CO2103 Home Work

This coursework contributes 60% towards your CO2103 semester two marks. The coursework is in two parts, i.e., part 1 covers APIs, and part 2 covers Android. All questions are compulsory and must be answered.

Plagiarism and Collusion

Plagiarism and/or collusion will result in penalties that might go beyond this assignment: https://www2.le.ac.uk/offices/sas2/assessments/plagiarism/penalties

Late Submission and Mitigating Circumstance

Late submission penalties will be applied. Read the sections on late coursework submission in your student handbook or here: https://www2.le.ac.uk/offices/sas2/assessments/late-submission.

Accepted mitigating circumstances are the only way to waive late submission penalties: https://www2.le.ac.uk/offices/sas2/regulations/mitigating-circumstances.

Submission Instructions

Submit your solution on Blackboard by Thursday the 2nd of May, 17:00 UK time. Submissions will not be accepted by any other means (e.g., via email).

- You **must submit a single zip file.** The file name must be **your usernames (i.e., ka388_ab123)**. This zip file must contain two different files (part1.zip and part2.zip), adhering to the rules stated at the beginning of each of the assignment parts. Upload your zip file (your usernames.zip) to Blackboard Assessment and Feedback > **Semester 2 Assignment Submission**.
- You may try it as many times as you like before the deadline. Only the last submission attempt will be marked.
- **Code must compile and execute without modifications**. Your project must compile and execute directly. If not, you will lose 20% of the marks assigned to this part of the coursework.
- **Naming requirements set in each task must be obeyed** (e.g., for files, Java classes, fields, packages, strings, fields etc.). If you do not follow instructions precisely, you will lose 20% of the marks assigned to this part.
- You must write your solution. If you reuse any artifact from any of the worksheets or labs, your work is treated as plagiarism and receives 0 marks.

Pair Programming

You are encouraged to work in pairs for this assignment. Use this ***form*** to create a pair. When generating your tasks, you may use either of your usernames. However, the student who is generating the tasks must also be submitting the solution on Blackboard.

Part 1

Submission Instructions

The file name must be ***part1s2.zip***.

- Package your final file as a zip folder. Ensure you name it **part1s2.zip**.
- Upload your exported zip file to **your usernames** folder that you have created.
- Finally, test/rerun your submission by starting a new IntelliJ workspace, importing your project from the file you added to your usernames folder, and checking that everything works.
- If your file is named differently or the project is not exported following these steps, **you will lose 20 marks**.

Tasks

Your expertise has been sought by an online bookstore to develop a system that efficiently manages their book inventory, authors, and customer orders. The data they are interested in tracking is as follows:

- Authors
    - Long id: unique identifier for an author
    - String name: the author's full name, e.g., “Jane Doe”
    - Int birthyear: the birth year of the author
    - String nationality: the nationality of the author
- Books
    - String ISBN: the International Standard Book Number, a unique identifier for a book
    - String title: the title of the book, e.g., "The Art of Programming"
    - int publicationYear: the year the book was published
    - double price: the price of the book
- Orders
    - Long id: unique identifier for an order
    - Timestamp datetime: the date and time when the order was placed
    - String customerName: the name of the customer placing the order

The relationships between these resources are as follows:

- An author can write multiple books.
- A book can have one or multiple authors.
- An order can contain one or multiple books.
- A book can be part of multiple orders.
- Deleting an author should delete all the books they wrote, except the books that have multiple authors.

The API you design should include the following endpoints:

- A rest controller named edu.leicester.co2103s2.controller.AuthorRestController with request mapping /authors
    - List all authors (i.e., GET /authors) (endpoint #1)
    - Create (POST), retrieve (GET), update (only PUT), and delete (DELETE) a specific author (endpoints #2 - 5)

    - List all books written by a specific author: /authors/{id}/books (endpoint #6)
- A rest controller named edu.leicester.co2103s2.controller.BookRestController with request mapping /books
    - List all books (endpoint #7)
    - Create (POST), retrieve (GET), update (PUT), and delete (DELETE) a specific book (endpoints #8-11)
    - List all authors of a book: /books/{ISBN}/authors (endpoint #12)
    - List all orders containing a specific book (endpoint #13)
- A rest controller named edu.leicester.co2103s2.controller.OrderRestController with request mapping /orders
    - List all orders (endpoint #14)
    - Create, retrieve, and update a specific order (POST, GET, PUT /orders/{id}) (endpoint #15 - 17)
    - List all books in an order: /orders/{id}/books {endpoint #18}
    - Add a book to an existing order (POST /orders/{id}/books) {endpoint #19}
    - Remove a book from an existing order (DELETE /orders/{id}/books/{ISBN}) {endpoint #20}

Task 1. Design and Documentation [40 marks, 2 marks for each endpoint]

Use Swagger Editor to design and document a REST API that provides all the actions needed in the scenario described above. For each endpoint, you must provide two responses: success (2xx) and error (4xx).

Make sure you are using Open API 3 (OAS3). This will be obvious if the first line in your file is openapi: 3.0.1. You have to click on Edit > Convert to OpenAPI 3 > Convert if it is not.

The Swagger Editor at http://editor.swagger.io/ is free to use. You can clear the editor and start from scratch (File > Clear Editor) or take inspiration from the Petstore API that loads up when you first visit the website. To work effectively and avoid losing your work, you will have to save your work to your local computer frequently with File > Save as YAML and load it to continue working on it with File > Import file.

For inspiration on how to describe the content of each of your resources (the content of your resources / JSON objects), you can read this article:
https://swagger.io/docs/specification/describing-request-body/

Submission

You must provide an Open API 3 spec in YAML format for this task. The name of the file must be part1.yaml. To produce this file on Swagger Editor, you will click on File > Save as YAML. Do not forget to rename your downloaded JSON spec to part1.yaml. Finally, place the file in the src/main/resources/ directory of the Spring project you will create in Task 2. The project template will contain a file with the following content:

```
openapi: 3.0.1


info:
```

title: PART1

Placeholder file, replace it with your Open API 3 spec

Task 2: Implementation [40 marks, 2 marks for each endpoint]

Use the attached *part1s2.zip* archive file as a starting project. It contains the packages you need.

Load the project on IntelliJ and use it to implement the REST API you designed for Task 1. You can follow an approach similar to the one you followed in Lab 6.

Your task is to implement all the endpoints described in the previous section in the appropriate rest controllers: AuthorRestController, BookRestController, and OrderRestController. These files already exist in the project, but you must write the code to handle all the required REST requests.

Once you are done, make sure your project runs and deploys on http://localhost:8080

Task 3. Testing [20 marks, 1 for each tested endpoint]

In this task, you will use Postman to create two test requests for each endpoint: one for a successful response and one for an error response. Create a Postman Collection and ensure all the tests run correctly in sequence.

Submission

Export your Postman Collection as a JSON file by clicking on *Export > Collection v2.1 (recommended)*. This will prompt you to save the file *part1.postman_collection.json*, which you will place in your project from Task 2 *src/main/resources/part1.postman_collection.json*. Your template project contains a placeholder file:

```
{
  "info": "Placeholder file, replace it with your Postman Collection"
}
```

Part 1 Assessment

Exceptional (up to 100)

- All the required endpoints have been correctly documented in the spec
- All the required endpoints have been implemented and run as expected, matching perfectly with the documentation
- The Postman collection contains one successful and one error test case for each endpoint and all tests execute correctly

Outstanding (80s)

- Most endpoints have been correctly documented in the spec
- Most endpoints have been implemented and run as expected, matching perfectly with the documentation
- The Postman collection contains one successful and one error test case for most of the endpoints

Excellent (70s)

- Most endpoints have been documented in the spec, although there are some gaps with respect to the requirements, e.g., the relationships between entities are not implemented
- Most endpoints have been implemented and run as expected, although the matching with the documentation is not perfect, e.g., the documentation includes an endpoint that is not implemented
- The Postman collection contains one successful and one error test case for most of the endpoints, but some of them are problematic and do not execute correctly

Competent (60s)

- At least 2/3 of the endpoints have been documented in the spec, with noticeable gaps with respect to the requirements
- At least 2/3 of the endpoints have been implemented and run as expected, although the matching with the documentation is not perfect
- The Postman collection contains one successful and one error test case for each of the implemented endpoints, but several of them are problematic and do not execute correctly

Satisfactory (50s)

- At least half of the endpoints have been documented in the spec, with noticeable gaps with respect to the requirements
- At least half of the endpoints have been implemented and run as expected, although the matching with the documentation is not perfect
- The Postman collection contains one successful and one error test case for each of the implemented endpoints, but several of them are problematic (e.g., incorrect responses or missing behaviour) and do not execute correctly

Adequate (40s)

- Less than half of the endpoints have been documented in the spec, with noticeable gaps with respect to the requirements, e.g., most relationships between entities have not been implemented
- Less than half of the endpoints have been implemented and run as expected with a noticeable mismatch with the documentation

- The Postman collection contains one successful and one error test case for each of the implemented endpoints, but several of them are problematic; few of them execute without errors

Marginal (to 35)

- Up to 1/3 of the endpoints have been documented in the spec, with noticeable gaps with respect to the requirements
- Up to 1/3 of the endpoints have been implemented and run as expected, with a noticeable mismatch with the documentation
- The Postman collection contains one successful and one error test case for each of the implemented endpoints, but several of them are problematic; few of them execute without errors

Little effort (to 20)

- The Open API 3 documentation does not reflect the requirements correctly
- The implementation is severely flawed, with multiple noticeable errors
- The Postman collection only includes trivial tests without relevance to the requirements

Non-adherence (10s)

- The Open API 3 JSON file is missing or is not a valid spec
- The implementation is missing or does not compile
- The Postman collection YAML file is missing or the file is invalid

Nominal (below 10)

- No meaningful work is observed in the submission

Part 2

Submission

- The file name must be **part2s2.zip**. Files named differently will not be marked.
- You must use Andriod studio **Hedgehog | 2023.1.1** version
- Unreadable files will not be marked (e.g., different file formats like `.tar` or `.tar.gz`). All submitted files must be **.zip** only.
- Code that does not build and execute will **not be marked**.
- Make sure your project executes correctly, i.e., the application is built and installed in your emulator/physical devise. You can execute gradle assemble from the command line, which should execute successfully and generate the apk file app/build/outputs/apk/debug/app-debug.apk.
- Clean up your project and remove unnecessary files by executing Build > Clean Project on Android Studio or ./gradlew clean from the command line.
- Export your Android Studio project as a zip file: File > Manage IDE Settings > Export to Zip File. The file name must be part2s2.zip.
- Move/copy the exported file to the your username folder you have created.
- Finally, rerun/test your project again - unzip it and import it on Android Studio to check that everything works.

Setup

To start working on your solution, download *part2.zip*, unzip it locally, and import it as an Android Studio project.

Tasks 4. Design and Implementation [70 marks]

Your first task is to implement a shopping list.

The data that you need to store for each shopping list is as follows:

- ***listId***: the unique id for a shopping list; *this is an internal field and must not be shown anywhere in the app*.
- ***name***: the name of the shopping list (a string); *each list name must be unique*.
- ***image***: an *optional* image to be associated with the shopping list.

For each product added to the shopping list, the app must allow the following fields:

- ***name***: the name of the product added to the shopping list (a string); *each product name must be unique in a shopping list*.
- ***quantity***: a value indicating the quantity of the product. The text field for this must be of input type *number*.
- ***unit***: the unit for the item, which should be one of "Unit", "Kg", "Litre" (add more units if you wish), implemented in the UI using a *Spi*nner view.

You will have to create the necessary Java classes to implement this. All your Java classes should be created in package *uk.ac.le.co2103.part2*.

Features

The features that your app needs to provide are the following:

1. List shopping lists on the main activity named ***MainActivity***. When the app starts, the existing shopping lists should be displayed on the main activity in a ***RecyclerView***. Each recycler view item should include the shopping list image, the shopping list name. This main activity would be initially empty when the app is launched, and no shopping lists have been created yet. [15 marks]

2. Create a shopping list using a second activity named ***CreateListActivity***. There should be a ***FloatingActionButton*** view with id *fab* on the main activity that leads the user to a second activity named ***CreateListActivity***, which contains a simple form to input the details for a shopping list: a name and an optional image (to be loaded from the phone's gallery). The name is essential for creating a shopping list; the image is optional. The form contains a button with the text ***"Create"*** to create the shopping list and navigate back to the main activity. [15 marks]

3. View shopping list. When the user taps on a shopping list in the main activity, the app should navigate to a third activity named ***ShoppingListActivity***. This activity should declare MainActivity as its parent. The activity will contain a ***RecyclerView***, and each item in the recycler view will contain the details of a product in the shopping list: name, quantity, and unit. Each item in your shopping list should have a toast message showing a brief description of the item. [15 marks]

4. Delete shopping list. When the user performs a long-click (long-press) on a specific shopping list, an option is given to "Delete" the shopping list; tapping on the option should delete the shopping list with cascade-deletion of all the products it contains. [15 marks]

5. Add product to shopping list using a fourth activity named ***AddProductActivity***. In ***ShoppingListActivity*** (the activity that displays a shopping list), add a ***FloatingActionButton*** with id ***fabAddProduct,*** which will navigate to ***AddProductActivity***. In ***AddProductActivity***, you will have two ***EditText*** views (one with id ***editTextName*** and input type *text* for the name of the product, and another one with id ***editTextQuantity*** and input type *number* for the quantity) and one *Spinner* view with id ***spinner*** that will allow you to choose between "Unit", "Kg", and "Liter". A button with text "Add" will save the product in the shopping list and navigate back to the product's ***ShoppingListActivity***, where the product should have been added, unless a product with the same name already exists in the list, in which case a *Toast* must be displayed with the text "Product already exists". [15 marks]

Edit or delete a product; tapping on a product in a shopping list should display a *dialog* with the options to "Edit" or "Delete" the selected product.

6. Selecting the "Edit" option navigates to the ***UpdateProductActivity*** activity, which will look similar to ***AddProductActivity***, except that the details of the product are pre-populated and there are two additional buttons with text "-" and "+" that allow to decrement or increment the quantity of the product; when either of "-" or "+" is pressed, the text view containing the quantity should be updated accordingly. A button with text "Save" saves the product and returns to ***ShoppingListActivity*** where the updated information should be displayed in the recycler view. [15 marks]

7. Selecting the "Delete" directly performs the deletion of the product from the shopping list and remains in the ***`ShoppingListActivity`***. [10 marks]

Part 2 Assessment

Exceptional (up to 100)

- All the required features have been implemented correctly.

Outstanding (80s)

- All the required features have been implemented, although with some issues.

Excellent (70s)

- Most of the required features have been implemented, although with several issues.

Competent (60s)

- At least half of the required features have been implemented correctly, with the rest either missing or showing severe issues.

Satisfactory (50s)

- Nearly half of the required features have been implemented correctly, with the rest either missing or showing severe issues.

Adequate (40s)

- Most of the required features are flawed or do not meet the brief, but the application runs and is overall functional.

Marginal (to 35)

- There is a genuine attempt at implementing the required features, the application runs and is functional, but barely meets the brief.

Little effort (to 20)

- There is an attempt at implementing some of the required features, the application runs and is somewhat functional.

Non-adherence (10s)

- There is a minimal attempt at implementing some of the required features, but no feature is functional.

Nominal (below 10)

- No meaningful work is observed in the submission.