ASSESSMENT 2

In this assignment, we had two problems. I chose to solve them by using the LogIn screen, which is the first of the Figma UI designs, as an example. Any student entering the system through the LogIn screen would have their email and password automatically registered in the system database. For the first problem, I organized the data saved with the HTTP POST method in MySQL by modifying SpringBoot and placing it in the tables.

I initially started this task with Android Studio.

Android Studio Implementation:

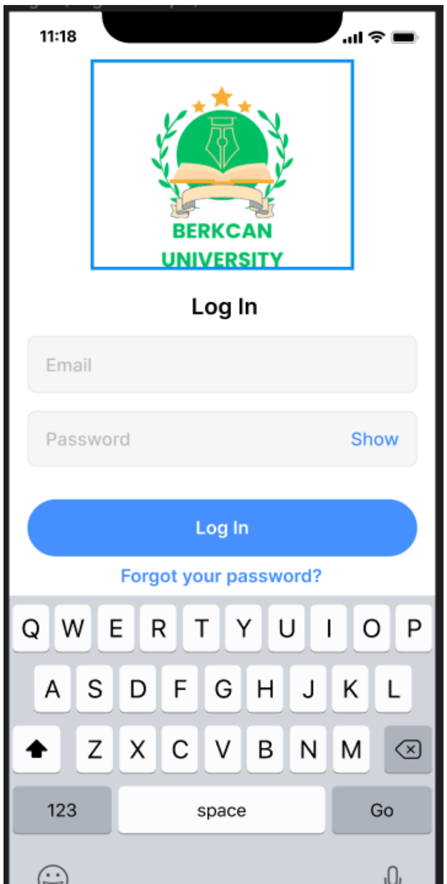
Problem 1: User Registration and Form Submission

Android UI Design

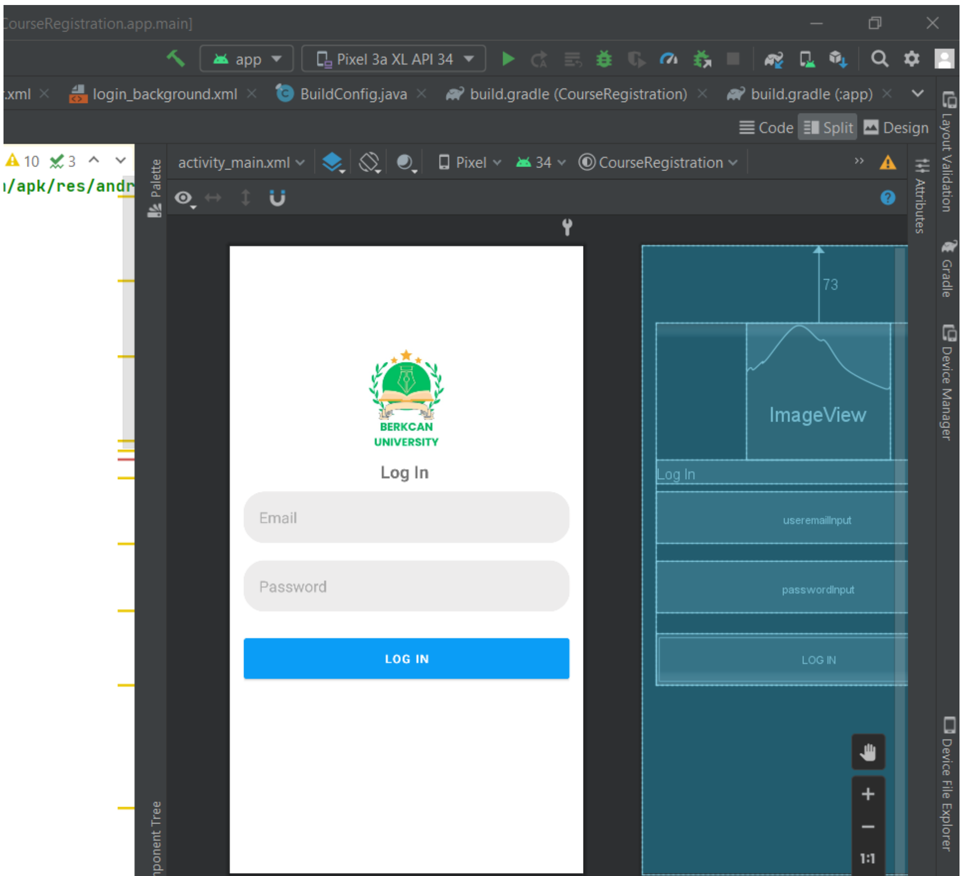
Created the similar UI (Figma UI-1) using XML in activity_main.xml for user email, password input, and a login button.

Utilized RoundedCorner drawable for styling the EditText components.

Figma UI-1



Android Studio (activity_main.xml)

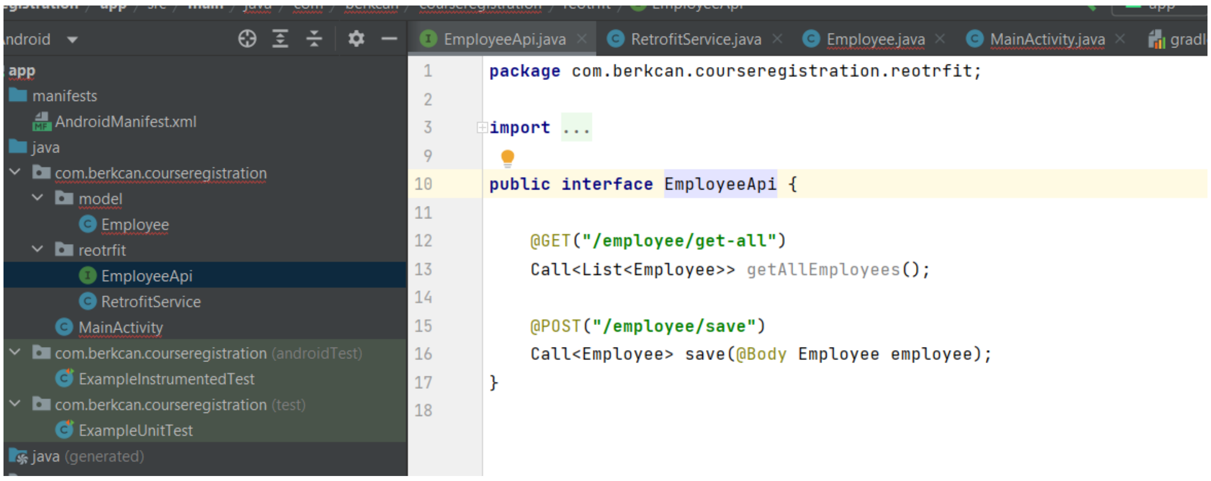


2: Retrofit Service Setup

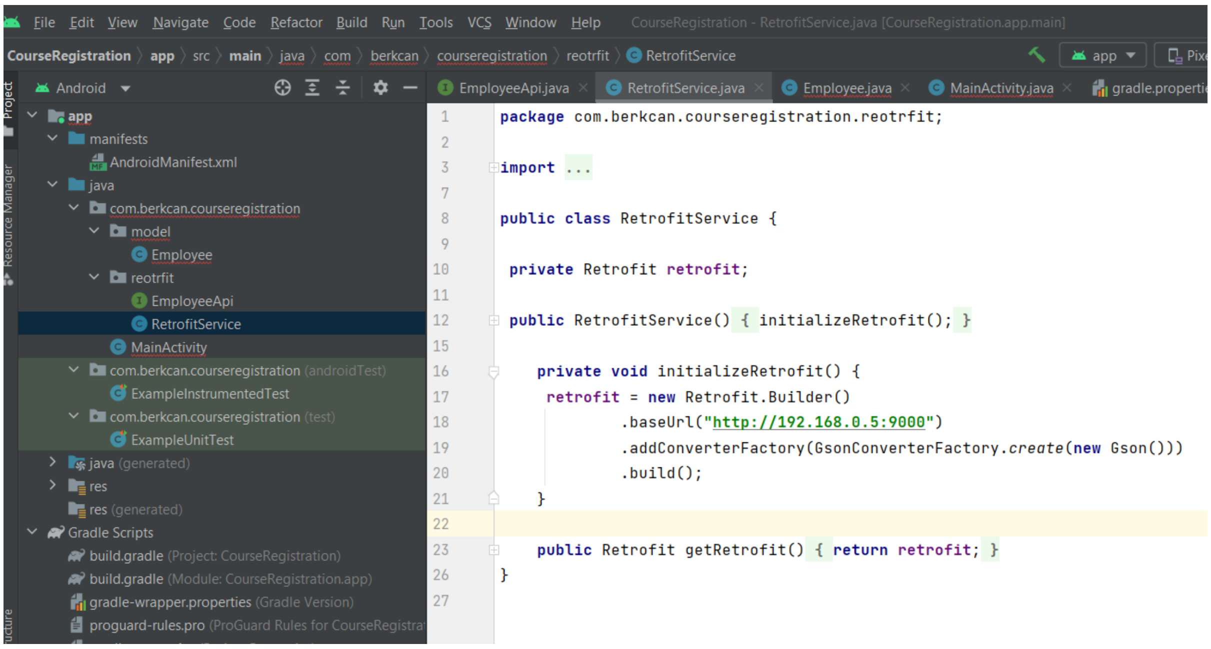
Developed RetrofitService class to initialize Retrofit for communication with the Spring Boot backend.

Specified the base URL as "http://192.168.0.5:9000". (Initiated the same port as running at IntelliJ IDEA SpringBoot)

Used ConverterFactory to handle JSON serialization/deserialization with Gson. (Because We can not contact directly in JSON with System)



```java
package com.berkcan.courseregistration.reotrfit;

import ...

public interface EmployeeApi {

    @GET("/employee/get-all")
    Call<List<Employee>> getAllEmployees();

    @POST("/employee/save")
    Call<Employee> save(@Body Employee employee);
}
```



```java
package com.berkcan.courseregistration.reotrfit;

import ...

public class RetrofitService {

 private Retrofit retrofit;

 public RetrofitService() { initializeRetrofit(); }

    private void initializeRetrofit() {
     retrofit = new Retrofit.Builder()
             .baseUrl("http://192.168.0.5:9000")
             .addConverterFactory(GsonConverterFactory.create(new Gson()))
             .build();
    }

    public Retrofit getRetrofit() { return retrofit; }
}
```

3: MainActivity Implementation

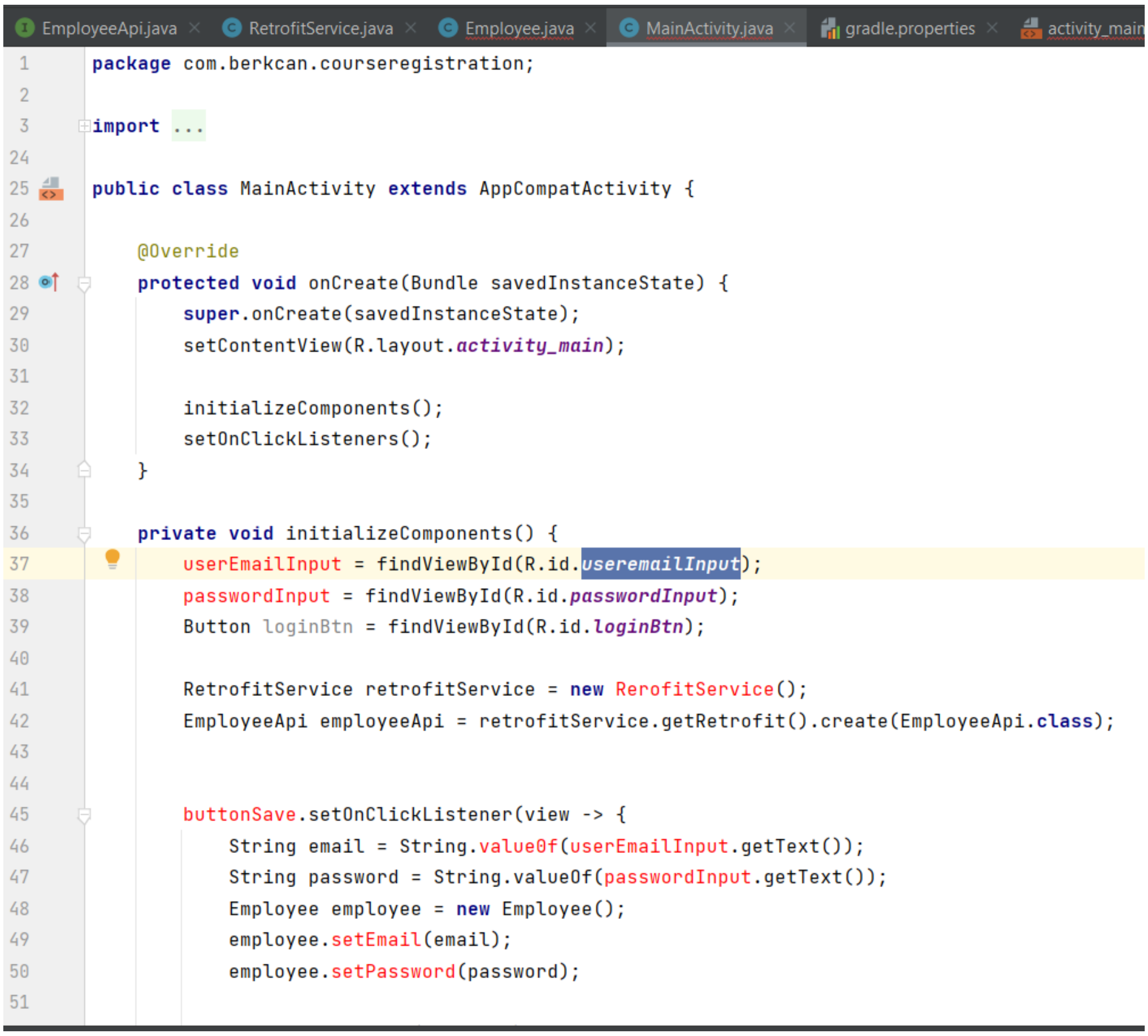
Extracted user input from the UI components.

Created an instance of the RetrofitService and obtained the EmployeeApi interface.

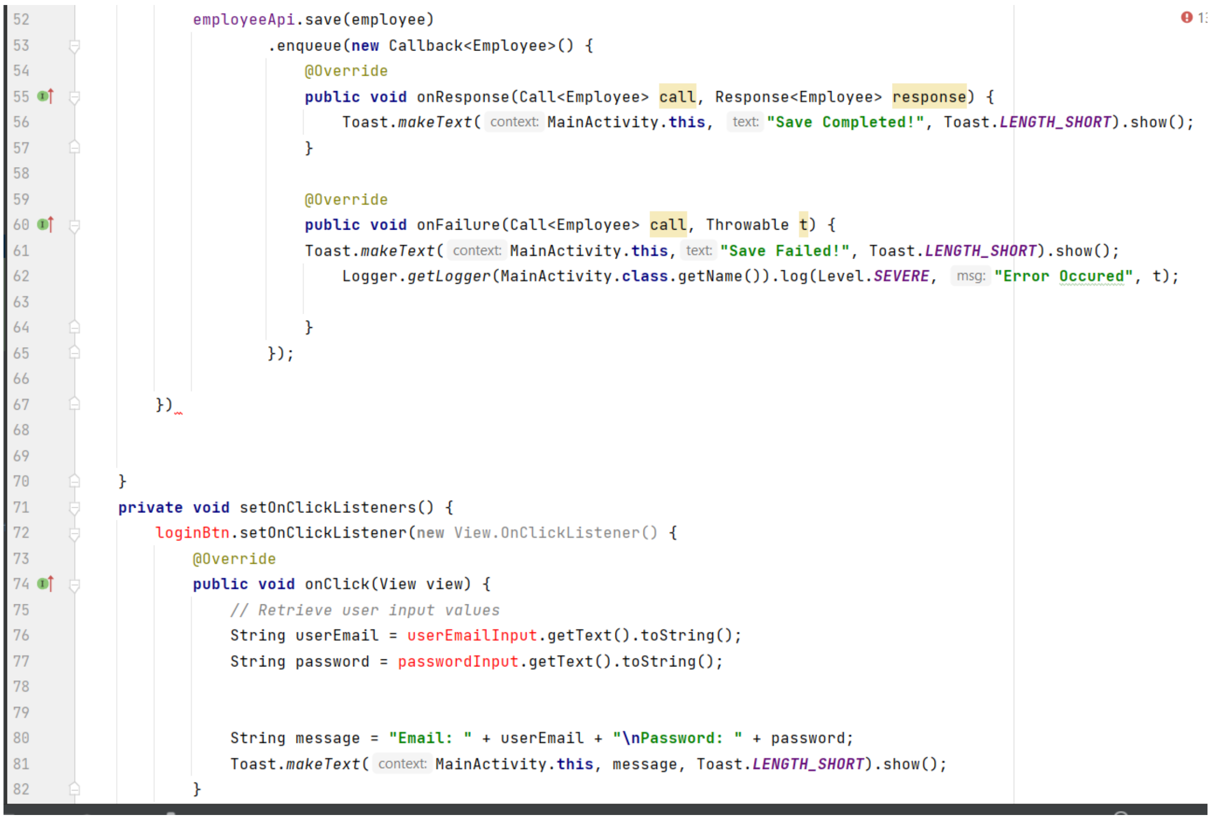
Utilized Retrofit to send an HTTP POST request to save user registration data.

Displayed appropriate toast

s based on the success or failure of the registration.



```java
package com.berkcan.courseregistration;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initializeComponents();
        setOnClickListeners();
    }

    private void initializeComponents() {
        userEmailInput = findViewById(R.id.useremailInput);
        passwordInput = findViewById(R.id.passwordInput);
        Button loginBtn = findViewById(R.id.loginBtn);

        RetrofitService retrofitService = new RerofitService();
        EmployeeApi employeeApi = retrofitService.getRetrofit().create(EmployeeApi.class);


        buttonSave.setOnClickListener(view -> {
            String email = String.valueOf(userEmailInput.getText());
            String password = String.valueOf(passwordInput.getText());
            Employee employee = new Employee();
            employee.setEmail(email);
            employee.setPassword(password);

```



```java
                employeeApi.save(employee)
                        .enqueue(new Callback<Employee>() {
                            @Override
                            public void onResponse(Call<Employee> call, Response<Employee> response) {
                                Toast.makeText( context: MainActivity.this,  text: "Save Completed!", Toast.LENGTH_SHORT).show();
                            }

                            @Override
                            public void onFailure(Call<Employee> call, Throwable t) {
                            Toast.makeText( context: MainActivity.this, text: "Save Failed!", Toast.LENGTH_SHORT).show();
                                Logger.getLogger(MainActivity.class.getName()).log(Level.SEVERE,  msg: "Error Occured", t);

                            }
                        });

        })


    }
    private void setOnClickListeners() {
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Retrieve user input values
                String userEmail = userEmailInput.getText().toString();
                String password = passwordInput.getText().toString();


                String message = "Email: " + userEmail + "\nPassword: " + password;
                Toast.makeText( context: MainActivity.this, message, Toast.LENGTH_SHORT).show();
            }
```

Problem 2: Displaying Email LoginList with Filtering

1: Retrofit API Interface

Modified EmployeeApi interface to include an HTTP GET request method for fetching the

[Grab your reader's attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

email lists.

Employee.java

```
package com.genuinecoder.springserver.model.employee;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  private String email;
  private String password;
 public int getId() {
   return id;
  }
public void setId(int id) {
   this.id = id;
  }
 public String getEmail() {
   return email;
  }
public void setEmail(String email) {
   this.email = email;
  }
public String getPassword() {
   return password;
  }
public void setPassword(String password) {
   this.password = password;
  }
@Override
  public String toString() {
   return "Employee{" +
        "id=" + id +
        ", email='" + email + '\'' +
        ", password='" + password + '\'' +
        '}';
  }
```

database using Java Persistence API

2: Updated MainActivity

Added UI elements to display the fetched course list.

Integrated Retrofit to make an HTTP GET request to retrieve the email list.

Implemented filtering options for the displayed course list based on user input.

2- IntelliJ (Spring Boot) Implementation:

1: Employee Entity and Repository

Created an Employee entity class annotated with JPA annotations for database interaction.

Developed an EmployeeRepository interface extending CrudRepository for basic CRUD operations.

2: EmployeeController

Implemented an EmployeeController class with @RestController annotation.

Added methods for handling HTTP GET (/employee/get-all) and POST (/employee/save) requests.

Utilized EmployeeDao for data manipulation.

3: EmployeeDao

Created EmployeeDao service class to interact with the repository.

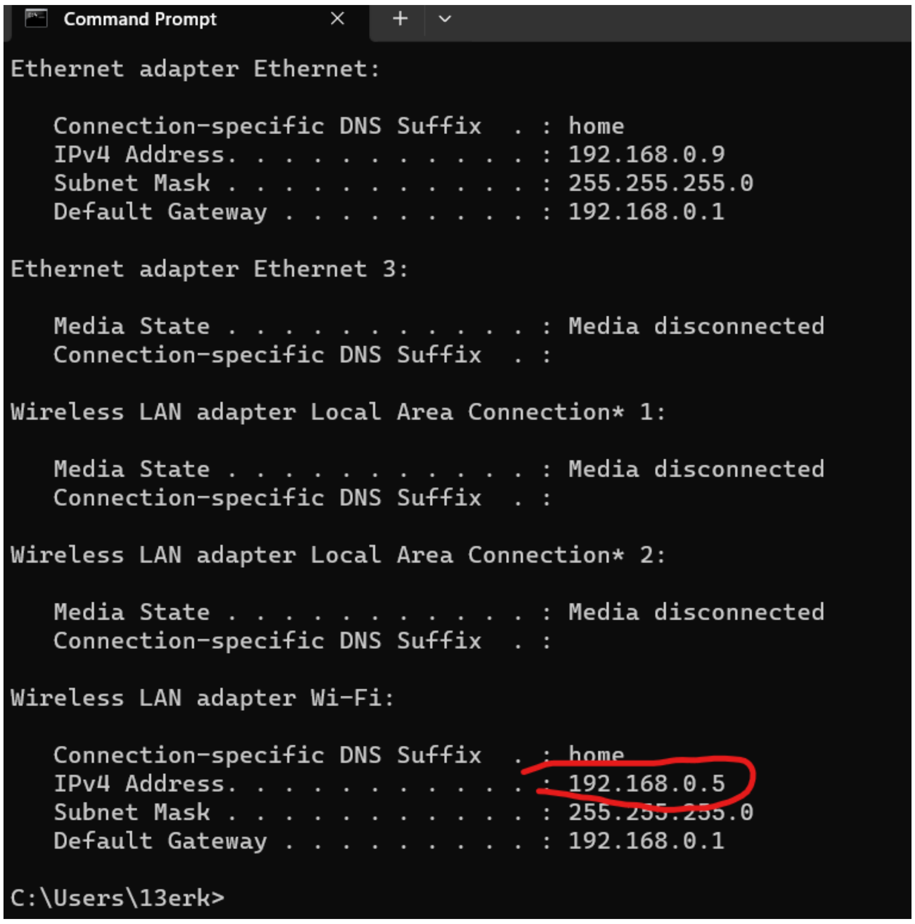
Implemented methods for saving, retrieving, and deleting employees.

4: Spring Boot Application

Configured a Spring Boot application class (SpringServerApplication) with @SpringBootApplication.

Defined @GetMapping and @PostMapping endpoints in the EmployeeController.

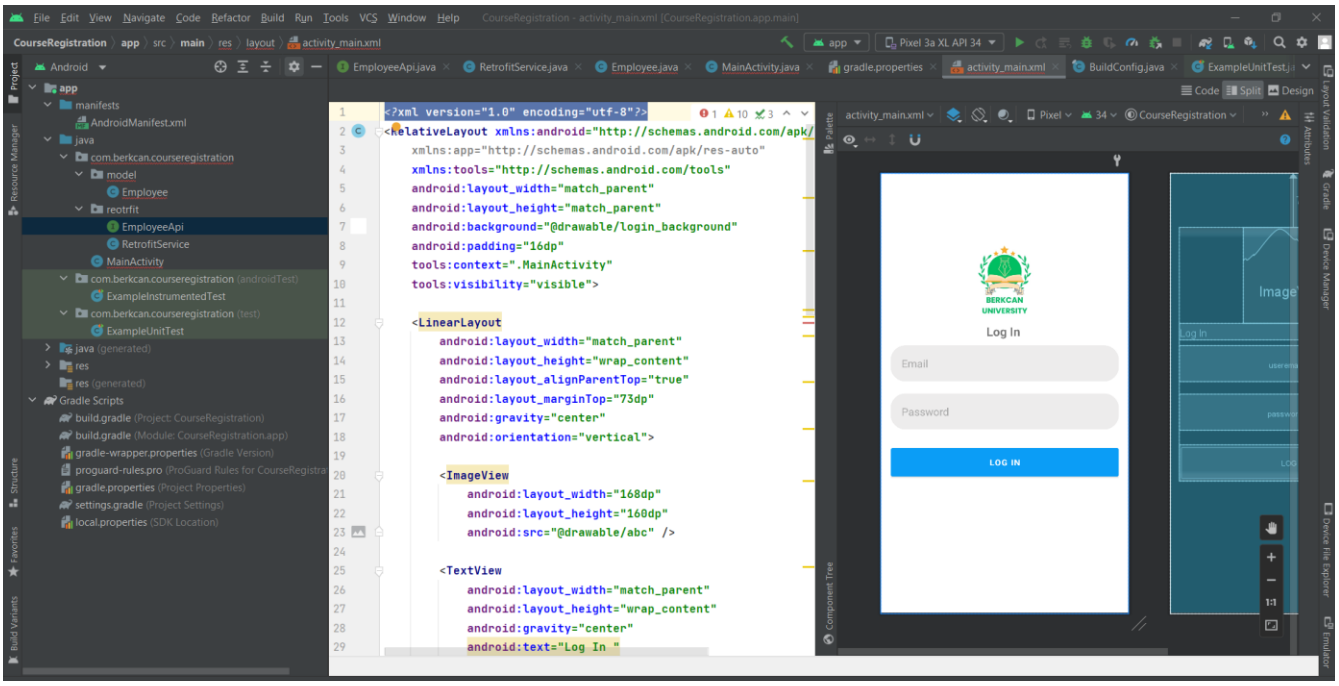
Configured Ipconfig



```
Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : home
   IPv4 Address. . . . . . . . . . . : 192.168.0.9
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1

Ethernet adapter Ethernet 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : home
   IPv4 Address. . . . . . . . . . . : 192.168.0.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1

C:\Users\13erk>
```

Testing:

Android Studio Testing:
Used the emulator to test the Android application.
Verified the correctness of UI elements, user input handling, and HTTP requests.



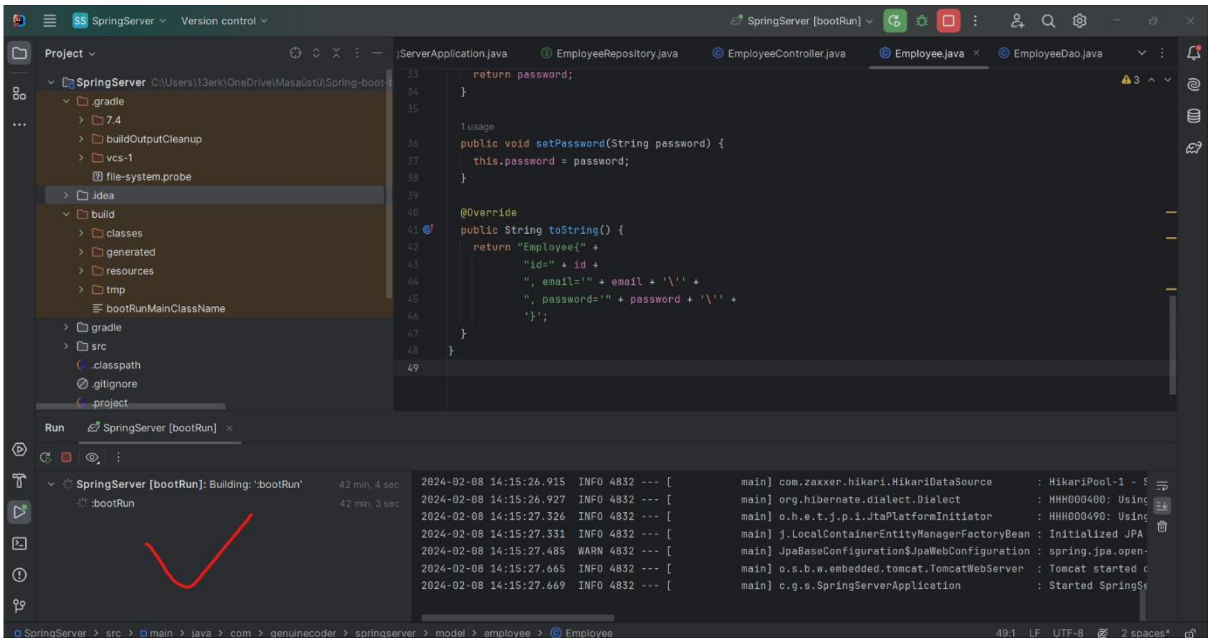
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/login_background"
    android:padding="16dp"
    tools:context=".MainActivity"
    tools:visibility="visible">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="73dp"
        android:gravity="center"
        android:orientation="vertical">

        <ImageView
            android:layout_width="168dp"
            android:layout_height="160dp"
            android:src="@drawable/abc" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="Log In "
```

IntelliJ Testing:

Executed JUnit tests using SpringServerApplicationTests to ensure correct functionality.

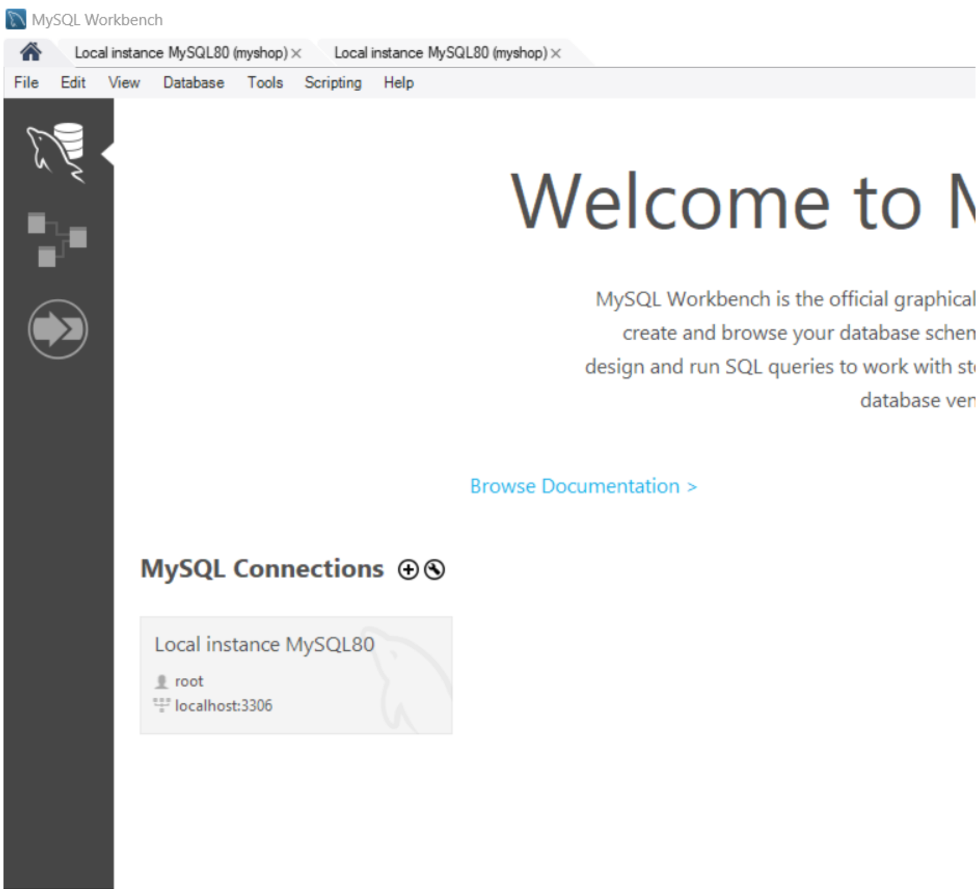
Manually tested the endpoints using tools like Postman to verify HTTP GET and POST operations.

Connected to MySQL



```java
        return password;
    }

    1 usage
    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", email='" + email + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

```
2024-02-08 14:15:26.915  INFO 4832 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - S
2024-02-08 14:15:26.927  INFO 4832 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using
2024-02-08 14:15:27.326  INFO 4832 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using
2024-02-08 14:15:27.331  INFO 4832 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA
2024-02-08 14:15:27.485  WARN 4832 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-
2024-02-08 14:15:27.665  INFO 4832 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started c
2024-02-08 14:15:27.669  INFO 4832 --- [           main] c.g.s.SpringServerApplication            : Started SpringSe
```



Welcome to M

MySQL Workbench is the official graphical
create and browse your database schen
design and run SQL queries to work with st
database ver

MySQL Connections

Local instance MySQL80
root
localhost:3306

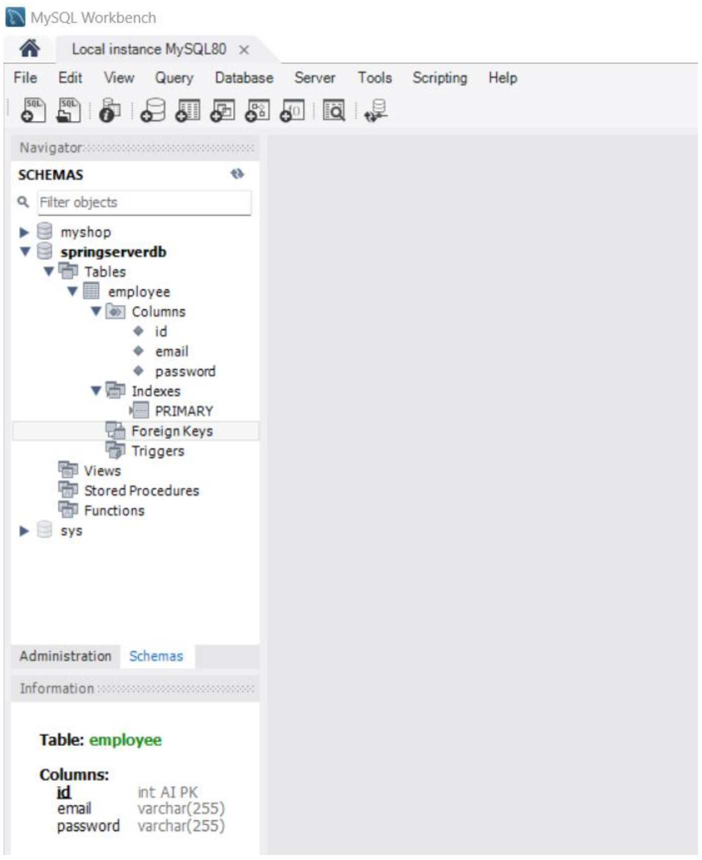


Table: employee

Columns:

```
id        int AI PK
email     varchar(255)
password  varchar(255)
```

Summary

The project was conceived with success in mind; however, due to coding errors, the full implementation couldn't be achieved as initially planned. Despite encountering challenges in the coding process, the integration of Android Studio and IntelliJ, along with the utilization of Retrofit and Spring Boot, laid a foundation for a functional Course Registration System. The Android client-side handles user registration seamlessly through HTTP POST requests, and the backend, despite some coding errors, manages to provide course lists with filtering capabilities via HTTP GET requests. This project serves as a valuable learning experience, highlighting areas for improvement and offering insights into the complexities of system integration.