./

GENESIS - Learning Outcome & Mini-project Summary Report



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be Approved** | **Remarks/Revision Details** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Details**

Contents

[Contents 3](#_Toc55470819)

[Miniproject -1 [Individual] 4](#_Toc55470820)

[Module/s 4](#_Toc55470821)

Intermediate C++ and Advanced C++

→ STL

→ Default and parameterized constructor

→ Operator Overloading

→ File Operations [4](#_Toc55470822)

[Objectives & Requirements 4](#_Toc55470823)

[Design 4](#_Toc55470824)

[Test Plan 4](#_Toc55470825)

[Implementation Summary 4](#_Toc55470826)

[Video Summary 4](#_Toc55470827)

[Git Link 4](#_Toc55470828)

[Git Dashboard 4](#_Toc55470829)

[Summary 4](#_Toc55470830)

[Individual Contribution & Highlights 5](#_Toc55470831)

[Summary 5](#_Toc55470832)

[Challenges faced and how were they overcome 5](#_Toc55470833)

[Future Scope (If applicable) 5](#_Toc55470834)

[Miniproject -2 [Team/Individual] 6](#_Toc55470835)

[Module/s 6](#_Toc55470836)

Linux and OS

[6](#_Toc55470837)

[Objectives & Requirements 6](#_Toc55470838)

[Design 6](#_Toc55470839)

[Test Plan 6](#_Toc55470840)

[Implementation Summary 6](#_Toc55470841)

[Git Link 6](#_Toc55470842)

[Git Dashboard 6](#_Toc55470843)

[Summary 6](#_Toc55470844)

[Individual Contribution & Highlights 6](#_Toc55470845)

[Summary 6](#_Toc55470846)

[Challenges faced and how were they overcome 6](#_Toc55470847)

# Miniproject -1 [Individual]

## Module/s

“Module linked to the miniproject- Intermediate/ Advanced C++

### Topic and Subtopics

→ STL(Standard Template Library)

→ Default and parameterized constructor

→ Operator Overloading

→ File Operations

STL was implemented using map container. When we read data from .DATA file, the data is stored in map container, which has key value pair. Default and parameterized constructors were used. Operator overloading was used on insertion and extraction operator because here we are passing user defined objects which are not defined, so we will overload insertion and extraction operator. Regarding file operations, reading file, writing file. Ostream, ofstream and istream is used. In ostream, contents are written to the file and in ofstream file is created and contents are written to it and in in-stream file is read.

## Objectives & Requirements

a. Objectives

→ To develop bank application with basic operations such as deposit, withdraw and display balance.

→ To implement deposit function to deposit amount from the bank.

→ To implement withdraw function to withdraw amount from the bank.

→ To implement display balance function to display balance to the user.

→ To implement open account function so that user can open bank account.

→ To implement close account function so that user can close bank account.

→ Display all function is specifically for bank officials which is used to display details of all customers.

b. Requirements

b.i High Level Requirements

|  |  |
| --- | --- |
| ID | Description |
| HL\_01 | Open Account function to open account |
| HL\_02 | Deposit function to deposit money |
| HL\_03 | Withdraw function to withdraw money |
| HL\_04 | Balance Enquiry to check for balance |
| HL\_05 | Close Account function to close account |
| HL\_06 | Display details of all accounts |

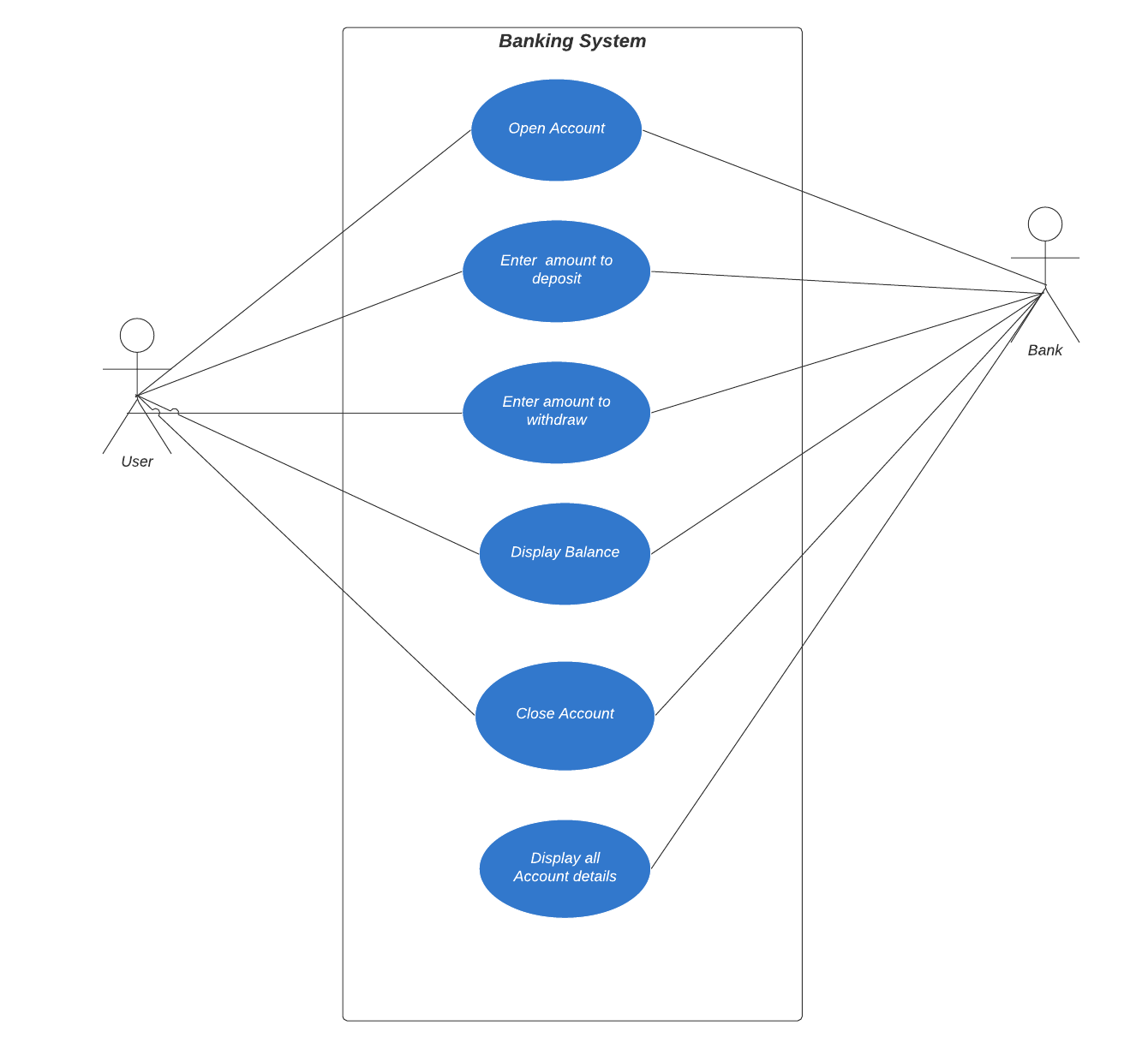
b.ii Low Level Requirements

|  |  |
| --- | --- |
| ID | Description |
| LL\_01\_HL\_01 | Enter valid first and last name to create account |
| LL\_02\_HL\_02 | Enter valid account number first and then enter appropriate amount to deposit |
| LL\_03\_HL\_03 | Enter valid account number and enter amount less than bank balance to withdraw |
| LL\_04\_HL\_04 | Enter valid account number for balance enquiry |
| LL\_05\_HL\_05 | Enter valid account number to close Account |
| LL\_06\_HL\_06 | Account details of accounts should be displayed |

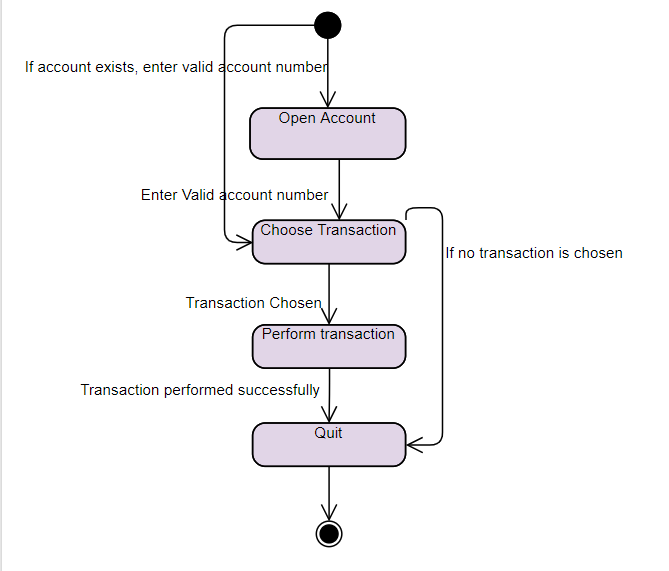
## Design

1**.Behavioral Diagram:**

1.a. Use Case Diagram

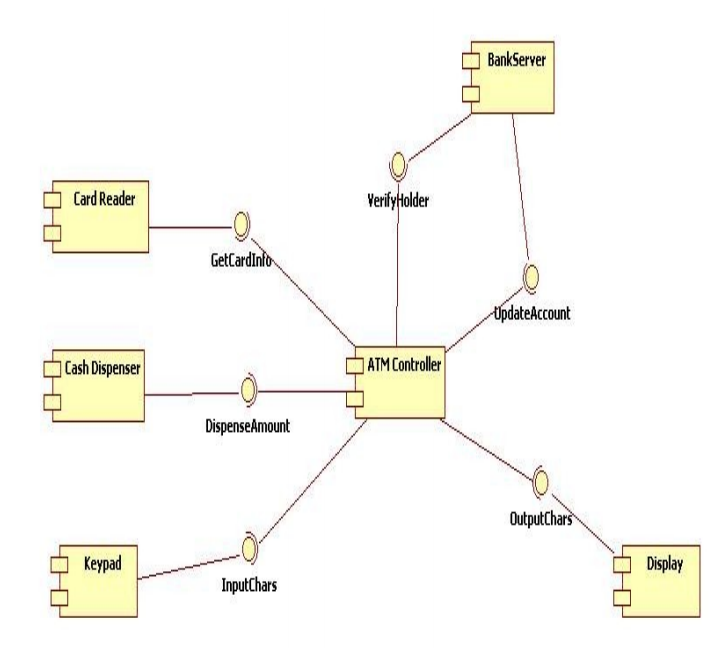


1.b. State Diagram



2. Structural Diagram

2. a Component Diagram



## Test Plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Pre-Condition | Expected Input | Expected Output | Actual Output |
| HL\_01 | Open Account | None | First and Last Name | Account created successfully |  |
| HL\_02 | Deposit money | Account should exist | Enter valid account number | Amount deposited successfully |  |
| HL\_03 | Withdraw money | There should be amount in bank | Enter valid account number and amount to be withdrawn | Amount withdrawn successfully |  |
| HL\_04 | Balance Enquiry | Account should exist | Enter valid account number | Balance displayed |  |
| HL\_05 | Close Account | Account should exist | Enter valid account number | Account closed successfully |  |
| HL\_06 | Display details of Accounts | Accounts should exist | None | Display account details of all users |  |
| LL\_01\_HL\_01 | Enter valid first and last name to create account | None | First and last name | Account is created successfully and message is displayed. |  |
| LL\_02\_HL\_02 | Enter valid account number and then enter appropriate amount to deposit | Account should exist | Account number and amount to deposit should be entered | Amount is deposited successfully |  |
| LL\_03\_HL\_03 | Enter valid account number and enter amount less than balance to withdraw | Account should exist | Enter account number and amount to be withdrawn should be entered | Amount is withdrawn and message is displayed |  |
| LL\_04\_HL\_04 | Enter valid account number for balance enquiry | Account should be present | Enter valid account number | Balance is displayed |  |
| LL\_05\_HL\_05 | Enter valid account number to close account | Account should be present | Enter valid account number | Account successfully closed and message is displayed |  |
| LL\_06\_HL\_06 | Display all account details | Accounts should be present | None | Account details of all accounts displayed. |  |

## Implementation Summary

The project is implemented in C++. Here I have used concepts like constructors(default and parameterized), file operations, Standard Template Library. Some functions which are implemented are Deposit, withdraw, display particular user details, display all accounts details, Open account and close Account. When user enters the details, it gets stored in the file and when details are retrieved from file, it is stored in Map STL container which has key as account number and value field as account details. Some file operations are used are: istream,ostream and ofstream. Here ofstream is used to open the existing file and istream is used to read contents from the file and ostream is used to display the output to the user. All these operations are overloaded because we pass user defined objects to these functions and it is not originally designed to accept user defined objects, so we will overload these operators to make sure it accepts user defined objects. There is empty class to check if amount entered to be withdrawn is greater than balance. If yes, it throws error.

Google Test is also implemented for credit, debit and display functions to check if program is passing all the unit test cases.

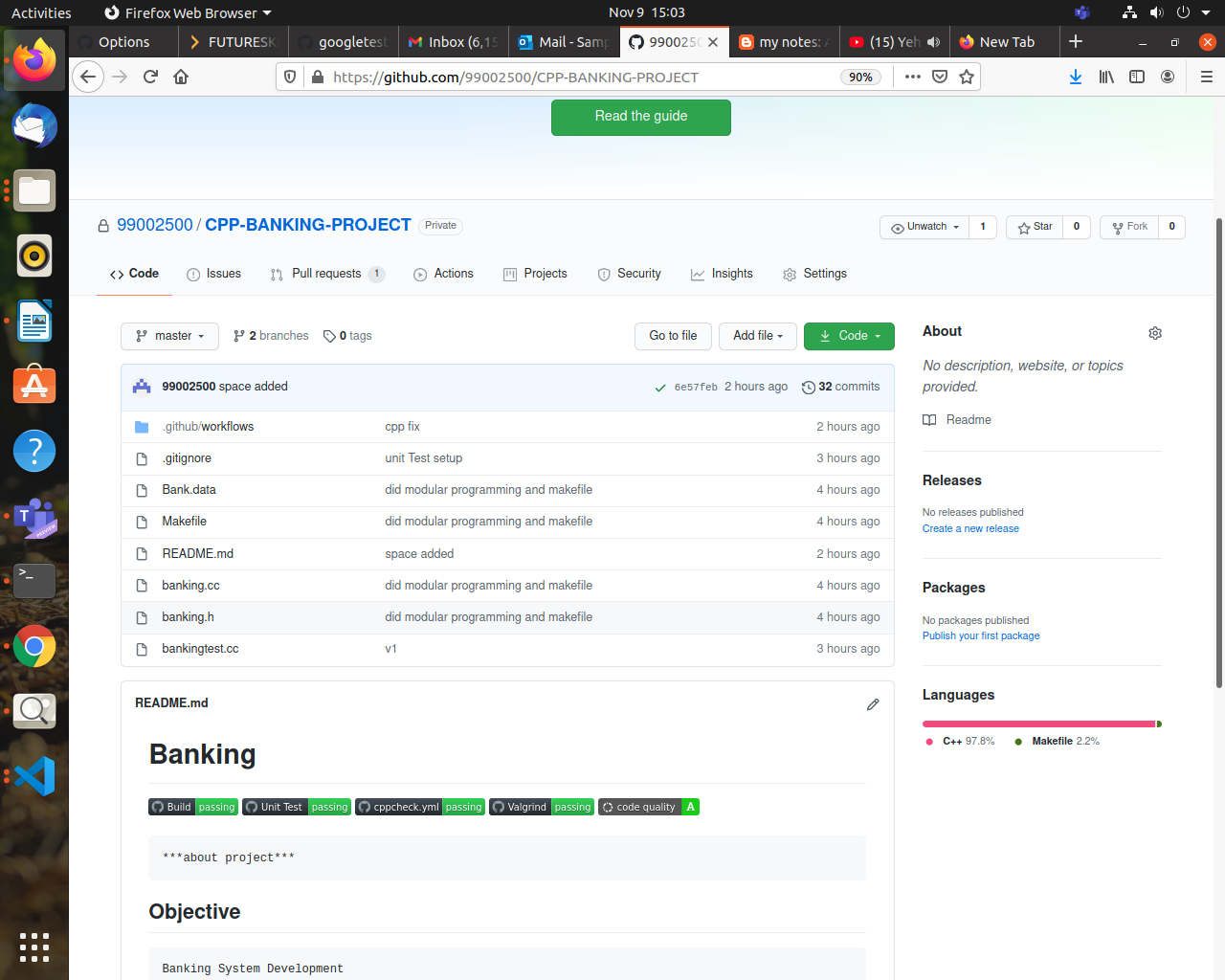
### Video Summary

“Please upload a short video on the repo for the walkthrough of the project (Team/Individual) less than 7min and less than 30MB File Size. Start is the Standard opening slide with title of miniproject + Team members followed by the walkthrough ”

### Git Link

https://github.com/99002500/CPP-BANKING-PROJECT

### Git Dashboard



### Summary

Banking project is implemented using C++. This application supports opening account, deposit, withdraw, display account details, close account. Display all account details is specifically for bank officials to see details of all bank accounts. Details entered by user gets stored in the file and then it is retrieved when user requests for it.

STL Map container is used to store the data retrieved from file and then it is displayed to the user. Account number is generated randomly for every user and if user wants to do any transaction, they have to enter valid account number. Overall, this application supports basic banking operations.

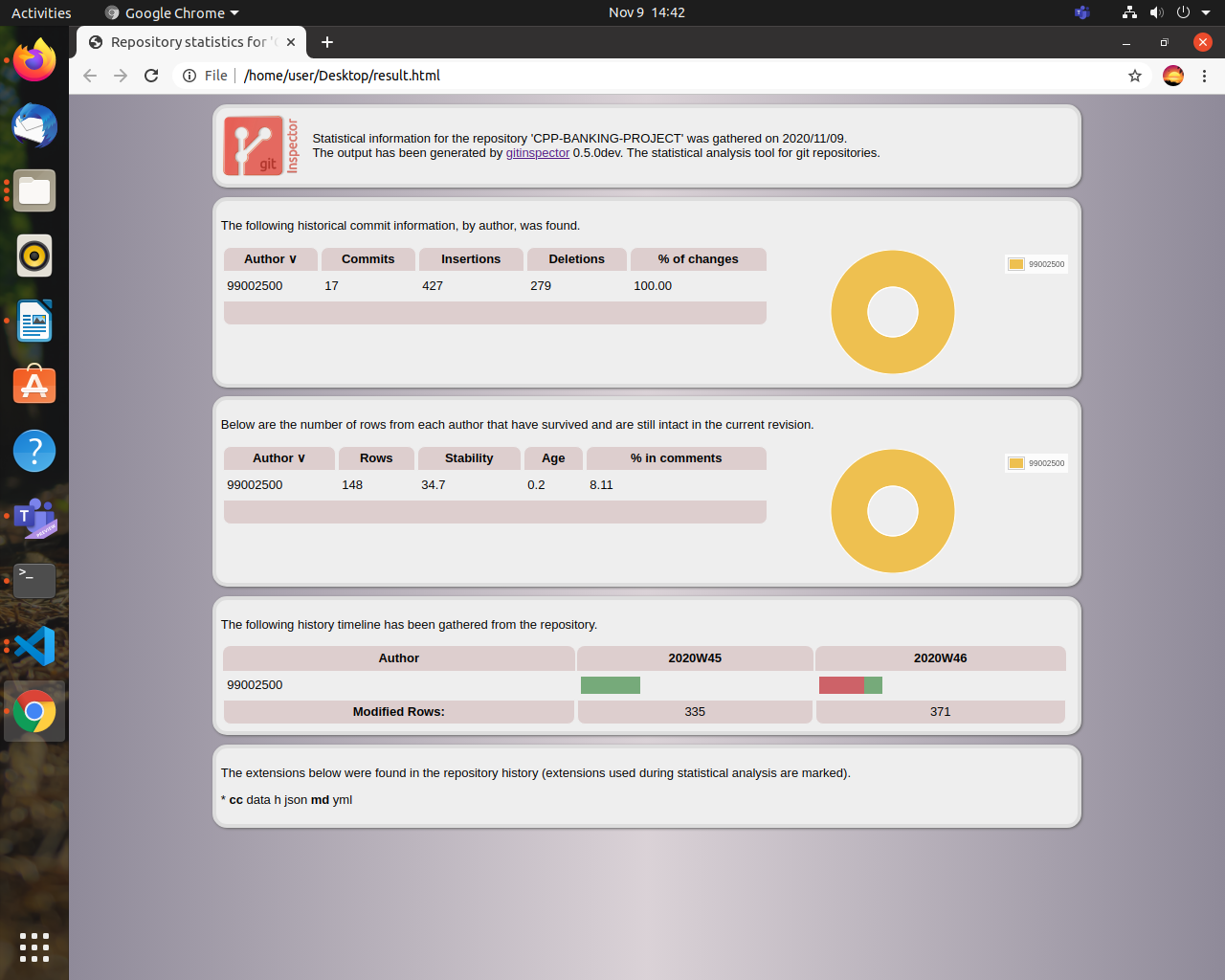
#### Git inspector summary

“In linux install gitinspector by doing sudo apt-get install gitinspector.

Download gitinspector repository from: https://github.com/ejwa/gitinspector

and go to gitinspector repository and run the command below:

./gitinspector.py -wTHl --format=htmlembedded --file-types=cs,cshtml,js,cc,md ~/Your repo path >~/Desktop/result.html.



#### Build

The application is developed using Microsoft Vs Code. The code is compiled using g++ compiler. MakeFile is used to generate output files.

Gtest: bankingtest.cc banking.h

g++ $^ -o $@ -lgtest -lgtest\_main -lpthread

The above code generates output file named **Gtest** by compiling bankingtest.cc and banking.h.

out: banking.cc

g++ $^ -o $@

The above code generates output file named **out** by compiling banking.cc.

#### Code quality and Issues or Bug Tracking

Cppcheck was used to check static code analysis and valgrind was used to check if there are any memory leaks which is basically dynamic code analysis. Code was also checked on codacy for detailed static code analysis and got A grade. Unit test was also done to check if each function is working properly as per requirements. Faced lot of issues while doing Github Actions like some mistakes in unit-test.yml and valgrind.yml because I had not mentioned make command. I had not overloaded operators properly which led to erros which was rectified later.

#### Unit Testing

Google Test was used to do unit Testing. Each function was thoroughly tested. Credit function was tested to check if amount is deposited, whether it is actually deposited. Debit function was tested to check if amount was debited successfully. Testing was also done to check if amount entered to withdraw is greater than bank balance. If it was greater, then test fails or else test passes. Test was also done for display function which displays the details of the particular user.

## Individual Contribution & Highlights

### Challenges faced and how were they overcome

→ Challenge was to figure out proper file operations for the applications. Figured out by searching on internet

→ Choosing the appropriate STL for the application. Checked all STL containers and felt Map is better container because it is faster and since account number is unique for each user, so went ahead with map.

→ Setting up google test in VsCode and setting up makefile to run program and google tests. Friend taught me how to set up lpthread in vscode

→ Shifting from traditional programming approach to modular programming approach.

→ Learning about debugging tactics from my friend and adding alias to bashrc file.

### Miniproject -2 [Individual]

## Linux and OS

### → Threads, mutex and semaphores

→ Networking concepts

→ Signals

→ File descriptors

## Objectives & Requirements

a. Objectives

→ To develop client server application where one client can communicate with other client via server.

→ To display details of user joined conversation, left conversation and chat details on server terminal.

→ To support multiple clients and single server and all communications are routed via server.

→ Clients can send message to other clients or receive message from other clients

→ Clients can join conversation by entering their name and leave the conversation when they are done.

b. Requirement Analysis

b.i High level Requirements

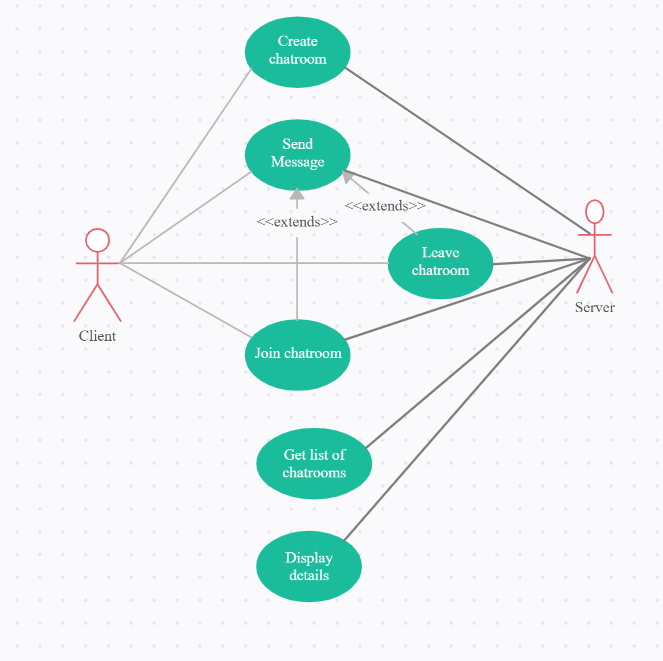
|  |  |
| --- | --- |
| ID | Description |
| HL\_01 | Application shall support communication among people only in chatroom |
| HL\_02 | Application supports only text based messaging |
| HL\_03 | Message should get delivered when user presses on enter. |

b.ii Low level Requirements

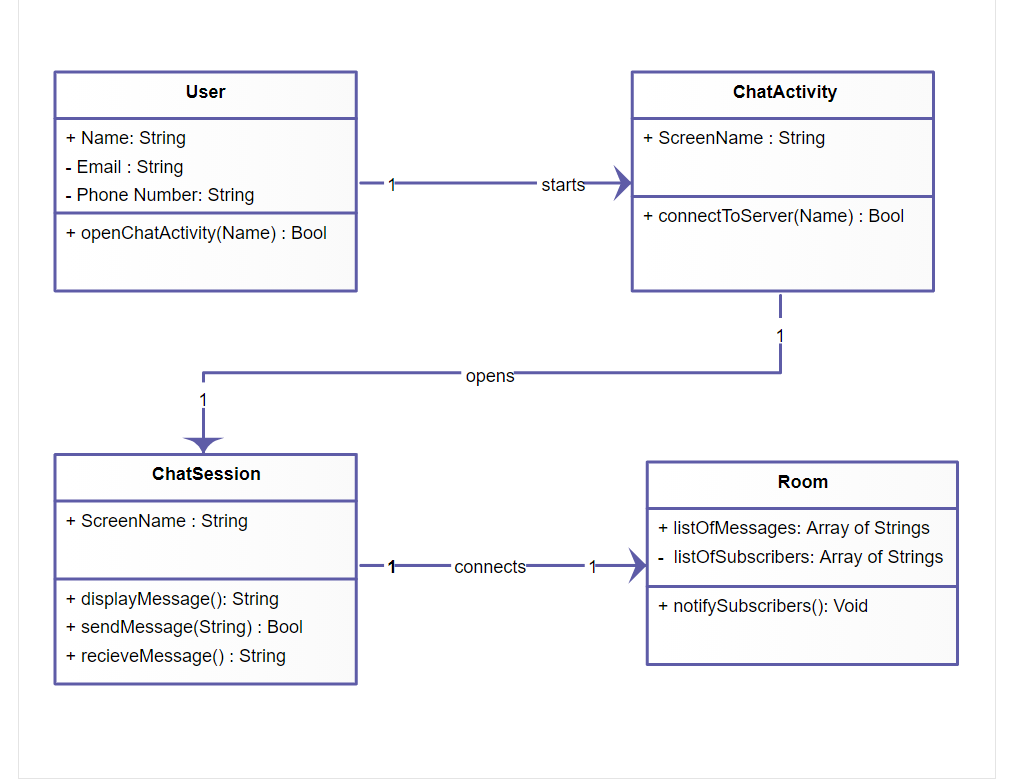
|  |  |
| --- | --- |
| ID | Description |
| LL\_01\_HL\_01 | Atleast 2 people should be there to start communication and maximum 100 people are allowed |
| LL\_02\_HL\_02 | Application doesnt support special characters in messaging |
| LL\_03\_HL\_03 | Message should get delivered when atleast 2 users are present in the chatroom |

## Design

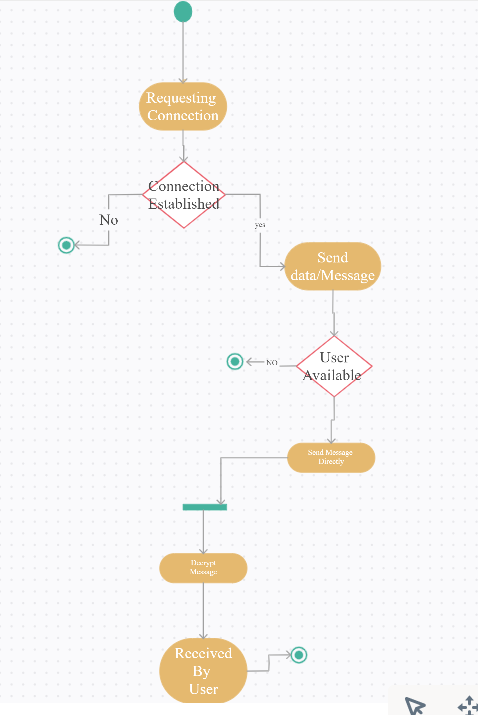
a. Use Case Diagram



b. Class Diagram



c. State Diagram



## Implementation Summary

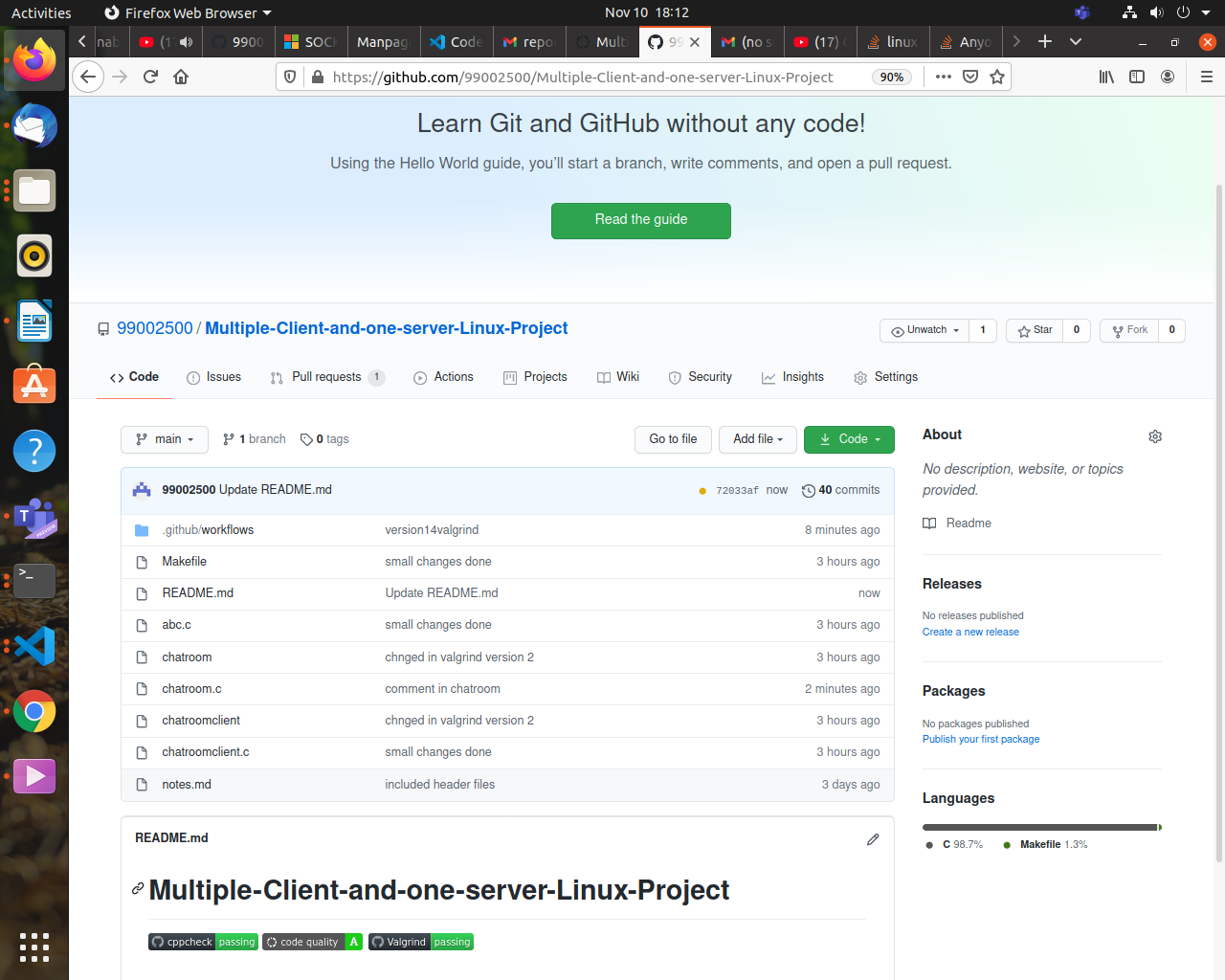
This application is implemented using C programming and I have used OS concepts such as threads, mutex and semaphores. There are 2 different files namely client.c and server.c. In server we will define structure basically like socketaddr, sockfd, userid and name.

Since there are multiple clients here, we need to add clients to the queue. When clients are done with the conversation, there are removed from the queue. For that queue\_remove function is implemented. We will then define function to handle communication between multiple clients which checks some conditions like whether name field is empty etc. If name is entered, it displays particular user has joined the conversation and message is sent between clients. In client application, we define functions for send message handler and receiver message handler. If user does not enter any port number we will display message that user has to enter the port number. Some settings are also applied for sockets.

### Git Link

https://github.com/99002500/Multiple-Client-and-one-server-Linux-Project

### Git Dashboard



### Summary

#### Git inspector summary

#### Build

Makefile was used to build the files. Commands were written in makefile and then we do: make Makefile compile. It compiles and generates 2 object files namely chatroom.o and chatroomclient.o.

#### Code quality

Code was run on Codacy for code analysis and got A grade. Static code analysis was also done using cppcheck and dynamic code analysis was done using valgrind.

## Individual Contribution & Highlights

→ Developed chatroom application which supports multiple clients and single server.

→ Communication made possible between multiple clients via server

→ Made use of socket programming concepts and OS concepts such as semaphores, locks etc.

### Summary

This application is basically designed to support communication between multiple clients via server. Clients have to enter their name and then join the chatroom and all the communications are routed via server. Concepts like semaphores, locks, mutexes are used to achieve objectives of this application. Application requires multiple terminals to run. One for server and atleast 2 terminals for client to run.

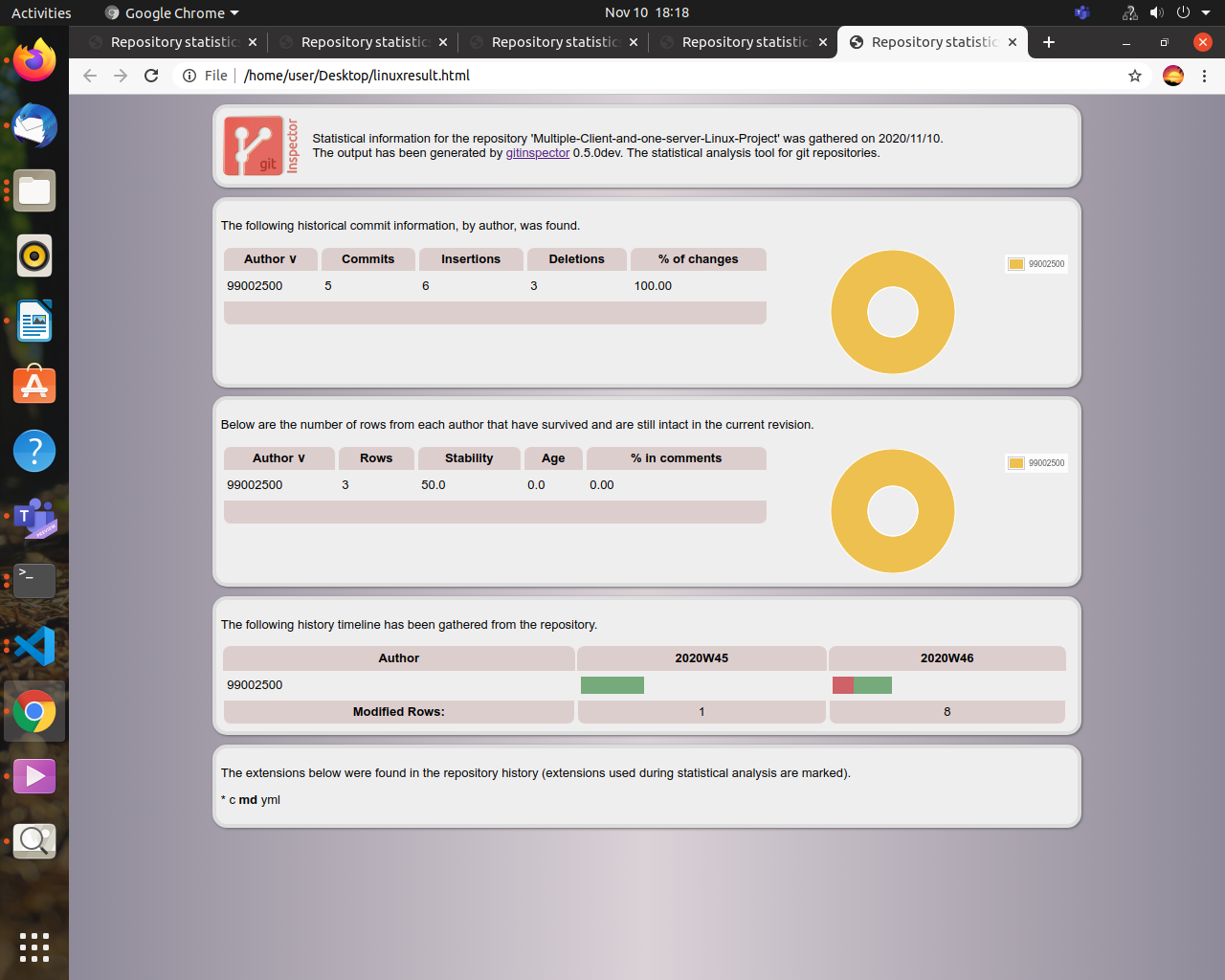
#### Git inspector summary

“In linux install gitinspector by doing sudo apt-get install gitinspector.

Download gitinspector repository from: https://github.com/ejwa/gitinspector

and go to gitinspector repository and run the command below:

./gitinspector.py -wTHl --format=htmlembedded --file-types=cs,cshtml,js,cc,md ~/Your repo path >~/Desktop/linuxresult.html.



### Challenges faced and how were they overcome

→ To implement the concepts such as semaphores, mutexes etc. Had to see some online tutorials and took help of friends and successfully implemented it

→ There were problems when message was sent it was not received and server could not detect the message sent or received. Had to debug the code for long time and could figure out I had not used functions properly and there were some problems in logic.

Miniproject -3 [Team]

Module/s

“Modules linked to the miniproject- Python Programming

Topic and Subtopics

→ Functions

→ Tweepy, TextBlob, pandas, Matplotlib

→ Regular Expressions

→ Loops

→ if and else statements

Objectives & Requirements

“High level and low level in the template ”

Design

“System Level and subsystem level UMLs – Structural and Behavioral”

Test Plan

“Integration level and unit level in the template”

Implementation Summary

“Section focused toward’ s implementation aspects. Here it is only core summary while all the details are in the Git Repo

Note: The GitHub private repo should be documented (Readme.md files at each folder level)

Ensure code quality and clean code and description practices

Mandatory: To add the GitHub user - stepin654321 as a contributor to the repo”

Video Summary

“Please upload a short video on the repo for the walkthrough of the project (Team/Individual) less than 7min and less than 30MB File Size. Start is the Standard opening slide with title of miniproject + Team members followed by the walkthrough ”

Git Link

“Link to the repo”

Git Dashboard

“Screenshot of the GitHub Repo page with all the badges and summary”

Summary

“Brief summary on the overall implementation”

Git inspector summary

“In linux install gitinspector and Run the command –

gitinspector -H -l -m -T -w -r --grading --format=html > gitinsp.html

and upload the same to your repo and paste the snapshot in the report”

Build

“Brief on outcome of the build and setup done”

Code quality and Issues or Bug Tracking

“Brief on code quality, errors and warnings flagged (issues created) and fixed ”

Unit Testing

“Unit Testing setup alignment with test plans and summary of outcome”

Individual Contribution & Highlights

“Brief on Contributions by you for Team”

Summary

“Key Highlights not covered till now, Softskills and technical side”

Challenges faced and how were they overcome

“Brief and crisp”

Future Scope (If applicable)