LAB WEEK 2

Back-end

Một công ty cần lập một **API** cho trang web bán hàng của họ. Thông tin cần lưu trữ bao gồm:

- employee (**emp_id**, full_name, dob, email, phone, address, *status*)
- product (**product_id**, name, description, unit, manufacturer_name, *status*)
- customer (**cust_id**, cust_name, email, phone, address)
- product_image (product_id**, image_id**, path, alternative)
- orders (**order_id**, order_date, emp_id, cust_id)
- order_detail **(order_id, product_id**, quantity, **price**, note)
- product_price (**product_id, price_date_time**, price, note)

Diễn giải:

- Một product có nhiều image, một image thuộc về một product. Status chỉ trạng thái kinh doanh của sản phẩm: 1- đang kinh doanh, 0 - tạm ngưng, -1 - không kinh doanh nữa.
- Employee có status: 1- đang làm việc, 0 - tạm nghỉ, -1 – nghỉ việc.
- Một order có nhiều order_detail, một order_detail thuộc về một order.
- Một order thuộc về một employee, một employeecó nhiều order.
- Một customer có nhiều order, một order chỉ thuộc một customer.
- Một product_detail có một giá (price) được lưu trong product_price. Một giá được xác định bằng product_id và price_date_time. Tại thời điểm lập order, giá được lấy với price_date_time gần nhất.

Lập REST API cho các thao tác cần thiết của yêu cầu này (dùng JakartaEE):

- Các thao tác CRUD cho các đối tượng, lập order. (dùng JPA).
- Thống kê order theo ngày, theo khoảng thời gian.
- Thống kê order theo nhân viên bán hàng trong 1 khoảng thời gian.
- …

Font-end

- Tạo các trang web cho việc hiển thị sản phẩm, chọn vào giỏ hàng và thanh toán (giả lập việc thanh toán qua thẻ nếu có thể)
- Vẽ biểu đồ giá theo thời gian
- **** và các chức năng khác tự nghĩ ra.*

HƯỚNG DẪN

Kiến trúc ứng dụng

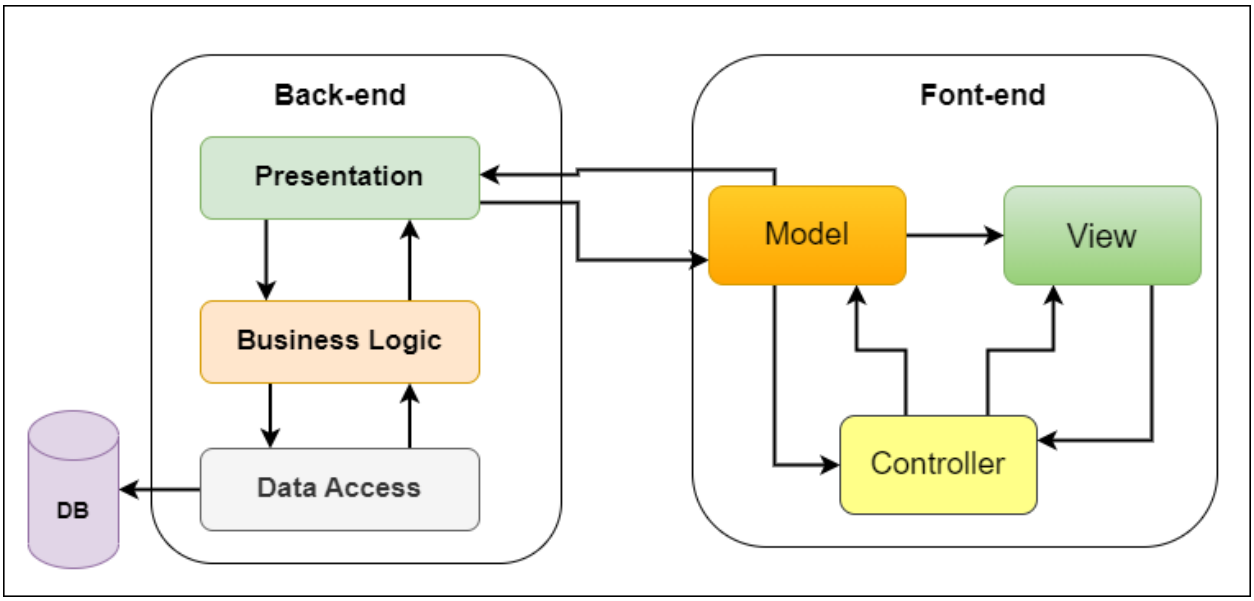

Figure 1: Application architecture

Back-end

Thiết kế theo kiến trúc 3-tiers. Trong đó:

- Tầng Data Access nhận nhiệm vụ truy xuất cơ sở dữ liệu.
- Tầng Business Logic là nơi xử lý nghiệp vụ của ứng dụng.
- Tầng Presentation đảm bảo làm giao diện cho phần back-end.

Cấu trúc project

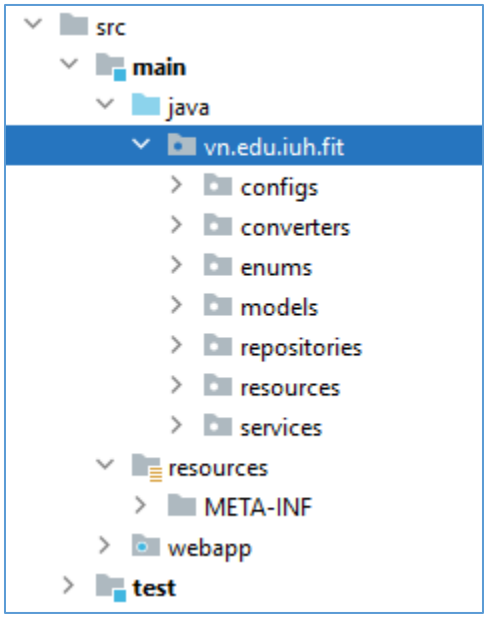

Figure 2: Project structure

Các packages được ánh xạ thành các tầng của kiến trúc như sau:

- Gói **repositories** chứa các lớp của tầng Data Access.
- Gói **services** chứa các lớp của tầng Business Logic.
- Gói **resources** chứa các lớp của tầng Presentation (sử dụng REST API).

Các gói phụ trợ khác gồm có:

- Gói **models** là nơi chứa các entities. Các lớp này nhận nhiệm vụ ánh xạ các bảng trong cơ sở dữ liệu đồng thời cũng là các đối tượng mang dữ liệu trao đổi giữa các tầng.
- Gói **configs** chứa các lớp dùng cho việc cấu hình ứng dụng.
- Gói **enums** chứa các tập hằng số cho các đối tượng. Trong ví dụ này, đó là hai tập hằng số chỉ các trạng thái của nhân viên (employee) và sản phẩm (product).
- Gói **converters** để chứa các lớp chuyển đổi một số kiểu dữ liệu đặc biệt sang json/xml/text.

Entities design

Thiết kế các entities ánh xạ các bảng dữ liệu và thiết lập các mối quan hệ

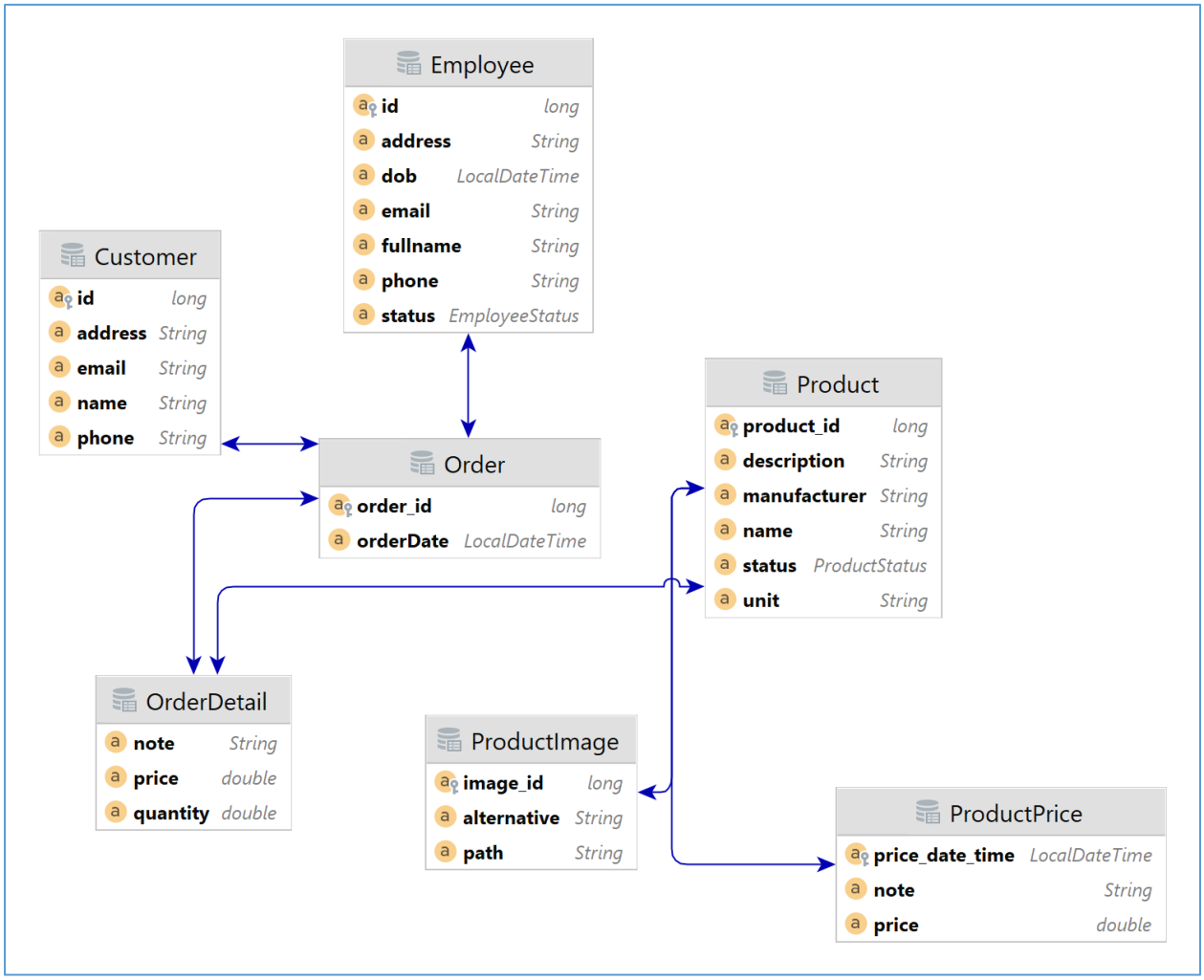

Figure 3: Persistence Unit view of entities

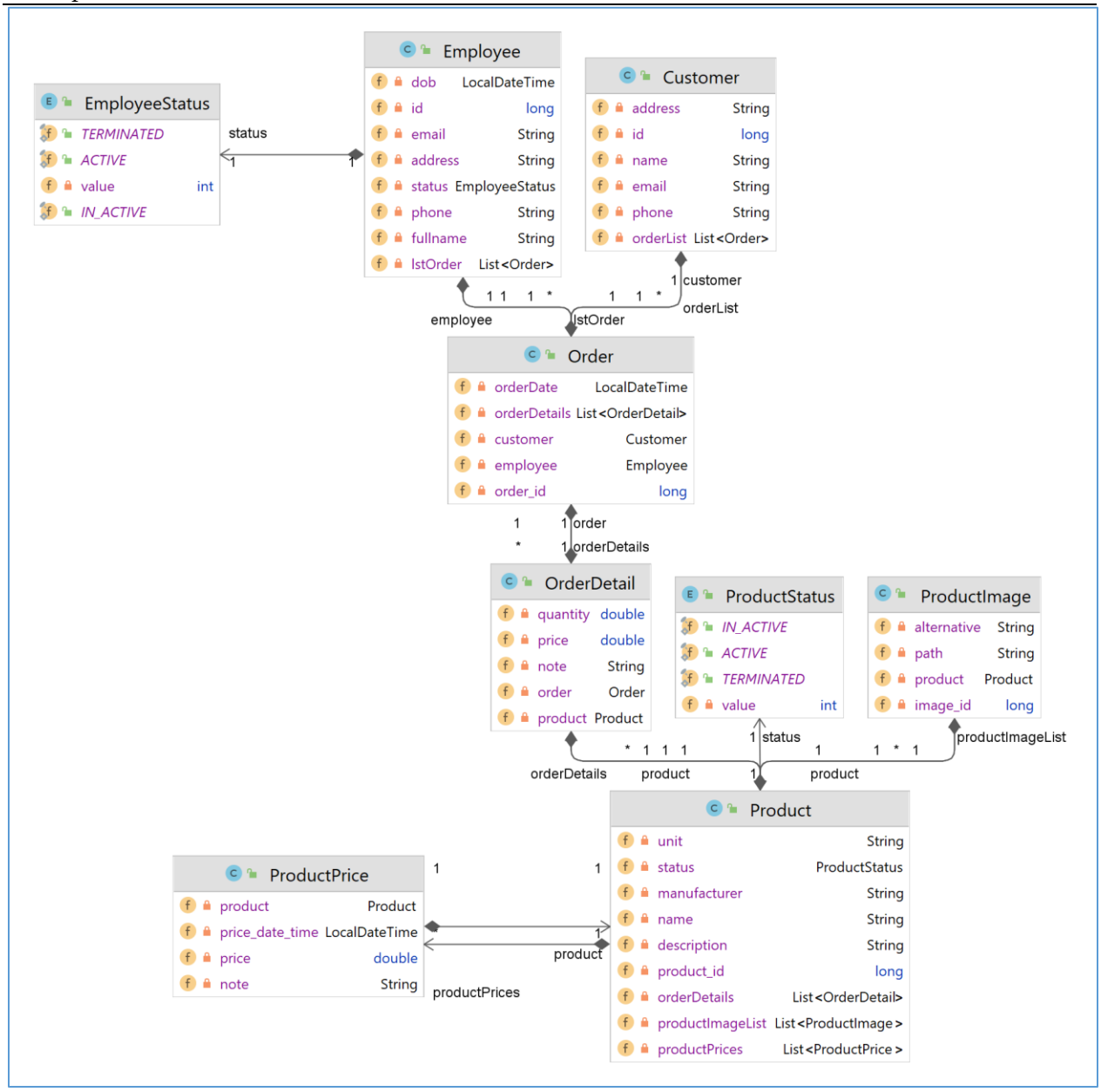

Figure 4: Class diagram of entities

Sau khi thiết kế và thực thi, database sẽ có Entity Relationship diagram như hình sau:

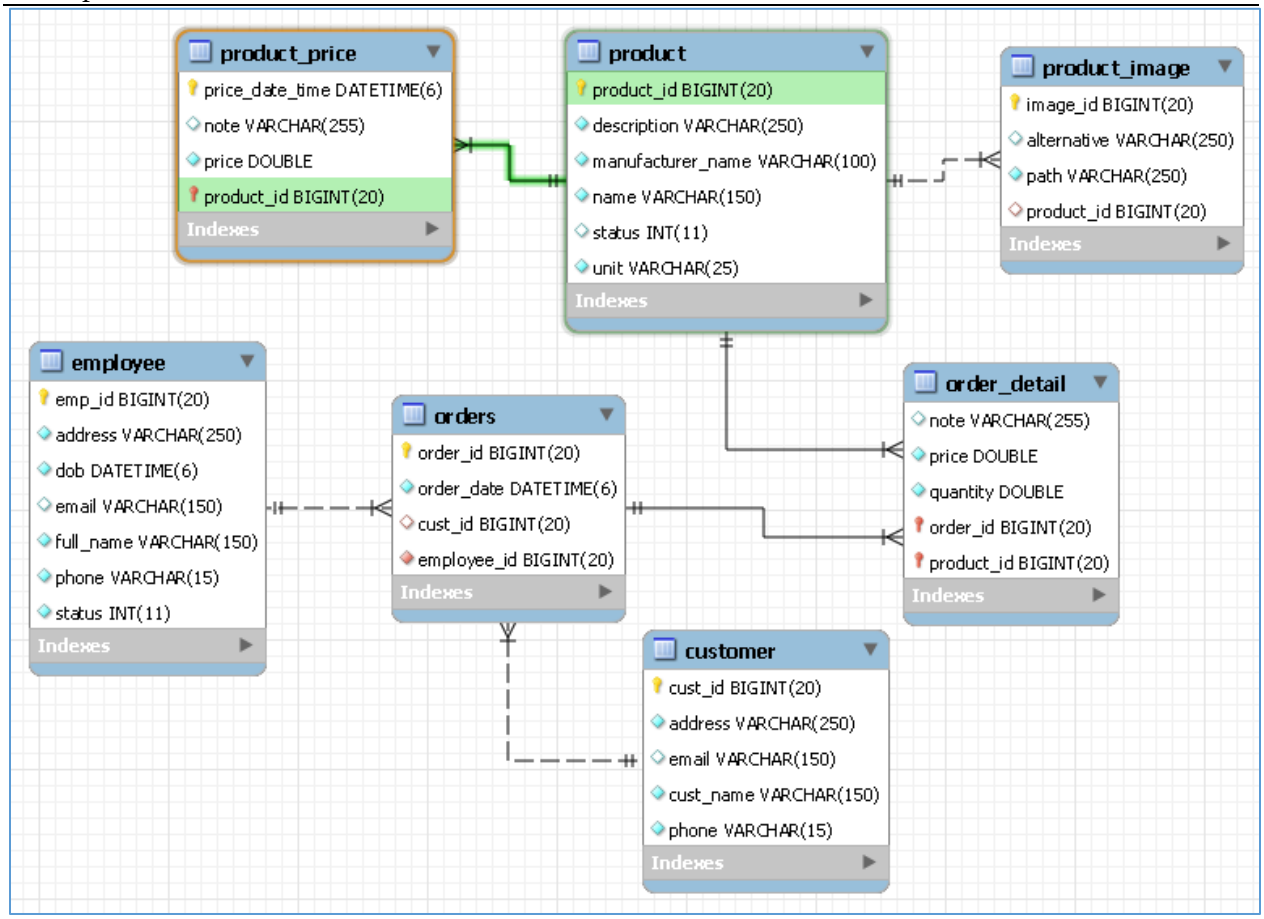

Figure 5: Entity Relationship Diagram

Dependencies

```
dependencies {
    compileOnly('jakarta.platform:jakarta.jakartaee-web-api:10.0.0')

    //for access DB
    implementation 'org.mariadb.jdbc:mariadb-java-client:3.2.0'
    implementation 'org.eclipse.persistence:eclipselink:4.0.2'

    //for logging
    implementation 'org.slf4j:slf4j-api:2.0.9'
    implementation 'org.slf4j:slf4j-simple:2.0.9'

    //for REST API
    implementation('org.glassfish.jersey.containers:jersey-container-
servlet:3.1.2')
    implementation('org.glassfish.jersey.media:jersey-media-json-
jackson:3.1.2')
    implementation('org.glassfish.jersey.inject:jersey-cdi2-se:3.1.2')
    implementation('org.jboss.weld.se:weld-se-core:5.1.0.Final')
    //for convert type in rest
    implementation 'com.fasterxml.jackson.datatype:jackson-datatype-
jsr310:2.15.2'
}
```

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
             https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">
    <persistence-unit name="lab_week_2" transaction-type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="jakarta.persistence.jdbc.driver"
value="org.mariadb.jdbc.Driver"/>
            <property name="jakarta.persistence.jdbc.url"

value="jdbc:mariadb://localhost:3306/mydb?createDatabaseIfNotExist=true"/>
            <property name="jakarta.persistence.jdbc.user" value="root"/>
            <property name="jakarta.persistence.jdbc.password"
value="password"/>
            <property name="jakarta.persistence.schema-
generation.database.action" value="create"/>
            <!--show query-->
            <property name="eclipselink.logging.level.sql" value="FINE"/>
            <property name="eclipselink.logging.parameters" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

Sample source code:

Cấu hình application resources bắt đầu bằng **api**. Ví dụ:
http://localhost:8080/week_02/**api**/employees

```
package vn.edu.iuh.fit.configs;

import jakarta.ws.rs.ApplicationPath;
import jakarta.ws.rs.core.Application;

@ApplicationPath("/api")
public class RootApplication extends Application {
    //.....
}
```

Ví dụ về lớp chuyển đổi kiểu enum EmployeeStatus sang kiểu Integer.

```
package vn.edu.iuh.fit.converters;

import jakarta.persistence.AttributeConverter;
import jakarta.persistence.Converter;
import vn.edu.iuh.fit.enums.EmployeeStatus;
import java.util.stream.Stream;

@Converter(autoApply = true)
public class EmployeeStatusConverter implements
```

```
AttributeConverter<EmployeeStatus, Integer> {
    @Override
    public Integer convertToDatabaseColumn(EmployeeStatus attribute) {
        if (attribute == null) {
            return null;
        }
        return attribute.getValue();
    }

    @Override
    public EmployeeStatus convertToEntityAttribute(Integer dbData) {
        if (dbData == null) {
            return null;
        }

        return Stream.of(EmployeeStatus.values())
                .filter(c -> c.getValue() == dbData)
                .findFirst()
                .orElseThrow(IllegalArgumentException::new);
    }
}
```

Chuyển đổi các dạng ngày giờ

```
package vn.edu.iuh.fit.converters;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import jakarta.ws.rs.ext.ContextResolver;
import jakarta.ws.rs.ext.Provider;

@Provider
public class ObjectMapperContextResolver implements
ContextResolver<ObjectMapper> {

    final ObjectMapper mapper = new ObjectMapper();

    public ObjectMapperContextResolver() {
        mapper.registerModule(new JavaTimeModule());
    }

    @Override
    public ObjectMapper getContext(Class<?> type) {
        return mapper;
    }
}
```

Example repository class

```
package vn.edu.iuh.fit.repositories;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;
import jakarta.persistence.TypedQuery;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import vn.edu.iuh.fit.enums.EmployeeStatus;
```

```
import vn.edu.iuh.fit.models.Employee;

import java.util.List;
import java.util.Optional;

public class EmployeeRepository {
    private EntityManager em;
    private EntityTransaction trans;

    private final Logger logger =
LoggerFactory.getLogger(this.getClass().getName());

    public EmployeeRepository() {
        em = Persistence
                .createEntityManagerFactory("lab_week_2")
                .createEntityManager();
        trans = em.getTransaction();
    }

    public void insertEmp(Employee employee) {
        try {
            trans.begin();
            em.persist(employee);
            trans.commit();
        } catch (Exception ex) {
            trans.rollback();
            logger.error(ex.getMessage());
        }
    }

    public void setStatus(Employee employee, EmployeeStatus status) {
        employee.setStatus(status);
    }

    public void update(Employee employee) {
        try {
            trans.begin();
            em.merge(employee);
            trans.commit();
        } catch (Exception ex) {
            trans.rollback();
            logger.error(ex.getMessage());
        }
    }

    public Optional<Employee> findbyId(long id) {
        TypedQuery<Employee> query = em.createQuery("select e from Employee
e where e.id=:id", Employee.class);
        query.setParameter("id", id);
        Employee emp = query.getSingleResult();
        return emp == null ? Optional.empty() : Optional.of(emp);
    }

    public List<Employee> getAllEmp() {
        return em.createNamedQuery("Employee.findAll", Employee.class)
                .setParameter(1, EmployeeStatus.ACTIVE)
                .getResultList();
    }
    //...
}
```

Sample service

```
package vn.edu.iuh.fit.services;

import vn.edu.iuh.fit.enums.EmployeeStatus;
import vn.edu.iuh.fit.models.Employee;
import vn.edu.iuh.fit.models.Order;
import vn.edu.iuh.fit.repositories.EmployeeRepository;

import java.sql.Date;
import java.util.List;
import java.util.Optional;

public class EmployeeServices {
    private EmployeeRepository repository;

    public EmployeeServices() {
        repository = new EmployeeRepository();
    }

    public void insertEmp(Employee employee) {
        repository.insertEmp(employee);
    }


    public Optional<Employee> findById(long id) {
        return repository.findbyId(id);
    }

    public boolean delete(long id) {
        Optional<Employee> op = findById(id);
        if (op.isPresent()) {
            Employee employee = op.get();
            employee.setStatus(EmployeeStatus.TERMINATED);
            return true;
        }
        return false;
    }

    public boolean activeEmp(long id) {
        Optional<Employee> op = findById(id);
        if (op.isPresent()) {
            Employee employee = op.get();
            employee.setStatus(EmployeeStatus.ACTIVE);
            return true;
        }
        return false;
    }

    public List<Employee> getAll() {
        return repository.getAllEmp();
    }

    public List<Order> getOrdersByPeriod(long empId, Date from, Date to) {
        //......
        return null;
    }
}
```

Code sample cho web-resources với employees

```
package vn.edu.iuh.fit.resources;

import jakarta.ws.rs.*;
import jakarta.ws.rs.core.Response;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import vn.edu.iuh.fit.models.Employee;
import vn.edu.iuh.fit.services.EmployeeServices;

import java.util.List;
import java.util.Optional;

@Path("/employees")
public class EmployeeResources {

    private final EmployeeServices employeeServices;
    private final Logger logger =
LoggerFactory.getLogger(this.getClass().getName());

    public EmployeeResources() {
        employeeServices = new EmployeeServices();
    }

    @GET
    @Produces("application/json")
    @Path("/{id}")
    public Response getEmp(@PathParam("id") long eid) {
        Optional<Employee> empOpt = employeeServices.findById(eid);
        if (empOpt.isPresent()) {
            return Response.ok(empOpt.get()).build();
        }
        return Response.status(Response.Status.BAD_REQUEST).build();
    }

    @GET
    @Produces("application/json")
    public Response getAll() {
        //paging if possible
        List<Employee> lst = employeeServices.getAll();
        return Response.ok(lst).build();
    }

    @POST
    @Produces("application/json")
    @Consumes("application/json")
    public Response insert(Employee employee) {
        //ResponseEntity
        employeeServices.insertEmp(employee);
        return Response.ok(employee).build();
    }

    @DELETE
    @Path("/{id}")
    public Response delete(@PathParam("id") long id) {
        if (employeeServices.delete(id))
```

```
            return Response.ok().build();
        return Response.status(Response.Status.NOT_FOUND).build();
    }

}
```