image001

**JAVA FULL STACK**

**SYNOPSIS REPORT**

**Submitted by:**

**PREKSHA MENDIRATTA**

**1591018095**

**Under the Supervision of**

**MR. TANMAYA ACHARYA**

Senior Consultant - L&D, Capgemini, Bangalore

ELECTRONICS AND COMMUNICATION ENGINEERING

MANIPAL UNIVERSITY JAIPUR  
JAIPUR-303007  
RAJASTHAN, INDIA

March 2019

PROJECT SUMMARY

|  |  |
| --- | --- |
| *Organization Details* | |
| Company Name | Capgemini Pvt. Ltd. |
| Full postal address with pin code | Capgemini Technology Services  Plot No-1, IT Park, 115/32&35,  Nanakramguda,  Gachibowli,  Hyderabad - 500032 |
| Website address | https://www.capgemini.com |
|  |  |
| *Supervisor Details* | |
| Supervisor Name | Mr. Vikas Mohindra |
| Designation | Senior Consultant |
| Phone Number (M) | +91 9912121321 |
| Email address | vikas.mohindra@capgemini.com |
|  |  |
| *Project Details* | |
| Project Title | XYZ Banking Application |
| Project Duration *(in weeks)* | 12 weeks |
| Date of project commencement | 16th January 2019 |
| Supervisor Sign with Seal |  |

1. **OBJECTIVE OF WORK**

Digitization is the conversion of data into a digital format with the adoption of technology. Adoption of [digitalization is very important especially for the](https://www.hcltech.com/digital-analytics-services/digital-transformation-consulting)[banking](https://www.hcltech.com/financial-services/retail-corporate-banking) sector. Our major objective was to create payment wallet application for XYZ bank using the following technologies:

1. JDBC implementation
2. Core Spring
3. JPA
4. Spring JPA Integration
5. Spring Data
6. Application as Restful web service
7. Spring Boot

By embracing digitalization, banks can provide enhanced customer services like, they will be able to park their money in the wallet, check the balance, deposit money, withdraw money, transfer funds and print all their transactions etc. The need of the hour is to be able to have round-the-clock access to the services offered by the bank with the help of the developed application.

**2. MOTIVATION**

Core banking is all about knowing the customers’ need and provide them with the right services at the right time through the right channel 24\*7 days a week. As inevitable banks and their services are in our lives, technology and adaptation to these technologies are also inevitable from the perspective of development and digitalization for the banks. Digitization is not an option for banking industry, rather it is imperative. Mobile banking is increasing at a fast pace more than online banking. It is observed that the customer experience is improved and the technology is non-discriminatory. Thus banks always try to adopt latest technologies to enhance customer experience.

It allows the client of private banks to carry out their day to day banking transactions. This application will enable them to park their money in the wallet, check their balance, deposit their money, withdraw their money, transfer their funds and print all their transactions.

1. **TARGET SPECIFICATIONS**

After the completion of the project, the user will be able to:

* Create an account in XYZ Bank.
* Check his available bank balance.
* Deposit/Withdraw a certain amount.
* Inter banking fund transfers.
* Print all the transactions in the passbook.

**4. FUNCTIONAL PARTITIONING OF PROJECT**

* 1. **- Identification and Implementation:**

The project began with the implementation in Core Java which involved basic identification of Java classes, interfaces, methods and their attributes followed by their implementation and inter linking using layered architecture.

* 1. **- Exception Handling:**

After creating of the basic outline of the working application, exceptions were handled through proper exception handling mechanism. Exception handling is done in order to prevent the abrupt termination of the program. The following mentioned are some of the exceptions that were handled by the modelled application:

* If the entered withdrawal amount is greater than the balance available in the account, the error message should be printed saying that “The available balance is insufficient”.
* If there is an attempt to make a transaction (deposit, withdraw, or transfer the funds) on an account that has not been created yet, the application should throw a message stating “The account does not exist”.

If the entered amount for deposit is zero or negative, the application throws a message stating “The amount entered is invalid”.

**4.3 - Application Testing:**

The next major objective was to verify if the application delivers the desired outputs before deployment of the application. A certain set of test cases were implemented in the persistence layer.

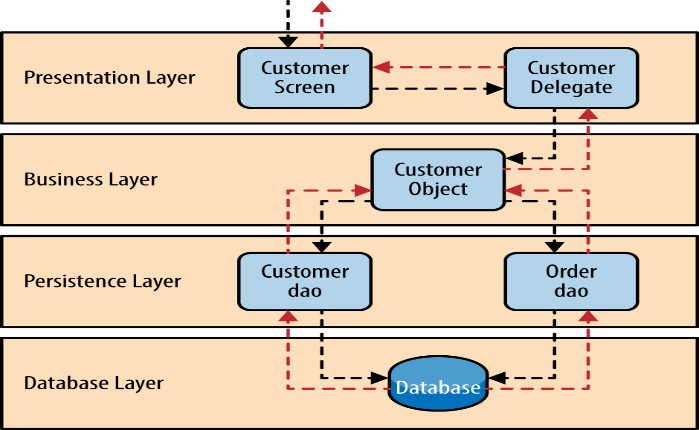
**5. METHODOLOGY**

The creation of the project can be categorized into three major steps: defining the layered architecture, implementing the collection framework, handling the exceptions and finally testing them with all the user defined test cases.

Initially, some customers and their details are added to the bank (Collection framework) and further modifications can be made using the menu driven program. New customers will have an option of creating their account using the (1) menu, the available balance in an account can be viewed by the (2) menu, (3) menu can be used to deposit a certain amount of money in the account number mentioned by the customer, similarly, withdrawal can be done using the (4) menu, fund transfers can be done using the (5) menu and all the transactions done under a single account number can be viewed by the (6) menu and driving straight out of the application can be done using the (7) menu.

**5.1 – Layered Architecture:**

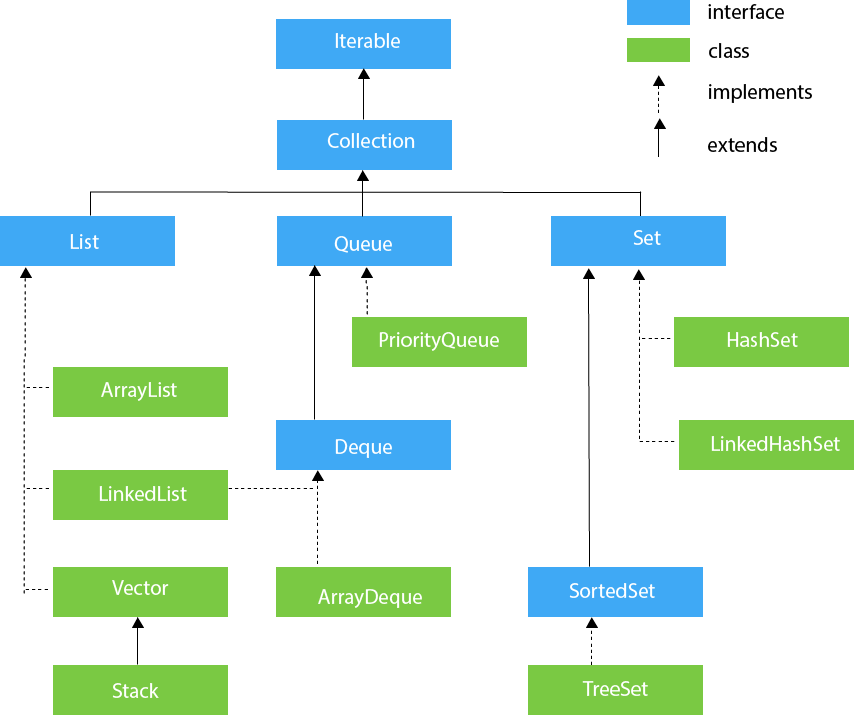
Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application. Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database (Figure 5.1). In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas large, complex business applications may contain more layers. Each layer of the layered architecture has a specific role within the application. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data, and pass that information up to the presentation layer.

****

**Figure 5. 1: Layered Architecture**

**5.2 – Collection Framework:**

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, inserting, manipulating, and deleting. Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, and Queue) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet). The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

****

**Figure 5. 2 Collection Hierarchy**

**5.3 – Exception Handling and Application Testing:**

Java being an object oriented programming language, whenever an error occurs while executing a statement, creates an **exception object** and then the normal flow of the program halts and Java Run-Time Environment (JRE) tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc. When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called **“throwing the exception”**.

Java provides specific keywords for exception handling purposes, we will look after them first and then we will write a simple program showing how to use them for exception handling.

1. **throw** – We know that if any exception occurs, an exception object is getting created and then Java runtime starts processing to handle them. Sometime we might want to generate exception explicitly in our code, for example in a user authentication program we should throw exception to client if the password is null. ‘**throw’** keyword is used to throw exception to the runtime to handle it.
2. **throws** – When we are throwing any exception in a method and not handling it, then we need to use **throws** keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to its caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with main() method also.
3. **try-catch** – We use try-catch block for exception handling in our code. ‘try’ is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. ‘catch’ block requires a parameter that should be of type Exception.
4. **finally** – finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. ‘finally’ block gets executed always, whether exception occurred or not.
5. **TOOLS REQUIRED**

* Spring Tool Suite (STS)
* .jar file (Java Archive)
* All necessary application program interfaces (API)

1. **WORK SCHEDULE (MONTH WISE)**

## (a) January 2019:

The Core Java module was undertaken which covered topics like OOPS (Abstraction, Inheritance and polymorphism), collections, Multithreading. JUnit Testing, Layered Architecture.

**(b) February 2019:**

The given project (XYZ Bank Application) was implemented by following the layered architecture which was taught in the Core Java module which consisted of persistence, presentation, service and util layer. The application testing was performed using the JUnit test cases.

**(c) March 2019:**

The Spring module was undertaken which consisted of:

* Database and SQL
* HTML, CSS, Bootstrap and JavaScript
* JPA with Hibernate
* Servlet and JSP
* Spring MVC
* Spring Boot
* Spring Restful

The same project will be implemented using the Spring Framework.

**(d) April 2019:**

A module on Behavior-driven development (BDD) will be undertaken.