

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
  
к лабораторной работе №2 по дисциплине «Микросервисы»

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гижевская В.Д.

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алкеев М.Г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лобанков А.А.

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc153156072)

[2 Результаты работы 5](#_Toc153156073)

[ПРИЛОЖЕНИЕ А Листинг программы 9](#_Toc153156074)

1. Постановка задачи

В процессе выполнения заданий ознакомиться с возможностями Spring Cloud Config Server.

Задание 1

Установить Docker. Создать docker-compose.yaml файл, в котором описать необходимые для работы сервисы (postgresql). Добавить настройки подключения к созданной базе данных для user-service и company-service (для каждого микросервиса укажите свою схему данных).

Задание 2

Подключить фреймворк для управления миграциями баз данных FlyWay в user-service и company-service. Создайте скрипт для создания таблицы user (имя, логин, пароль, емэйл, признак активации/деактивации, идентификатор компании). Для companyservice создайте скрипт для создания таблицы компаний (название, огрн, описание дейтельности, идентификатор директора).

Задание 3

Создайте соотвеnствующие Entity, Dto (для пользователей должно возвращаться название компании, для компаний – ФИО пользователя), Repository, Service для таблиц из задания 2

Задание 4

Создайте контроллер в user-service, в котором реализуйте следующие end-points:

* Получение списка всех пользователей (информация о компании должна подтягиваться синхронное из company-service)
* Активация/деактивация пользователя
* Создание пользователя (должна быть синхронная проверка из company-service, на существование компании. Если компании не существует – кидать 404 ошибку с соответствующим сообщением)
* Обновление информации о пользователе (только имя, емэйл и компанию, для компании должна быть проверка см. выше )
* Проверка существования пользователя по идентификатору (если пользователь существует, но неактивен – тоже кидаем 404 ошибку)

Создайте контроллер в company-service, в котором реализуйте следующие end-points:

* Проверка существования компании по ее идентификатору
* Получение списка всех компаний ( с ФИО директора)
* Создание компании (должна быть синхронная проверка из user-service, на существование директора.
* Если директора не существует – кидать 404 ошибку с соответствующим сообщением).

1. Результаты работы

Ниже приведены скриншоты с результатами работы программы для user-service.



Рисунок 1 - Получение списка всех пользователей (информация о компании подтягивается синхронно из company-service)

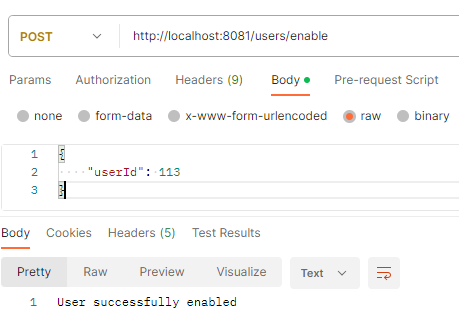


Рисунок 2 - Активация пользователя

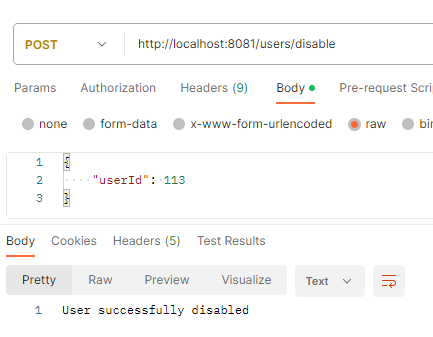


Рисунок 3 - Деактивация пользователя

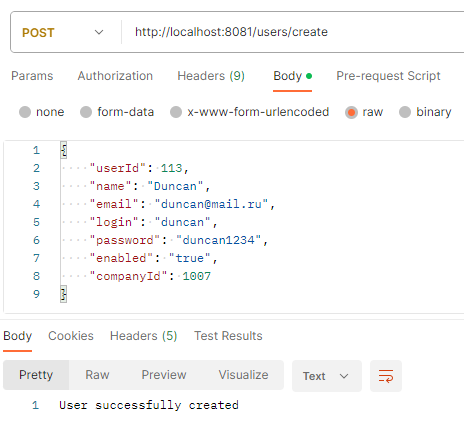


Рисунок 4 - Создание пользователя

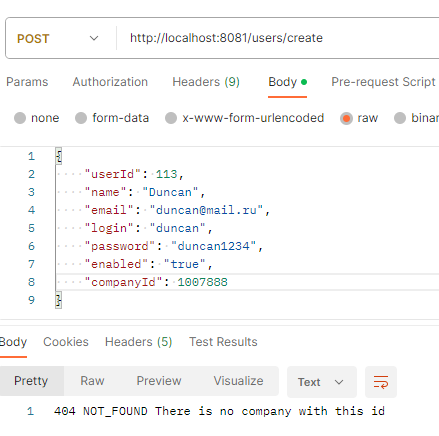


Рисунок 5 - Создание пользователя (синхронная проверка из company-service на существование компании показала, что компании не существует и кинула 404 ошибку с соответствующим сообщением)

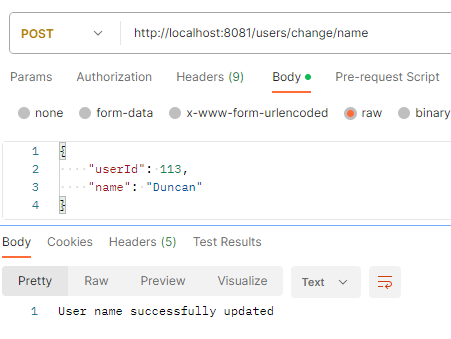


Рисунок 6 - Обновление информации о пользователе (Имя)

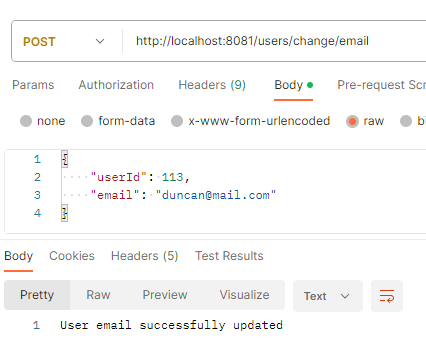


Рисунок 7 - Обновление информации о пользователе (Эл.почта)

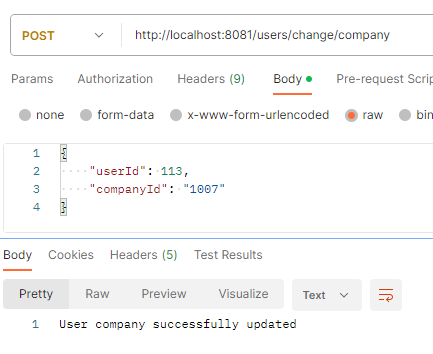


Рисунок 8 - Обновление информации о пользователе (Компания)

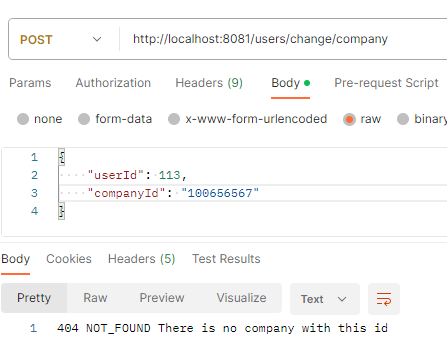


Рисунок 9 - Обновление информации о пользователе (Компания, которой не существует)

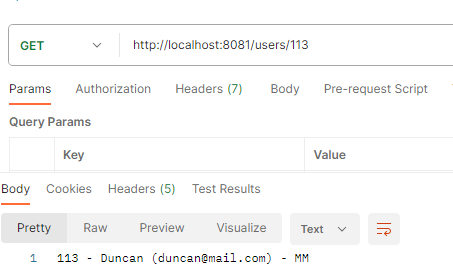


Рисунок 10 - Проверка существования пользователя по идентификатору

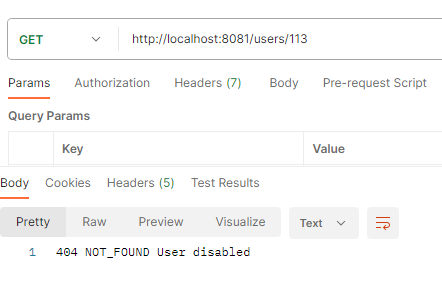


Рисунок 11 - Проверка существования пользователя по идентификатору (Пользователь не активен)

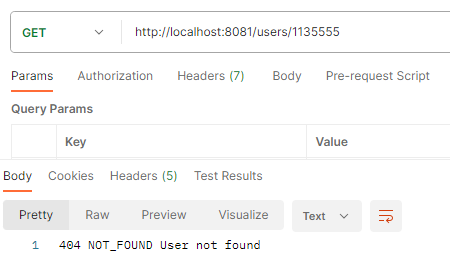


Рисунок 11 - Проверка существования пользователя по идентификатору (Пользователя не существует)

ПРИЛОЖЕНИЕ А  
Листинг программы

**docker-compose.yml**

version: '3.7'  
  
services:  
 postgres:  
 image: postgres:15.2  
 container\_name: user-company-sql-db  
 environment:  
 POSTGRES\_USER: "postgres"  
 POSTGRES\_PASSWORD: "1111"  
 POSTGRES\_DB: "microservices"  
 ports:  
 - "5432:5432"

**company-service.application.properties:**

spring.application.name=company-service  
spring.cloud.config.uri=${CONFIG\_SERVER\_URL}  
  
spring.datasource.url=jdbc:postgresql://localhost:5432/microservices?currentSchema=company-schema  
spring.datasource.username=postgres  
spring.datasource.password=1111

**user-service.application.properties:**

spring.application.name=user-service  
spring.cloud.config.uri=${CONFIG\_SERVER\_URL}  
  
spring.datasource.url=jdbc:postgresql://localhost:5432/microservices?currentSchema=user-schema  
spring.datasource.username=postgres  
spring.datasource.password=1111

**V1\_create\_CompaniesTable:**

-- Table: company-schema.companies  
  
-- DROP TABLE IF EXISTS "company-schema".companies;  
  
CREATE TABLE IF NOT EXISTS "company-schema".companies  
(  
 company\_id integer NOT NULL,  
 name character varying COLLATE pg\_catalog."default" NOT NULL,  
 ogrn character varying COLLATE pg\_catalog."default" NOT NULL,  
 description text COLLATE pg\_catalog."default" NOT NULL,  
 director\_id integer NOT NULL,  
 CONSTRAINT companies\_pkey PRIMARY KEY (company\_id)  
)  
  
 TABLESPACE pg\_default;  
  
ALTER TABLE IF EXISTS "company-schema".companies  
 OWNER to postgres;

**V1\_createUsersTable:**

CREATE TABLE IF NOT EXISTS "user-schema".users  
(  
 user\_id integer NOT NULL,  
 name character varying COLLATE pg\_catalog."default" NOT NULL,  
 email character varying COLLATE pg\_catalog."default" NOT NULL,  
 login character varying COLLATE pg\_catalog."default" NOT NULL,  
 password character varying COLLATE pg\_catalog."default" NOT NULL,  
 enabled boolean NOT NULL,  
 company\_id integer NOT NULL,  
 CONSTRAINT users\_pkey PRIMARY KEY (user\_id)  
 )  
  
 TABLESPACE pg\_default;  
  
ALTER TABLE IF EXISTS "user-schema".users  
 OWNER to postgres;

**UserEntity:**

package com.userservice;  
  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Table;  
import org.springframework.data.annotation.Id;  
  
@Entity  
@Table(schema = "user-schema", name = "users")  
public class UserEntity {  
  
 @jakarta.persistence.Id  
 @Id  
 @Column(name = "user\_id")  
 private Integer userId;  
  
 @Column(name = "name")  
 private String name;  
  
 @Column(name = "email")  
 private String email;  
  
 @Column(name = "login")  
 private String login;  
  
 @Column(name = "password")  
 private String password;  
  
 @Column(name = "enabled")  
 private Boolean enabled;  
  
 @Column(name = "company\_id")  
 private Integer companyId;  
  
  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
  
 public Integer getUserId() {  
 return userId;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public Boolean getEnabled() {  
 return enabled;  
 }  
  
 public void setEnabled(Boolean enabled) {  
 this.enabled = enabled;  
 }  
  
 public Integer getCompanyId() {  
 return companyId;  
 }  
  
 public void setCompanyId(Integer companyId) {  
 this.companyId = companyId;  
 }  
}

**UserDTO:**

package com.userservice;  
  
import org.springframework.web.client.RestTemplate;  
  
public class UserDTO {  
 private Integer userId;  
 private String name;  
 private String email;  
 private String login;  
 private String password;  
 private Boolean enabled;  
 private Integer companyId;  
 private String companyName;  
  
 public Integer getUserId() {  
 return userId;  
 }  
  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public Boolean getEnabled() {  
 return enabled;  
 }  
  
 public void setEnabled(Boolean enabled) {  
 this.enabled = enabled;  
 }  
  
 public Integer getCompanyId() {  
 return companyId;  
 }  
  
 public void setCompanyId(Integer companyId) {  
 this.companyId = companyId;  
 }  
  
 public String getCompanyName() {  
 return companyName;  
 }  
  
 public void setCompanyName(String companyName) {  
 this.companyName = companyName;  
 }  
  
}

**UserRepository:**

package com.userservice;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface UserRepository extends JpaRepository<UserEntity, Integer> {  
  
}

**UserService:**

package com.userservice;  
  
import jakarta.persistence.EntityNotFoundException;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.stereotype.Service;  
import org.springframework.web.client.RestTemplate;  
import org.springframework.web.server.ResponseStatusException;  
  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
@Service  
public class UserService {  
 private final UserRepository userRepository;  
 private final CompanyServiceFeignClient companyServiceFeignClient;  
  
  
 @Autowired  
 public UserService(UserRepository userRepository, CompanyServiceFeignClient companyServiceFeignClient) {  
 this.userRepository = userRepository;  
 this.companyServiceFeignClient = companyServiceFeignClient;  
 }  
  
  
 public boolean doesCompanyExist(Integer companyId) {  
 return companyServiceFeignClient.existById(companyId);  
 }  
  
 public boolean doesUserExist(Integer userId) {  
 return userRepository.existsById(userId);  
 }  
  
 public List<UserDTO> getAllUsers() {  
 List<UserEntity> users = userRepository.findAll();  
 return transformToDTOList(users);  
 }  
  
 public UserDTO getUserById(Integer userId) {  
 UserEntity user = userRepository.findById(userId)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "User not found"));  
 if (!user.getEnabled()) {  
 throw new ResponseStatusException(HttpStatus.NOT\_FOUND, "User disabled");  
 }  
 return transformToDTO(user);  
 }  
  
 public String getUserNameByID(Integer userId) {  
 UserEntity user = userRepository.findById(userId)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "User not found"));  
 return user.getName();  
 }  
  
 public UserDTO createUser(UserDTO userDTO) {  
 UserEntity userEntity = transformToEntity(userDTO);  
 UserEntity savedUser = userRepository.save(userEntity);  
 if (!doesCompanyExist(userDTO.getCompanyId())) {  
 throw new ResponseStatusException(HttpStatus.NOT\_FOUND, "There is no company with this id");  
 }  
 return transformToDTO(savedUser);  
 }  
  
 public UserDTO changeUserEnabledField(Integer userId, boolean enabled) {  
 UserEntity user = userRepository.findById(userId)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "User not found"));  
 user.setEnabled(enabled);  
 UserEntity updatedUser = userRepository.save(user);  
 return transformToDTO(updatedUser);  
 }  
  
 public UserDTO changeUserName(Integer userId, String newName) {  
 UserEntity user = userRepository.findById(userId)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "User not found"));  
 user.setName(newName);  
 UserEntity updatedUser = userRepository.save(user);  
 return transformToDTO(updatedUser);  
 }  
  
 public UserDTO changeUserEmail(Integer userId, String newEmail) {  
 UserEntity user = userRepository.findById(userId)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "User not found"));  
 user.setEmail(newEmail);  
 UserEntity updatedUser = userRepository.save(user);  
 return transformToDTO(updatedUser);  
 }  
  
 public UserDTO changeUserCompany(Integer userId, Integer newCompanyId) {  
 UserEntity user = userRepository.findById(userId)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "User not found"));  
 if (!doesCompanyExist(newCompanyId)) {  
 throw new ResponseStatusException(HttpStatus.NOT\_FOUND, "There is no company with this id");  
 }  
 user.setCompanyId(newCompanyId);  
 UserEntity updatedUser = userRepository.save(user);  
 return transformToDTO(updatedUser);  
 }  
  
  
 private UserDTO transformToDTO(UserEntity user) {  
 UserDTO userDTO = new UserDTO();  
 userDTO.setUserId(user.getUserId());  
 userDTO.setName(user.getName());  
 userDTO.setEmail(user.getEmail());  
 userDTO.setLogin(user.getLogin());  
 userDTO.setPassword(user.getPassword());  
 userDTO.setEnabled(user.getEnabled());  
 userDTO.setCompanyId(user.getCompanyId());  
 userDTO.setCompanyName(companyServiceFeignClient.getCompanyNameById(user.getCompanyId()));  
 return userDTO;  
 }  
  
 private List<UserDTO> transformToDTOList(List<UserEntity> users) {  
 return users.stream()  
 .map(this::transformToDTO)  
 .collect(Collectors.toList());  
 }  
  
 private UserEntity transformToEntity(UserDTO userDTO) {  
 UserEntity userEntity = new UserEntity();  
 userEntity.setUserId(userDTO.getUserId());  
 userEntity.setName(userDTO.getName());  
 userEntity.setEmail(userDTO.getEmail());  
 userEntity.setLogin(userDTO.getLogin());  
 userEntity.setPassword(userDTO.getPassword());  
 userEntity.setEnabled(userDTO.getEnabled());  
 userEntity.setCompanyId(userDTO.getCompanyId());  
 return userEntity;  
 }  
}

**UserController:**

package com.userservice;  
  
import jakarta.persistence.EntityNotFoundException;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.server.ResponseStatusException;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/users")  
public class UserController {  
 private final UserService userService;  
  
  
 @Autowired  
 public UserController(UserService userService) {  
 this.userService = userService;  
 }  
  
 @GetMapping  
 public List<UserDTO> getAllUsers() {  
 return userService.getAllUsers();  
 }  
  
 @GetMapping("/exists/{userId}")  
 public Boolean doesUserExist(@PathVariable("userId") Integer userId) {  
 return userService.doesUserExist(userId);  
 }  
  
  
 @GetMapping("/{userId}")  
 public ResponseEntity<String> getUserById(@PathVariable("userId") Integer userId) {  
 try {  
 UserDTO userDTO = userService.getUserById(userId);  
 String userInformation = userDTO.getUserId() + " - " + userDTO.getName() + " (" + userDTO.getEmail() + ") - " + userDTO.getCompanyName();  
 return ResponseEntity.ok(userInformation);  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
 @GetMapping("/{userId}/name")  
 public String getUserNameByID(@PathVariable("userId") Integer userId) {  
 try {  
 return userService.getUserNameByID(userId);  
 } catch (ResponseStatusException ex) {  
 return "not found";  
 }  
 }  
  
 @PostMapping("/create")  
 public ResponseEntity<String> createUser(@RequestBody UserDTO userDTO) {  
 try {  
 userService.createUser(userDTO);  
 return ResponseEntity.ok("User successfully created");  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
 @PostMapping("/enable")  
 public ResponseEntity<String> enableUser(@RequestBody UserDTO userDTO) {  
 try {  
 userService.changeUserEnabledField(userDTO.getUserId(),true);  
 return ResponseEntity.ok("User successfully enabled");  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
 @PostMapping("/disable")  
 public ResponseEntity<String> disableUser(@RequestBody UserDTO userDTO) {  
 try {  
 userService.changeUserEnabledField(userDTO.getUserId(), false);  
 return ResponseEntity.ok("User successfully disabled");  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
 @PostMapping("/change/name")  
 public ResponseEntity<String> changeUserName(@RequestBody UserDTO userDTO) {  
 try {  
 userService.changeUserName(userDTO.getUserId(),userDTO.getName());  
 return ResponseEntity.ok("User name successfully updated");  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
 @PostMapping("/change/email")  
 public ResponseEntity<String> changeUserEmail(@RequestBody UserDTO userDTO) {  
 try {  
 userService.changeUserEmail(userDTO.getUserId(),userDTO.getEmail());  
 return ResponseEntity.ok("User email successfully updated");  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
 @PostMapping("/change/company")  
 public ResponseEntity<String> changeUserCompany(@RequestBody UserDTO userDTO) {  
 try {  
 userService.changeUserCompany(userDTO.getUserId(),userDTO.getCompanyId());  
 return ResponseEntity.ok("User company successfully updated");  
 } catch (ResponseStatusException ex) {  
 String errorMessage = ex.getStatusCode() + " " + ex.getReason();  
 return ResponseEntity.status(ex.getStatusCode()).body(errorMessage);  
 }  
 }  
  
}