**Master SpringBoot**

**02-Rest JPA and H2 database:**

**Step-01: Usecase Introduction – User Management Service**

* **Create User – POST - /users**
* **Get All Users – GET - /users**
* **Get User by Id – GET - /users/{id}**
* **Update User by Id – PUT - /users/{id}**
* **Delete User by Id – DELETE - /users/{id}**
* **Get User by username – GET - /users/byusername/{username}**

**Step-02: Verify pom.xml for all Dependencies**

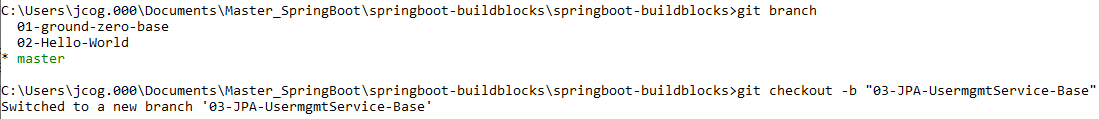
* **Mandatory**
  + **Spring Boot Starter Web**
  + **Spring Boot Data JPA**
  + **H2 Database**
* **Optional**
  + **Spring Boot DevTools**
  + **Create GIT Branch named “03-JPA-UsermgmtService-Base”**

**Verify pom.xml dependencies:**



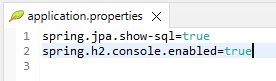
**To create a new Branch make sure the master is the latest with the branch (Locally)**

**git branch git checkout –b “03-JPA-UsermgmtService-Base”**



**Step-03: application.properties**

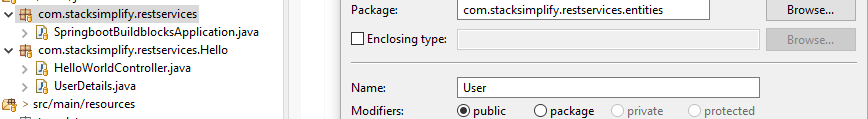
* **spring.jpa.show-sql=true**
* **spring.h2.console.enabled=true**

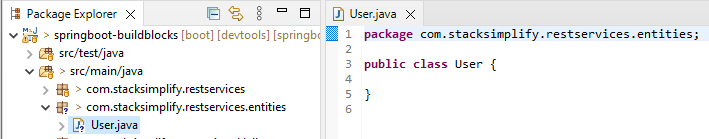


**Step-04: Entity Creation (User Entity)**

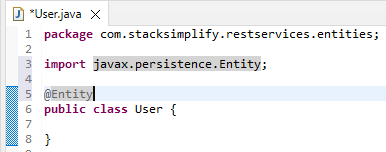
* **@Entity**
* **@Table**
* **Fields or Variables (id, username, firstname, lastname, email, role, ssn)**
* **@Id**
* **@GeneratedValue**
* **@Column (name, length, nullable, unique)**
* **No Arg Constructor**
* **Fields Constructor**
* **ToString (Optional for bean logging)**

**Create the POJO or bean User:**

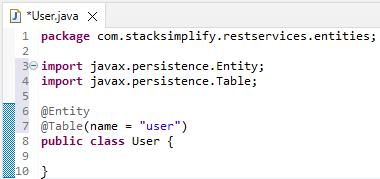




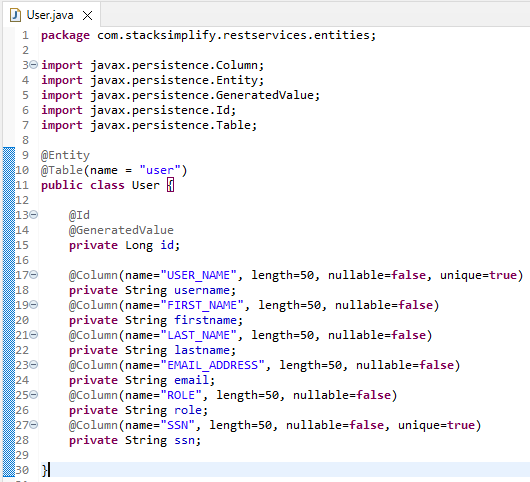
**Create the class Entity**



**Now add the database table name for this User entity (class)**

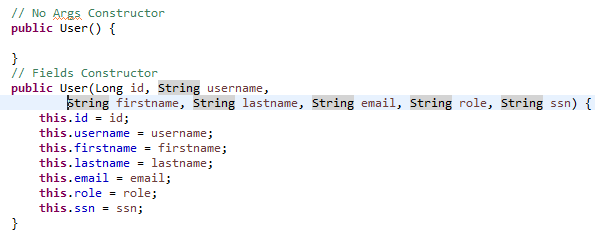


**Create the variables:**

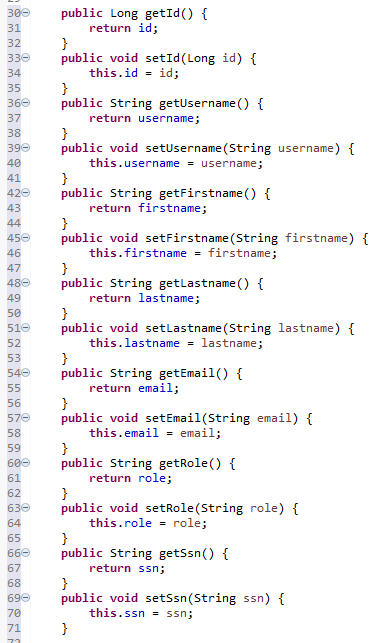


**Define No Args Constructor**

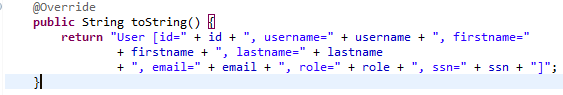
**Define Field Args Constructor**



**Define GETTERS and SETTERS**



**Define toString() method**



**Step-05: H2 Database**

* **In Memory Database (data will be lost when we restart JVM or when JVM reloads)**
* **Prepopulate DB during runtime**
* **Create a data.sql in src/main/resources**
* **Note: Columns will be created in Alphabetical order in DB except primary key Id**
* **Note: So insert statement values should be in alphabetical order as displayed in H2 DB.**
* **H2 Console:** [**http://localhost:8080/h2-console**](http://localhost:8080/h2-console)
* **JDBC URL: jdbc:h2:mem:testdb**

**H2 DB Records**

**insert into user values(101, ‘jcoggins@example.com’, ‘Joe’, ‘Coggins’, ‘admin’, ‘ssn101’, ‘jcoggins’ )**

**insert into user values(102, ‘bshwartz@example.com’, ‘Bill’, ‘Shwartz’, ‘admin’, ‘ssn102’, ’bshwartz’ )**

**insert into user values(103, ‘junderwood@example.com’, ‘John’, ‘Underwood’, ‘admin’, ‘ssn103’, ‘junderwood’ )**

**Start the Spring Boot App, Check create Table process:**

Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.

2022-01-01 15:13:46.945 ERROR 13616 --- [ restartedMain] o.s.b.d.LoggingFailureAnalysisReporter :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

APPLICATION FAILED TO START

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

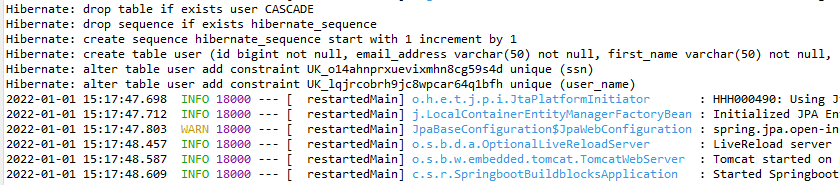
Description:

Web server failed to start. Port 8080 was already in use.

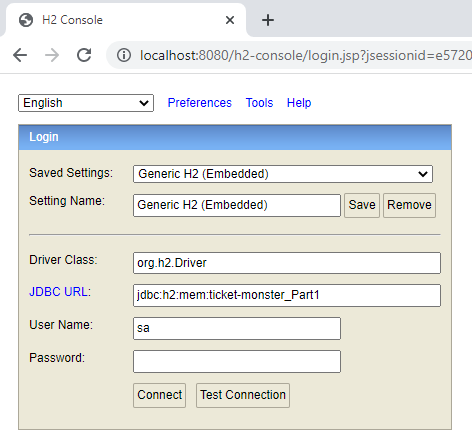
**Restart the Spring Boot app by first killing the tomcat server:**

**Click on the largest red square!**

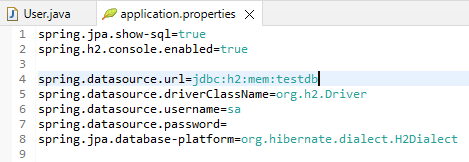
**Stops the tomcat server**



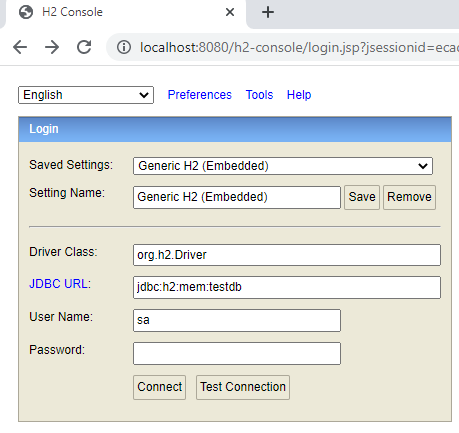
**View the H2 console:** [**http://localhost:8080/h2-console**](http://localhost:8080/h2-console)



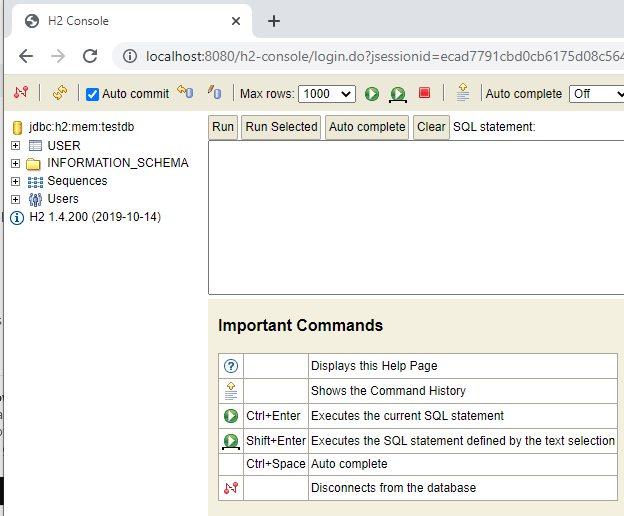
**Replace the JDBC URL: with jdbc:h2:mem:testdb to fix the interference**



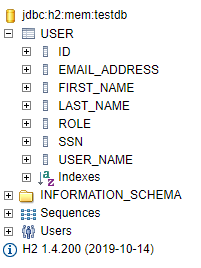
**Resave the changes and relaunch the Web Browser:**



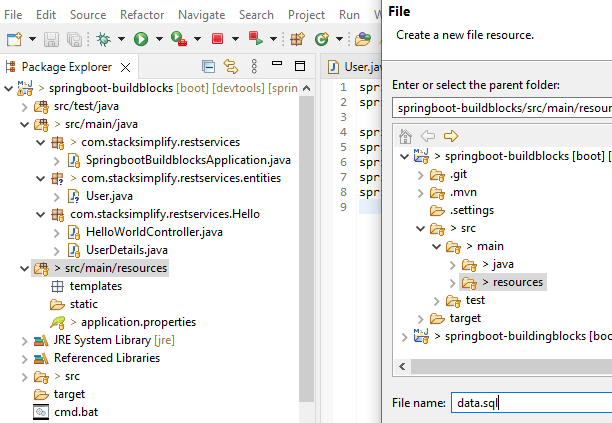
**Click Connect to log in to the Database:**



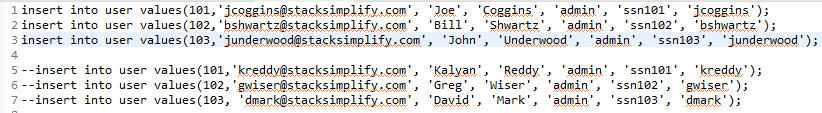
**User table:**



**Let us populate the initial insert statements:**

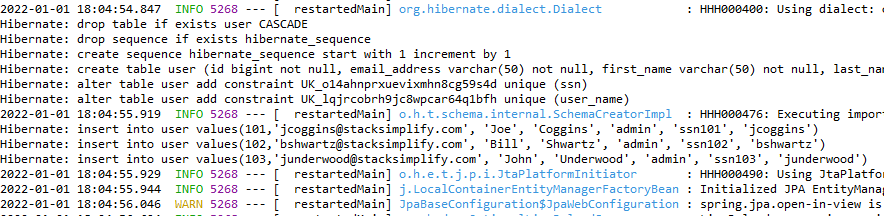


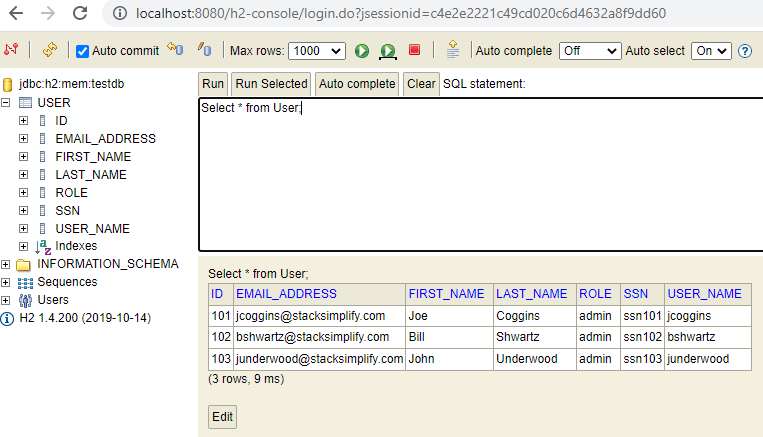
**Had to refactor the file name data.sql as import.sql**



**Restarted the Spring Boot Application:**

**In the stacktrace you can see the insert statements are loading successfully:**

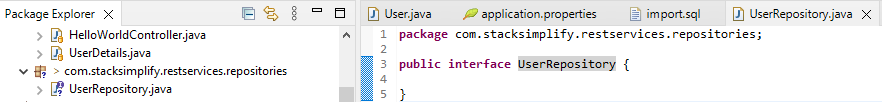




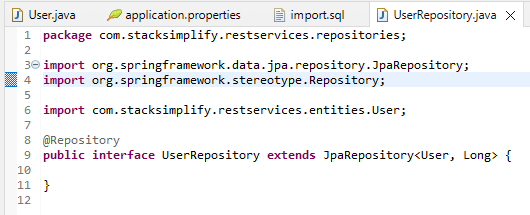
**Step-06: Repository**

* **Create an Interface**
* **Use extends for JpaRepository**
* **@Repository**

**Create the interface:**



**UserRepository will use the User entity and primary key ID:**



**Step-07: Implement getAllUsers Method**

* **Service:**

**- Create a UserService class**

**- Annotate it with @Service**

**- @Autowired (Autowire UserRepository)**

**- Create getAllUsers Method**

**- Controller**

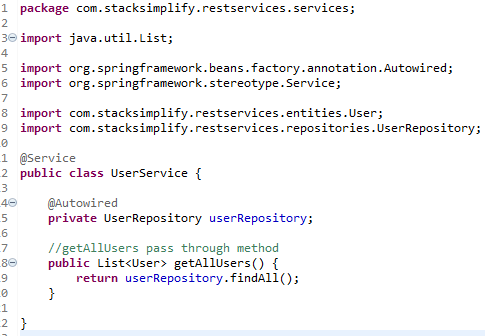
**- Create a UserController class**

**- Annotate it with @RestController**

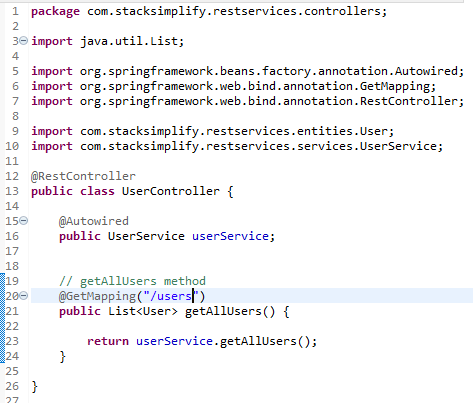
**- @Autowired (Autowire UserService)**

**- Create @GetMapping for getAllUsers Method**

**Create the UserService class and implement the Service:**



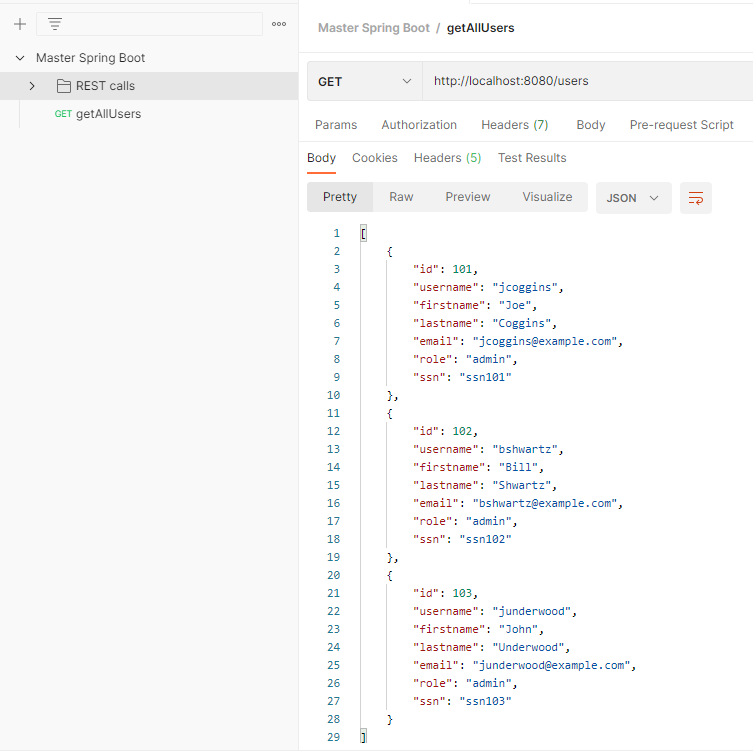
**Create the UserController class and user the UserService:**



**Step-08: Test using REST Client – Postman**

* **Download & Install Postman –** [**https://www.getpostman.com/**](https://www.getpostman.com/)
* **Create Collection “SpringBoot-BuildingBlocks”**
* **Create a request for getAllUsers**
  + **Method: GET**
  + **URIL** [**http://localhost:8080/users**](http://localhost:8080/users)
* **Test and Verify getAllUsers RESTful service.**

**Test with PostMan client:**

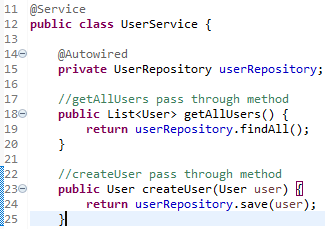


**It works and displays JSON for all 3 users**

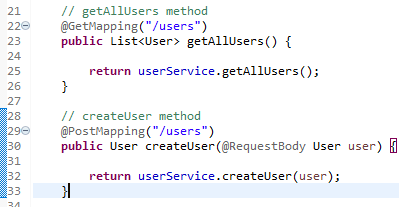
**Step-09: Implement createUser Method in Service & Controller Layers**

* **Service Layer:**
  + **Implement createUser method**
* **Controller Layer:**
  + **Implement createUser method**
    - **@RequestBody Annotation**
  + **Add @PostMapping Annotation to CreateUser Method**
* **POSTMAN: Verify the createUser RESTful Service in postman REST Client**
  + **Create a request for createUser**
    - **Body (select raw)**
    - **Content Type: JSON (application/json)**
    - **Method: POST**
  + **Test using postman**

**Implement createUser in the Service Layer:**

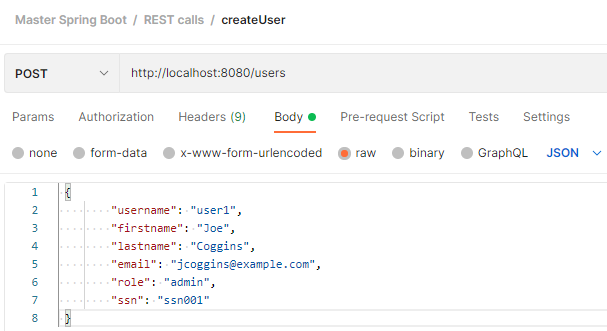


**Implement createUser in the Controller Layer:**

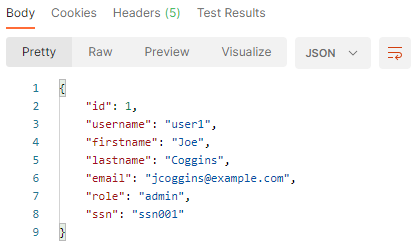


**Now, use POSTMAN to test:**

**Create the user: id is autogenerated**



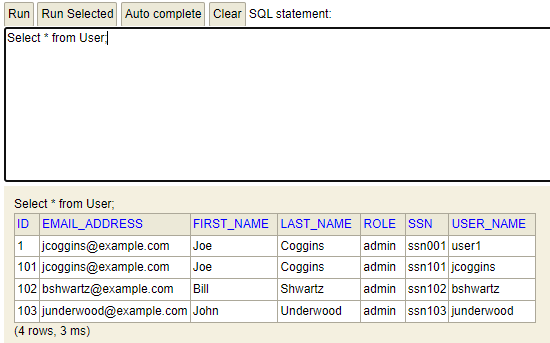
**Save and then Send**



**Check the getAllUsers() method: (should be 4 users)**



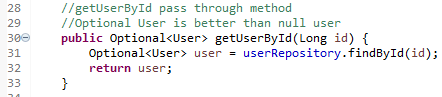
**Now, check the database H2:**



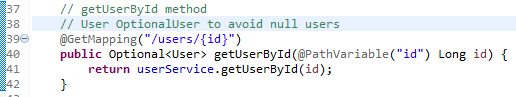
**Step-10: Implement getUserById Method in Service and Controllers Layers**

* **Service Layer:**
  + **Create getUserById Method**
  + **Talk about return type Optional Object**
* **Controller Layer:**
  + **Create getUserById Method and apply Optional return type**
  + **@PathVariable Annotation**
  + **Annotate @GetMapping for getUserById Method**
  + **URI: /users/{id}**
* **POSTMAN: Create & Verify the getUserById RESTful Service in postman REST client**
  + **Method: GET**
  + **URL:** [**http://localhost:8080/users/{id}**](http://localhost:8080/users/%7bid%7d)
  + **URL:** [**http://localhost:8080/users/103**](http://localhost:8080/users/103)

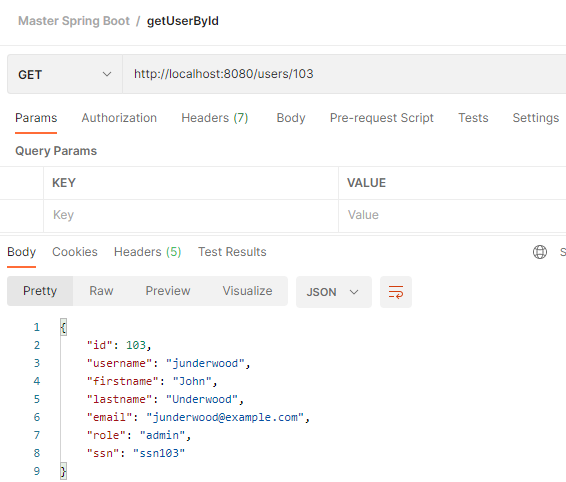
**Implement getUserById in the Service Layer:**



**Implement getUserById in the Controller Layer:**



**Now, use POSTMAN to test:**



**It works!**

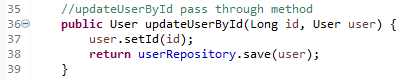
**OptionalUser is a temporary solution until we can implement Exception Handling.**

**The return can be null.**

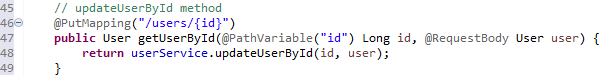
**Step-11: Implement updateUserById Method in Service and Controller Layer**

* **Service Layer**
  + **Create updateUserById Method**
  + **Inputs (User, Id)**
* **Controller Layer**
  + **Create updateUserById Method**
  + **@PathVariable**
  + **@RequestBody**
  + **Annotate @PutMapping for updateUserById Method**
* **POSTMAN: Verify updateUserById RESTful Service in postman REST client**
  + **Create a request for updateUserById**
    - **Body (select raw)**
    - **Content Type: JSON (application/json)**
    - **Method: PUT**
    - **In URI: /users/{id}**
  + **Test using postman**

**Implement updateUserById in the Service Layer:**

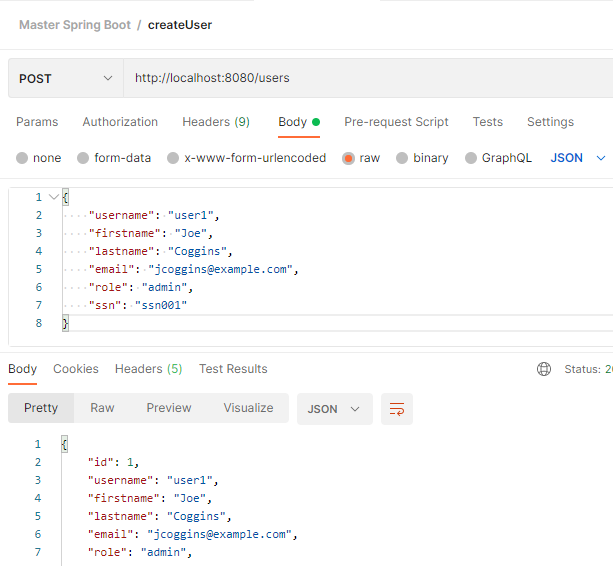


**Implement updateUserById in the Controller Layer:**



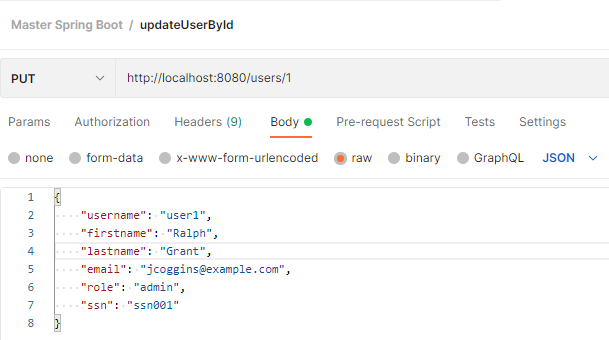
**Now, use POSTMAN to test:**

**Create a user to update:**

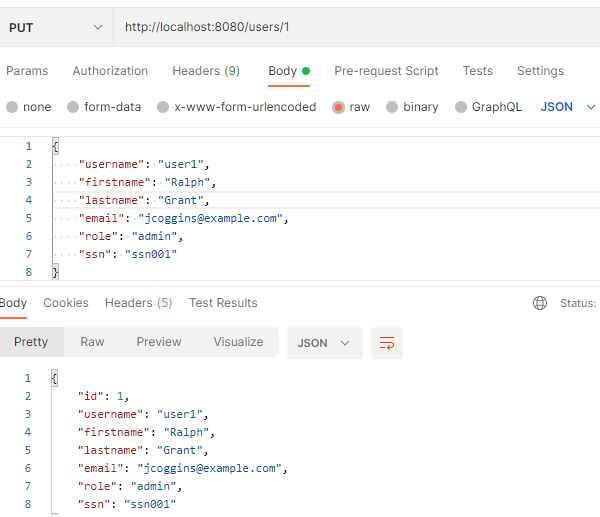


**Copy the user by removing the id, change the username and ssn (both unique)**

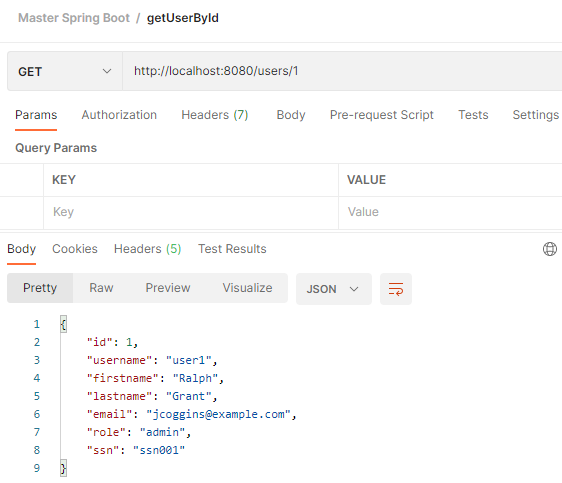
**Altered the firstname to Ralph and lastname to Grant for user1**



**Save and Send**



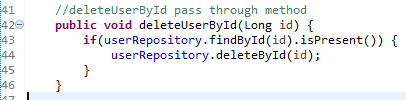
**Verify with getUserById firstname is not Joe, and lastname is not Coggins**



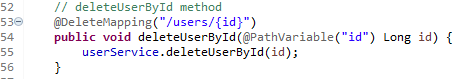
**Step-12: Implement deleteUserById Method in Service and Controller Layers**

* **Service Layer:**
  + **Create deleteUserById Method**
  + **Verify user exists and then delete it**
* **Controller Layer:**
  + **Create deleteUserById Method**
  + **@PathVariable Annotation**
  + **Annotate @DeleteMapping for deleteUserById Method**
* **POSTMAN: Verify deleteUseById Service in postman REST client**
  + **Method: DELETE**
  + **URL:** [**http://localhost:8080/users/{id}**](http://localhost:8080/users/%7bid%7d)
  + **URL:** [**http://localhost:8080/users/103**](http://localhost:8080/users/103)

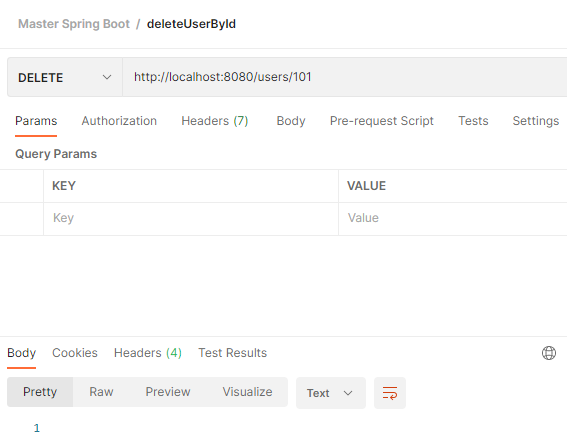
**Implement deleteUserById in the Service Layer**



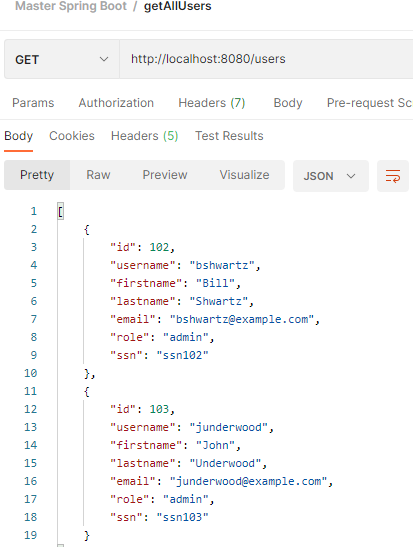
**Implement deleteUserById in the Controller Layer**



**Test with POSTMAN**



**Verify user with id 101 no longer exists by running getAllUsers**

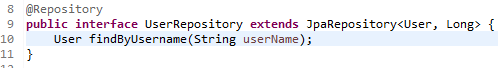


**It works, user id 101 is missing**

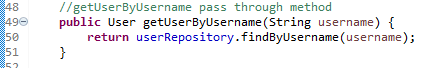
**Step-13: Implement getUserByUsername Method in Repository, Services & Controller Layers**

* **Repository Layer:**
  + **Define “findByUsername” in UserRepository Interface**
* **Service Layer:**
  + **Create getUserByUsername Method**
  + **Input as username**
* **Controller Layer:**
  + **Create getUserByUsername Method**
  + **@PathVariable Annotation**
  + **Annotate @GetMapping for getUserByUsername Method**
  + **URI: /users/byusername/{username}**
* **POSTMAN: Create & Verify the getUserByUsername RESTful Service in postman REST client**
  + **Method: GET**
  + **URL:** [**http://localhost:8080/users/byusername/{username}**](http://localhost:8080/users/byusername/%7busername%7d)
  + **URL:** [**http://localhost:8080/users/byusername/{jcoggins}**](http://localhost:8080/users/byusername/%7bjcoggins%7d)

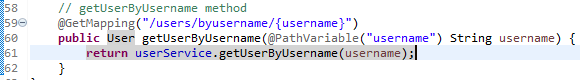
**Define the signature in the Interface – JpaRepository**



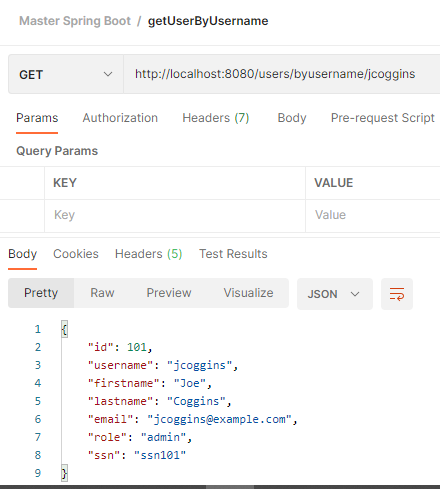
**Implement the getUserByUsername in the Service Layer**



**Implement the getUserByUsername in the Controller Layer**



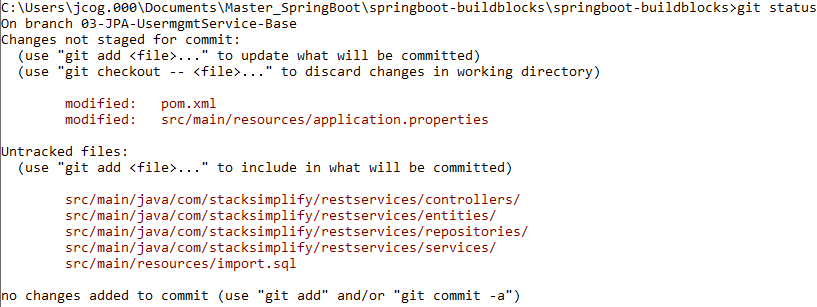
**Test it using POSTMAN**



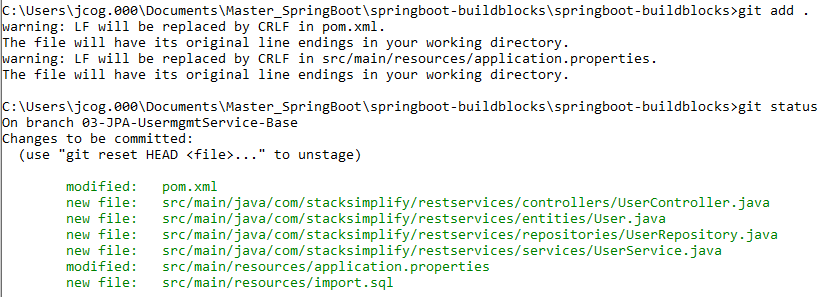
**Step-14: GIT Commit & Merge to Master & Push**

* **Branch: 03-UserMgmtService-Base**
  + **Commit & Push**
* **Branch: Master**
  + **Merge from branch 03-UserMgmtService-Base**
  + **Commit & Push**
* **Verify in github**

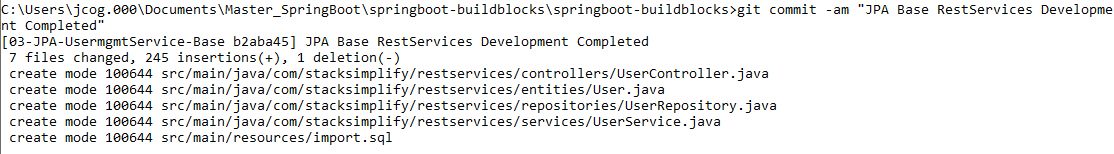
**Check the status of the changes: git status changes need to be staged**



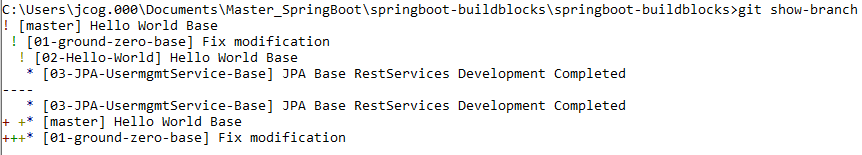
**Now stage the changes: git add . git status**



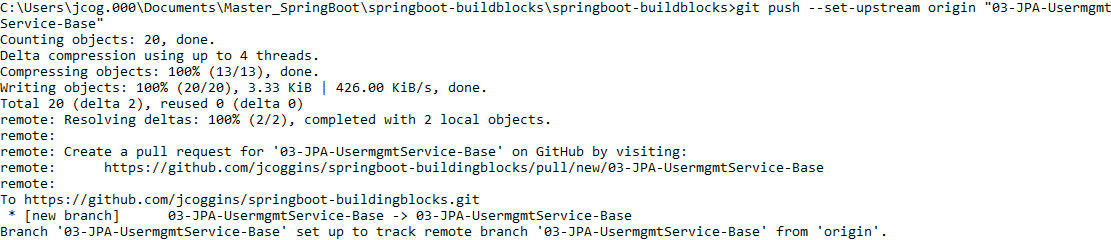
**Now commit the staged changes: git commit –am “JPA Base RestServices Development Completed”**



**Show the state: git show-branch**



**Remote Origin needs to created git push –set-upstream origin “03-JPA-UsermgmtService-Base”**



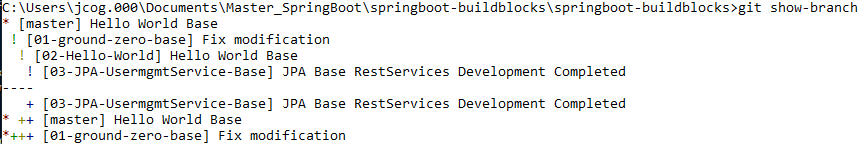
**Push to the remote branch**



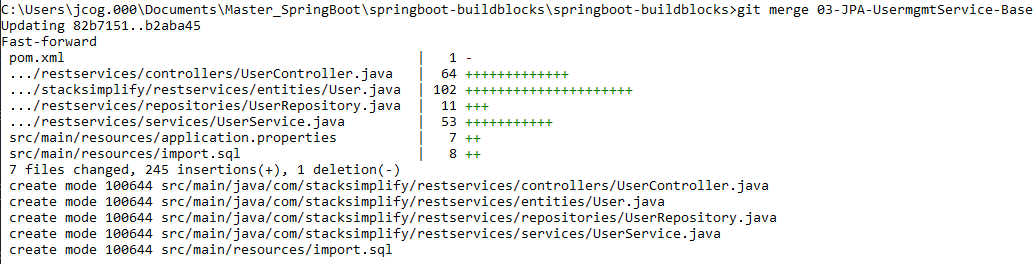
**Now checkout the Local master, the origins are done git checkout master**



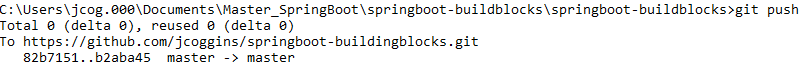
**Show the master that is checked out git show-branch**



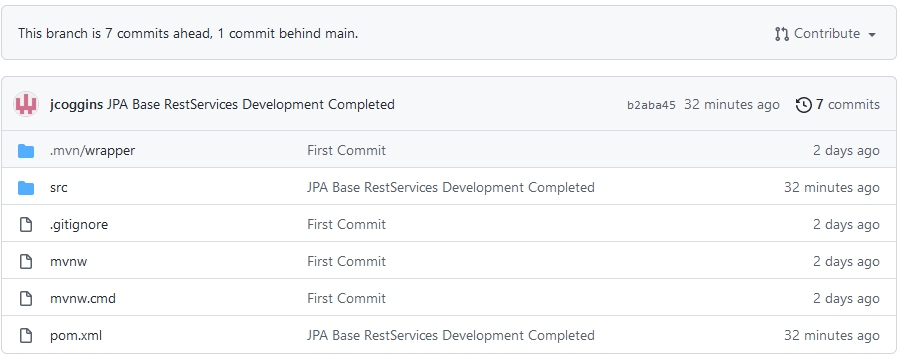
**Merge the Local origin changes onto the master git merge 03-JPA-UsermgmtService-Base**

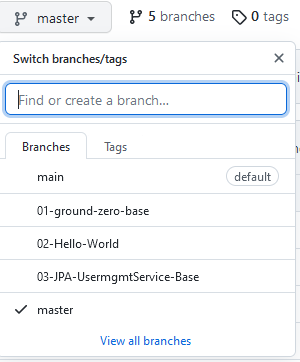


**Now from the local master move the changes onto the Remote master git push**



**Now check the GitHub repository:**





**GIT GUI on eclipse:**

