T.C. Mudanya Üniversitesi
Veritabanı Yönetim Sistemleri Dersi
Proje Ödevi

Okul No:	234001077
Ad:	Oktay
Soyad:	Aydoğan

Proje Tanıtımı
Proje Adı: Kullanıcı Yönetim Sistemi
Teknolojiler: Spring Boot, JPA (Hibernate), PostgreSQL
Proje Amacı: Kullanıcı bilgilerini ve adreslerini yönetmek için bir RESTful API geliştirmek.

İçerik

Proje Kodu ve Açıklamaları

1. Giriş

ORM Nedir? ORM (Object Relational Mapping), nesne yönelimli programlama ile ilişkisel veri tabanları arasındaki uyumu sağlamak için kullanılan bir tekniktir. ORM, veri tabanındaki tabloları sınıflara, satırları nesnelere ve sütunları özelliklere dönüştürerek, veritabanı işlemlerini daha az kod yazarak gerçekleştirmemizi sağlar.

JPA ve Hibernate JPA (Java Persistence API), Java platformunda veri tabanı işlemlerini gerçekleştirmek için kullanılan bir spesifikasyondur. Hibernate ise, JPA spesifikasyonunu uygulayan popüler bir ORM aracıdır.

Spring Boot Spring Boot, Spring Framework'ün konfigürasyonel karmaşıklığını azaltan, hızlı ve üretim hazır uygulamalar geliştirmeyi kolaylaştıran bir araçtır.

2. Projenin Kurulumu ve Çalıştırılması

Gerekli Yazılımlar

- Java Development Kit (JDK) 8 veya üzeri
- Gradle
- PostgreSQL

Kurulum Adımları

1. PostgreSQL'i kurun ve **USERS-DB** adında bir veri tabanı oluşturun.
2. **application.yml** dosyasındaki veri tabanı bağlantı bilgilerini kendi sisteminize göre güncelleyin:

```yaml
spring:
  application:
    name: finalExam
  datasource:
    url: jdbc:postgresql://localhost:5400/USERS-DB
    username: postgres
    password: users123
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
server:
  port: 8099
```

1. Proje dizininde **./gradlew build** komutunu çalıştırarak bağımlılıkları yükleyin.
2. **./gradlew bootRun** komutunu çalıştırarak uygulamayı başlatın.

3. Proje Yapısı

Katmanlı Mimari

- **Controller**: HTTP isteklerini karşılar ve iş mantığı için Service katmanına yönlendirir.
- **Service**: İş mantığını içerir ve veri tabanı işlemleri için Repository katmanına erişir.
- **Repository**: Veri tabanı işlemlerini gerçekleştirmek için JPA'yı kullanır.
- **Model**: Veri tabanı tablolarını temsil eden sınıflardır.

DTO ve Entity

- **DTO (Data Transfer Object)**: Veri transferi sırasında kullanılan nesnelerdir. Genellikle istemci ve sunucu arasında veri taşımak için kullanılır.
- **Entity**: Veri tabanındaki tabloları temsil eden sınıflardır. JPA tarafından yönetilirler.

4. Veri Tabanı Tasarımı

Kullanıcı Tablosu (Users)

Alan Adı	Tip	Açıklama
id	Long	Birincil anahtar
name	String	Kullanıcı adı
surname	String	Kullanıcı soyadı
email	String	Kullanıcı email adresi
address	Users_Addresses	Kullanıcı adresi ile ilişki (One-to-One)

Adres Tablosu (Users_Addresses)

Alan Adı	Tip	Açıklama
id	Long	Birincil anahtar
city	String	Şehir
district	String	İlçe
hometown	String	Memleket
user	Users	Kullanıcı ile ilişki (One-to-One)

İlişkiler

- **Users** ve **Users_Addresses** tabloları arasında bir One-to-One ilişkisi bulunur.

5. API Endpoints

Kullanıcı Kayıt (POST /register)

```
@PostMapping("/register")  Oktay Aydoğan
public ResponseEntity<BaseResponseDto> register(@RequestBody UserDto dtoObj) {
    try {
        _userService.register(dtoObj);
        _logger.info("User registered successfully");
        return ResponseEntity.status(201).body(dtoObj);
    } catch (DataIntegrityViolationException e) {
        _logger.error("An DataIntegrity error occurred while adding the user. Exception: {}", e.getMessage());
        return ResponseEntity.status(400).body(ExceptionDto.builder().message("User already exists.").build());
    } catch (Exception e) {
        _logger.error("An error occurred while adding the user. Exception: {}", e.getMessage());
        return ResponseEntity.status(500).body(ExceptionDto.builder().message(e.getMessage()).build());
    }
}
```

Kullanıcı Arama (POST /search)

```
@PostMapping("/search")  Oktay Aydoğan
public ResponseEntity<List<UserDto>> search(@RequestBody UserDto dtoObj) {
    try {
        List<UserDto> userList = _userService.search(dtoObj.getCity());
        _logger.info("Fetching all users by city.");
        return ResponseEntity.status(200).body(userList);
    } catch (Exception e) {
        _logger.error("An error occurred while fetching all users by city. Exception: {}", e.getMessage());
        return ResponseEntity.status(500).body(null);
    }
}
```

Tüm Kullanıcıları Listeleme (GET /all)

```
@GetMapping("/all")  Oktay Aydoğan
public ResponseEntity<List<UserDto>>  getAllUsers() {
    try {
        List<UserDto> userList = _userService.getAllUsers();

        _logger.info("Fetching all users.");
        return ResponseEntity.status(200).body(userList);
    } catch (Exception e) {
        _logger.error("An error occurred while fetching all users. Exception: {}", e.getMessage());
        return ResponseEntity.status(500).body(null);
    }
}
```

6. Güvenlik ve Yedekleme

Spring Security ile Güvenlik Spring Security kullanarak temel kimlik doğrulama ve yetkilendirme işlemleri gerçekleştirilebilir. Projeye güvenlik katmanı eklemek, API'lerin yetkisiz erişimlere karşı korunmasını sağlar.

Veri Tabanı Yedekleme

- PostgreSQL yedekleme araçları kullanarak düzenli yedeklemeler alınabilir.
- Yedekleme stratejileri ile veri kaybı riski minimize edilir.

7. Sonuç

Bu proje, ORM kullanarak Spring Boot ile nasıl bir kullanıcı yönetim sistemi geliştirilebileceğini göstermektedir. İleriye dönük olarak, kullanıcı yönetim sistemine ek özellikler eklenebilir ve güvenlik önlemleri daha da güçlendirilebilir.

```java
package org.MTSG.finalExam;

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

@SpringBootApplication
public class FinalExamApplication {

    public static void main(String[] args) {
        SpringApplication.run(FinalExamApplication.class,
args);
    }

}

```

```java
package org.MTSG.finalExam.dto;

import lombok.*;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class UserDto extends BaseResponseDto {
    private String name;
    private String surname;
    private String email;
    private String city;
    private String district;
    private String hometown;
}

```

```java
package org.MTSG.finalExam.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@Builder
public class ExceptionDto extends BaseResponseDto {
    private String message;
}

```

File - /Volumes/Sandisk/finalExam/src/main/java/org/MTSG/finalExam/dto/BaseResponseDto.java

```java
package org.MTSG.finalExam.dto;

public class BaseResponseDto {
}

```

```java
package org.MTSG.finalExam.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@Table(name = "users")
public class Users {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private long id;

    @Column(name = "NAME")
    private String name;

    @Column(name = "SURNAME")
    private String surname;

    @Column(name = "EMAIL")
    private String email;

    @OneToOne(cascade = CascadeType.ALL)
    private Users_Addresses address;
}

```

```java
package org.MTSG.finalExam.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@Table(name = "users_addresses")
public class Users_Addresses {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private long id;

    @Column(name = "CITY")
    private String city;

    @Column(name = "DISTRICT")
    private String district;

    @Column(name = "HOMETOWN")
    private String hometown;

    @OneToOne(mappedBy = "address")
    private Users user;
}

```

```java
package org.MTSG.finalExam.service;

import org.MTSG.finalExam.dto.UserDto;

import java.util.List;

public interface UserService {
    List<UserDto> getAllUsers();
    void register(UserDto userDto);
    List<UserDto> search(String city);
}

```

```java
package org.MTSG.finalExam.service.impl;

import org.MTSG.finalExam.dto.UserDto;
import org.MTSG.finalExam.model.Users;
import org.MTSG.finalExam.model.Users_Addresses;
import org.MTSG.finalExam.repository.UserRepository;
import org.MTSG.finalExam.service.UserService;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Service;

import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class UserService_Impl implements UserService {
    private final Logger _logger = LoggerFactory.getLogger(UserService_Impl.class);
    private final UserRepository _userRepository;

    @Override
    public List<UserDto> getAllUsers() {
        try {
            List<Users> userList = _userRepository.findAll();

            if(!userList.isEmpty()){
                _logger.info("Fetching all users.");

                return userList.stream()
                        .map(user → UserDto.builder()
                                .name(user.getName())
                                .surname(user.getSurname())
                                .email(user.getEmail())
                                .city(user.getAddress().getCity())
                                .district(user.getAddress().getDistrict())
                                .hometown(user.getAddress().getHometown())
                                .build())
                        .collect(Collectors.toList());
            }
```

```
43
44              _logger.info("No users found.");
45              return Collections.emptyList();
46          } catch (Exception e) {
47              _logger.error("An error occurred while fetching
    the users. Exception: {}", e.getMessage());
48              throw new RuntimeException("An error occurred
   while fetching the users.");
49          }
50      }
51
52      @Override
53      public void register(UserDto obj) {
54          try {
55
56              Users_Addresses address = Users_Addresses.
   builder()
57                      .city(obj.getCity())
58                      .district(obj.getDistrict())
59                      .hometown(obj.getHometown())
60                      .build();
61
62              Users user = Users.builder()
63                      .name(obj.getName())
64                      .surname(obj.getSurname())
65                      .email(obj.getEmail())
66                      .address(address)
67                      .build();
68
69              Users savedEntity = _userRepository.save(user);
70              _logger.info("User saved: {}", savedEntity);
71          } catch (DataIntegrityViolationException e) {
72              _logger.error("An DataIntegrity error occurred
   while saving the User. Exception: {}", e.getMessage());
73              throw new DataIntegrityViolationException(e.
   getMessage());
74          } catch (Exception e) {
75              _logger.error("An error occurred while saving
   the User. Exception: {}", e.getMessage());
76              throw new RuntimeException(e.getMessage());
77          }
78      }
79
80      public List<UserDto> search(String city) {
81          try {
82              List<Users> userList = _userRepository.
   findByAddress_City(city);
83
```

```
 84              if(!userList.isEmpty()){
 85                  _logger.info("Fetching all users by city."
    );
 86
 87                  return userList.stream()
 88                          .map(user → UserDto.builder()
 89                                  .name(user.getName())
 90                                  .surname(user.getSurname
    ())
 91                                  .email(user.getEmail())
 92                                  .city(user.getAddress().
    getCity())
 93                                  .district(user.getAddress
    ().getDistrict())
 94                                  .hometown(user.getAddress
    ().getHometown())
 95                                  .build())
 96                          .collect(Collectors.toList());
 97              }
 98
 99              _logger.info("No users found.");
100              return Collections.emptyList();
101          } catch (Exception e) {
102              _logger.error("An error occurred while
    fetching the users by city. Exception: {}", e.getMessage
    ());
103              throw new RuntimeException("An error occurred
    while fetching the users by city.");
104          }
105      }
106 }
107
```

File - /Volumes/Sandisk/finalExam/src/main/java/org/MTSG/finalExam/controller/UsersController.java

```java
package org.MTSG.finalExam.controller;

import org.MTSG.finalExam.dto.BaseResponseDto;
import org.MTSG.finalExam.dto.ExceptionDto;
import org.MTSG.finalExam.dto.UserDto;
import lombok.RequiredArgsConstructor;
import org.MTSG.finalExam.service.impl.UserService_Impl;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.
DataIntegrityViolationException;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/user")
@RequiredArgsConstructor
public class UsersController {
    private final Logger _logger = LoggerFactory.getLogger(
UsersController.class);
    private final UserService_Impl _userService;

    @GetMapping("/all")
    public ResponseEntity<List<UserDto>>  getAllUsers() {
        try {
            List<UserDto> userList = _userService.
getAllUsers();

            _logger.info("Fetching all users.");
            return ResponseEntity.status(200).body(userList
);
        } catch (Exception e) {
            _logger.error("An error occurred while fetching
 all users. Exception: {}", e.getMessage());
            return ResponseEntity.status(500).body(null);
        }
    }

    @PostMapping("/register")
    public ResponseEntity<BaseResponseDto> register(@
RequestBody UserDto dtoObj) {
        try {
            _userService.register(dtoObj);
            _logger.info("User registered successfully");
            return ResponseEntity.status(201).body(dtoObj);
        } catch (DataIntegrityViolationException e) {
```

```
43              _logger.error("An DataIntegrity error occurred
   while adding the user. Exception: {}", e.getMessage());
44              return ResponseEntity.status(400).body(
   ExceptionDto.builder().message("User already exists.").
   build());
45          } catch (Exception e) {
46              _logger.error("An error occurred while adding
   the user. Exception: {}", e.getMessage());
47              return ResponseEntity.status(500).body(
   ExceptionDto.builder().message(e.getMessage()).build());
48          }
49      }
50
51      @PostMapping("/search")
52      public ResponseEntity<List<UserDto>> search(@
   RequestBody UserDto dtoObj) {
53          try {
54              List<UserDto> userList = _userService.search(
   dtoObj.getCity());
55              _logger.info("Fetching all users by city.");
56              return ResponseEntity.status(200).body(userList
   );
57          } catch (Exception e) {
58              _logger.error("An error occurred while fetching
    all users by city. Exception: {}", e.getMessage());
59              return ResponseEntity.status(500).body(null);
60          }
61      }
62
63 }
64
```

File - /Volumes/Sandisk/finalExam/src/main/java/org/MTSG/finalExam/repository/UserRepository.java

```java
package org.MTSG.finalExam.repository;

import org.MTSG.finalExam.model.Users;
import org.springframework.data.jpa.repository.
JpaRepository;

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

public interface UserRepository extends JpaRepository<Users
, Long> {
    List<Users> findByAddress_City(String city);
}

```

File - /Volumes/Sandisk/finalExam/src/main/resources/application.yml

```yaml
spring:
  application:
    name: finalExam
  datasource:
    url: jdbc:postgresql://localhost:5400/USERS-DB
    username: postgres
    password: users123
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
server:
  port: 8099
```