

**Assignment Cover Letter****(Individual Work)****Student Information:**

1.

Surname

Sanjaya

Given Names

Hengky

Student ID Number

2201852492

Course Code : COMP6510**Course Name** : Programming Languages**Class** : L2CC**Name of Lecturer(s)** : Jude Joseph Lamug Martinez**Major** : Computer Science**Title of Assignment** : Coffee Shop Android App using Java API**Type of Assignment** : Final Project**Submission Pattern****Due Date** :**Submission Date** :

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

(Name of Student)

Hengky Sanjaya

Table of Contents

Plagiarism/Cheating	1
Declaration of Originality	1
I. Program Description	3
II. Class Diagram	3
III. Application Flow.....	5
IV. Lessons that Have Been Learned	7
V. Project Technical Description	7
VI. Code Explanation	9
VII. Project Link.....	14
VIII. References	14

“Coffee Shop Android App using Java API”

Name : Hengky Sanjaya

ID : 2201852492

I. Program Description

This is a simple Coffee shop application running on android platform integrated with API (Application Programming Interface) as a middle-man for the exchange of information between two services. The API is built using Spring Boot framework using Java language meanwhile the android app is built using Kotlin language.

As a coffee lover this application is very recommended for you to find and get more information about coffee while enjoying a cup of coffee. This application provides an easy way and user friendly for you to order coffee. It's also provides several features for the admin to manage their coffee shop such as manage their variant of coffees and view report about how well their coffee sold.

II. Class Diagram

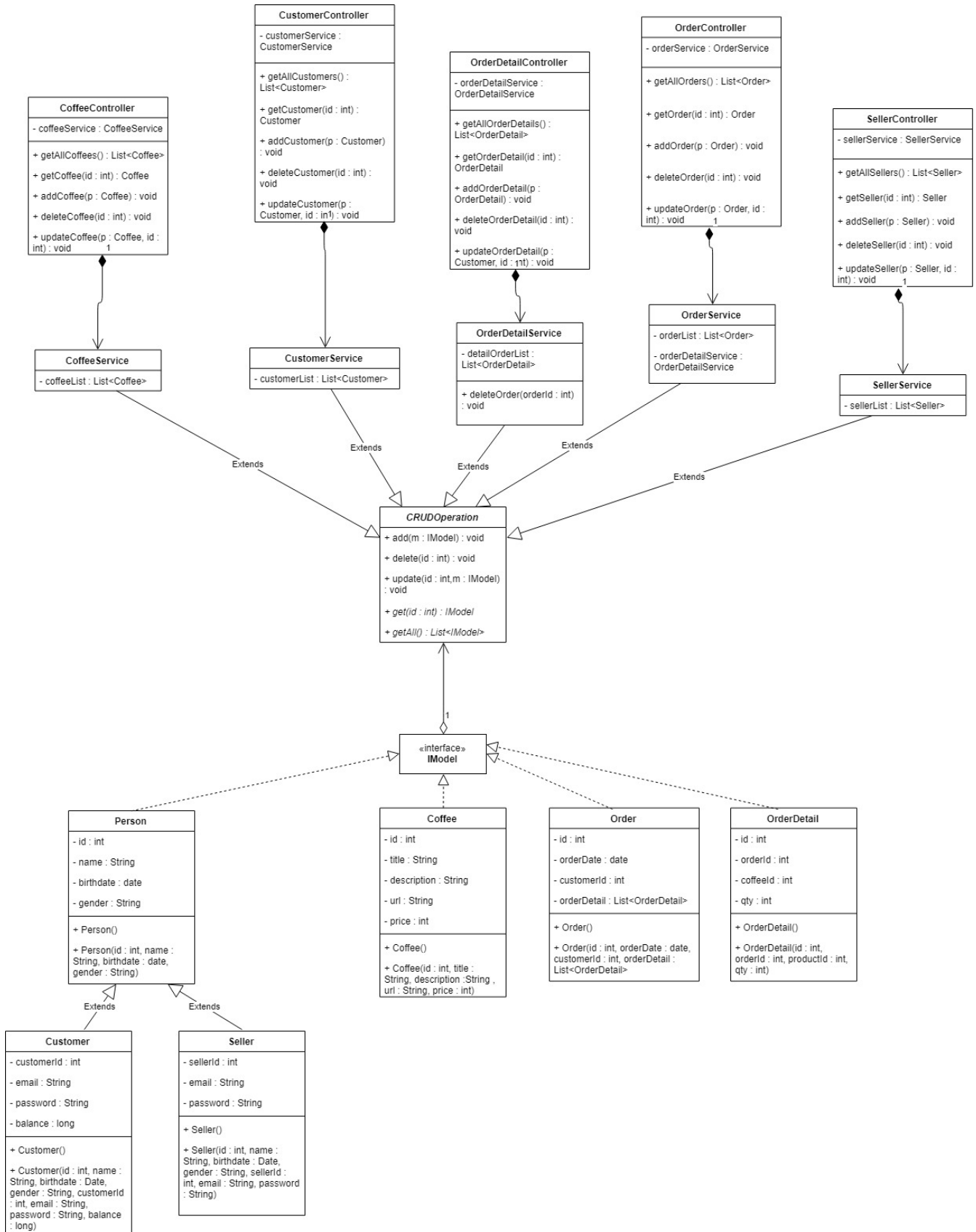
- Coffee controller, Customer controller, OrderDetail controller, Order controller, Seller controller to accept input and performs the corresponding update
- Coffee service, Customer service, OrderDetail service, Order service, Seller service to store the corresponding data and operations
- **abstract class** “CRUDOperation” as an abstract parent class and as an outline of all the services class such as add, edit, delete, and update, and retrieve operations
- **interface** “IModel” to set all the model classes into one single unit so we can use the “IModel” interface as parameter and it automatically able to accepts all model.

For examples:

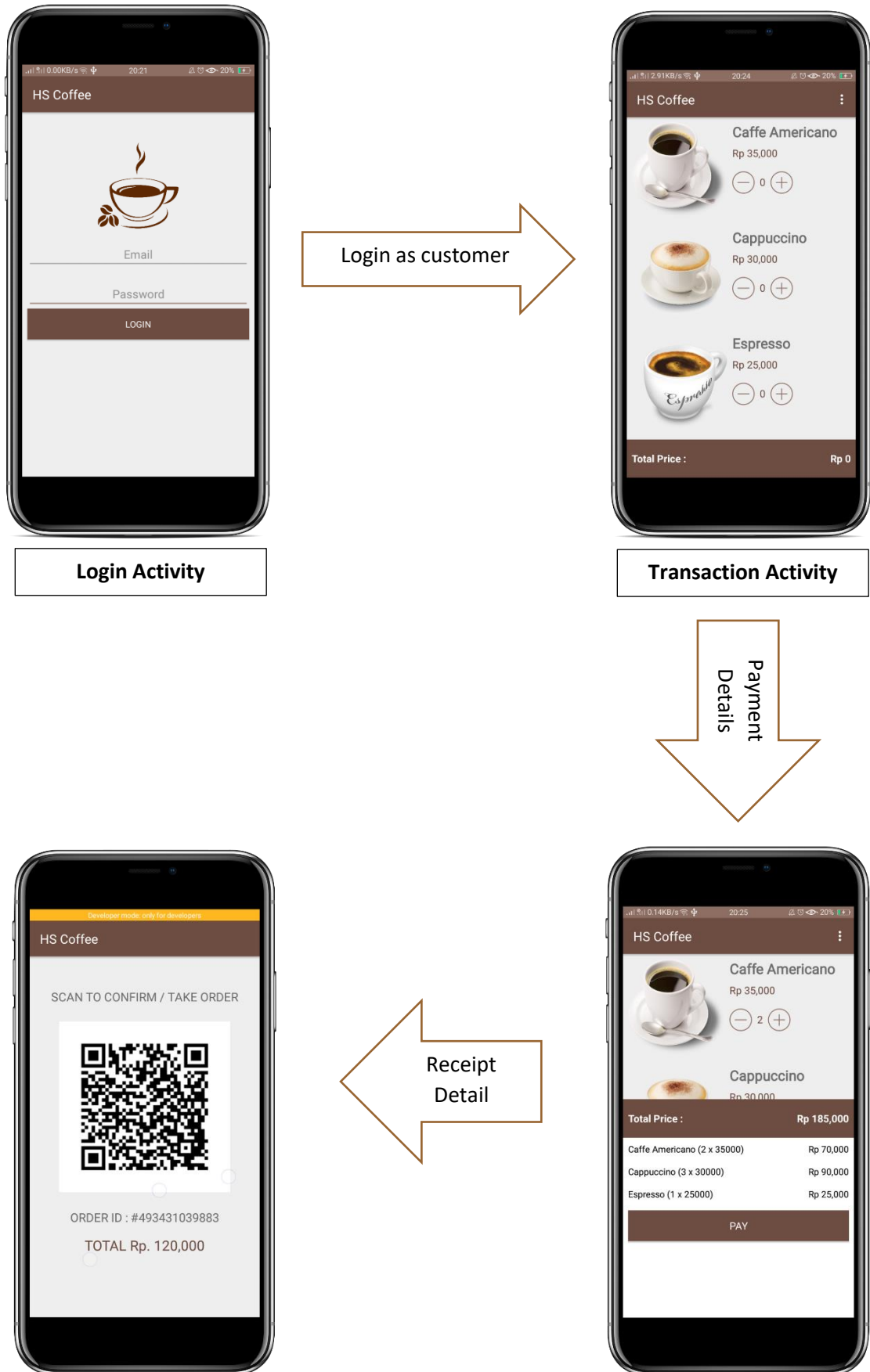
```
@Override
public IModel get(int id) { return coffeeList.stream().filter(t -> t.getId() == id).findFirst().get(); }

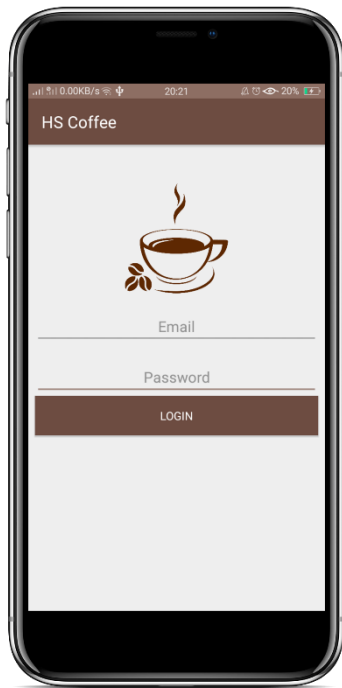
@Override
public List<? extends IModel> getAll() { return coffeeList; }
```

- “Person” class as parent class for customer and seller class.

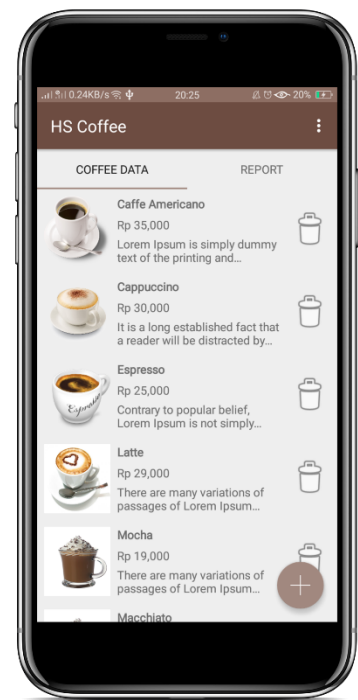
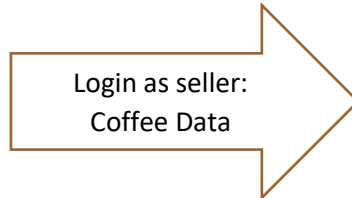


III. Application Flow

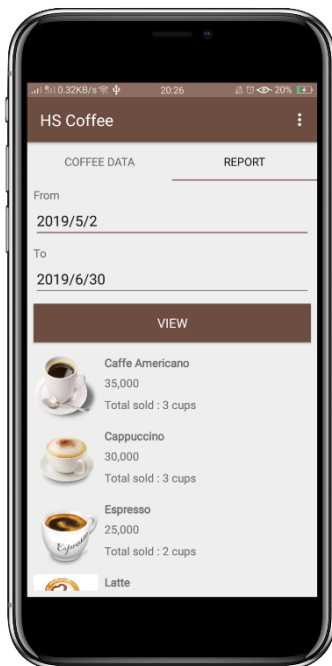
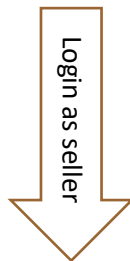




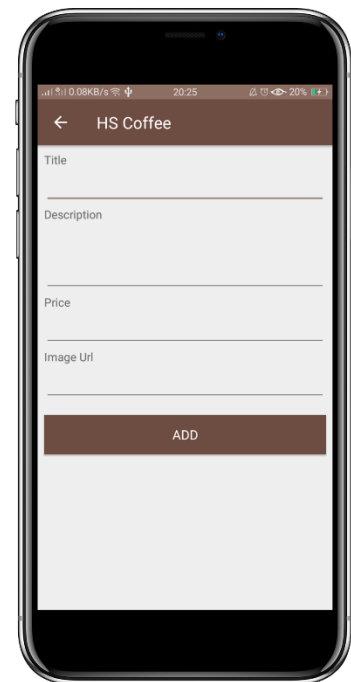
Login Activity



Manage Coffee Activity



Transaction Report

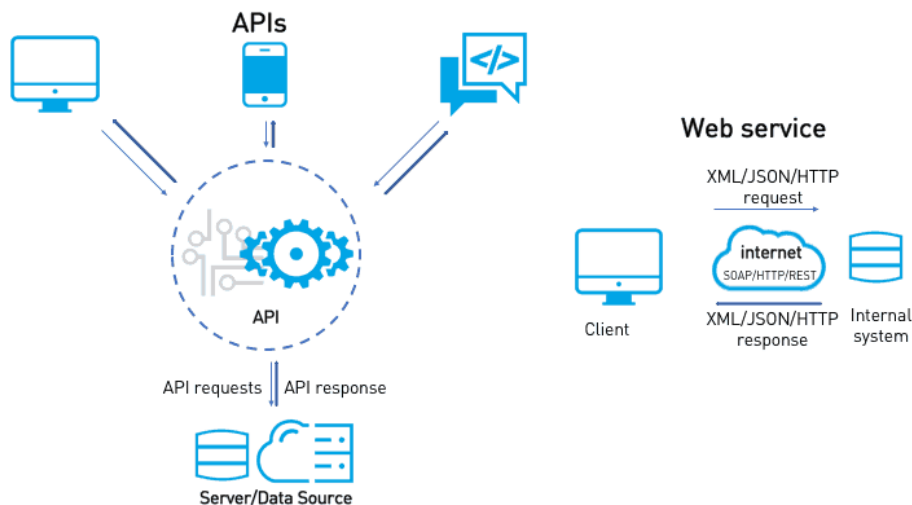


Add New Coffee Activity

IV. Lessons that Have Been Learned

In this project I have learned so much things. I got new experience in creating web service using Java language and the framework called Spring Boot, introduced with several design pattern more well such as MVP, singleton, etc. I learned how to design classes in an efficient and correct way, I can develop and improve my understanding in OOP. In this project I also got opportunity to improve and learn deeper more about developing android apps and integrate it with Java API. In addition, I also used another programming language besides Java to develop the android app called “Kotlin”.

V. Project Technical Description



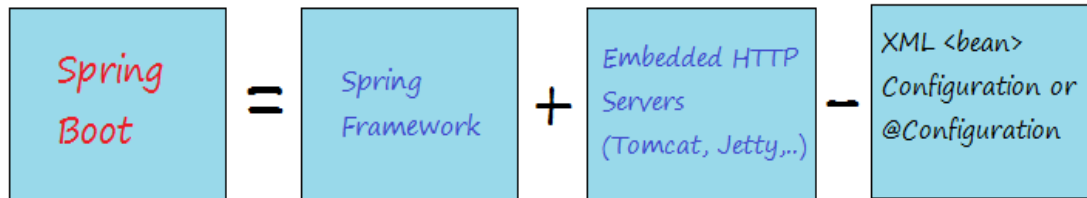
API

API stands for Application Programming Interface. An API is a software intermediary that allows two applications to talk to each other. In other words, an API is the messenger that delivers your request to the provider that you're requesting it from and then delivers the response back to you.

In this project I am using API as the intermediary between my android app and the server. In this case the android app will retrieve, send the data and do all operations by sending a message to the server through this API. This API is built using Java Spring Boot framework.

Spring Boot Framework

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.



Spring Boot is a project built on the top of the Spring framework. It provides a simpler and faster way to set up, configure, and run both simple and web-based applications.

Android



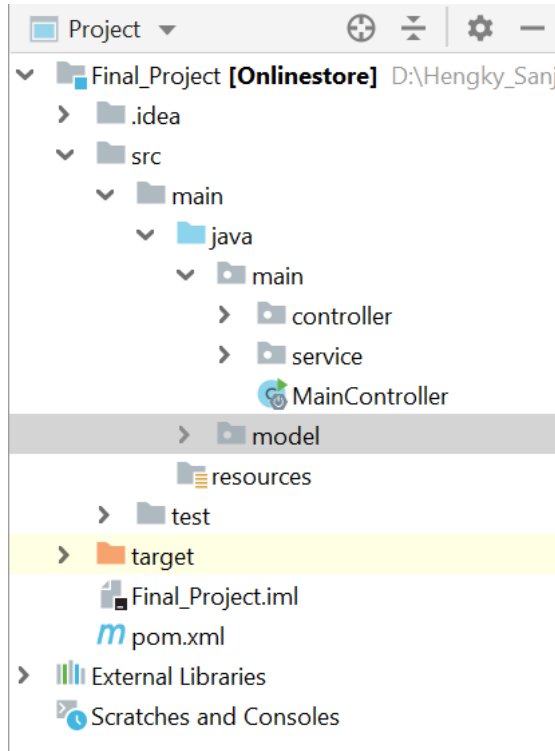
Android is a mobile operating system developed by Google. It is based on a modified version of the Linux kernel and other open source software and is designed primarily for touchscreen mobile devices such as smartphones and tablets.

Android apps can be written using Kotlin, Java, and C++ languages. Kotlin is a cross-platform, statically typed, general-purpose programming language with type

inference. Kotlin is designed to interoperate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise.

In this project I am using Kotlin language to build the android app.

VI. Code Explanation



Project Structure

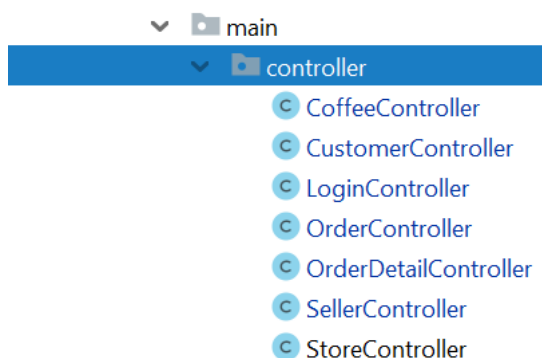
Here I have the controller to accept input and convert it to commands for the model or view.

Service to store the data and do all the corresponding operations.

Model as the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

Spring MVC is the primary web framework built on the Servlet API. It is built on the popular MVC design pattern. MVC (Model-View-Controller) is a software architecture pattern, which separates application into three areas: model, view, and controller. The model represents a Java object carrying data. The view represents the visualization of the data that the model contains. The controller controls the data flow into model object and updates the view when the data changes.

Controller



CoffeeController.java

```
@RestController
public class CoffeeController {
    @Autowired
    private CoffeeService coffeeService;

    // function to return all coffees data in json format
    @RequestMapping("/coffees")
    public List<Coffee> getAllCoffees() {
        return (List<Coffee>) coffeeService.getAll();
    }

    // function to return coffee by specific coffee id inputted from parameter
    @RequestMapping("/coffees/{id}")
    public Coffee getCoffee(@PathVariable int id) { return (Coffee) coffeeService.get(id); }

    // function to add coffee
    @RequestMapping(method = RequestMethod.POST, value = "/coffees")
    public void addCoffee(@RequestBody Coffee p) {
        try {
            coffeeService.add(p);
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
```

@RestController

@RestController is a convenience annotation for creating Restful controllers. It is a specialization of @Component and is autodetected through class path scanning. It adds the @Controller and @ResponseBody annotations. It converts the response to JSON or XML. It does not work with the view technology, so the methods cannot return ModelAndView. It is typically used in combination with annotated handler methods based on the @RequestMapping annotation.

@Autowired

@Autowired Annotations. This annotation allows Spring to resolve and inject collaborating beans into your bean.

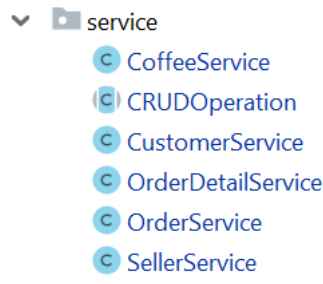
@RequestMapping

This annotation maps HTTP requests to handler methods of MVC and REST controllers. The HTTP method parameter has no default – so if we don't specify a value, it's going to map to any HTTP request.

- GET
The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
- HEAD
Same as GET but transfers the status line and header section only.
- POST
A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
- PUT
Replaces all current representations of the target resource with the uploaded content.

- **DELETE**
Removes all current representations of the target resource given by a URI.
- **CONNECT**
Establishes a tunnel to the server identified by a given URI.
- **OPTIONS**
Describes the communication options for the target resource.
- **TRACE**
Performs a message loop-back test along the path to the target resource.

Service



CRUDOperation.java

```
// as a blueprint or outline to all service classes
public abstract class CRUDOperation {

    public void add(IModel m) { System.out.println("Data added successfully"); }

    public void delete(int id) {
        System.out.println("Data deleted successfully");
    }

    public void update(int id, IModel m) {
        System.out.println("Data updated successfully");
    }

    public abstract IModel get(int id);

    public abstract List<? extends IModel> getAll();
}
```

“*CRUDOperation*” is an abstract class with several functions (add, delete, update) with default implementation and two abstract functions without any implementation. It is used to be a blueprint to all services class.

CoffeeService.java

```
@Service
public class CoffeeService extends CRUDOperation {

    // initialize coffee data
    private List<Coffee> coffeeList = new ArrayList<> (Arrays.asList(
        new Coffee( id: 1, title: "Caffe Americano", description: "Lorem Ipsum is simply dummy text of the printing and typographic industry.", ),
        new Coffee( id: 2, title: "Cappuccino", description: "It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.", ),
        new Coffee( id: 3, title: "Espresso", description: "Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old.", ),
        new Coffee( id: 4, title: "Latte", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", ),
        new Coffee( id: 5, title: "Mocha", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", ),
        new Coffee( id: 6, title: "Macchiato", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", ),
        new Coffee( id: 7, title: "Signature Chocolate", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", ),
        new Coffee( id: 8, title: "Tea", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", ),
        new Coffee( id: 9, title: "Green Tea", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", ),
        new Coffee( id: 10, title: "White Chocolate Mocha", description: "There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable.", )
    ));

    // function to add coffee
    @Override
    public void add(IModel m) {
        coffeeList.add((Coffee) m);
        super.add(m);
    }

    // function to delete coffee
    @Override
    public void delete(int id) {
        coffeeList.removeIf(t -> t.getId() == id);
        super.delete(id);
    }
}
```

@Service

We mark beans with `@Service` to indicate that it's holding the business logic. So there's no any other specialty except using it in the service layer.

```
// function to update coffee
@Override
public void update(int id, IModel m) {
    for (int i = 0; i < coffeeList.size(); i++) {
        if (coffeeList.get(i).getId() == id) {
            coffeeList.set(i, (Coffee) m);
            super.update(id, m);
            break;
        }
    }
}

// function to get coffee by given coffee id
@Override
public IModel get(int id) { return coffeeList.stream().filter(t -> t.getId() == id).findFirst().get(); }

// function to get all coffee data
@Override
public List<? extends IModel> getAll() { return coffeeList; }
}
```

CoffeeService

This class is used to manage all operations related to the coffee data such as add, delete, edit and retrieve the data. It extends the “*CRUDOperation*” abstract class to use the overridden function from the parent and add its own implementation to corresponding function.

Model

A Model , which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.

```
package model;

public class Coffee implements IModel {
    public int id;
    public String title;
    public String description;
    public String url;
    public int price;

    public Coffee() {
    }

    public Coffee(int id, String title, String description, String url, int price) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.url = url;
        this.price = price;
    }

    public int getId() { return id; }

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

    public String getTitle() { return title; }

    public void setTitle(String title) { this.title = title; }

    public String getDescription() { return description; }

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

    public String getUrl() { return url; }

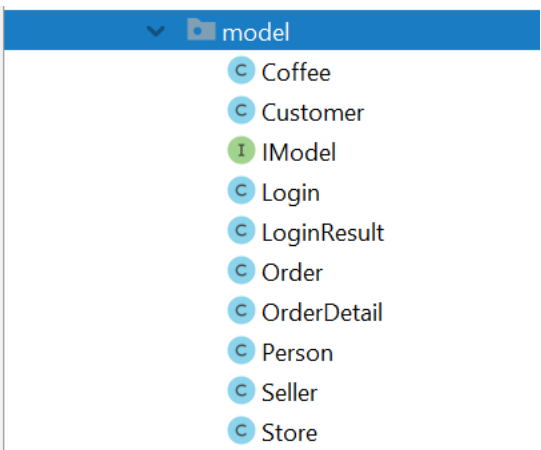
    public void setUrl(String url) { this.url = url; }

    public int getPrice() { return price; }

    public void setPrice(int price) { this.price = price; }
}
```

Coffee.java

This class is Coffee Model class with several attributes, default constructor, constructor with given several parameters and the setters getters. This class implements the “IModel” interface so that other classes know the model classes as one unit “IModel”.



VII. [Project Link](#)

<https://github.com/hengkysanjaya123/FinalProjectProgrammingLanguages>

VIII. [References](#)

https://www.tutorialspoint.com/spring_boot/index.htm

<https://stackoverflow.com/>

<https://kotlinlang.org/docs/reference/>