

```
1 package org.MTSG.finalExam;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.
  SpringApplication;
5
6 @SpringBootApplication
7 public class FinalExamApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(FinalExamApplication.class,
11             args);
12     }
13 }
14
```

```
1 package org.MTSG.finalExam.dto;
2
3 import lombok.*;
4
5 @Getter
6 @Setter
7 @Builder
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class UserDto extends BaseResponseDto {
11     private String name;
12     private String surname;
13     private String email;
14     private String city;
15     private String district;
16     private String hometown;
17 }
18
```

```
1 package org.MTSG.finalExam.dto;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Builder;
5 import lombok.Getter;
6 import lombok.Setter;
7
8 @Getter
9 @Setter
10 @AllArgsConstructor
11 @Builder
12 public class ExceptionDto extends BaseResponseDto {
13     private String message;
14 }
15
```

```
1 package org.MTSG.finalExam.dto;  
2  
3 public class BaseResponseDto {  
4 }  
5
```

```
1 package org.MTSG.finalExam.model;
2
3 import jakarta.persistence.*;
4 import lombok.*;
5
6 @Entity
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Getter
10 @Setter
11 @Builder
12 @Table(name = "users")
13 public class Users {
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     @Column(name = "ID")
17     private long id;
18
19     @Column(name = "NAME")
20     private String name;
21
22     @Column(name = "SURNAME")
23     private String surname;
24
25     @Column(name = "EMAIL")
26     private String email;
27
28     @OneToOne(cascade = CascadeType.ALL)
29     private Users_Addresses address;
30 }
31
```

```
1 package org.MTSG.finalExam.model;
2
3 import jakarta.persistence.*;
4 import lombok.*;
5
6 @Entity
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @Getter
10 @Setter
11 @Builder
12 @Table(name = "users_addresses")
13 public class Users_Addresses {
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     @Column(name = "ID")
17     private long id;
18
19     @Column(name = "CITY")
20     private String city;
21
22     @Column(name = "DISTRICT")
23     private String district;
24
25     @Column(name = "HOMETOWN")
26     private String hometown;
27
28     @OneToOne(mappedBy = "address")
29     private Users user;
30 }
31
```

```
1 package org.MTSG.finalExam.service;
2
3 import org.MTSG.finalExam.dto.UserDto;
4
5 import java.util.List;
6
7 public interface UserService {
8     List<UserDto> getAllUsers();
9     void register(UserDto userDto);
10    List<UserDto> search(String city);
11 }
12
```

```

1  package org.MTSG.finalExam.service.impl;
2
3  import org.MTSG.finalExam.dto.UserDto;
4  import org.MTSG.finalExam.model.Users;
5  import org.MTSG.finalExam.model.Users_Addresses;
6  import org.MTSG.finalExam.repository.UserRepository;
7  import org.MTSG.finalExam.service.UserService;
8  import lombok.RequiredArgsConstructor;
9  import org.slf4j.Logger;
10 import org.slf4j.LoggerFactory;
11 import org.springframework.dao.
    DataIntegrityViolationException;
12 import org.springframework.stereotype.Service;
13
14 import java.util.Collections;
15 import java.util.List;
16 import java.util.stream.Collectors;
17
18 @Service
19 @RequiredArgsConstructor
20 public class UserService_Impl implements UserService {
21     private final Logger _logger = LoggerFactory.getLogger(
        UserService_Impl.class);
22     private final UserRepository _userRepository;
23
24     @Override
25     public List<UserDto> getAllUsers() {
26         try {
27             List<Users> userList = _userRepository.findAll
        ();
28
29             if(!userList.isEmpty()){
30                 _logger.info("Fetching all users.");
31
32                 return userList.stream()
33                     .map(user → UserDto.builder()
34                         .name(user.getName())
35                         .surname(user.getSurname())
36                         .email(user.getEmail())
37                         .city(user.getAddress().
        getCity())
38                         .district(user.getAddress
        ().getDistrict())
39                         .hometown(user.getAddress
        ().getHometown())
40                         .build())
41                     .collect(Collectors.toList());
42             }

```



```

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                     ())
92                 .email(user.getEmail())
93                 .city(user.getAddress().
94                     getCity())
95                 .district(user.getAddress
96                     ().getDistrict())
97                 .hometown(user.getAddress
98                     ().getHometown())
99                 .build())
100             .collect(Collectors.toList());
101         }
102         _logger.info("No users found.");
103         return Collections.emptyList();
104     } catch (Exception e) {
105         _logger.error("An error occurred while
106         fetching the users by city. Exception: {}", e.getMessage
107         ());
108         throw new RuntimeException("An error occurred
109         while fetching the users by city.");
110     }
111 }
```

```

1  package org.MTSG.finalExam.controller;
2
3  import org.MTSG.finalExam.dto.BaseResponseDto;
4  import org.MTSG.finalExam.dto.ExceptionDto;
5  import org.MTSG.finalExam.dto.UserDto;
6  import lombok.RequiredArgsConstructor;
7  import org.MTSG.finalExam.service.impl.UserService_Impl;
8  import org.slf4j.Logger;
9  import org.slf4j.LoggerFactory;
10 import org.springframework.dao.
    DataIntegrityViolationException;
11 import org.springframework.http.ResponseEntity;
12 import org.springframework.web.bind.annotation.*;
13
14 import java.util.List;
15
16 @RestController
17 @RequestMapping("/api/user")
18 @RequiredArgsConstructor
19 public class UsersController {
20     private final Logger _logger = LoggerFactory.getLogger(
        UsersController.class);
21     private final UserService_Impl _userService;
22
23     @GetMapping("/all")
24     public ResponseEntity<List<UserDto>> getAllUsers() {
25         try {
26             List<UserDto> userList = _userService.
                getAllUsers();
27
28             _logger.info("Fetching all users.");
29             return ResponseEntity.status(200).body(userList
        );
30         } catch (Exception e) {
31             _logger.error("An error occurred while fetching
        all users. Exception: {}", e.getMessage());
32             return ResponseEntity.status(500).body(null);
33         }
34     }
35
36     @PostMapping("/register")
37     public ResponseEntity<BaseResponseDto> register(@
        RequestBody UserDto dtoObj) {
38         try {
39             _userService.register(dtoObj);
40             _logger.info("User registered successfully");
41             return ResponseEntity.status(201).body(dtoObj);
42         } 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
```

```
1 package org.MTSG.finalExam.repository;
2
3 import org.MTSG.finalExam.model.Users;
4 import org.springframework.data.jpa.repository.
    JpaRepository;
5
6 import java.util.List;
7 import java.util.Optional;
8
9 public interface UserRepository extends JpaRepository<Users
    , Long> {
10     List<Users> findByAddress_City(String city);
11 }
12
```

```
1 spring:  
2   application:  
3     name: finalExam  
4   datasource:  
5     url: jdbc:postgresql://localhost:5400/USERS-DB  
6     username: postgres  
7     password: users123  
8     driver-class-name: org.postgresql.Driver  
9   jpa:  
10     hibernate:  
11       ddl-auto: update  
12       show-sql: true  
13 server:  
14   port: 8099
```