

Java Persistent API

Spring Boot 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. It also provides a runtime EntityManager API for processing queries and transactions on the objects against the database. It uses a platform- independent object-oriented query language JPQL

JPA is not a framework. It defines a concept that can be implemented by any framework.

Object-Relation Mapping (ORM)

In ORM, the mapping of Java objects to database tables, and vice-versa is called Object-Relational Mapping. The ORM mapping works as a bridge between a relational database (tables and records) and Java application (classes and objects).

Add the following Dependency:-

***Spring Web**

***Spring data JPA**

***MYSQL Driver**

MainController.java

```
package com.example.demo;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
@Controller
```

```
@RequestMapping(path="/save")
```

```
public class MainController {
```

```
    @Autowired
```

```
    private UserRepository userRepository;
```

```
    @PostMapping(path="/add")
```

```
    public @ResponseBody String addNewUser (@RequestParam String name
        , @RequestParam String email) {
```

```
        User n = new User();
```

```
        n.setName(name);
```

```
        n.setEmail(name);
```

```
        userRepository.save(n);
```

```
        return "Saved";
```

```
    }
```

```
    @GetMapping(path="/all")
```

```
    public @ResponseBody Iterable<User> getAllUsers() {
```

```
        return userRepository.findAll();
```

```
    }
```

```
}
```

Mysql1Application.java

```
package com.example.demo;

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

@SpringBootApplication

public class Mysql1Application {

    public static void main(String[] args) {
        SpringApplication.run(Mysql1Application.class, args);
    }

}
```

User.java

```
package com.example.demo;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity

public class User{

    @Id
```

```
@GeneratedValue(strategy=GenerationType.AUTO)
```

```
private Integer id;
```

```
private String name;
```

```
private String email;
```

```
public Integer getId() {  
    return id;  
}
```

```
public void setId(Integer id) {  
    this.id=id;  
}
```

```
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;  
}
```

```
}
```

UserRepository.java

```
package com.example.demo;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
public interface UserRepository extends CrudRepository<User, Integer> {
```

```
}
```

application.properties

```
spring.application.name=mysql-1
```

```
server.port = 4040
```

```
spring.jpa.hibernate.ddl-auto=update
```

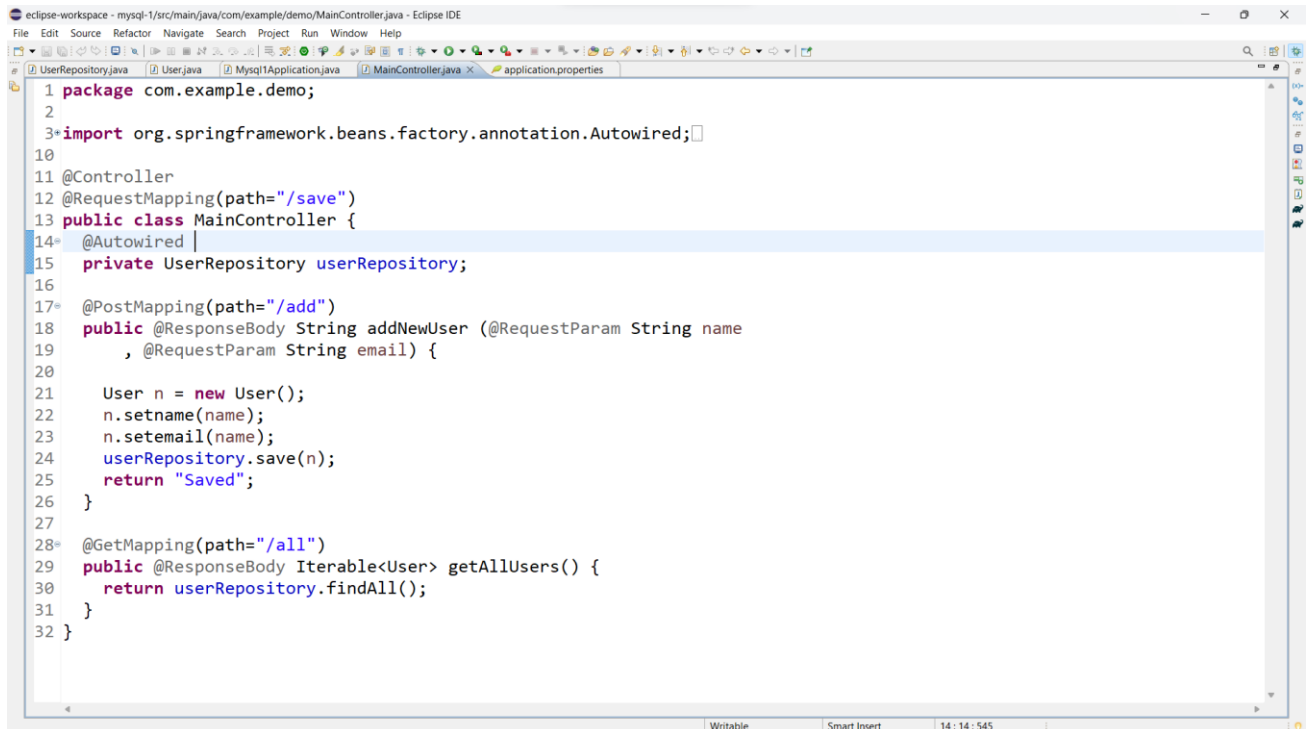
```
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3305/sonya
```

```
spring.datasource.username=root
```

```
spring.datasource.password=kishan
```

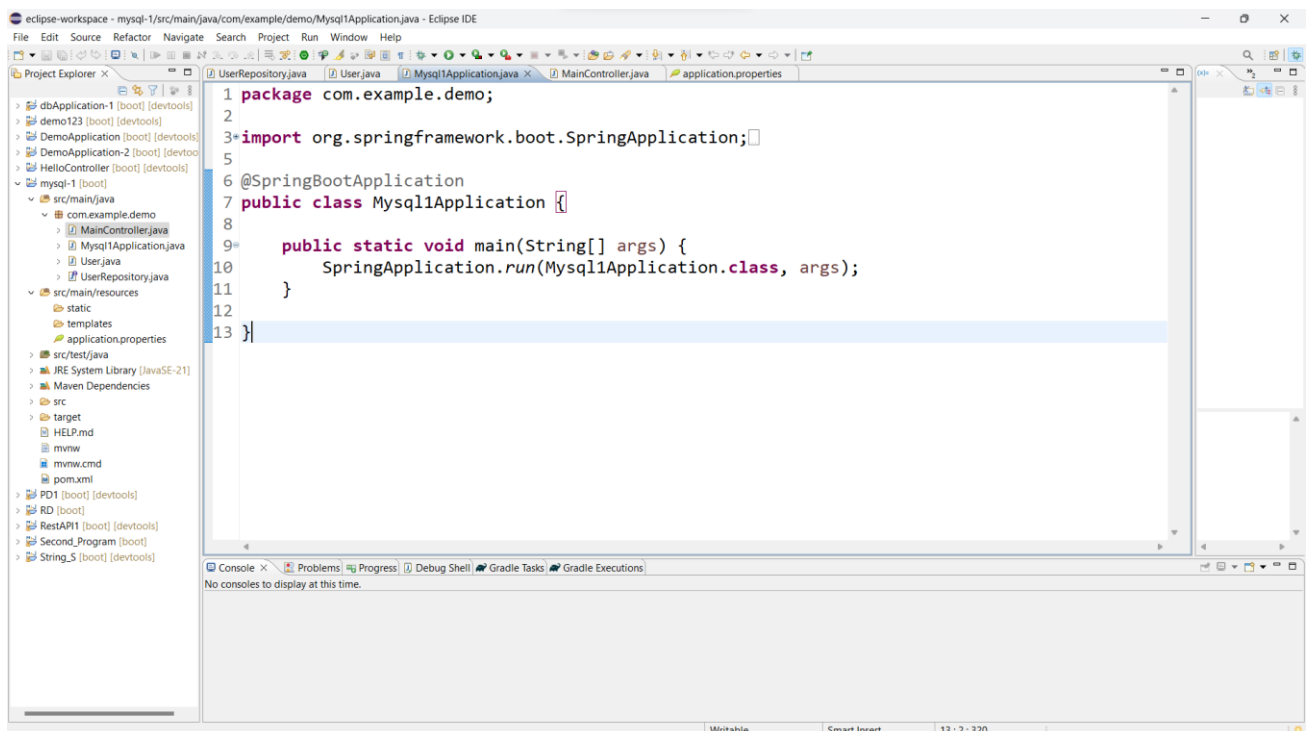
```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
#spring.jpa.show-sql: true
```



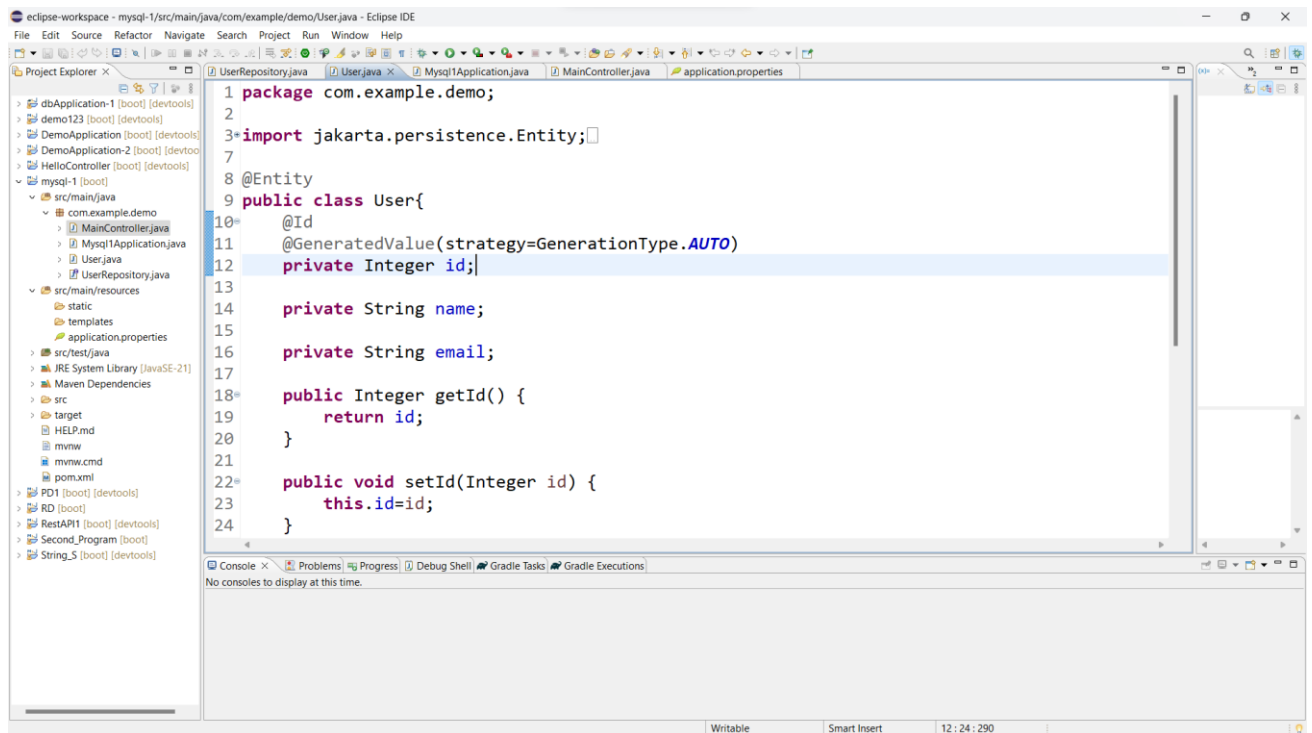
The screenshot shows the Eclipse IDE with the file `MainController.java` open. The code is a Spring MVC controller for a demo application. It includes package declarations, imports for Spring annotations, and two methods: `addNewUser` and `getAllUsers`. The `addNewUser` method uses `@PostMapping` and `@RequestBody` to handle a new user creation request. The `getAllUsers` method uses `@GetMapping` and `@ResponseBody` to retrieve all users. The IDE interface includes a menu bar, toolbar, and a status bar at the bottom.

```
1 package com.example.demo;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
10
11 @Controller
12 @RequestMapping(path="/save")
13 public class MainController {
14     @Autowired
15     private UserRepository userRepository;
16
17     @PostMapping(path="/add")
18     public @ResponseBody String addNewUser (@RequestParam String name
19         , @RequestParam String email) {
20
21         User n = new User();
22         n.setname(name);
23         n.setemail(name);
24         userRepository.save(n);
25         return "Saved";
26     }
27
28     @GetMapping(path="/all")
29     public @ResponseBody Iterable<User> getAllUsers() {
30         return userRepository.findAll();
31     }
32 }
```

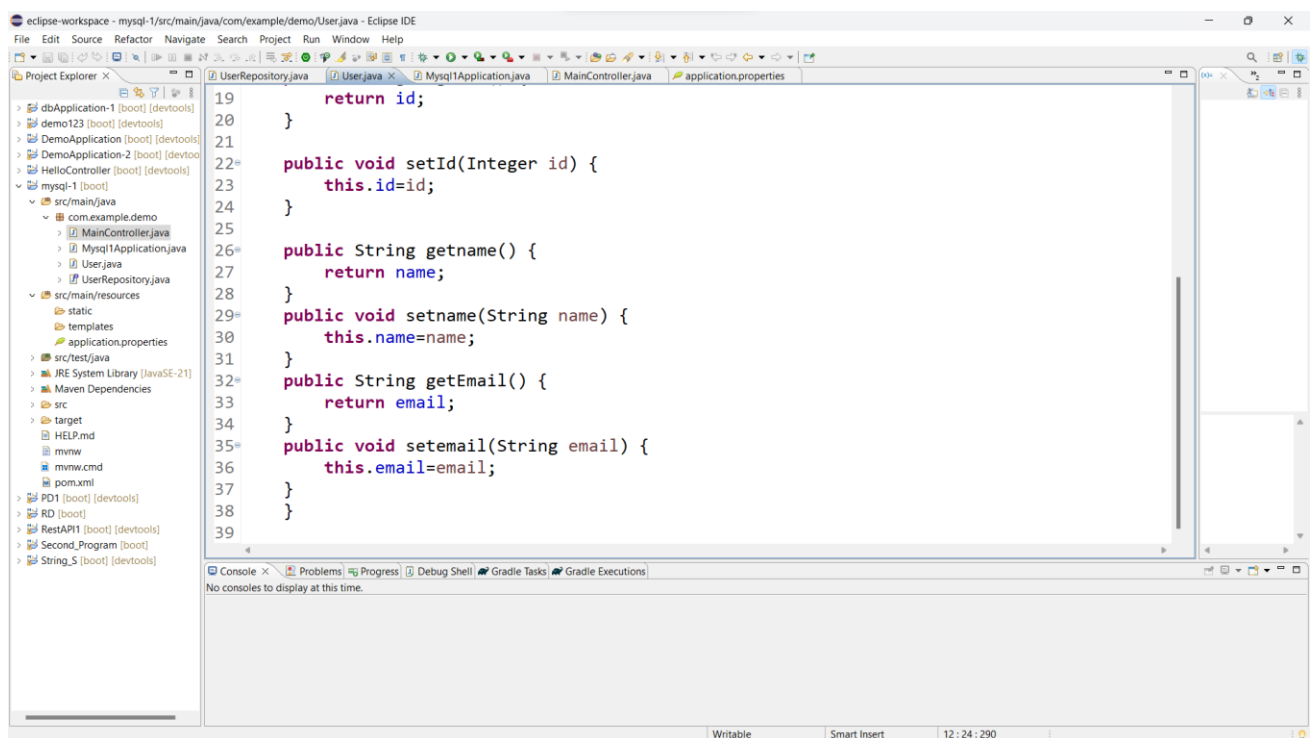


The screenshot shows the Eclipse IDE with the file `MySQL1Application.java` open. The code is a Spring Boot application class. It includes package declarations, imports for Spring Boot annotations, and a `main` method. The `main` method uses `@SpringBootApplication` and `SpringApplication.run` to launch the application. The IDE interface includes a menu bar, toolbar, and a status bar at the bottom. The Project Explorer on the left shows the project structure, including the `src/main/java` directory and the `com.example.demo` package.

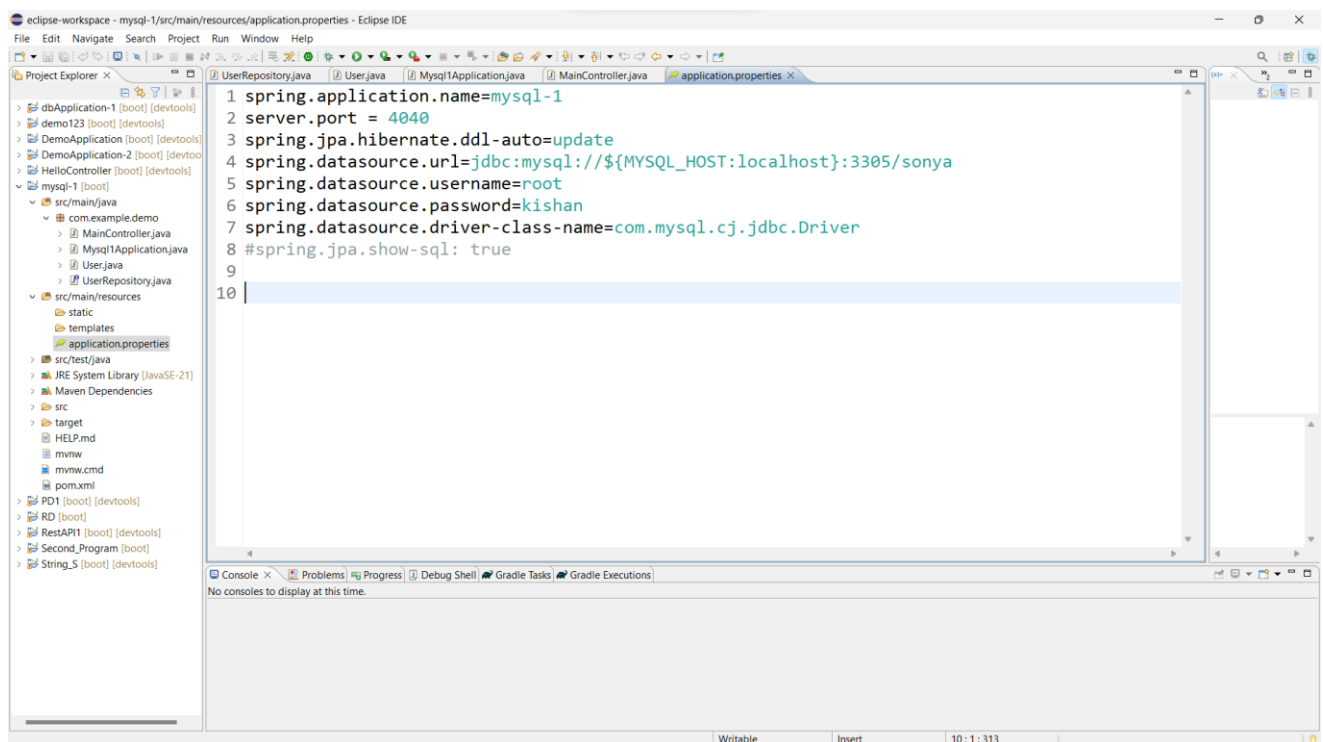
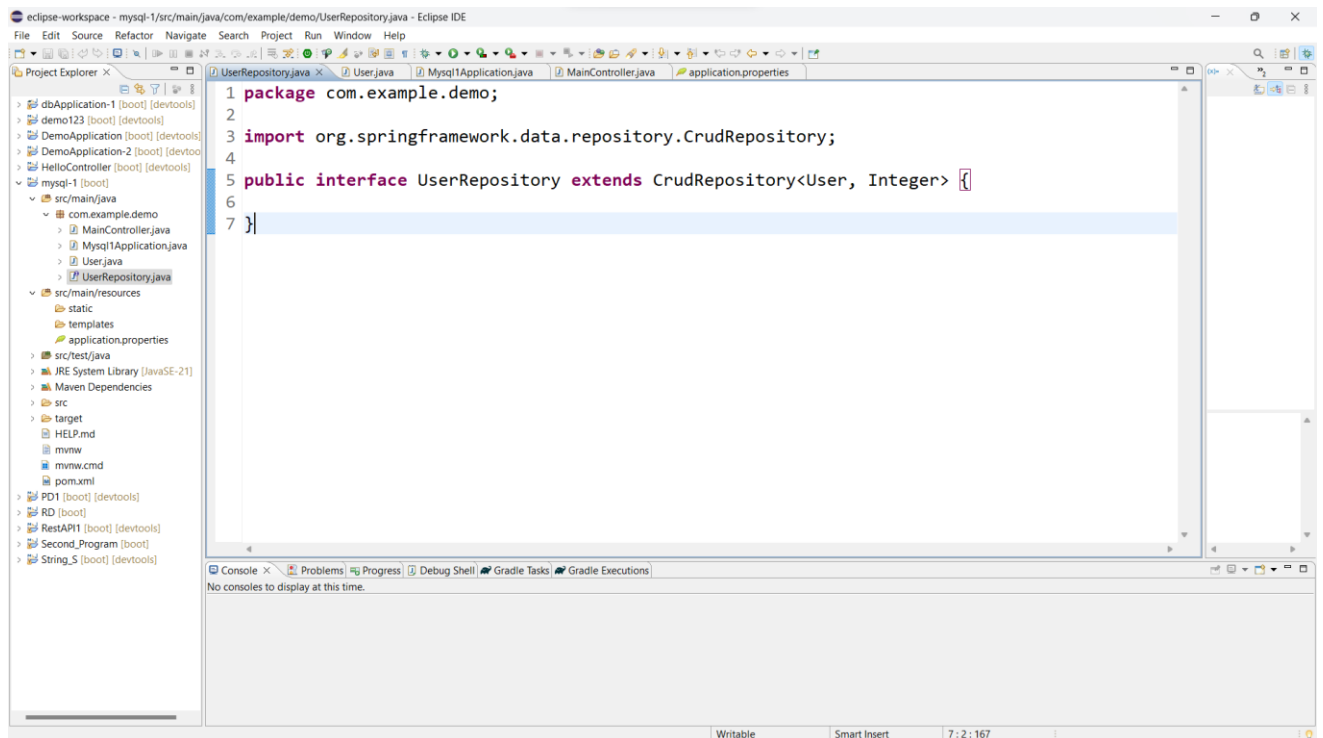
```
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class MySQL1Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MySQL1Application.class, args);
11     }
12
13 }
```



```
1 package com.example.demo;
2
3 import jakarta.persistence.Entity;
4
5
6
7
8 @Entity
9 public class User{
10     @Id
11     @GeneratedValue(strategy=GenerationType.AUTO)
12     private Integer id;
13
14     private String name;
15
16     private String email;
17
18     public Integer getId() {
19         return id;
20     }
21
22     public void setId(Integer id) {
23         this.id=id;
24     }
```



```
19     return id;
20 }
21
22 public void setId(Integer id) {
23     this.id=id;
24 }
25
26 public String getName() {
27     return name;
28 }
29 public void setName(String name) {
30     this.name=name;
31 }
32 public String getEmail() {
33     return email;
34 }
35 public void setEmail(String email) {
36     this.email=email;
37 }
38 }
39
```



Postman interface showing a POST request to `http://localhost:4040/save/add?id=1&name=Dinu&email=Dinu@gamil.com`. The request is configured with the following query parameters:

| Key | Value | Description |
|-------|----------------|-------------|
| id | 1 | |
| name | Dinu | |
| email | Dinu@gamil.com | |

The response is `200 OK` with a status of `426 ms` and `168 B`. The response body is `1 Saved`.

```
MySQL 8.0 Command Line Client
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use sonja;
Database changed
mysql> desc user;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int           | YES  |     | NULL    |       |
| email | varchar(255)  | YES  |     | NULL    |       |
| name  | varchar(255)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)

mysql> select * from user;
+-----+-----+-----+
| id  | email | name |
+-----+-----+-----+
| 1   | tom   | tom  |
| 2   | Ajju  | Ajju |
| 52  | Dinu  | Dinu |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

