Groapa Mihai, 343C5

Aplicatie Web – Gestiunea unei firme de transporturi

1. Descrierea temei

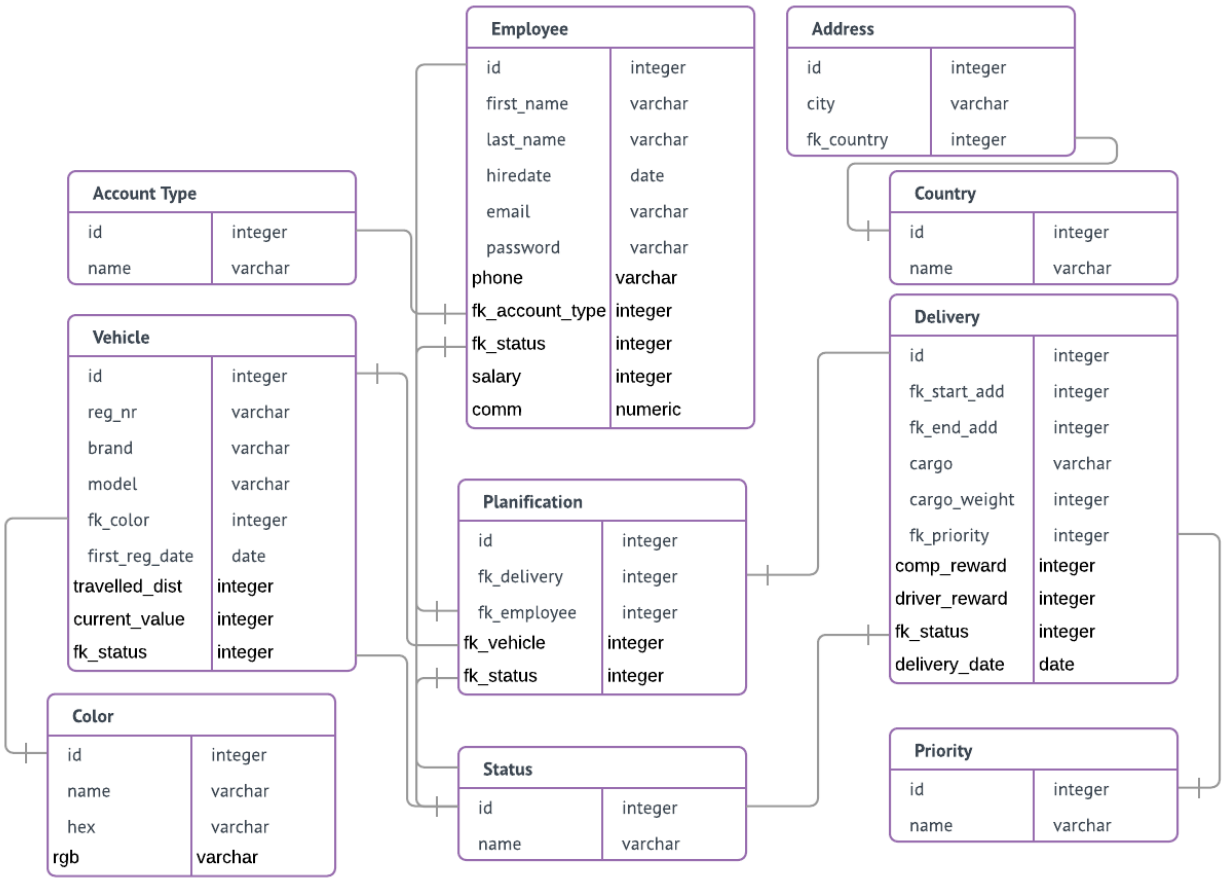
Sistemul permite gestiunea vehiculelor, a angajatilor si a livrarilor care trebuiesc efectuate. In sistem exista doua roluri de utilizator:

- **Coordinator**: inregistreaza vehicule, adauga angajati (le creeaza profil), adauga curse (specificand sursa, destinatia, continutul cargo-ului, valoarea cuvenita soferului ce va realiza transportul si altele), are posibilitatea de a realiza distribuirea curselor catre angajati si asignarea vehiculelor destinate transportului. O ultima actiune specifica coordonatorului este modificarea venitului soferilor, pe baza unor specificatii legate de performantele avute de acestia intr.o anumita perioada.
- **Driver**: soferul, dupa autentificare are posibilitatea de a accesa un istoric al livrarilor efectuate de el, precum si vizualizarea primei curse viitoare, asignata de unul dintre coordonatori lui. In momentul in care termina

livrarea, soferul marcheaza acest lucru in portal, el si camionul cu care a efectuat transportul devenind disponibili unor noi curse.

2. Baza de date

Pentru baza de date am folosit PgAdmin4 cu PostgreSQL.



Pentru toate operatiile executate am folosit functii stocate in baza, inclusive pentru operatiile de tip CRUD simple.

```
DECLARE
    employee_row employee%rowtype;
    deliveries_number_for_employee BIGINT;
    company_revenue_from_employee BIGINT;
BEGIN
    FOR employee_row IN SELECT * FROM get_all_employees()
    LOOP
        deliveries_number_for_employee := get_deliveries_nr_between_dates_for_employee(start_date, end_date,
        company_revenue_from_employee := get_company_revenue_between_dates_from_employee(start_date, end_dat

        IF operator_v = 'AND' THEN
            IF deliveries_number_for_employee >= min_deliveries AND company_revenue_from_employee >= min_com
                RETURN NEXT employee_row;
            END IF;
        ELSIF operator_v = 'OR' THEN
            IF deliveries_number_for_employee >= min_deliveries OR company_revenue_from_employee >= min_comp
                RETURN NEXT employee_row;
            END IF;
        END IF;
    END LOOP;
    RETURN;
END
```

Functie care intoarce lista angajatilor ce intr.un interval incadrat de doua date [d1.m1.y1, d2.m2.y2] au realizat un minim de ‘min_deliveries’ curse si/sau (in functie de caz), au adus in firma un venit minim de ‘min_company_revenue’ in aceasta perioada.

```
DECLARE
    employee_row employee%rowtype;
    new_salary INTEGER;
    new_comm NUMERIC;
BEGIN
    FOR employee_row IN SELECT * FROM get_employees_eligible_for_revenue_inc(min_deliveries, operator_v, min
    LOOP
        IF salary_increase_type = 'Fixed Value' THEN
            new_salary := employee_row.salary + salary_increase;
        ELSIF salary_increase_type = 'Percentage' THEN
            new_salary := employee_row.salary * salary_increase;
        END IF;
```

```
        IF comm_increase_type = 'Fixed Value' THEN
            new_comm := employee_row.comm + comm_increase;
        ELSIF comm_increase_type = 'Percentage' THEN
            new_comm := employee_row.comm * comm_increase;
        END IF;

        employee_row.salary := new_salary;
        employee_row.comm := new_comm;

        UPDATE employee
        SET salary = new_salary, comm = new_comm
        WHERE id = employee_row.id;

        RETURN NEXT employee_row;
    END LOOP;
    RETURN;
END
```

Procedura care aplica o crestere a salariului si a comisionului (procentuala sau cu valoare fixata) pentru angajatii intorsi de functia precedenta.

```
BEGIN
    IF (TG_OP='INSERT') THEN
        INSERT INTO planification_audits
        VALUES((select nextval('seq_planification_audits')),
    END IF;

    RETURN NEW;
END;

```

Trigger declansat la inserarea in baza a unei planificari, ce salveaza intr.o tabela suplimentara numita

‘planification_audits’ id.ul planificarii introduce si timestamp-ul current.

Constrangeri

O planificare este compusa din: un vehicul cu care se realizeaza transportul, o entitate delivery, ce la randul ei are o sursa si o destinatie (fk catre tabela address), un status al livrarii (Available, In Transit, Delivered, valori preluate prin fk catre tabela status) si legatura catre soferul ce face deplasarea (fk_employee). In afara de acestea, fiecare tabela are constrangere de tip cheie primara, prin campul id.

3. Descrierea aplicatiei

Aplicatia urmareste modelul Model-View-Controller. Partea de Backend este separata de modulul pentru Frontend.

In Backend, este urmat fluxul:

RestController -> Service -> Dao -> Entity

Asadar, pentru fiecare tip de entitate din aplicatie am 4 clase.

RestController-ul face maparea pe url-uri. Exemplu:

```java
@CrossOrigin(allowedHeaders="*",allowCredentials="true")
@PostMapping(value = "/employee/driversRevenueIncrease")
@ResponseBody
public List<Employee> increaseRevenues(@RequestBody RevenueIncrease jsonRevenueIncrease) {
    return employeeService.increaseRevenues(jsonRevenueIncrease);
}
```

Service-ul are rol de intermediar intre controller si Dao:

```java
@Override
public List<Employee> increaseRevenues(RevenueIncrease revenueIncrease) {
    return employeeDao.increaseRevenues(revenueIncrease);
}
```

In componenta Dao fac apelul catre procedurile/functiile stocate:

```java
public List<Employee> increaseRevenues(RevenueIncrease revenueIncrease) {
    CallableStatement callableStatement;
    ResultSet resultSet = null;

    try {
        callableStatement = connection.prepareCall( sql: "{call apply_revenue_inc(?,?,?,?,?,?,?,?,?)}");
        callableStatement.setBigDecimal( parameterIndex: 1, revenueIncrease.getSalaryIncrease());
        callableStatement.setString( parameterIndex: 2, revenueIncrease.getSalaryIncreaseType());
        callableStatement.setBigDecimal( parameterIndex: 3, revenueIncrease.getCommIncrease());
        callableStatement.setString( parameterIndex: 4, revenueIncrease.getCommIncreaseType());
        callableStatement.setInt( parameterIndex: 5, revenueIncrease.getMinDeliveries());
        callableStatement.setString( parameterIndex: 6, revenueIncrease.getOperator());
        callableStatement.setInt( parameterIndex: 7, revenueIncrease.getMinCompanyRevenueFromDriver());
        callableStatement.setDate( parameterIndex: 8, revenueIncrease.getStartDate());
        callableStatement.setDate( parameterIndex: 9, revenueIncrease.getEndDate());
        callableStatement.executeUpdate();
        resultSet = callableStatement.getResultSet();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return extractEmployeesFromResultSet(resultSet);
}
```

In Frontend, se urmeaza tiparul:

View -> Controller -> Service[s]

Asadar, fiecare ecran are asociat un controller angular in care, prin dependency injection accesez serviciile. Acestea din urma sunt responsabile cu requesturile de tip GET, POST, PUT catre server.

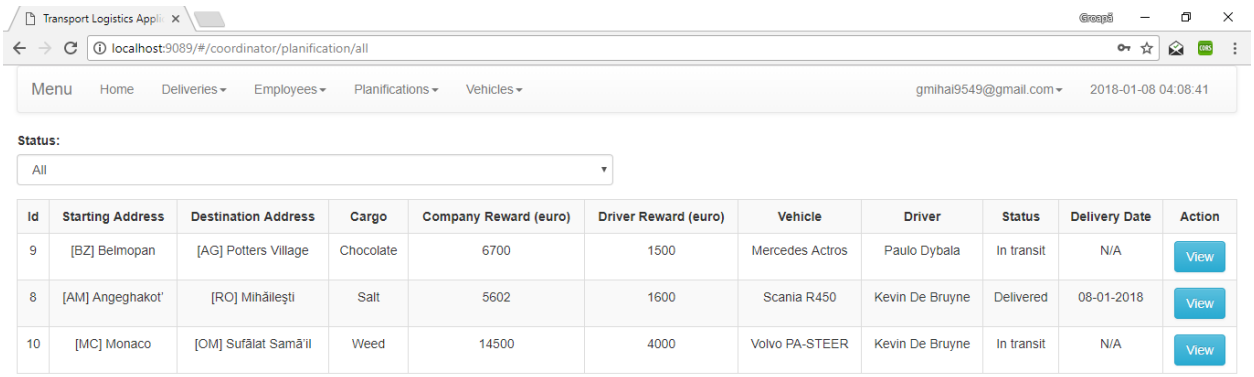
Conexiunea la baza de date

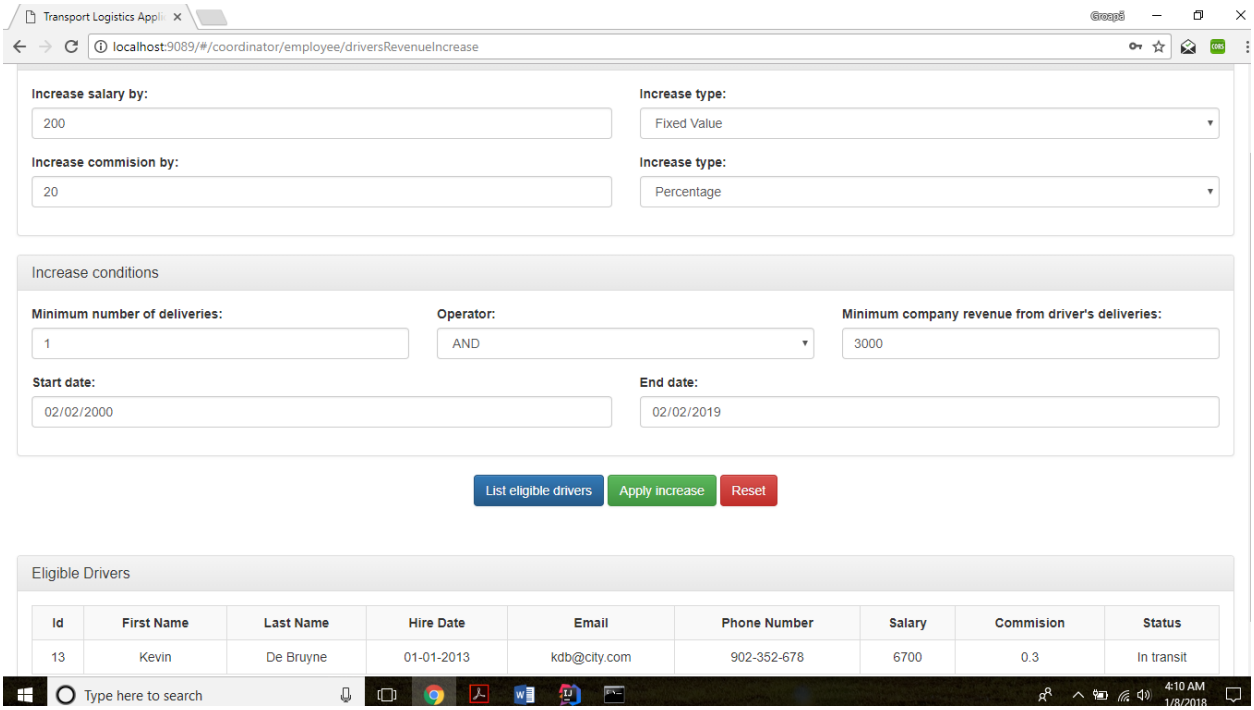
Am folosit Spring, astfel ca pentru conexiune a fost necesar sa declar dependenta pentru postgres in pom.xml, in fisierul application.properties sa declar parametrii bazei de date si nu in ultimul rand, in clasa de configurare a proiectului sa declar un bean de tip DataSource, din care se poate extrage obiectul connection.

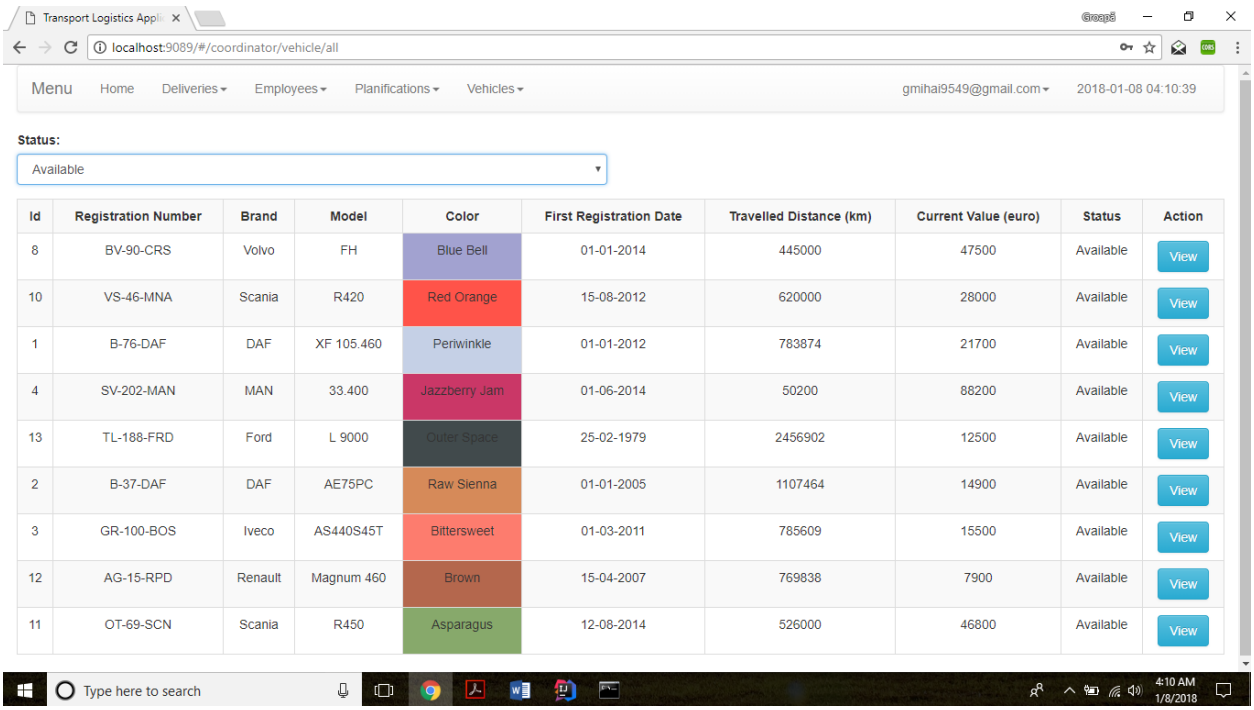
```java
@Configuration
@EnableAutoConfiguration
@EnableTransactionManagement
@ComponentScan(basePackages = {"ro.transport.demo.controllers"})
@EntityScan(value = "ro.transport.demo.domain")
public class ProjectConfiguration {

    private static final String PACKAGES_TO_SCAN = "ro.transport.demo.domain";
    private static final String USERNAME_DATABASE = "postgres";
    private static final String PASSWORD_DATABASE = "postgres";
    private static final String URL_DATABASE = "jdbc:postgresql://localhost:5432/Transport_Logistics_DB";
    private static final String ENTITY_MANAGER_DIALECT = "org.hibernate.dialect.PostgreSQLDialect";

    @Bean
    public DataSource dataSource() {
        SimpleDriverDataSource dataSource = new SimpleDriverDataSource();
        dataSource.setDriverClass(org.postgresql.Driver.class);
        dataSource.setUsername(USERNAME_DATABASE);
        dataSource.setPassword(PASSWORD_DATABASE);
        dataSource.setUrl(URL_DATABASE);

        return dataSource;
    }
```

```xml
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
```

```
spring.datasource.url=jdbc:postgresql://localhost:5432/Transport_Logistics_DB
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.jpa.generate-ddl=false
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.id.new_generator_mappings=false
liquibase.change-log=classpath:/liquibase/demo.xml
server.port=9090
```







Concluzii

- Proiect foarte interesant, am invatat multe lucruri despre baze de date, relatii intre tabele (de exemplu, e foarte important sa stabilesti de la inceput ce coloane sunt necesare pentru o entitate, daca o faci ulterior modifici mult cod).
- Foarte mult de lucru, mai ales ca a trebuit sa scriu proceduri/functii pentru fiecare operatie cu baza (in final au iesit undeva in jur de 45).

Bibliografie: niciuna, am aplicat cunostinte acumulate la locul de munca, cam asta e modul de lucru acolo, mai putin partea de procedure stocate, folosim JPA.