

## **CHAPTER 1**

### **Introduction**

#### **1.1 Project Overview**

This project aimed at developing integration layer component for the open banking and also peripheral services using Eureka naming server, Ribbon and config server that would support the smooth execution and usage of the microservices developed under this project.

#### **1.2 Project Purpose**

Under the Open Banking Standard, banking data will be shared through secure open APIs so that customers, be it individuals or businesses, can more effectively manage their wealth. Open APIs would allow third party developers to create helpful services and tools that customers can utilize. This project aims at developing such services so that bank can share financial information electronically, securely, and only under conditions that customers approve of. Application programming interfaces allow third-parties to access financial information efficiently, which promotes the development of new apps and services. which would help individual save money, borrow easily and pay painlessly.

#### **1.3 Scope**

Open Banking leverages the concept of banks being a banking services provider, with functions of the bank - opening and maintaining an account, making payments, and so on - being decomposed and consumable in a more fine-grained way. The “Open” in Open Banking tends to focus on these services being available via an open API <sup>[1]</sup>. But API doesn’t offer the consumer choice directly - they typically don’t build software - but it allows third parties their choice to act securely on their behalf. Consumers can therefore still hold an account with a bank, but consume a multitude of services from other organizations that their account holding bank may not offer. This project focuses on building such API using concept of microservices. The future of microservices will focus more on the integration layer that ties multiple microservices together.

## **CHAPTER 2**

### **Literature Survey**

Like many industries, the financial sector has come increasingly to rely on technology to underpin the services and products it offers. The financial innovations are transforming the financial industry. One of the primary drivers for this change is ‘digitalization’ – a socio-technical process of applying technology across broader social and institutional domains. Banks have been utilizing digital technologies since the late 1960s when bank accounts became electrometrically held. Further digitization occurred and continued in the 1970s with electronic stock markets (NASDAQ) and the introduction of global communication networks (SWIFT). The latest trend in digital transformation, finalization and initialization goes beyond digitization of accounts, stock markets, and communication. New innovations are fundamentally changing the industry in terms of the character and structure of business models. Today, technological innovations have become deeply embedded and integral to the transformation of the financial services industry and business model design.

### **2.1 Open Banking**

Open banking is part of financialization process in which change is driven through complementarities and cohesion among supportive regulations, market forces and technological change, whereby new practices and arrangements emerge. Existing practice that collectively make up the current status quo of financial services are in a state of evolution as open banking practices drive widespread change and transformation of current practice. We find new entrants who are attacking the incumbents with new technology innovations and business models, while the financial industry incumbents try to fight off the attacks by investing more and more in their existing infrastructure. For financialization, where intermediate markets are not well filled as is typical in early phases of sectoral development, there are opportunities for firms to find new niches to populate, such as pure on-line banks or mobile payments. As new products and services are devised by the newcomers and adopted by consumers and incumbents, they transform the competitive landscape and change established practices. They stimulate competition among the newcomers and change the strategic vision of entrepreneurs to focus more on planning for takeover. They also shift the competitive criteria prevalent among the

incumbents. Technologically enabled financial in Open Banking: Emergent Roles, Risks & Opportunities Twenty-Sixth European Conference on Information Systems (ECIS2018), Portsmouth, UK, 2018 4 innovations, such as open banking, are challenging long established business models. For instance, open banking allows for the cohesion of products and services, which previously were separated, into new and unusual offerings.

### **2.1.1 APIs and Interoperability standards**

Before the term API was coined, the underlying idea of separating interfaces between software systems has been a dominant and important aspect in software development (e.g., service-oriented architecture). APIs describe operational, inputs and outputs of software components. APIs allow programmers to understand how to use a piece of software without knowing the internal algorithms and by following rules stipulating appropriate inputs and outputs. Thus, multiple software components can be connected and amalgamated to create new functionality. One of the first concrete mentions of APIs, though in the name RCP, was from 1998 when a news article detailed how Microsoft was working on a protocol that allows you to do what they call Remote Procedure Calls. This would later turn into the Simple Object Access Protocol (SOAP) which is still used today. Fielding published a thesis describing the concept of representational state transfer (REST), which is the standard today for designing APIs: so-called RESTful APIs. Recently, a new movement towards Open APIs (APIs that are open to the public) have been picking up speed. The first publicly available API is regarded to be the one released by Salesforce.com on February 7th, 2000 which allowed access to the Salesforce.com application through an XML-based API. Later, eBay released an API to partners and selected developers. Open API is the technical realization of Open Banking. Open' does not mean that every third party can access a firm's system at their discretion. There will always be some form of control by the firm, in order to preserve security, privacy and contractual conditions. Most digital market participants have used API technologies to meet their business objectives and ultimately create customer or platform value. They have discovered that using APIs in 'opening up' systems (to the outside world) is essential for driving traffic to software assets, for co creating end customer value in the ecosystem and for sharing the burden and benefits (including the profits) between the parties involved when unlocking new markets. An important means of value co creation through APIs is through

enabling third parties to build applications ‘on top’ of the platform examples include Facebook, Amazon, eBay, PayPal, Twitter, and Google. Developers can reuse existing functionality or use multiple data sources to enrich their own applications. This lowers cost and speeds up time-to-market, but also creates additional dependencies on third party developers. For API providers, this way of value co-creation provides a wider distribution network, creating traffic and minimizing innovation costs, which are carried by third parties. APIs which support modularity and dissemination of financial services have gained much traction recently. This is evidenced by the number of open-APIs created in the financial sector: according to Programmable Web (a directory of APIs), the 10th most populated API category is “financial” and in 17th place we find “payments”. Open-APIs facilitate the establishment and the enhancements of digital platforms that can be easily accessed and connected to and the platform-based business model. Since 2015, Open APIs and Open Banking have increasingly gained attention and have grown from being purely technical topics to being of business relevance for banking practitioners and academics.

### **2.1.2 Regulation and Competition**

Opening services and making them available to other market participants challenges the traditional boundaries of the firm and may underpin new value creation strategies primarily aimed at banking customers but also impacting other stakeholders, including investors and regulators. Indeed, regulators have become increasingly aware of the power of APIs. One example is the UK’s Competition and Market’s Authority (CMA) who have led the “Open Banking Initiative.” Their investigation concluded, “older and larger banks do not have to compete hard enough for customers’ business, and smaller and newer banks find it difficult to grow. This means that many people Open Banking: Emergent Roles, Risks & Opportunities Twenty-Sixth European Conference on Information Systems, Portsmouth, UK, 2018 5 are paying more than they should for their banking activities and are not benefiting from new services.” As a consequence of these issues, the CMA has focused on driving competition between and across new entrants and incumbents through innovations focused on the quality of services and product received by consumers of banking services. By early 2018, the larger retail banks operating within the UK will be obligated to create standards for APIs. These APIs will have both read and write capabilities and will enable transaction data from personal and

business account to be made available to third parties. In addition to allowing third parties to access transactional data sets this will also allow third parties to initiate payments directly without the need for credit or debit cards. A further related benefit is to make financial organizations more transparent regarding their product offerings, as well as their customer satisfaction scores and other service level indicators. Similarly, the European Unions' Payment Services Directive (PSD2) will require banks to offer third party providers and vendors access to their customers' accounts through openly available APIs if their customers so request (European Commission, 2015). The aim is to create further competition by allowing third-parties to deliver new innovative services by leveraging the banks' data and infrastructure. Outside of the EU similar efforts by regulators are being observed in the Asian-Pacific region. To summarize, today's fintech movement and provisions for access-to-accounts are partly being driven by regulators keen to accelerate the competition and digital disruption that is reshaping the financial services industry and also to further increase transparency and reduce information asymmetries.

## CHAPTER 3

### Models and Thread Theory

#### 3.1 Waterfall Model

The Waterfall model is the earliest SDLC approach that was used for software development. The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete.

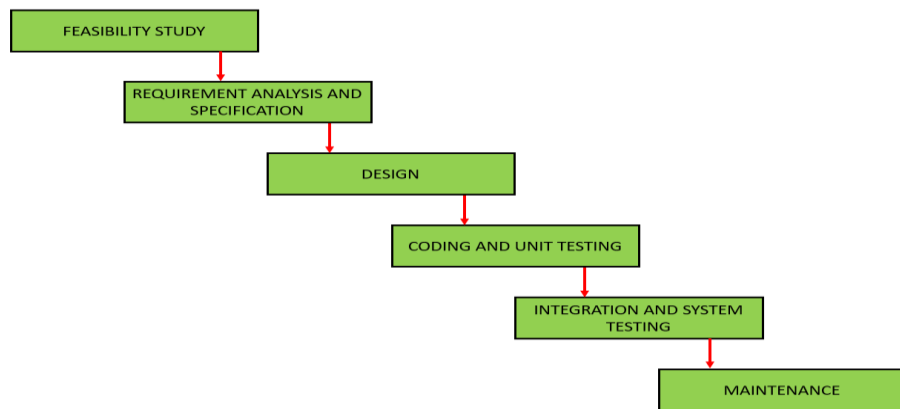


Fig 3.1 The classical waterfall model and it's components

#### 3.2 Agile Development Model

Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). **Agile methodology** is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Both development and testing activities are concurrent unlike the Waterfall model.

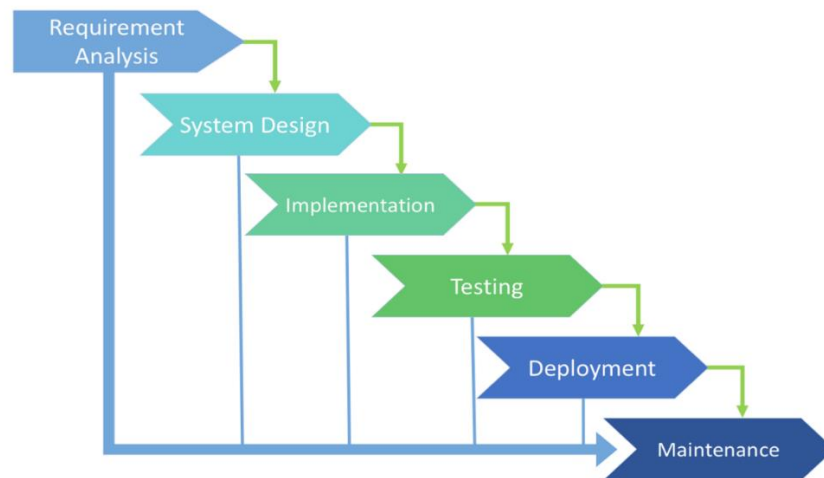


Fig 3.2 Agile Software Development or Iterative Waterfall Model

### 3.3 Comparison between Waterfall model and Agile Development Model

PROJECT TRAIT/FACTOR	AGILE	PLAN-DRIVEN (WATERFALL)	COMMENTS
CUSTOMER AVAILABILITY	Prefers customer available throughout project.	Requires customer involvement only at milestones.	Customer involvement reduces risk in either model.
SCOPE/FEATURES	Welcomes changes, but changes come at the expensive of Cost, Schedule, or other Features. Works well when scope is not known in advance.	Works well when scope is known in advance, or when contract terms limit changes.	Change is a reality so we should prefer adaptability where possible. Contract terms sometimes restrict it.
FEATURE PRIORITIZATION	Prioritization by value ensures the most valuable features are implemented first,	“Do everything we agreed on” approach ensure the customer gets everything they asked for; “all or	Contract terms may not permit partial success and may require “do everything.”

	<p>thus reducing risk of having an unusable product once funding runs out. Funding efficiency is maximized.</p> <p>Decreases risk of complete failure by allowing “partial” success.</p>	<p>nothing” approach increases risk of failure.</p>	
TEAM	<p>Prefers smaller, dedicated teams with a high degree of coordination and synchronization.</p>	<p>Team coordination /synchronization is limited handoff points.</p>	<p>Teams that work together work better, but when contracts are issued to different aspects of the project, high degrees of synchronization may not work.</p>
FUNDING	<p>Works extremely well with time and materials or other non-fixed funding, may increase stress in fixed-price scenarios.</p>	<p>Reduces risk in Firm Fixed Price contracts by getting agreement up-front.</p>	<p>Fixed price is tough when scope is not known in advance, but many government contracts require it.</p>
SUMMARY	<p>Agile is better, where it is feasible.</p>	<p>Plan-Driven may reduce risk in the face of certain constraints in a contract between a vendor and external</p>	<p>Through educating our customers about the strength and weaknesses of each model, we hope to</p>



		customer such as the government.	steer them towards a move Agile approach. This may require changes to how our customers, particularly the government, approach software development projects.
--	--	----------------------------------	---

Table 3.1 ALIGNING PROJECT TRAITS with DEVELOPMENT METHODOLOGIES

Agile and Waterfall are very different software development methodologies and are good in their respective way. However, there are certain major differences highlighted below -

- Waterfall model is ideal for projects which have defined requirements, and no changes are expected. On the other hand, Agile is best suited where there is a higher chance of frequent requirement changes.
- The waterfall is easy to manage, sequential, and rigid method.
- Agile is very flexible and it possible to make changes in any phase.
- In Agile process, requirements can change frequently. However, in a waterfall model, it is defined only once by the business analyst.
- In Agile Description of project, details can be altered anytime during the SDLC process which is not possible in Waterfall method.

Hence, in this project we implement our tasks by following the Agile software development methodology.

### 3.4 Development of UI and API

The **user stories**(jargon for tasks within a project) were decided and each members were assigned respective tasks to perform. Each user story was ranked according to priorities and were divided during the course of two sprints. The UI and APIs were developed in Sprint I.

#### 3.4.1 Sprints in Agile software development

In product development, a sprint is a set period of time during which specific work has to be completed and made ready for review.

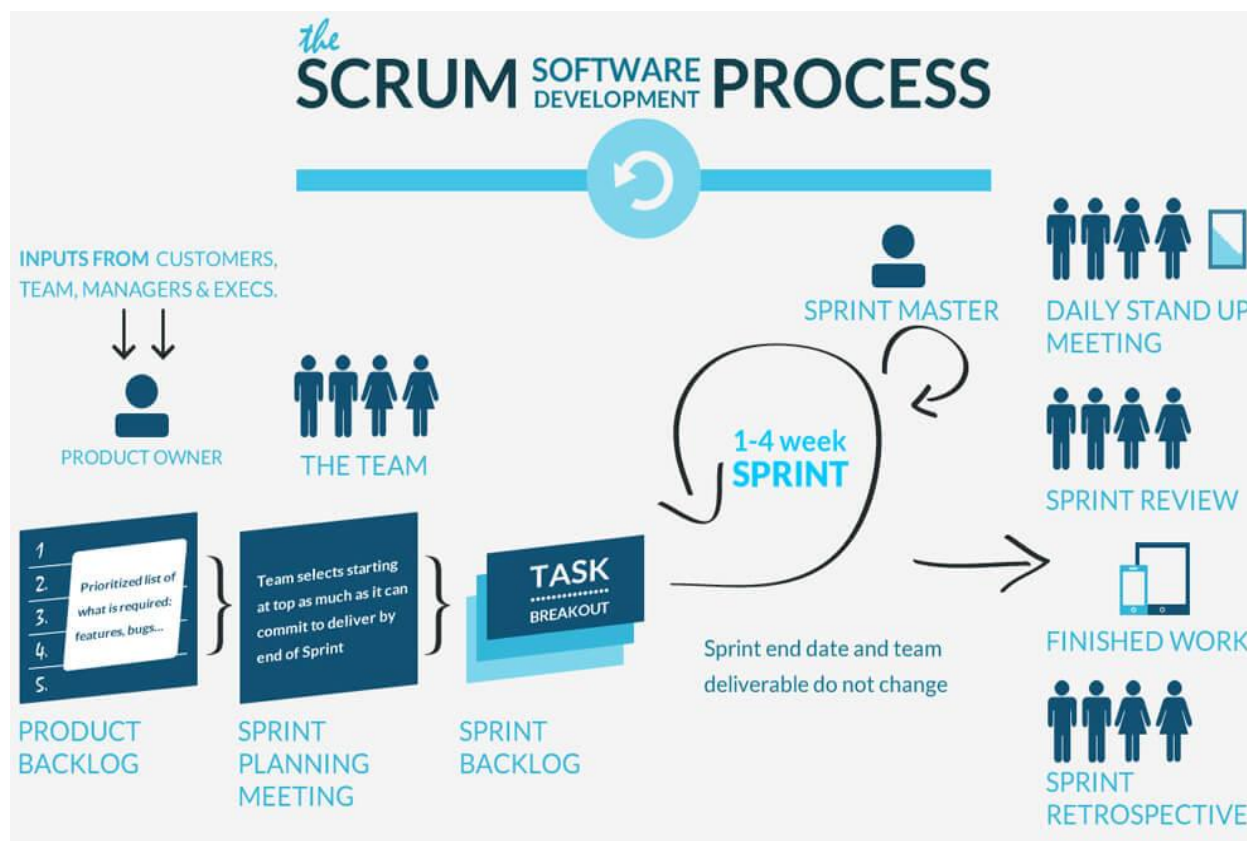
Each sprint begins with a planning meeting. During the meeting, the product owner (the person requesting the work) and the development team agree upon exactly what work will be accomplished during the sprint. The development team has the final say when it comes to determining how much work can realistically be accomplished during the sprint, and the product owner has the final say on what criteria need to be met for the work to be approved and accepted.

The duration of a sprint is determined by the **scrum master**, the team's facilitator. Once the team reaches a consensus for how many days a sprint should last, all future sprints should be the same.

#### 3.4.2 Scrum and Scrum Master in Agile software development

**Scrum** is a methodology that allows a team to self-organize and make changes quickly, in accordance with agile principles. Whereas, a **scrum master** is the facilitator for an agile development team.

The following schematic diagram helps in understanding the flow of Agile development methodology:



**Fig 3.3 Components during Agile or Scrum Development Process**

### 3.4.3 Open APIs for banking

APIs are at the heart of open banking. If executed correctly propose to increase innovation, foster collaboration, extend customer reach and lower costs compared to existing legacy systems. A key concept in the open banking paradigm is to use open source technologies to enable third-party developers to build financial applications on top of the banks' existing infrastructure. This will most likely spark fears of becoming a commodity and giving away the customer interface for many bankers and may seem like the banks will be relegated to the back seat while third-party technology companies are driving the car.

The use of APIs is fundamental to the concept of Open Banking. The need for new products and services to serve increasingly multichannel customers and create more personalized relationships requires the development of Open APIs.

The following is the workflow of Open API framework in open banking:

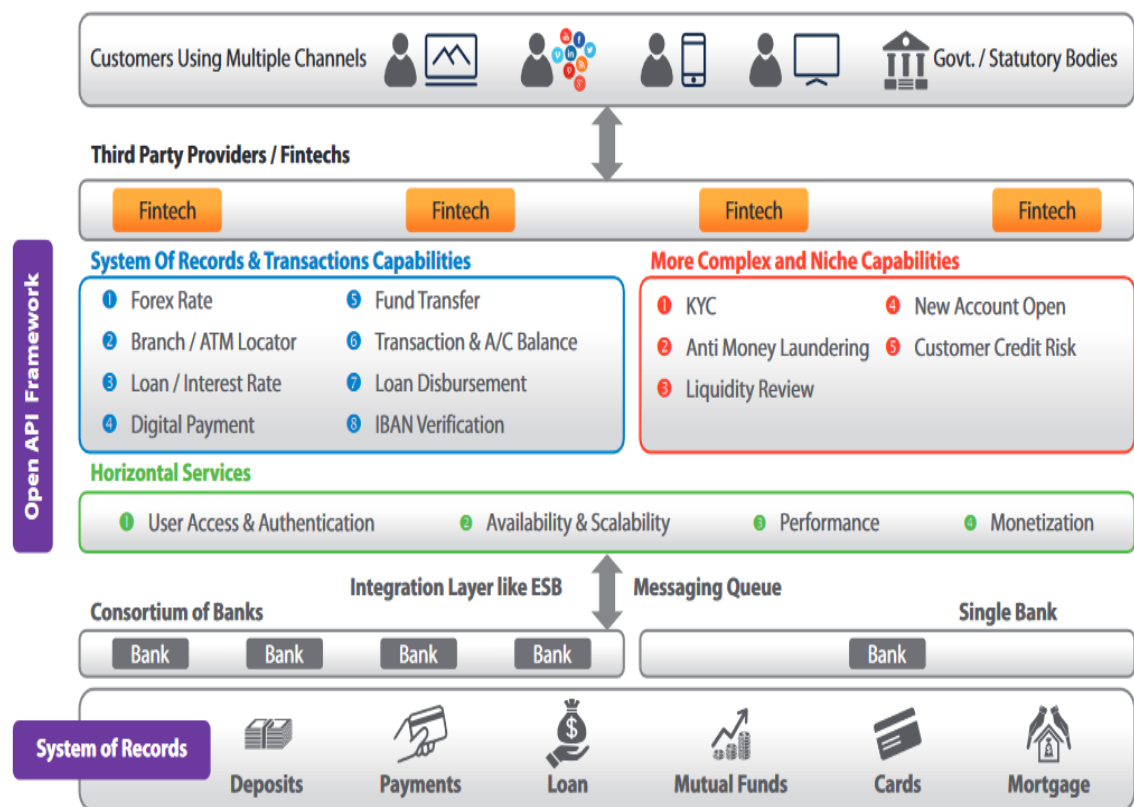


Fig 3.4 Schematic diagram to depict role of Open API framework in banking sector

### 3.5 Thread pool in java

Server Programs such as database and web servers repeatedly execute requests from multiple clients and these are oriented around processing a large number of short tasks. An approach for building a server application would be to create a new thread each time a request arrives and service this new request in the newly created thread. While this approach seems simple to

implement, it has significant disadvantages. A server that creates a new thread for every request would spend more time and consume more system resources in creating and destroying threads than processing actual requests.

Since active threads consume system resources, a JVM creating too many threads at the same time can cause the system to run out of memory. This necessitates the need to limit the number of threads being created. Java thread pool manages the pool of worker threads; it contains a queue that keeps tasks waiting to get executed. We can use `ThreadPoolExecutor` to create thread pool in Java.

Java thread pool manages the collection of `Runnable` threads. The worker threads execute `Runnable` threads from the queue. `java.util.concurrent.Executors` provide factory and support methods for `java.util.concurrent.Executor` interface to create the thread pool in java.

`Executors` is a utility class that also provides useful methods to work with `ExecutorService`, `ScheduledExecutorService`, `ThreadFactory`, and `Callable` classes through various factory methods.

A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing. Since the thread is already existing when the request arrives, the delay introduced by thread creation is eliminated, making the application more responsive.

- Java provides the `Executor` framework which is centered around the `Executor` interface, its sub-interface – `ExecutorService` and the class - `ThreadPoolExecutor`, which implements both of these interfaces. By using the executor, one only has to implement the `Runnable` objects and send them to the executor to execute.
- They allow you to take advantage of threading, but focus on the tasks that you want the thread to perform, instead of thread mechanics.
- To use thread pools, we first create a object of `ExecutorService` and pass a set of tasks to it. `ThreadPoolExecutor` class allows to set the core and maximum pool size. The `Runnable` that are run by a particular thread are executed sequentially.

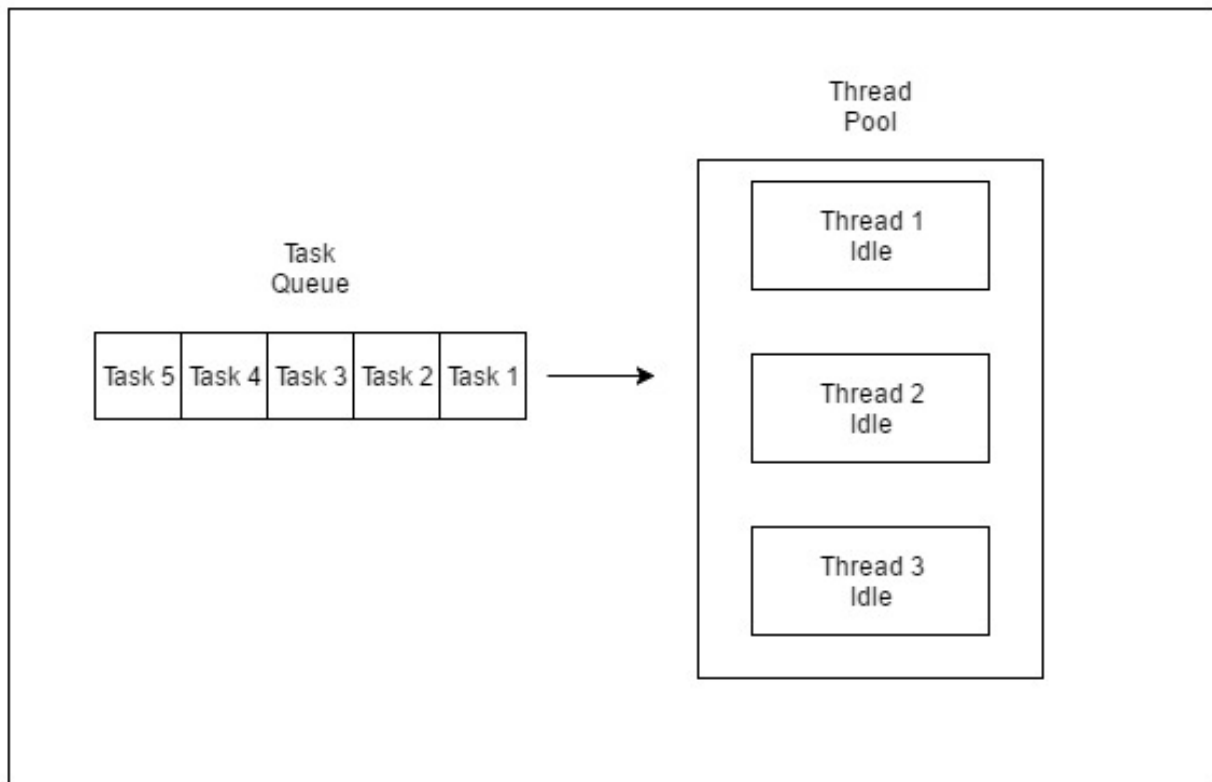


Fig:3.5 Schematic diagram of thread pool

### 3.5.1 Load Testing using threads

Each time a new user accesses the login page, a new thread is generated. Threads run continuously and work as sessions for every user. Load testing can be performed by threads by means of traffic created on the server by them, hence giving the idea of the number of users that a website can handle at a given moment without compromising its responsive nature.

This method of load testing is also called Thread Testing. Thread testing is defined as a software testing type, which verify the key functional capabilities of a specific task(thread). It is usually conducted at the early stage of Integration Testing phase.

Thread based testing is one of the incremental strategies adopted during System Integration Testing.

Thread based testing are classified into two categories

- **Single thread testing:** A single thread testing involves one application transaction at a time
- **Multi-thread testing:** A multi-thread testing involves several concurrently active transactions at a time

#### Features

The thread process focuses on the integration activities rather than the full development lifecycle. For Example,

- Thread-based testing is a generalized form of session-based testing, in that sessions are a form of thread, but a thread is not necessarily a session.
- For thread testing, the thread or program (small functionality) are integrated and tested incrementally as a subsystem, and then executed for a whole system.
- At the lowest level, it provided integrators with better knowledge of the scope of what to test
- Rather than testing software components directly, it required integrators to concentrate on testing logical execution paths in the context of the entire system.

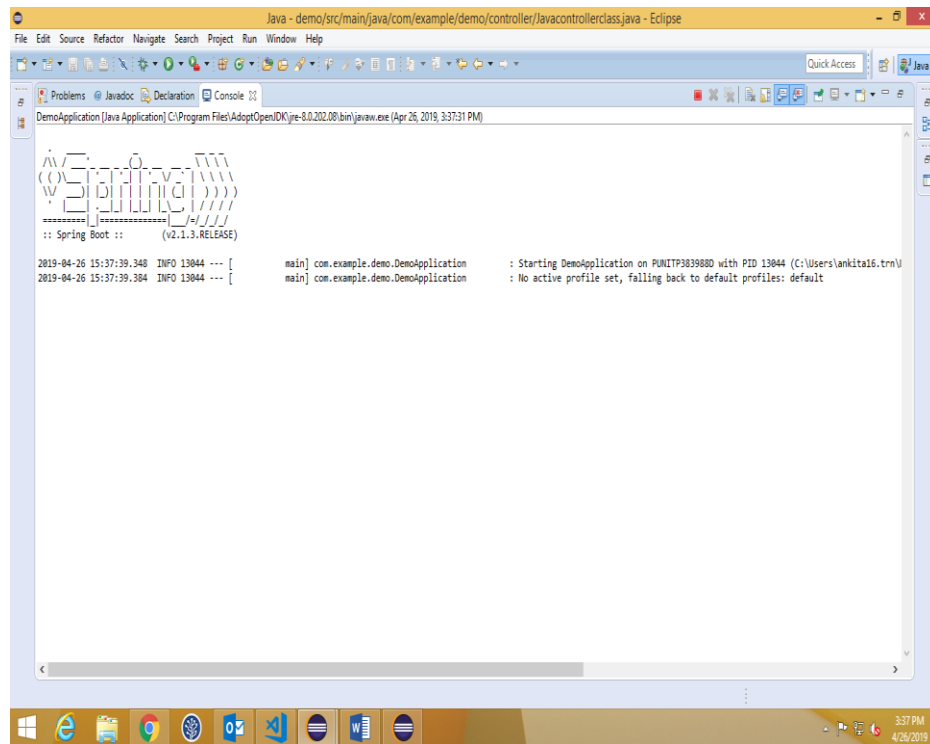


Fig 3.5 Console snippet from Thread pooling in Java

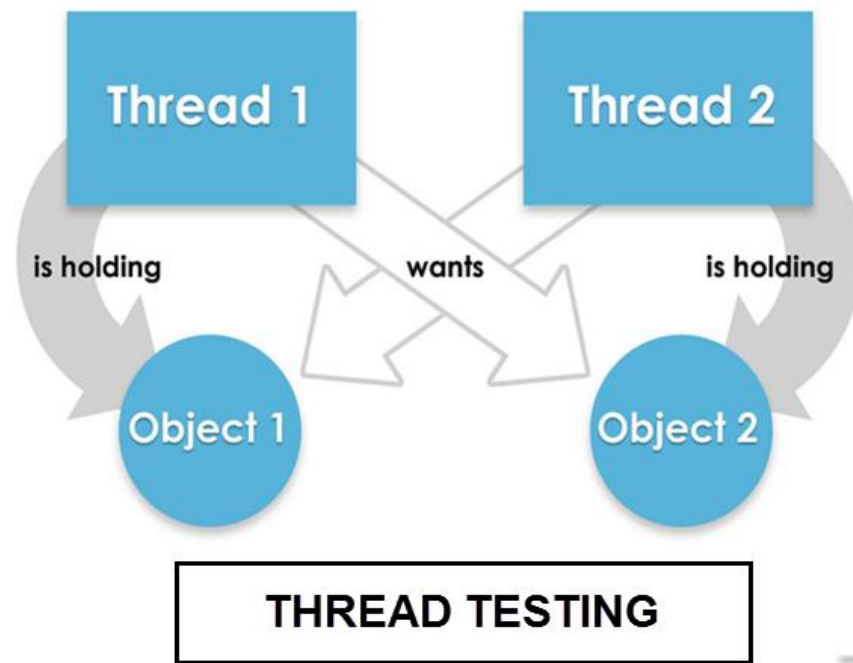


Fig 3.6 Deadlock type situation causing the server performance delay



## CHAPTER 4

### 4.1 Problem Statement

Develop the user interface and implement APIs using Swagger UI for an open banking login portal. Create a dummy service to implement the authentication process of the user details and develop integration layer component for the open banking and also peripheral services using Eureka naming server that would support the smooth execution and usage of the microservices.

### 4.2 Software Requirement

#### 4.2.1 Java<sup>[2]</sup>

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]). With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms such as for example: J2EE for Enterprise Applications, J2ME for Mobile Applications and runs on variety of platform as Windows, Mac OS and various versions of UNIX. Java is guaranteed to be Write Once, Run Anywhere.

#### Java Features:

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** – Java is designed for the distributed environment of the internet.

- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

#### 4.2.2 Eclipse IDE<sup>[3]</sup>

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE.[6] It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, JavaScript, Julia,[7] Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Rust, Scala, and Scheme.

#### 4.2.3 RDBMS<sup>[4]</sup>

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds other kinds of data stores can be used such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those types of systems. So nowadays we use relational database management system.

RDBMS stands for Relational Database Management Systems. All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS. It is called Relational Data Base Management System (RDBMS) because it is based on relational model introduced by E.F. Codd. It allows the user to construct, modify and administer a relational database. Most of the databases that exist today are an extension of this age-old model. The data is structured in database tables, fields and records.

### Features of RDBMS:

- The system caters to a wide variety of applications and quite a few of its stand out features enable its worldwide use. The features include:
- First of all, its number one feature is the ability to store data in tables. The fact that the very storage of data is in a structured form can significantly reduce iteration time.
- Data persists in the form of rows and columns and allows for a facility primary key to define unique identification of rows.
- It creates indexes for quicker data retrieval.
- Allows for various types of data integrity like (i) Entity Integrity; wherein no duplicate rows in a table exist, (ii)Domain Integrity; that enforces valid entries for a given column by filtering the type, the format, or the wide use of values, (iii)Referential Integrity; which disables the deletion of rows that are in use by other records and (iv)User Defined Integrity; providing some specific business rules that do not fall into the above three.
- Also allows for the virtual table creation which provides a safe means to store and secure sensitive content.
- Common column implementation and also multi user accessibility is included in the RDBMS features.

### RDBMS Terminology:

1. Database: Database is a collection of tables with related data.
2. Table: A table is a collection of rows and columns.
3. Column: one column contains data of same kind for example column city.
4. Row:Row is also called a tuple in RDBM (also entry, record) is a group of related data.
5. Primary key:A column or group of columns in a table which helps us to uniquely identifies every row in that table is called a primary key.

6. **Foreign Key:**A foreign key is a column which is added to create a relationship with another table. Foreign keys help us to maintain data integrity and also allows navigation between two different instances of an entity.
7. **Compound key:**Compound key has many fields which allow you to uniquely recognize a specific record. It is possible that each column may be not unique by itself within the database.
8. **Index:** An index in a database resembles an index at the back of book.
9. **Referential integrity:**Referential integrity is a property of data stating that all of its references are valid. In the context of relational databases, it requires that if a value of one attribute (column) of a relation (table) references a value of another attribute (either in the same or a different relation), then the referenced value must exist.

#### **4.2.4 MySQL Workbench<sup>[5]</sup>**

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.

##### **4.2.4.1 Design**

MySQL Workbench enables a DBA, developer, or data architect to visually design, model, generate, and manage databases. It includes everything a data modeler needs for creating complex ER models, forward and reverse engineering, and also delivers key features for performing difficult change management and documentation tasks that normally require much time and effort.

#### **4.2.4.2 Develop**

MySQL Workbench delivers visual tools for creating, executing, and optimizing SQL queries. The SQL Editor provides color syntax highlighting, auto-complete, reuse of SQL snippets, and execution history of SQL. The Database Connections Panel enables developers to easily manage standard database connections, including MySQL Fabric. The Object Browser provides instant access to database schema and objects.

#### **4.2.4.3 Administer**

MySQL Workbench provides a visual console to easily administer MySQL environments and gain better visibility into databases. Developers and DBAs can use the visual tools for configuring servers, administering users, performing backup and recovery, inspecting audit data, and viewing database health.

#### **4.2.4.4 Visual Performance Dashboard**

MySQL Workbench provides a suite of tools to improve the performance of MySQL applications. DBAs can quickly view key performance indicators using the Performance Dashboard. Performance Reports provide easy identification and access to IO hotspots, high cost SQL statements, and more. Plus, with 1 click, developers can see where to optimize their query with the improved and easy to use Visual Explain Plan.

#### **4.2.4.5 Database Migration**

MySQL Workbench now provides a complete, easy to use solution for migrating Microsoft SQL Server, Microsoft Access, Sybase ASE, PostgreSQL, and other RDBMS tables, objects and data to MySQL. Developers and DBAs can quickly and easily convert existing applications to run on MySQL both on Windows and other platforms. Migration also supports migrating from earlier versions of MySQL to the latest releases.

#### 4.2.5 SPRING BOOT AND MAVEN<sup>[6]</sup>

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. 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 automatically configures your application based on the dependencies you have added to the project by using `@EnableAutoConfiguration` annotation. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains `@SpringBootApplication` annotation and the main method.

Spring Boot automatically scans all the components included in the project by using `@ComponentScan` annotation.

Advantages:

Spring Boot offers the following advantages to its developers –

- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time
- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container

Goals:

Spring Boot is designed with the following goals –

- To avoid complex XML configuration in Spring
- To develop a production ready Spring application in an easier way
- To reduce the development time and run the application independently
- Offer an easier way of getting started with the application

#### **4.2.5.1 Embedded server**

Spring boot applications always include tomcat as embedded server dependency. It means you can run the Spring boot applications from the command prompt without needing complex server infrastructure.

You can exclude tomcat and include any other embedded server if you want. Or you can make exclude server environment altogether. It's all configuration based.

#### **4.2.5.2 MAVEN<sup>[7]</sup>**

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development team's environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following –

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs



- Releases
- Distribution
- Mailing list

#### 4.2.6 RESTFUL WEBSERVICES<sup>[8]</sup>

RESTful Web Service or Representational State Transfer (REST) adheres to set of architectural principles which makes RESTful Webservice fast, scalable, modifiable. Below is the list of principles. The framework is built on 3 main components:

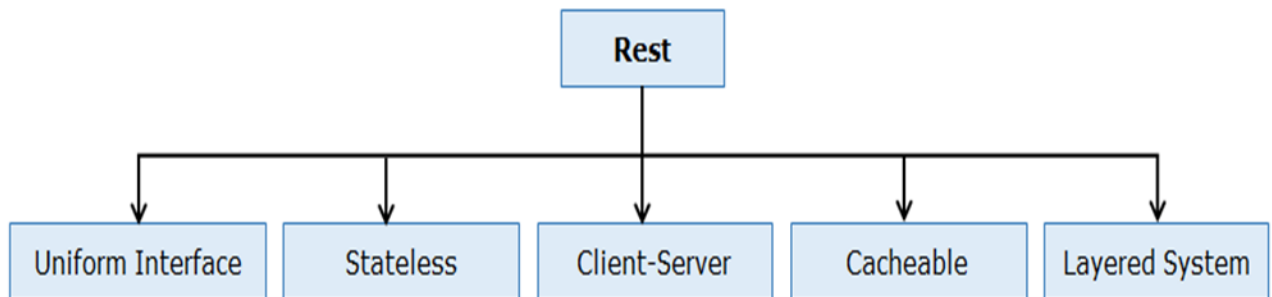


Fig-4.1

##### 4.2.6.1 Uniform Interface

In REST, data and functionality are considered as a Resource. Every resource has a Representation (XML/JSON) and an URI to uniquely identify it. Resources are manipulated using GET, POST, PUT and DELETE methods of HTTP to perform read, create, update and delete of the resource.

##### 4.2.6.2 Stateless

The communication between service consumer (client) and service provider(server) must be stateless between requests. Stateless requires each request from a service consumer to contain all the necessary information for the service to understand the meaning of the request and all session state data should then be returned to the service consumer at the end of each request.

#### **4.2.6.3 Client Server**

Client-Server enforces the separation of concerns in the form of a client-server architecture, this helps establish a distributed architecture.

#### **4.2.6.4 Cacheable**

Service consumers can cache and reuse response message data. HTTP GET, PUT and DELETE operations are cacheable. RESTful Web Services use HTTP protocol as a carrier for the data, they can use the metadata available in HTTP headers to enable caching. For example: Cache-Control headers are used to optimize caching for enhancing performance.

#### **4.2.7 MICROSERVICES<sup>[9]</sup>**

Microservices also known as the microservice architecture. It is an architectural style that structures an application as a collection of services that are:

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities.

Its main principle is to break single large monolithic system into multiple independent component

Traditional monolithic application was good solution in earlier times in this all the functionalities are implemented and embedded as a part of single application. It fails in some areas as follows:

- Entire application has to be redeployed even for a small change.
- Larger deployment and startup time in case of large app.
- Difficulty in case of switching to another technology.
- Scaling is not easy.

Microservice architecture offers:

- **Fast Development:** Developers can independently develop and deploy modules or services as there is loose coupling between them.
- **Scalability:** Scale services as needed.
- **Better fault tolerance:** If one microservice fails other will continue to deliver.
- **Eliminates technology dependency:** As technologies are ever changing, whenever a new and efficient technology we can try it out on just one service without changing other services.
- **Easy understanding:** Smaller services are easy to understand for a new developer as it can be traced end to end easily

#### 4.2.7.1 Monolithic Architecture

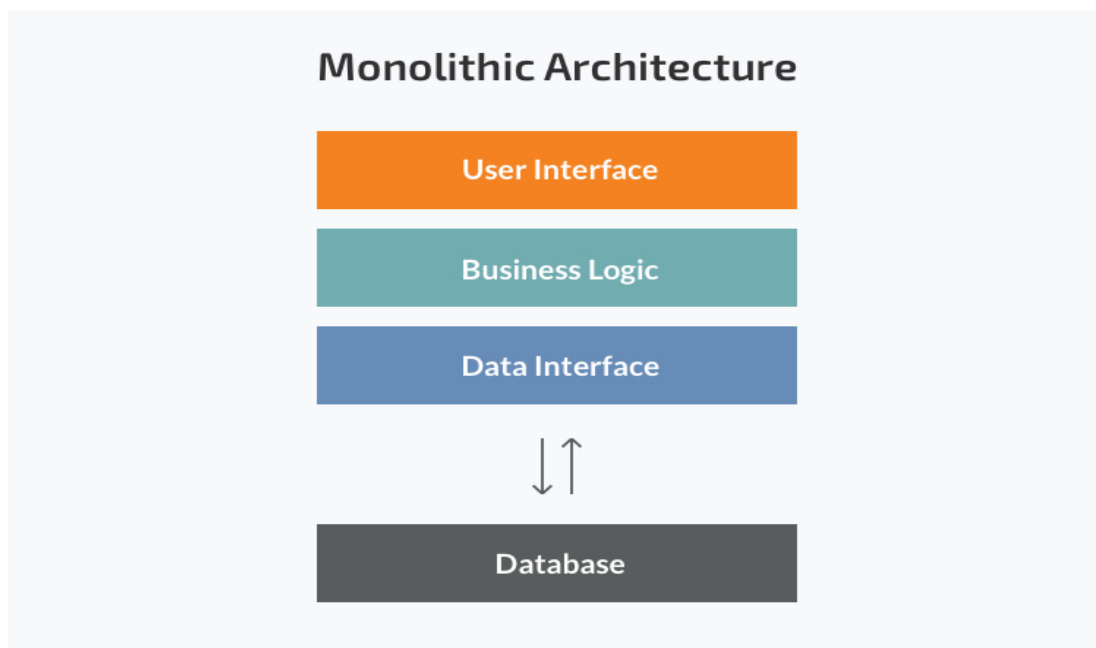


Fig:4.2

#### 4.2.7.2 Microservice Architecture

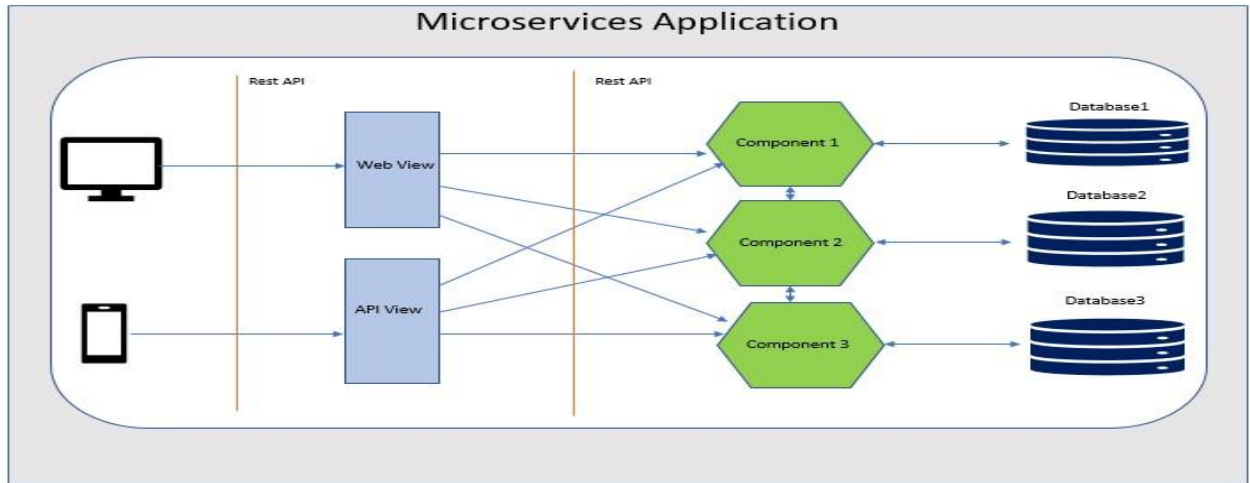


Fig:4.3

#### 4.2.8 JPA <sup>[10]</sup>

The Java Persistence API (JPA) is the standard way of persisting Java objects into relational databases. The JPA consists of two parts: a mapping subsystem to map classes onto relational tables as well as an EntityManager API to access the objects, define and execute queries, and more. JPA abstracts a variety of implementations such as Hibernate, EclipseLink, OpenJpa, and others.

Hibernate is one of the popular implementations of JPA. Hibernate understands the mappings that we add between objects and tables. It ensures that data is stored/retrieved from the database based on the mappings. Hibernate also provides additional features on top of JPA.

Dependency required for JPA:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jpa</artifactId>

</dependency>

#### 4.2.8.1 JpaRepository

JpaRepository is a spring data repository interface which extends PagingAndSortingRepository which in turn extends CrudRepository. Each of these defines its own functionality:

- CrudRepository provides CRUD functions
- PagingAndSortingRepository provides methods to do pagination and sort records
- JpaRepository provides JPA related methods such as flushing the persistence context and delete records in a batch

And so, because of this inheritance relationship, the JpaRepository contains the full API of CrudRepository and PagingAndSortingRepository.

Typical CRUD functionality:

- save (...) – save an Iterable of entities. Here, we can pass multiple objects to save them in a batch
- findOne(...) – get a single entity based on passed primary key value
- findAll() – get an Iterable of all available entities in database
- count() – return the count of total entities in a table
- delete(...) – delete an entity based on the passed object
- exists(...) – verify if an entity exists based on the passed primary key value
- flush() – flush all pending task to the database
- saveAndFlush(...) – save the entity and flush changes immediately
- deleteInBatch(...) – delete an Iterable of entities. Here, we can pass multiple objects to delete them in a batch.

#### 4.2.9 REST TEMPLATE<sup>[8]</sup>

Rest Template is used to create applications that consume RESTful Web Services. You can use the exchange() method to consume the web services for all HTTP methods.

You will have to follow the given points to consume the GET API –

- Autowired the Rest Template Object.
- Use HttpHeaders to set the Request Headers.
- Use HttpEntity to wrap the request object.
- Provide the URL, HttpMethod, and Return type for Exchange() method.

You will have to follow the points given below to consume the POST API –

- Autowired the Rest Template Object.
- Use the HttpHeaders to set the Request Headers.
- Use the HttpEntity to wrap the request object. Here, we wrap the Product object to send it to the request body.
- Provide the URL, HttpMethod, and Return type for exchange() method.

#### **4.2.10 SPRING BATCH<sup>[11]</sup>**

Batch processing is a processing mode which involves execution of series of automated complex jobs without user interaction. A batch process handles bulk data and runs for a long time. For example reading a csv file and storing its data into database.

Several Enterprise applications require to process huge data to perform operations involving –

- Time-based events such as periodic calculations.
- Periodic applications that are processed repetitively over large datasets.
- Applications that deals with processing and validation of the data available in a transactional manner.

Therefore, batch processing is used in enterprise applications to perform such transactions.

Spring batch is a lightweight framework which is used to develop Batch Applications that are used in Enterprise Applications.

In addition to bulk processing, this framework provides functions for –

- Including logging and tracing
- Transaction management
- Job processing statistics
- Job restart
- Skip and Resource management

Features of Spring Batch:

Following are the notable features of Spring Batch –

1. Flexibility – Spring Batch applications are flexible. You simply need to change an XML file to alter the order of processing in an application.
2. Maintainability – Spring Batch applications are easy to maintain. A Spring Batch job includes steps and each step can be decoupled, tested, and updated, without effecting the other steps.
3. Scalability – Using the portioning techniques, you can scale the Spring Batch applications.
4. Execute the steps of a job in parallel.
5. Execute a single thread in parallel.
6. Reliability – In case of any failure, you can restart the job from exactly where it was stopped, by decoupling the steps.
7. Support for multiple file formats – Spring Batch provides support for a large set of readers and writers such as XML, Flat file, CSV, MYSQL, Hibernate, JDBC, Mongo, Neo4j, etc.
8. Multiple ways to launch a job – You can launch a Spring Batch job using web applications, Java programs, Command Line, etc.

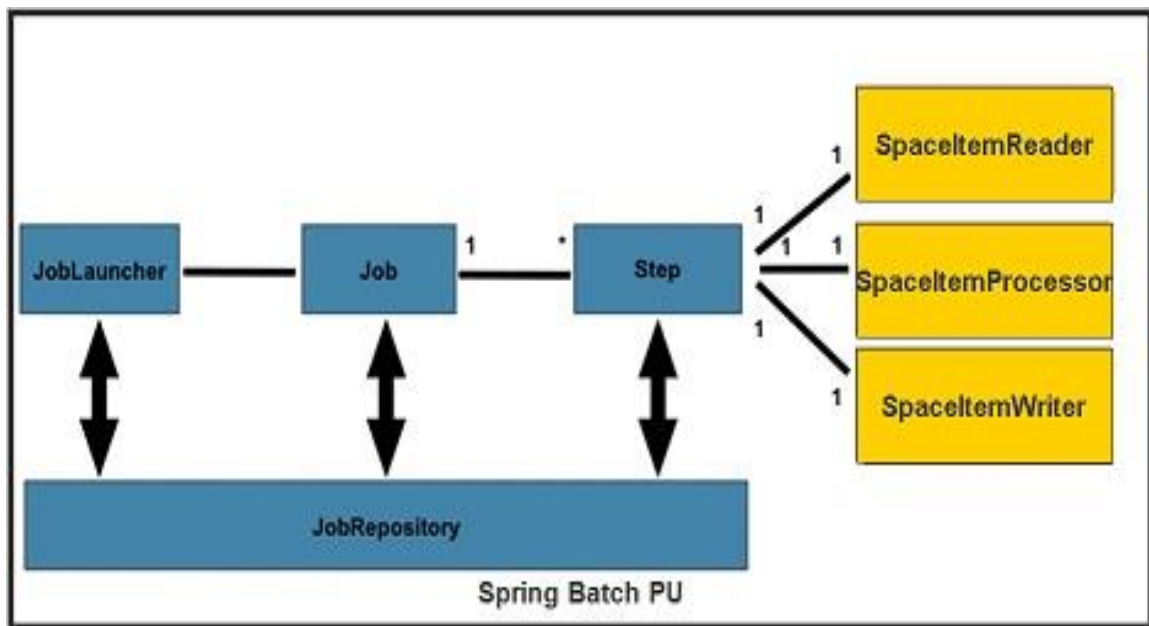


Fig:4.4-Spring Batch Process

Any Spring Batch job has 2 main components, Job, and Steps. A job may have multiple steps. Each step has a compulsory reader and writer routines and an optional processor unit.

**Step:** A Step is a fundamental unit of any Job. It is a domain object that encapsulates an independent, sequential phase of a batch job. A Step defines necessary information to define and control the actual batch processing.

**Item Reader:** ItemReader is an abstract representation of how data is provided as input to a Step. When the inputs are exhausted, the ItemReader returns null.

**Item Processor:** ItemProcessor represents the business processing of an item. The data read by ItemReader can be passed on to ItemProcessor. In this unit, the data is transformed and sent for writing. If, while processing the item, it becomes invalid for further processing, you can return null. The nulls are not written by ItemWriter.

**Item Writer:** ItemWriter is the output of a Step. The writer writes one batch or chunk of items at a time to the target system. ItemWriter has no knowledge of the input it will receive next, only the item that was passed in its current invocation.

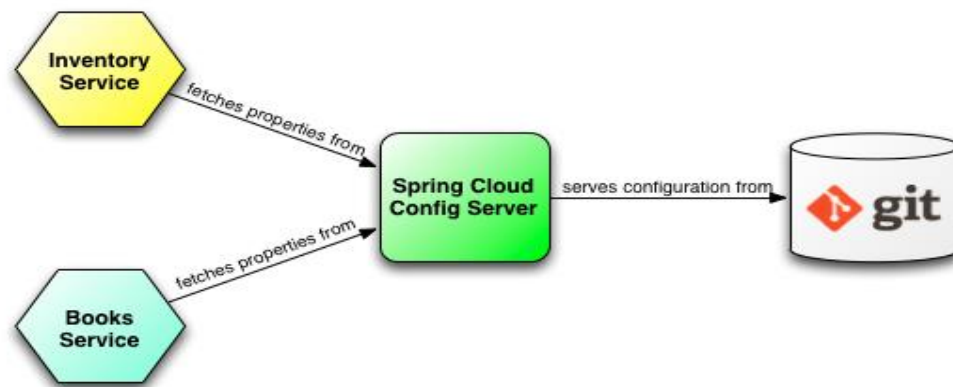


## 4.2.11 CONFIG SERVER AND EUREKA NAMING SERVER<sup>[12]</sup>

### 4.2.11.1 Config Server

Spring Cloud Config is Spring's client/server approach for storing and serving distributed configurations across multiple applications and environments. Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system. With the Config Server you have a central place to manage external properties for applications across all environments.

- Features
- It is an Http resource-based API for external configuration.
- It has encryption and decryption values also known as symmetric and asymmetric.
- It can be used easily in a Spring Boot application using `@EnableConfigServer`.



#### 4.2.11.1.1 Config Server – Server-Side Configuration

- Generate the project structure
- Import the project in Eclipse
- Build in eclipse
- Add Config Sever Annotation
- Create the Git repository
- Point the git repo from Config Server
- Verify Server-Side Configuration

#### **4.2.11.1.2 Config Server – Client-Side Configuration**

- Create Maven Project
- Create REST Resource
- Bind with the Config Server
- Verify Client Config

#### **4.2.11.2 Eureka Naming Server**

Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.

##### **4.2.11.2.1 Enabling Eureka**

EnableEurekaServer in SpringBootMicroserviceEurekaNamingServerApplication.

@SpringBootApplication

@EnableEurekaServer

```
Public class SpringBootMicroserviceEurekaNamingServerApplication {  
}
```

##### **4.2.11.2.2 Launching Eureka Naming Server**

Launch SpringBootMicroserviceEurekaNamingServerApplication as a Java application.  
Launching up Eureka at <http://localhost:8761>.

#### **4.2.12 JENKINS<sup>[13]</sup>**

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. It is an open source automation server written in Java. This helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery.

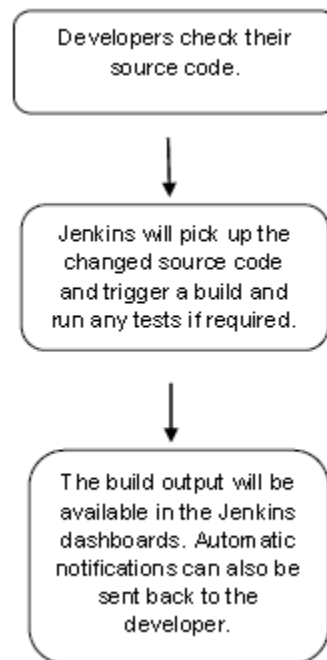


Fig:4.5

#### 4.2.12.1 Starting Jenkins

Open the command prompt. From the command prompt, browse to the directory where the jenkins.war file is present. Run the following command

```
D:\>Java -jar Jenkins.war
```

After the command is run, various tasks will run, one of which is the extraction of the war file which is done by an embedded webserver called winstone.

#### 4.2.12.2 Accessing Jenkins

Once Jenkins is up and running, one can access Jenkins from the link – <http://localhost:8080>

#### 4.2.13 FEIGN<sup>[14]</sup>

Feign is a declarative web service client. It makes writing web service clients easier. To use Feign create an interface and annotate it. It has pluggable annotation support including Feign annotations and JAX-RS annotations. Feign also supports pluggable encoders and decoders.

#### 4.2.13.1 Including Feign

To include Feign in your project use the starter with group `org.springframework.cloud` and artifact id `spring-cloud-starter-openfeign`.

```
@SpringBootApplication
@EnableFeignClients
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

@FeignClient("stores")
public interface StoreClient {
    @RequestMapping(method = RequestMethod.GET, value = "/stores")
    List<Store> getStores();

    @RequestMapping(method = RequestMethod.POST, value = "/stores/{storeId}", consumes =
"application/json")
    Store update(@PathVariable("storeId") Long storeId, Store store);
}
```

#### 4.2.14 SWAGGER<sup>[16]</sup>

Swagger allows you to describe the structure of your APIs so that machines can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger. Swagger is a set of rules (in other words, a specification) for a format describing REST APIs. The format is both machine-readable and human-readable. As a result, it can be used to share documentation among product managers, testers and developers, but can also be used by various tools to automate API-related processes.

##### 4.2.14.1 Swagger Framework and benefits

First and foremost, as Swagger uses a common language that everyone can understand, it's easily comprehensible for both developers and non-developers. Also, as Swagger is easily adjustable, it can be successfully used for API testing and bug fixing. Another important point is that the same documentation can be used for accelerating various API-dependent processes.

Swagger provides a set of great tools for designing APIs and improving the work with web services:

- Swagger Editor

It enables to write API documentation, design and describe new APIs, and edit the existing ones. The first open-source editor visually renders OAS/Swagger definition with error handling and real-time feedback.

- Swagger Codegen

It allows developers to generate client library code for different platforms. As the tool helps facilitate the dev process by generating server stubs and client SDKs, software engineers get the ability to faster build your API and better focus on its adoption.

- Swagger UI

It allows engineers to get self-generated documentation for different platforms. Swagger UI is a fully customizable tool that can be hosted in any environment. A great plus is that it enables developers to save a lot of time for API documentation.

#### **4.2.15 Angular 4<sup>[17]</sup>**

Angular 4 is a JavaScript framework for building web applications and apps in JavaScript, html, and Typescript, which is a superset of JavaScript. It provides built-in features for animation, http service, and materials which in turn has features such as auto-complete, navigation, toolbar, menus, etc. The code is written in Typescript, which compiles to JavaScript and displays the same in the browser. There are three major releases of Angular. The first version that was released is Angular1, which is also called AngularJS. Angular 4 released in March 2017 proves to be a major breakthrough and is the latest release from the Angular team after Angular2. While, we create apps using HTML, CSS and JavaScript, Angular requires us to

know **Typescript**, kind of stricter version of JavaScript provided with OOPS features. Although, other alternative could be Dart, typescript is the most widely used language for Angular apps.

#### Angular CLI

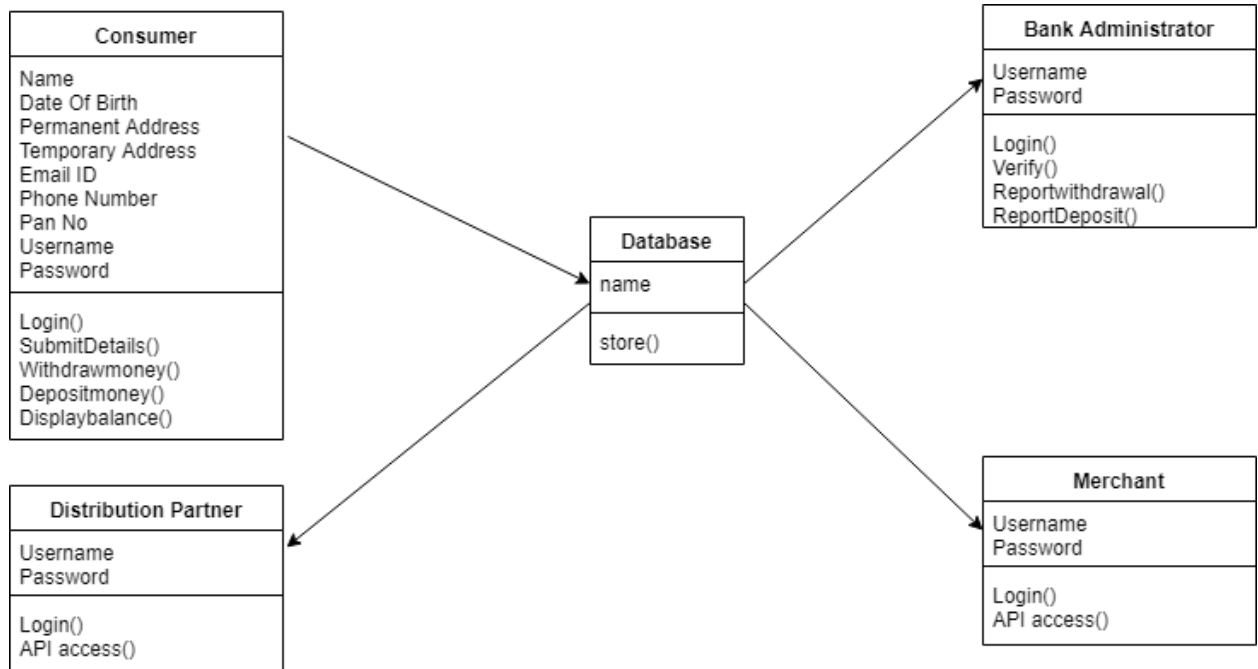
- Command Line Interface (CLI) can be used to create our Angular JS application. It also helps in creating a unit and end-to-end tests for the application.
- Commands are used to use CLI:
  - Command to install angular CLI tool installed in your system:  
**“npm install -g @angular/cli”**
  - Command to make and run the angular project:  
**“ng new my-angular-app”**  
**“cd my-angular-app”**  
**“ng serve –open”**
  - Command to make new component:  
**“ng new component myapp”**

## CHAPTER 5

### SOFTWARE DESIGN

#### 5.1 UML Diagrams

##### 5.1.1 Class Diagram



##### 5.1.1.1 Documentation of Class Diagram

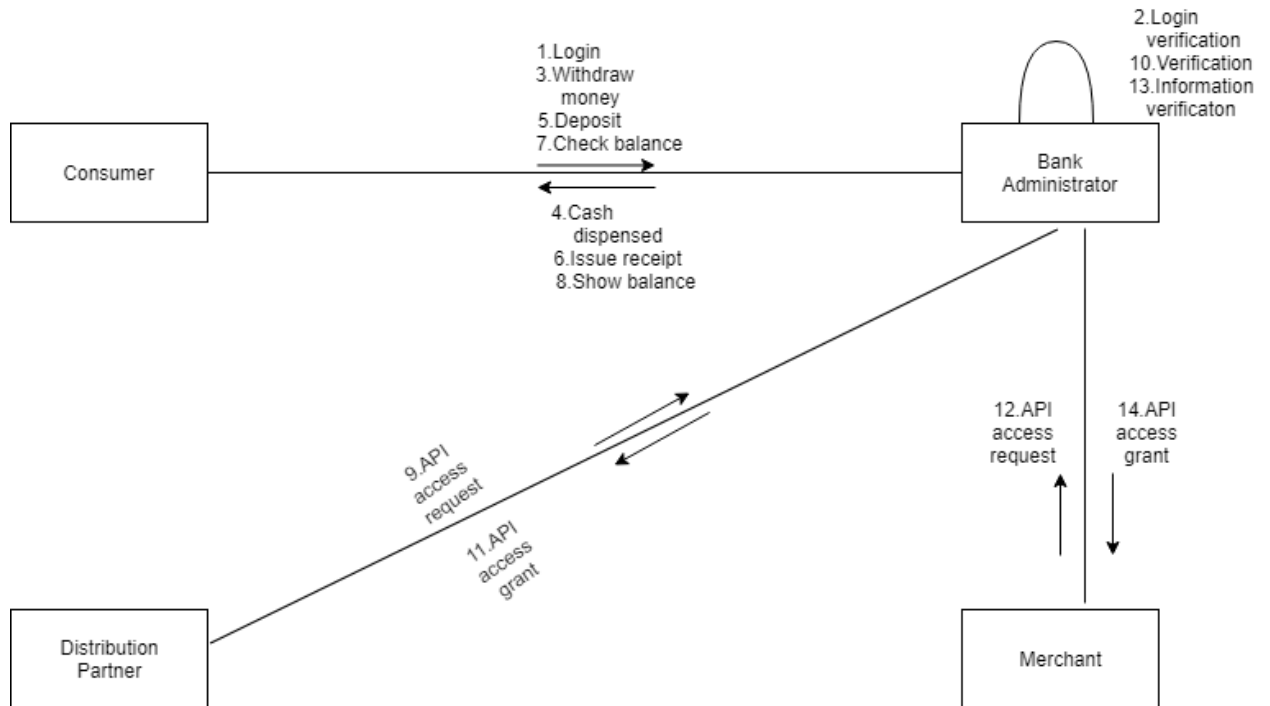
**a) Consumer:** The consumer has attribute such as name and password and operations are login, submitdetails,withdrawmoney,depositmoney and logout. The applicant loginand fill the details that are required for verification and withdrawing and depositing money.

**b) Bank Administrator:**The bank administrator hasattribute such as name and operation are getting details, verify details andsend. The regional administrator gets the details form database andverify with their database.

**c)Distribution Partner:**The distribution partner has attributes such as name and password and operations are login and API access.

**d)Database:**The database has attributed such as name andoperation is store. The purpose is to store the data.

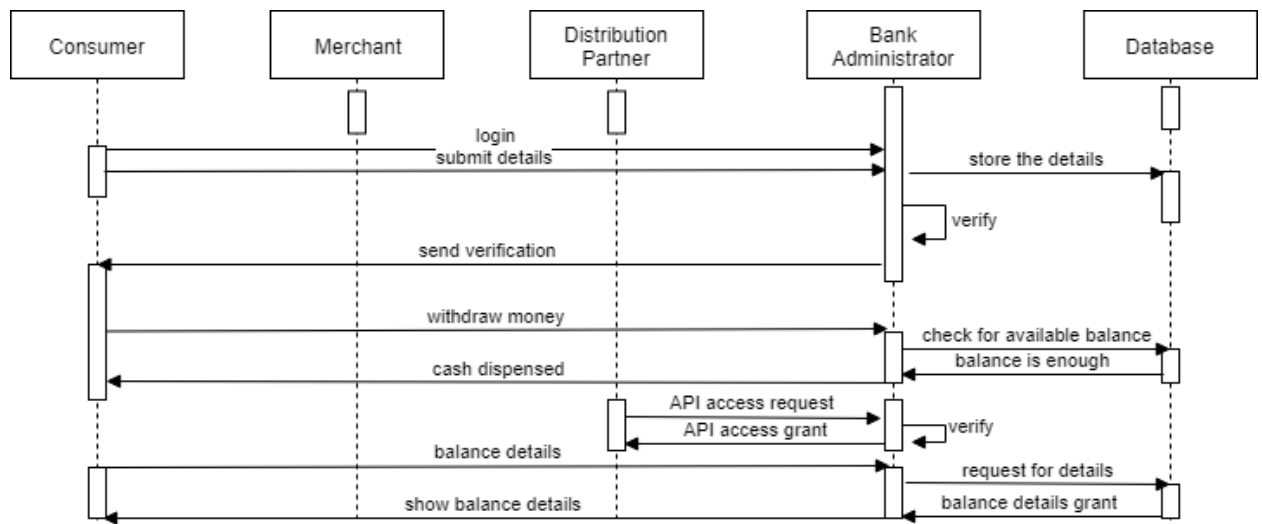
### 5.1.2 Collaboration Diagram



#### 5.1.2.1 Documentation of Collaboration Diagram

The collaboration diagram is to show how the consumer registers and the authorities maintains the details of the logged in consumer in the bank system. Here the sequence is numbered according to the flow of execution.

### 5.1.3 Sequence Diagram



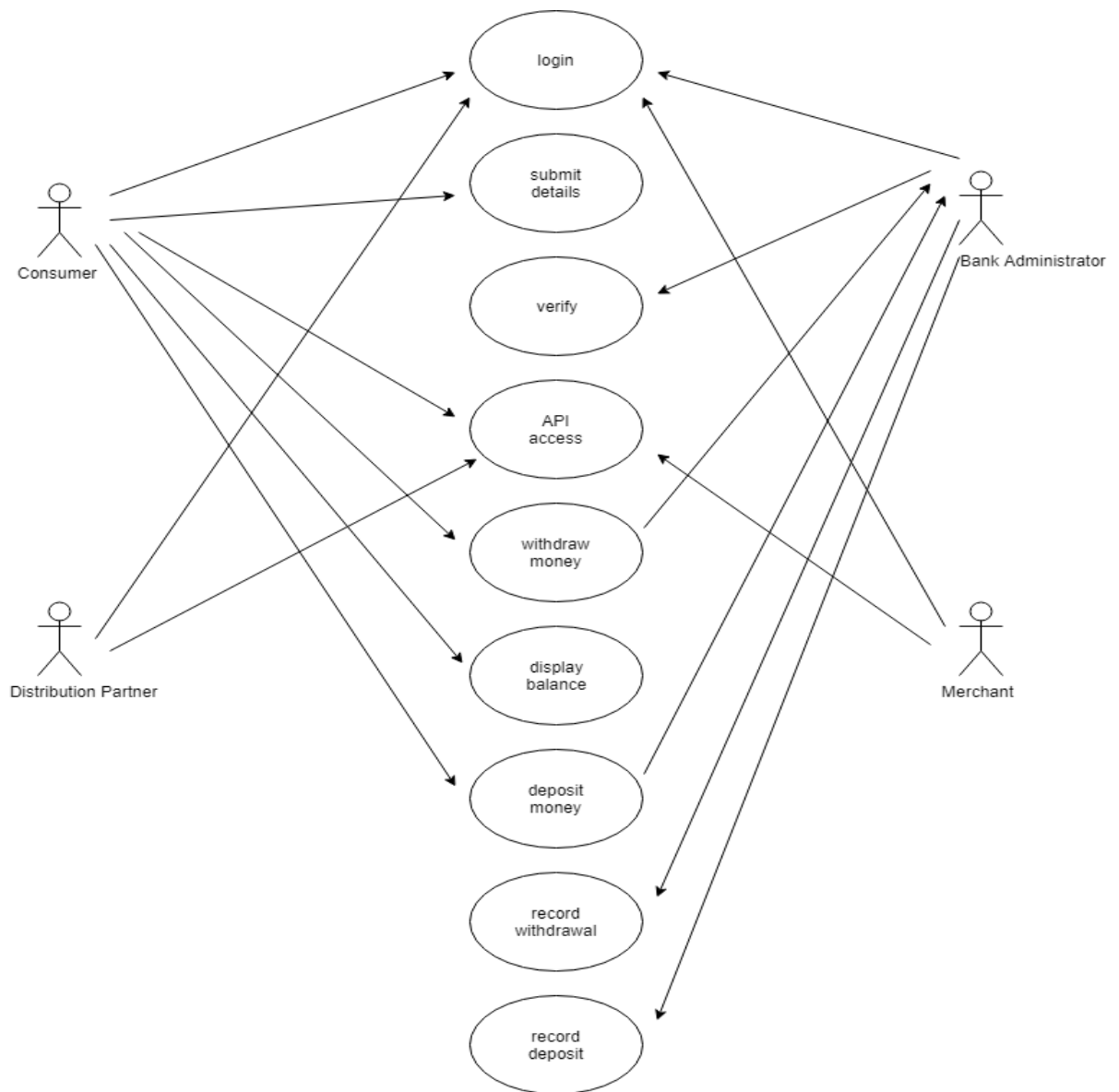


### 5.1.3.1 Documentation of Sequence Diagram

The sequence diagram describes the sequence of steps to show

- The consumer submitting details and ask for different functions from the bank administrator.
- The verification done by the bank system and sending acknowledgement for actions.
- Searching the database with login and displaying it for maintenance.

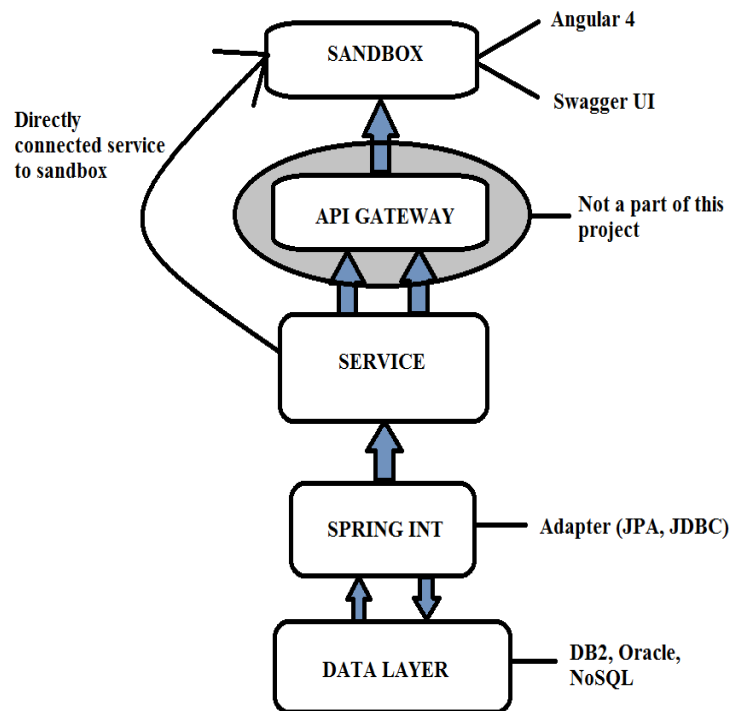
### 5.1.4 UseCase Diagram



#### 5.1.4.1 Documentation of Use-Case Diagram

- a. The actors in use case diagram are Consumer, bank administrator, database, merchant, and distribution partner.
- b. The use cases are Login, give details, verification, withdraw money, deposit money, report, deposit, API access and report withdraw.
- c. The actors use the use case are denoted by the arrow
- d. The login use case checks the username and password for Consumer, merchant, distribution partner.
- e. The submit details use case is used by the consumer for submitting his details.
- f. The check balance use case is used by the consumer for checking the status of his account.
- g. The verification of details, report withdraw and report deposit use case is used by bank administrator.
- h. The verify use case is used for verifying the details by comparing the data in the database.

#### 5.1.5 Control Flow Diagram



## **CHAPTER 6**

### **Software and Hardware requirements**

#### **6.1 Software required**

- Java 8
- Eclipse Oxygen IDE for Java developers, 64-bit
- Visual Studio Code v1.31, 64-bit
- Apache Spring Boot v2.0.4
- Apache Maven v3.5.3
- AngularJS v4.0
- Maven
- Swagger
- Postman
- JPA
- Eureka

#### **6.2 Hardware required**

- Windows 7 Enterprise 64-bit/10 64-bit
- 8 GB RAM (minimum)
- Ethernet
- Processor: 2GHz (minimum)

## CHAPTER 7

### Coding Explanation

#### 7.1 First Part

##### 7.1.1 First Service:

Entity class: A Java class that is marked (annotated) as having the ability to represent objects in the database.

We created a class named **Account Entity** for mapping with database in our project.

Hibernate detects that the @Id annotation is on a field and assumes that it should access properties of an object directly through fields at runtime.

Following are the annotations used in the Account Entity class:

- @Entity: It marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.
- @Table: It allows to specify the details of the table that will be used to persist the entity in the database. The @Table annotation provides four attributes, allowing to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table. For now, we used table name, which is account\_details.
- @Id: Each entity bean will have a primary key, which annotate on the class with the @Id annotation. The primary key can be a single field or a combination of multiple fields depending on the table structure. We took account\_id as primary key.
- @Column: is used to specify the details of the column to which a field or property will be mapped. We used it to override the name of customer\_name column.

Model class: It is a class which represents data object which can be used for transferring data in java application. It encapsulates direct access to data in object and ensures all data in object is accessed via getter methods.

We created a class named **Account** in our project.

Following are the functions defined in this class:

- `Account(AccountEntity obj)`: This parameterized constructor used to initialize the parameters like `account_no` by the value of `account_no` get by the `AccountEntity` class.
- `getAccount_type()`: This function return the `account_type` to the class.
- `setAccount_type(String account_type)`: This function is to initialize the local `account_type` parameter with the returned `account_type`.

Service class: Spring `@Service` annotation is used with classes that provide some business functionalities. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used.

We created a class named **Account** Service in our project for the main business logic. We used `@Autowired` annotation to inject the object dependency implicitly for `AccountDao` class.

Following are two functions defined in this class:

- `getAllAccount()`: This function uses the inbuilt `findAll()` method of JPA repository for fetching the data. The data later on stored in the list and the function returns the `accountlist`.
- `getById()`: This function uses the inbuilt `findOne(id)` method of JPA repository for fetching the data uniquely and returns it.

Controller class: Spring 4.0 introduced `@RestController`, a specialized version of the controller which is a convenience annotation that does nothing more than add the `@Controller` and `@ResponseBody` annotations. By annotating the controller class with `@RestController` annotation, you no longer need to add `@ResponseBody` to all the request mapping methods and autowiring of `AccountService` is done.

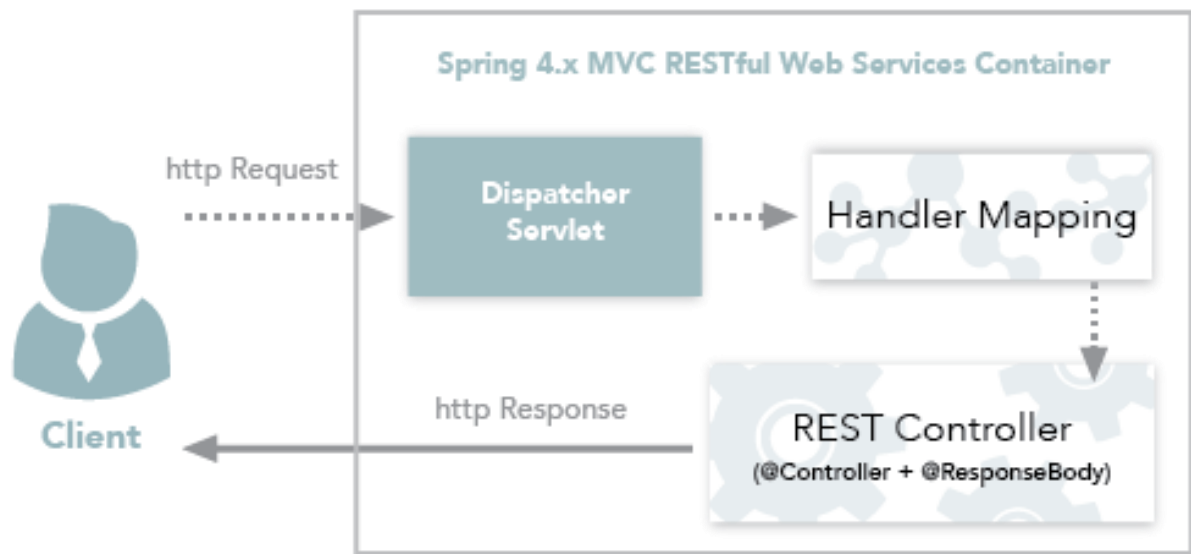


Fig:7.1

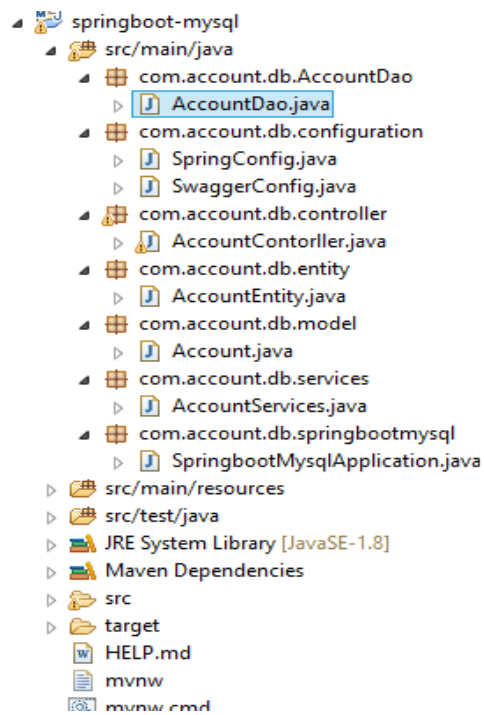


Fig:7.2-Account MicroService Producer

### 7.1.2 Second Service:

Main class: We created rest template bean by using the @Bean annotation(A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container.)

Service class: In this project we created a service class named as **Account Service**.We used @autowired for autowiring the RestTemplate. We provide the serviceUrl as <http://localhost:8080/api/>.

Following are the functions defined in this class:

- getAllAccount():This method is override from the FirstService AccountService class. In this method a function getbody() is used which is inherited from springframework.http.HttpEntity class for fetching the details.
- getById(Id): This method is also override from the FirstService AccountService class. In this method a function getbody() is used which is inherited from springframework.http.HttpEntity class for fetching the details of single account which matches the account\_id uniquely.

Controller class: In this project we created a class named as **Account Controller**.

Following are the annotations used in this class:

- @RequestMapping("api"): This annotation maps HTTP requests to handler methods of MVC and REST controllers.
- @GetMapping("/accounts"): This annotation to map HTTP GET requests onto specific handler methods for fetching all the accounts details.
- @GetMapping("/account/{id}"): This annotation to map HTTP GET requests onto specific handler methods for fetching the unique account details based on account\_id.

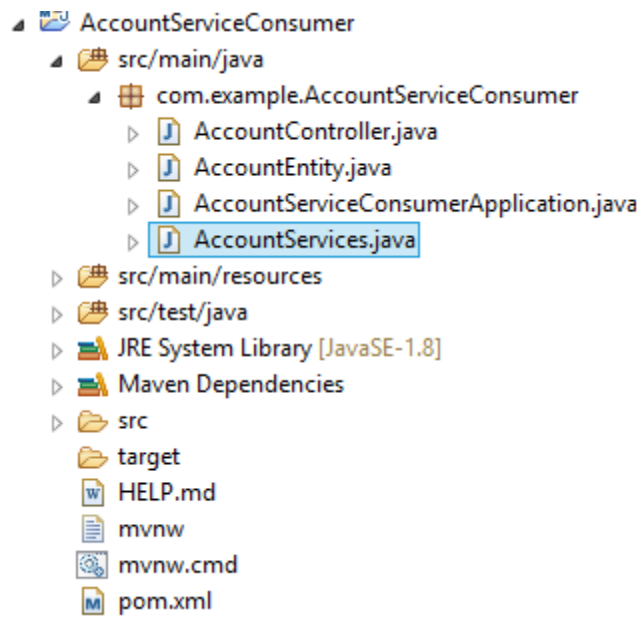


Fig:7.3

## 7.2 Angular Swagger Explanation

### 7.2.1 API Service For the testing

**Main Class:** Main class is connected to the swagger config class. And in Swagger Config class we are calling the API Class.

**API Class:** This Class has main function.

**Main function:** Main function has three main method as Get method and Post method and database setting method. Main API is using @RequestMapping annotation to make the things clearer the function that we are using is doing what work. Get API is getting all the values of the people who are there as the customer by first setting the values of the customer details. @GET is used above this to annotate it as the get api. Post API method takes the value from the person of customer account number and then traversing the data base and then getting all the details of that account number in the database.

**Customer Class:** Class contain the get and set method for the variables as accountnumber, customername.



All the above Api are being displayed through the swagger UI tool which provide the user interface which use the services and display the api so that the API can be checked through it.

### **7.2.2 Angular Part**

In angular part we have used the command `ng new component` to make the new component and after the component is build, we just make changes in the files as “app.componet.html”.

In app.componet.html we have used various bootstrap part as nav bars in which we have used and called the router outlet by using tag `<routeroutlet></routeroulet>`.

Then we have made another component for login page and made changes in “app.module.ts” and “app-routing.module.ts”.

In app.module.ts we have added a component as LoginComponet and in app-routing.module.ts we have added the routers part and add the link of the login component.

In login part we have used validators which will validate that the account number should be a number and it should be of 12 numbers only, customer name should be letters only and the should be match regression as “A-Za-z ” and password should be more than 8 letters.

In login.componet.html we have used made the lay out of the page by adding the fields as account number, customer name and password and used `ngIf` which validate all the fields and when all the fields are okay the login button will appear which is embedded in code using `<button>` tag.

And when the login is done the Swagger local host is open and the API is tested.

Login host used for displaying the main web page is [loginhost:4200/login](http://loginhost:4200/login).

Login host used for displaying swagger ui is [loginhost:8080/swagger-ui.html](http://loginhost:8080/swagger-ui.html).

## CHAPTER 8

### Testing of Java API

Postman is an API(application programming interface) development tool which helps to build, test and modify APIs. Almost any functionality that could be needed by any developer is encapsulated in this tool. It has the ability to make various types of HTTP requests(GET, POST, PUT, PATCH), saving environments for later use, converting the API to code for various languages(like JavaScript, Python).

All account details

<http://localhost:2222/api/accounts>

The screenshot shows the Postman interface with a GET request to `http://localhost:2222/api/accounts`. The response is a JSON array of account details, displayed in the 'Body' tab. The status is 200 OK, time is 222 ms, and size is 1.06 KB.

```
1 [{"account_no": "11111", "account_type": "current", "name": "Arnab", "contact": "7235647896", "balance": 44000}, {"account_no": "11112", "account_type": "savings", "name": "Anjali Ahuja", "contact": "9557090189", "balance": 20000}, {"account_no": "11113", "account_type": "savings", "name": "Arunima Jain", "contact": "7415266195", "balance": 300000}, {"account_no": "11115", "account_type": "savings", "name": "Chandrashekhar", "contact": "8103485620", "balance": 640000}, {"account_no": "11116", "account_type": "savings", "name": "Garima Rathore", "contact": "7894023591", "balance": 70000}, {"account_no": "11117", "account_type": "savings", "name": "Harsh Soni", "contact": "9684123051", "balance": 10000}, {"account_no": "11118", "account_type": "savings", "name": "Isha Agarwal", "contact": "7845964100", "balance": 8000}, {"account_no": "11119", "account_type": "savings", "name": "Jasmine", "contact": "8954756620", "balance": 6500}, {"account_no": "11120", "account_type": "savings", "name": "Kajol", "contact": "9988264741", "balance": 45000}]
```

## Account get by Id

<http://localhost:2222/api/account/11118>

The screenshot shows a REST client interface with a GET request to `http://localhost:2222/api/account/11118`. The response is a JSON object with the following details:

KEY	VALUE	DESCRIPTION
Key	Value	Description

The response body is displayed in the 'Body' tab, showing a JSON object with the following details:

```
{ "account_no": "11118", "account_type": "savings", "name": "Isha Agarwal", "contact": "7845964100", "balance": 8000 }
```

## Testing of Angular And Swagger API

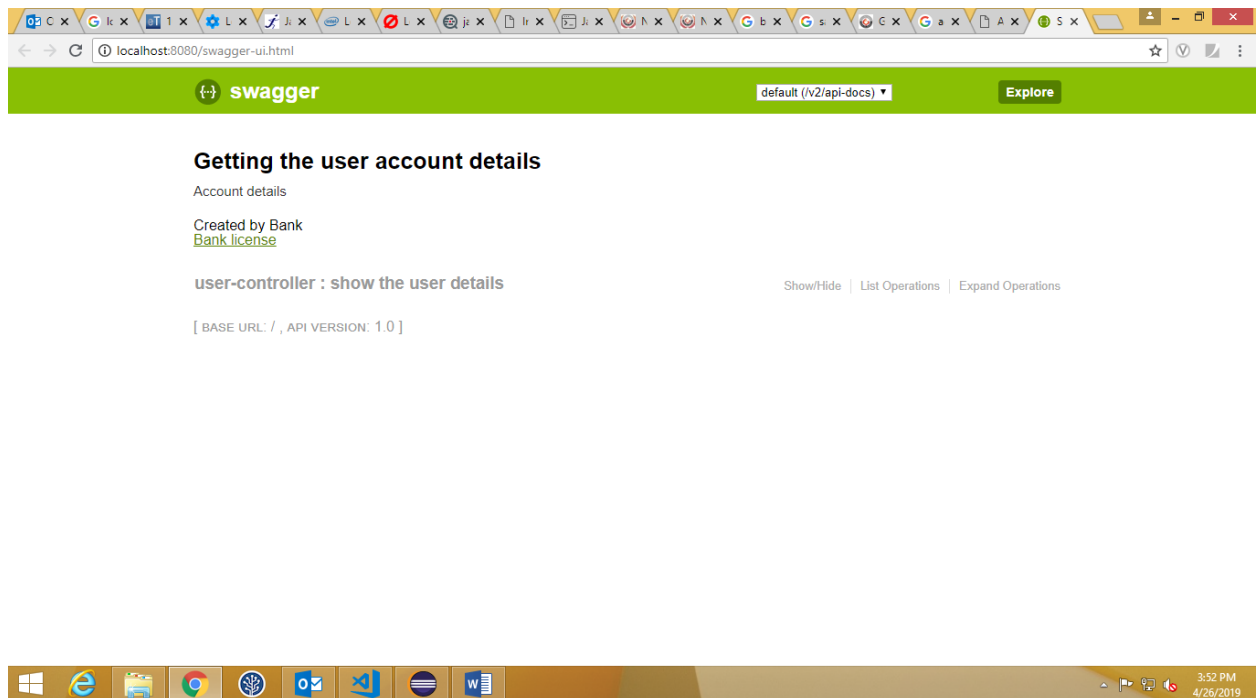
We have details of two customers in our API. One can either check their account details or add a new account if the user is not previously registered. The customer details we have are:

1. Account Holder Name: XYZ  
Account Number: 123456789100  
Account Password: ABC123
2. Account Holder Name: INFOSCIION  
Account Number: 112019202000  
Account Password: MYS123

## 8.1 White Box Testing

Since the valid inputs are known to the tester in White Box Testing, after successful inputs we login through the portal and reach the Swagger UI.

On providing the exact details as of user 1, we get the following screen:



**Fig 8.1 Swagger UI after successful login into the portal**

## 8.2 Black Box Testing

Since, the valid inputs aren't known to the tester, we deliberately try to provide false inputs to find out the outputs.

On providing the input as:

Account Holder Name: a2 (Not completely a alphabetic string)

Account Number: 125 (Not a 12 digit number)

Account Password: M (String size less than 8)

We get the following error messages respectively as shown in the figure:

The screenshot shows a web browser window with the address bar displaying 'localhost:4200/login'. The page title is 'Online Bank Application' and there is a link 'Check your account details'. The main content area is a light blue box with the heading 'Fill your Account details'. It contains three input fields: 'Account Number' with the value '125', 'Account Holder name' with the value 'a2', and 'Account Password' with a single dot. Below each input field is a red error message. The error message for the Account Number is 'Account number should be number and of length 12.' The error message for the Account Holder name is 'Account holder name should be of text type.' The error message for the Account Password is 'Account password is mandatory and min length is 8.' The background of the login form is a dark blue grid with glowing lines and dots. The Windows taskbar is visible at the bottom of the screen.

Online Bank Application [Check your account details](#)

Fill your Account details

Account Number

125

We'll never share your account number with anyone else.

Account number should be number and of length 12.

Account Holder name

a2

Account holder name should be of text type.

Account Password

•


Account password is mandatory and min length is 8.

**Fig 8.2 Error messages after invalid inputs in the login portal**

## CHAPTER 9

### Output Screens

#### 9.1 Eureka server

HOME LAST 1000 SINCE STARTUP

### System Status

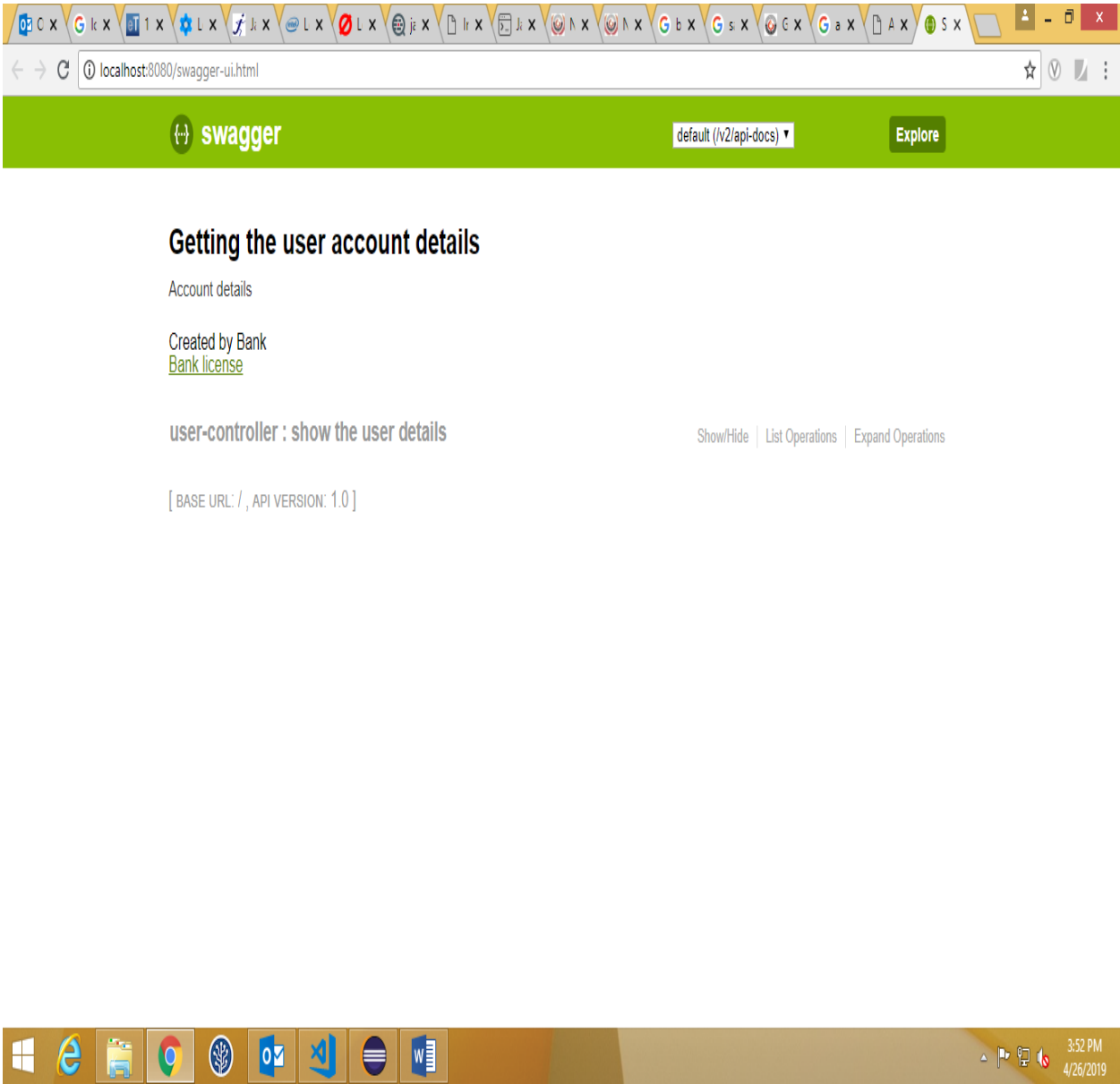
Environment	test	Current time	2019-04-30T13:29:21 +0530
Data center	default	Uptime	00:02
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	2

### DS Replicas

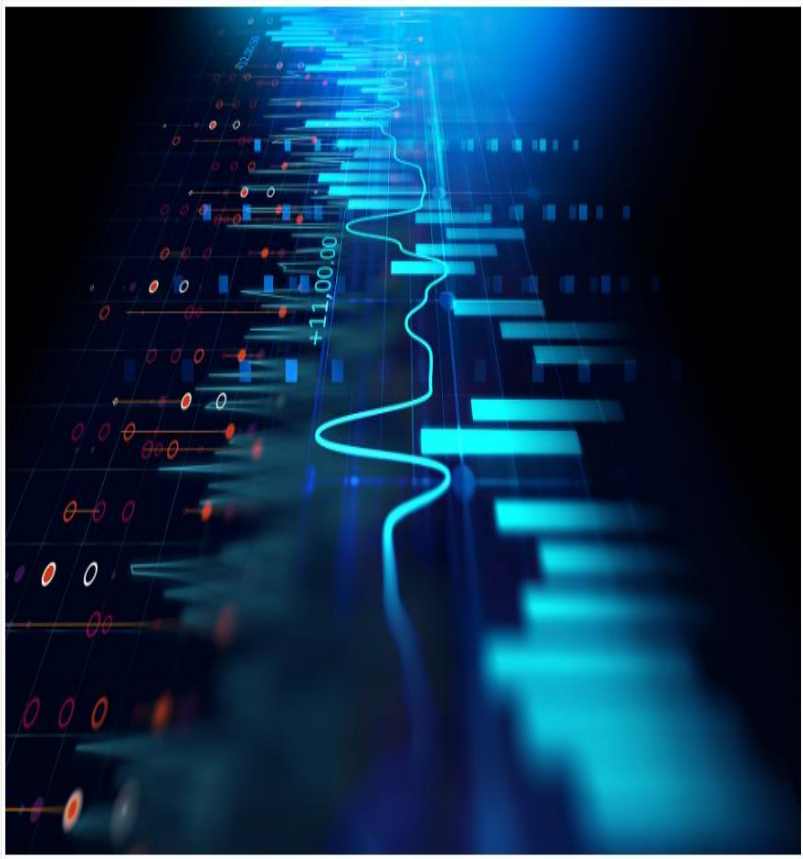
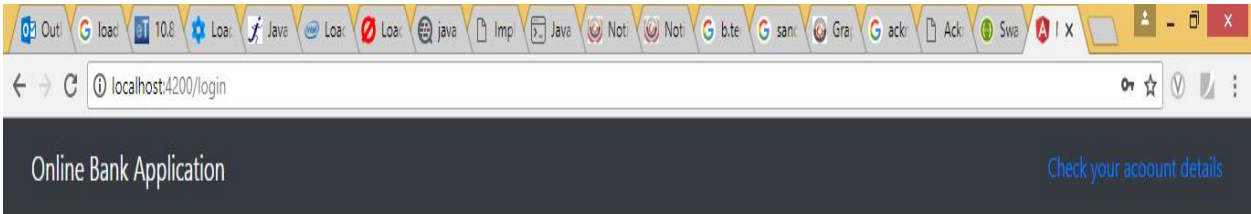
### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNT-MICROSERVICE-CONSUMER	n/a (1)	(1)	UP (1) - PUNITP383716D.ad.infosys.com:account-microservice-consumer:2222
ACCOUNT-MICROSERVICE-PRODUCER	n/a (1)	(1)	UP (1) - PUNITP383716D.ad.infosys.com:account-microservice-producer

9.2 Swagger UI



9.3 User Interface



Fill your Account details

Account Number

We'll never share your account number with anyone else.

Account number should be number and of length 12.

Account Holder name

Account holder name should be of text type.

Account Password

Account password is mandatory and min length is 8.





## CHAPTER 10

### Conclusion

Today bank regulators are looking to drive better deals for the clients by generating more competition by innovating customer information sharing, transaction initiation, and payment mechanisms through open APIs. With an open API economy, banks get an option to introduce their customers to new products and services through collation among the business units within a bank, across industries, and between banks and other complementary sectors of the economy, especially data businesses and technology.

Microservice splits monolithic applications into a set of services that communicate with each other through open APIs. Each service acts out one particular function pretty well; the service and its API are a product that is detectable, well-defined and carefully maintained. These self-contained services are then constituted as needed so as to deliver complex functionality, even if they are deployed independently of each other. Services can calibrate independently as well, making the software flexible at runtime. And if a single service folds, it doesn't mean it will bring down the entire system because of the flexibility built into microservice-based digital banking solutions.

It has come as a blessing for the banks because this approach will provide favorable circumstances to deliver value at subsequent intervals and not wait for a massive technology shift to happen. Open API banking combined with microservices-based architecture will define the success in banking space in the near future, where banks would be in a better position to accelerate time to market in the real sense.

## **CHAPTER 11**

### **Future Recommendations**

Open Banking is one of the most exciting developments in financial services for years, with the potential to revolutionize how consumers engage with financial products and services. The immediate future will likely be a deeper integration of microservice offerings in Google Cloud, AWS, and Azure because it can help them with their bottom line and make their resource usage in a more predictable manner. I also see this as taking a large bite out of commercial virtualization software, and maybe even taking them out of business since containers offer much of the same end-result and we can use them on free and open source software. They also require less training to become proficient in that virtual machine deployments and it's easier to script against.

## BIBLIOGRAPHY

- [1]The Financial Brand. What is Open Banking and Why does it matter [Online].  
Available:<https://thefinancialbrand.com/58913/open-banking-standard-api-regulation-fintech/>
- [2]w3schools. Java Introduction [Online]. Available:  
[https://www.w3schools.com/java/java\\_intro.asp](https://www.w3schools.com/java/java_intro.asp)
- [3]Wikipedia. Eclipse(software) [Online]. Available:  
[https://en.wikipedia.org/wiki/Eclipse\\_\(software\)](https://en.wikipedia.org/wiki/Eclipse_(software))
- [4]Tutorials Point. SQL-RDBMS Concepts [Online]. Available:  
<https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm>
- [5]Guru99. MySQL Workbench Tutorial [Online]. Available:  
<https://www.guru99.com/introduction-to-mysql-workbench.html>
- [6] Mahozad. (2018, Jun, 22). What exactly is Spring Framework for [Online].  
Available:<https://stackoverflow.com/questions/1061717/what-exactly-is-spring-framework-for>
- [7]Tutorials Point. Maven-Overview [Online]. Available:  
[https://www.tutorialspoint.com/maven/maven\\_overview.htm](https://www.tutorialspoint.com/maven/maven_overview.htm)
- [8]Tutorials Point. RESTful Web Services [Online]. Available:  
<https://www.tutorialspoint.com/restful/>
- [9]Tutorials Point. Microservice Architecture - Useful Resources [Online].  
Available:[https://www.tutorialspoint.com/microservice\\_architecture/microservice\\_architecture\\_useful\\_resources.htm](https://www.tutorialspoint.com/microservice_architecture/microservice_architecture_useful_resources.htm)
- [10]Javatpoint. JPA tutorial [Online]. Available: <https://www.javatpoint.com/jpa-tutorial>
- [11]Javatpoint. Spring Tutorial [Online]. Available: <https://www.javatpoint.com/spring-tutorial>
- [12]Tutorials Point. Spring Boot-Eureka Server [Online]. Available:  
[https://www.tutorialspoint.com/spring\\_boot/spring\\_boot\\_eureka\\_server.htm](https://www.tutorialspoint.com/spring_boot/spring_boot_eureka_server.htm)
- [13]Tutorials Point. Jenkins Tutorial [Online]. Available:  
<https://www.tutorialspoint.com/jenkins/>
- [14]Umesh. (2018, Dec, 18). Introduction to Feign [Online]. Available:  
<https://www.javadevjournal.com/spring/feign/>

- [15]HowToDoInJava. Spring cloud ribbo with Java [Online]. Available: <https://howtodoinjava.com/spring-cloud/spring-boot-ribbon-eureka/>
- [16]Springboottutorial. (2017, Dec, 30). Spring Boot and Swagger - Documenting RESTful Services [Online]. Available: <https://www.springboottutorial.com/spring-boot-swagger-documentation-for-rest-services>



