

## **Software Engineering Lab Experiment No. 1**

**Aim:** Identify the tool for requirements analysis and installation and setup of the software.

### **Objectives:**

1. To install and set up Figma for software requirements analysis.
2. To gain a theoretical understanding of requirements analysis in software development.
3. To perform requirements analysis using Figma.
4. To document the entire process and capture output screen shots for analysis and reporting.

### **Requirements:**

1. Computer with internet access.
2. A Figma account.
3. Sample software project or problem statement for requirements analysis.
4. Word processing software for creating the lab report.

### **Concept:**

Requirements analysis is a crucial phase in software development that involves identifying, documenting, and understanding the needs and constraints of a software project. The primary objectives of requirements analysis include:

- Understanding user needs and expectations.
- Identifying functional and non-functional requirements.
- Documenting clear and unambiguous requirements.
- Analysing, prioritizing, and validating requirements.
- Ensuring that the software solution aligns with the client's objectives.

The process of requirements analysis typically involves steps like gathering user stories, creating use cases, defining system requirements, and producing detailed documentation.

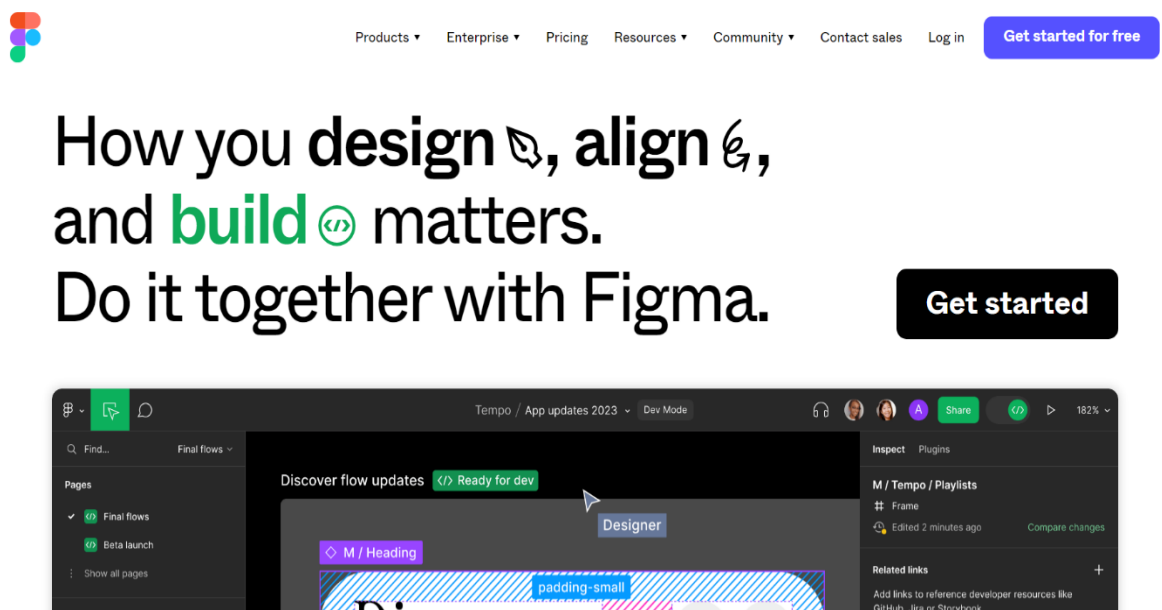
### **Tools:**

- **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes. It enables real-time collaboration and can be accessed from anywhere and on any device.

## Steps:

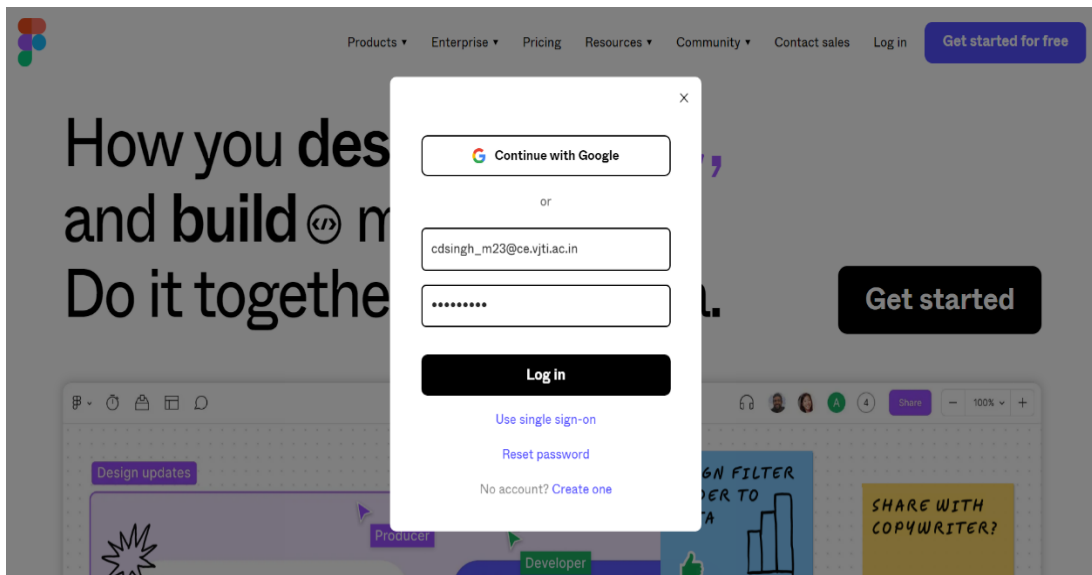
### 1. Identifying the tool :- Key reasons to choose Figma:

- **Real-time Collaboration:** Multiple team members can work on the same project simultaneously.
- **Cross-Platform Compatibility:** Accessible on the web and desktop for macOS and Windows.
- **Version Control:** Automatically saves and tracks design file versions.
- **Prototyping and Interaction Design:** Supports interactive and realistic prototypes.
- **Design Handoff:** Simplifies design-to-development communication.
- **Component Libraries:** Facilitates the creation and management of design components.
- **Third-Party Integrations:** Offers integration with various design and project management tools.
- **Design Systems:** Tools for creating and maintaining design consistency.
- **Community and Resources:** Access to a large user community and design resources.
- **Scalability:** Suitable for small and large design projects.
- **Security and Permissions:** Provides security features and user permission settings.
- **Regular Updates:** Consistently updated with new features and improvements.

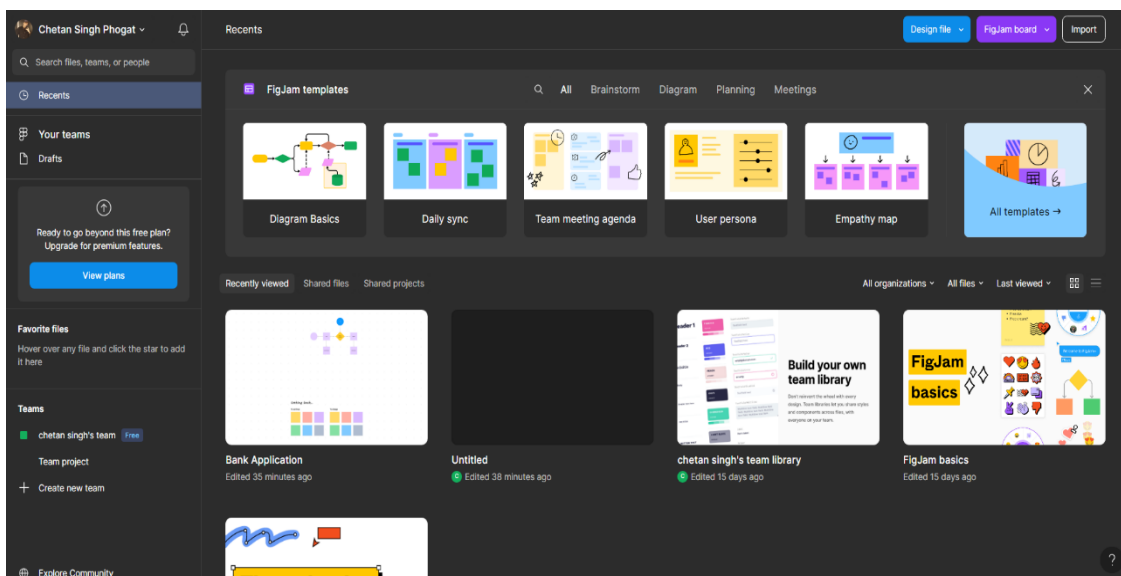


## 2. Setting Up Figma:

- Go to the Figma website (<https://www.figma.com/>) and sign up for an account if you don't already have one.
- Install the Figma desktop application (if required).
- Explore the Figma interface and understand its features for design and collaboration.



## 3. Home Page after login



**Conclusion:**

This lab report aimed to provide a practical understanding of setting up Figma for software design and conducting requirements analysis using the tool. Figma is a valuable platform for creating, sharing, and collaborating on design and requirements-related tasks. The report also emphasized the importance of effective requirements analysis in the software development process, providing a theoretical foundation for this critical phase.

**References:**

- Figma - Collaborative interface design tool (<https://www.figma.com/>)
- Software Engineering: A Practitioner's Approach by Roger S. Pressman (For further reading on requirements analysis in software engineering).

## **Software Engineering Lab Experiment No. 2**

**Aim:** Creating Data flow diagram and use case based on any project.

### **Objectives:**

1. UML class diagrams help in visualizing the key classes, their attributes, and relationships within the system.
2. Understanding Class Relationships.
3. Making use case diagrams to help and identify and document user needs.
4. To outlines the key actions that the customer can perform when using the system.

### **Requirements:**

1. Computer with internet access.
2. A Figma account.
3. Sample software project or problem statement for requirements analysis.
4. Word processing software for creating the lab report.

### **Concept:**

#### **1. UML Diagram**

- A UML (Unified Modeling Language) diagram is a visual representation of a system's structure and behaviour.
- It uses standardized symbols and notations to depict classes, objects, relationships, interactions, and more.
- UML diagrams help to model, document, and communicate complex software and system designs, making them easier to understand for stakeholders and developers.

#### **2. Use Case Diagram**

- A use case diagram is a specific type of UML diagram that focuses on the functional requirements of a system.
- It represents how users (actors) interact with a system by defining use cases (functionalities or services) and their relationships.
- Use case diagrams provide a high-level view of the system's behaviour and helps us capturing user requirements, system functionality, and the roles of different actors in the system.

### **Tools:**

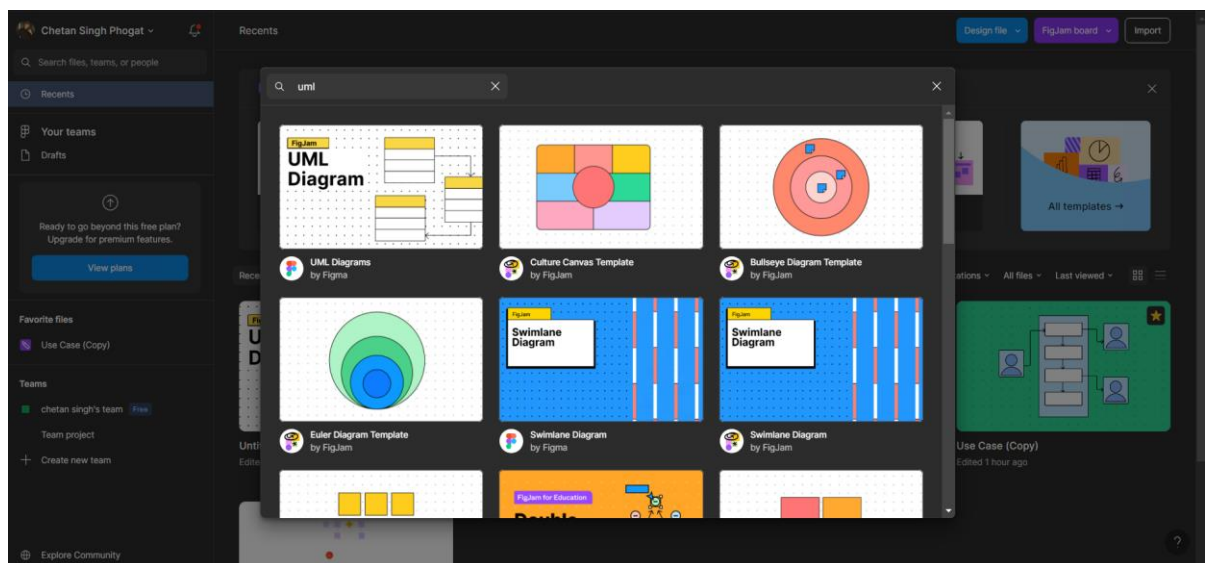
- **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes. It enables real-time collaboration and can be accessed from anywhere and on any device.

## About Project

**Bank Application:** It is an online banking application which allows user to create a bank account and login. This application has different modules like login, account details, transactions and Loan. User can apply for loan using this application, upload documents, check approval status etc.

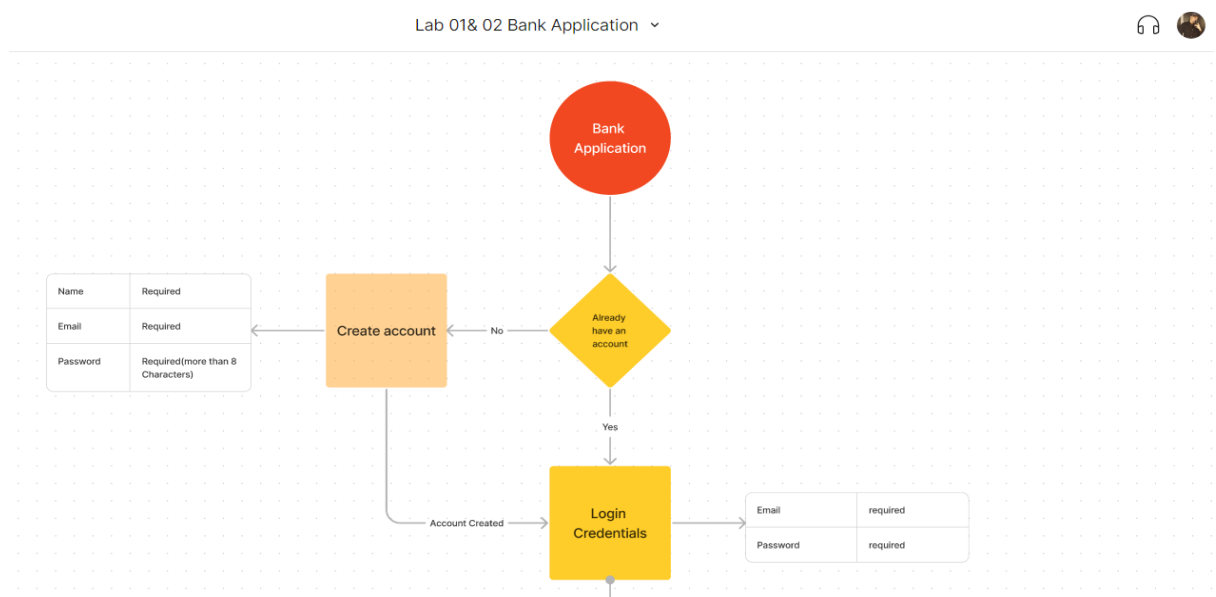
### Steps: *General steps for creating a UML diagram:*

1. **Identify the System or Process:**
  - Clearly define the system, process, or concept that you want to model with the UML diagram.
2. **Identify the UML Diagram Type:**
  - Choose the appropriate type of UML diagram based on what you want to represent.
3. **Gather Requirements and Information:**
  - Collect all relevant information about the system, including classes, objects, relationships, behaviours, and interactions.
4. **Select UML Diagramming Tool:**
  - **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes.



**5. Add Elements and Relationships:**

- Populate the diagram with elements and define their relationships. For example:  
In a DFD diagram explain flow of project according to your project.

**Steps: General steps for creating a Use Case:****1. Define the Purpose and Scope**

- Use case for an "Account Management" class for user management or authentication.
- This use case describes how a user can change their password.

**2. Identify Actors**

- **User:** Represents individuals who have user accounts and need to manage their passwords.

**3. Identify Use Cases**

- **Change Password:** The user can change their account password.
- **Reset Password:** The user can request a password reset their current password.
- **Unlock Account:** The user can request to unlock their account if it gets locked after multiple failed login attempts.

#### 4. Create the Use Case Diagram

This use case describes how a user can change their password.

**Use Case:** Change Password

**Primary Actor:** User

**Precondition:** The User is authenticated and has access to their account.

**Main Success Scenario:**

- 1) The User selects the "Change Password" option in their account settings.
- 2) The system prompts the User to enter their current password and the new password.
- 3) The User enters the required information.
- 4) If the validation is successful, the system updates the User's password.
- 5) The system confirms the successful password change to the User.

**Exception Scenarios:**

- 1) Invalid Current Password
- 2) Weak New Password
- 3) Technical Failure (Database connection failure)

This use case describes the interaction between the User and the Account class when changing their password. It covers the main successful path as well as potential exception scenarios to ensure a comprehensive understanding of how this feature works.

#### **Conclusion:**

- This lab report aimed to provide a practical understanding of UML Data flow diagram provide a visual representation.
- The use case diagram outlines the main functionalities of the system as seen from the customer's perspective.

#### **References:**

- Figma - Collaborative interface design tool (<https://www.figma.com/>)
- Software Engineering: A Practitioner's Approach by Roger S. Pressman (For further reading on requirements analysis in software engineering).



## **Software Engineering Lab Experiment No. 3**

**Aim:** Create an SRS for the project

**Objectives:**

1. Capture and Document User Needs.
2. Enhance Project Transparency.
3. Use the SRS to develop use case.
4. Use the information gathered in the SRS to create UML diagrams, such as class diagrams.

**Requirements:**

1. Computer with internet access.
2. A Figma account.
3. Sample software project or problem statement for requirements analysis.
4. Word processing software for creating the lab report & SRS document.

**Tools:**

- **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes. It enables real-time collaboration and can be accessed from anywhere and on any device.
- **Word document** for creating the lab report & SRS document.

## **Table of Contents**

### **1. INTRODUCTION**

#### **1.1 PURPOSE**

#### **1.2 SCOPE**

#### **1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS**

#### **1.4 REFERENCES**

#### **1.5 OVERVIEW**

### **2. GENERAL DESCRIPTION**

#### **2.1 PRODUCT PERSPECTIVE**

#### **2.2 ASSUMPTIONS AND DEPENDENCIES**

### **3. SPECIFIC REQUIREMENTS**

#### **3.1 FUNCTIONAL REQUIREMENTS**

##### **3.1.1 Account Creation**

##### **3.1.2 Deposit**

#### **3.2 USE CASES**

##### **3.2.1 Use Case #1: Account Registration**

##### **3.2.2 Use Case #2: Deposit Funds**

#### **3.3 CLASSES / OBJECTS**

##### **3.3.1 User**

##### **3.3.2 Account**

### **4. ANALYSIS MODELS**

#### **4.1 UNIFIED MODELING LANGUAGE (UML)**

### **5. USE CASE**

## **1. Introduction**

### **1.1. Purpose**

The purpose of this Software Requirements Specification (SRS) is to provide a detailed description of the Account Transaction System, outlining its features, functionality, and requirements.

### **1.2. Scope**

The Account Transaction System is designed to facilitate account management, allowing users to perform various banking transactions. This system will cover the core functionality of account creation, deposit, withdrawal, balance inquiry, and fund transfer.

### **1.3. Definitions, Acronyms, and Abbreviations**

- SRS: Software Requirements Specification
- PIN: Personal Identification Number

### **1.4. References**

- Used IEEE SRS template for creating SRS document.

### **1.5. Overview**

The Account Transaction System is a web-based application that enables customers to manage their bank accounts efficiently. Users can perform basic banking transactions from the comfort of their homes or on the go, ensuring the security and reliability of their financial operations.

## **2. General Description**

### **2.1 Product Perspective**

The Account Transaction System is a standalone application that interacts with a central banking database for account management. It is not dependent on any other external systems or applications.

## **2.2 Assumptions and Dependencies**

- Users have a valid account with the bank.
- Secure authentication mechanisms are in place, including username and PIN.
- The system relies on a stable internet connection for real-time transaction processing.

## **3. Specific Requirements**

### **3.1 Functional Requirements**

#### **3.1.1 Account Creation**

- Users can create new bank accounts.
- Users must provide their personal information, including name, address, and contact details.
- The system generates a unique account number and associates it with the user.

#### **3.1.2 Deposit**

- Users can deposit money into their accounts.
- Deposits can be made in various forms, including cash and electronic transfers.
- The system updates the account balance accordingly.

### **3.2 Use Cases**

#### **3.2.1 Use Case #1: Account Registration**

- User initiates the account creation process.
- User provides personal details.
- System generates an account number and assigns it to the user.

### 3.2.2 Use Case #2: Deposit Funds

- User selects the deposit option.
- User specifies the amount and deposit method.
- System updates the account balance.

## 3.3 Classes / Objects

### 3.3.1 User

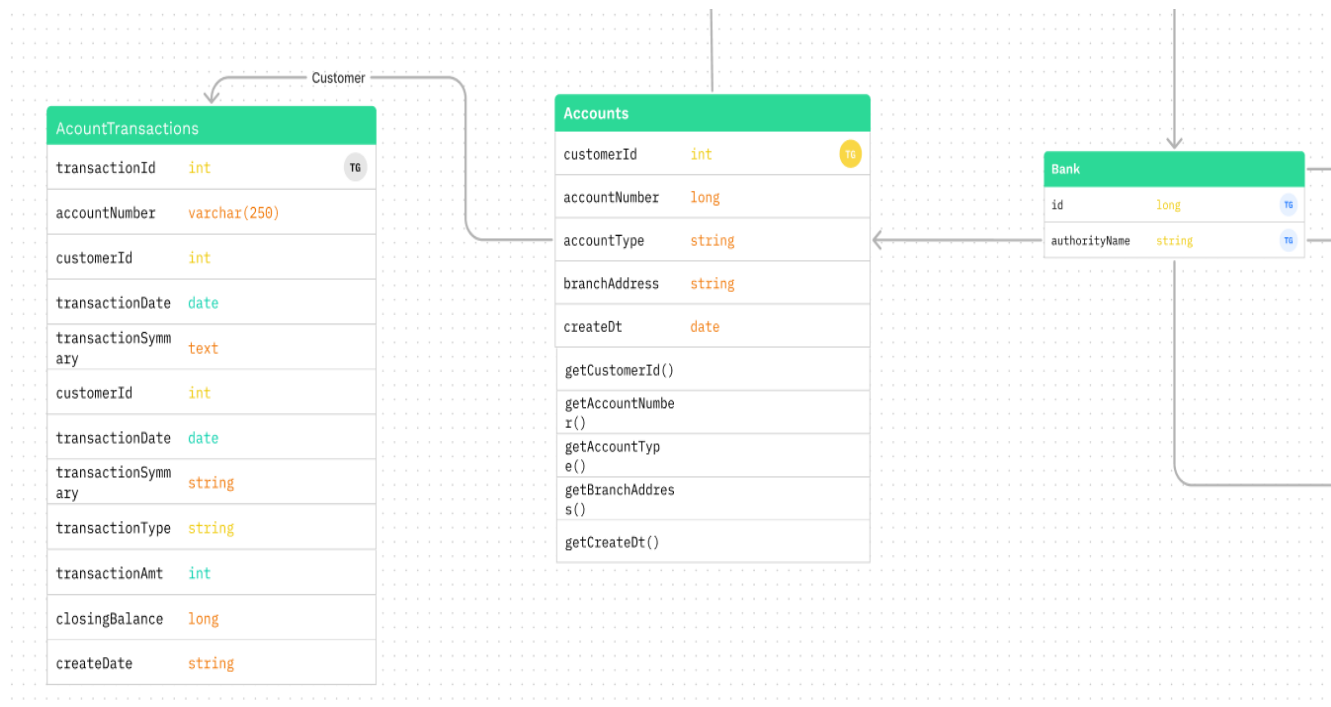
- **Attributes:** Name, Address, Contact Details, Account Number
- **Methods:** CreateAccount(), DepositFunds(), WithdrawFunds()

### 3.3.2 Account

- **Attributes:** Account Number, Balance
- **Methods:** GetBalance(), UpdateBalance()

## 4. Analysis Models

### 4.1 Unified Modeling Language (UML)



## **5. Use Case**

**Use Case:** Account Transaction

**Primary Actor:** User

**Precondition:** The User is authenticated and has access to their account.

**Main Success Scenario:**

1. The User selects the "Account Transaction" option in their account settings.
2. The system presents options for different types of account transactions (e.g., transfer funds, make a payment, view transaction history).
3. The User selects the desired account transaction type.
4. The system guides the User through the specific steps for the chosen transaction, which may include entering transaction details, recipient information, and amount.
5. The User enters the required information.
6. If the validation is successful, the system processes the account transaction.
7. The system confirms the successful completion of the transaction to the User.

**Exception Scenarios:**

1. Invalid Transaction Details: If the User provides incomplete or incorrect transaction details.
2. Insufficient Funds: If the User's account balance is insufficient for the selected transaction.
3. Technical Failure (e.g., Network Error): If there is a technical issue that prevents the transaction from being completed.
4. Transaction Limit Exceeded: If the User exceeds their transaction limit.
5. Transaction Cancelled by User: If the User decides to cancel the transaction during the process.

## **Software Engineering Lab Experiment No. 4**

**Aim:** Create class diagram for the project

### **Objectives:**

1. The objective of this lab experiment is to create a class diagram for the software project.
2. Providing a visual representation of the structure and relationships between classes.
3. This class diagram will serve as a reference for developers and other stakeholders to understand the architecture of the system.

### **Requirements:**

1. Computer with internet access.
2. A Figma account.
3. Sample software project or problem statement for requirements analysis.
4. Word processing software for creating the lab report.

### **Concept:**

#### **1) Class Diagram**

- A class diagram is a type of UML (Unified Modeling Language) diagram that provides a visual representation of the structure and relationships within a system or software application.

### **Tools:**

- **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes. It enables real-time collaboration and can be accessed from anywhere and on any device.

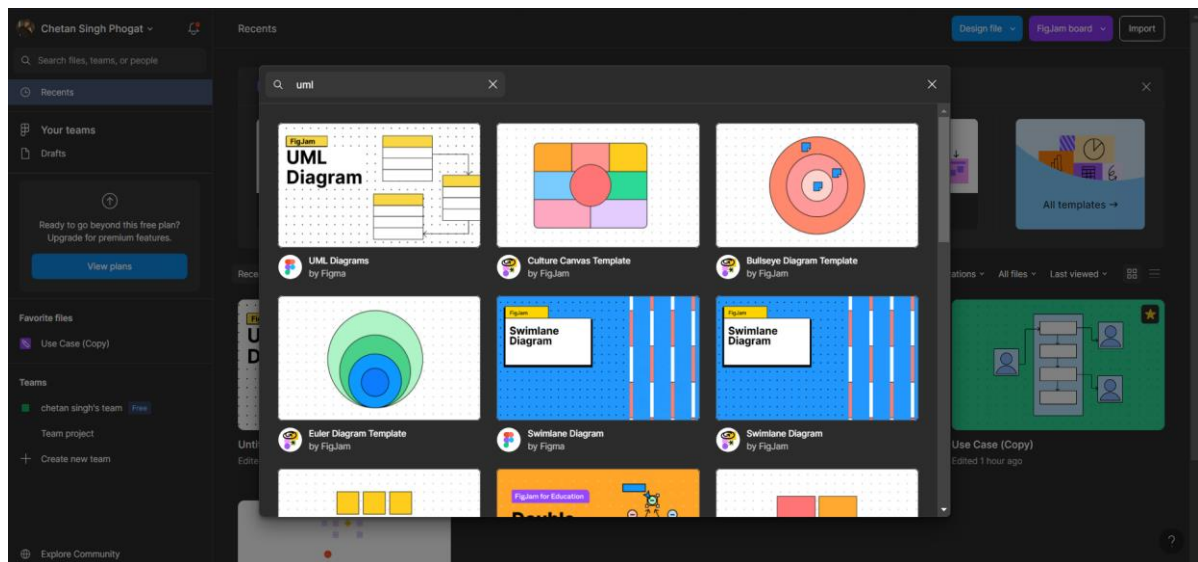
### **Steps:** *General steps for creating a Class diagram:*

1. **Identify Classes:** Begin by identifying the key classes in your project. These classes represent the major components, entities, and functionalities in your software system.
2. **Define Attributes and Methods:** These attributes and methods represent the state and behaviour of each class.

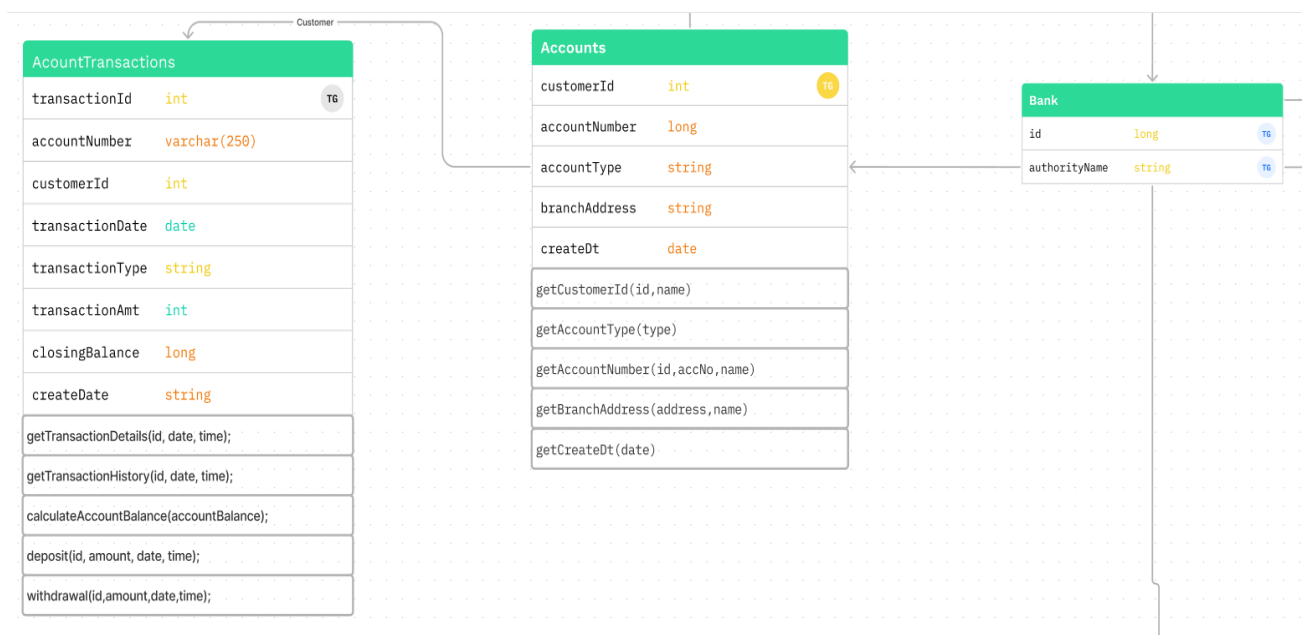
**3. Identify Relationships:** Determine the relationships between classes. Common relationships include associations, aggregations, compositions, and inheritances. Understand how classes are connected and interact with each other.

**4. Select UML Diagramming Tool:**

**Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes.



**5. Create the Class Diagram:** Use a UML Modeling tool or drawing software to create the class diagram. Place classes, attributes, methods, and relationships on the diagram. Use appropriate symbols and notations to represent these elements.





#### **6. Review and Validate:**

Review the class diagram with the project team and other stakeholders. Ensure that it accurately reflects the project's design and that all relationships and dependencies are correctly represented.

#### **Conclusion:**

- The class diagram provides a clear visual representation of classes, their attributes, methods, and relationships. By following the outlined steps, we have successfully documented the system's design, allowing developers and stakeholders to have a common understanding of the project's structure.

#### **References:**

- Figma - Collaborative interface design tool (<https://www.figma.com/>)
- Software Engineering: A Practitioner's Approach by Roger S. Pressman (For further reading on requirements analysis in software engineering).

## **Software Engineering Lab Experiment No. 5**

**Aim:** Create sequence diagram for the project

### **Objectives:**

1. The objective of this lab experiment is to create a sequence diagram for the software project.
2. The sequence diagram will serve as a visual representation of how different elements collaborate to specific tasks or scenarios in the software application.

### **Requirements:**

1. Computer with internet access.
2. A Figma account.
3. Sample software project or problem statement for requirements analysis.
4. Word processing software for creating the lab report.

### **Concept:**

- **Sequence Diagram** - A sequence diagram is another type of UML (Unified Modeling Language) diagram used to visualize and represent the interactions and flow of messages between various components, objects, or actors in a system over time.

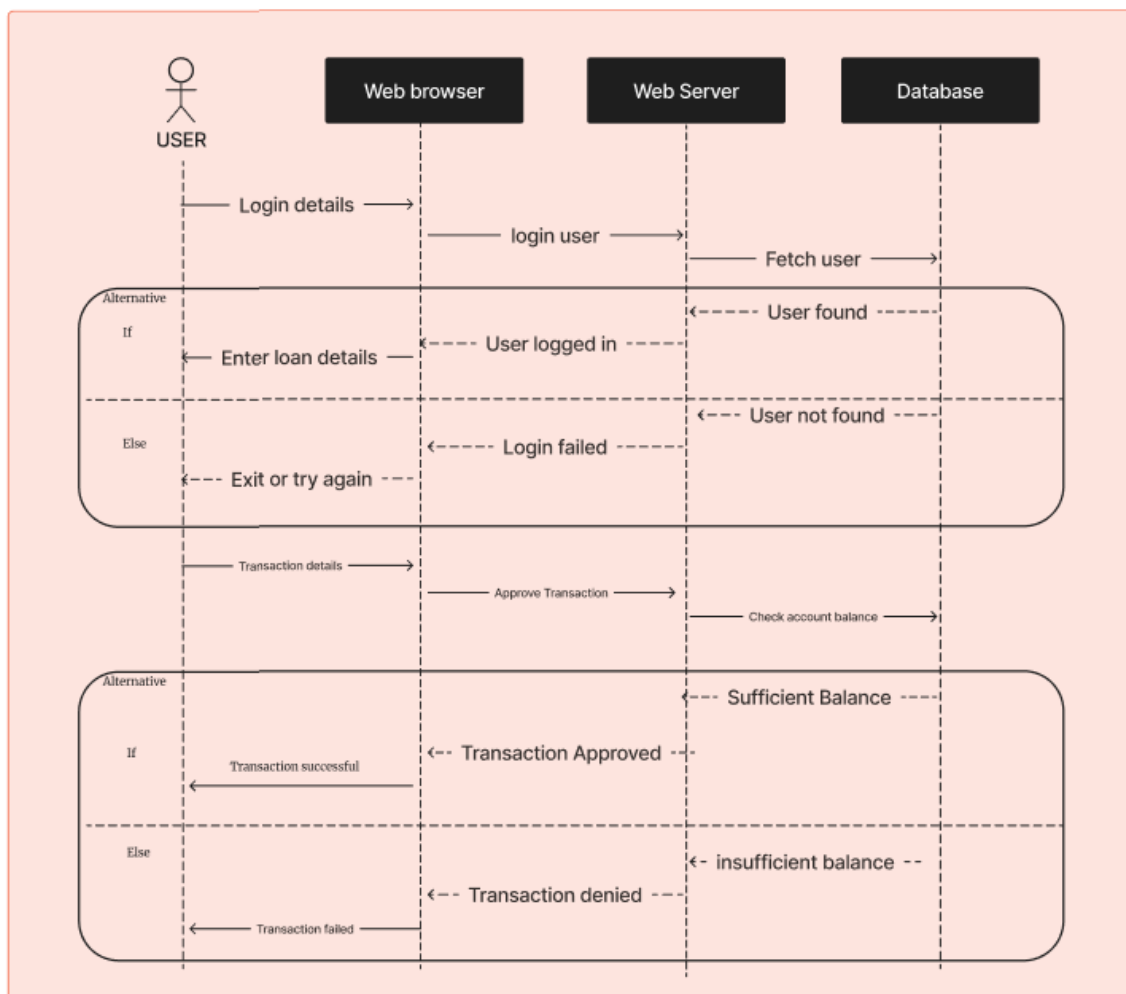
### **Tools:**

- **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes. It enables real-time collaboration and can be accessed from anywhere and on any device.

### **Steps:** *General steps for creating a Sequence diagram:*

- a. **Identify the Scenario:** Begin by identifying the specific scenario or use case for which you want to create a sequence diagram. This scenario should represent a typical interaction within the system.

- b. **Identify Participants:** Identify the key participants in the scenario. These can be objects, classes, actors, or any other relevant entities that are involved in the interaction.
- c. **Define Lifelines:** Create lifelines for each participant on the diagram. Lifelines represent the existence of participants over time.
- d. **Sequence of Messages:** Define the sequence of messages exchanged between participants. Messages represent interactions and communication between participants.



### **Conclusion:**

- The sequence diagram created for this project serves as a valuable tool for understanding the dynamic behaviour and interactions within the software system.
- It provides a clear visual representation of how different participants collaborate to achieve specific tasks or scenarios.

### **References:**

- Figma - Collaborative interface design tool (<https://www.figma.com/>)
- Software Engineering: A Practitioner's Approach by Roger S. Pressman  
(For further reading on requirements analysis in software engineering).

## **Software Engineering Lab Experiment No. 6**

**Aim:** To prepare a software test plan for the application.

**Objective:** The objective of this experiment is to create a test plan for the application to ensure its functionality, security, and performance meet the specified requirements.

### **Requirements:**

1. Computer with internet access.
2. Sample software project or problem statement for requirements analysis.
3. Word processing software for creating the lab report.

### **Concept:**

#### **Software test plan:**

A software test plan is a detailed document outlining the objectives, scope, resources, schedules, and approach to testing a software application. It describes the testing strategy, including test objectives, methodologies, test environments, entry and exit criteria, responsibilities, and risk assessment. This plan serves as a roadmap for the testing process, ensuring systematic and comprehensive testing of the software to identify and resolve any issues before its release.

### **Software Test Plan- Bank Application (Account Transactions)**

#### **Introduction:**

A Spring Boot bank application is a software solution built using the Spring Boot framework for developing Java-based enterprise applications. The application typically provides a set of features and functionalities related to banking operations, account management, transactions, and loans.

**Objective:**

The primary goal of the testing process is to identify, document, and address defects within the account transaction functionality. This encompasses a thorough examination of account creation, fund transfers, transaction history, and any other related processes. By validating and rectifying potential issues, the aim is to enhance user satisfaction and confidence in the account transaction process.

**Risk and Risk mitigation**

<b>Risk</b>	<b>Mitigation</b>
Inadequate Error Handling and Recovery	Implement robust error-handling mechanisms and conduct systematic testing for error scenarios. Use JUnit testing.
Incomplete Loan Processing:	Conduct thorough functional testing with tools like Selenium
Compatibility Issues with Various Browsers and Devices:	Implement cross-browser and cross-device testing using tools like Selenium or Cypress to ensure the Bank application's compatibility and functionality across various environments.
Inadequate Handling of Concurrent User Loads	Perform load testing using tools like JMeter or LoadRunner to simulate heavy user traffic and identify performance bottlenecks.

**Types of Testing:****Unit Testing:**

This phase involves breaking down the account transaction functionality into its individual units, thoroughly assessing their functionality in isolation. It includes tests for account creation, fund transfers, transaction history, and balance verification to ensure each unit works as intended.

**Black Box - Functional Testing:**

The primary focus here is on the external behavior of the account transaction section without considering the internal code structure. It encompasses the testing of user interactions, including creating accounts, transferring funds, and checking transaction history.

**Load Testing:**

This testing type evaluates the account transaction's performance under varying user loads. Simulating different levels of concurrent users, it ensures stability, scalability, and the ability to manage peak traffic without performance degradation or system crashes.

**Scope:**

Testing will focus on the core account transaction functionality, including its critical elements like creating accounts, fund transfers, and transaction history.

**Out of Scope:**

Testing will not extend to other features or areas of the application not directly related to account transactions, such as loan processing, customer information, admin login, ATM, or unrelated backend systems.

**Tools:**Unit testing

**JUnit** is a widely used testing framework for Java applications, including Spring Boot projects. It provides annotations to identify test methods, asserts for testing expected results, and other features to facilitate the testing process.

Black-box testing

Selenium: Widely used for automated testing of web applications, Selenium is versatile and supports multiple programming languages. It enables the testing of web applications across various browsers and platforms.

Load Balancing

JMeter: While primarily used for load testing, JMeter can also perform functional and black-box testing by simulating heavy loads on servers, networks, or objects to assess their performance under varied conditions.

**Schedule:**

Unit Testing: Week 1 – Rigorous testing of individual account transaction components.

Functional Testing: Weeks 2-3 – Detailed examination of the account transaction functionality and user experience.

Load Testing: Week 4 – Simulating various load conditions to validate the account transaction application's performance and stability.

**Conclusion:** We have created a software plan for the intended application



## **Software Engineering Lab Experiment No. 7**

**Aim:** Perform Functional testing for the project

### **Objectives:**

1. The primary objective of functional testing for login and account transactions is to ensure that these critical components of the Spring Boot bank application perform as intended.
2. This includes validating the accuracy and security of the login process and verifying the correctness of account transactions such as fund transfers, balance inquiries, and transaction history.

### **Requirements:**

1. Computer with internet access.
2. Sample software project or problem statement for requirements analysis.
3. Word processing software for creating the lab report.

### **Concept:**

- Functional testing involves assessing the application's functionalities by testing its inputs, outputs, and the interaction of its components. For login and account transactions, this means evaluating the user interface, data processing, security features, and the overall user experience.

### **Tools:**

1. **Postman:** Useful for API testing, especially for account transactions that may involve backend APIs.

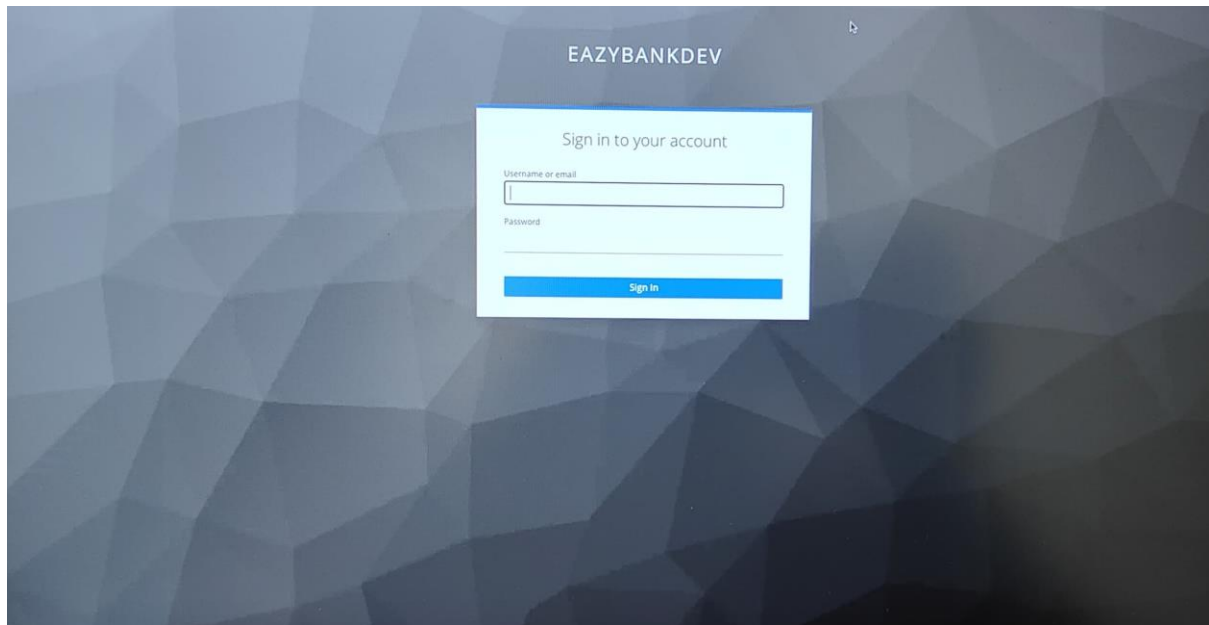
**Steps:**

## 1. Writing down test cases in excel sheet

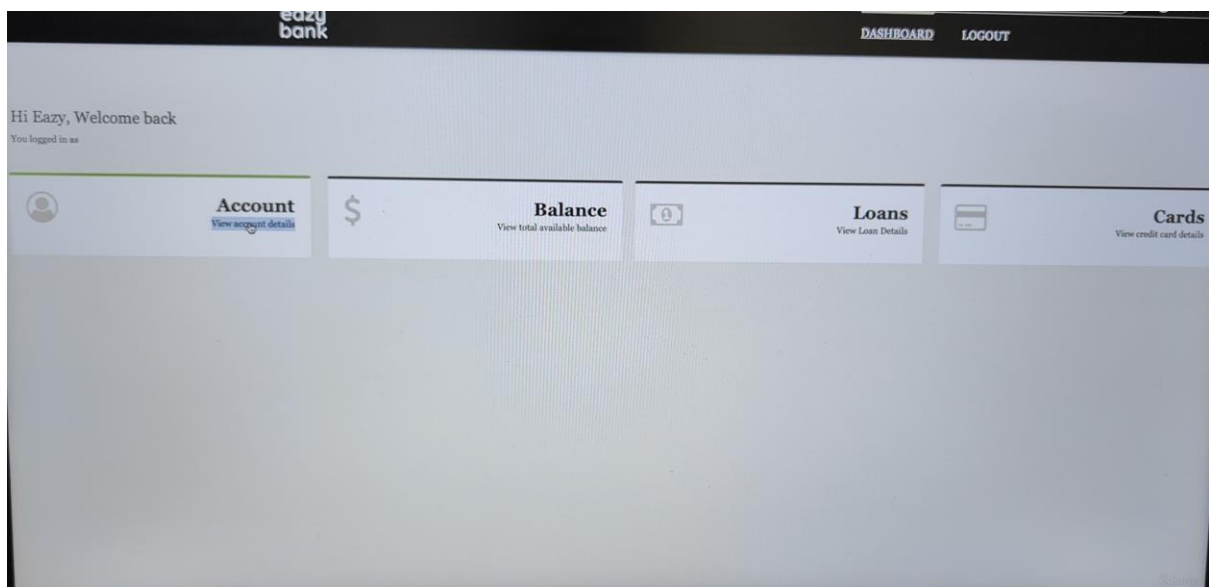
Bank Application Test Cases					
Module	Description	Steps	Expected Result	Actual result	Status (Passed/Failed)
Login	Login with correct CustomerId and password	1. Enter 12 digit Customer Id	Login successful	Login successful	Pass
		2. Enter password(8 characters ,1 lower case one upper case, one special char)			
		3. Enter OPT on mobile number			
		4. Fill captcha			
		5. Click on submit			
	Login with incorrect CustomerId and correct password	1. Enter 12 digit Customer Id	Login failed (Invalid captcha)	Login failed (Error message not displayed)	Fail
		2. Enter password(8 characters ,1 lower case one upper case, one special char)			
		3. Enter OPT on mobile number			
		4. Fill captcha			
		5. Click on submit			
	Login with correct CustomerId and incorrect password	1. Enter 10 digit Customer Id	Login failed (Error message displayed)	Login failed (Error message displayed)	Pass
		2. Enter password(8 characters ,1 lower case one upper case, one special char)			
		3. Enter OPT on mobile number			
		4. Fill captcha			
		5. Click on submit			
	Login with correct CustomerId and correct password and incorrect OTP	1. Enter 10 digit Customer Id	Login failed (Resend OTP message displayed)	Login failed (Resend OTP message displayed)	Pass
		2. Enter password(8 characters ,1 lower case one upper case, one special char)			
		3. Enter OPT on mobile number			
		4. Fill captcha			
		5. Click on submit			
Account transaction	User initiates a transaction (deposit, withdrawal)	1. Select account	Transaction Successful (withdrawal, deposit)	Transaction Successful (withdrawal, deposit)	Pass
		2. Enter IFSC code			
		3. Enter transferable amount			
		4. Enter OTP			
	User tries to withdraw money that is available in their account	1. Enter account type (Saving/Current)	Transaction Successful(Amount debited)	Transaction Successful(Amount debited)	Pass
		2. Enter amount			
		3. Enter transferable amount			
		4. Enter captcha			
		5. Enter OTP			
	User tries to withdraw more money than is available in their account	1. Enter account type (Saving/Current)	Transaction Successful(Amount debited)	Transaction Failed (Amount debited)	Failed
		2. Enter amount			
		3. Enter transferable amount			
		4. Enter captcha			
		5. Enter OTP			
	User attempts a transaction that exceeds daily or per-transaction limits	1. Enter account type (Saving/Current)	Transaction Failed (Limit exceed for the day)	Transaction Successful (Amount debited)	Failed
		2. Enter amount			
		3. Enter transferable amount			
		4. Enter captcha			
		5. Enter OTP			
	User attempts a transaction	1. Enter account type (Saving/Current)	Transaction Failed (Wrong OTP entered)	Transaction Failed (Wrong OTP entered)	Pass
		2. Enter amount			
		3. Enter transferable amount			
		4. Enter captcha			
		5. Enter OTP			

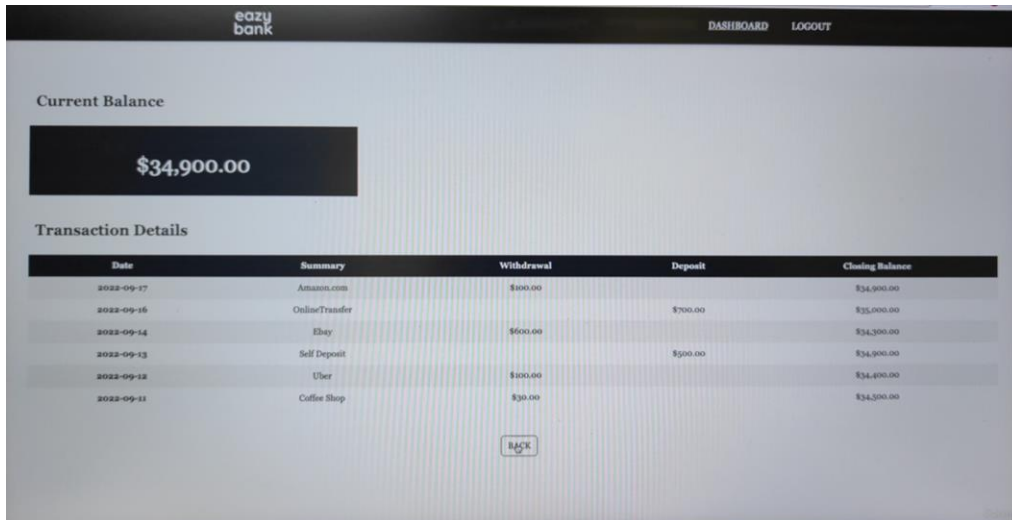
2. User tries to withdraw money that is available in their account (Transaction failed but money debited)

-Login Page



- Welcome, Page after successful login.



**-Account Transaction Section**

The screenshot displays the 'eazy bank' dashboard. At the top, there is a navigation bar with 'eazy bank' on the left and 'DASHBOARD' and 'LOGOUT' on the right. Below the navigation bar, the 'Current Balance' is shown as '\$34,900.00' in a dark box. Underneath, the 'Transaction Details' section contains a table with the following data:

Date	Summary	Withdrawal	Deposit	Closing Balance
2022-09-17	Amazon.com	\$100.00		\$34,900.00
2022-09-16	Online Transfer		\$700.00	\$35,000.00
2022-09-14	Ebay	\$600.00		\$34,300.00
2022-09-13	Self Deposit		\$500.00	\$34,900.00
2022-09-12	Uber	\$100.00		\$34,400.00
2022-09-11	Coffee Shop	\$30.00		\$34,500.00

Below the table, there is a 'BACK' button.

**Conclusion:**

We learnt that, Functional testing for login and account transactions is crucial to ensure the reliability, security, and user-friendliness of these core functionalities. By systematically testing each component, automating repetitive tasks, and validating the security measures in place, the testing process aims to enhance the overall quality of the Spring Boot bank application.

**References:**

- <https://www.softwaretestinghelp.com/black-box-testing/>
- <https://birdeatsbug.com/blog/black-box-testing>

## **Software Engineering Lab Experiment No. 8**

**Aim:** Perform White box testing for the project

**Objectives:**

1. Code Coverage: Ensure that all parts of the code are executed at least once during testing.
2. This includes validating the accuracy and security of the login process and verifying the correctness of account transactions such as fund transfers, balance inquiries, and transaction history.

**Requirements:**

1. Computer with internet access.
2. Sample software project or problem statement for requirements analysis.
3. Word processing software for creating the lab report.
4. IDE for running project

**Concept:**

- **White box testing** is based on the understanding of the internal structure of the code. Test cases are designed to exercise different paths through the code, including loops, branches, and conditions. The goal is to ensure that the code behaves correctly under various scenarios.

**Tools:**

1. **JUnit:** A widely used testing framework for Java that supports the creation and execution of unit tests.
2. **JaCoCo:** A Java Code Coverage Library that shows how much of your code is covered by tests.
3. **SonarQube:** An open-source platform for continuous inspection of code quality.
4. **OWASP Dependency-Check:** A tool for identifying project dependencies and checking if there are any known, publicly disclosed, vulnerabilities.

**Steps:** Steps for White Box Testing

1. **Code Review:** Understand the codebase through code review to identify potential areas of interest and risk.:

**1. Successful Login:**

- Scenario: User provides correct username and password.
- Expected Result: The login should succeed.

**2. Failed Login - Incorrect Password:**

- Scenario: User provides a correct username but an incorrect password.
- Expected Result: The login should fail.

**3. Failed Login - Invalid Username:**

- Scenario: User provides an invalid username.
- Expected Result: The login should fail.

**4. Failed Login - Empty Password:**

- Scenario: User provides a valid username but an empty password.
- Expected Result: The login should fail.

**5. Failed Login - Empty Username:**

- Scenario: User provides an empty username.
- Expected Result: The login should fail.

**6. Deposit Positive Amount:**

- Scenario: User deposits a positive amount into the account.
- Expected Result: The account balance should increase.

**7. Withdraw Positive Amount:**

- Scenario: User withdraws a positive amount from the account with sufficient funds.

- Expected Result: The account balance should decrease.

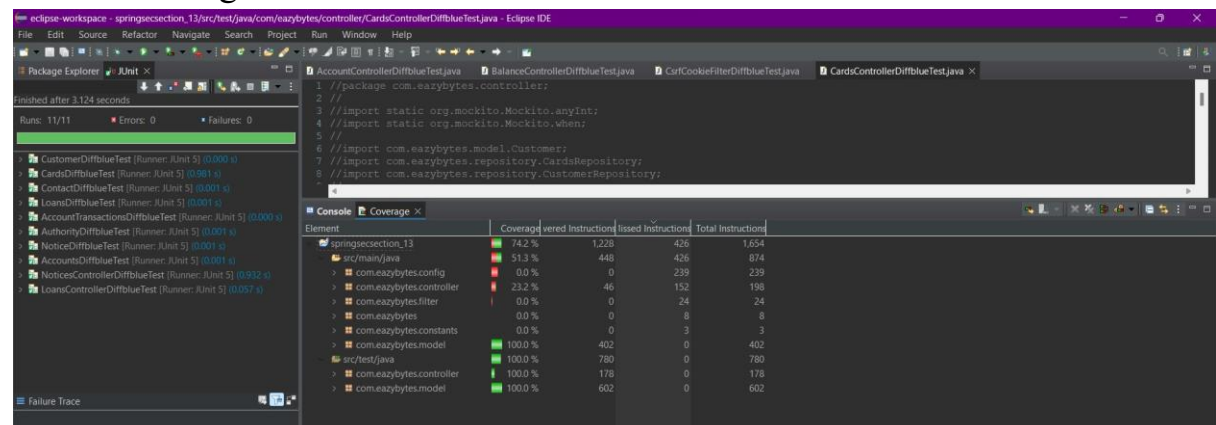
**8. Withdraw Insufficient Funds:**

- Scenario: User attempts to withdraw more than the account balance.
- Expected Result: An 'InsufficientFundsException' should be thrown.

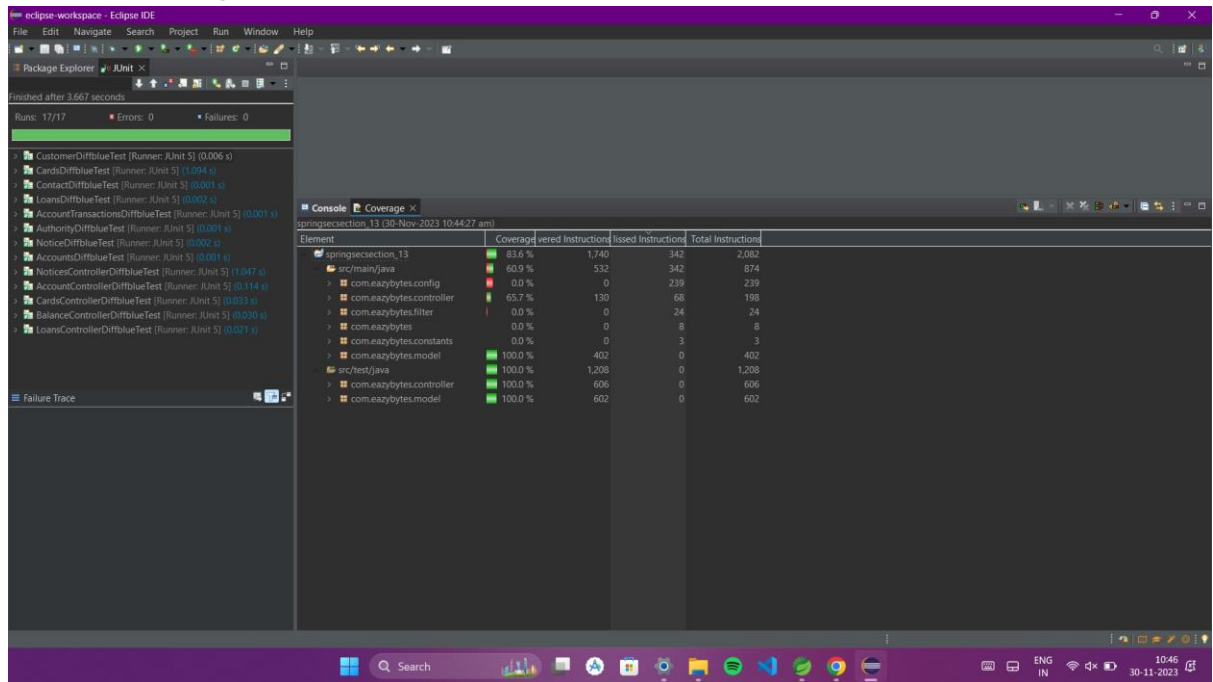
2. **Unit Testing:** Write and execute unit tests for individual functions or methods. Use tools like JUnit for Java.

- Total tests run = 11

Code coverage = 74.2%



- Total tests run = 17
- Code coverage = 83.6%



## Conclusion:

We learnt that white box testing is crucial for ensuring the reliability, security, and performance of a Java project. By testing the internal logic and structure of the code, before they impact the functionality or security of the application.

## References:

- <https://www.softwaretestinghelp.com/black-box-testing/>
- <https://birdeatsbug.com/blog/black-box-testing>

## **Software Engineering Lab Experiment No. 9**

**Aim:** Project Size estimation using function point

**Objectives:**

1. The primary objective of Project Size Estimation using Function Point Analysis is to quantify the functional size of a software project based on the functionality it is expected to deliver.
2. Function points provide a standardized measure of the functional size of a software application. This size is independent of the technology or programming language used to implement the system.

**Requirements:**

1. Computer with internet access.
2. Sample software project or problem statement for requirements analysis.
3. Word processing software for creating the lab report.

**Concept:**

Function Point Analysis (FPA) is a method used for estimating the size of a software project based on the functionality provided by the system. The process involves identifying and quantifying the functionality of the software in terms of "function points." The function points are then used to estimate the size of the project in terms of lines of code or person-months.

**Tools:**

1. Function point calculator
2. FP, UFP, CAF



**Steps:**

1. Calculate function points for each module.

External Inputs (EI)

External Outputs (EO)

External Inquiries (EQ)

Internal Logical Files (ILF)

External Interface Files (EIF)

For login module

Function Point Calculator					Total	Factor	FP
The Madison Utilities, Department of Computer Science, James Madison University					51	1.10	56
Direct Measure	Simple	Count	Complex	Weighted Measure			
External Inputs (EIs)	6	0	0	18			
External Outputs (EOs)	3	0	0	12			
External Inquiries (EQs)	3	0	0	9			
Internal Logical Files (ILFs)	1	0	0	7			
External Interface Files (EIFs)	1	0	0	5			
<input type="button" value="Clear"/>							

Value Adjustment Factor	0	1	2	3	4	5
The system requires reliable backup and recovery.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Specialized data communications are required.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
There are distributed processing functions.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Performance is critical.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
The system runs in an existing, heavily utilized operational environment.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The system requires on-line data entry.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The on-line data entry requires transactions over multiple screens operations.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
ILFs are updated on-line.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
The inputs, outputs, files or inquiries are complex.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
The internal processing is complex.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The code is designed to be reusable.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Conversions /installation are included in the design.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The system is designed for multiple installations in different organizations.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The system is designed to facilitate change and ease of use.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

For account transaction module

Function Point Calculator					Total	Factor	FP
The Madison Utilities, Department of Computer Science, James Madison University					52	1.16	60
Direct Measure	Simple	Count	Complex	Weighted Measure			
External Inputs (EIs)	6	0	0	18			
External Outputs (EOs)	2	0	0	8			
External Inquiries (EQs)	4	0	0	12			
Internal Logical Files (ILFs)	2	0	0	14			
External Interface Files (EIFs)	0	0	0	0			
<input type="button" value="Clear"/>							

Value Adjustment Factor	0	1	2	3	4	5
The system requires reliable backup and recovery.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Specialized data communications are required.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
There are distributed processing functions.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Performance is critical.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The system runs in an existing, heavily utilized operational environment.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The system requires on-line data entry.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The on-line data entry requires transactions over multiple screens operations.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
ILFs are updated on-line.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The inputs, outputs, files or inquiries are complex.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The internal processing is complex.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
The code is designed to be reusable.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Conversions /installation are included in the design.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The system is designed for multiple installations in different organizations.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
The system is designed to facilitate change and ease of use.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2. Calculate total function points for whole project. (add FP of all modules)

Size Estimation using function point									
Sr No	Requirement	External Inputs(EIs)	External Outputs(EOs)	External Inquiries (EQs)	Internal Logical Files (ILFs)	External Interface Files (EIFs)	UAF	CAF	SIZE (FP)
1	Login	6	3	3	1	1	51	1.1	56
2	Account transaction	3	5	3	1	3	60	1.1	66
Total Size									122

3. Multiply Unidentified Function Points with Complexity Adjustment Factor and calculate Function points. (FP=UFP\*CAF)

### **Conclusion:**

We learnt that Function Point Analysis (FPA) is a structured and widely used method for quantifying the functional size of a software application. It provides a standardized and objective way to measure the functionality provided by a system, independent of the technology or programming language used.

### **References:**

- <https://w3.cs.jmu.edu/bernstdh/web/common/webapps/oop/fpcalculator/FunctionPointCalculator.html>
- <https://birdeatsbug.com/blog/black-box-testing>
- <https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/>

## **Software Engineering Lab Experiment No. 10**

**Aim:** Effort and Cost Estimation for the project.

### **Objectives:**

The effort and cost estimation of a software project is a crucial aspect of project planning and management. The primary objectives of effort and cost estimation in a software project include:

- Project Planning
- Resource Allocation
- Budgeting
- Risk Management
- Schedule Management
- Client Communication

### **Requirements:**

1. Computer with internet access.
2. Sample software project or problem statement for requirements analysis.
3. Word processing software for creating the lab report.

### **Concept:**

**Effort estimation:** This involves predicting the amount of work or labour hours required to complete specific tasks, phases, or the entire software development project.

**Cost estimation:** involves predicting the financial resources needed to complete the software development project. It includes direct costs (e.g., salaries, software licenses) and indirect costs (e.g., overhead, administrative expenses).

### **Tools:**

1. Excel

**Steps:**

## 1. Calculate efforts

$$\text{Efforts} = \text{Size/Productivity (PDs)}$$

Productivity = 0.7 (taken from online data: average productivity of Java projects)

Estimated efforts = 174 PDs

Table No. 1 Effort Estimation				
Sr No.		Size of project(FP)	Estimated Productivity	Estimated Efforts(PD)
1	Login	56	0.7	80
2	Account transactions	66	0.7	94
<b>Total</b>		<b>122</b>		<b>174</b>

## 1. Suppose cost per PD is 100\$ (8300 rupees)

Cost for 174 PDs =  $174 \times 8300 = 14,44,200$

Total cost after adding 30% overhead cost and 15% margin = 20,94,090 rupees.

Table No. 2 Cost Estimation				
Sr No.	Cost Heads	Efforts(PDs)	Cost/PD	Total Cost
1	Project cost	174	5819	14,44,200
2	Overhead cost	0	1745	4,33,260
3	Profit margin	0	872	2,16,630
<b>Total</b>		<b>174</b>		<b>20,94,090</b>

## 3. Distribution of efforts across the project

Note: distribution of efforts taken from general data for java projects.

Table No. 3			
Sr. No.	Distribution of efforts across project	% Distribution	Efforts
1	Requirement analysis	6%	7.32
2	Design	10%	12.2
3	Coding	32%	39.04
4	Unit testing	30%	36.6
5	Acceptance testing	12%	14.64
6	Management	10%	12.2
<b>Total</b>		<b>100%</b>	<b>122</b>

**Note:** Productivity, efforts distribution and cost/pd is taken from online data: average values for Java projects.

**Conclusion:**

In conclusion, effort and cost analysis are important project management in software development. Estimating the effort and cost for a project is a critical aspect of project planning. Several factors influence the estimation process, and it requires careful consideration of various elements.

**References:**

1. <https://w3.cs.jmu.edu/bernstdh/web/common/webapps/oop/fpcalculator/FunctionPointCalculator.html>
2. <https://birdeatsbug.com/blog/black-box-testing>