#  
# Build stage  
#  
FROM maven:3.6.3-openjdk-17 AS *build*COPY src /home/app/src  
COPY pom.xml /home/app  
RUN mvn -f /home/app/pom.xml clean package  
  
#  
# Package stage  
#  
FROM openjdk:17-jdk-slim  
  
COPY --from=*build* /home/app/target/PsychologistsMP-0.0.1-SNAPSHOT.jar /usr/local/lib/demo.jar  
EXPOSE 8080  
ENTRYPOINT ["java","-jar","/usr/local/lib/demo.jar"]

version: '3.1'  
  
services:  
  
 postgres:  
 image: 'postgres:latest'  
  
 restart: always  
 container\_name: containermarket  
  
 environment:  
 POSTGRES\_DB: dbnew\_2783  
 POSTGRES\_PASSWORD: post  
 POSTGRES\_USER: postgres  
  
  
 ports:  
 - '5432:5432'  
 networks:  
 - networkmarket  
  
  
 marketplase:  
 build:  
 context: .  
 dockerfile: Dockerfile  
 ports:  
 - '8080:8080'  
 depends\_on:  
 - postgres  
 networks:  
 - networkmarket  
  
  
networks:  
 networkmarket:  
 driver: bridge

spring.datasource.url=jdbc:postgresql://dpg-ck9gj16gtj9c739cr9dg-a.frankfurt-postgres.render.com:5432/dbnew\_2783  
spring.datasource.username=user  
spring.datasource.password=t4ndy7sz6JvhfGjkSTJzNd1ddmQoQKE7

package com.vvvital.psychologistsmp.dto;  
  
import com.vvvital.psychologistsmp.model.Categories;  
  
import java.util.Set;  
  
public class PsychologistUpdateDTO {  
 private UserRequestDTO userRequestDTO;  
 private Integer price;  
 private Integer rating;  
 private Integer experience;  
 private String description;  
 private String photoLink;  
 private Set<Categories> categories;  
  
 public UserRequestDTO getUserRequestDTO() {  
 return userRequestDTO;  
 }  
  
 public void setUserRequestDTO(UserRequestDTO userRequestDTO) {  
 this.userRequestDTO = userRequestDTO;  
 }  
  
 public Integer getPrice() {  
 return price;  
 }  
  
 public void setPrice(Integer price) {  
 this.price = price;  
 }  
  
 public Integer getRating() {  
 return rating;  
 }  
  
 public void setRating(Integer rating) {  
 this.rating = rating;  
 }  
  
 public Integer getExperience() {  
 return experience;  
 }  
  
 public void setExperience(Integer experience) {  
 this.experience = experience;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 public String getPhotoLink() {  
 return photoLink;  
 }  
  
 public void setPhotoLink(String photoLink) {  
 this.photoLink = photoLink;  
 }  
  
 public Set<Categories> getCategories() {  
 return categories;  
 }  
  
 public void setCategories(Set<Categories> categories) {  
 this.categories = categories;  
 }  
  
 public PsychologistUpdateDTO() {  
 }  
  
 public PsychologistUpdateDTO(UserRequestDTO userRequestDTO, Integer price,  
 Integer rating, Integer experience, String description,  
 String photoLink, Set<Categories> categories) {  
 this.userRequestDTO = userRequestDTO;  
 this.price = price;  
 this.rating = rating;  
 this.experience = experience;  
 this.description = description;  
 this.photoLink = photoLink;  
 this.categories = categories;  
 }  
}

**servise**

public void deleteUser(String email) {  
 User users = findByEmail(email);  
 userRepository.delete(users);  
}

public User updateUser(Long id, UserRequestDTO userRequestDTO) {  
 User existingUser = userRepository.findById(id).orElseThrow(() -> new EntityNotFoundException("User not found with id: " + id));  
  
 existingUser.setEmail(userRequestDTO.getEmail());  
 existingUser.setPassword(userRequestDTO.getPassword());  
 existingUser.setFirstName(userRequestDTO.getFirstName());  
 existingUser.setLastName(userRequestDTO.getLastName());  
 existingUser.setRole(userRequestDTO.getRole());  
 existingUser.setLocation(userRequestDTO.getLocation());  
  
 return userRepository.save(existingUser);  
 }  
  
 public User patchUser(Long id, UserRequestDTO userRequestDTO) {  
 User existingUser = userRepository.findById(id).orElseThrow(() -> new EntityNotFoundException("User not found with id: " + id));  
  
 if (userRequestDTO.getEmail() != null) {  
 existingUser.setEmail(userRequestDTO.getEmail());  
 }  
 if (userRequestDTO.getPassword() != null) {  
 existingUser.setPassword(userRequestDTO.getPassword());  
 }  
 if (userRequestDTO.getFirstName() != null) {  
 existingUser.setFirstName(userRequestDTO.getFirstName());  
 }  
 if (userRequestDTO.getLastName() != null) {  
 existingUser.setLastName(userRequestDTO.getLastName());  
 }  
 if (userRequestDTO.getRole() != null) {  
 existingUser.setRole(userRequestDTO.getRole());  
 }  
 if (userRequestDTO.getLocation() != null) {  
 existingUser.setLocation(userRequestDTO.getLocation());  
 }  
 return userRepository.save(existingUser);  
 }  
  
}

**controller**

@PutMapping("/{id}")  
 @Operation(summary = "Update user")  
 public ResponseEntity<UserResponseDTO> updateUser(@PathVariable Long id, @RequestBody UserRequestDTO userRequestDTO) {  
 User updatedUser = userService.updateUser(id, userRequestDTO);  
 UserResponseDTO responseDTO = userDTOMapper.userToUserResponseDTO(updatedUser);  
 return ResponseEntity.*ok*(responseDTO);  
 }  
  
 @PatchMapping("/{id}")  
 @Operation(summary = "Update user")  
 public ResponseEntity<UserResponseDTO> patchUser(@PathVariable Long id, @RequestBody UserRequestDTO userRequestDTO) {  
 User patchedUser = userService.patchUser(id, userRequestDTO);  
 UserResponseDTO responseDTO = userDTOMapper.userToUserResponseDTO(patchedUser);  
 return ResponseEntity.*ok*(responseDTO);  
 }  
}

@DeleteMapping("/delete/{email}")  
@Operation(summary = "Delete user")  
public ResponseEntity<User> delete(@PathVariable String email) {  
 logger.info("\*\*\*\*\*\*\*\*\* find by email {}", email);  
 User user = userService.findByEmail(email);  
 userService.deleteUser(email);  
 return ResponseEntity.*ok*(user);  
  
}