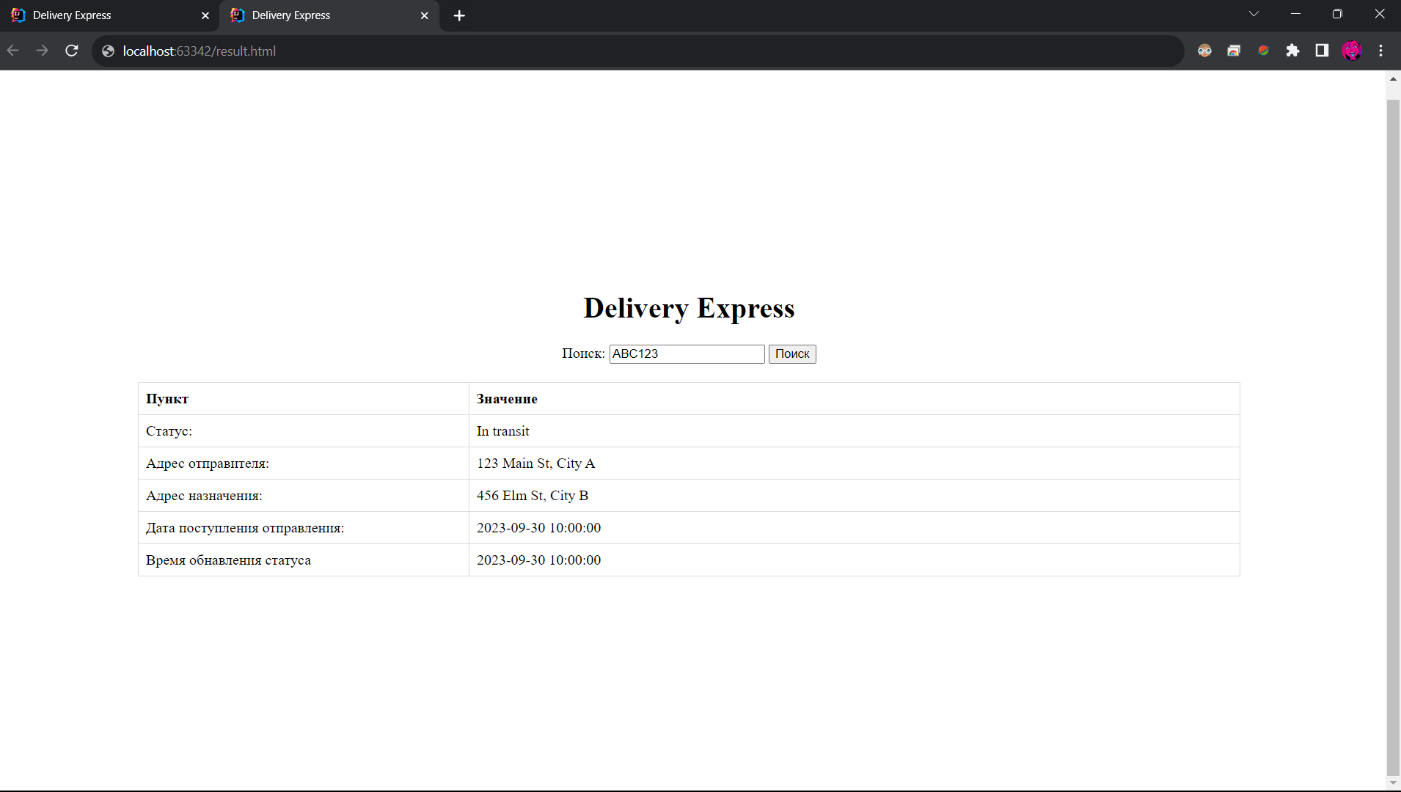
Рис 3 – главная страница

Рис 4 – отправление ABC123

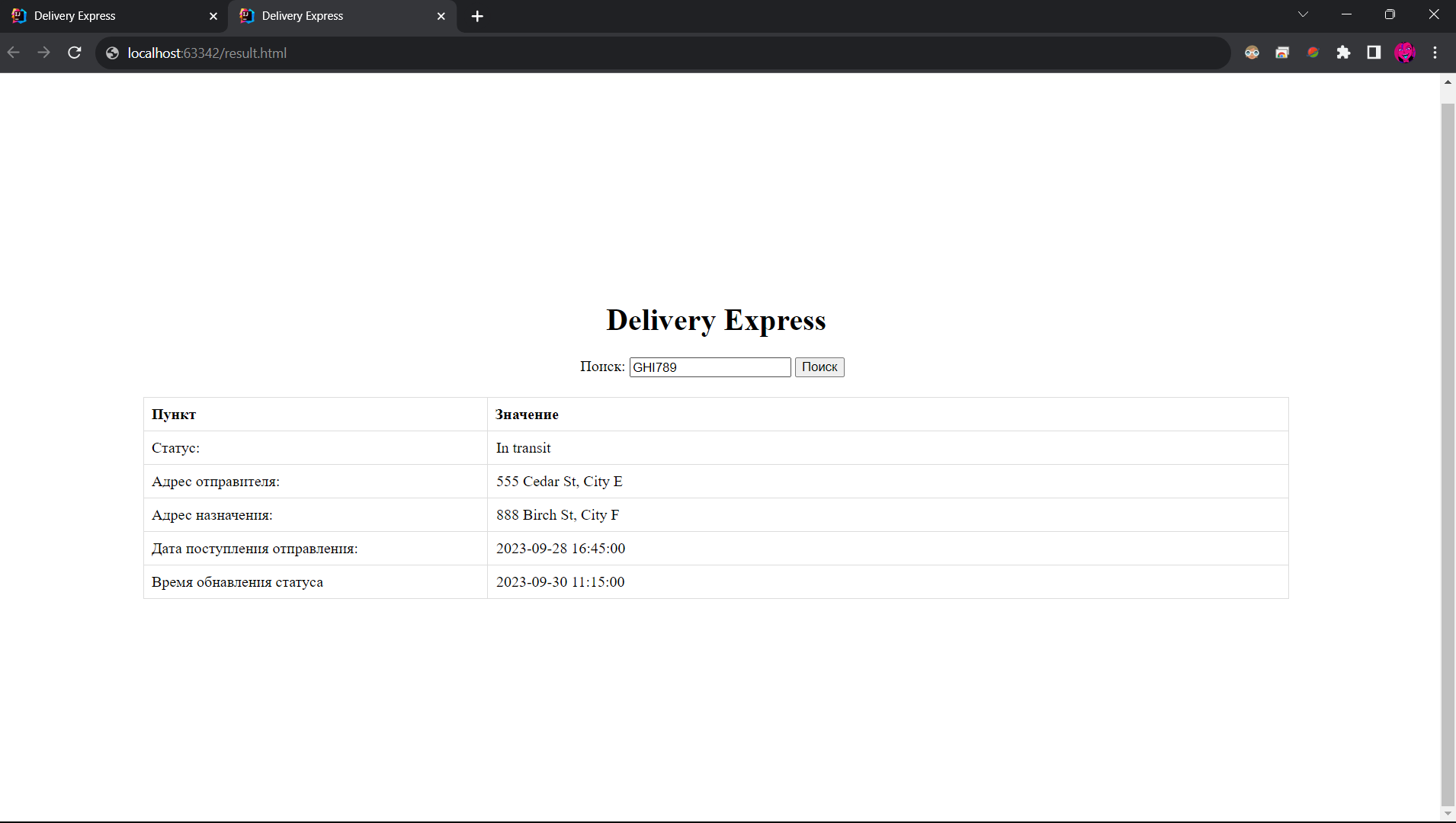
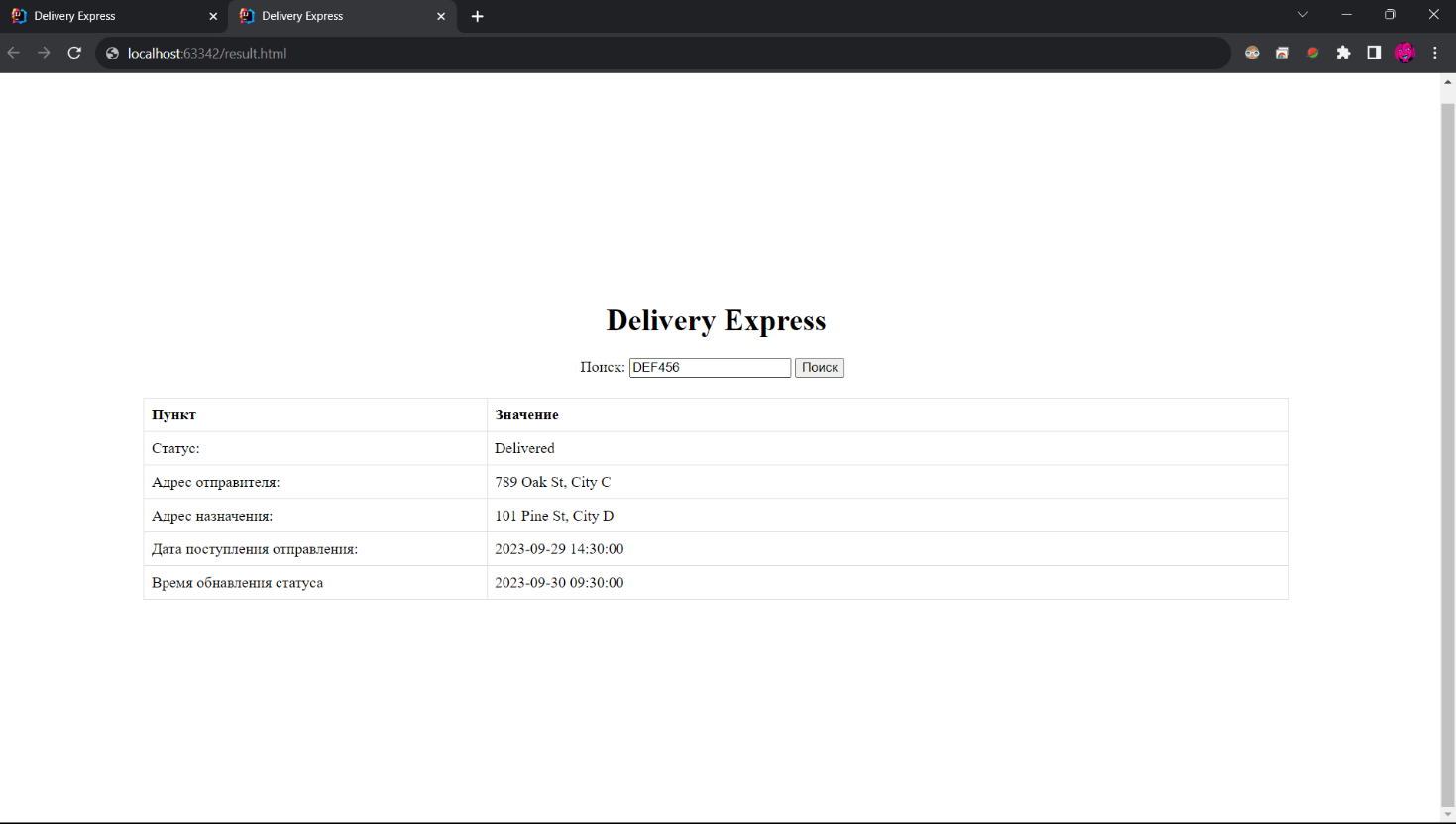
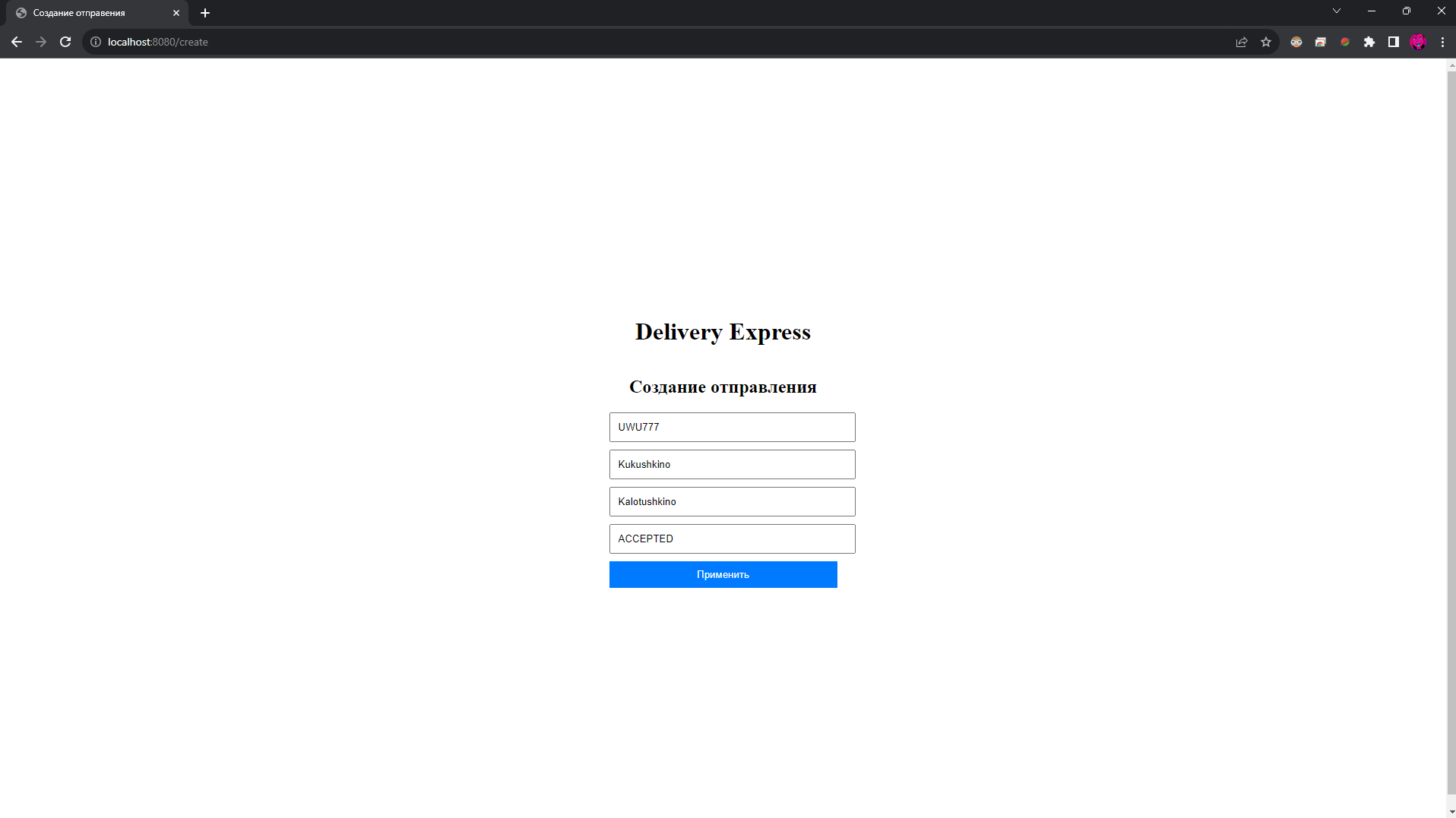
 Рис 5 – отправление DEF456

Рис 6 – отправление GHI789



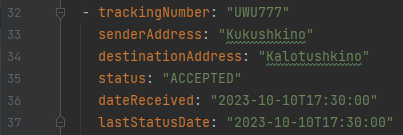
Рис 7 – создание отправления

Рис 8 – Результат создания отправления в БД yaml

# ВЫВОД

В рамках курсового проекта была разработана информационная система для управления и мониторинга почтовыми отправлениями, используя Java Spring и формат хранения данных YAML. Целью было исследование способов хранения, обработки и визуализации данных о почтовых отправлениях, а также создание примера приложения.

Для реализации данной системы мы использовали Java Spring, мощный фреймворк для создания Java-приложений. Spring предоставил нам инструменты для управления бизнес-логикой, внедрения зависимостей и создания веб-приложения. Мы также использовали Spring Boot, который упростил настройку и развертывание приложения. В итоге, созданное приложение предоставляет эффективные инструменты для контроля и мониторинга состояния почтовых отправлений с использованием одного из наиболее популярного веб-фреймворка Java Spring.

# ПРИЛОЖЕНИЕ

**Pom.xml**

**<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**

**<**project xmlns**=**"http://maven.apache.org/POM/4.0.0" xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"

xsi**:**schemaLocation**=**"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>**

**<**modelVersion**>**4.0.0**</**modelVersion**>**

**<**parent**>**

**<**groupId**>**org**.**springframework**.**boot**</**groupId**>**

**<**artifactId**>**spring**-**boot**-**starter**-**parent**</**artifactId**>**

**<**version**>**3.1.4**</**version**>**

**<**relativePath**/>** **<!--** lookup parent from repository **-->**

**</**parent**>**

**<**groupId**>**com**.**example**</**groupId**>**

**<**artifactId**>**DeliveryExpress**</**artifactId**>**

**<**version**>**0.0.1**-**SNAPSHOT**</**version**>**

**<**name**>**DeliveryExpress**</**name**>**

**<**description**>**DeliveryExpress**</**description**>**

**<**properties**>**

**<**java**.**version**>**17.0.2**</**java**.**version**>**

**</**properties**>**

**<**dependencies**>**

**<**dependency**>**

**<**groupId**>**org**.**springframework**.**boot**</**groupId**>**

**<**artifactId**>**spring**-**boot**-**starter**-**web**</**artifactId**>**

**</**dependency**>**

**<**dependency**>**

**<**groupId**>**org**.**springframework**.**boot**</**groupId**>**

**<**artifactId**>**spring**-**boot**-**devtools**</**artifactId**>**

**<**scope**>**runtime**</**scope**>**

**<**optional**>true</**optional**>**

**</**dependency**>**

**<**dependency**>**

**<**groupId**>**org**.**springframework**.**boot**</**groupId**>**

**<**artifactId**>**spring**-**boot**-**starter**-**test**</**artifactId**>**

**<**scope**>**test**</**scope**>**

**</**dependency**>**

**<**dependency**>**

**<**groupId**>**javax**.**persistence**</**groupId**>**

**<**artifactId**>**javax**.**persistence**-**api**</**artifactId**>**

**<**version**>**2.2**</**version**>**

**</**dependency**>**

**<**dependency**>**

**<**groupId**>**org**.**yaml**</**groupId**>**

**<**artifactId**>**snakeyaml**</**artifactId**>**

**</**dependency**>**

**</**dependencies**>**

**<**build**>**

**<**plugins**>**

**<**plugin**>**

**<**groupId**>**org**.**springframework**.**boot**</**groupId**>**

**<**artifactId**>**spring**-**boot**-**maven**-**plugin**</**artifactId**>**

**</**plugin**>**

**</**plugins**>**

**</**build**>**

**</**project**>**

**Shipments.yaml**

shipments**:**

**-** trackingNumber**:** "ABC123"

senderAddress**:** "123 Main St, City A"

destinationAddress**:** "456 Elm St, City B"

status**:** "In transit"

dateReceived**:** "2023-09-30T10:00:00"

lastStatusDate**:** "2023-09-30T10:00:00"

**-** trackingNumber**:** "DEF456"

senderAddress**:** "789 Oak St, City C"

destinationAddress**:** "101 Pine St, City D"

status**:** "Delivered"

dateReceived**:** "2023-09-29T14:30:00"

lastStatusDate**:** "2023-09-30T09:30:00"

**-** trackingNumber**:** "GHI789"

senderAddress**:** "555 Cedar St, City E"

destinationAddress**:** "888 Birch St, City F"

status**:** "In transit"

dateReceived**:** "2023-09-28T16:45:00"

lastStatusDate**:** "2023-09-30T11:15:00"

**-** trackingNumber**:** "JKL101"

senderAddress**:** "222 Willow St, City G"

destinationAddress**:** "777 Maple St, City H"

status**:** "In transit"

dateReceived**:** "2023-09-30T09:00:00"

lastStatusDate**:** "2023-09-30T09:00:00"

**-** trackingNumber**:** "MNO123"

senderAddress**:** "444 Pine St, City I"

destinationAddress**:** "999 Oak St, City J"

status**:** "In transit"

dateReceived**:** "2023-09-27T17:30:00"

lastStatusDate**:** "2023-09-30T10:45:00"

**-** trackingNumber**:** "UWU777"

senderAddress**:** "Kukushkino"

destinationAddress**:** "Kalotushkino"

status**:** "ACCEPTED"

dateReceived**:** "2023-10-10T17:30:00"

lastStatusDate**:** "2023-10-10T17:30:00"

**MainController.java**

package com**.**example**.**deliveryexpress**.**controller**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** com**.**example**.**deliveryexpress**.**service**.**ShipmentService**;**

**import** org**.**springframework**.**stereotype**.**Controller**;**

**import** org**.**springframework**.**ui**.**Model**;**

**import** org**.**springframework**.**web**.**bind**.**annotation**.\*;**

@Controller

public class MainController **{**

private final ShipmentService shipmentService**;**

public MainController**(**ShipmentService shipmentService**)** **{**

**this.**shipmentService **=** shipmentService**;**

**}**

@GetMapping**(**"/"**)**

public String showHomePage**(**Model model**)** **{**

**return** "index.html"**;**

**}**

@GetMapping**(**"/search"**)**

public String search**(**@RequestParam String trackingNumber**,** Model model**)** **{**

Shipment shipment **=** shipmentService**.**findShipmentByTrackingNumber**(**trackingNumber**);**

**if** **(**shipment **!=** **null)** **{**

model**.**addAttribute**(**"shipment"**,** shipment**);**

**return** "redirect:/result.html?trackingNumber=" **+** trackingNumber**;**

**}** **else** **{**

**return** "redirect:/index.html"**;**

**}**

**}**

@GetMapping**(**"/result"**)**

public String showResultPage**(**@RequestParam String trackingNumber**,** Model model**)** **{**

Shipment shipment **=** shipmentService**.**findShipmentByTrackingNumber**(**trackingNumber**);**

**if** **(**shipment **!=** **null)** **{**

model**.**addAttribute**(**"shipment"**,** shipment**);**

**return** "result.html"**;**

**}** **else** **{**

**return** "index.html"**;**

**}**

**}**

@GetMapping**(**"/create"**)**

public String showCreatePage**()** **{**

**return** "create.html"**;**

**}**

@PostMapping**(**"/create"**)**

public String createShipment**(**@ModelAttribute ShipmentForm shipmentForm**)** **{**

shipmentService**.**createShipmentFromForm**(**shipmentForm**);**

**return** "redirect:/index.html"**;**

**}**

**}**

**ShipmentController.java**

package com**.**example**.**deliveryexpress**.**controller**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentStatus**;**

**import** com**.**example**.**deliveryexpress**.**service**.**ShipmentService**;**

**import** org**.**springframework**.**beans**.**factory**.**annotation**.**Autowired**;**

**import** org**.**springframework**.**web**.**bind**.**annotation**.\*;**

**import** java**.**util**.**List**;**

@RestController

@RequestMapping**(**"/api/shipments"**)**

public class ShipmentController **{**

private final ShipmentService shipmentService**;**

@Autowired

public ShipmentController**(**ShipmentService shipmentService**)** **{**

**this.**shipmentService **=** shipmentService**;**

**}**

@PostMapping

public Shipment createShipment**(**@RequestBody Shipment shipment**)** **{**

// Обработка HTTP POST-запроса для создания нового отправления

**return** shipmentService**.**createShipment**(**shipment**);**

**}**

@PutMapping**(**"/{shipmentId}/status"**)**

public Shipment updateShipmentStatus**(**@PathVariable Long shipmentId**,** @RequestParam ShipmentStatus newStatus**)** **{**

// Обработка HTTP PUT-запроса для обновления статуса отправления

**return** shipmentService**.**updateShipmentStatus**(**shipmentId**,** newStatus**);**

**}**

@GetMapping

public List**<**Shipment**>** getAllShipments**()** **{**

// Обработка HTTP GET-запроса для получения всех отправлений

**return** shipmentService**.**getAllShipments**();**

**}**

@GetMapping**(**"/{shipmentId}"**)**

public Shipment getShipmentById**(**@PathVariable Long shipmentId**)** **{**

// Обработка HTTP GET-запроса для получения отправления по его ID

**return** shipmentService**.**getShipmentById**(**shipmentId**);**

**}**

**}**

**ShipmentForm.java**

package com**.**example**.**deliveryexpress**.**controller**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentStatus**;**

public class ShipmentForm **{**

private String trackingNumber**;**

private String senderAddress**;**

private String destinationAddress**;**

private ShipmentStatus status**;**

public String getTrackingNumber**()** **{**

**return** trackingNumber**;**

**}**

public void setTrackingNumber**(**String trackingNumber**)** **{**

**this.**trackingNumber **=** trackingNumber**;**

**}**

public String getSenderAddress**()** **{**

**return** senderAddress**;**

**}**

public void setSenderAddress**(**String senderAddress**)** **{**

**this.**senderAddress **=** senderAddress**;**

**}**

public String getDestinationAddress**()** **{**

**return** destinationAddress**;**

**}**

public void setDestinationAddress**(**String destinationAddress**)** **{**

**this.**destinationAddress **=** destinationAddress**;**

**}**

public ShipmentStatus getStatus**()** **{**

**return** status**;**

**}**

public void setStatus**(**ShipmentStatus status**)** **{**

**this.**status **=** status**;**

**}**

**}**

**Shipment.java**

package com**.**example**.**deliveryexpress**.**model**;**

**import** java**.**util**.**Date**;**

**import** javax**.**persistence**.\*;**

@Entity /// @Table(name = "shipments")

public class Shipment **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

private Long id**;**

@Column**(**nullable **=** **false,** unique **=** **true)**

private String trackingNumber**;**

@Column**(**nullable **=** **false)**

private String senderAddress**;**

@Column**(**nullable **=** **false)**

private String destinationAddress**;**

@Column**(**nullable **=** **false)**

@Enumerated**(**EnumType**.**STRING**)**

private ShipmentStatus status**;**

@Temporal**(**TemporalType**.**TIMESTAMP**)**

@Column**(**name **=** "date\_received"**,** nullable **=** **false)**

private Date dateReceived**;**

@Temporal**(**TemporalType**.**TIMESTAMP**)**

@Column**(**name **=** "last\_status\_date"**,** nullable **=** **false)**

private Date lastStatusDate**;**

// Геттеры и сеттеры для всех полей

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public String getTrackingNumber**()** **{**

**return** trackingNumber**;**

**}**

public void setTrackingNumber**(**String trackingNumber**)** **{**

**this.**trackingNumber **=** trackingNumber**;**

**}**

public String getSenderAddress**()** **{**

**return** senderAddress**;**

**}**

public void setSenderAddress**(**String senderAddress**)** **{**

**this.**senderAddress **=** senderAddress**;**

**}**

public String getDestinationAddress**()** **{**

**return** destinationAddress**;**

**}**

public void setDestinationAddress**(**String destinationAddress**)** **{**

**this.**destinationAddress **=** destinationAddress**;**

**}**

public ShipmentStatus getStatus**()** **{**

**return** ShipmentStatus**.**valueOf**(**String**.**valueOf**(**status**));**

**}**

public void setStatus**(**ShipmentStatus status**)** **{**

**this.**status **=** ShipmentStatus**.**valueOf**(**status**.**name**());**

**}**

public Date getDateReceived**()** **{**

**return** dateReceived**;**

**}**

public void setDateReceived**(**Date dateReceived**)** **{**

**this.**dateReceived **=** dateReceived**;**

**}**

public Date getLastStatusDate**()** **{**

**return** lastStatusDate**;**

**}**

public void setLastStatusDate**(**Date lastStatusDate**)** **{**

**this.**lastStatusDate **=** lastStatusDate**;**

**}**

@Override

public String toString**()** **{**

**return** "Shipment{" **+**

"id=" **+** id **+**

", trackingNumber='" **+** trackingNumber **+** '\'' **+**

", senderAddress='" **+** senderAddress **+** '\'' **+**

", destinationAddress='" **+** destinationAddress **+** '\'' **+**

", status='" **+** status **+** '\'' **+**

", dateReceived=" **+** dateReceived **+**

", lastStatusDate=" **+** lastStatusDate **+**

'}'**;**

**}**

**}**

**ShipmentsData.java**

package com**.**example**.**deliveryexpress**.**model**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

public class ShipmentsData **{**

private List**<**Shipment**>** shipments**;**

public ShipmentsData**(**List**<**Shipment**>** shipments**)** **{**

// Инициализация списка shipments

**this.**shipments **=** **new** ArrayList**<>();**

**}**

public List**<**Shipment**>** getShipments**()** **{**

**return** shipments**;**

**}**

public void setShipments**(**List**<**Shipment**>** shipments**)** **{**

**this.**shipments **=** shipments**;**

**}**

// Добавление новой отправки в список

public void addShipment**(**Shipment shipment**)** **{**

shipments**.**add**(**shipment**);**

**}**

// Получение отправки по индексу

public Shipment getShipment**(**int index**)** **{**

**if** **(**index **>=** 0 **&&** index **<** shipments**.**size**())** **{**

**return** shipments**.**get**(**index**);**

**}**

**return** **null;**

**}**

// Удаление отправки по индексу

public void removeShipment**(**int index**)** **{**

**if** **(**index **>=** 0 **&&** index **<** shipments**.**size**())** **{**

shipments**.**remove**(**index**);**

**}**

**}**

**}**

**ShipmentStatus.java**

package com**.**example**.**deliveryexpress**.**model**;**

public enum ShipmentStatus **{**

ACCEPTED**,**

IN\_TRANSIT**,**

FINAL\_DESTINATION**,**

SUCCESSFULLY\_DELIVERY**,**

ERROR\_DELIVERY

**}**

**ShipmentRepository.java**

package com**.**example**.**deliveryexpress**.**repository**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** org**.**springframework**.**stereotype**.**Repository**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Optional**;**

public interface ShipmentRepository **{**

Shipment findByTrackingNumber**(**String trackingNumber**);**

Shipment save**(**Shipment shipment**);**

List**<**Shipment**>** findAll**();**

Optional**<**Shipment**>** findById**(**Long id**);**

**}**

**YamlShipmentRepository.java**

package com**.**example**.**deliveryexpress**.**repository**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentsData**;**

**import** org**.**yaml**.**snakeyaml**.**constructor**.**Constructor**;**

**import** org**.**springframework**.**stereotype**.**Repository**;**

**import** org**.**yaml**.**snakeyaml**.**DumperOptions**;**

**import** org**.**yaml**.**snakeyaml**.**Yaml**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**InputStream**;**

**import** java**.**io**.\*;**

**import** java**.**lang**.**reflect**.**Array**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Optional**;**

@Repository

public class YamlShipmentRepository **implements** ShipmentRepository **{**

private final String yamlFilePath **=** "shipments.yaml"**;** // Путь к файлу YAML

@Override

public Shipment findByTrackingNumber**(**String trackingNumber**)** **{**

List**<**Shipment**>** shipments **=** loadShipmentsFromYamlFile**();**

**for** **(**Shipment shipment **:** shipments**)** **{**

**if** **(**shipment**.**getTrackingNumber**().**equals**(**trackingNumber**))** **{**

**return** shipment**;**

**}**

**}**

**return** **null;**

**}**

@Override

public Shipment save**(**Shipment shipment**)** **{**

List**<**Shipment**>** shipments **=** findAll**();**

shipments**.**add**(**shipment**);**

saveAll**(**shipments**);**

**return** shipment**;**

**}**

@Override

public List**<**Shipment**>** findAll**()** **{**

**try** **(**InputStream inputStream **=** **new** FileInputStream**(**yamlFilePath**))** **{**

Yaml yaml **=** **new** Yaml**();**

ShipmentsData shipmentsData **=** yaml**.**loadAs**(**inputStream**,** ShipmentsData**.**class**);**

**return** shipmentsData **!=** **null** **?** shipmentsData**.**getShipments**()** **:** **new** ArrayList**<>();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**return** **new** ArrayList**<>();**

**}**

**}**

@Override

public Optional**<**Shipment**>** findById**(**Long id**)** **{**

List**<**Shipment**>** shipments **=** findAll**();**

**return** shipments**.**stream**()**

**.**filter**(**shipment **->** shipment**.**getId**().**equals**(**id**))**

**.**findFirst**();**

**}**

private void saveAll**(**List**<**Shipment**>** shipments**)** **{**

**try** **(**OutputStream outputStream **=** **new** FileOutputStream**(**yamlFilePath**))** **{**

DumperOptions options **=** **new** DumperOptions**();**

options**.**setIndent**(**4**);**

Yaml yaml **=** **new** Yaml**(**options**);**

ShipmentsData shipmentsData **=** **new** ShipmentsData**(**shipments**);**

yaml**.**dump**(**shipmentsData**,** **new** OutputStreamWriter**(**outputStream**));**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

private List**<**Shipment**>** loadShipmentsFromYamlFile**()** **{**

Yaml yaml **=** **new** Yaml**(new** Constructor**(**Shipment**.**class**));**

**try** **(**InputStream inputStream **=** getClass**().**getClassLoader**().**getResourceAsStream**(**yamlFilePath**))** **{**

ShipmentsData shipmentsData **=** yaml**.**load**(**inputStream**);**

**return** shipmentsData**.**getShipments**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**return** **null;**

**}**

**}**

**ShipmentService.java**

package com**.**example**.**deliveryexpress**.**service**;**

**import** com**.**example**.**deliveryexpress**.**controller**.**ShipmentForm**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentStatus**;**

**import** java**.**util**.**List**;**

public interface ShipmentService **{**

Shipment createShipment**(**Shipment shipment**);**

Shipment updateShipmentStatus**(**Long shipmentId**,** ShipmentStatus newStatus**);**

List**<**Shipment**>** getAllShipments**();**

Shipment getShipmentById**(**Long shipmentId**);**

Shipment findShipmentByTrackingNumber**(**String trackingNumber**);**

void createShipmentFromForm**(**ShipmentForm shipmentForm**);**

**}**

**ShipmentServiceImpl.java**

package com**.**example**.**deliveryexpress**.**service**;**

**import** com**.**example**.**deliveryexpress**.**controller**.**ShipmentForm**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentStatus**;**

**import** com**.**example**.**deliveryexpress**.**repository**.**ShipmentRepository**;**

**import** org**.**springframework**.**beans**.**factory**.**annotation**.**Autowired**;**

**import** org**.**springframework**.**stereotype**.**Service**;**

**import** java**.**util**.**List**;**

@Service

public class ShipmentServiceImpl **implements** ShipmentService **{**

private final ShipmentRepository shipmentRepository**;**

@Autowired

public ShipmentServiceImpl**(**ShipmentRepository shipmentRepository**)** **{**

**this.**shipmentRepository **=** shipmentRepository**;**

**}**

@Override

public Shipment createShipment**(**Shipment shipment**)** **{**

// логика создания отправления, например, сохранение в репозитории

**return** shipmentRepository**.**save**(**shipment**);**

**}**

@Override

public Shipment updateShipmentStatus**(**Long shipmentId**,** ShipmentStatus newStatus**)** **{**

**return** **null;**

**}**

@Override

public List**<**Shipment**>** getAllShipments**()** **{**

**return** shipmentRepository**.**findAll**();**

**}**

@Override

public Shipment getShipmentById**(**Long shipmentId**)** **{**

**return** shipmentRepository**.**findById**(**shipmentId**).**orElse**(null);**

**}**

@Override

public Shipment findShipmentByTrackingNumber**(**String trackingNumber**)** **{**

**return** shipmentRepository**.**findByTrackingNumber**(**trackingNumber**);**

**}**

@Override

public void createShipmentFromForm**(**ShipmentForm shipmentForm**)** **{**

Shipment newShipment **=** **new** Shipment**();**

newShipment**.**setTrackingNumber**(**shipmentForm**.**getTrackingNumber**());**

newShipment**.**setSenderAddress**(**shipmentForm**.**getSenderAddress**());**

newShipment**.**setDestinationAddress**(**shipmentForm**.**getDestinationAddress**());**

//

**}**

**}**

**YamlShipmentService.java**

package com**.**example**.**deliveryexpress**.**service**;**

**import** com**.**example**.**deliveryexpress**.**controller**.**ShipmentForm**;**

**import** com**.**example**.**deliveryexpress**.**model**.**Shipment**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentStatus**;**

**import** com**.**example**.**deliveryexpress**.**model**.**ShipmentsData**;**

**import** com**.**example**.**deliveryexpress**.**repository**.**ShipmentRepository**;**

**import** com**.**fasterxml**.**jackson**.**databind**.**ObjectMapper**;**

**import** org**.**springframework**.**beans**.**factory**.**annotation**.**Autowired**;**

**import** org**.**springframework**.**context**.**annotation**.**Primary**;**

**import** org**.**springframework**.**stereotype**.**Service**;**

**import** org**.**yaml**.**snakeyaml**.**Yaml**;**

**import** org**.**yaml**.**snakeyaml**.**constructor**.**Constructor**;**

**import** java**.**io**.**FileWriter**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**InputStream**;**

**import** java**.**lang**.**reflect**.**Array**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**Optional**;**

@Service

@Primary

public class YamlShipmentService **implements** ShipmentService **{**

private final String YAML\_FILE **=** "shipments.yaml"**;**

private final ShipmentRepository shipmentRepository**;**

@Autowired

private ObjectMapper objectMapper**;**

@Autowired

public YamlShipmentService**(**ShipmentRepository shipmentRepository**)** **{**

**this.**shipmentRepository **=** shipmentRepository**;**

**}**

@Override

public Shipment createShipment**(**Shipment shipment**)** **{**

shipment**.**setDateReceived**(new** Date**());**

shipment**.**setLastStatusDate**(new** Date**());**

List**<**Shipment**>** shipments **=** loadShipmentsFromYamlFile**();**

shipments**.**add**(**shipment**);**

saveShipmentsToYamlFile**(**shipments**,** YAML\_FILE**);**

**return** shipment**;**

**}**

@Override

public Shipment updateShipmentStatus**(**Long shipmentId**,** ShipmentStatus newStatus**)** **{**

Optional**<**Shipment**>** optionalShipment **=** shipmentRepository**.**findById**(**shipmentId**);**

Shipment shipment**;**

**if** **(**optionalShipment**.**isPresent**())** **{**

shipment **=** optionalShipment**.**get**();**

System**.**out**.**println**(**"Найдено отправление:"**);**

System**.**out**.**println**(**"ID: " **+** shipment**.**getId**());**

**}** **else** **{**

**return** **null;**

**}**

shipment**.**setStatus**(**newStatus**);**

**return** shipmentRepository**.**save**(**shipment**);**

**}**

@Override

public List**<**Shipment**>** getAllShipments**()** **{**

Yaml yaml **=** **new** Yaml**();**

System**.**out**.**println**(**"File path: " **+** getClass**().**getResource**(**"/shipments.yaml"**));**

**try** **(**InputStream inputStream **=** getClass**().**getClassLoader**().**getResourceAsStream**(**YAML\_FILE**))** **{**

ShipmentsData shipmentsData **=** yaml**.**load**(**inputStream**);**

**return** shipmentsData**.**getShipments**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**return** **null;**

**}**

@Override

public Shipment getShipmentById**(**Long shipmentId**)** **{**

**return** **null;**

**}**

@Override

public Shipment findShipmentByTrackingNumber**(**String trackingNumber**)** **{**

List**<**Shipment**>** shipments **=** getAllShipments**();**

**for** **(**Shipment shipment **:** shipments**)** **{**

**if** **(**shipment**.**getTrackingNumber**().**equals**(**trackingNumber**))** **{**

**return** shipment**;**

**}**

**}**

// Если отправление не найдено, вернуть null

**return** **null;**

**}**

private List**<**Shipment**>** loadShipmentsFromYamlFile**()** **{**

Yaml yaml **=** **new** Yaml**();**

**try** **(**InputStream inputStream **=** getClass**().**getClassLoader**().**getResourceAsStream**(**YAML\_FILE**))** **{**

**if** **(**inputStream **!=** **null)** **{**

ShipmentsData shipmentsData **=** yaml**.**load**(**inputStream**);**

**return** shipmentsData**.**getShipments**();**

**}** **else** **{**

**throw** **new** RuntimeException**(**"Failed to load YAML file"**);**

**}**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**throw** **new** RuntimeException**(**"Failed to load YAML file"**,** e**);**

**}**

**}**

public void createShipmentFromForm**(**ShipmentForm shipmentForm**)** **{**

List**<**Shipment**>** shipments **=** loadShipmentsFromYamlFile**();**

Shipment newShipment **=** **new** Shipment**();**

newShipment**.**setTrackingNumber**(**shipmentForm**.**getTrackingNumber**());**

newShipment**.**setSenderAddress**(**shipmentForm**.**getSenderAddress**());**

newShipment**.**setDestinationAddress**(**shipmentForm**.**getDestinationAddress**());**

newShipment**.**setStatus**(**shipmentForm**.**getStatus**());**

shipments**.**add**(**newShipment**);**

saveShipmentsToYamlFile**(**shipments**,** YAML\_FILE**);**

**}**

private void saveShipmentsToYamlFile**(**List**<**Shipment**>** shipments**,** String filePath**)** **{**

Yaml yaml **=** **new** Yaml**();**

**try** **(**FileWriter fileWriter **=** **new** FileWriter**(**filePath**))** **{**

yaml**.**dump**(new** ShipmentsData**(**shipments**),** fileWriter**);**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**