

When the `@SpringBootApplication` class runs, it initializes the Spring container and scans for components, configurations, and beans defined in your application. Here's how it works step by step:

1. Component Scanning:

The `@SpringBootApplication` annotation enables component scanning by default. Spring scans the package containing the main application class and its sub-packages for classes annotated with `@Component`, `@Service`, `@Repository`, `@Controller`, or `@RestController`.

2. Bean Discovery:

During the scan, any method or class annotated with `@Bean` or a stereotype annotation like `@Component` is registered as a bean in the Spring context.

3. Dependency Injection:

Spring resolves dependencies by injecting the appropriate beans wherever required. For example, beans are injected into fields, constructor parameters, or methods annotated with `@Autowired` or via constructor injection.

4. Runtime Execution:

When your `SpringBootApplication` runs, it initializes the Spring Application Context and instantiates the discovered beans. The context manages their lifecycle, and they are ready for use.

If you had a simple `@Bean` that returned, say, a `String`, and a `@Component` or `@Service` that used it, Spring would wire them together and print the output when invoked.

```
1 package net.javaguides.springboot;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.boot.CommandLineRunner;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 import javax.sql.DataSource;
9 import java.sql.Connection;
10 import java.sql.SQLException;
11
12 @Configuration no usages ✘ kashanuk
13 public class DatabaseConnectionTest {
14
15     @Autowired 1 usage
16     private DataSource dataSource;
17
18     @Bean no usages ✘ kashanuk
19     public CommandLineRunner testConnection() {
20         return String[] args -> {
21             try (Connection connection = dataSource.getConnection()) {
22                 System.out.println("Connection Established: " + connection.getCatalog());
23             } catch (SQLException e) {
24                 System.err.println("Failed to establish connection: " + e.getMessage());
25             }
26         };
27     }
28 }
```

springboot-restful-webservices) × SpringbootRestfulWebservicesApplication.java × application.properties ×

```
~/Dowr 1 spring.datasource.url=jdbc:mysql://localhost:3306/user_management
        2 spring.datasource.username=root
        3 spring.datasource.password=Mysql@123
        4
        5 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
        6 spring.jpa.hibernate.ddl-auto=update
        7
```

```
7 import lombok.Setter;
8
9 @Getter
10 @Setter
11 @NoArgsConstructor
12 @AllArgsConstructor
13 @Entity
14 @Table(name = "users")
15 public class User {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20     @Column(name = "first_name", nullable = false)
21     private String firstName;
22     private String lastName;
23     private String email;
24 }
```

```
7 import lombok.AllArgsConstructor;
8 import lombok.Getter;
9 import lombok.NoArgsConstructor;
10 import lombok.Setter;
11
12 @Getter
13 @Setter
14 @NoArgsConstructor
15 @AllArgsConstructor
16 @Entity
17 @Table(name = "users")
18 public class User {
19
20     @Id
21     @GeneratedValue(strategy = GenerationType.AUTO)
22     private Long id;
23     private String firstName;
24     private String lastName;
25     private String email;
```

The screenshot shows an IDE interface with Java code for a User entity. A tooltip is displayed over the `strategy` parameter of the `@GeneratedValue` annotation at line 21. The tooltip lists several options for the `GenerationType` enum, each with its package name and a small icon: `GENERATIONTYPE.AUTO` (jakarta.persistence), `IDENTITY` (jakarta.persistence), `jakarta.persistence`, `TABLE` (jakarta.persistence), `SEQUENCE` (jakarta.persistence), `GeneratedValue` (jakarta.persistence), `lombok`, `javax.annotation.processing`, `jakarta.annotation`, `org.hibernate.annotations`, `GenericArrayType` (java.lang.reflect), and `ConcurrentDAOAnnotation` (java.lang.reflect). The `AUTO` option is highlighted with a red border. A tip at the bottom of the tooltip states: '^↓ and ^↑ will move caret down and up in the editor' and includes a link to 'Next Tip'.

Project

The screenshot shows the IntelliJ IDEA interface with the 'Project' tool window open. The project structure is as follows:

- springboot-restful-webservices** (~/D...)
 - .idea
 - .mvn
 - src
 - main
 - java
 - net.javaguides.springboot
 - controller
 - entity
 - User
 - repository
 - UserRepository
 - service
 - SpringbootRestfulWeb
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

```
1  /.../
2
3  package org.springframework.data.jpa.repository;
4
5  import ...
6
7
8  @RepositoryBean
9  public interface JpaRepository<T, ID> extends CrudRepository<T, ID>, PagingAndSortingRepository<T, ID> {
10    List<T> findAll();
11
12    List<T> findAll(Sort sort);
13
14    List<T> findAllById(Iterable<ID> ids);
15
16    <S extends T> List<S> saveAll(Iterable<S> entities);
17
18    void flush();
19
20    <S extends T> S saveAndFlush(S entity);
21
22    <S extends T> List<S> saveAllAndFlush(Iterable<S> entities);
23
24
25
26
27
28
29
30
31
32  /** @deprecated */
```

The image shows two side-by-side application windows. The left window is a database management tool with a toolbar, a query editor containing a SELECT statement, and a result grid showing a single row of data. The right window is a REST client tool showing a POST request to a local API endpoint, the response body, and the resulting created user object.

Left Window (Database Management System):

- Toolbar:** Database, Server, Tools, Scripting, Help.
- Query Editor:** Query 1, users, contains the SQL statement: `SELECT * FROM user_management.users;`
- Result Grid:** Shows a single row of data:

	id	email	first_name	last_name
▶	1	joesmith@gmail.com	Joe	Smith
*	NULL	NULL	NULL	NULL
- Right Sidebar:** Includes icons for Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.

Right Window (REST Client):

- Header Bar:** Workspaces, More, Upgrade, Search, User, Bell, No environment.
- Request Section:** Method: POST, URL: http://localhost:8080/api/users, Headers: 9, Body (selected), Scripts, Settings.
- Body Section:** Raw JSON input:

```
1 {  
2   "firstName": "Joe",  
3   "lastName": "Smith",  
4   "email" : "joesmith@gmail.com"  
5 }
```
- Response Section:** Status: 201 Created, Time: 351 ms, Size: 243 B, Save Response.
- Response Body:** JSON output:

```
1 {  
2   "id": 1,  
3   "firstName": "Joe",  
4   "lastName": "Smith",  
5   "email": "joesmith@gmail.com"  
6 }
```
- Bottom Navigation:** Action Output, Console, Postbot, Runner, Vault, Help.

```
import org.springframework.stereotype.Service;
import java.util.Optional;
@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private UserRepository userRepository;
    @Override
    public User createUser(User user) {
        return userRepository.save(user);
    }
    @Override
    public User getUserId(Long userId) {
        Optional<User> optionalUser = userRepository.findById(userId);
        return optionalUser.get();
    }
}
```

```
private UserService userService;

// build create User REST API
@PostMapping
public ResponseEntity<User> createUser(@RequestBody User user){
    User savedUser = userService.createUser(user);
    return new ResponseEntity<>(savedUser, HttpStatus.CREATED);
}

// build get user by id REST API
// http://localhost:8080/api/users/1
@GetMapping("{id}")
public ResponseEntity<User> getUserId(@PathVariable("id") Long userId){
    User user = userService.getUserId(userId);
    return new ResponseEntity<User>(user, HttpStatus.OK);
}
```

URI TEMPLATE VARIABLE

My Workspace

New Import < New E Rest-L POST crea GET fetch PUT upda DEL Delel GET Untitl POST http POST Pos GET http:/ > + No environment

Collections + ...

Environments > Rest-Demo1
Rest-DemoDB 1

History

POST Post

HTTP http://localhost:8080/api/users/1 4

GET http://localhost:8080/api/users/1 2

Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (5) Test Results 200 OK 278 ms 238 B

3 { } JSON ▾ Preview Visualize

```
1 {
2   "id": 1,
3   "firstName": "Joe",
4   "lastName": "Smith",
5   "email": "joesmith@gmail.com"
6 }
```

http://localhost:8080/api/users/2

Save



GET

http://localhost:8080/api/users/2

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (5)

Test Results



200 OK

24 ms

234 B

Save Response

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "id": 2,  
3   "firstName": "tom",  
4   "lastName": "cruise".
```

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure is displayed under the **springboot-restful-webservices** project. It includes:
 - src**: Contains **main** and **test**.
 - main** contains **java**, which has **net.javaguides.springboot** (containing **controller**, **entity**, **repository**, **service**, and **impl**), **resources** (containing **static**, **templates**, and **application.properties**), and **SpringbootRestfulWebservice**.
 - resources** contains **static**, **templates**, and **application.properties**.
- Code Editor:** The right pane shows the **UserService.java** file with the following code:

```
package net.javaguides.springboot.service;

import net.javaguides.springboot.entity.User;

public interface UserService {
    User createUser(User user);

    User getUserById(Long userId);
}
```
- Annotations:** Red callout numbers indicate specific parts of the code:
 - 1**: Points to the **SpringbootRestfulWebservice** class.
 - 2**: Points to the **UserService** interface.
 - 3**: Points to the **UserController** class within the **controller** package.
- Text Overlay:** A large red text overlay provides a summary of the design pattern:

remember Service will have
business logic, controller is
responsible for api calls,
repository has database logic
to fetch the data from db
directly

UserService.java

```
1 package net.javaguides.springboot.service;
2
3 import net.javaguides.springboot.entity.User;
4
5 import java.util.List;
6
7 public interface UserService {
8     User createUser(User user);
9
10    User getUserById(Long userId);
11
12    List<User> getAllUsers();
13 }
14
```

1 related problem

acting as method enforcer to impl class, lets go to impl

```
22     User savedUser = userService.createUser(user);
23     return new ResponseEntity<>(savedUser, HttpStatus.CREATED);
24 }
25
26 // build get user by id REST API
27 // http://localhost:8080/api/users/1
28 @GetMapping("{id}")
29 public ResponseEntity<User> getUserById(@PathVariable("id") Long userId){
30     User user = userService.getUserById(userId);
31     return new ResponseEntity<>(user, HttpStatus.OK);
32 }
33
34 // Build Get All Users REST API
35 // http://localhost:8080/api/users
36 @GetMapping
37 public ResponseEntity<List<User>> getAllUsers(){
38     List<User> users = userService.getAllUsers();
39     return new ResponseEntity<>(users, HttpStatus.OK);
40 }
```

in the controller

```
14 public class UserServiceImpl implements UserService {  
15     1  
16     private UserRepository userRepository;  
17  
18     @Override  
19     public User createUser(User user) { return userRepository.save(user); }  
20  
21     @Override  
22     public User getUserById(Long userId) {  
23         Optional<User> optionalUser = userRepository.findById(userId);  
24         1  
25         return optionalUser.get();  
26     }  
27  
28     2  
29     @Override  
30     public List<User> getAllUsers() {  
31         = return userRepository.findAll();  
32     }  
33 }
```

1 GET ▾ http://localhost:8080/api/users 2

3 Send ▾

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

s,

Body Cookies Headers (5) Test Results

200 OK 175 ms 315 B Save Response ▾

Pretty Raw Preview Visualize JSON ▾

1 [2 { 3 "id": 1, 4 "firstName": 5 "lastName": "

4

The screenshot shows the Postman application interface. At the top, there's a red circle with the number 1 over the 'GET' method and URL field. A red circle with the number 2 is over the 'Headers (6)' tab. Another red circle with the number 3 is over the 'Send' button. A red circle with the number 4 is over the first item in the JSON response body, which is a user object with id 1.

you can
post 1
more
user and
then call
that user
using get
all or
specific
id

UserService.java

```
1 package net.javaguides.springboot.service;
2
3 import net.javaguides.springboot.entity.User;
4
5 import java.util.List;
6
7 public interface UserService {
8     User createUser(User user);
9
10    User getUserById(Long userId);
11
12    List<User> getAllUsers();
13
14    User updateUser(User user);
15 }
16
```

1 related problem

1

2

```
23  
24     @Override  
25     public User getUserById(Long userId) {  
26         Optional<User> optionalUser = userRepository.findById(userId);  
27         return optionalUser.get();  
28     }  
29  
30     @Override  
31     public List<User> getAllUsers() {  
32         return userRepository.findAll();  
33     }  
34  
35     @Override  
36     public User updateUser(User user) {  
37         return null;  
38     }  
39 }  
40
```

guess which class
it is
UserServiceImpl

first we will get the
use and then will do
the update

```
27     public User getUserById(Long userId) {  
28         Optional<User> optionalUser = userRepository.findById(userId);  
29         return optionalUser.get();  
30     }  
31  
32     @Override  
33     public List<User> getAllUsers() {  
34         return userRepository.findAll();  
35     }  
36  
37     @Override  
38     public User updateUser(User user) {  
39         User existingUser = userRepository.findById(user.getId()).get();  
40         existingUser.setFirstName(user.getFirstName());  
41         existingUser.setLastName(user.getLastName());  
42         existingUser.setEmail(user.getEmail());  
43         return null;  
44     }  
}
```

updating from the user object
to existing user object to
bring it available for update

```
27     public User getUserById(Long userId) {  
28         Optional<User> optionalUser = userRepository.findById(userId);  
29         return optionalUser.get();  
30     }  
31  
32     @Override  
33     public List<User> getAllUsers() {  
34         return userRepository.findAll();  
35     }  
36  
37     @Override  
38     public User updateUser(User user) {  
39         User existingUser = userRepository.findById(user.getId()).get();  
40         existingUser.setFirstName(user.getFirstName());  
41         existingUser.setLastName(user.getLastName());  
42         existingUser.setEmail(user.getEmail());  
43         return null;  
44     }  
45 }
```

next we are going to save this user into existing database and recall which layer is responsible for saving in the database ?

repository

```
30     User user = userService.getUserById(userId),  
31     return new ResponseEntity<>(user, HttpStatus.OK);  
32 }  
  
33 // Build Get All Users REST API  
34 // http://localhost:8080/api/users  
35 @GetMapping  
36 public ResponseEntity<List<User>> getAllUsers(){  
37     List<User> users = userService.getAllUsers();  
38     return new ResponseEntity<>(users, HttpStatus.OK);  
39 }  
  
40 // Build Update User REST API  
41 // http://localhost:8080/api/users/  
42 @PutMapping  
43 public ResponseEntity<User> updateUser(User user){  
44     User updatedUser = userService.updateUser(user);  
45     return new ResponseEntity<>(updatedUser, HttpStatus.OK);  
46 }  
47 }
```

2

can you guess
what we are
missing here ?

1

Now in controller we are
going to call the method to
update user and then
controller will let the user
know using http response
that is updated

now we will save it
into the database
and return the
updated user as
this method
returns the User
Object

```
27     public User getUserById(Long userId) {  
28         Optional<User> optionalUser = userRepository.findById(userId);  
29         return optionalUser.get();  
30     }  
31  
32     @Override  
33     public List<User> getAllUsers() {  
34         return userRepository.findAll();  
35     }  
36  
37     @Override  
38     public User updateUser(User user) {  
39         User existingUser = userRepository.findById(user.getId()).get();  
40         existingUser.setFirstName(user.getFirstName());  
41         existingUser.setLastName(user.getLastName());  
42         existingUser.setEmail(user.getEmail());  
43         User updatedUser = userRepository.save(existingUser);  
44         return updatedUser; |
```

```
gt
30     user = userService.getUserById(userId);
31     return new ResponseEntity<>(user, HttpStatus.OK);
32 }
33
34 // Build Get All Users REST API
35 // http://localhost:8080/api/users
36
37 /**
38 * @GetMapping("/users")
39 * @CrossOrigin(origins = "http://localhost:4200")
40 */
41
```

@PutMapping annotation is used to map HTTP PUT request onto specific handler method.

```
    ser>> getAllUsers(){
        service.getAllUsers();
        return new ResponseEntity<>(users, HttpStatus.OK);
    }
```

```
41
42     // Build Update User REST API
43     @PutMapping("/")
44     public ResponseEntity<User> updateUser(User user){
45         User updatedUser = userService.updateUser(user);
46         return new ResponseEntity<>(updatedUser, HttpStatus.OK);
47     }
48 }
```

```
gt
35 // http://localhost:8080/api/users
36 @GetMapping
37 public ResponseEntity<List<User>> getAllUsers(){
38     List<User> users = userService.getAllUsers();
39     return new ResponseEntity<>(users, HttpStatus.OK);
40 }
41
42 // Build Update User REST API
43 @PutMapping("{id}")
44 // http://localhost:8080/api/users/1
45 public ResponseEntity<User> updateUser(User user){
46     User updatedUser = userService.updateUser(user);
47     return new ResponseEntity<>(updatedUser, HttpStatus.OK);
48 }
49
50 }
```

remember we did pathvariable same needs to
be done to refresh

//localhost:8080/api/users

ping

```
 ResponseEntity<List<User>> users = userService.getUsers();
    return new ResponseEntity<>(users, HttpStatus.OK);
```

Update User REST API

ping("{id}")

//localhost:8080/api/user/{id}

```
 ResponseEntity<User> updateUser(@PathVariable("id") Long userId,
    @RequestBody User user){
    updatedUser = userService.updateUser(user);
    return new ResponseEntity<>(updatedUser, HttpStatus.OK);
```

The screenshot shows a POSTMAN interface. At the top, it says "PUT" and "http://localhost:8080/api/users/1". Below that, there are tabs for "Params", "Authorization", "Headers (8)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is selected, showing "raw" selected. The JSON body is:

```
1 {
2     "firstName": "ram",
3     "lastName": "jadav",
4     "email": "ram@gmail.com"
5 }
```

Below the body, there are tabs for "Body", "Cookies", "Headers (5)", and "Test Results". Under "Test Results", it shows a 200 OK response with 317 ms and 234 B. The response body is:

```
1 {
2     "id": 1,
3     "firstName": "ram",
4     "lastName": "jadav",
5     "email": "ram@gmail.com"
6 }
```

convert json
user object
to java
object

va > net > javaguides > springboot > controller > UserController > m updateUser

SpringbootRestfulWebservicesApplication

UserServiceImpl.java Objects.java Strings.class String.java UserController.java JpaRepository.class

Project Maven

```
public ResponseEntity<User> updateUser(@PathVariable("id") Long userId,
                                         @RequestBody User user) {
    User updatedUser = userService.updateUser(user);
    return new ResponseEntity<User>(updatedUser, HttpStatus.OK);
}
```

Structure

Bookmarks

Project Structure Marks

needed to add this because in update user we are using id to update

```
va > net > javaguides > springboot > controller > UserController > updateUser
```

```
SpringbootRestfulWebservicesApplication
```

```
UserServiceImpl.java Objects.java Strings.class String.java UserController.java JpaRepository.class
```

```
public ResponseEntity<User> getUserById(@PathVariable("id") Long userId) {
    User user = userService.getUserById(userId);
    return new ResponseEntity<>(user, HttpStatus.OK);
}

// Build Get All Users REST API
// http://localhost:8080/api/users
@GetMapping
public ResponseEntity<List<User>> getAllUsers(){
    List<User> users = userService.getAllUsers();
    return new ResponseEntity<>(users, HttpStatus.OK);
}

// Build Update User REST API
@PutMapping("{id}")
// http://localhost:8080/api/users/1
public ResponseEntity<User> updateUser(@PathVariable("id") Long userId,
                                         @RequestBody User user){
    1 user.setId(userId);
    User updatedUser = userService.updateUser(user);
    return new ResponseEntity<>(updatedUser, HttpStatus.OK);
}
```

Project Structure Marks

2

```
et > javaguides > springboot > service > impl > UserServiceImpl > updateUser
```

```
SpringbootRestfulWebservicesApplication
```

```
UserServiceImpl.java Objects.java Strings.class String.java UserController.java JpaRepo
```

```
public User getUserById(Long userId) {
    Optional<User> optionalUser = userRepository.findById(userId);
    return optionalUser.get();
}

@Override
public List<User> getAllUsers() {
    return userRepository.findAll();
}

@Override
public User updateUser(User user) {
    2 User existingUser = userRepository.findById(user.getId()).get();
    existingUser.setFirstName(user.getFirstName());
    existingUser.setLastName(user.getLastName());
    existingUser.setEmail(user.getEmail());
    User updatedUser = userRepository.save(existingUser);
    return updatedUser;
}
```

Local instance 3306

Administration Schemas

SCHEMAS

Filter objects

myblog
myblogapp
registration_db
search_api
sms
sys
transaction_demo
user_management

Tables
users
Views
Stored Procedures
Functions

wikimedia

Object Info Session

Table: **users**

Columns:

id	bigint AI PK	
email	varchar(255)	
first_name	varchar(255)	
last_name	varchar(255)	

Query 1 users users users users users

Limit to 1000 rows

1 • `SELECT * FROM user_management.users;`

100% 1:1

Result Grid Filter Rows: Search Edit: Export/Import: Result Grid

	id	email	first_name	last_name
▶	1	ramesh@gmail.com	Ramesh	Fadatare
	2	tom@gmail.com	tom	cruise
	3	umesh@gmail.com	umesh	fadatare
	HULL	HULL	HULL	HULL

Action Output

	Time	Action	Response	Duration / Fetch Time
1	13:04:31	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00067 sec / 0.0000...
2	13:05:06	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00038 sec / 0.000...

Apply Revert

Postman

Home Workspaces Reports Explore Search Postman Invite Settings Bell Upgrade

Spring Boot Unit Testing New Import ↺ POST h. GET h. POST h. GET h. PUT h. ↻ No Environment ↻

collections + ☰ ... 1

PUT http://localhost:8080/api/users/1 2 4

Params Authorization Headers (8) Body 3 Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

1 "firstName": "ram",
2 "lastName": "jadhav",
3 "email": "ram@gmail.com"
4
5

You don't have any collections 2

Collections let you group related requests, making them easier to access and run.

Create Collection

Body Cookies Headers (5) Test Results 200 OK 317 ms 234 B Save Response

Pretty Raw Preview Visualize JSON 5

1 "id": 1,
2 "firstName": "ram",
3 "lastName": "jadhav",
4 "email": "ram@gmail.com"
5
6

Local instance 3306

Administration Schemas

SCHEMAS

Filter objects

myblog
myblogapp
registration_db
search_api
sms
sys
transaction_demo
user_management

Tables
users
Views
Stored Procedures
Functions

wikimaria

Object Info Session

Table: **users**

Columns:

id	bigint AI PK
email	varchar(255)
first_name	varchar(255)
last_name	varchar(255)
1	ram@gmail.com
2	tom@gmail.com
3	umesh@gmail.com
HULL	HULL

Action Output

	Time	Action	Response	Duration / Fetch Time
1	13:04:31	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00067 sec / 0.000...
2	13:05:06	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00038 sec / 0.000...
3	13:06:35	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00060 sec / 0.000...

Query 1 users users users users users users

Limit to 1000 rows

1 • **SELECT * FROM user_management.users;**

Result Grid

Filter Rows: Search Edit: Export/Import:

id	email	first_name	last_name
1	ram@gmail.com	ram	jadhav
2	tom@gmail.com	tom	cruise
3	umesh@gmail.com	umesh	fadatare
HULL	HULL	HULL	HULL

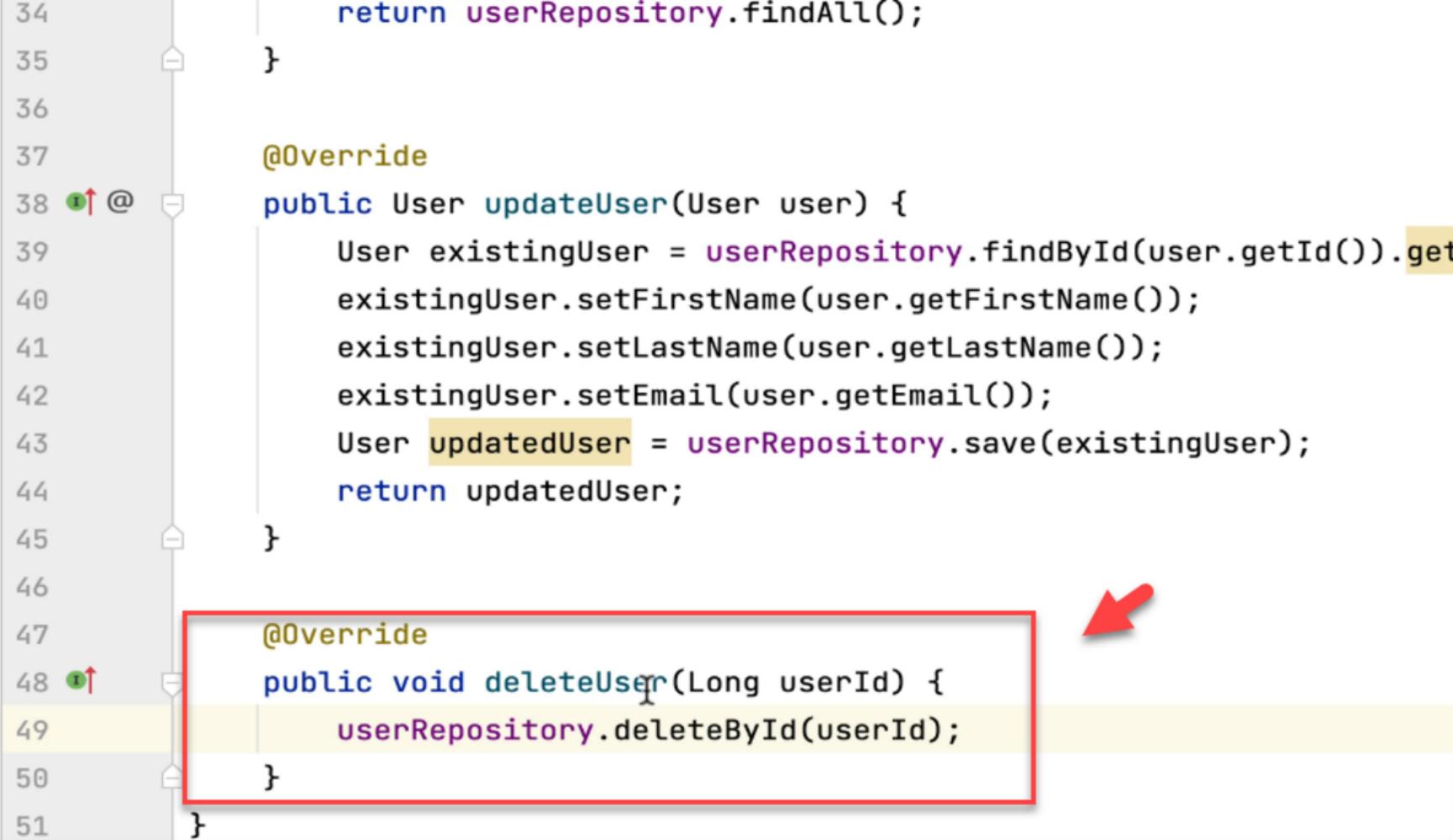
Result Grid Form Editor Field Types

avaguides.springboot
controller
UserController
entity
User
repository
UserRepository
service
impl
UserServiceimpl
UserService
pringbootRestfulWebs
es
lates

```
3 import net.javaguides.springboot.entity.User;
4
5 import java.util.List;
6
7 1 related problem
8
9
10
11
12
13
14
15
16 void deleteUser(Long userId); 1
17 }
18
```

avaguides.springboot
controller
UserController
entity
User
repository
UserRepository
service
impl
UserServiceImp
UserService
pringbootRestfulWe
es
c
lates

```
34             return userRepository.findAll();
35     }
36
37     @Override
38     public User updateUser(User user) {
39         User existingUser = userRepository.findById(user.getId()).get();
40         existingUser.setFirstName(user.getFirstName());
41         existingUser.setLastName(user.getLastName());
42         existingUser.setEmail(user.getEmail());
43         User updatedUser = userRepository.save(existingUser);
44         return updatedUser;
45     }
46
47     @Override
48     public void deleteUser(Long userId) {
49         userRepository.deleteById(userId);
50     }
51 }
```



des.springboot
er
Controller
ry
Repository

UserServiceImp
Service
ootRestfulWe

properties

```
41  
42     // Build Update User REST API  
43     @PutMapping("{id}")  
44     // http://localhost:8080/api/users/1  
45     public ResponseEntity<User> updateUser(@PathVariable("id") Long userId,  
46                                         @RequestBody User user){  
47         user.setId(userId);  
48         User updatedUser = userService.updateUser(user);  
49         return new ResponseEntity<>(updatedUser, HttpStatus.OK);  
50     }  
51  
52     // Build Delete User REST API  
53     @DeleteMapping("{id}")  
54     public ResponseEntity<String> deleteUser(@PathVariable("id") Long userId){  
55         userService.deleteUser(userId);  
56         return new ResponseEntity<>(body: "User successfully deleted!", HttpStatus.OK);  
57     }  
58 }  
59
```

ort ← ● GET h. ● POST h. ● GET h. ● PUT h. ● DEL h. ● → + ... No Environment

... http://localhost:8080/api/users/1 Save ↴ ⌂

DELETE http://localhost:8080/api/users/1 Send ↴

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK 149 ms 190 B Save Response ↴

Pretty Raw Preview Visualize Text ↴

1 User successfully deleted!

Udemy

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree, with 'user_management' selected and its 'Tables' node expanded, showing 'users' as the selected table. The main area shows a 'Result Grid' for the query `SELECT * FROM user_management.users;`. The grid has columns: id, email, first_name, and last_name. The data shows three rows: (id=2, tom@gmail.com, tom, cruise), (id=3, umesh@gmail.com, umesh, faulstare), and a blank row (id=1). A red arrow points to the first row (id=2). To the right of the grid, large red text reads 'id = 1 is deleted'. The bottom section shows the 'Action Output' tab with a history of four SELECT queries, all returning 3 row(s) returned.

	Time	Action	Response	Duration / Fetch Time
✓ 1	13:04:31	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00067 sec / 0.000...
✓ 2	13:05:06	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00038 sec / 0.000...
✓ 3	13:06:35	SELECT * FROM user_management.users LIMIT 0, 1000	3 row(s) returned	0.00060 sec / 0.000...
✓ 4	13:12:02	SELECT * FROM user_management.users LIMIT 0, 1000	2 row(s) returned	0.00058 sec / 0.000...