

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Системотехніки

Звіт

з практичної роботи №2

з дисципліни: «Інтернет-технології Розподіленої Обробки Інформації»

з теми: «СТВОРЕННЯ САМОСТІЙНОГО REST ВЕБ-СЕРВІСУ»

Виконав:

здобувач освіти першого  
(бакалаврського) рівня освіти гр.  
КНТ-22-1  
Орлов О. С.

Перевірила:

Асистент кафедри СТ  
Мірошниченко Н.С.

## 2 СТВОРЕННЯ САМОСТІЙНОГО REST ВЕБ-СЕРВІСУ

### 2.1 Мета роботи

Вивчення можливостей і засобів розробки REST Веб-сервісів.

### 2.2 Хід роботи

Для виконання роботи було обрано тему ІС «Логістична компанія». Після початкового аналізу було виділено три основні сутності: транспортні засоби, вантажі та маршрути.

```
plugins {
    id("java")
    alias(libs.plugins.spring.boot)
}

group = "ua.com.dxrkness"
version = "1.0-SNAPSHOT"

repositories {
    mavenCentral()
}

dependencies {
    // spring
    implementation(platform(org.springframework.boot.gradle.plugin.SpringBootP
    implementation(libs.spring.boot.starter.web)
    implementation(libs.spring.boot.starter.web.test)
    implementation(libs.spring.boot.starter.hateoas)
    developmentOnly(libs.spring.boot.devtools)

    // http client
    implementation(libs.apache.http.client)

    // openapi docs
    implementation(libs.springdoc.openapi.starter.webmvc.ui)

    // xml
    implementation(libs.jackson.dataformat.xml)

    // testing
    testImplementation(platform(libs.junit.bom))
    testImplementation(libs.junit.jupiter)
    testRuntimeOnly(libs.junit.platform.launcher)

    implementation(libs.jspecify)
}

tasks.test {
    useJUnitPlatform()
}

[versions]
junit = "6.0.1"
spring-boot-plugin = "4.0.0"
```

```

jspecify = "1.0.0"
springdoc = "3.0.0"
apache-http-client = "4.5.14"

[plugins]
spring-boot = { id = "org.springframework.boot", version.ref =
"spring-boot-plugin" }

[libraries]
# spring
spring-boot-starter-web = { group = "org.springframework.boot", name =
"spring-boot-starter-webmvc" }
spring-boot-starter-web-test = { group = "org.springframework.boot",
name = "spring-boot-starter-webmvc-test" }
spring-boot-starter-hateoas = { group = "org.springframework.boot",
name = "spring-boot-starter-hateoas" }
spring-boot-devtools = { group = "org.springframework.boot", name =
"spring-boot-devtools" }

# http client
apache-http-client = { group = "org.apache.httpcomponents", name =
"httpclient", version.ref = "apache-http-client" }

# openapi
springdoc-openapi-starter-webmvc-ui = { group = "org.springdoc", name =
"springdoc-openapi-starter-webmvc-ui", version.ref = "springdoc" }

# xml
jackson-dataformat-xml = { group = "tools.jackson.dataformat", name =
"jackson-dataformat-xml" }

# testing
junit-bom = { group = "org.junit", name = "junit-bom", version.ref =
"junit" }
junit-jupiter = { group = "org.junit.jupiter", name = "junit-
jupiter" }
junit-platform-launcher = { group = "org.junit.platform", name =
"junit-platform-launcher" }

jspecify = { group = "org.jspecify", name = "jspecify", version.ref =
"jspecify" }

# build cache
org.gradle.caching=true

org.gradle.configuration-cache=true
org.gradle.configuration-cache.problems=warn
org.gradle.configuration-cache.parallel=true

rootProject.name = "itroi"
enableFeaturePreview("STABLE_CONFIGURATION_CACHE")

package ua.com.dxrkness;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LogisticsApplication {
    static void main(String[] args) {

```

```

        SpringApplication.run(LogisticsApplication.class, args);
    }
}

package ua.com.dxrkness.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.jspecify.annotations.NullMarked;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.com.dxrkness.model.Freight;
import ua.com.dxrkness.service.FreightService;

import java.util.List;

@RestController
@RequestMapping(
    value = "/freights",
    produces = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE},
    consumes = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE}
)
@Tag(name = "Freight", description = "Freights management")
@NullMarked
public class FreightController {
    private final FreightService service;

    public FreightController(FreightService service) {
        this.service = service;
    }

    @Operation(
        summary = "Retrieve all freights",
        description = "Gets a list of all freight records in the
system"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Successfully retrieved list of freights
(may be empty!)",
        content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            schema = @Schema(implementation = Freight.class)
        )
    )
    @GetMapping(consumes = MediaType.ALL_VALUE)
    public List<Freight> getAll() {
        return service.getAll();
    }

    @Operation(
        summary = "Get freight by ID",
        description = "Finds a freight record by its id"
    )
    @ApiResponse(

```

```

        responseCode = "200",
        description = "Successfully retrieved freight",
        content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            schema = @Schema(implementation = Freight.class)
        )
    )
    @ApiResponse(
        responseCode = "404",
        description = "Freight not found"
    )
    @GetMapping(value =("/{id}", consumes = MediaType.ALL_VALUE)
    public ResponseEntity<Freight> getById(@PathVariable("id") long
id) {
        return ResponseEntity.ofNullable(service.getById(id));
    }

    @Operation(
        summary = "Create a new freight",
        description = "Adds a new freight"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Freight successfully created",
        content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            schema = @Schema(implementation = Freight.class)
        )
    )
    @ApiResponse(
        responseCode = "400",
        description = "Invalid input"
    )
    @PostMapping
    public ResponseEntity<Freight> add(@RequestBody Freight
newFreight) {
        return ResponseEntity.ok(service.add(newFreight));
    }

    @Operation(
        summary = "Update a freight (full)",
        description = "Updates all fields of an existing freight
record"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Freight successfully updated",
        content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            schema = @Schema(implementation = Freight.class)
        )
    )
    @ApiResponse(
        responseCode = "400",
        description = "Invalid input"
    )
    @PutMapping("/{id}")
    public ResponseEntity<Freight> update(@PathVariable("id") long
id, @RequestBody Freight newFreight) {
        return ResponseEntity.ok(service.update(id, newFreight));
    }

```

```

        @Operation(
            summary = "Update a freight (partial)",
            description = "Updates specific fields of an existing
freight record"
        )
        @ApiResponse(
            responseCode = "200",
            description = "Freight successfully updated",
            content = @Content(
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                schema = @Schema(implementation = Freight.class)
            )
        )
        @ApiResponse(
            responseCode = "400",
            description = "Invalid input"
        )
        @PatchMapping("/{id}")
        public ResponseEntity<Freight> updatePatch(@PathVariable("id")
long id, @RequestBody Freight newFreight) {
            return ResponseEntity.ok(service.update(id, newFreight));
        }

        @Operation(
            summary = "Delete a freight",
            description = "Removes a freight"
        )
        @ApiResponse(
            responseCode = "200",
            description = "Freight successfully deleted",
            content = @Content(
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                schema = @Schema(implementation = Freight.class)
            )
        )
        @ApiResponse(
            responseCode = "404",
            description = "Freight not found"
        )
        @DeleteMapping(value =("/{id}", consumes = MediaType.ALL_VALUE)
public ResponseEntity<Freight> delete(@PathVariable("id") long
id) {
            return ResponseEntity.ofNullable(service.delete(id));
        }
    }
}

```

```
package ua.com.dxrkness.controller;
```

```

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.jspecify.annotations.NullMarked;
import org.jspecify.annotations.Nullable;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.com.dxrkness.model.Route;
import ua.com.dxrkness.service.RouteService;

```

```

import java.util.List;

@RestController
@RequestMapping(
    value = "/routes",
    produces = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE},
    consumes = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE}
)
@Tag(name = "Route", description = "Routes management")
@NullMarked
public class RouteController {
    private final RouteService service;

    public RouteController(RouteService service) {
        this.service = service;
    }

    @Operation(
        summary = "Get routes",
        description = "Get all routes or filter by freight ID/
vehicle ID. May result in empty list!"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Successfully retrieved routes",
        content = @Content(
            schema = @Schema(implementation = Route.class)
        )
    )
    @GetMapping(consumes = MediaType.ALL_VALUE)
    public List<Route> getAll(
        @Parameter(description = "Filter routes by freight ID")
        @RequestParam(name = "freightId", required = false)
        @Nullable Long freightId,
        @Parameter(description = "Filter routes by vehicle ID")
        @RequestParam(name = "vehicleId", required = false)
        @Nullable Long vehicleId) {
        if (vehicleId != null) {
            return service.getByVehicleId(vehicleId);
        }
        if (freightId != null) {
            var route = service.getByFreightId(freightId);
            if (route == null) {
                return List.of();
            }
            return List.of(route);
        }
        return service.getAll();
    }

    @Operation(
        summary = "Create new route",
        description = "Add new route"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Route successfully created",
        content = @Content(
            schema = @Schema(implementation = Route.class)
        )
    )

```

```

        )
    )
    @ApiResponse(
        responseCode = "400",
        description = "Invalid route data provided"
    )
    @PostMapping
    public ResponseEntity<Route> add(
        @io.swagger.v3.oas.annotations.parameters.RequestBody(
            description = "Route object to be created",
            required = true,
            content = @Content(
                schema = @Schema(implementation =
Route.class)
            )
        )
        @RequestBody Route newRoute) {
        return ResponseEntity.ok(service.add(newRoute));
    }

    @Operation(
        summary = "Update a route (full)",
        description = "Update all fields of an existing route by
ID"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Route successfully updated",
        content = @Content(
            schema = @Schema(implementation = Route.class)
        )
    )
    @ApiResponse(
        responseCode = "400",
        description = "Invalid route data provided"
    )
    @PutMapping("/{id}")
    public ResponseEntity<Route> update(
        @Parameter(description = "ID of the route to update",
required = true)
        @PathVariable("id") long id,
        @io.swagger.v3.oas.annotations.parameters.RequestBody(
            description = "Updated route object",
            required = true,
            content = @Content(
                schema = @Schema(implementation =
Route.class)
            )
        )
        @RequestBody Route newRoute) {
        return ResponseEntity.ok(service.update(id, newRoute));
    }

    @Operation(
        summary = "Update a route (partial)",
        description = "Update specific fields of an existing
route by ID"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Route successfully updated",
        content = @Content(

```



```

        schema = @Schema(implementation = Route.class)
    )
    )
    @ApiResponse(
        responseCode = "400",
        description = "Invalid route data provided"
    )
    @PatchMapping("/{id}")
    public ResponseEntity<Route> updatePatch(
        @Parameter(description = "ID of the route to update",
required = true)
        @PathVariable("id") long id,
        @io.swagger.v3.oas.annotations.parameters.RequestBody(
update",
            description = "Route object with fields to
            required = true,
            content = @Content(
                schema = @Schema(implementation =
Route.class)
            )
        )
        @RequestBody Route newRoute) {
        return ResponseEntity.ok(service.update(id, newRoute));
    }

    @Operation(
        summary = "Retrieve a route by ID",
        description = "Get a single route by its ID"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Route found",
        content = @Content(
            schema = @Schema(implementation = Route.class)
        )
    )
    @ApiResponse(
        responseCode = "404",
        description = "Route not found"
    )
    @GetMapping(value =("/{id}", consumes = MediaType.ALL_VALUE)
    public ResponseEntity<Route> getById(
        @Parameter(description = "ID of the route to retrieve",
required = true)
        @PathVariable("id") long id) {
        return ResponseEntity.ofNullable(service.getById(id));
    }

    @Operation(
        summary = "Delete a route",
        description = "Delete a route by ID"
    )
    @ApiResponse(
        responseCode = "200",
        description = "Route successfully deleted",
        content = @Content(schema = @Schema(implementation =
Route.class)
    )
    )
    @ApiResponse(
        responseCode = "404",
        description = "Route not found"
    )

```

```

    )
    @DeleteMapping(value =("/{id}", consumes = MediaType.ALL_VALUE)
    public ResponseEntity<Route> delete(
        @Parameter(description = "ID of the route to delete",
            required = true)
        @PathVariable("id") long id) {
        return ResponseEntity.ofNullable(service.delete(id));
    }
}

```

```
package ua.com.dxrkness.controller;
```

```

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.jspecify.annotations.NullMarked;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.com.dxrkness.model.Vehicle;
import ua.com.dxrkness.service.VehicleService;

```

```
import java.util.List;
```

```

@RestController
@RequestMapping(
    value = "/vehicles",
    produces = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE},
    consumes = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE}
)
@Tag(name = "Vehicle", description = "Vehicles management")
@NullMarked
public class VehicleController {
    private final VehicleService service;

    public VehicleController(VehicleService service) {
        this.service = service;
    }

    @Operation(
        summary = "Retrieve all vehicles",
        description = "Get a list of all vehicles"
    )
    @ApiResponse(responseCode = "200", description = "Successfully
        retrieved list of vehicles",
        content = @Content(schema = @Schema(implementation =
            Vehicle.class)))
    @GetMapping(consumes = MediaType.ALL_VALUE)
    public List<Vehicle> getAll() {
        return service.getAll();
    }

    @Operation(
        summary = "Retrieve a vehicle by ID",
        description = "Get a single vehicle by its ID"
    )

```

```

        @ApiResponse(responseCode = "200", description = "Vehicle found",
            content = @Content(schema = @Schema(implementation =
Vehicle.class)))
        @ApiResponse(responseCode = "404", description = "Vehicle not
found")
        @GetMapping(value =("/{id}", consumes = MediaType.ALL_VALUE)
        public ResponseEntity<Vehicle> getById(
            @Parameter(description = "ID of the vehicle to retrieve",
required = true)
                @PathVariable("id") long id) {
            return ResponseEntity.ofNullable(service.getById(id));
        }

        @Operation(
            summary = "Create a new vehicle",
            description = "Add a new vehicle to the system"
        )
        @ApiResponse(responseCode = "200", description = "Vehicle
successfully created",
            content = @Content(schema = @Schema(implementation =
Vehicle.class)))
        @ApiResponse(responseCode = "400", description = "Invalid vehicle
data provided")
        @PostMapping
        public ResponseEntity<Vehicle> add(
            @io.swagger.v3.oas.annotations.parameters.RequestBody(
                description = "Vehicle object to be created",
                required = true,
                content = @Content(schema =
@Schema(implementation = Vehicle.class)))
                @RequestBody Vehicle newVehicle) {
            return ResponseEntity.ok(service.add(newVehicle));
        }

        @Operation(
            summary = "Update a vehicle (full)",
            description = "Update all fields of an existing vehicle
by ID"
        )
        @ApiResponse(responseCode = "200", description = "Vehicle
successfully updated",
            content = @Content(schema = @Schema(implementation =
Vehicle.class)))
        @ApiResponse(responseCode = "400", description = "Invalid vehicle
data provided")
        @PutMapping("/{id}")
        public ResponseEntity<Vehicle> update(
            @Parameter(description = "ID of the vehicle to update",
required = true)
                @PathVariable("id") long id,
                @io.swagger.v3.oas.annotations.parameters.RequestBody(
                    description = "Updated vehicle object",
                    required = true,
                    content = @Content(schema =
@Schema(implementation = Vehicle.class)))
                    @RequestBody Vehicle newVehicle) {
            return ResponseEntity.ok(service.update(id, newVehicle));
        }

        @Operation(
            summary = "Update a vehicle (partial)",
            description = "Update specific fields of an existing

```

```

vehicle by ID"
    )
    @ApiResponse(responseCode = "200", description = "Vehicle
successfully updated",
        content = @Content(schema = @Schema(implementation =
Vehicle.class)))
    @ApiResponse(responseCode = "400", description = "Invalid vehicle
data provided")
    @PatchMapping("/{id}")
    public ResponseEntity<Vehicle> updatePatch(
        @Parameter(description = "ID of the vehicle to update",
required = true)
        @PathVariable("id") long id,
        @io.swagger.v3.oas.annotations.parameters.RequestBody(
description = "Vehicle object with fields to
update",
            required = true,
            content = @Content(schema =
@Schema(implementation = Vehicle.class)))
        @RequestBody Vehicle newVehicle) {
        return ResponseEntity.ok(service.update(id, newVehicle));
    }

    @Operation(
        summary = "Delete a vehicle",
        description = "Delete a vehicle by its ID"
    )
    @ApiResponse(responseCode = "200", description = "Vehicle
successfully deleted",
        content = @Content(schema = @Schema(implementation =
Vehicle.class)))
    @ApiResponse(responseCode = "404", description = "Vehicle not
found")
    @DeleteMapping(value =("/{id}", consumes = MediaType.ALL_VALUE)
    public ResponseEntity<Vehicle> delete(
        @Parameter(description = "ID of the vehicle to delete",
required = true)
        @PathVariable("id") long id) {
        return ResponseEntity.ofNullable(service.delete(id));
    }
}

package ua.com.dxrkness.model;

import tools.jackson.databind.PropertyNamingStrategies;
import tools.jackson.databind.annotation.JsonNaming;

@JsonNaming(PropertyNamingStrategies.SnakeCaseStrategy.class)
public record Freight(long id,
    String name,
    String description,
    int weightKg,
    Dimensions dimensions,
    Status status) implements Identifiable {
    public record Dimensions(int widthCm,
        int heightCm,
        int lengthCm) {

    }

    public enum Status {

```

```

        PENDING, IN_TRANSIT, DELIVERED
    }
}

package ua.com.dxrkness.model;

public interface Identifiable {
    long id();
}

package ua.com.dxrkness.model;

import tools.jackson.databind.PropertyNamingStrategies;
import tools.jackson.databind.annotation.JsonNaming;

import java.util.List;

@JsonNaming(PropertyNamingStrategies.SnakeCaseStrategy.class)
public record Route(long id,
                   long vehicleId,
                   List<Long> freightId,
                   String startLocation,
                   String endLocation,
                   Double distanceKm,
                   Double estimatedDurationHours,
                   Status status) implements Identifiable {
    public enum Status {
        PLANNED,
        IN_PROGRESS,
        COMPLETED,
        CANCELLED
    }
}

package ua.com.dxrkness.model;

import tools.jackson.databind.PropertyNamingStrategies;
import tools.jackson.databind.annotation.JsonNaming;

@JsonNaming(PropertyNamingStrategies.SnakeCaseStrategy.class)
public record Vehicle(long id,
                     String brand,
                     String model,
                     String licensePlate,
                     int year,
                     int capacityKg,
                     Status status) implements Identifiable {
    public enum Status {
        AVAILABLE,
        IN_TRANSIT,
        MAINTENANCE
    }
}

@NullMarked
package ua.com.dxrkness;

import org.jspecify.annotations.NullMarked;

```

```

package ua.com.dxrkness.repository;

import org.jspecify.annotations.Nullable;
import ua.com.dxrkness.model.Identifiable;

import java.util.ArrayList;
import java.util.List;

abstract class CrudRepository<T extends Identifiable> {
    protected final List<T> STORAGE = new ArrayList<>();

    public List<T> getAll() {
        return new ArrayList<>(STORAGE); // defensive copy
    }

    public @Nullable T getById(long id) {
        T found = null;
        for (var el : STORAGE) {
            if (el.id() == id) {
                found = el;
                break;
            }
        }
        return found;
    }

    public int add(T toAdd) {
        STORAGE.add(toAdd);
        return STORAGE.size();
    }

    public T update(long id, T newT) {
        for (int i = 0; i < STORAGE.size(); i++) {
            if (STORAGE.get(i).id() == id) {
                STORAGE.set(i, newT);
                break;
            }
        }
        return newT;
    }

    public @Nullable T delete(long id) {
        T deleted = null;
        for (var itr = STORAGE.iterator(); itr.hasNext(); ) {
            T next = itr.next();
            if (next.id() == id) {
                deleted = next;
                itr.remove();
                break;
            }
        }
        return deleted;
    }

    public void deleteAll() {
        STORAGE.clear();
    }
}

package ua.com.dxrkness.repository;

```

```

import org.springframework.stereotype.Repository;
import ua.com.dxrkness.model.Freight;

@Repository
public final class FreightRepository extends CrudRepository<Freight>
{
}

package ua.com.dxrkness.repository;

import org.jspecify.annotations.Nullable;
import org.springframework.stereotype.Repository;
import ua.com.dxrkness.model.Route;

import java.util.ArrayList;
import java.util.List;

@Repository
public final class RouteRepository extends CrudRepository<Route> {
    public @Nullable Route getByFreightId(long freightId) {
        Route found = null;
        out: for (var route : STORAGE) {
            var freights = route.freightId();
            for (var freight : freights) {
                if (freight == freightId) {
                    found = route;
                    break out;
                }
            }
        }
        return found;
    }

    public List<Route> getByVehicleId(long vehicleId) {
        List<Route> found = new ArrayList<>();
        for (var route : STORAGE) {
            if (route.vehicleId() == vehicleId) {
                found.add(route);
                break;
            }
        }
        return found;
    }
}

package ua.com.dxrkness.repository;

import org.springframework.stereotype.Repository;
import ua.com.dxrkness.model.Vehicle;

@Repository
public final class VehicleRepository extends CrudRepository<Vehicle>
{
}

package ua.com.dxrkness.service;

import org.jspecify.annotations.Nullable;
import org.springframework.stereotype.Service;
import ua.com.dxrkness.model.Freight;

```

```

import ua.com.dxrkness.repository.FreightRepository;

import java.util.List;

@Service
public final class FreightService {
    private final FreightRepository repo;

    public FreightService(FreightRepository repo) {
        this.repo = repo;
    }

    public List<Freight> getAll() {
        return repo.getAll();
    }

    public Freight add(Freight freight) {
        int id = repo.add(freight);
        return new Freight(id, freight.name(), freight.description(),
freight.weightKg(), freight.dimensions(), freight.status());
    }

    public @Nullable Freight getById(long id) {
        return repo.getById(id);
    }

    public Freight update(long id, Freight newFreight) {
        newFreight = new Freight(id, newFreight.name(),
newFreight.description(), newFreight.weightKg(),
newFreight.dimensions(), newFreight.status());
        return repo.update(id, newFreight);
    }

    public @Nullable Freight delete(long id) {
        return repo.delete(id);
    }
}

package ua.com.dxrkness.service;

import org.jspecify.annotations.NullMarked;
import org.springframework.stereotype.Service;
import ua.com.dxrkness.model.Freight;
import ua.com.dxrkness.model.Route;
import ua.com.dxrkness.model.Vehicle;
import ua.com.dxrkness.repository.FreightRepository;
import ua.com.dxrkness.repository.RouteRepository;
import ua.com.dxrkness.repository.VehicleRepository;

import java.util.ArrayList;
import java.util.List;

@Service
@NullMarked
public class PopulateService {
    private final FreightRepository freightRepository;
    private final RouteRepository routeRepository;
    private final VehicleRepository vehicleRepository;

    public PopulateService(FreightRepository freightRepository,
RouteRepository routeRepository, VehicleRepository vehicleRepository)

```



```

{
    this.freightRepository = freightRepository;
    this.routeRepository = routeRepository;
    this.vehicleRepository = vehicleRepository;
}

public void populate(int vals){
    for (int i = 0; i < vals; i++) {
        var f = new Freight(i, "", "", 0, null, null);
        freightRepository.add(f);
    }

    for (int i = 0; i < vals; i++) {
        var v = new Vehicle(i, "", "", "", 0, 0, null);
        vehicleRepository.add(v);
    }

    for (int i = 0; i < vals; i++)
        routeRepository.add(new Route(i, i, List.of((long)i), "",
"", 0., 0., null));
}

public void clear() {
    routeRepository.deleteAll();
    freightRepository.deleteAll();
    vehicleRepository.deleteAll();
}
}

package ua.com.dxrkness.service;

import org.jspecify.annotations.Nullable;
import org.springframework.stereotype.Service;
import ua.com.dxrkness.model.Route;
import ua.com.dxrkness.repository.RouteRepository;

import java.util.List;

@Service
public final class RouteService {
    private final RouteRepository repo;

    public RouteService(RouteRepository repo) {
        this.repo = repo;
    }

    public List<Route> getAll() {
        return repo.getAll();
    }

    public @Nullable Route getById(long id) {
        return repo.getById(id);
    }

    public Route add(Route route) {
        int id = repo.add(route);
        return new Route(id,
            route.vehicleId(),
            route.freightId(),
            route.startLocation(),
            route.endLocation(),

```

```

        route.distanceKm(),
        route.estimatedDurationHours(),
        route.status());
    }

    public @Nullable Route getByFreightId(long freightId) {
        return repo.getByFreightId(freightId);
    }

    public List<Route> getByVehicleId(long vehicleId) {
        return repo.getByVehicleId(vehicleId);
    }

    public Route update(long id, Route newRoute) {
        newRoute = new Route(id,
            newRoute.vehicleId(),
            newRoute.freightId(),
            newRoute.startLocation(),
            newRoute.endLocation(),
            newRoute.distanceKm(),
            newRoute.estimatedDurationHours(),
            newRoute.status());
        return repo.update(id, newRoute);
    }

    public @Nullable Route delete(long id) {
        return repo.delete(id);
    }
}

package ua.com.dxrkness.service;

import org.jspecify.annotations.Nullable;
import org.springframework.stereotype.Service;
import ua.com.dxrkness.model.Vehicle;
import ua.com.dxrkness.repository.VehicleRepository;

import java.util.List;

@Service
public final class VehicleService {
    private final VehicleRepository repo;

    public VehicleService(VehicleRepository repo) {
        this.repo = repo;
    }

    public List<Vehicle> getAll() {
        return repo.getAll();
    }

    public Vehicle add(Vehicle veh) {
        int id = repo.add(veh);
        return new Vehicle(id, veh.brand(), veh.model(),
            veh.licensePlate(), veh.year(), veh.capacityKg(), veh.status());
    }

    public @Nullable Vehicle getById(long id) {
        return repo.getById(id);
    }
}

```

```

    public Vehicle update(long id, Vehicle newVehicle) {
        newVehicle = new Vehicle(id,
            newVehicle.brand(),
            newVehicle.model(),
            newVehicle.licensePlate(),
            newVehicle.year(),
            newVehicle.capacityKg(),
            newVehicle.status());
        return repo.update(id, newVehicle);
    }

    public @Nullable Vehicle delete(long id) {
        return repo.delete(id);
    }
}

spring:
    application:
        name: itroi

package ua.com.dxrkness.integration;

import org.jspecify.annotations.NullMarked;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.client.RestTestClient;
import org.springframework.web.context.WebApplicationContext;
import ua.com.dxrkness.service.PopulateService;
import ua.com.dxrkness.model.Freight;

import java.util.List;
import java.util.stream.Stream;

import static org.assertj.core.api.BDDAssertions.then;

@SpringBootTest
@NullMarked
class FreightIntegrationTest {
    private static final int AMOUNT = 100;
    @Autowired
    private PopulateService populateService;

    private RestTestClient testClient;

    @BeforeEach
    void setUp(WebApplicationContext context) {
        populateService.populate(AMOUNT);
        testClient =
RestTestClient.bindToApplicationContext(context).baseUrl("/
freights").build();
    }

    @AfterEach

```

```

void teardown() {
    populateService.clear();
}

private static Stream<MediaType> mediaTypesClientConsumes() {
    return Stream.of(MediaType.APPLICATION_JSON,
        MediaType.APPLICATION_XML);
}

private static Stream<Arguments>
mediaTypesClientConsumesProduces() {
    return Stream.of(
        Arguments.of(MediaType.APPLICATION_JSON,
            MediaType.APPLICATION_JSON),
        Arguments.of(MediaType.APPLICATION_XML,
            MediaType.APPLICATION_XML),
        Arguments.of(MediaType.APPLICATION_JSON,
            MediaType.APPLICATION_XML),
        Arguments.of(MediaType.APPLICATION_XML,
            MediaType.APPLICATION_JSON)
    );
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getAll(MediaType mediaType) {
    // given

    // when
    // then
    testClient.get()
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(new
ParameterizedTypeReference<List<Freight>>() {
    })
        .consumeWith(res ->
then(res.getResponseBody()).hasSize(AMOUNT));
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getIdReturnsEntity(MediaType mediaType) {
    // given
    final var id = 1;

    // when
    // then
    testClient.get().uri("/{id}", id)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Freight.class)
        .value(val -> {
            then(val).isNotNull();
            then(val.id()).isEqualTo(id);
        });
}

```

```

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getByIdReturnsNotFound(MediaType mediaType) {
    // given
    final var id = -1;

    // when
    // then
    testClient.get().uri("/{id}", id)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void add(MediaType clientConsumes, MediaType clientProduces) {
    // given
    final var newEntity = new Freight(444, "123", "321", 123, new
Freight.Dimensions(123, 321, 444), Freight.Status.DELIVERED);

    // when
    // then
    testClient.post()
        .accept(clientConsumes)
        .contentType(clientProduces)
        .body(newEntity)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Freight.class)
        .value(val -> {
            then(val.id()).isEqualTo(AMOUNT+1);
            then(val).usingRecursiveComparison()
                .ignoringFields("id")
                .isEqualTo(newEntity);
        });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void update(MediaType clientConsumes, MediaType clientProduces) {
    // given
    final var existingEntityId = AMOUNT-1;
    final var newEntity = new Freight(444, "123", "321", 123, new
Freight.Dimensions(123, 321, 444), Freight.Status.DELIVERED);

    // when
    // then
    testClient.put().uri("/{id}", existingEntityId)
        .accept(clientConsumes)
        .contentType(clientProduces)
        .body(newEntity)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Freight.class)
        .value(val -> {
            then(val).isNotNull();
            then(val.id()).isEqualTo(existingEntityId);
            then(val).usingRecursiveComparison()

```

```

        .ignoringFields("id")
        .isEqualTo(newEntity);
    });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void updatePatch(MediaType clientConsumes, MediaType
clientProduces) {
    // given
    final var existingEntityId = AMOUNT-1;
    final var newEntity = new Freight(439, "123", "321", 123, new
Freight.Dimensions(123, 321, 444), Freight.Status.DELIVERED);

    // when
    // then
    testClient.patch().uri("/{id}", existingEntityId)
        .accept(clientConsumes)
        .contentType(clientProduces)
        .body(newEntity)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Freight.class)
        .value(val -> {
            then(val).isNotNull();
            then(val.id()).isEqualTo(existingEntityId);
            then(val).usingRecursiveComparison()
                .ignoringFields("id")
                .isEqualTo(newEntity);
        });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void delete(MediaType mediaType) {
    // given
    final var existingEntityId = AMOUNT-1;

    // when
    // then
    testClient.delete().uri("/{id}", existingEntityId)
        .accept(mediaType)
        .exchange()
        .expectBody(Freight.class)
        .value(val -> {
            then(val).isNotNull();
            then(val.id()).isEqualTo(existingEntityId);
        });

    testClient.get().uri("/{id}", existingEntityId)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void deleteIfEntityNotFound(MediaType mediaType) {
    // given
    final var existingEntityId = AMOUNT+1;

```

```

        // when
        // then
        testClient.delete().uri("/{id}", existingEntityId)
            .accept(mediaType)
            .exchange()
            .expectStatus()
            .isNotFound();
    }
}

package ua.com.dxrkness.integration;

import org.jspecify.annotations.NullMarked;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.client.RestTestClient;
import org.springframework.web.context.WebApplicationContext;
import ua.com.dxrkness.model.Freight;
import ua.com.dxrkness.model.Route;
import ua.com.dxrkness.model.Vehicle;
import ua.com.dxrkness.service.PopulateService;

import java.util.List;
import java.util.stream.Stream;

import static org.assertj.core.api.BDDAssertions.then;

@SpringBootTest
@NullMarked
class RouteIntegrationTest {
    private static final int AMOUNT = 100;
    @Autowired
    private PopulateService populateService;

    private RestTestClient testClient;

    @BeforeEach
    void setUp(WebApplicationContext context) {
        populateService.populate(AMOUNT);
        testClient =
RestTestClient.bindToApplicationContext(context).baseUrl("/
routes").build();
    }

    @AfterEach
    void teardown() {
        populateService.clear();
    }

    private static Stream<MediaType> mediaTypesClientConsumes() {
        return Stream.of(MediaType.APPLICATION_JSON,
MediaType.APPLICATION_XML);
    }
}

```

```

        private static Stream<Arguments>
mediaTypesClientConsumesProduces() {
    return Stream.of(
        Arguments.of(MediaType.APPLICATION_JSON,
MediaType.APPLICATION_JSON),
        Arguments.of(MediaType.APPLICATION_XML,
MediaType.APPLICATION_XML),
        Arguments.of(MediaType.APPLICATION_JSON,
MediaType.APPLICATION_XML),
        Arguments.of(MediaType.APPLICATION_XML,
MediaType.APPLICATION_JSON)
    );
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getAll(MediaType mediaType) {
    // given

    // when
    // then
    testClient.get()
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(new
ParameterizedTypeReference<List<Route>>() {
    })
        .consumeWith(res ->
then(res.getResponseBody()).hasSize(AMOUNT));
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getByVehicleIdReturnsEntity(MediaType mediaType) {
    // given
    int vehicleId = 1;

    // when
    // then
    testClient.get().uri(b -> b.queryParam("vehicleId",
vehicleId).build())
        .accept(mediaType)
        .exchangeSuccessfully()
        .expectStatus()
        .isOk()
        .expectBody(new
ParameterizedTypeReference<List<Route>>() {
    })
        .value(val -> then(val).isNotNull()
            .hasSize(1)
            .allSatisfy(route ->
then(route.vehicleId()).isEqualTo(vehicleId)));
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getByVehicleIdReturns404(MediaType mediaType) {
    // given
    int vehicleId = AMOUNT + 1;

```



```

        // when
        // then
        testClient.get().uri(b -> b.queryParam("vehicleId",
vehicleId).build())
            .accept(mediaType)
            .exchangeSuccessfully()
            .expectStatus()
            .isOk()
            .expectBody(List.class)
            .isEqualTo(List.of());
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumes")
    void getByFreightIdReturnsEntity(MediaType mediaType) {
        // given
        int freightId = 1;

        // when
        // then
        testClient.get().uri(b -> b.queryParam("freightId",
freightId).build())
            .accept(mediaType)
            .exchangeSuccessfully()
            .expectStatus()
            .isOk()
            .expectBody(new
ParameterizedTypeReference<List<Route>>() {
            })
            .value(val -> then(val).isNotNull()
                .hasSize(1)
                .allSatisfy(route -> then(route.freightId())
                    .isNotNull()
                    .isNotEmpty()
                    .anySatisfy(fre ->
then(fre).isEqualTo(freightId)))));
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumes")
    void getByFreightIdReturns404(MediaType mediaType) {
        // given
        int freightId = AMOUNT + 1;

        // when
        // then
        testClient.get().uri(b -> b.queryParam("freightId",
freightId).build())
            .accept(mediaType)
            .exchangeSuccessfully()
            .expectStatus()
            .isOk()
            .expectBody(List.class)
            .isEqualTo(List.of());
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumes")
    void getIdReturnsEntity(MediaType mediaType) {
        // given
        final var id = 1;

```

```

// when
// then
testClient.get().uri("/{id}", id)
    .accept(mediaType)
    .exchange()
    .expectStatus()
    .isOk()
    .expectBody(Route.class)
    .value(val -> {
        then(val).isNotNull();
        then(val.id()).isEqualTo(id);
    });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getIdReturnsNotFound(MediaType mediaType) {
    // given
    final var id = -1;

    // when
    // then
    testClient.get().uri("/{id}", id)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void add(MediaType clientConsumes, MediaType clientProduces) {
    // given
    final var newEntity = new Route(444, 1, List.of(1L), "123",
    "321", 1., 2., Route.Status.IN_PROGRESS);

    // when
    // then
    testClient.post()
        .accept(clientConsumes)
        .contentType(clientProduces)
        .body(newEntity)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Route.class)
        .value(val -> {
            then(val.id()).isEqualTo(AMOUNT+1);
            then(val).usingRecursiveComparison()
                .ignoringFields("id")
                .isEqualTo(newEntity);
        });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void update(MediaType clientConsumes, MediaType clientProduces) {
    // given
    final var existingEntityId = AMOUNT - 1;
    final var newEntity = new Route(432, 1, List.of(1L), "123",
    "321", 22., 44., Route.Status.PLANNED);

```

```

// when
// then
testClient.put().uri("/{id}", existingEntityId)
    .accept(clientConsumes)
    .contentType(clientProduces)
    .body(newEntity)
    .exchange()
    .expectStatus()
    .isOk()
    .expectBody(Route.class)
    .value(val -> {
        then(val.id()).isEqualTo(existingEntityId);
        then(val).usingRecursiveComparison()
            .ignoringFields("id")
            .isEqualTo(newEntity);
    });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void updatePatch(MediaType clientConsumes, MediaType
clientProduces) {
    // given
    final var existingEntityId = AMOUNT - 1;
    final var newEntity = new Route(432, 1, List.of(1L), "123",
"321", 22., 44., Route.Status.CANCELLED);

    // when
    // then
testClient.patch().uri("/{id}", existingEntityId)
    .accept(clientConsumes)
    .contentType(clientProduces)
    .body(newEntity)
    .exchange()
    .expectStatus()
    .isOk()
    .expectBody(Route.class)
    .value(val -> {
        then(val).isNotNull();
        then(val.id()).isEqualTo(existingEntityId);
        then(val).usingRecursiveComparison()
            .ignoringFields("id")
            .isEqualTo(newEntity);
    });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void delete(MediaType mediaType) {
    // given
    final var existingEntityId = AMOUNT - 1;

    // when
    // then
testClient.delete().uri("/{id}", existingEntityId)
    .accept(mediaType)
    .exchange()
    .expectBody(Route.class)
    .value(val -> {
        then(val).isNotNull();
        then(val.id()).isEqualTo(existingEntityId);
    });
}

```

```

    });

    testClient.get().uri("/{id}", existingEntityId)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void deleteIfEntityNotFound(MediaType mediaType) {
    // given
    final var existingEntityId = AMOUNT + 1;

    // when
    // then
    testClient.delete().uri("/{id}", existingEntityId)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}
}

package ua.com.dxrkness.integration;

import org.jspecify.annotations.NullMarked;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.client.RestTestClient;
import org.springframework.web.context.WebApplicationContext;
import ua.com.dxrkness.model.Vehicle;
import ua.com.dxrkness.service.PopulateService;

import java.util.List;
import java.util.stream.Stream;

import static org.assertj.core.api.BDDAssertions.then;

@SpringBootTest
@NullMarked
class VehicleIntegrationTest {
    private static final int AMOUNT = 100;
    @Autowired
    private PopulateService populateService;

    private RestTestClient testClient;

    @BeforeEach
    void setUp(WebApplicationContext context) {
        populateService.populate(AMOUNT);
        testClient =
RestTestClient.bindToApplicationContext(context).baseUrl("/")

```

```

vehicles").build();
    }

    @AfterEach
    void teardown() {
        populateService.clear();
    }

    private static Stream<MediaType> mediaTypesClientConsumes() {
        return Stream.of(MediaType.APPLICATION_JSON,
            MediaType.APPLICATION_XML);
    }

    private static Stream<Arguments>
    mediaTypesClientConsumesProduces() {
        return Stream.of(
            Arguments.of(MediaType.APPLICATION_JSON,
                MediaType.APPLICATION_JSON),
            Arguments.of(MediaType.APPLICATION_XML,
                MediaType.APPLICATION_XML),
            Arguments.of(MediaType.APPLICATION_JSON,
                MediaType.APPLICATION_XML),
            Arguments.of(MediaType.APPLICATION_XML,
                MediaType.APPLICATION_JSON)
        );
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumes")
    void getAll(MediaType mediaType) {
        // given

        // when
        // then
        testClient.get()
            .accept(mediaType)
            .exchange()
            .expectStatus()
            .isOk()
            .expectBody(new
                ParameterizedTypeReference<List<Vehicle>>() {
                })
            .consumeWith(res ->
                then(res.getResponseBody()).hasSize(AMOUNT));
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumes")
    void getByIdReturnsEntity(MediaType mediaType) {
        // given
        final var id = 1;

        // when
        // then
        testClient.get().uri("/{id}", id)
            .accept(mediaType)
            .exchange()
            .expectStatus()
            .isOk()
            .expectBody(Vehicle.class)
            .value(val -> {
                then(val).isNotNull();
            });
    }

```

```

        then(val.id()).isEqualTo(id);
    });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumes")
void getByIdReturnsNotFound(MediaType mediaType) {
    // given
    final var id = -1;

    // when
    // then
    testClient.get().uri("/{id}", id)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void add(MediaType clientConsumes, MediaType clientProduces) {
    // given
    final var newEntity = new Vehicle(444, "123", "321",
"123321", 2001, 15000, Vehicle.Status.MAINTENANCE);

    // when
    // then
    testClient.post()
        .accept(clientConsumes)
        .contentType(clientProduces)
        .body(newEntity)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Vehicle.class)
        .value(val -> {
            then(val.id()).isEqualTo(AMOUNT+1);
            then(val).usingRecursiveComparison()
                .ignoringFields("id")
                .isEqualTo(newEntity);
        });
}

@ParameterizedTest
@MethodSource("mediaTypesClientConsumesProduces")
void update(MediaType clientConsumes, MediaType clientProduces) {
    // given
    final var existingEntityId = AMOUNT-1;
    final var newEntity = new Vehicle(444, "123", "321",
"123321", 2001, 15000, Vehicle.Status.MAINTENANCE);

    // when
    // then
    testClient.put().uri("/{id}", existingEntityId)
        .accept(clientConsumes)
        .contentType(clientProduces)
        .body(newEntity)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(Vehicle.class)

```

```

        .value(val -> {
            then(val).isNotNull();
            then(val.id()).isEqualTo(existingEntityId);
            then(val).usingRecursiveComparison()
                .ignoringFields("id")
                .isEqualTo(newEntity);
        });
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumesProduces")
    void updatePatch(MediaType clientConsumes, MediaType
clientProduces) {
        // given
        final var existingEntityId = AMOUNT-1;
        final var newEntity = new Vehicle(444, "123", "321",
"123321", 2001, 15000, Vehicle.Status.MAINTENANCE);

        // when
        // then
        testClient.patch().uri("/{id}", existingEntityId)
            .accept(clientConsumes)
            .contentType(clientProduces)
            .body(newEntity)
            .exchange()
            .expectStatus()
            .isOk()
            .expectBody(Vehicle.class)
            .value(val -> {
                then(val).isNotNull();
                then(val.id()).isEqualTo(existingEntityId);
                then(val).usingRecursiveComparison()
                    .ignoringFields("id")
                    .isEqualTo(newEntity);
            });
    }

    @ParameterizedTest
    @MethodSource("mediaTypesClientConsumes")
    void delete(MediaType mediaType) {
        // given
        final var existingEntityId = AMOUNT-1;

        // when
        // then
        testClient.delete().uri("/{id}", existingEntityId)
            .accept(mediaType)
            .exchange()
            .expectBody(Vehicle.class)
            .value(val -> {
                then(val).isNotNull();
                then(val.id()).isEqualTo(existingEntityId);
            });

        testClient.get().uri("/{id}", existingEntityId)
            .accept(mediaType)
            .exchange()
            .expectStatus()
            .isNotFound();
    }

    @ParameterizedTest

```

```
@MethodSource("mediaTypesClientConsumes")
void deleteIfEntityNotFound(MediaType mediaType) {
    // given
    final var existingEntityId = AMOUNT+1;

    // when
    // then
    testClient.delete().uri("/{id}", existingEntityId)
        .accept(mediaType)
        .exchange()
        .expectStatus()
        .isNotFound();
}
```

## 2.3 Висновки