Національний технічний університет України «Київський політехнічний інсти тут імені Ігоря Сікорського»

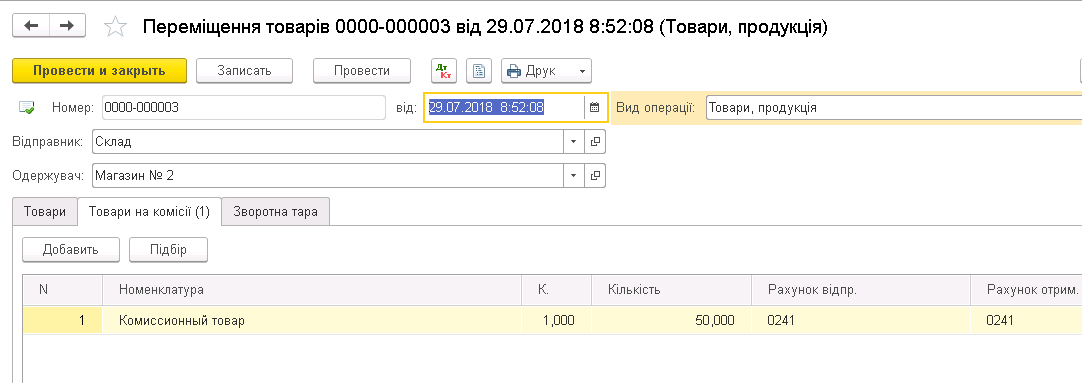
Практична робота № 4.1)

Студента Погорєлова  
Богдана Юрійовича  
группа ПМ-11

2021 рік

Програмне забезпечення автоматизованої системи управління з/д перевезеннями.

Аналіз ринку  
В Україні широкого розповсюдження зазнав комплекс програмного забезпечення 1С:Підприємство.  
  
 Ця система має такі поля:



* трек номер
* дата
* вид операції
* відправник
* отримувач
* товар (вид, кількість, . розрахунок)

Таким чином з замовником потрібно узгодити наступні питання:

* Інтерфейс системи
  + Гнучкість інтерфейсу
  + Цільову аудиторію користувачів
* Об’єми замовлень
* Логістику перевезень
* Термін підтримки програмного забезпечення
* Технічні характеристики пристроїв на яких працюватиме програмне забезпечення
  + Операційна система пристрою
  + Характеристику апаратного забезпечення
* Можливість офлайн роботи
* Характеристики мережі синхронізації
* Необхідність бекапів
* Вимоги ДСТУ
* Створення курсу підготовки кваліфікованих користувачів програмного забезпечення

Технічне завдання

Введення

Технічне завдання на розроблення програмного забезпечення автоматизованої системи управління з/д перевезеннями. ПЗ для автоматизованої системи менеджменту з/д перевезеннями. Даний продукт буде застосовуватись адміністраторами під час транспортного шляху вантажу. ПЗ буде реалізоване двома частинами: бекенд для адміністраторів для можливості внесення, перегляду та редагування інформації про вантаж, та бекендом на стороні сервера, який буде відповідати за логіку роботи. На даний момент існуючі застосунки не здатні виконати всі вимоги замовника. ПЗ буде покривати усі проблеми, котрі можна автоматизувати, та спростить роботу управління. У майбутньому буде забезпечення оновлення для виконання нових задач. Користувачами будуть оператори, адміністратори та менеджери підприємства.

Вимоги

Програмний продукт має виконувати наступні функції:

1. Створення накладної під час реєстрації вантажного перевезення
2. Зберігання інформації про вантажне перевезення
3. Контроль інформації про вантажне перевезення
4. Створення, коригування та перевірка маршруту транспорту вантажного перевезення
5. Реалізація засобу підтвердження про вантажне перевезення на пропускних пунктах
6. Перевірка вантажу та закриття квитанції на кінцевій точці
7. Інтеграція з наявною системої менеджменту логістики перевезення вантажів
8. Контролювання прав доступу шляхом паролю, фізичного ключа та двофакторної аутентифікації
9. Інтеграція у наявне ПЗ бухгалтерського обліку

Інформація для квитанції перевезення:

1. Назва товару
2. Кількість товару
3. Маршрут перевезення
4. Ідентифікаційний номер
5. Реквізити замовника

Вимоги до продуктивності:

1. Масштабування для майбутнього збільшення кількості обігу вантажів
2. Можливість роботи під час старту на не вибагливому апаратному забезпечені
3. Можливість роботи на різноманітних каналах зв’язку

Вимоги до зручності використання та інтерфейсу:

Особливі вимоги відсутні, продукт повинний бути реалізованим в рамках норм UI/UX

Вимоги до бази даних:

1. Виконання норм захисту даних
2. Можливість інтеграції з наявною системою зберігання інформації

Проектні обмеження:

1. Необхідний робочий продукт в терміні 6 місяців
2. Реалізація підтримки ПЗ терміном 5 років

Додаток

Скорочення та акроніми:

* ПЗ – сукупність програм системи оброблення інформації та програмних документів, необхідних для експлуатації цих програм.
* UX (User Experience) перекладається як «користувацький досвід». Призначений для користувача досвід визначається тим, як користувачі взаємодіють з додатком/сервісом.
* UI (User Interface) перекладається як «призначений для користувача інтерфейс».

ПЗ

ПЗ складається з класів:

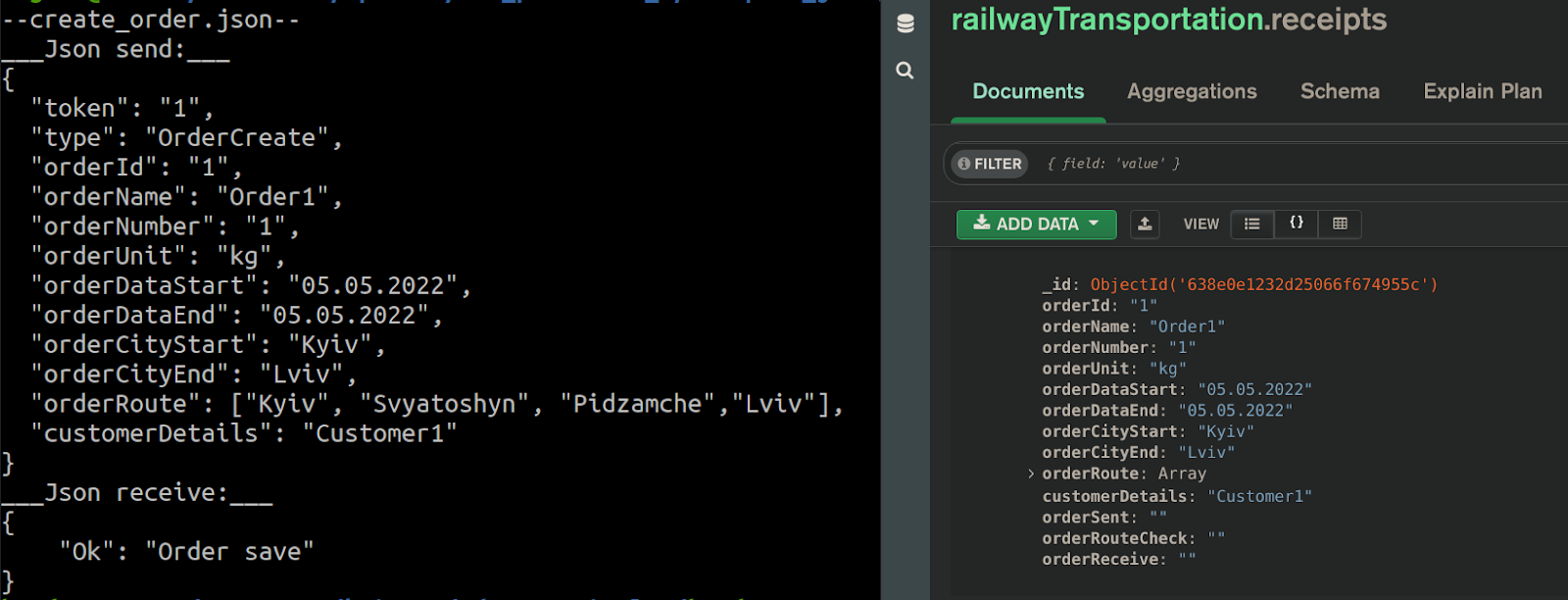
* TokenManager — програма для створення, відображення та видалення токенів доступу до бази накладних.
* Main — точка входу програми: запуск http серверу для обслуговквання json запитів та під’єднання до бази данних.
* Server
  + Start — клас для реалізації методів серверу: запуск та обробка запитів.
  + Dispatcher — клас обробки запитів.
* DB
  + Mongo — клас для реалізації методів бази данних: підключення та створення запитів.
  + Query — клас для реалізації необхідних запитів до БД.

Використані бібліотеки:

* org.mongodb
* com.sun.net.httpserver
* org.bson
* java.util.logging.Level
* java.text.SimpleDateFormat
* java.util.Date
* java.io.IOException
* java.net.InetSocketAddress
* java.nio.charset.Charset
* java.nio.charset.StandardCharsets

Приклади роботи ПЗ

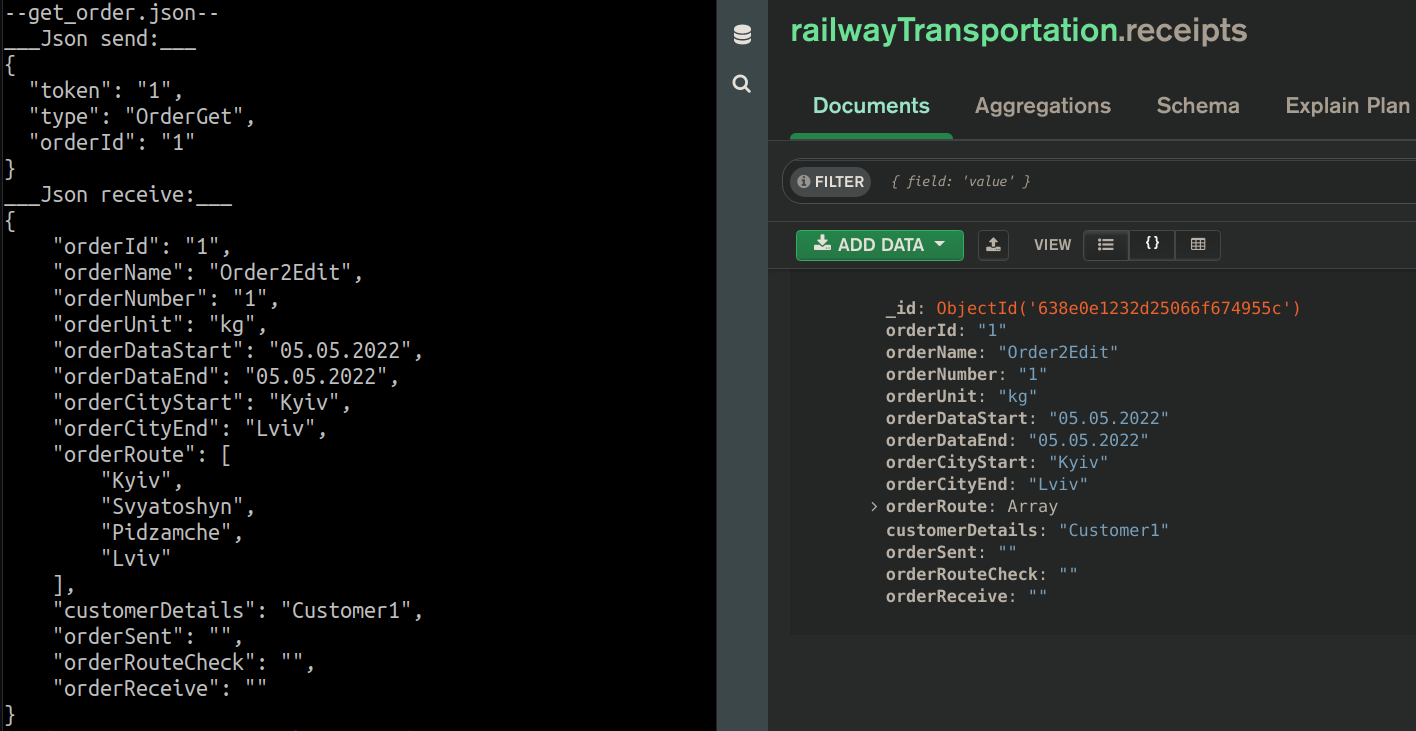
Відправка запиту настворення накладної (зліва) та вміст бази данних (справа)



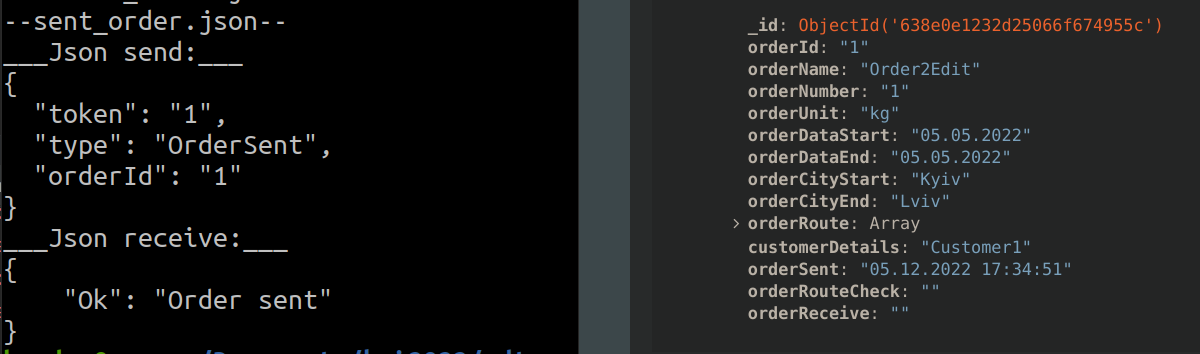
Відправка запиту на редагування імені накладної



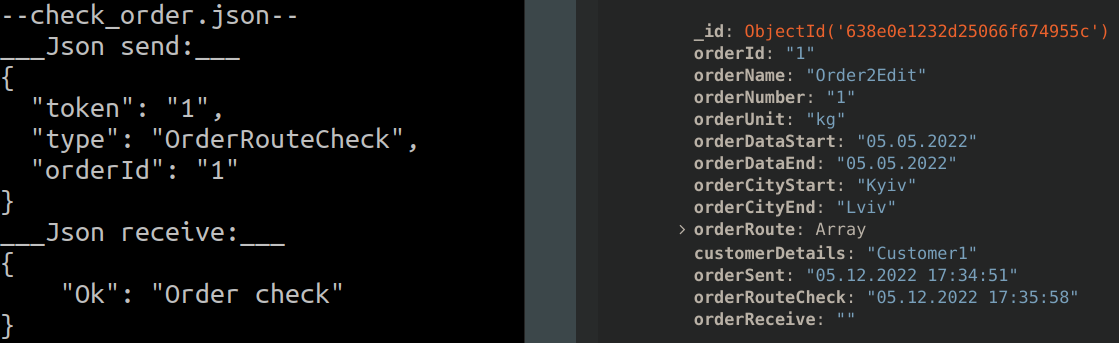
Відпрака запиту на отримання данних накладної



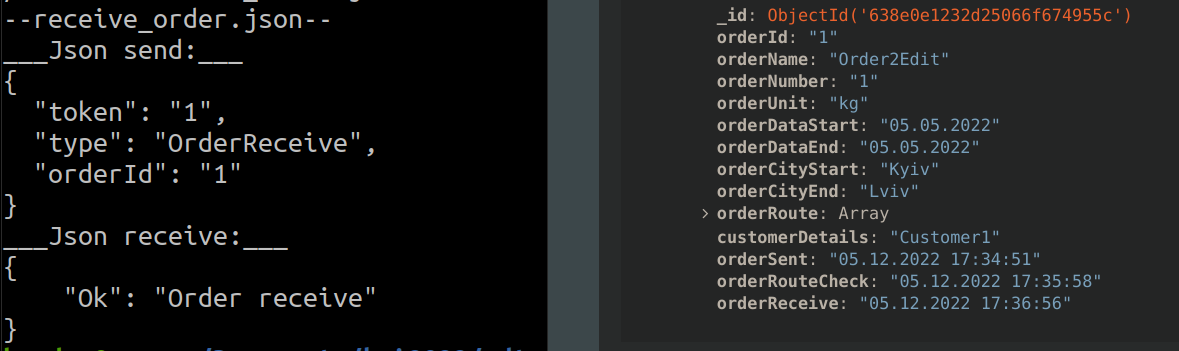
Відправка сигналу про надсилання товару



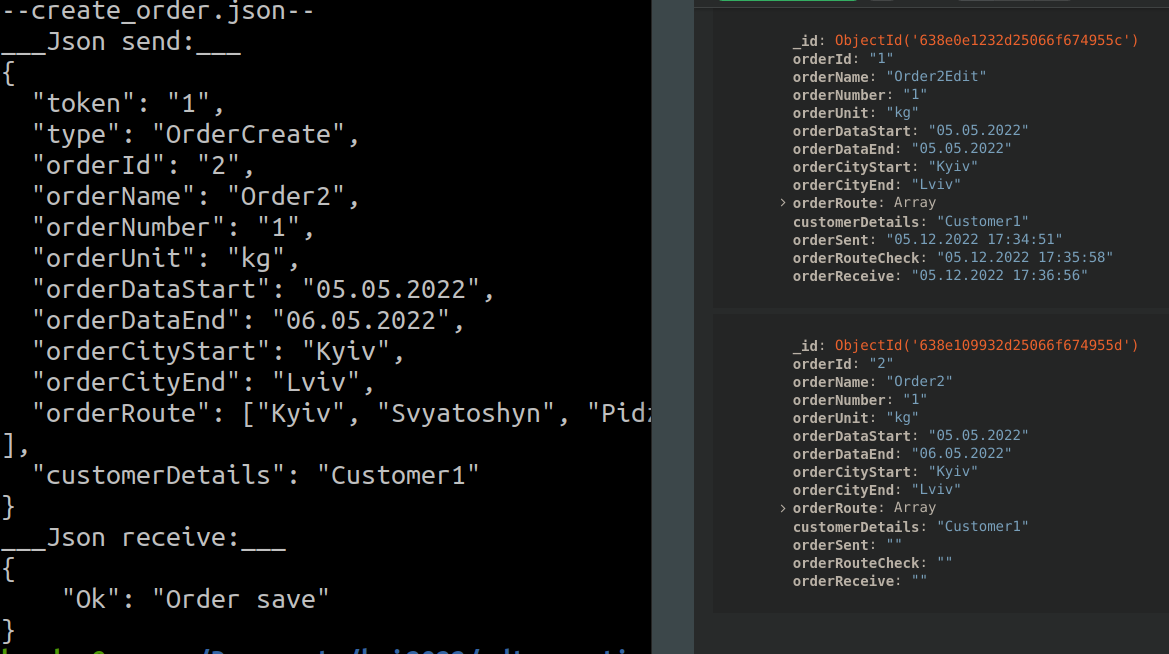
Відправка сигналу про перевірку товару



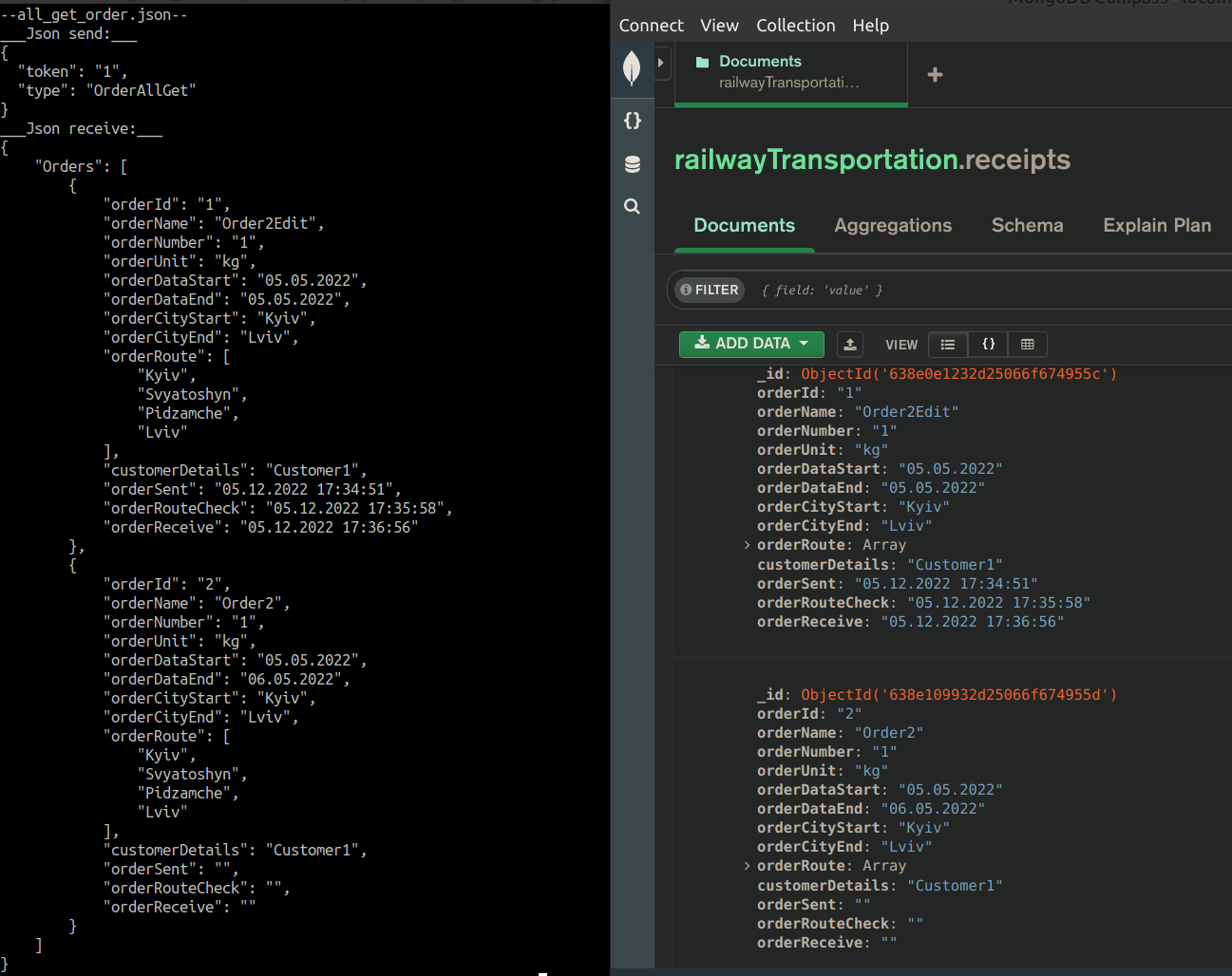
Відправка сигналу про отримання товару



Створення накладної

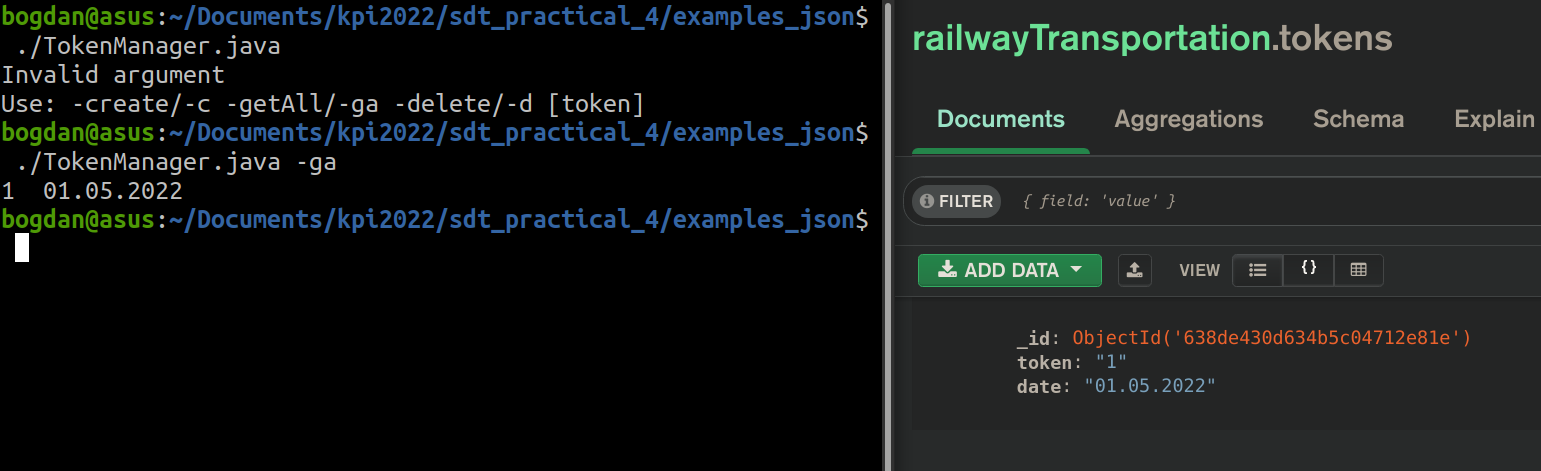


Відправка запиту на отримання данних про всі накладні

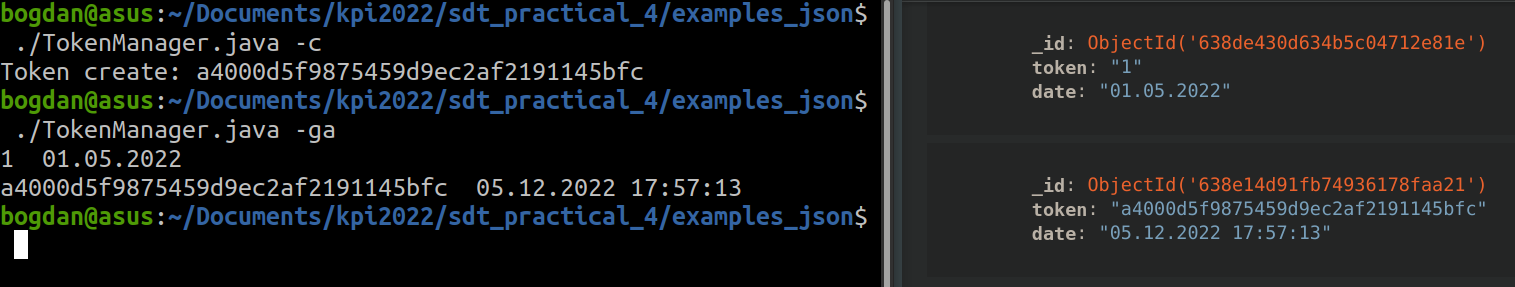


Запуск класу TokenManager з аргументами

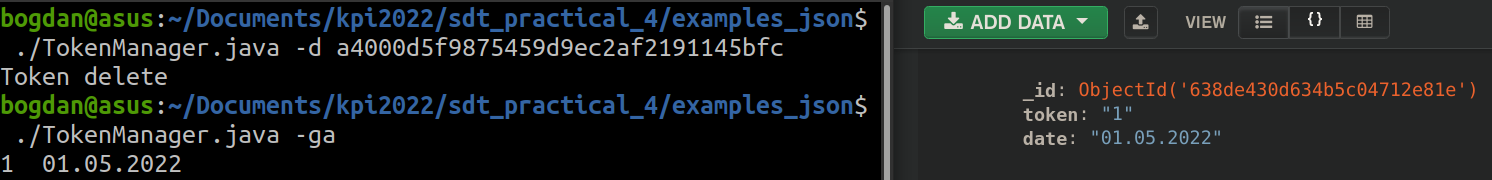
Запуск з відсутністю аргументів, отримання усіх токенів



Створення токену, отримання усіх токенів



Видалення токену, отримання усіх токенів



Код

Клас Main

package org.sdt\_practical\_4;

import java.io.IOException;

import java.util.logging.Level;

import static java.util.logging.Logger.*getLogger*;

import static org.sdt\_practical\_4.DB.Mongo.*connect*;

import static org.sdt\_practical\_4.Server.Start.*startServer*;

public class Main {

public static void main(String[] args) throws IOException {

*getLogger*("org.mongodb").setLevel(Level.*OFF*);

*connect*();

*startServer*();

}

}

Клас TokenManager

package org.sdt\_practical\_4;

import org.sdt\_practical\_4.DB.Mongo;

import org.sdt\_practical\_4.DB.Query;

import java.util.UUID;

import java.util.logging.Level;

import static java.util.logging.Logger.getLogger;

public class TokenManager {

public static void main(String[] args){

getLogger("org.mongodb").setLevel(Level.OFF);

Mongo.connect();

if (args.length > 0) {

switch (args[0]) {

case "-create": case "-c":

String token = UUID.randomUUID().toString().replace("-", "");

Query.tokenCreate(token);

System.out.println("Token create: " + token);

break;

case "-getAll": case "-ga":

System.out.println(Query.tokenGetAll());

break;

case "-delete": case "-d":

if(args.length == 2){

System.out.println(Query.tokenDelete(args[1]));

} else { error(); }

break;

default:

error();

break;

}

} else { error(); }

Mongo.disconnect();

}

private static void error(){

System.out.println("Invalid argument");

System.out.println("Use: -create/-c -getAll/-ga -delete/-d [token]");

}

}

Клас Mongo

package org.sdt\_practical\_4.DB;

import com.mongodb.MongoClient;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

public class Mongo {

static MongoClient client;

static MongoCollection<Document> receiptsDB;

static MongoCollection<Document> tokensDB;

public static void connect(){

try {

client = new MongoClient("localhost", 27017);

MongoDatabase database = client.getDatabase("railwayTransportation");

receiptsDB = database.getCollection("receipts");

tokensDB = database.getCollection("tokens");

} catch(Exception e){ System.out.println(e); }

}

public static void disconnect(){ client.close(); }

}

Клас Query

package org.sdt\_practical\_4.DB;

import com.mongodb.client.MongoCursor;

import org.bson.Document;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Query extends Mongo {

public static String idCorrect(Object id){

if (id == null){ return "empty"; }

final Document doc = receiptsDB.find(new Document("orderId", id.toString())).first();

if (doc == null){ return "not found"; }

return "found";

}

public static String orderGetAll() {

StringBuilder items = new StringBuilder();

MongoCursor<Document> cursor = receiptsDB.find().iterator();

try (cursor) {

while (cursor.hasNext()) {

Document item = cursor.next();

item.remove("\_id");

items.append(item.toJson());

if (cursor.hasNext()) { items.append(","); }

}

}

return String.format("{\"Orders\": [ %s ]}", items); //items.toString()

}

public static String orderGet(String id){

if (idCorrect(id).equals("found")) {

final Document doc = receiptsDB.find(new Document("orderId", id)).first();

assert doc != null;

doc.remove("\_id");

return doc.toJson();

} return new Document("Error", "Invalid orderId").toJson();

}

public static String orderCreate(Document request){

try { receiptsDB.insertOne(

new Document("orderId", request.get("orderId"))

.append("orderName", request.get("orderName"))

.append("orderNumber", request.get("orderNumber"))

.append("orderUnit", request.get("orderUnit"))

.append("orderDataStart", request.get("orderDataStart"))

.append("orderDataEnd", request.get("orderDataEnd"))

.append("orderCityStart", request.get("orderCityStart"))

.append("orderCityEnd", request.get("orderCityEnd"))

.append("orderRoute", request.get("orderRoute"))

.append("customerDetails", request.get("customerDetails"))

.append("orderSent", "")

.append("orderRouteCheck", "")

.append("orderReceive", "") );

return new Document("Ok", "Order save").toJson();

} catch (Exception e) { return new Document("Error", "Order not save").toJson(); }

}

public static String orderRouteCheck(Document request) {

return setQuery(request.get("orderId").toString(),"orderRouteCheck", dateNow(), "check");

}

public static String orderSent(Document request) {

return setQuery(request.get("orderId").toString(),"orderSent", dateNow(), "sent");

}

public static String orderReceive(Document request) {

return setQuery(request.get("orderId").toString(),"orderReceive", dateNow(), "receive");

}

public static String orderEdit(Document request) {

return setQuery(request.get("orderId").toString(), request.get("editKey").toString(),

request.get("editValue"), "edit");

}

private static String setQuery(String id, String editKey, Object editValue, String msg){

try { receiptsDB.updateOne(

new Document("orderId", id),

new Document( "$set", new Document(editKey, editValue) ) );

return new Document("Ok", "Order " + msg).toJson();

} catch (Exception e) { return new Document("Error", "Order not " + msg).toJson(); }

}

public static boolean tokenCheckFalse(String token) {

final Document doc = tokensDB.find(new Document("token", token)).first();

if (doc == null || token.length() == 0){ return true;}

return !doc.get("token").toString().equals(token);

}

public static void tokenCreate(String token){

tokensDB.insertOne( new Document("token", token).append("date", dateNow()) );

}

public static String tokenGetAll(){

StringBuilder items = new StringBuilder();

MongoCursor<Document> cursor = tokensDB.find().iterator();

try (cursor) {

while (cursor.hasNext()) {

Document item = cursor.next();

items.append(item.get("token")).append(" ").append(item.get("date"));

if (cursor.hasNext()) { items.append("\n"); }

}

}

return items.toString();

}

public static String tokenDelete(String token){

if (tokenCheckFalse(token)) { return "Invalid token"; }

tokensDB.deleteOne(new Document("token", token));

return "Token delete";

}

private static String dateNow() {

SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");

Date date = new Date();

return formatter.format(date);

}

}

Клас Start

package org.sdt\_practical\_4.Server;

import com.sun.net.httpserver.Headers;

import com.sun.net.httpserver.HttpServer;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.nio.charset.Charset;

import java.nio.charset.StandardCharsets;

public class Start {

private static final Charset CHARSET = StandardCharsets.UTF\_8;

public static void startServer() throws IOException {

final HttpServer server = HttpServer.create(new InetSocketAddress("localhost", 8080), 1);

server.createContext("/response", he -> {

try (he) {

final Headers headers = he.getResponseHeaders();

final String requestMethod = he.getRequestMethod().toUpperCase();

if ("POST".equals(requestMethod)) {

final String requestBody = new String(he.getRequestBody().readAllBytes());

final byte[] response = Dispatcher.getResponse(requestBody).getBytes(CHARSET);

headers.set("Content-Type", String.format("application/json; charset=%s", CHARSET));

he.sendResponseHeaders(200, response.length);

he.getResponseBody().write(response);

} else {

headers.set("Allow", "POST");

he.sendResponseHeaders(405, -1);

}

}

});

server.start();

}

}

Клас Dispatcher

package org.sdt\_practical\_4.Server;

import org.bson.Document;

import org.sdt\_practical\_4.DB.Query;

public class Dispatcher {

public static String getResponse(String requestString) {

final Document request = Document.parse(requestString);

if (Query.tokenCheckFalse(request.get("token").toString())) { return getError("Invalid token"); }

String isIdCorrect = Query.idCorrect(request.get("orderId"));

switch (request.get("type").toString()) {

case "OrderCreate" -> {

if (!orderCreateFieldsCheck(request)) { return getError("Required fields are missing"); }

if (isIdCorrect.equals("found")) { return getError("OrderId already use"); }

return Query.orderCreate(request);

}

case "OrderAllGet" -> {

if (!isIdCorrect.equals("empty")) { return getError("Not need orderId "); }

return Query.orderGetAll();

}

}

if (!isIdCorrect.equals("found")) { return getError("Invalid orderId"); }

switch (request.get("type").toString()) {

case "OrderGet" -> { return Query.orderGet(request.get("orderId").toString()); }

case "OrderRouteCheck" -> { return Query.orderRouteCheck(request); }

case "OrderSent" -> { return Query.orderSent(request); }

case "OrderReceive" -> { return Query.orderReceive(request); }

case "OrderEdit" -> {

if (!orderEditFieldsCheck(request)) { return getError("Required fields are missing"); }

return Query.orderEdit(request);

}

default -> { return getError("Invalid request type"); }

}

}

private static boolean orderCreateFieldsCheck(Document request){

final String[] orderFields = {"orderName", "orderNumber", "orderUnit",

"orderDataStart","orderDataEnd", "orderCityStart",

"orderCityEnd", "orderRoute", "customerDetails",};

return fieldsCheck(request, orderFields);

}

private static boolean orderEditFieldsCheck(Document request){

final String[] orderFields = {"editKey", "editValue"};

return fieldsCheck(request, orderFields);

}

private static boolean fieldsCheck(Document request, String[] orderFields){

for (String field: orderFields){

String item = request.get(field).toString();

if (item.equals("null") || item.length()==0) { return false; }

}

return true;

}

private static String getError(String text){ return new Document("Error", text).toJson(); }

}