Ryde Back-end Developer Test

userdemo API Documentation

Zhao Qi

Contents

1. Introduction

This documentation aims to describe the implementation details of the Restful API for Ryde Back-end Developer Test.

Application Name: userdemo

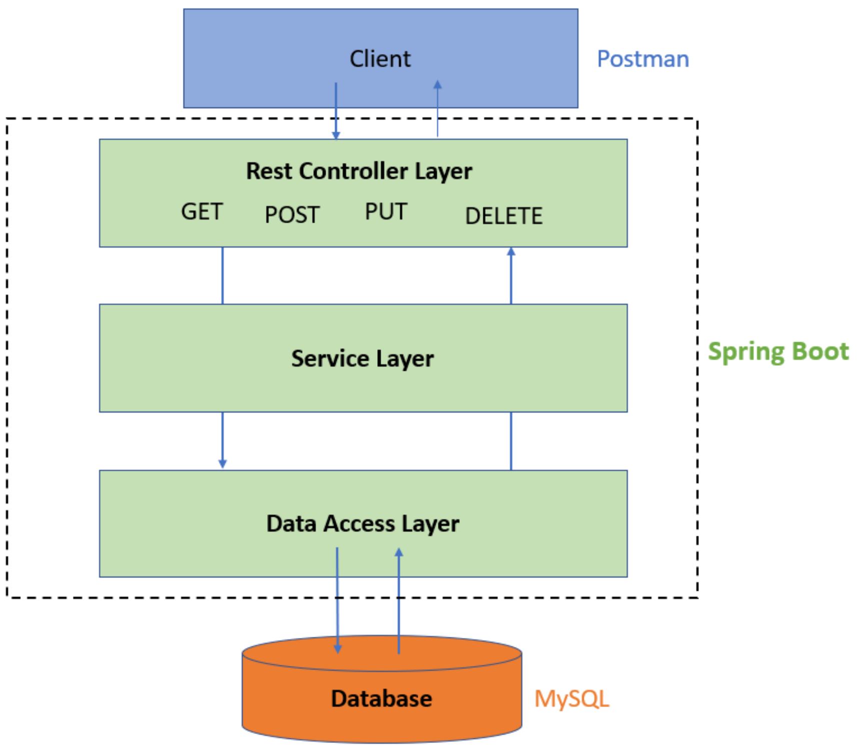
Language: Java

Framework: Spring Boot

Database: MySQL

2. Architecture

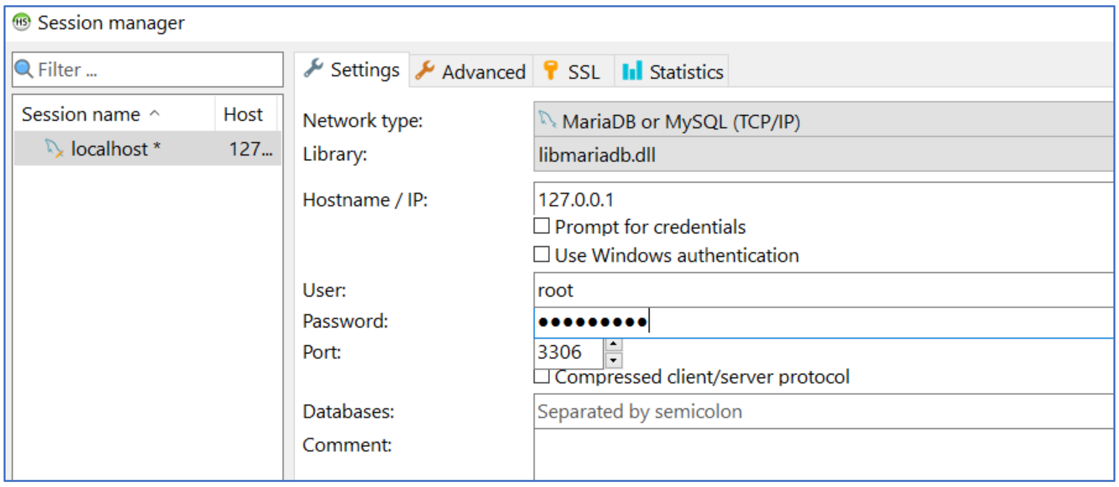
Below picture shows the architecture of userdemo application. We use Spring Boot as the framework, MySQL as the database server, Postman as the client to test http request.



3. Project Setup

3.1 Create MySQL Database

First, connect to the MySQL server by specifying “User” and “Password”.



Network type: MariaDB or MySQL (TCP/IP)
Library: libmariadb.dll
Hostname / IP: 127.0.0.1
☐ Prompt for credentials
☐ Use Windows authentication
User: root
Password: ●●●●●●●●●
Port: 3306
☐ Compressed client/server protocol
Databases: Separated by semicolon
Comment:

Then create a new database called “userdemo”: `CREATE SCHEMA `userdemo``

Refresh the page, we find “userdemo” is created.

3.2 Configure Maven Dependencies

Specify the following configuration in the project’s pom.xml file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.5.3</version>
                <relativePath /> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.example</groupId>
        <artifactId>userdemo</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>userdemo</name>
        <description>Demo project for Spring Boot</description>
        <properties>
                <java.version>16</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-validation</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                        <scope>runtime</scope>
                        <optional>true</optional>
                </dependency>
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <scope>runtime</scope>
                </dependency>


                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                        <exclusions>
                                <exclusion>
                                        <groupId>junit</groupId>
                                        <artifactId>junit</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
                <dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter</artifactId>
                        <version>5.6.0</version>
                        <scope>test</scope>
                </dependency>


                <dependency>
                <groupId>org.powermock</groupId>
                <artifactId>powermock-core</artifactId>
                <version>2.0.0-RC.4</version>
                <scope>test</scope>
        </dependency>
                <dependency>
                        <groupId>org.powermock</groupId>
                <artifactId>powermock-module-junit4</artifactId>
                <version>2.0.0-beta.5</version>
                <scope>test</scope>
                </dependency>
                <dependency>
                <groupId>org.powermock</groupId>
                <artifactId>powermock-api-mockito2</artifactId>
                <version>2.0.0-beta.5</version>
                <scope>test</scope>
                </dependency>
                <dependency>
                <groupId>cglib</groupId>
                <artifactId>cglib</artifactId>
                <version>3.2.9</version>
                </dependency>
                <dependency>
                <groupId>org.mockito</groupId>
                <artifactId>mockito-inline</artifactId>
                <version>2.15.0</version>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-oauth2-client</artifactId>
                </dependency>


                <dependency>
                        <groupId>org.n52.jackson</groupId>
                        <artifactId>jackson-datatype-jts</artifactId>
```

```
                        <version>1.2.6</version>
                </dependency>

        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

The artifact spring-boot-starter-web is for Spring Web MVC, RESTful webservices and embedded Tomcat server.

The artifact spring-boot-starter-data-jpa is for Spring Data JPA and Hibernate.

The artifact mysql-connector-java is for JDBC driver for MySQL.

The artifact spring-boot-starter-oauth2-client is for login authentication.

The artifact spring-boot-devtools is for automatic restart so you don't have to manually restart the application during development.

The artifact powermock-core, powermock-module-junit4, powermock-api-mockito2, mockito-inline are for Unit Test purpose.

The artifact jackson-datatype-jts for serializing and deserializing JSON.

3.3 Configure Data Source Properties

Next, we need to specify database connection information. Create the application.properties file under src/main/resources directory with the following content:

```
spring.datasource.url=jdbc:mysql://localhost:3306/userdemo?useSSL=false&ser
verTimezone=Asia/Singapore
spring.datasource.username=root
spring.datasource.password=
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL5InnoDBDialect
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.security.oauth2.client.registration.google.client-id=
spring.security.oauth2.client.registration.google.client-secret=

# Logger settings
debug=true
logging.level.org.springframework=ERROR
```

Remember to update url, username and password according to the MySQL database server.

Specify the Oauth2 client-id and client-secret from your own google settings.

Specify the logging message at the ERROR level.

3.4 Code Domain Model Class

Next, create the User class under src/main/java/sg.nus.iss.userdemo.model directory to map with the user table in the database. Create the UserFriends class under the same directory to map with the user_friends table in the database.

```
@Entity
@Table(name = "user")
public class User {

	@Id
	@GeneratedValue (strategy = GenerationType.IDENTITY)
	private int id;
	private String name;

	@Past
	private LocalDate dob;

	private Point address;

	private String description;
	@NotEmpty
	private String email;
	private LocalDate createAt;


	@ManyToMany
	@JoinTable(name = "user_friends",
	joinColumns = @JoinColumn(name = "userId"),
	inverseJoinColumns = @JoinColumn(name = "friendId") )
	private Set<User> userFriends;

	// the method to add UserFriends
	public void addUserFriends(User user2) {

		if (CollectionUtils.isEmpty(this.userFriends)) {
			this.userFriends = new HashSet<>();
		}

		userFriends.add(user2);
	}


	// constructor without arguments
	public User() { }

	// constructors for test purpose
	public User(int id,
			String name,
			@Past LocalDate dob,
			String description,
                 @NotEmpty String email) {
        super();
        this.id = id;
        this.name = name;
        this.dob = dob;
        this.description = description;
        this.email = email;
    }

    public User( String name,
                 @Past LocalDate dob,
                 String description,
                 @NotEmpty String email) {
        super();
        this.name = name;
        this.dob = dob;
        this.description = description;
        this.email = email;
    }


    public User(int id,
                 String name,
                 @Past LocalDate dob,
                 String description,
                 @NotEmpty String email,
                 LocalDate createAt) {
        super();
        this.id = id;
        this.name = name;
        this.dob = dob;
        this.description = description;
        this.email = email;
        this.createAt = createAt;
    }

    public User(int id,
                 String name,
                 @Past LocalDate dob,
                 Point address,
                 String description,
                 @NotEmpty String email,
                 LocalDate createAt) {
        super();
        this.id = id;
        this.name = name;
        this.dob = dob;
        this.address = address;
        this.description = description;
        this.email = email;
        this.createAt = createAt;
    }


    // constructors including all attributes
    public User(int id,
                 String name,
                 @Past LocalDate dob,
                 Point address,
```

```java
                String description,
                @NotEmpty String email,
                LocalDate createAt,
                Set<User> followers,
                Set<User> userFriends) {
            super();
            this.id = id;
            this.name = name;
            this.dob = dob;
            this.address = address;
            this.description = description;
            this.email = email;
            this.createAt = createAt;
            this.userFriends = userFriends;
    }



    // Getters and Setters
    public int getId() {
            return id;
    }


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


    public String getName() {
            return name;
    }


    public void setName(String name) {
            this.name = name;
    }

    public LocalDate getDob() {
            return dob;
    }

    public void setDob(LocalDate dob) {
            this.dob = dob;
    }

    public Point getAddress() {
            return address;
    }

    public void setAddress(Point address) {
            this.address = address;
    }

    public String getDescription() {
            return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public LocalDate getCreateAt() {
        return createAt;
    }

    public void setCreateAt(LocalDate createAt) {
        this.createAt = createAt;
    }


    public Set<User> getUserFriends() {
        return userFriends;
    }

    public void setUserFriends(Set<User> userFriends) {
        this.userFriends = userFriends;
    }
}
```

```
@Entity
@Table(name = "user_friends")
public class UserFriends {

    @Id
    @Column
    private int friendId;

    @ManyToOne
    @JoinColumn(name = "userId", nullable = false)
    private User user;

    public int getFriendId() {
        return friendId;
    }

    public void setFriendId(int friendId) {
        this.friendId = friendId;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
        }
}
```

Note the relationship annotation @ManyToMany and @ManyToOne, since one user can have many friends.

We specify attribute "Address" to be type of "Point" since we will get the geo-coordinates of user in this application.

3.5 Code initial Controller Layer, Service Layer and DAO Layer

3.5.1 Code Rest Controller Class

Here, we come to the part that actually exposes RESTful APIs for CRUD operations – a Spring controller following REST style. Create the UserDemoApiController class under src/main/java/ sg.nus.iss.userdemo.controller directory with some initial code as below:

```
@RestController
public class UserDemoApiController {

private final Logger logger =
LoggerFactory.getLogger(UserDemoApiController.class);

        private UserService uService;


        @Autowired
        public UserDemoApiController(UserService uService) {
                this.uService = uService;
        }

        // get all users
        // get a user by id
        // add a new user
        // update a user
        // delete a user by id

        // add friend by providing two emails
        // get all friends for a user
        // find friends nearby
}
```

3.5.2 Code Service Interface and Implementation Class

Next, code an interface and an implementation class that act as a middle layer between persistence layer (repository) and controller layer. Create the UserService interface and UserServiceImpl class with the following code:

```
public interface UserService {

}
```

```
@Service
public class UserServiceImpl implements UserService {

	private UserRepository uRepo;

	@Autowired
	public UserServiceImpl(UserRepository uRepo) {
		this.uRepo = uRepo;
	}

}
```

3.5.3 Code Repository Interface

To take advantages of Spring Data JPA, create the UserRepository interface as below:

```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

}
```

Then Spring Data JPA will generate implementation code for the most common CRUD operations – we don't have to write a single query.

4. Implement User CRUD

4.1 Read users

We will implement two reading methods: **get all users, get a user by id**.

Add the following codes in UserDemoApiController class:

```
// get all users
@GetMapping("/users")
public ResponseEntity<List<User>> getAllUsers() {
	try {
		List<User> userlist= uService.findAllUsers();
		if (userlist.isEmpty() || userlist.size() == 0) {
			return new ResponseEntity<List<User>>(HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<List<User>>(userlist, HttpStatus.OK);
		} catch(NoSuchElementException ex) {
			// log exception first, then return Conflict
		 logger.error(ex.getMessage());
			return new ResponseEntity<List<User>>(HttpStatus.NOT_FOUND);
		}
	}

	// get a user by id
	@GetMapping("/users/{userid}")
	public ResponseEntity<User> getUserbyId(@PathVariable("userid") int id) {
```

```
            try {
                    User user = uService.findUserById(id);
                    return new ResponseEntity<User>(user, HttpStatus.OK);
            } catch(NoSuchElementException ex) {
                    // log exception first, then return Conflict
             logger.error(ex.getMessage());
                    return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
            }
    }
```

Add the following codes in UserService interface:

```
    List<User> findAllUsers();

    User findUserById(int id);
```

Add the following codes in UserServiceImpl class:

```
    // find all users
    @Override
    public List<User> findAllUsers() {

            return uRepo.findAll();
    }

    // find a user by Id
    @Override
    public User findUserById(int id) {

            return uRepo.findById(id).get();
    }
```

In the controller, we use **GetMapping** for reading transactions.

We don't need to specify anything in UserRepository interface, since findAll() and findById(id).get() are default methods in DAO layer.

In addition, if users can be found, the server sends a response that includes JSON representation of the Product object with HTTP status OK (200). Else if no user is found, it returns HTTP status Not Found (404).

4.2 Create a New User

Add the following codes in UserDemoApiController class:

```
// add a new user
@PostMapping("/users/adduser")
public ResponseEntity<User> addNewUser(@RequestBody User user) {
        try {
                User newuser = uService.addNewUser(user);
                return new ResponseEntity<User>(newuser, HttpStatus.CREATED);

        } catch (Exception ex) {
                // log exception first, then return Conflict
                logger.error(ex.getMessage());
                return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
}
```

Add the following codes in UserService interface:

```
User addNewUser(User user);
```

Add the following codes in UserServiceImpl class:

```
        // add new user
        @Override
        public User addNewUser(User user) {
                // we assume email is unique for a user
                Optional<User> exists = uRepo.findById(user.getId());

                if (exists.isPresent()) {
                        throw new IllegalStateException("email taken");
                }
                User newUser = uRepo.save(user);
                return newUser;
        }
```

In the controller, we use **PostMapping** for creating transactions. A JSON object representing a user is passed via the request body, then we check whether the email is taken by an existing user, if email is not taken, the new user will be created.

We don't need to specify anything in UserRepository interface, since findById(), save() are the default method in DAO layer.

If user can be created, the server sends a response that includes JSON representation of the Product object with HTTP status CREATED (201). Else if user cannot be created, it returns HTTP Status INTERNAL_SERVER_ERROR(500).

4.3 Update a User

Add the following codes in UserDemoApiController class:

```
// update a user
@PutMapping("/users/update/{userid}")
public ResponseEntity<User> updateUser (
                        @PathVariable("userid") int userId,
                        @Valid @RequestBody User userDetails) {

        try {
                return new ResponseEntity<User>(
                        uService.updateUser(userId, userDetails), HttpStatus.OK);

        } catch (Exception ex) {
                logger.error(ex.getMessage());
                return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
}
```

Add the following codes in UserService interface:

```
User updateUser(int userId, @Valid User userDetails);
```

Add the following codes in UserServiceImpl class:

```
// update a user
@Override
public User updateUser(int userId, @Valid User userDetails) {
	User userFound = uRepo.findById(userId)
			.orElseThrow(() -> new IllegalStateException(
				user with id" + userId + "doesn't exist"));
	userFound.setName(userDetails.getName());
	userFound.setDescription(userDetails.getDescription());
	userFound.setEmail(userDetails.getEmail());

	User updatedUser = uRepo.save(userFound);
	return updatedUser;

}
```

In the controller, we use **PutMapping** for updating transactions. To update a user, we specify the user id in URL, and pass the new information of JSON object representing a user in the request body.

For the given user id, if no user is found, an IllegalStateException is thrown. If the user exists in database, we just call the setters method of the user. Finally save the user in database. We don't need to specify anything in UserRepository interface, since findById(), save() are the default methods in DAO layer.

If the user can be updated, the server sends a response that includes JSON representation of the Product object with HTTP status OK (200). Else if the user cannot be updated, it returns HTTP Status INTERNAL_SERVER_ERROR(500).

4.4 Delete a User

Add the following codes in UserDemoApiController class:

```
	// delete a user by id
	@DeleteMapping("/users/delete/{userid}")
	public ResponseEntity<HttpStatus> deleteUser(
			@PathVariable("userid") int userId) {
		try {
			uService.deleteUser(userId);
			return new ResponseEntity<HttpStatus>(
						HttpStatus.NO_CONTENT);

		}catch (Exception ex) {
			logger.error(ex.getMessage());
			return new ResponseEntity<HttpStatus>(
					HttpStatus.INTERNAL_SERVER_ERROR);
		}
	}
```

Add the following codes in UserService interface:

```
void deleteUser(int userId);
```

Add the following codes in UserServiceImpl class:

```
	// delete user
	@Override
	public void deleteUser(int userId) {

		boolean exists = uRepo.existsById(userId);

		if (!exists) {
			throw new IllegalStateException(
				"user with id" + userId + "doesn't exist");
		}
		uRepo.deleteById(userId);
	}
```

In the controller, we use **DeleteMapping** for deleting transactions. To delete a user, first we check whether the user exists in the database, if the user doesn't exist, an IllegalStateException will be thrown.

We don't need to specify anything in UserRepository interface, since existsById(), deleteById() are the default methods in DAO layer.

If the user can be deleted, the server sends a response HTTP status NO_COTENT (204). Else if the user cannot be deleted, it returns HTTP Status INTERNAL_SERVER_ERROR (500).

5. Implement User Friends

We will implement three methods in this section: **add friend, get friends by user id, get nearby friends**.

5.1 Add Friend

Two users can be friends to each other by providing their emails, whenever there is a UserFriendRequest, we will add user2 to the friend set of user1and add user1 to the friend set of user2. Follow the steps below:

Create a UserFriendsRequestEntity class under src/main/java/sg.nus.iss.userdemo.request directory with the following code:

```
public class UserFriendsRequestEntity {

	private List<String> friends;

	public List<String> getFriends() {
		return friends;
	}

	public void setFriends(List<String> friends) {
		this.friends = friends;
	}

}
```

Add the following codes in UserDemoApiController class:

```
// add friend by providing two emails
```

```
@PostMapping("/users/userFriendRequest")
public ResponseEntity<Map<String, Object>> userFriendRequest(
                        @RequestBody UserFriendsRequestEntity userFriendsRequestEntity) {

        return uService.addUserFriends(userFriendsRequestEntity);
}
```

Add the following codes in UserService interface:

```
ResponseEntity<Map<String, Object>> addUserFriends(
                        UserFriendsRequestEntity userFriendsRequestEntity);
```

Add the following codes in UserServiceImpl class:

```
// add friends with two different email
@Override
public ResponseEntity<Map<String, Object>> addUserFriends(
                UserFriendsRequestEntity userFriendsRequestEntity) {

        Map<String, Object> result = new HashMap<String, Object>();

        if (userFriendsRequestEntity == null) {
                result.put("Error : ", "Invalid request");
                return new ResponseEntity<Map<String, Object>>(
                                result, HttpStatus.BAD_REQUEST);
        }

        if (CollectionUtils.isEmpty(userFriendsRequestEntity.getFriends())) {
                result.put("Error : ", "Friend list cannot be empty");
                return new ResponseEntity<Map<String, Object>>(
                                result, HttpStatus.BAD_REQUEST);
        }
        if (userFriendsRequestEntity.getFriends().size() != 2) {
                result.put("Info : ", "Please provide 2 emails to make them friends");
                return new ResponseEntity<Map<String, Object>>(
                                result, HttpStatus.BAD_REQUEST);
        }

        String email1 = userFriendsRequestEntity.getFriends().get(0);
        String email2 = userFriendsRequestEntity.getFriends().get(1);

        if (email1.equals(email2)) {
                result.put("Info : ", "Cannot make friends, if users are same");
                return new ResponseEntity<Map<String, Object>>(
                                result, HttpStatus.BAD_REQUEST);
        }

        User user1 = null;
        User user2 = null;
        user1 = this.saveIfNotExist(email1);
        user2 = this.saveIfNotExist(email2);

        if (user1.getUserFriends().contains(user2)) {
                result.put("Info : ", "Can't add, they are already friends");
                return new ResponseEntity<Map<String, Object>>(
                                result, HttpStatus.OK);
        }

        // add user2 to the friend list of user1
        user1.addUserFriends(user2);
        this.uRepo.save(user1);

        // add user1 to the friend list of user2
        user2.addUserFriends(user1);
        this.uRepo.save(user2);

        result.put("Success", true);

        return new ResponseEntity<Map<String, Object>>(
                        result, HttpStatus.OK);
}
```

```
// given an email, if user not exists, save user;
// if user exists, return user.
private User saveIfNotExist(String email) {

        User existingUser = this.uRepo.findByEmail(email);
        if (existingUser == null) {
                existingUser = new User();
                existingUser.setEmail(email);
                return this.uRepo.save(existingUser);
        } else {
                return existingUser;
        }

}
```

Add the following codes in UserRepository interface:

```
User findByEmail(String email);
```

In the controller, we use **PostMapping** for UserFriendRequest transactions. To add friends for two users, their emails have to be provided. If they are already friends, then an error message will be shown. If the transaction is successful, the server sends a response with HTTP status OK (200). If the transaction fails due to the causes specified in service layer, it will sends respons with HTTP Status BAD_REQUEST (400).

5.2 Get Friends by User Id

Add the following codes in UserDemoApiController class:

```
// get all friends for a user
@GetMapping("/users/friends/{userid}")
public ResponseEntity<Set<String>> getAllFriendsByuserId(
                @PathVariable("userid") int userId) {

        try {
                //show the email instead of user object
                Set<String> allFriendsByUserId=
                                uService.getAllFriendsByUserId(userId);

                return new ResponseEntity<Set<String>>(
                                allFriendsByUserId, HttpStatus.OK);
        } catch(NoSuchElementException ex) {
                        // log exception first, then return Conflict
                 logger.error(ex.getMessage());
                return new ResponseEntity<Set<String>>(
                                HttpStatus.NOT_FOUND);
        }
}
```

Add the following codes in UserService interface:

```
Set<String> getAllFriendsByUserId(int userId);
```

Add the following codes in UserServiceImpl class:

```
// get all friends by user id
```

```
@Override
public Set<String> getAllFriendsByUserId(int userId) {

	User user = uRepo.findById(userId)
			.orElseThrow(() -> new IllegalStateException(
				"user with id" + userId + "doesn't exist"));

	// friends can be null
	Set<String> friendsEmails =
			user.getUserFriends()
				.stream()
				.map(User::getEmail)
				.collect(Collectors.toSet());

	return friendsEmails;
}
```

In the controller, we use **GetMapping** for reading transactions. Note we return a Set of String instead of User, since we want to show the email of friends which is more straightforward. In the service layer, we use java stream to get the email of friends and convert to collection of Set.

5.3 Get Nearby Friends

Add the following codes in UserDemoApiController class:

```
// get friends nearby
@GetMapping("/users/friendsnearby/{name}")
public ResponseEntity<List<String>> getFriendsNearby(
		@PathVariable("name") String name) {

	try {
		// only show the top 2 nearest friends
		List<String> friendsNearby =
				uService.getFriendsByDistance(name);

		return new ResponseEntity<List<String>>(
				friendsNearby, HttpStatus.OK);

	} catch(NoSuchElementException ex) {
		// log exception first, then return Conflict
		logger.error(ex.getMessage());

		return new ResponseEntity<List<String>>(
				HttpStatus.NOT_FOUND);
	}
}
```

Add the following codes in UserService interface:

```
List<String> getFriendsByDistance(String name);
```

Add the following codes in UserServiceImpl class:

```
// get nearby friends
@Override
public List<String> getFriendsByDistance(String name) {
        Set<Double> distanceFromFriends = new HashSet<Double>();
        Map<User, Double> friends_distances = new HashMap<User, Double>();
        Map<User, Double> friends_distances_sorted = new HashMap<User, Double>();
        List<String> FriendsEmail_distances_sorted = new ArrayList<String>();

        // there may be more than one user having the same username
        List<User> usersByName = uRepo.findUsersByName(name);

        if (CollectionUtils.isEmpty(usersByName)) {
                throw new IllegalStateException("user with name" + name + "doesn't exist");
        }

        for(User user: usersByName) {

                // get the user's address coordinates
                double userx = user.getAddress().getX();
                double usery = user.getAddress().getY();

                // get the friends of the user
                Set<User> friends = user.getUserFriends();

                for (User friend: friends) {
                        double friendx = friend.getAddress().getX();
                        double friendy = friend.getAddress().getY();
                        double distance = Math.sqrt((userx - friendx) * (userx - friendx) +
(usery - friendy) *  (usery - friendy));

                        // put in the map: friend is key, distance is value
                        friends_distances.put(friend, distance);
                }
        }

        // sort the map
        friends_distances.entrySet().stream()
                        .sorted(Map.Entry.<User, Double>comparingByValue())
                        .forEachOrdered(x -> friends_distances_sorted.put(x.getKey(),
x.getValue()));


        for(User friend: friends_distances_sorted.keySet()) {
                // get the top two nearest friends
                if (FriendsEmail_distances_sorted.size() < 2) {
                        FriendsEmail_distances_sorted.add(friend.getEmail());
                }
        }

        return FriendsEmail_distances_sorted;
}
```

Add the following codes in UserRepository interface:

```
List<User> findUsersByName(String username);
```

In the controller, we use **GetMapping** for reading transactions. Note we return a Set of String instead of User, since we want to show the email of friends which is more straightforward.

In order to differentiate from get all friends method, we only retrieve the top 2 nearest friends. We use the “address” attribute to get the geo-coordinates of the user and the friends, then calculate the distance and sort friends collection by distance in ascending order.

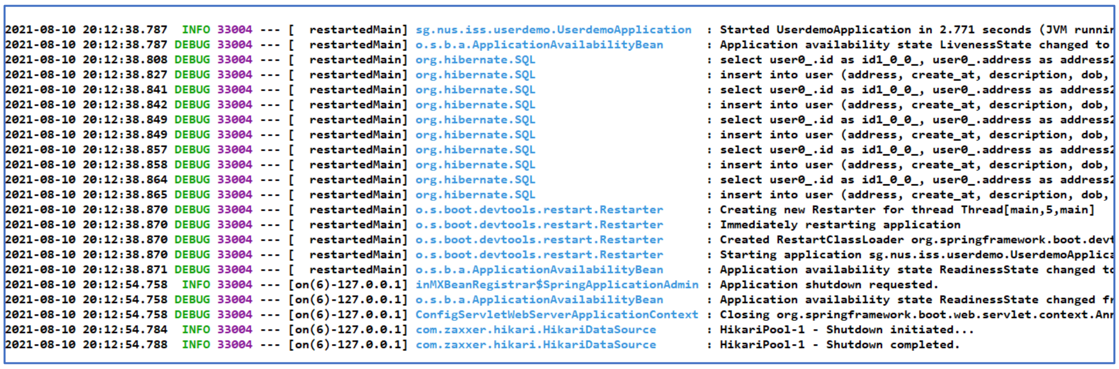
6. Test HTTP Requests in Postman

6.1 Run the application

Firstly, in the UserdemoApplication class under src/main/java/sg.nus.iss.userdemo directory, populate 5 User Instances in CommandLineRunner as the following code:

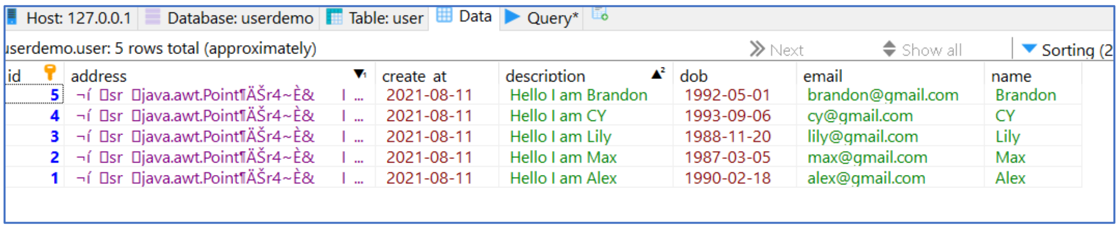
```
@SpringBootApplication
public class UserdemoApplication {

	@Autowired
	UserRepository uRepo;

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

	@Bean
	CommandLineRunner runner() {
		return args -> {

			User user1 = new User(
					1,
					"Alex",
					LocalDate.of(1990, 02, 18),
					new Point(1, 101),
					"Hello I am Alex",
					"alex@gmail.com",
					LocalDate.now());

			User user2 = new User(
					2,
					"Max",
					LocalDate.of(1987, 03, 05),
					new Point(1, 102),
					"Hello I am Max",
					"max@gmail.com",
					LocalDate.now());

			User user3 = new User(
					3,
					"Lily",
					LocalDate.of(1988, 11, 20),
					new Point(1, 103),
					"Hello I am Lily",
					"lily@gmail.com",
					LocalDate.now());

			User user4 = new User(
					4,
					"CY",
					LocalDate.of(1993, 9, 6),
					new Point(1, 104),
					"Hello I am CY",
					"cy@gmail.com",
					LocalDate.now());

			User user5 = new User(5,
					"Brandon",
					LocalDate.of(1992, 5, 1),
                                new Point(1, 105),
                                "Hello I am Brandon",
                                "brandon@gmail.com",
                                LocalDate.now());

                uRepo.save(user1);
                uRepo.save(user2);
                uRepo.save(user3);
                uRepo.save(user4);
                uRepo.save(user5);

        };
    }
}
```

Second, run userdemo project as Spring Boot App. We can see from the console that the application runs successfully without any error, refer to the capture below:



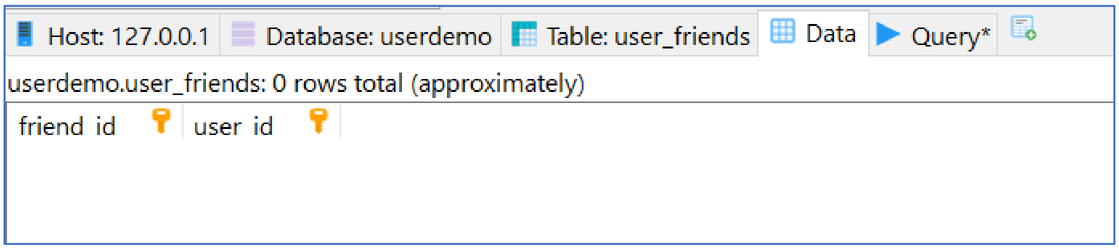
```
2021-08-10 20:12:38.787  INFO 33004 --- [  restartedMain] sg.nus.iss.userdemo.UserdemoApplication  : Started UserdemoApplication in 2.771 seconds (JVM runnir
2021-08-10 20:12:38.787 DEBUG 33004 --- [  restartedMain] o.s.b.a.ApplicationAvailabilityBean      : Application availability state LivenessState changed to
2021-08-10 20:12:38.808 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : select user0_.id as id1_0_0_, user0_.address as address2
2021-08-10 20:12:38.827 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : insert into user (address, create_at, description, dob,
2021-08-10 20:12:38.841 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : select user0_.id as id1_0_0_, user0_.address as address2
2021-08-10 20:12:38.842 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : insert into user (address, create_at, description, dob,
2021-08-10 20:12:38.849 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : select user0_.id as id1_0_0_, user0_.address as address2
2021-08-10 20:12:38.849 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : insert into user (address, create_at, description, dob,
2021-08-10 20:12:38.857 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : select user0_.id as id1_0_0_, user0_.address as address2
2021-08-10 20:12:38.858 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : insert into user (address, create_at, description, dob,
2021-08-10 20:12:38.864 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : select user0_.id as id1_0_0_, user0_.address as address2
2021-08-10 20:12:38.865 DEBUG 33004 --- [  restartedMain] org.hibernate.SQL                        : insert into user (address, create_at, description, dob,
2021-08-10 20:12:38.870 DEBUG 33004 --- [  restartedMain] o.s.boot.devtools.restart.Restarter      : Creating new Restarter for thread Thread[main,5,main]
2021-08-10 20:12:38.870 DEBUG 33004 --- [  restartedMain] o.s.boot.devtools.restart.Restarter      : Immediately restarting application
2021-08-10 20:12:38.870 DEBUG 33004 --- [  restartedMain] o.s.boot.devtools.restart.Restarter      : Created RestartClassLoader org.springframework.boot.devt
2021-08-10 20:12:38.870 DEBUG 33004 --- [  restartedMain] o.s.boot.devtools.restart.Restarter      : Starting application sg.nus.iss.userdemo.UserdemoApplica
2021-08-10 20:12:38.871 DEBUG 33004 --- [  restartedMain] o.s.b.a.ApplicationAvailabilityBean      : Application availability state ReadinessState changed to
2021-08-10 20:12:54.758  INFO 33004 --- [on(6)-127.0.0.1] inMXBeanRegistrar$SpringApplicationAdmin : Application shutdown requested.
2021-08-10 20:12:54.758 DEBUG 33004 --- [on(6)-127.0.0.1] o.s.b.a.ApplicationAvailabilityBean      : Application availability state ReadinessState changed fr
2021-08-10 20:12:54.758 DEBUG 33004 --- [on(6)-127.0.0.1] ConfigServletWebServerApplicationContext : Closing org.springframework.boot.web.servlet.context.Anr
2021-08-10 20:12:54.784  INFO 33004 --- [on(6)-127.0.0.1] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2021-08-10 20:12:54.788  INFO 33004 --- [on(6)-127.0.0.1] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

Third, let's look at the database in MySQL server, the five users are successfully populated into database. The user_friends table doesn't have any data record since we haven't added friends yet. Refer to the capture below:



userdemo.user: 5 rows total (approximately)

id	address	create at	description	dob	email	name
5	¬í Dsr Djava.awt.Point¶ÄŠr4~È& I ...	2021-08-11	Hello I am Brandon	1992-05-01	brandon@gmail.com	Brandon
4	¬í Dsr Djava.awt.Point¶ÄŠr4~È& I ...	2021-08-11	Hello I am CY	1993-09-06	cy@gmail.com	CY
3	¬í Dsr Djava.awt.Point¶ÄŠr4~È& I ...	2021-08-11	Hello I am Lily	1988-11-20	lily@gmail.com	Lily
2	¬í Dsr Djava.awt.Point¶ÄŠr4~È& I ...	2021-08-11	Hello I am Max	1987-03-05	max@gmail.com	Max
1	¬í Dsr Djava.awt.Point¶ÄŠr4~È& I ...	2021-08-11	Hello I am Alex	1990-02-18	alex@gmail.com	Alex



userdemo.user_friends: 0 rows total (approximately)

friend id	user id

6.2 Test getAllUsers()

Method: GET
Postman URL: http://localhost:8080/users
Result: return five User Objects.

```
[
    {
        "id": 1,
        "name": "Alex",
        "dob": "1990-02-18",
        "address": {
            "x": 1.0,
            "y": 101.0
        },
        "description": "Hello I am Alex",
        "email": "alex@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 2,
        "name": "Max",
        "dob": "1987-03-05",
        "address": {
            "x": 1.0,
            "y": 102.0
        },
        "description": "Hello I am Max",
        "email": "max@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 3,
        "name": "Lily",
        "dob": "1988-11-20",
        "address": {
            "x": 1.0,
            "y": 103.0
        },
        "description": "Hello I am Lily",
        "email": "lily@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 4,
        "name": "CY",
        "dob": "1993-09-06",
        "address": {
            "x": 1.0,
            "y": 104.0
        },
        "description": "Hello I am CY",
        "email": "cy@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 5,
        "name": "Brandon",
        "dob": "1992-05-01",
        "address": {
            "x": 1.0,
            "y": 105.0
        },
        "description": "Hello I am Brandon",
        "email": "brandon@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    }
]
```

6.3 Test getUserbyId()

Method: GET
Postman URL: http://localhost:8080/users/2
Result: return the User Object with id = 2

```
{
    "id": 2,
    "name": "Max",
    "dob": "1987-03-05",
    "address": {
        "x": 1.0,
        "y": 102.0
    },
    "description": "Hello I am Max",
    "email": "max@gmail.com",
    "createAt": "2021-08-11",
    "userFriends": []
}
```

6.4 Test addNewUser()

Method: POST
Postman URL: http://localhost:8080/users/adduser
Response Body:

```
{
    "name": "Ronnie",
    "dob": "1989-03-10",
    "description": "Hello I am Ronnie",
    "email": "ronnie@gmail.com"
}
```

Result: a new User Object with id = 6 is created. id = 6 is automatically generated because we have specified `@GeneratedValue (strategy = GenerationType.IDENTITY)`in Spring application.

```
{
    "id": 6,
    "name": "Ronnie",
    "dob": "1989-03-10",
    "address": null,
    "description": "Hello I am Ronnie",
    "email": "ronnie@gmail.com",
    "createAt": null,
    "userFriends": null
}
```

Let’s retrieve all users from the database to check whether the new user has been added into database.

Method: GET
Postman URL: http://localhost:8080/users
Result: return six User Objects, which represents the new user has been created in database.

```
[
    {
        "id": 1,
        "name": "Alex",
        "dob": "1990-02-18",
        "address": {
            "x": 1.0,
            "y": 101.0
        },
        "description": "Hello I am Alex",
        "email": "alex@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 2,
        "name": "Max",
        "dob": "1987-03-05",
        "address": {
            "x": 1.0,
            "y": 102.0
        },
        "description": "Hello I am Max",
        "email": "max@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 3,
        "name": "Lily",
        "dob": "1988-11-20",
        "address": {
            "x": 1.0,
            "y": 103.0
        },
        "description": "Hello I am Lily",
        "email": "lily@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 4,
        "name": "CY",
        "dob": "1993-09-06",
        "address": {
            "x": 1.0,
            "y": 104.0
        },
        "description": "Hello I am CY",
        "email": "cy@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 5,
        "name": "Brandon",
        "dob": "1992-05-01",
        "address": {
            "x": 1.0,
            "y": 105.0
        },
        "description": "Hello I am Brandon",
        "email": "brandon@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 6,
        "name": "Ronnie",
        "dob": "1989-03-10",
        "address": null,
        "description": "Hello I am Ronnie",
        "email": "ronnie@gmail.com",
        "createAt": null,
        "userFriends": []
    }
]
```

6.5 Test updateUser()

Method: PUT
Postman URL: http://localhost:8080/users/update/6
Response Body:

```
{
        "name": "Ronnie123",
        "dob": "1989-03-10",
        "description": "Hello I am Ronnie123",
        "email": "ronnie123@gmail.com"
}
```

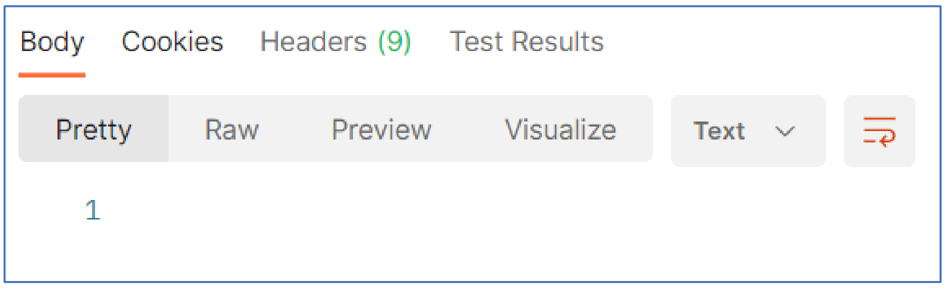
Result:

```
{
    "id": 6,
    "name": "Ronnie123",
    "dob": "1989-03-10",
    "address": null,
    "description": "Hello I am Ronnie123",
    "email": "ronnie123@gmail.com",
    "createAt": null,
    "userFriends": []
}
```

We update the attributes “name”, “description”, “email” of user id = 6. From the return result in postman, the information has been successfully updated.

6.6 Test deleteUser()

Method: DELETE
Postman URL: http://localhost:8080/users/delete/6
Result: return 1



```
1
```

Let’s retrieve all users from database to check whether user with id = 6 has been deleted.

Method: GET
Postman URL: http://localhost:8080/users
Result: return five User Objects, which represents the user with id = 6 has been deleted from database.

```
[
```

```json
    {
        "id": 1,
        "name": "Alex",
        "dob": "1990-02-18",
        "address": {
            "x": 1.0,
            "y": 101.0
        },
        "description": "Hello I am Alex",
        "email": "alex@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 2,
        "name": "Max",
        "dob": "1987-03-05",
        "address": {
            "x": 1.0,
            "y": 102.0
        },
        "description": "Hello I am Max",
        "email": "max@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 3,
        "name": "Lily",
        "dob": "1988-11-20",
        "address": {
            "x": 1.0,
            "y": 103.0
        },
        "description": "Hello I am Lily",
        "email": "lily@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 4,
        "name": "CY",
        "dob": "1993-09-06",
        "address": {
            "x": 1.0,
            "y": 104.0
        },
        "description": "Hello I am CY",
        "email": "cy@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    },
    {
        "id": 5,
        "name": "Brandon",
        "dob": "1992-05-01",
        "address": {
            "x": 1.0,
            "y": 105.0
        },
        "description": "Hello I am Brandon",
        "email": "brandon@gmail.com",
        "createAt": "2021-08-11",
        "userFriends": []
    }
]
```

6.7 Test userFriendRequest()

Method: POST
Postman URL: http://localhost:8080/users/userfriendsrequest
Response Body: we run three times

```
{
            "friends":
                    [
                    "max@gmail.com",
                    "lily@gmail.com"
                    ]
}
```

```
{
            "friends":
                    [
                    "max@gmail.com",
                    "cy@gmail.com"
                    ]
}
```

```
{
            "friends":
                    [
                    "max@gmail.com",
                    "brandon@gmail.com"
                    ]
}
```

Result: all three requests are successful.

```
{
    "Success": true
}
```

Let’s retrieve all friends for user with id = 2 and name = Max to check whether all three friends have been added.

Method: GET
Postman URL: http://localhost:8080/users/friends/2
Result: return three emails from corresponding three users, which represents all the three friends have been added to user named Max.

```
[
    "lily@gmail.com",
    "cy@gmail.com",
    "brandon@gmail.com"
]
```

Next, let’s retrieve all friends for user with id = 3 and name = Lily to check whether the user named Max has been added as friend to user named Lily.

Method: GET
Postman URL: http://localhost:8080/users/friends/3
Result: return the email of user named Max, which represents user named Max has been added as friend to user named Lily.

```
[
    "max@gmail.com"
]
```

6.8 Test getAllFriendsByuserId()

Method: GET
Postman URL: http://localhost:8080/users/friends/2
Result: return emails from three users, which represents all the three friends have been added to user named Max.

```
[
    "lily@gmail.com",
    "cy@gmail.com",
    "brandon@gmail.com"
]
```

6.9 Test getFriendsNearby()

Method: GET
Postman URL: http://localhost:8080/users/friendsnearby/Max

Result: return the emails of only two users, since Lily and CY are the top 2 nearest friends to Max.

```
[
    "lily@gmail.com",
    "cy@gmail.com"
]
```

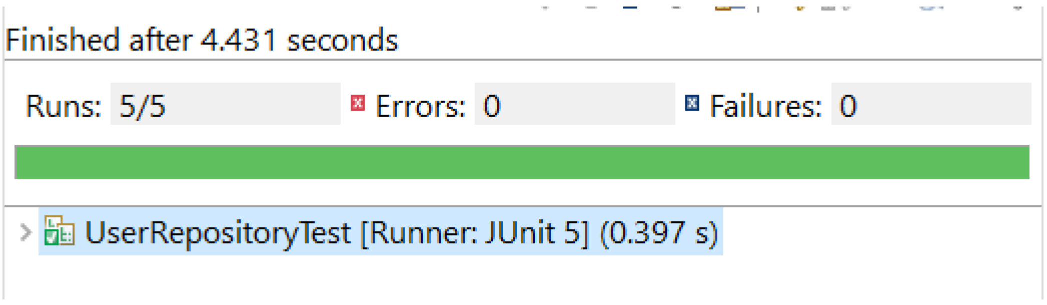
7. Implement Unit Test

7.1 DAO Layer Unit Test

First, create a UserRepositoryTest class under src/test/java/sg.nus.iss.userdemo directory with the following code:

```
@ExtendWith(SpringExtension.class)
@SpringBootTest
@TestMethodOrder(OrderAnnotation.class)
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
public class UserRepositoryTest {


        @Autowired
        private UserRepository urepo;

        @Test
        @Order(1)
        void testCreateUser() {
                //given
                User user = new User(6, "Ronnie", LocalDate.of(1987, 6, 1),
                                "Hello I am Ronnie", "ronnie@gmail.com", LocalDate.now());

                // when
                User saved = urepo.save(user);
                // then
                assertNotNull(saved);
        }


        @Test
        @Order(2)
        void testGetUserByEmail() {

                //given
                String email= "ronnie@gmail.com";

                //when
                User saved = urepo.findByEmail(email);

                //then
                assertNotNull(saved);
        }

        @Test
        @Order(3)
        public void testUpdateUser() {

                //given
                String email= "ronnie@gmail.com";
                User given = urepo.findByEmail(email);

                //when
                given.setEmail("ronnie123@gmail.com");
                User saved = urepo.save(given);

                //then
                assertNotNull(saved);

        }


        @Test
        @Order(4)
        public void testListUsers() {
                // given
                List<User> list = new ArrayList<User>();
                // when
                list = urepo.findAll();
                // then
                assertTrue(list.size() > 0);
        }

        @Test
        @Order(5)
        public void testDeleteUsers() {
                // given
                String email = "ronnie123@gmail.com";
                // when
                User selected = urepo.findByEmail(email);
                urepo.delete(selected);
                // then
                assertTrue(urepo.findByEmail(email) == null);
        }

}
```

Next, run this class file as Junit Test. We can see the green bar in the below capture, which represents the all five tests successfully.


```
Finished after 4.431 seconds

Runs: 5/5        Errors: 0        Failures: 0

UserRepositoryTest [Runner: JUnit 5] (0.397 s)
```

7.2 Controller Layer Unit Test

First, create a UserDemoApiControllerTest class under src/test/java/sg.nus.iss.userdemo directory with the following code:

```
@SpringBootTest
@AutoConfigureMockMvc
public class UserDemoApiControllerTest {
        @Autowired
        MockMvc mockMvc;

        @Autowired
        ObjectMapper mapper;

        @MockBean
        UserService uService;

        @Autowired
    WebApplicationContext wac;


        // define dummy data
        User user1 = new User(1, "Alex",
                        LocalDate.of(1990, 02, 18), "Hello I am Alex",
                         "alex@gmail.com", LocalDate.now());

        User user2 = new User(2, "Max",
                         LocalDate.of(1987, 03, 05), "Hello I am Max",
                         "max@gmail.com", LocalDate.now());

        User user3 = new User(3, "Lily",
                         LocalDate.of(1988, 11, 20), "Hello I am Lily",
                         "lily@gmail.com", LocalDate.now());

        User user4 = new User(4, "CY",
                         LocalDate.of(1993, 9, 6), "Hello I am CY",
                         "cy@gmail.com", LocalDate.now());

        User user5 = new User(5, "Brandon",
                         LocalDate.of(1992, 5, 1), "Hello I am Brandon",
                         "brandon@gmail.com", LocalDate.now());



        // test the method getAllUsers()
        @Test
        public void getAllUsersTest() throws Exception{
                List<User> records = new ArrayList<User>();
                records.add(user1);
                records.add(user2);
                records.add(user3);
                records.add(user4);
                records.add(user5);

                Mockito.when(uService.findAllUsers()).thenReturn(records);

                // Execute the GET request
                mockMvc.perform(get("/users"))
                                // validate the response code and content type
                                .andExpect(status().isOk())
                                .andExpect(content().contentType(MediaType.APPLICATION_JSON))
                                // validate the returned fields
                                .andExpect(jsonPath("$", hasSize(5)))
                                .andExpect(jsonPath("$[0].name", is("Alex")))
                                .andExpect(jsonPath("$[1].name", is("Max")))
                                .andExpect(jsonPath("$[2].name", is("Lily")))
                                .andExpect(jsonPath("$[3].name", is("CY")))
                                .andExpect(jsonPath("$[4].name", is("Brandon")));

        }

        // Test the method getUserById()
        @Test
        public void getUserbyIdTest() throws Exception{

                Mockito.when(uService.findUserById(1)).thenReturn(user1);

                        mockMvc.perform(MockMvcRequestBuilders.get("/users/1")
                        .contentType(MediaType.APPLICATION_JSON))
                        .andExpect(status().isOk())
                        .andExpect(jsonPath("$", notNullValue()))
                        .andExpect(jsonPath("$.name", is("Alex")));
        }

        // Test the method addNewUser()
        @Test
        public void addNewUserTest() throws Exception {
                User user6 = new User(6, "Ronnie", LocalDate.of(1987, 6, 1),
                                "Hello I am Ronnie", "ronnie@gmail.com", LocalDate.now());

                MockHttpServletRequestBuilder mockRequest =
                                MockMvcRequestBuilders.post("users/adduser")
                        .contentType(MediaType.APPLICATION_JSON)
                        .accept(MediaType.APPLICATION_JSON)
                        .content(this.mapper.writeValueAsString(user6));

                        mockMvc.perform(mockRequest)
                                        .andExpect(status().isOk())
```

```
                                        .andExpect(jsonPath("$.name", is("Ronnie")))
                                        .andExpect(jsonPath("$.description", is("Hello I am 
Ronnie")))
                                        .andExpect(jsonPath("$.email", 
is("ronnie@gmail.com")));

        }

        // Test the method deleteUser()
        @Test
        public void delelteUserTest() throws Exception{
                int userid = 1;
                Mockito.when(uService.findUserById(userid)).thenReturn(user1);

                mockMvc.perform(MockMvcRequestBuilders.delete("/users/delete/1")
                                .contentType(MediaType.APPLICATION_JSON))
                                .andExpect(status().isOk());

        }

        // Test the method updateUser()
        @Test
        public void updateUserTest() throws Exception {

                // update name and email
                user2.setName("max123");
                user2.setEmail("max123@gmail.com");


                MockHttpServletRequestBuilder mockRequest = 
                                MockMvcRequestBuilders.put("/users/update/2")
                        .contentType(MediaType.APPLICATION_JSON)
                        .accept(MediaType.APPLICATION_JSON)
                        .content(this.mapper.writeValueAsString(user2));

                mockMvc.perform(mockRequest)
                                .andExpect(status().isOk())
                                .andExpect(jsonPath("$.id", is(2)))
                                .andExpect(jsonPath("$.name", is("max123")))
                                .andExpect(jsonPath("$.email",is("max123@gmail")));

        }
}
```

Next, run this class file as Junit Test. Currently the test fail to run due to the version of Mockito library is incompatible with the version of Spring Boot and Junit Test. We will explore it further.

8. Implement OAuth2 Authentication

8.1 Configuration

First, create an AppSecurityConfig class under src/test/java/sg.nus.iss.userdemo.configuration directory with the following code:

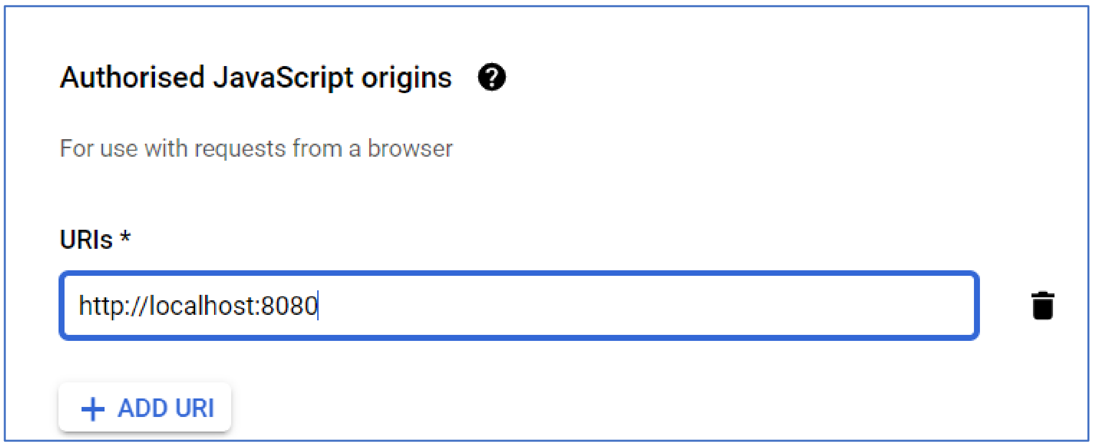
```
@Configuration
@EnableWebSecurity
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {


        @Override
        public void configure(HttpSecurity http) throws Exception {
                http.antMatcher("/**").authorizeRequests()
```

```
                    .antMatchers("/").permitAll()
                    .anyRequest().authenticated()
                    .and()
                    .oauth2Login();

        }
}
```

Then, dd the following codes in UserDemoApiController class:

```
// test OAuth2 login, no need to login
      @GetMapping("/")
      public String helloWorld() {
            return "Hello World! you don't need to be logged in.";
      }

      // test OAuth2 login, need to login
      @GetMapping("/restricted")
      public String restricted() {
            return "if you see this you are logged in";
      }
```

Next, we configure the credentials in Google account, and get the client-id and client-secret.

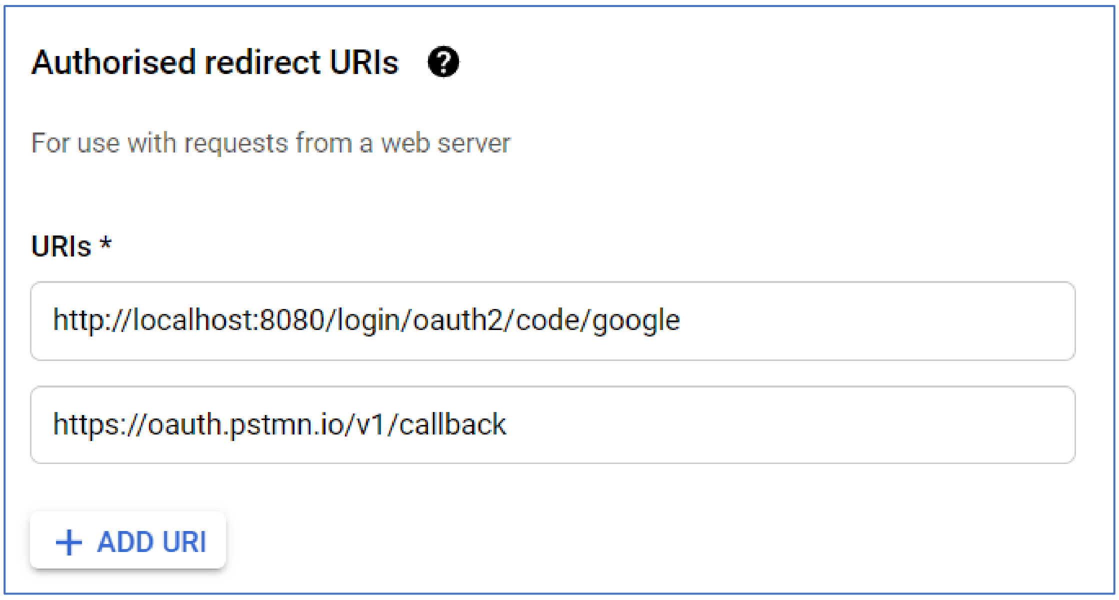


Authorised JavaScript origins

For use with requests from a browser

URIs *

http://localhost:8080



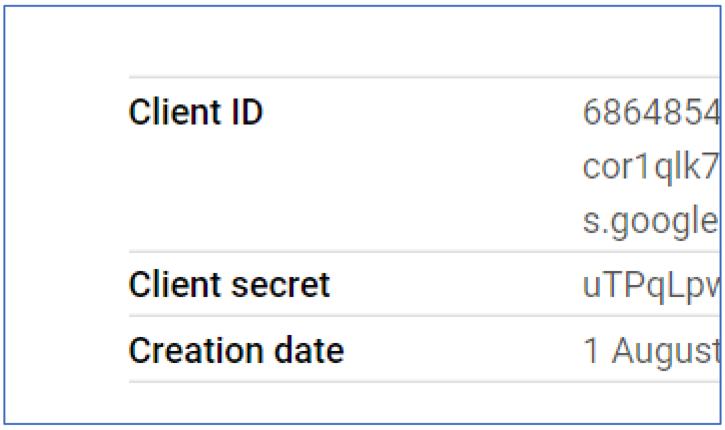
Authorised redirect URIs

For use with requests from a web server

URIs *

http://localhost:8080/login/oauth2/code/google

https://oauth.pstmn.io/v1/callback



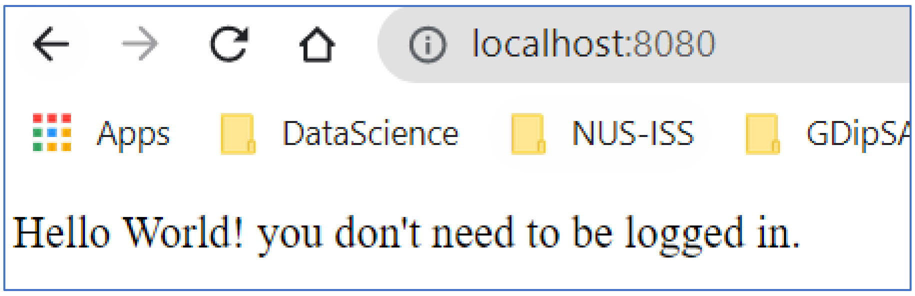
Client ID	6864854 cor1qlk7 s.google
Client secret	uTPqLpv
Creation date	1 August

Next, specify the client-id and client-secret in application.properties file.

```
spring.security.oauth2.client.registration.google.client-id=
spring.security.oauth2.client.registration.google.client-secret=
```

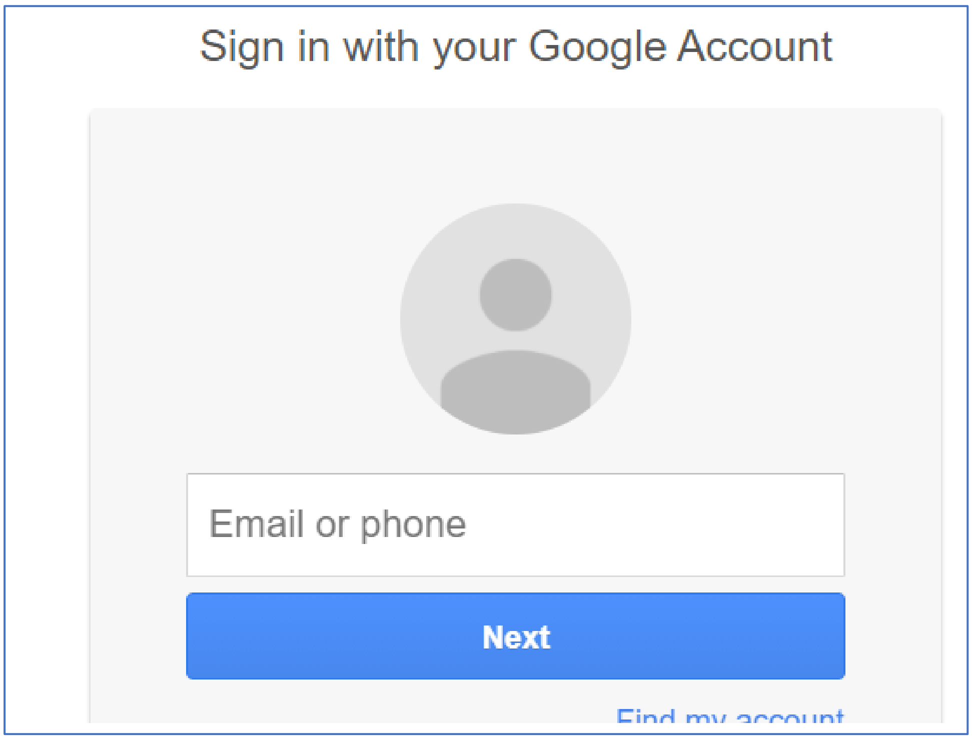
8.2 Test OAuth2

Run the application, enter url http://localhost:8080 in the browser, we can see the following output, which represents this url doesn't require login.

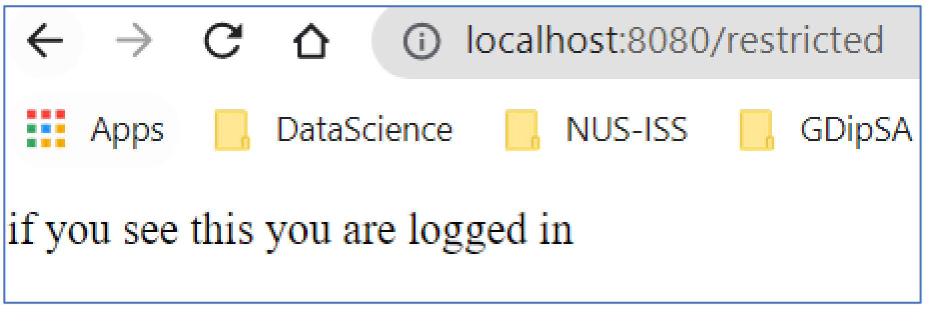


Hello World! you don't need to be logged in.

Enter url http://localhost:8080/restricted in the browser, it will redirects to the login page of Google Account.



After login, we can see the following output, which represents we have already logged in.



```
if you see this you are logged in
```

9. Implement Logger

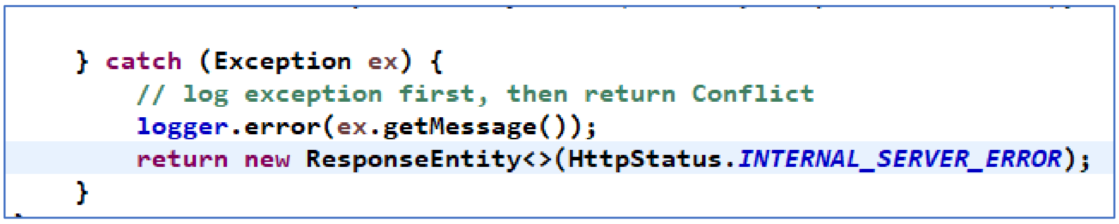
Specify the following codes in application.properties file.

```
# Logger settings
debug=true
logging.level.org.springframework=ERROR
```

Add the following codes in UserDemoApiController class.

```
private final Logger logger =
                          LoggerFactory.getLogger(UserDemoApiController.class);
```

Specify the following codes in the catch block in UserDemoApiController class.



```
    } catch (Exception ex) {
        // log exception first, then return Conflict
        logger.error(ex.getMessage());
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
```

10. Conclusions

We have gone through the implementation process for the userdemo Restful API.

We have showed how to implement CRUD of users and the “Friends” function. Postman is adopted to test http request.

In Unit Test, DAO Layer and the Controller Layer are testes. DAO Layer test runs successfully while the controller layer test fails due to the Mockito library version which requires further exploration.

We also demonstrated how to configure the OAuth Authentication and the Logger Strategy.