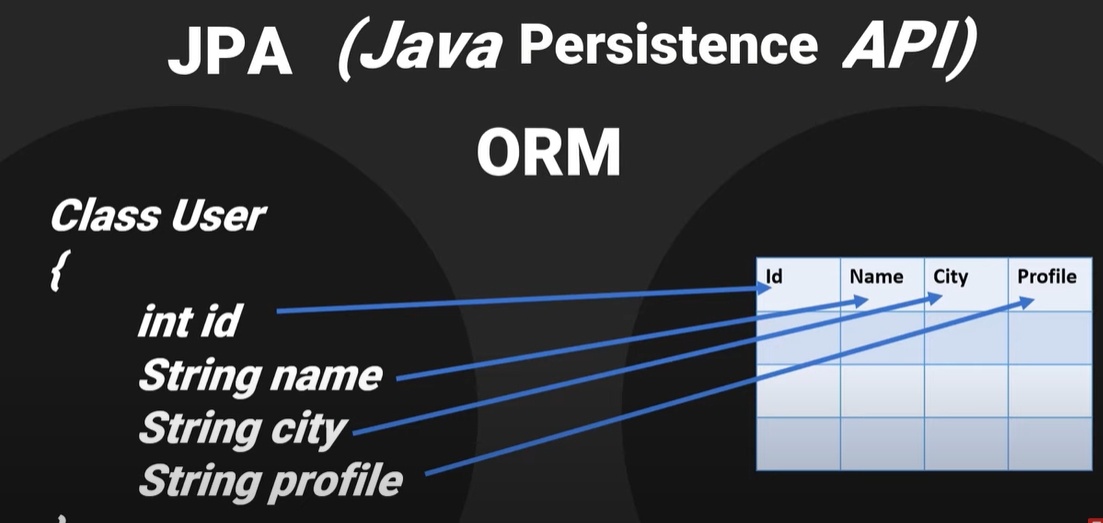
What is JPA?

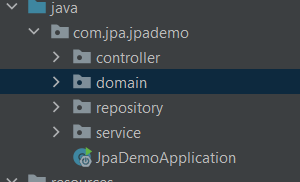
**Spring Boot JPA**is a Java specification for managing **relational** data in Java applications. It allows us to access and persist data between Java object/ class and relational database. JPA follows **Object-Relation Mapping**(ORM). It is a set of interfaces.



ORM maps a java class in to DBMS table, and all the variables of class to columns of table. So it maps class objects to DBMS table and vice versa automatically. As a developer we only need to provide the mapping information between java class and database.

**EntityManager interface** – It provides CRUD functionality, i.e. provide methods for Create, read, update and delete.

1. -Create a project with web, jpa and mysql dependencies.
2. Download it, extract it, and open with intelliJ.
3. Create the following packages –



1. – Define User class within domain –

package com.jpa.jpademo.domain;  
  
import javax.persistence.Entity;  
import javax.persistence.Id;  
  
@Entity  
public class User {  
 @Id  
 private String email;  
 private String password;  
 private String firstName;  
 private String lastName;  
  
 public User() {  
 }  
  
 public User(String email, String password, String firstName, String lastName) {  
 this.email = email;  
 this.password = password;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
}

1. – Define UserRepositroy interface within repository package –
2. package com.jpa.jpademo.repository;  
     
   import com.jpa.jpademo.domain.User;  
   import org.springframework.data.jpa.repository.JpaRepository;  
   import org.springframework.stereotype.Repository;  
     
   import java.util.List;  
     
   @Repository  
   public interface UserRepository extends JpaRepository<User, String > {  
     
    List<User> findByLastName(String lastName);  
   }
3. Define UserService interface in service package –

package com.jpa.jpademo.service;  
  
import com.jpa.jpademo.domain.User;  
  
import java.util.List;  
  
public interface UserService {  
  
 User saveUser(User user);  
 List<User> getAllUsers();  
 User updateUser(User user, String email);  
 boolean deleteUserByEmail(String email);  
 List<User> getUserByLastName(String lastName);  
}

8 – Define UserServiceImpl class -

package com.jpa.jpademo.service;  
  
import com.jpa.jpademo.domain.User;  
import com.jpa.jpademo.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class UserServideImpl implements UserService{  
  
 private final UserRepository userRepository;  
  
 @Autowired  
 public UserServideImpl(UserRepository userRepository){  
 this.userRepository= userRepository;  
 }  
 @Override  
 public User saveUser(User user) {  
 return userRepository.save(user);  
 }  
  
 @Override  
 public List<User> getAllUsers() {  
 return (List<User>) userRepository.findAll();  
 }  
  
 @Override  
 public User updateUser(User user, String email) {  
 Optional<User> optionalUser = userRepository.findById(email);  
 if (optionalUser.isEmpty()){  
 return null;  
 }  
 User existingUser = optionalUser.get();  
 if(user.getFirstName()!= null){  
 existingUser.setFirstName(user.getFirstName());  
 }  
 if (user.getLastName()!=null){  
 existingUser.setLastName(user.getLastName());  
 }  
 if (user.getPassword()!= null){  
 existingUser.setPassword(user.getPassword());  
 }  
 return userRepository.save(existingUser);  
 }  
  
 @Override  
 public boolean deleteUserByEmail(String email) {  
 userRepository.deleteById(email);  
 return true;  
 }  
  
 @Override  
 public List<User> getUserByLastName(String lastName) {  
 return userRepository.findByLastName(lastName);  
 }  
}

The [*Optional*](https://www.baeldung.com/java-optional) type was introduced in Java 8.  It provides a clear and explicit way to convey the message that there may not be a value, without using *null*.

When getting an *Optional* return type, we're likely to check if the value is missing, leading to fewer *NullPointerException*s in the applications.

1. – Define UserController in controller package –
2. package com.jpa.jpademo.controller;  
     
   import com.jpa.jpademo.domain.User;  
   import com.jpa.jpademo.service.UserService;  
   import org.springframework.beans.factory.annotation.Autowired;  
   import org.springframework.http.HttpStatus;  
   import org.springframework.http.ResponseEntity;  
   import org.springframework.web.bind.annotation.\*;  
     
   @RestController  
   @RequestMapping("/api/v1")  
   public class UserController {  
    private UserService userService;  
    @Autowired  
    public UserController(UserService userService){  
    this.userService = userService;  
    }  
     
    @PostMapping("/user")  
    public ResponseEntity<?> saveUser(@RequestBody User user){  
    return new ResponseEntity<>(userService.saveUser(user), HttpStatus.*CREATED*);  
    }  
     
    @GetMapping("/users")  
    public ResponseEntity<?> getAllUsers(){  
    return new ResponseEntity<>(userService.getAllUsers(), HttpStatus.*FOUND*);  
    }  
     
    @GetMapping("/users/{lastName}")  
    public ResponseEntity<?> getAllUsersByLastName(@PathVariable String lastName){  
     
    return new ResponseEntity<>(userService.getUserByLastName(lastName), HttpStatus.*FOUND*);  
    }  
     
    @DeleteMapping("/user/{email}")  
    public ResponseEntity<?> deleteUser(@PathVariable String email){  
    return new ResponseEntity<>(userService.deleteUserByEmail(email), HttpStatus.*OK*);  
    }  
     
    @PutMapping("/user/{email}")  
    public ResponseEntity<?> updateUser(@RequestBody User user, @PathVariable String email){  
    return new ResponseEntity<>(userService.updateUser(user, email), HttpStatus.*OK*);  
    }  
     
     
     
   }
3. – Update application.properties –

server.port=8081  
spring.datasource.url=jdbc:mysql://localhost:3306/test  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true

Now run the main class

11- (OPTIONAL) Using docker –

docker network create my-net

docker pull mysql

docker run -it --network my-net --name mysqlservice -e MYSQL\_ROOT\_PASSWORD=root -d mysql

docker exec -it mysqlservice bash

mysql> create database test

use test;