

In this post, we will learn how to create REST API with Spring Boot, JPA, Hibernate, and MySQL.

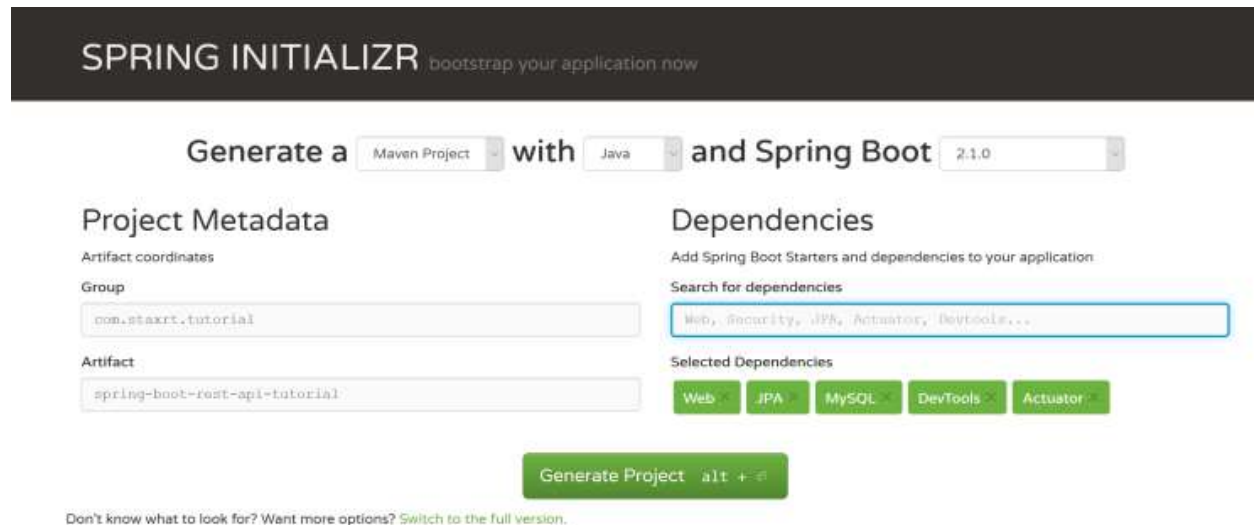
1. Create the Spring Boot Project.
2. Define Database configurations.
3. Create an Entity Class.
4. Create JPA Data Repository layer.
5. Create Rest Controllers and map API requests.
6. Create Unit Testing for API requests and run the unit testing.
7. Build and run the Project.

## Requirement

We need to create a simple REST API to store user data to [MySQL database](#) with that API so we can create, update, delete, and get users information.

## Create the Spring Boot Project

First, go to [Spring Initializr](#) and create a project with below settings



The screenshot shows the Spring Initializr web application interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there are dropdown menus to "Generate a Maven Project with Java and Spring Boot 2.1.0". The interface is divided into two main sections: "Project Metadata" and "Dependencies".

**Project Metadata:**

- Artifact coordinates:**
  - Group:** com.staxrt.tutorial
  - Artifact:** spring-boot-rest-api-tutorial

**Dependencies:**

- Add Spring Boot Starters and dependencies to your application:**
- Search for dependencies:** Web, Security, JPA, Actuator, DevTools...
- Selected Dependencies:** Web, JPA, MySQL, DevTools, Actuator

At the bottom, there is a green button labeled "Generate Project alt + ⌘". Below the button, a small text link says "Don't know what to look for? Want more options? Switch to the full version."

**Web** — Full-stack web development with Tomcat and [Spring MVC](#)

**DevTools** — Spring Boot Development Tools

**JPA** — Java Persistence API including spring-data-JPA, spring-orm, and Hibernate

**MySQL** — MySQL JDBC driver

**Actuator** — Production-ready features to help you monitor and manage your application

Generate, download, and import to development IDE.

## Define Database Configurations

Next Create the database name called *user\_database* in MySQL database server and define connection properties in *spring-boot-rest-api-tutorial/src/main/resources/application.properties*

```
1
2 ## Database Properties
3
4 spring.datasource.url = jdbc:mysql://localhost:3306/users_database?useSSL=false
5
6 spring.datasource.username = root
7
8 spring.datasource.password = root
9
10
11 ## Hibernate Properties
12
13 # The SQL dialect makes Hibernate generate better SQL for the chosen database
14
15 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
16
17
18 # Hibernate ddl auto (create, create-drop, validate, update)
19
20 spring.jpa.hibernate.ddl-auto = update
```

## Create Entity Class

```
1
2 @Entity
3
4 @Table(name = "users")
5
```

@EntityListeners(AuditingEntityListener.class)	4
public class User {	5
	6
@Id	7
@GeneratedValue(strategy = GenerationType.AUTO)	8
private long id;	9
	10
@Column(name = "first_name", nullable = false)	11
private String firstName;	12
	13
@Column(name = "last_name", nullable = false)	14
private String lastName;	15
	16
@Column(name = "email_address", nullable = false)	17
private String email;	18
	19
@Column(name = "created_at", nullable = false)	20
@CreatedDate	21
private Date createdAt;	22
	23
@Column(name = "created_by", nullable = false)	24
@CreatedBy	25
private String createdBy;	26
	27
@Column(name = "updated_at", nullable = false)	28
@LastModifiedDate	29
private Date updatedAt;	30

	31
@Column(name = "updated_by", nullable = false)	32
@LastModifiedBy	33
private String updatedBy;	34
	35
/**	36
* Gets id.	37
*	38
* @return the id	39
*/	40
public long getId() {	41
return id;	42
}	43
	44
/**	45
* Sets id.	46
*	47
* @param id the id	48
*/	49
public void setId(long id) {	50
this.id = id;	51
}	52
	53
/**	54
* Gets first name.	55
*	56
* @return the first name	57
*/	58

<code>public String getFirstName() {</code>	59
<code>    return firstName;</code>	60
<code>}</code>	61
	62
<code>/**</code>	63
<code> * Sets first name.</code>	64
<code> *</code>	65
<code> * @param firstName the first name</code>	66
<code> */</code>	67
<code>public void setFirstName(String firstName) {</code>	68
<code>    this.firstName = firstName;</code>	69
<code>}</code>	70
	71
<code>/**</code>	72
<code> * Gets last name.</code>	73
<code> *</code>	74
<code> * @return the last name</code>	75
<code> */</code>	76
<code>public String getLastName() {</code>	77
<code>    return lastName;</code>	78
<code>}</code>	79
	80
<code>/**</code>	81
<code> * Sets last name.</code>	82
<code> *</code>	83
<code> * @param lastName the last name</code>	84
<code> */</code>	85
<code>public void setLastName(String lastName) {</code>	

	86
<code>this.lastName = lastName;</code>	87
<code>}</code>	88
	89
<code>/**</code>	90
<code> * Gets email.</code>	91
<code> *</code>	92
<code> * @return the email</code>	93
<code> */</code>	94
<code>public String getEmail() {</code>	95
<code>    return email;</code>	96
<code>}</code>	97
	98
<code>/**</code>	99
<code> * Sets email.</code>	100
<code> *</code>	101
<code> * @param email the email</code>	102
<code> */</code>	103
<code>public void setEmail(String email) {</code>	104
<code>    this.email = email;</code>	105
<code>}</code>	106
	107
<code>/**</code>	108
<code> * Gets created at.</code>	109
<code> *</code>	110
<code> * @return the created at</code>	111
<code> */</code>	112
<code>public Date getCreatedAt() {</code>	113

<code>return createdAt;</code>	
	114
<code>}</code>	
	115
	116
<code>/**</code>	
	117
<code> * Sets created at.</code>	
	118
<code> *</code>	
	119
<code> * @param createdAt the created at</code>	
	120
<code> */</code>	
	121
<code>public void setCreatedAt(Date createdAt) {</code>	
	122
<code>    this.createdAt = createdAt;</code>	
	123
<code>}</code>	
	124
	125
<code>/**</code>	
	126
<code> * Gets created by.</code>	
	127
<code> *</code>	
	128
<code> * @return the created by</code>	
	129
<code> */</code>	
	130
<code>public String getCreatedBy() {</code>	
	131
<code>    return createdBy;</code>	
	132
<code>}</code>	
	133
	134
<code>/**</code>	
	135
<code> * Sets created by.</code>	
	136
<code> *</code>	
	137
<code> * @param createdBy the created by</code>	
	138
<code> */</code>	
	139
<code>public void setCreatedBy(String createdBy) {</code>	
	140
<code>    this.createdBy = createdBy;</code>	

	141
}	142
	143
/**	144
* Gets updated at.	145
*	146
* @return the updated at	147
*/	148
public Date getUpdatedAt() {	149
return updatedAt;	150
}	151
	152
/**	153
* Sets updated at.	154
*	155
* @param updatedAt the updated at	156
*/	157
public void setUpdatedAt(Date updatedAt) {	158
this.updatedAt = updatedAt;	159
}	160
	161
/**	162
* Gets updated by.	163
*	164
* @return the updated by	165
*/	166
public String getUpdatedBy() {	167
return updatedBy;	168



```

    }
169
170
171 /**
172  * Sets updated by.
173  *
174  * @param updatedBy the updated by
175  */
176 public void setUpdatedBy(String updatedBy) {
177     this.updatedBy = updatedBy;
178 }
179
180 }

```

## Create JPA Data Repository Layer

```

1
@Repository
2
public interface UserRepository extends JpaRepository<User, Long> {}

```

## Create Rest Controllers and Map API Requests

```

1
@RestController
2
@RequestMapping("/api/v1")
3
public class UserController {
4
5
6     @Autowired
7     private UserRepository userRepository;
8
9
10    /**
11     * Get all users list.
12     *
13     * @return the list

```

```

    */
    13
    @GetMapping("/users")
    14
    public List<User> getAllUsers() {
    15
        return userRepository.findAll();
    16
    }
    17
    /**
    18
    * Gets users by id.
    19
    *
    20
    * @param userId the user id
    21
    * @return the users by id
    22
    * @throws ResourceNotFoundException the resource not found exception
    23
    */
    24
    25
    @GetMapping("/users/{id}")
    26
    public ResponseEntity<User> getUsersById(@PathVariable(value = "id") Long u
    27
    serId)
    28
    throws ResourceNotFoundException {
    29
        User user =
    30
        userRepository
    31
        .findById(userId)
    32
        .orElseThrow(() -> new ResourceNotFoundException("User not found
    33
        on :: " + userId));
    34
        return ResponseEntity.ok().body(user);
    35
    }
    36
    /**
    37
    * Create user user.
    38
    *
    39
    * @param user the user
    40

```

```

    * @return the user
40
    */
41
    @PostMapping("/users")
42
    public User createUser(@Valid @RequestBody User user) {
43
        return userRepository.save(user);
44
    }
45
46
    /**
47
    * Update user response entity.
48
    *
49
    * @param userId the user id
50
    * @param userDetails the user details
51
    * @return the response entity
52
    * @throws ResourceNotFoundException the resource not found exception
53
    */
54
    @PutMapping("/users/{id}")
55
    public ResponseEntity<User> updateUser(
56
        @PathVariable(value = "id") Long userId, @Valid @RequestBody User userD
57
        etails)
58
        throws ResourceNotFoundException {
59
60
        User user =
61
            userRepository
62
                .findById(userId)
63
                .orElseThrow(() -> new ResourceNotFoundException("User not found
64
on :: " + userId));
65
        user.setEmail(userDetails.getEmail());
66
        user.setLastName(userDetails.getLastName());

```

user.setFirstName(userDetails.getFirstName());	67
user.setUpdatedAt(new Date());	68
final User updatedUser = userRepository.save(user);	69
return ResponseEntity.ok(updatedUser);	70
}	71
	72
/**	73
* Delete user map.	74
*	75
* @param userId the user id	76
* @return the map	77
* @throws Exception the exception	78
*/	79
@DeleteMapping("/user/{id}")	80
public Map<String, Boolean> deleteUser(@PathVariable(value = "id") Long use rId) throws Exception {	81
User user =	82
userRepository	83
.findById(userId)	84
.orElseThrow(() -> new ResourceNotFoundException("User not found on :: " + userId));	85
	86
userRepository.delete(user);	87
Map<String, Boolean> response = new HashMap<>();	88
response.put("deleted", Boolean.TRUE);	89
return response;	90
}	91
}	

# Create Unit Testing for API Requests and Run the Unit Testing

```
1
@RunWith(SpringRunner.class)
2
@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.
WebEnvironment.RANDOM_PORT)
3
public class ApplicationTests {
4
5
@Autowired
6
private TestRestTemplate restTemplate;
7
8
@LocalServerPort
9
private int port;
10
11
private String getRootUrl() {
12
return "http://localhost:" + port;
13
}
14
15
@Test
16
public void contextLoads() {
17
}
18
19
@Test
20
public void testGetAllUsers() {
21
HttpHeaders headers = new HttpHeaders();
22
HttpEntity<String> entity = new HttpEntity<String>(null, headers);
23
24
ResponseEntity<String> response = restTemplate.exchange(getRootUrl() + "/user
s",
25
```

HttpMethod.GET, entity, String.class);	26
	27
Assert.assertNotNull(response.getBody());	28
}	29
	30
@Test	31
public void testGetUserById() {	32
User user = restTemplate.getForObject(getRootUrl() + "/users/1", User.class);	33
System.out.println(user.getFirstName());	34
Assert.assertNotNull(user);	35
}	36
	37
@Test	38
public void testCreateUser() {	39
User user = new User();	40
user.setEmail("admin@gmail.com");	41
user.setFirstName("admin");	42
user.setLastName("admin");	43
user.setCreatedBy("admin");	44
user.setUpdatedBy("admin");	45
	46
ResponseEntity<User> postResponse = restTemplate.postForEntity(getRootUrl() + "/users", user, User.class);	47
Assert.assertNotNull(postResponse);	48
Assert.assertNotNull(postResponse.getBody());	49
}	50
	51
@Test	52

```

public void testUpdatePost() {
    int id = 1;
    User user = restTemplate.getForObject(getRootUrl() + "/users/" + id, User.class);
    user.setFirstName("admin1");
    user.setLastName("admin2");
    restTemplate.put(getRootUrl() + "/users/" + id, user);
    User updatedUser = restTemplate.getForObject(getRootUrl() + "/users/" + id, User.class);
    Assert.assertNotNull(updatedUser);
}

@Test
public void testDeletePost() {
    int id = 2;
    User user = restTemplate.getForObject(getRootUrl() + "/users/" + id, User.class);
    Assert.assertNotNull(user);
    restTemplate.delete(getRootUrl() + "/users/" + id);
    try {
        user = restTemplate.getForObject(getRootUrl() + "/users/" + id, User.class);
    } catch (final HttpClientErrorException e) {
        Assert.assertEquals(e.getStatusCode(), HttpStatus.NOT_FOUND);
    }
}

```

```
}
```

## Build and Run the Project

```
mvn package
```

```
java -jar target/spring-boot-rest-api-tutorial-1.0.0.jar
```

Alternatively, you can run the app without packaging it using:

```
mvn spring-boot:run
```

The app will start running at <http://localhost:8080>.