ADVANCED PROGRAMMING PRACTICES

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

SOEN 6441

SHUBHAM SHARMA 40198289 (SECTION W)

SHREYANS SHARMA 40196896 (SECTION W)

URL to Git Repository: https://github.com/iamshreyans/SOEN6441_CocktailProject.git

CONCORDIA UNIVERSITY

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

This full stack project is designed as an web application to fetch data from an Application Programming Interface about DRINKS and store it into the local database in SQLite. SQLite fulfils the need of embedding the database with our web application. We have used JDBC, Java Database Connectivity, to access and query our database management system, SQLite. JDBC eases for the interaction with the database from within our Java program and acts as a bridge from our code to the database. The data about different types of drinks is extracted by a call to the API, stored in our local database and then is displayed on our User-Interface that is created using JSP. JSP or Java Server Pages is used to create dynamically generated web pages based on HTML,XML,SOAP, etc. And our application uses HTML for the frontend development with JSP. The UI or website is hosted on a local server with the use of Apache Tomcat which is an open source web server host implemented by Java Servlet. Basically, it provides us with a HTTP web server environment pure to java and can also be called as a Java web application server. The web application is then used to query the data on the basis of name of the drink and then display it on our application.

1.1 Deliverable

A web application that is fast and scalable and built in a single-page application format which provides the users with the ability to search and display for data extracted from a particular API in JSON format, converted into POJO(Plain Old Java Object), stored into local database, and sent to the web application for further implementation of displaying and querying on the basis of a specified sub-field of the data. The developed project is also tested using unit testing methods by implementing JUnit.

1.2 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

2. Architectural Representation

This document presents the architecture as a series of views; use case view, logical view, process view. There is no separate implementation view described in this document. These are views on an underlying Unified Modeling Language (UML) model using draw.io .

2.1 Use-Case View

A description of the use-case view of the software architecture. The Use Case View is important input to the selection of the set of scenarios and/or use cases that are the focus of an iteration. It describes the set of scenarios and/or use cases that represent some significant, central functionality. It also describes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or that stress or illustrate a specific, delicate point of the architecture.

2.1.1 Architecturally Significant Use Cases

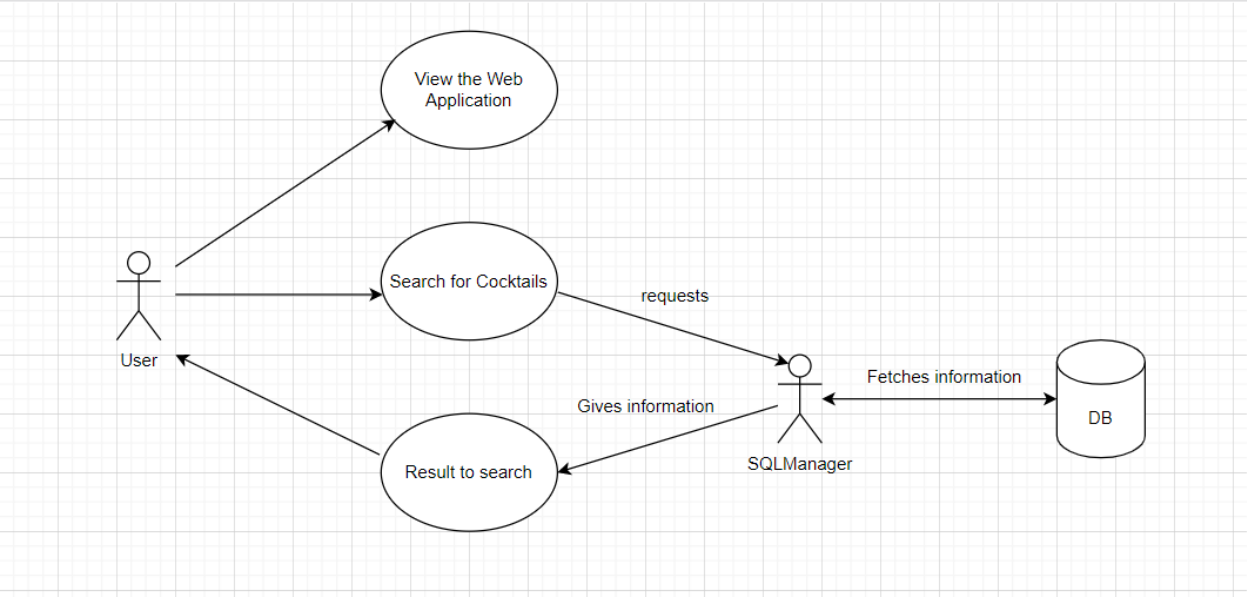

Fig 1. Use Case Diagram

The 2 use cases are as below:

1. The first is when the user can see the web application with the list of cocktails fetched from the API.
2. The second use case is when the user searches for a specific cocktail's data by entering the name of the cocktail as the search field. This query is handled by the SQLManager which further fetches the information for the provided search field from the database and returns it to the UI where the user requested.

2.2 Logical View

The logical view of the course registration system is comprised of the 3 main packages: User Interface, Business Services, and Business Objects. The User Interface Package contains classes for each of the forms that the actors use to communicate with the System. Boundary classes exist to support viewing and searching for specific cocktail's data. The Business Services Package contains control classes for interfacing with the implementation of HTTPClient and HTTPRequest objects interaction. The Business Objects Package includes entity classes for the parsing JSON objects and conversion of JSON objects to POJO(Plain Old Java Objects).

2.2.1 Architecture Overview – Package and Subsystem Layering

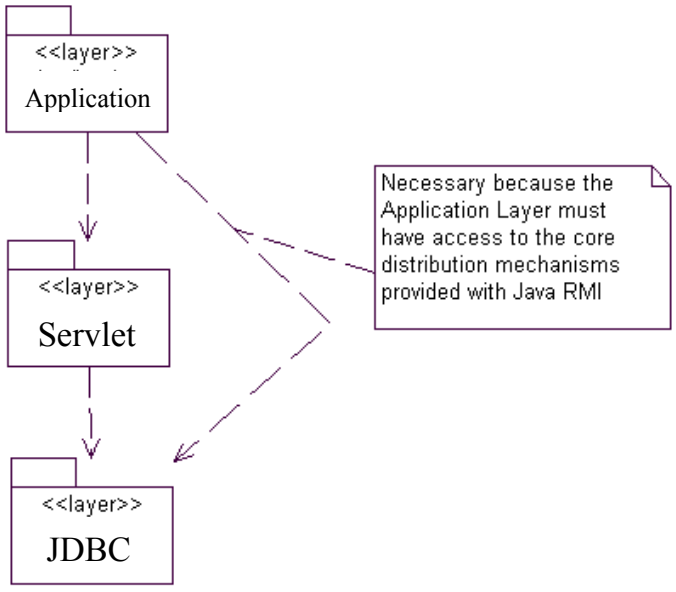

Fig 2. Architecture Overview

2.3 Process View

A description of the process view of the architecture. Describes the tasks (processes and threads) involved in the system's execution, their interactions and configurations. Also describes the allocation of objects and classes to tasks. The Process Model illustrates the course registration classes organised as executable processes.

2.3.1. Processes

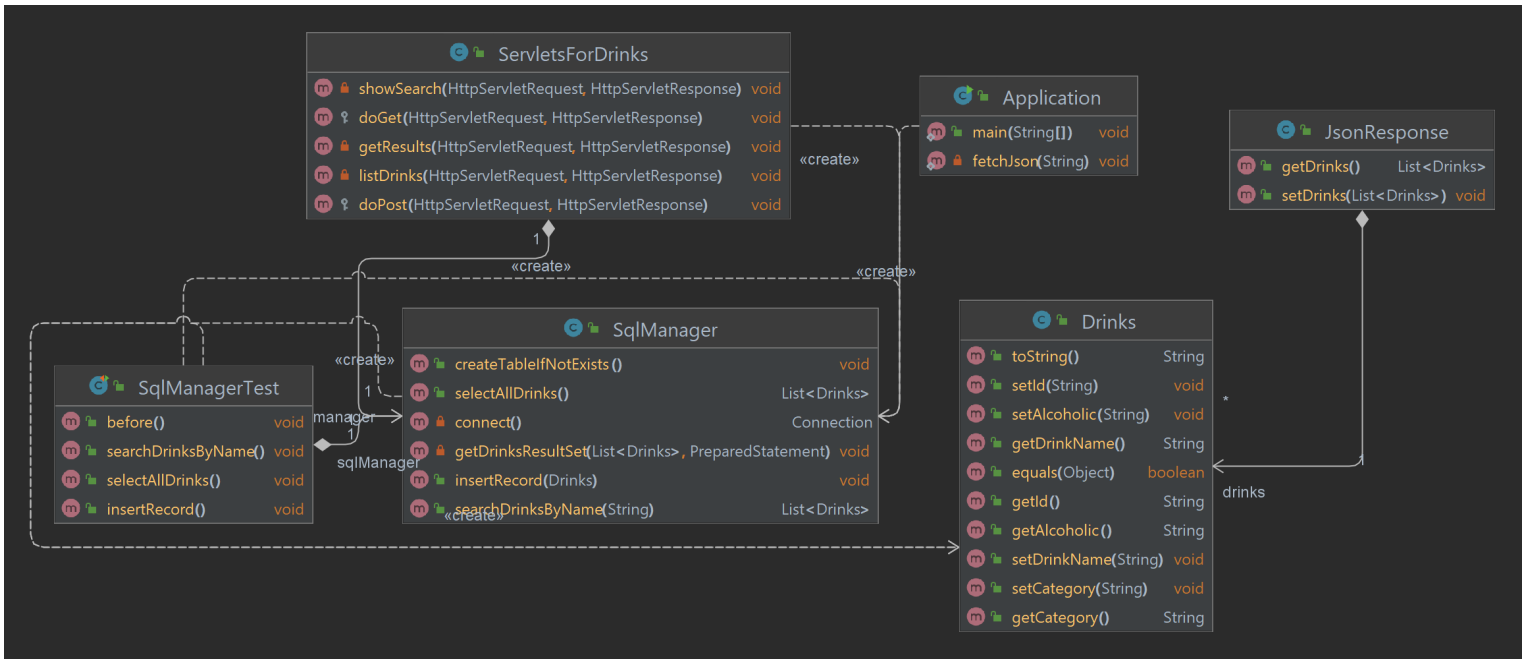

Fig 3. Processes View

The above figure represents the interdependencies between different classes. The methods of each class are responsible for different tasks involved in the system's execution.

2.3.2 Process to Design

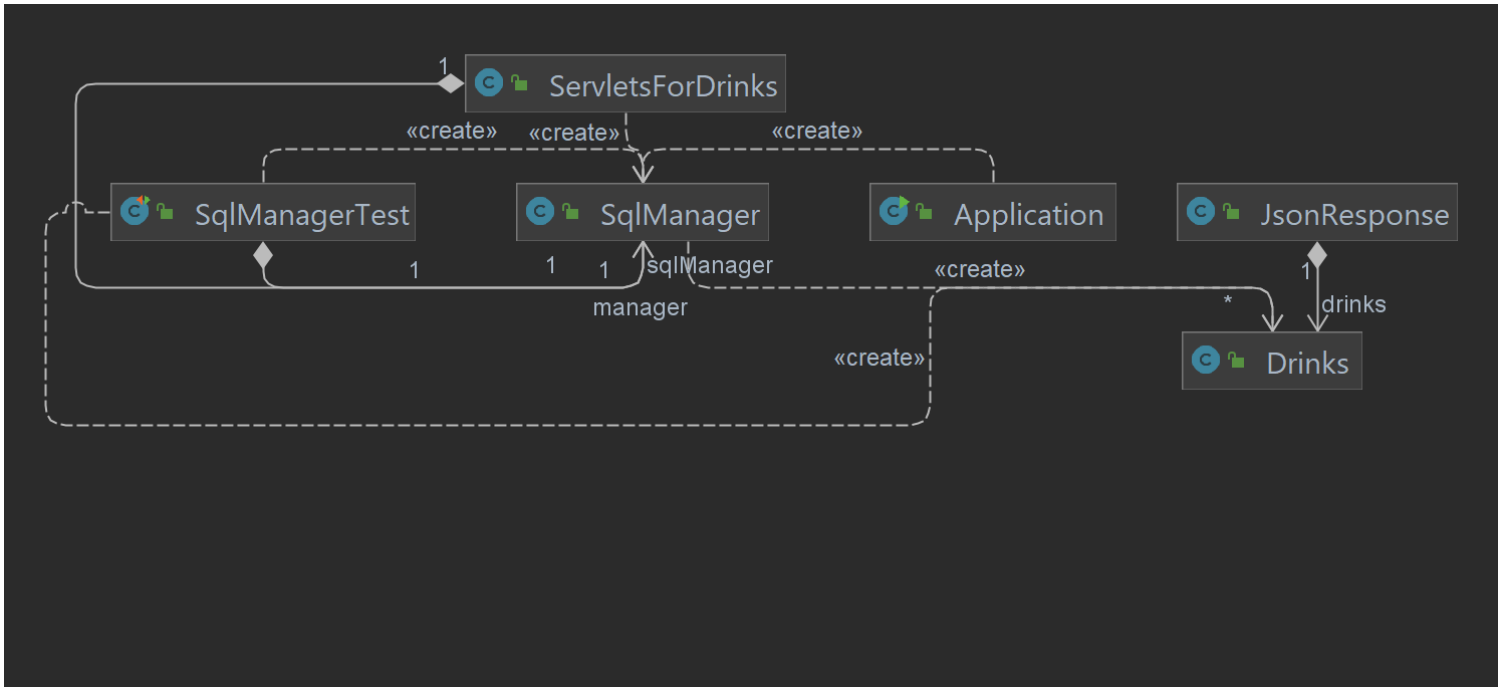

Fig 4. Process to Design View

The above figure represents interlinkage of different classes and how coupled is each class and also the figure of cohesion can be determined by looking at the figure above.

3. TECHNOLOGY STACK

3.1 IntelliJ Integrated Development Environment.

IntelliJ IDEA is an integrated development environment (IDE) written in Java for developing computer software written in Java, Kotlin, Groovy, and other JVM-based languages. It is developed by JetBrains (formerly known as IntelliJ) and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. Both can be used for commercial development. Development of modern applications involves using multiple languages, tools, frameworks, and technologies. IntelliJ IDEA is designed as an IDE for JVM languages but numerous plugins can extend it to provide a polyglot experience.

3.2 Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need to recompile. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub, particularly for client–server web applications, with a reported 9 million developers.

3.3 JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity. It is part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and update data in a database, and is oriented toward relational databases.

3.4 SQLite

SQLite is a database engine written in the C programming language. It is not a standalone app; rather, it is a library that software developers embed in their apps. As such, it belongs to the family of embedded databases. It is the most widely deployed database engine, as it is used by several of the top web browsers, operating systems, mobile phones, and other embedded systems. Many programming languages have bindings to the SQLite library. It generally follows PostgreSQL syntax, but does not enforce type checking by default. This means that one can, for example, insert a string into a column defined as an integer.

3.5 Apache Tomcat

Apache Tomcat (called "Tomcat" for short) is a free and open-source implementation of the Jakarta Servlet, Jakarta Expression Language, and WebSocket technologies. It provides a "pure Java" HTTP web server environment in which Java code can also run. Thus it's a Java web application server, although not a full JEE application server. Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under the Apache License 2.0 license. Apache Tomcat software powers numerous large-scale, mission-critical web applications across a diverse range of industries and organisations. Some of these users and their stories are listed on the PoweredBy wiki page. Apache Tomcat, Tomcat, Apache, the Apache feather, and the Apache Tomcat project logo are trademarks of the Apache Software Foundation.

3.6 JSP

Jakarta Server Pages (formerly Java Server Pages) is a Java standard technology that developers use to write dynamic, data-driven web pages for Java web applications. JSP is built on top of the Java Servlet (aka Jakarta Servlet) specification and is one of the Java web technologies included for ongoing support and upgrades in Jakarta EE. JSP and servlets typically work together, especially in older Java web applications. From a coding perspective, the most obvious difference between JSP and servlets is that with servlets you write Java code and then embed client-side markup (like HTML) into that code. With JSP, you start with the client-side script or markup, then embed JSP tags to connect your page to the Java back end.

3.7 HTML

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document. HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items. HTML elements are delineated by tags, written using angle brackets."Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web.

3.8 BootStrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.Bootstrap is an HTML, CSS and JS Library that focuses on simplifying the development of informative web pages (as opposed to web apps). The primary purpose of adding it to a web project is to apply Bootstrap's choices of colour, size, font and layout to that project. As such, the primary factor is whether the developers in charge find those choices to their liking. Once added to a project, Bootstrap provides basic style definitions for all HTML elements. The result is a uniform appearance for prose, tables and form elements across web browsers. In addition, developers can take advantage of CSS classes defined in Bootstrap to further customise the appearance of their contents. For example, Bootstrap has provisioned for light- and dark-coloured tables, page headings, more prominent pull quotes, and text with a highlight.Bootstrap is developed mobile first, a strategy in which we optimise code for mobile devices first and then scale up components as necessary using CSS media queries.

3.9 MAVEN

Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project. Maven addresses two aspects of building software: how software is built and its dependencies. Unlike earlier tools like Apache Ant, it uses conventions for the build procedure. Only exceptions need to be specified. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache. This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

3.10 JACKSON

In computing, Jackson is a high-performance JSON processor for Java. Its developers extol the combination of fast, correct, lightweight, and ergonomic attributes of the library. Jackson provides multiple approaches to working with JSON, including using binding annotations on POJO classes for simple use cases. In software engineering, a plain old Java object (POJO) is an ordinary Java object, not bound by any special restriction. The term was coined by Martin Fowler, Rebecca Parsons and Josh MacKenzie in September 2000. The term "POJO" initially denoted a Java object which does not follow any of the major Java object models, conventions, or frameworks; nowadays "POJO" may be used as an acronym for plain old JavaScript object as well, in which case the term denotes a JavaScript object of similar pedigree. So, Jackson is a very popular and efficient java based library to serialise or map java objects to JSON and vice versa.

3.11 JUnit

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. JUnit is linked as a JAR at compile-time.

4. CODING STANDARDS

4.1 Standard Libraries

Java is known for its rich set of libraries. However, it is not that every library is perfectly designed but for the most part of it they are optimal. Java releases new libraries every now and then and improves the existing ones actively. Therefore, one should always use classes, methods, interfaces, enums, and annotations from the library as much as possible. This can reduce production time considerably. Moreover, the features included are well tested. It is better to use a them from the library whenever required rather than reinvent the wheel. Because the Java platform evolves frequently, it is extremely important that we keep an eye on the features which we include from the third-party libraries or frameworks are already present in the Java Standard Library. The rule of thumb is to first exhaust the in-house resource before culling support from external sources.

4.2 Naming Conventions

Java naming conventions are set of rules to make Java code look uniform across Java projects and the library. They are not strict rules, but a guideline to adhere to as a good programming practice. It is, therefore, not a good idea to violate the sanctity of the code uniformity either due to haste or rebellion. The rules are pretty simple. Package names are types in lowercase: javax.sql, org.junit, java.lang. Class, enum, interface, and annotation names are typed in uppercase: Thread, Runnable, @Override. Methods and field names are typed in a camel case: deleteCharAt, add, isNull. Constants are types in uppercase separated by an underscore: SIZE, MIN_VALUE. Local variables are typed in camel case: employeeName, birthDate, email. These conventions have become so much apart of Java that almost every programmer follows them religiously. As a result, any change in this looks outlandish and wrong. An obvious benefit of following these conventions is that the style of code aligns with the library or framework code. It also helps other programmers to quickly pick up the code when they have to, therefore leveraging overall readability of the code.

4.3 Class Fields

In Java, methods either belong to a class or to an interface. Therefore, there is a chance that local variables may be given the same name as a class member variable unintentionally by the programmer. The Java compiler, however, is able to pick the right one from the scope. Also, modern IDEs are sophisticated enough to identify the conflict. In any case, programmers themselves should be responsible enough to avoid such conflicts because the result can be quite disastrous.

4.4 Treating Method Arguments as Local Variables

In Java, a variable once declared can be reused. Therefore, the non-final local variables declared in the method arguments can also be reused with a different value. However, this is not a good idea because the variable sends as a method argument what is supposed to hold the value it has brought, the original value that the method with work upon. If we change the value, we'll completely lose the original content that is brought forth in the method. Instead, what we must do is copy the value to another variable and do the necessary processing. We, however, can completely restrict and make the argument variable a constant by using the final keyword.

4.5 Variable Scopes and Readability.

In Java, every variable declared has a scope. This means that the visibility and use of the variables must be restricted within the scope only. In case of a local variable, it is visible from the point of its declaration to the end of the method or code block it is declared in. It is a good practice to declare a variable close to the point of its possible use. This not only enhances readability of the code but also makes debugging simpler.

4.6 Immutability

The concept of immutability is very important. We must decide whether the classes we intend to design can be made immutable or not, because immutability guarantees that it could be used almost everywhere without any trouble from concurrent modification. Unfortunately, not all classes can be designed as immutable. But, make sure we must do so whenever we can. This makes life much easier.

4.7 Testing

Test driven practices (TDD) are a symbol of quality code among the Java community. Testing is a part of modern-day Java development, and the Java Standard Library has the Junit framework to assist in that direction. So, a budding Java programmer should not shy away from writing test cases with the code. Try to keep tests simple and short, focusing on one thing at a time. There can be hundreds of test cases in production environment. An obvious benefit of writing test cases is that they provide immediate feedback of the features under development.

4.8 Strings

No other types in Java have be extended as much as String. Java strings are represented in UTF-16 format and are immutable objects. This means that every time we perform an operation like concatenation, that needs modification of the original string, a new string object is created. This intermediate string object, created only to perform the needed operation, is a waste and inefficient. This is because intermediary object creation is extraneous; it involves garbage collection, although we can avoid it. There are two companion string classes, called StringBuffer and StringBuilder, which can aptly facilitate the kind of string manipulation we may need. These two classes are built for that. The only difference between StringBuffer and StringBuilder is that the former is thread-safe.

4.9 Boxing and Unboxing

In Java, boxing and unboxing are the flipside of the same technique called conversion between primitive types and their corresponding wrapper classes. Wrapper classes are nothing, but are the “class” version of primitive types such as int to Integer, float to Float, double to Double, char to Character, byte to Byte, Boolean to Boolean, and so on. Boxing is converting, say, int to Integer and unboxing is converting back from Integer to int. The compiler often performs the conversion behind the scenes; this is called autoboxing. But, sometimes, this autoboxing may produce an unexpected result and we must be aware of that.

4.10 Handling Constants Efficiently.

Constants help improve memory as they are cached by the JVM. For values that are reused across multiple places, create a constant file that holds static values. Favour database-driven values over dynamic values. Also, use ENUMs to group constants.

4.11 Checking for proper clean Up.

It is common during development to use procedures that help with your coding and debugging (hard coded variables, for example). It is good practice to remove these items and others such as:

- Console print statements
- Unnecessary comments
- Use @deprecated on method/variable names that aren't meant for future use.

4.12 Ensuring the code follows appropriate error handling procedures.

The NullPointerException is one of the most common and frustrating exceptions. However, they can be avoided by performing a null check on a variable before operating on it. The best practice is to use checked exceptions for recoverable operations and use runtime exceptions for programming errors. Another area to evaluate during a Java code review is to ensure all exceptions are wrapped in custom exceptions. In this way, the stack trace is preserved, making it easier to debug when things go wrong. Also, it should declare specific checked exceptions that the method throws rather than generic ones. Doing so doesn't give you the option to handle and debug the issue appropriately.

4.13 Making sure variables don't cause memory leaks.

Creating a bunch of unnecessary variables can overwhelm the heap and lead to memory leaks and cause performance problems. This occurs when an object is present in the heap but is no longer used, and the garbage collection cannot remove it from memory.

4.14 Validating that the code follows separation of concerns.

Ensure there is no duplication. Each class or method should be small and focus on one thing. For example:

EmployeeDao.java - Data access logic
Employee.java - Domain Logic
EmployeerService.java - Business Logic
EmployeeValidator.java - Validating Input Fields

5. APPLICABLE PATTERNS

5.1 Design Pattern -> MVC

The Model View Controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects. MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application. You're still going to need business logic layer, maybe some service layer and data access layer.

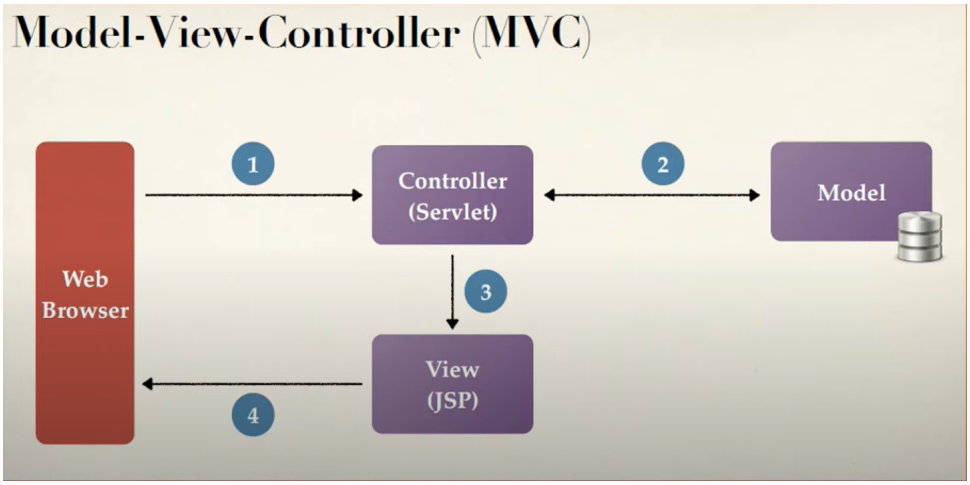

Fig 5. MVC Design Pattern Used in Project

5.2 Object Relational Structural Pattern -> Simple Mapping

In Simple Mapping, we simply map the objects in the application to tables in the database. In simple systems, individual classes can map to separate database tables. Rows of tables can map to instances of the domain classes. Columns of the tables are mapped to attributes of the domain objects. There is a one-to-one mapping of class attribute to table columns.

5.3 Data Source Architectural Pattern -> TDG

Table Data Gateway has a simple interface, usually consisting of several find methods to get data from the database and update, insert, and delete methods. Each method maps the input parameters into a SQL call and executes the SQL against a database connection.The Table Data Gateway is usually stateless, as its role is to push data back and forth.

6. REFACTORING STRATEGIES

6.1 Composing Methods -> Extract Methods

Extract Method refactoring is a technique to separate a method by extracting some code from an existing method as a new method [1]. The purpose of this refactoring is to improve the comprehension of the program by splitting a method that is too long, or a method that implements multiple features into each one.

6.2 Moving features between objects -> Extract Class

Extract class refactoring (ECR) is the refactoring technique used to address God Class design flaw. It refers to the activities of partitioning a large class with many responsibilities into smaller classes such that each class has one responsibility.

6.3 Replace Method with Method Objects

A method is too long and you can't separate it due to tangled masses of local variables that are hard to isolate from each other. The first step is to isolate the entire method into a separate class and turn its local variables into fields of the class. Firstly, this allows isolating the problem at the class level. Secondly, it paves the way for splitting a large and unwieldy method into smaller ones that wouldn't fit with the purpose of the original class anyway.

6.4 Assertions in Testing

For a portion of code to work correctly, certain conditions or values must be true. So, we replace these assumptions with specific assertion checks. Main benefit of assertions is, if an assumption isn't true and the code therefore gives the wrong result, it's better to stop execution before this causes fatal consequences and data corruption. This also means that you neglected to write a necessary test when devising ways to perform testing of the program.

7. ENVIRONMENT SETUP

We start with setting up an Intelligent Development Environment for creating our system. For this purpose, we chose IntelliJ Development Environment. After creating the needed classes for all of the requirements in our IDE, we import our necessary classes for the implementation of our logic. After importing the basic java classes, we import Jackson JAVA JSON Library that is a high-performance JSON processor for Java. Jackson provides multiple approaches to working with JSON, including using binding annotations on POJO(Plain Old Java Objects) classes for simple use cases. Jackson is used to convert Json data object into POJO objects so that we can store the downloaded data into our local database in Sqlite. Furthermore, Maven is used for our dependencies of such libraries. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache. this local cache of downloaded artifacts can also be updated with artifacts created by local projects. Maven addresses two aspects of building software: how software is built and its dependencies. These dependencies mainly refers to the libraries or JAR files. The tool helps get the right JAR files for each project as there may be different versions of separate packages.

Eventually after the Backend development, we need a server to host our application on the internet. For this purpose, we set up Apache Tomcat to provide our java web application a HTTP web server environment in which our java code also runs.

Subsequently for the frontend as soon as we start the Apache Tomcat server it hosts the application on localhost on the designated port (we used 8080) and as soon the web application is hosted one can see the results.

8. IMPLEMENTATION

The implementation starts from us finding an open-source Application Programming Interface from a couple of resources on internet. We find and choose an API about Cocktails which shows us data about the same with fields like their “Drink Ids”, “Drink Name”, “Alcoholic/Non-Alcoholic”, “Drink Colour”, “Ingredients” in multiple fields.
So, we extract this JSON data from the API through the JAVA program in the class named “JsonResponse“. To access this data, Java Database connectivity Application programming interface as JDBC eases the interaction with the database from within our Java program and acts as a bridge from our code to the database. Then, the JSON object is converted into POJO, that is, Plain Old Java Object using Java’s Jackson Library. Jackson is a very popular and efficient java based library to serialise or map java objects to JSON object and vice versa. The conversion to POJO feature enables us to apply the rich manipulation, management features to our data through the Java program and further store into the local database. The local Database is created in the database management software named Sqlite. As the database is not very big, we chose to use SQLite as it does all of the required work for a small to medium database with very ease and thus it’s totally reliable.
The required Data that we want to display on the web application is then fetched and displayed on screen through the Web Interface which is hosted on a local host with the help of Apache Tomcat. Apache Tomcat is web application server purely based for applications built in Java.

The FrontEnd aspect of the project JSP along with HTML, and BootStrap is used for the web application User-Interface. We chose JSP, that is, Jakarta Server Pages (formerly JavaServer Pages), to write dynamic, data-driven web page for Java web application. All the structure of the web page document in application interface is written in HTML and BootStrap is used for responsive front-end designing.

The testing aspect of the project meets the requirements with the implementation of JUnit which also a library used to perform unit testing. The most crucial methods of the project which store and fetch data back and through were tested and the results came out to be positive.

Here, we have our full working web application designed to extract data from an API, produce it on our application further allowing parameterised querying on the local database managed in SQLite database management software.

9. CONCLUSION

In Conclusion, we were successfully able to build a web project in Java that mainly does all the needed tasks of extracting data from an API about Cocktails, download it and store into a local database using Sqlite. Then, fetch the data and produce it onto a web application with a User-Interface allowing users to query the data on the basis of “name” of the drinks.

We learnt about full stack development throughout this project, as to how to manage data downloaded from an Application Programming Interface in a local database. This part was tricky as we were trying to store the data in the JSON format directly in Sqlite’s database which was unsuccessful. So we learnt about POJO while completing this process and how to convert the JSON data into POJO made us learn about the Jackson Library and the use of Maven Software Project Management Tool to manage these dependencies of such libraries in our project. Then following the MVC architecture was very intimidating at first, but we got to learn a lot about good software programming practices by that. Also, we learnt about Object-relational mapping (ORM) as a mechanism that makes it possible to address, access and manipulate objects without having to consider how those objects relate to their data sources which in turn made our work easier somehow.

Hosting the application through Apache Tomcat was pretty straightforward. We experienced no to minor difficulty in that but setting it up at the first place was a task.

In the frontend development, one of the main challenges that we faced and learnt from is employing the right framework which was BootStrap in our case. Deploying an optimised design for our application with the help of HTML and JSP can be counted next. In addition to this, it was not just these three that were understood but the very first step of creating a roadmap as to what all should we pick to create this project while comparing the frameworks, languages, libraries, and etc helped us learn the benefits and drawbacks simultaneously of various technologies side by side.

All in all, it was a good exercise to learn and create simultaneously for the industry level knowledge.

10. REFERENCES

1. https://www.wikipedia.org
2. https://stackoverflow.com/
3. How to build a Web Application Using Java- https://www.javatpoint.com/how-to-build-a-web-application-using-java
4. Java Web Application Tutorial for Beginners- https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners
5. Jackson JSON Java Parser API Example Tutorial- https://www.digitalocean.com/community/tutorials/jackson-json-java-parser-api-example-tutorial
6. Maven in 5 Minutes- https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
7. JDBC Tutorial- https://www.tutorialspoint.com/jdbc/index.htm
8. TomCat Tutorial- https://www.devdungeon.com/content/tomcat-tutorial
9. Tutorial of JSP for Beginners - https://www.javatpoint.com/jsp-tutorial
10. JSP Tutorial- https://www.tutorialspoint.com/jsp/index.htm
11. Bootstrap Tutorial- https://www.javatpoint.com/bootstrap-tutorial
12. Big List of Free and Open Public APIs - https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/
13. JSP Servlet JDBC MySQL CRUD Tutorial- https://www.youtube.com/watch?v=RqiuxA_OFOk&ab_channel=JavaGuides