Topic: Bug Prediction using Neural Network

Slot: L43+L44

Group Members:

Nimish K. Aggarwal – 19BCE0014

Vrushali Deshmukh – 19BCE0033

Tarang Garg – 19BCE0053

Shashwat Chaudhary – 19BCE0056

ACKNOWLEDGEMENT

The project has been done under the guidance of Vellore Institute of Technology as well as our faculty for Software Engineering Ms. Swathi JN. Various information regarding the project has been obtained from various other open-source websites.

EXECUTIVE SUMMARY

This document presents the Software Design Specification for the Bug Prediction project by using Squeezenet Neural Networks. The major sections of the document address the system decomposition by module, concurrent process, and data entity. The system dependencies are also described.

Section 2, Decomposition Description, gives a view of the whole system design including concurrent processes and data entities that are common amongst all system modules. An important discussion of how much accurate the software is in its bug-prediction ability is included in this section. This discussion includes a UML Class Diagram that depicts the entire system.

Section 4, Interface Description, goes into detail about the user interface for each module of the Bug-Prediction Software. This is followed by an important discussion of the processes implemented in logic for each module of the system.

Section 5, Detailed Design, extends the design discussion found in Section 2 and describes the design for each system module in more detail. A UML Class diagram is included for each module design discussion. This is followed by a description of the data requirements for each module and the design of those data elements.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

Definition, Acronym, or Abbreviation	Description
SDS	Software Design Specification.
SRS	Software Requirement Specification

1. INTRODUCTION

1.1 Purpose

Software code is composed of several components which further consist of various different classes, libraries and functions. Testing all these components could be quite expensive considering time and resources. If we know beforehand that all components are more likely to contain defects then we can save a huge amount of time in testing and allocate resources optimally.

Software Defect Prediction is concerned with finding the defect-prone components of our software. We can build models that can help to identify components likely to be defective by using past software releases and bug fixes as training data for Machine Learning.

1.2 Scope

It is within the scope of the Software Design Specification to describe the specific system design of the Bug Prediction project. This would include user interface design, object-oriented class design, process design, and data design. Any specific detail that is needed about the standards or technology used to design the software are within the scope of this document.

2. PROJECT DESCRIPTION AND GOALS

Software code is composed of several components which further consist of various different classes, libraries and functions. Testing all these components could be quite expensive considering time and resources. If we know beforehand that all components are more likely to contain defects then we can save a huge amount of time in testing and allocate resources optimally. Software Defect Prediction is concerned with finding the defect-prone components of our software. We can build models that can help to identify components likely to be defective by using past software releases and bug fixes as training data for Machine Learning.

3. TECHNICAL SPECIFICATION

Bug Prediction using Neural Networks

Software Requirements Specification

Version: 1.0

List of Contributors

Vrushali Deshmukh	VD	Bugs Bunny	vrushali.deshmukh2019@vitstudent.ac.in
Nimish K. Aggarwal	NA	Bugs Bunny	nimishk.aggarwal2019@vitstudent.ac.in
Tarang Garg	TG	Bugs Bunny	tarang.garg2019@vitstudent.ac.in
Shashwat Chaudhary	SC	Bugs Bunny	shashwat.chaudhary2019@vitstudent.ac.in

Change History

1.0	07.06.2021	Version 1.0 released

Preface

This document represents the Software Requirements Specification for the Bug Prediction project using Squeezenet Neural Network. The document begins with an Introduction section that describes the purpose of the document and what is considered to be in the scope of this document as well as what is outside the scope of this document.

The next section is an Overall Description of the requirements and functions. This section includes the overall constraints that the project is working within as well as the assumptions made by the project as far as the defining the requirements is concerned. Lastly, the project dependencies are also listed in this section.

The Specific Requirements section comes next and is the most important section of this document. This section goes into detail about each specific requirement of the Bug Prediction Project. A description, use case with sequence of events, and any related requirements is given for each requirement. This section also gives a detailed description of the External Interfaces for the project including a description of the user interface for the software.

The Specific Requirements section also describes the Performance Requirements that are to be met by this Bug Prediction. Design Constraints and Standards Compliance are also considered in this section. Lastly, various System Attributes are discussed including Maintainability, Security, and Portability.

1 Introduction

1.1 Purpose

The purpose of the Software Requirements Specification is to describe the specific requirements of the Bug Prediction project that are to be met by the implementation effort of a neural network and a Machine Learning Model constructed and trained with the help of PROMISE dataset of errors. Included with the description of the requirements is a description of any constraints or assumptions that the project is working within.

This document also provides a description of any project dependencies that need to be explicitly expressed. Along with the requirements descriptions, it is also the purpose of this document to describe any performance requirements that need to be met. If there are any standards that need to be considered when developing the software are also listed.

Lastly, the purpose of this document is to communicate the system attributes of the Big Prediction software. These system attributes include reliability, availability, scalability, maintainability, and portability.

1.2 Scope

It is within the scope of the Software Requirements Specification to describe the specific system requirements of the Bug Prediction project. This would include performance requirements, system constraints, and project assumptions. Any specific detail that is needed about the standards or technology used to define these requirements, constraints, and assumptions are within the scope of this document.

It is outside the scope of this document to predict the exact errors withing a submitted code. It doesn't deal with the problem of needing to correct an incorrect code in order to make in run properly either. It is also outside the scope of this document to describe in any detail at all how certain mentioned standards or technologies work and operate.

2 Definitions, Acronyms, and Abbreviations

Table of Definitions, Acronyms, and Abbreviations

Definition, Acronym, or Abbreviation	Description
SRS	Software Requirements Specification.

3 References

Table of References

References	Description
Software Development Plan	The Software Development Plan from the Electronic Stamp project was referenced.

4 Overall Description

4.1 Product Perspective

The Bug Prediction software that is to be developed by PROMISE dataset of errors is not a complete software by itself. This Bug Prediction software by using neural networks and its requirements is only pertaining to the functionality needed to implement the initial objective i.e., predicting the correct percentage of the code.

Since the Machine Learning Model is not a complete email system by itself, the software is being developed by integrating this model with front end and by providing necessary features to load a code into the system and run it against the developed ML model in order to determine its correctness.

By doing this, the Bug Prediction software will provide the standard code-check functionality enabled with the help of ML Model along with the new functionality defined by the complete Bug Prediction project.

4.2 Product Functions

The follow is a table of the requirements that the system SHALL meet. The list of requirements was produced from the initial project documentation provided by the requirements expert.

Table of Shall Requirements

ID	Origin	Shall Requirement
19	*Project Description Document*	The user SHALL be able to create a code file in the system for checking it against the model.
20	*Project Description Document*	The user SHALL be able to log in to the system in order to use the software.
21	*Project Description Document*	The user SHALL be able to load the file of code into the system without any difficulty.
22	*Project Description Document*	The user SHALL bear a unique username which is to be generated during the process of creating an account.
23	*Project Description Document*	The user SHALL be able to run the loaded code file against the ML model to test its accuracy. The software doesn't give the exact output of the code, nor does it point to the errors withing the code. It simply predicts the extent to which the code could be correct.

ID	Origin	Shall Requirement
24	*Project Description Document*	The user SHALL be able to check the accuracy of the code after it has been run.
25	*Project Description Document*	The user SHALL be able to add comments against the code file within the project after the aforementioned code file has been run and tested. This is done for future references.
26	*Project Description Document*	The user SHALL be able to view previous history in order to check the results of previously run code files.
27	*Project Description Document*	Finally, the user SHALL be able to log out of the system and be able to save his entire work.

4.3 Constraints

The follow is a table of the design constraints that the system SHALL meet. The list of constraints was produced from the initial project documentation provided by the requirements expert.

Table of Design Constraints

ID	Origin	***Shall Requirement***
1	*Project Description Document*	*A user SHALL not be able to write the code directly into the software and run it to check its accuracy. The input can be given in the form of a code file only.*
2	*Project Description Document*	*In order to check the accuracy of any code, the user SHALL log into the software first and only then check the code. It's not possible otherwise.*
3	*Project Description Document*	*Only .java files SHALL be loaded into the system to check their accuracy. Any other type of code file such as that of C, C++, Python, C#, etc. will not be accepted by the software.*
4	*Project Description Document*	*The software SHALL not allow creation of any account if any of the sign-up details i.e., username, email id, or password are similar to any previously created account.*

ID	Origin	***Shall Requirement***
5	*Project Description Document*	*The Bug Prediction software SHALL not give the exact output of the code or the exact errors of the code as its result. Nor will it even inform us about the lines of the code that have an error. It will tell us only about the accuracy of the code with the help of a trained ML Model.*
6	*Project Description Document*	*The system SHALL not allow the user to edit or modify the code once the file has already been loaded into the system. Any changes have to be done only before the file is uploaded on the system.*
7	*Project Description Document*	*The system SHALL have no features of an Integrated Development Environment. The software tests only the accuracy and doesn't give any insights as to what the actual output of the code might be.*

4.4 User Characteristics

The following table identifies and describes the different users of the Bug Prediction software. The information gathered about the different users of the system helped define what the software needs to do. Also, these users are referenced in the requirements and diagrams.

Table of User Characteristics

User	**Description**
Programmer	A programmer is any person who writes a piece of code to execute any given task. The job of a programmer is to solve complex issues in the minimum number of times with minimum amount of memory. Therefore, a software like this could be of great importance to any programmer who wishes to verify their work and see how accurate and error-free their piece of code is.
Educationist	An educationist is a person who is involved in either the teaching sector or the learning sector. Therefore, all students and teachers fall into this category. This software could come in handy to the students as it allows them to check the accuracy of their code and make any changes whatsoever to increase this accuracy. Similarly, teachers also benefit a great deal from this software. They can verify the accuracy of the codes written by their students and can probably assign the marks accordingly, or give them suggestions so as to increase this accuracy.
Industry Experts	This software had great utility in the industry. A lot of jobs pertaining to the IT sector depend on the ability of writing a code that performs the initial task quickly and occupies as little memory as possible. With this tool, people working in the industry can differentiate between the 2 or more sets of code and go ahead with the one that has maximum accuracy and consequently, the minimum amount of errors. This would make the end product stronger and more efficient.

4.5 Entity Relationships

Figure 1 shows the entity relationships for the Electronic Stamp project.

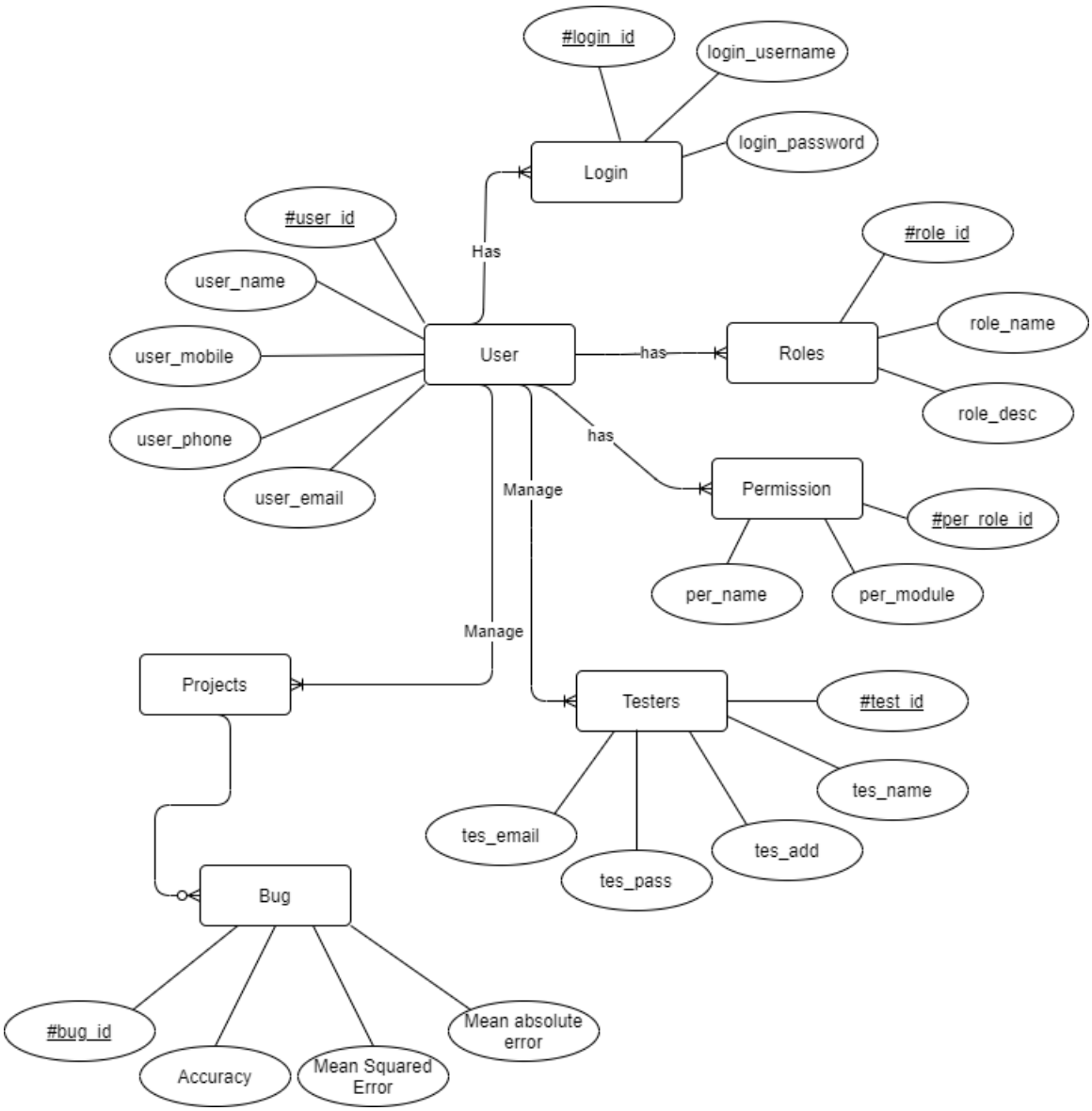

Figure 1 Entity Relationship Diagram

4.6 Data Flows

The following figures represent the data flow diagrams of the Bug Prediction software. The first data flow diagram, figure 2, is the top-level data flow of the Bug Prediction software. This is followed by the more detailed data flow of the software.

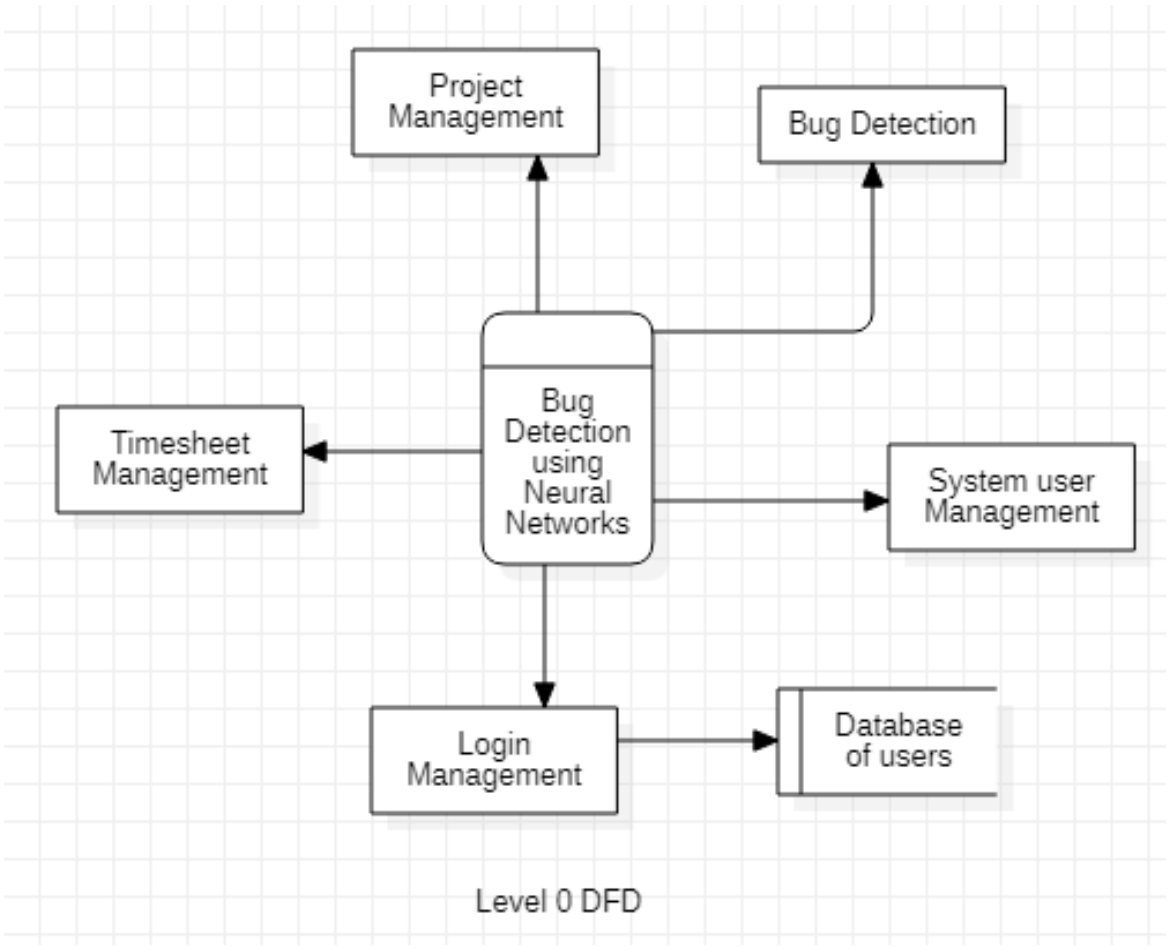

Figure 2 Data Flow Diagram (Level 0)

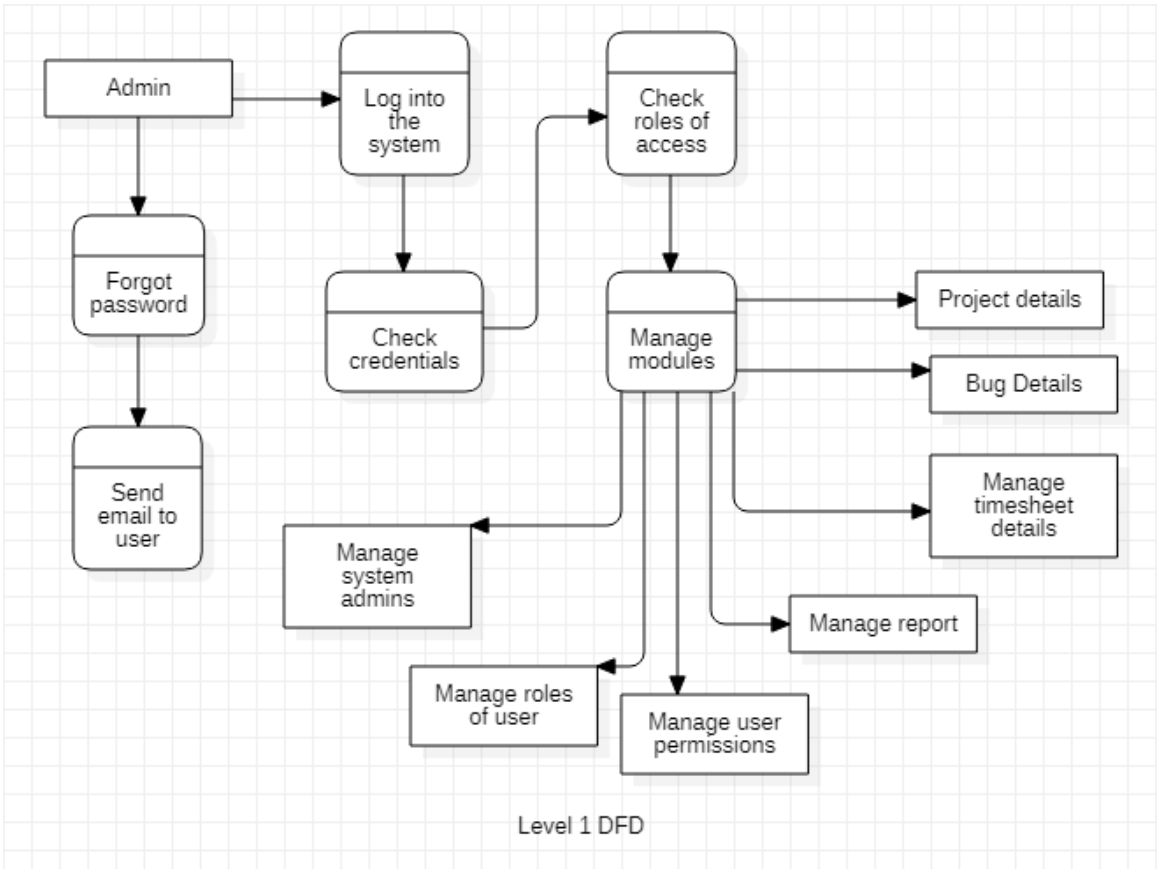

Figure 3 Data Flow Diagram (Level 1)

4.7 Data Dictionary

The follow tables in this section make up the data dictionary for the Electronic Stamp project. Using the Data Flow Diagrams, the following Data Dictionary elements were defined.

- Admin
- Project
- Bug
- Timesheet
- User
- User Permission
- Report

Table of Data Dictionary

Data Dictionary Attribute	Detail
Name	Admin
Aliases	Developer
Where Used / How Used	Authenticates user details and validates login
Description	Admin is the developer of the software and the one who validates user details and allows them to log into the system and use it. The job of the admin is to oversee how the system is working and detect glitches or errors (if any) within the software.

Data Dictionary Attribute	Detail
Name	Project
Aliases	Code File
Where Used / How Used	Interacts with ML Model (input) Accuracy is determined (output) Mean Squared Error is determined (output) Mean Absolute Error is determined (output)
Description	A .java code file is loaded into the system for authentication. It is this file that is run against the ML Model built and trained by us. This particular code file is studied and the model then predicts its accuracy, mean squared error and mean absolute error.

Data Dictionary Attribute	Detail
Name	Bug
Aliases	No
Where Used / How Used	Bugs are loaded into the .csv files that are further put together to generate the PROMISE dataset. This dataset consists of 44 such .csv files.
Description	These datasets of bugs are used to train the ML model so that it analyses the bugs present in the loaded code file and compares with those present in the dataset. It then uses this information to calculate the accuracy and efficiency of the code.

Data Dictionary Attribute	Detail
Name	Timesheet
Aliases	No
Where Used / How Used	Output
Description	Final output is displayed as a time sheet which gives the value of all three variables i.e., accuracy, mean squared error and mean absolute error.

Data Dictionary Attribute	Detail
Name	User
Aliases	No
Where Used / How Used	Load and tests a file of code
Description	User of software is the person who loads a file of code and tests its accuracy in order to determine how correct the code is. This is done by the software that runs the code against the ML model.

Data Dictionary Attribute	Detail
Name	User Permission
Aliases	No
Where Used / How Used	Determining the extent to which user can use the software
Description	User is granted permission to load a code file and test its accuracy. However, several features like editing the code after it has been loaded as well as checking the previous reports are provided only after the user has obtained required permissions from the admin.

Data Dictionary Attribute	Detail
Name	Report
Aliases	No
Where Used / How Used	Getting the final output
Description	The final report is generated once the user has loaded the file of code and tested it against the ML model. The report gives an insight about how accurate the code is. It could be uses to determine whether or not the code has any practical usage (nil for codes with low accuracy as they might take up larger memory and longer time to execute). The report gives three types of final results. The first one is the Accuracy of the code. Second is the Mean Squared Error, and the third is the Mean Absolute Error.

4.8 State Transitions

Please refer to Appendix A for the State Transition Diagram. It explains the process of loading a file of code and testing its accuracy with the help of ML Model.

4.9 Assumptions

The following table lists the assumptions made by the requirements that define the Electronic Stamp software.

Assumption	Description
Accurate ML Model	Although we have incorporated a tremendous number of datasets to train our model, there is always scope for more accuracy. If we increase the epoch value, we can get higher accuracy but that would make computation slow. Therefore, for the purposes of this project, we have kept the epoch value as 3 and assumed the ML model to be accurate.

5 Specific Requirements

5.1 System Features

5.1.1 Introduction

The Bug Prediction software shall allow a user to upload a file of code into the software for testing. The user must ensure that the code is written for java files only since the software cannot run accuracy tests on any other type of file. This test file is then checked for its accuracy with the help of our trained ML Model.

5.1.2 Functional Requirements

Purpose: Testing the accuracy of the code and predicting if it should be used for any software development activity. This is because only fast and efficient codes that take up less memory should be used.

Input: .java code file which is loaded into the system that checks it against our ML Model.

Processing: The ML model which has been trained by accounting for different types of bugs using PROMISE dataset is used to determine the accuracy of the loaded code file. The Model checks the code file and determines the bugs that are present in it.

Output: The ML Model performs several computations on the loaded code file and determines the accuracy of the code along with Mean Absolute Error and Mean Squared Error.

5.2 Stimulus Response

User Actions	System Actions
(1) Tries to login	
	(2) Processes request
	(3) If invalid, denies login
	(4) If valid, allows login
(5) User is logged in	
(6) Load a file of code	
(7) Submission of code file	
	(8) ML Model is loaded
	(9) Checks the code file
	(10) Determines bugs
	(11) Tests Accuracy
	(12) Tests Mean Squared Error
	(13) Tests Mean Absolute Error
(14) Final output is displayed to the user	
(15) User tries to log out of the system	
	(16) Processes request.
(17) User logs out.	

5.3 Performance Requirements

The following tables list the performance requirements of the Bug Prediction software.

Table of Performance Requirements

Performance Requirement	Description
Highly trained ML Model	The testing ability of the ML Model should be significantly higher in order to make the actual results close to accurate and insightful. If the model is not trained enough, the results produced by it will be skewed and meaningless.
Appropriate result generation	The Bug Prediction software will test the code and generate the output as its accuracy, mean squared error and mean absolute error.

5.4 Design Constraints

The Bug Prediction project is being run on visual studio for generating the front end and back end. The ML model has been constructed on Jupyter but for the purposes of presentation, the model is being run on Kaggle. The code along with the PROMISE dataset has been added to Github as well.

Table of Design Constraints

Design Constraint	Description
Visual Studio	The code for front end as well as back end has been executed in Visual Studio.
ML Model	The ML model has been constructed on Jupyter but for the purposes of presentation, the model is being run on Kaggle. It has been prepared by training it with 44 .csv files containing information about several types of bugs present in the PROMISE dataset.
Java	The software can take only .java code files as input. It cannot determine the accuracy of any other code file.

5.5 Software System Attributes

5.5.1 Reliability

Reliability in the Electronic Stamp software will be ensured by the efficiency of the ML Model. Therefore, the model has to be thoroughly trained in order to make the output more accurate. By adopting Spiral Model for development of this product, it will be ensured that all user reviews from alpha and beta testing are taken into account. Customer reviews will also be properly monitored in order to update the product as per the needs of the user.

5.5.2 Security

The Bug Prediction software will utilize authentication feature therefore no will be able to access the code files of another person without proper credentials.

5.5.3 Maintainability

The Bug Prediction software extends the functionality of our trained ML Model. The entire code for the software is written in Jupyter and Visual Studio. The language used for programming is portable. This promotes good design practices due to the inherent structure of a Java program.

Along with the well-formed programming enforced by the programming language, best practice development conventions will be enforced for the construction of the Bug Prediction software. These will include adequate commenting within the source code that complies with and uses the automated Java documentation standard so that source code documentation will be able to be automatically generated. Consistent variable naming conventions will be used by all the programmers. Consistent spacing will be used in the source code by all the programmers.

5.5.4 Portability

As mentioned above, the Bug Prediction software extends the functionality of the ML Model. The Bug Prediction software will be written in a well-formed programming in order to gain the portability providing by that language.

Portability is the ease with which a software system can be transferred from its current hardware or software environment to another environment. Portability requirements address the user concern for how easy it is to transport the system. After developing the entire product, it will be fairly easy to transport the product as well as its source code from one system to another. The software will continue to work fine as long some tweaks depending upon the system in which the product is being transported to are done.

APPENDIX A: State Transition Diagram

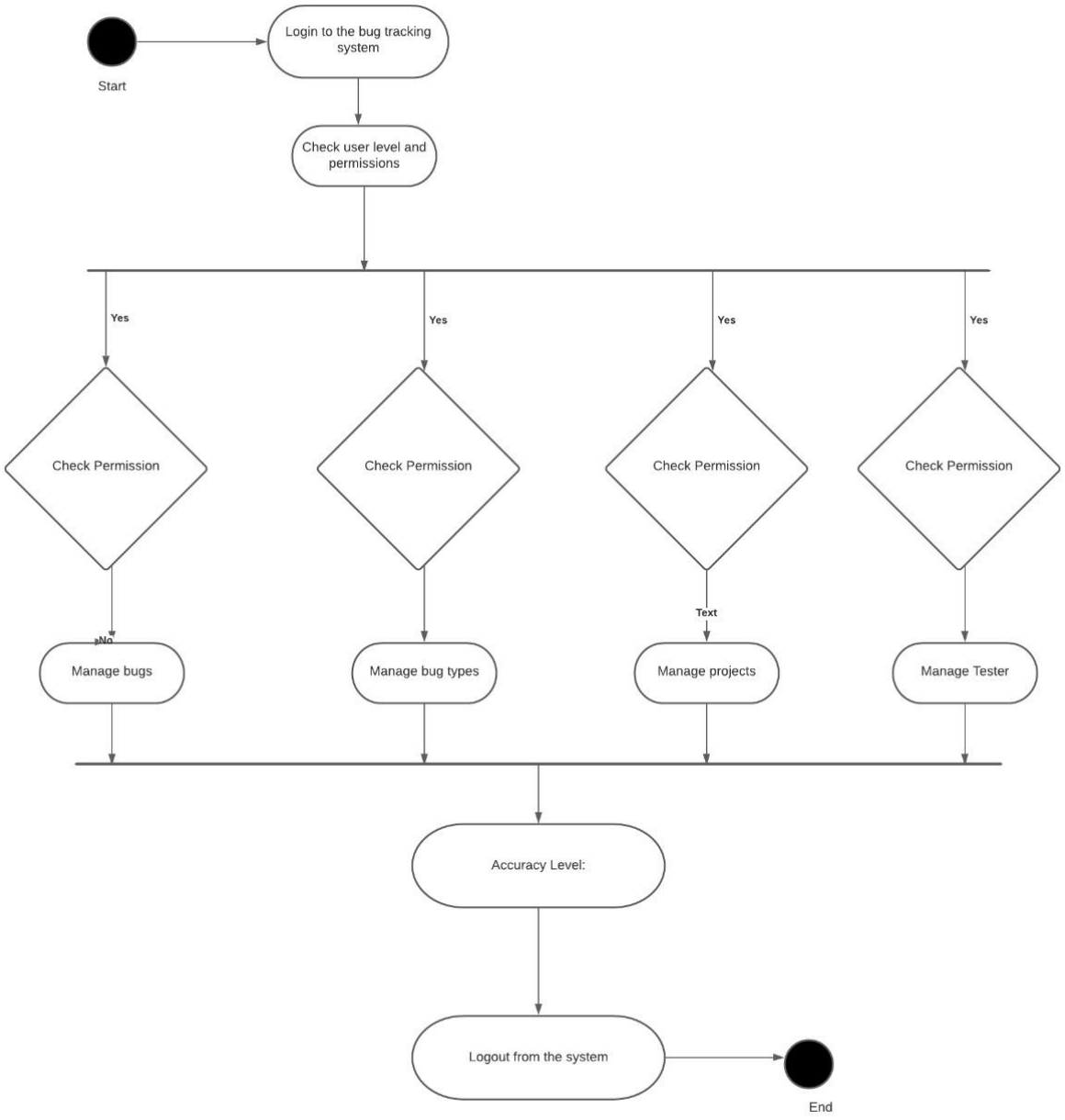

Figure 4. State Transition Diagram

4. DESIGN APPROACH AND DETAILS

4 .1 Decomposition Description

4.1.1 Module Decomposition

The Bug-Prediction Software has been decomposed into the following modules.

- Login Module: The user can either sign up or log in to the server before the testing process begins.
- Execution Module: This module executes the code and collects the dataset that consists of various parameters required for identifying a software such as Weighted Methods for Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), etc. from the user to be used by the software and merges all the .csv files into a dataframe.
- Normalization Module: This module applies min-max normalization to scale the data. It also applies random oversampling.
- Squeezenet Module: This module applies Squeezenet Neural Network in order to train the model.
- Evaluation Module: This module collects the results from the software and gives the output in the form of percent accuracy, mean squared error and mean absolute error.

4.1.2 Concurrent Process Decomposition

A complete view of the project suggests that there are two processes, the input process and the software testing process. In the input process, the code interacts with the user and takes a dataset as the input.

This dataset is then fed into the testing software which normalizes it and predicts the accuracy of bug-prediction along with mean squared error and mean absolute error.

4.2 Dependency Description

4.2.1 Module Decomposition

The Bug-Prediction Software has been decomposed into the following modules.

- Login Module: The user can either sign up or log in to the server before the testing process begins.
- Execution Module: This module executes the code and collects the dataset that consists of various parameters required for identifying a software such as Weighted Methods for Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), etc. from the user to be used by the software and merges all the .csv files into a dataframe.
- Normalization Module: This module applies min-max normalization to scale the data. It also applies random oversampling.
- Squeezenet Module: This module applies Squeezenet Neural Network in order to train the model.
- Evaluation Module: This module collects the results from the software and gives the output in the form of percent accuracy, mean squared error and mean absolute error.

4.2.2 Inter-module Dependencies

4.2.2.1 Independent Modules

No modules are independent in this bug-prediction project. The modules rely on other modules to initiate them or to provide data.

4.2.2.2 Dependent Modules

The following modules are dependent on one another for their functioning.

- Execution Module: This Execution module runs successfully only when the program is run and the dataset is fed into it. For this module to complete its function, the dataset is absolutely necessary.
- Normalization Module: This module is executed when the code receives the dataset. It applies normalization techniques to the dataset to scale the data.
- Evaluation Module: This module depends on the dataset of the type of bugs as .csv files. It gives an output about the accuracy of bug prediction.

4.2.3 Inter-process Dependencies

There exist inter-process dependencies between Normalization module and Squeezenet Module. With the help of Squeezenet module, the generated images are put into the Convolutional Neural Network (CNN) for generating the output. This task must be preceded by min-max normalization of the dataset of .csv files.

4.2.4 *Data Dependencies*

The following Data Flow Diagram shows the data dependencies between the various entities and modules.

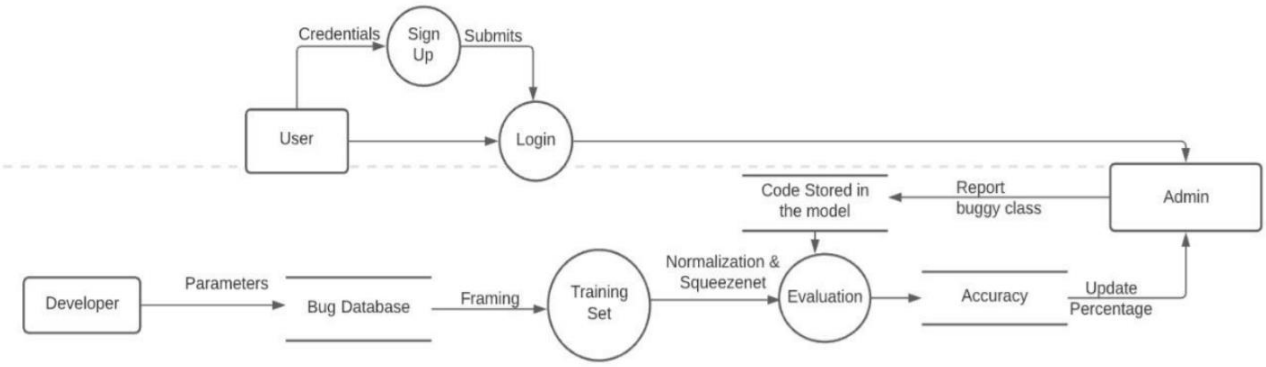

Figure 5, Data Flow Diagram

4.3 Interface Description

4.3.1 Module Interface

4.3.1.1 Login Module Description

a) User Interface Design

Figure 6, Login Module

b) Description

The first screen you get when you fire up the software is that of the login module. Here, the user can either log in, or sign up to create an account. Once the details are saved, the user proceeds to the next page.

4.3.1.2 Input Module Description

a) User Interface Design

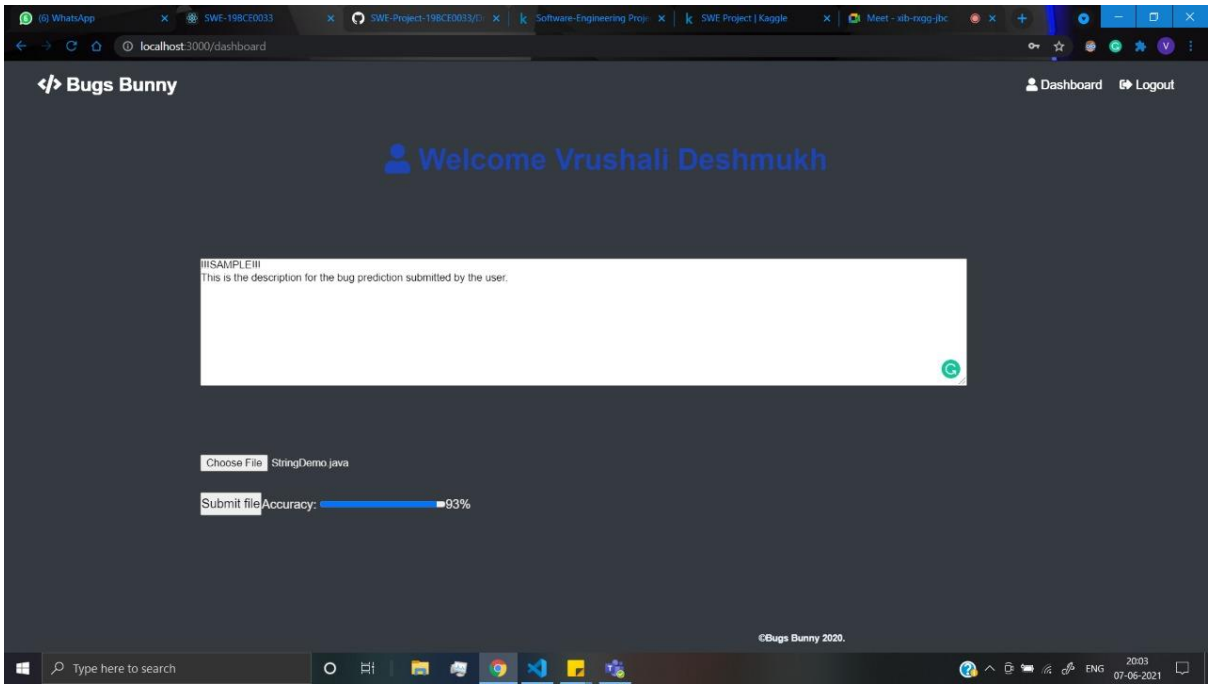

Figure 7, Input Module

b) Description

The following figure shows the user interfaces for this module. The interface is interactive and takes an input code from the user. After entering the code, the user can press submit. There is also a “Delete” button to remove all the text from the box.

4.3.1.3 Execution Module Description

a) User Interface Design

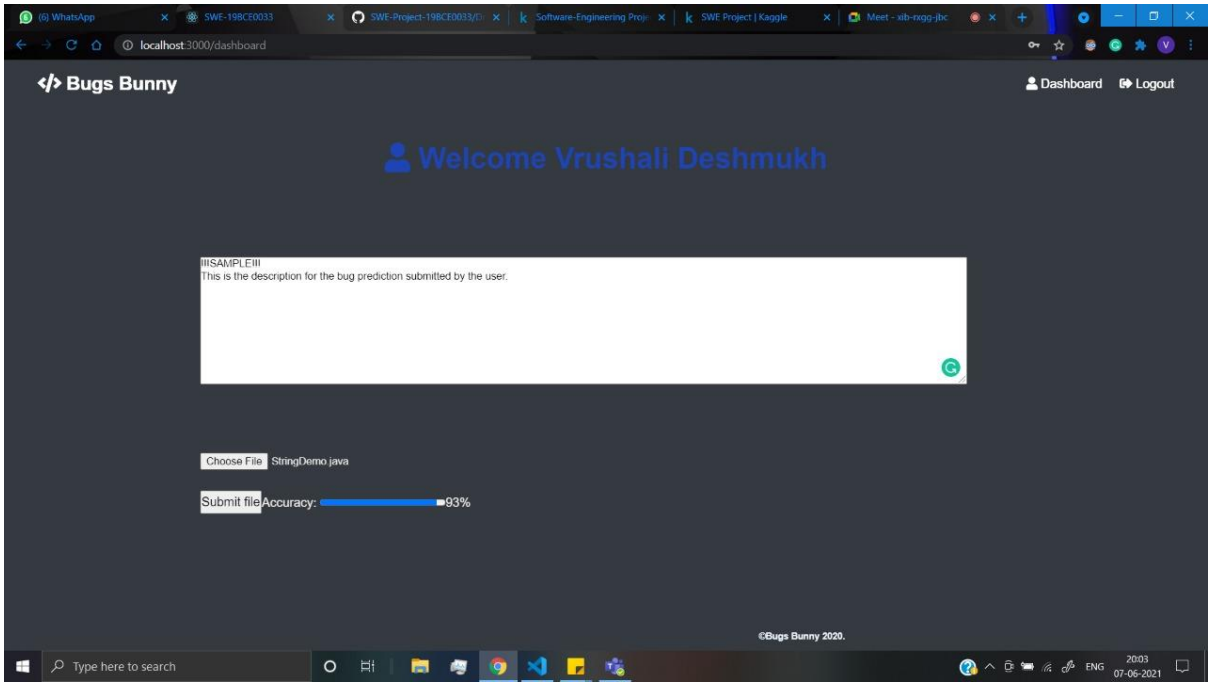

Figure 8, Execution Module with percent accuracy as output.

a) Description

The User Interfaces give the output needed in the project. It gives the information about the percent accuracy in detecting bugs. For the purposes of this SDS document, we have not yet shown Mean Square Error and Mean Absolute Error in the output section however, that will be incorporated in later reviews.

Other than the output accuracy for a certain input, there is also an option to check the previous test reports of earlier inputs. A list of these test cases along with their resultant output accuracy are also displayed in this representation.

4.3.2 Process Interface

4.3.2.1 Input Process Description

This module takes the input code from the user. The software model then runs this code against its database and tries to determine the extent to which the code may be bugged.

4.3.2.2 Execution Process Description

The primary objective of this module is to execute the model and collect the training dataset that consists of various parameters required for identifying bugs in a software such as Weighted Methods for Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), etc. from the user to be used by the model and merges all the .csv files into a dataframe. This process helps in training the model to enhance its bugs predicting ability.

4.3.2.3 Normalization Process Description

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

This module applies min-max normalization to scale the data. It also applies random oversampling. Min-max normalization is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.

We used random oversampling to balance the dataset as it contained some irregularities. Random oversampling duplicates examples from the minority class in the training dataset and can result in overfitting for some models.

4.3.2.4 Squeezenet Process Description

This module applies Squeezenet Neural Network in order to train the model. Squeezenet is a deep neural network for computer vision i.e., it helps in generating images of arrays that store the information about bugs. These images are serves as an input in a Convolutional Neural Network (CNN).

4.3.2.5 Evaluation Process Description

This module makes the use of deep neural network to feed output array images to CNN and generate an output that tells us about the efficacy of the bug-prediction model. Along with these array images, the output also consists of the percent accuracy of the input module, mean squared error, and mean absolute error.

4.4 Class Diagram

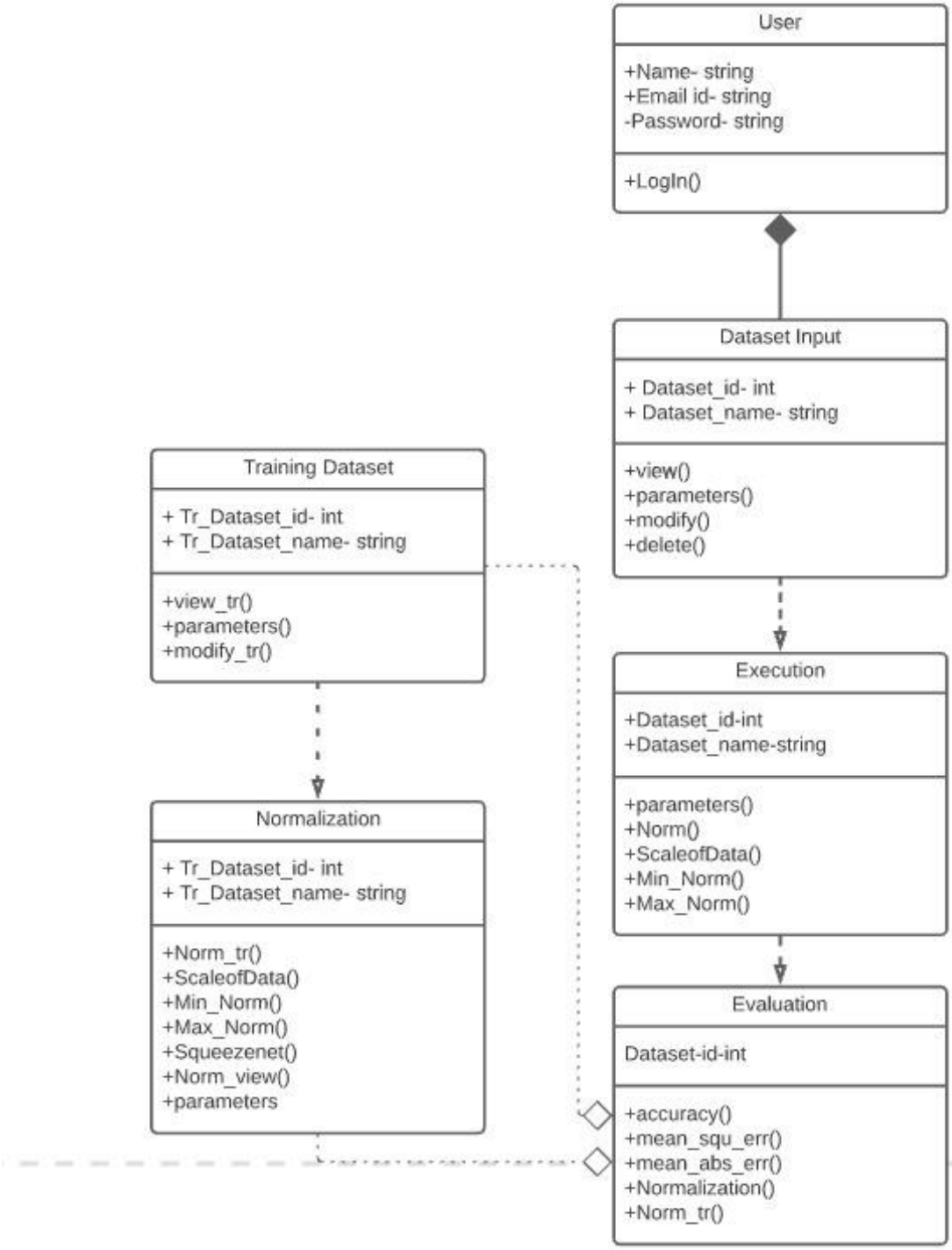

Figure 9, Class Diagram

4.5 Collaboration Diagram

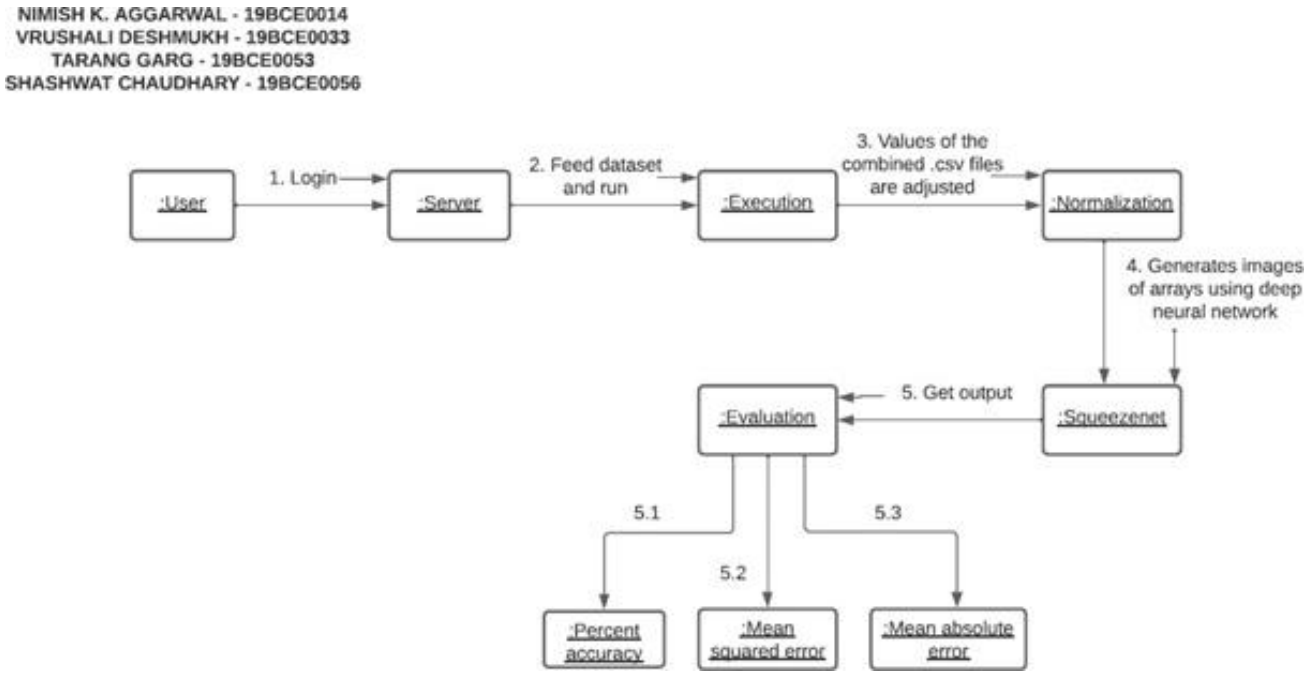

Figure 10, Collaboration Diagram

2. Control Systems Diagram

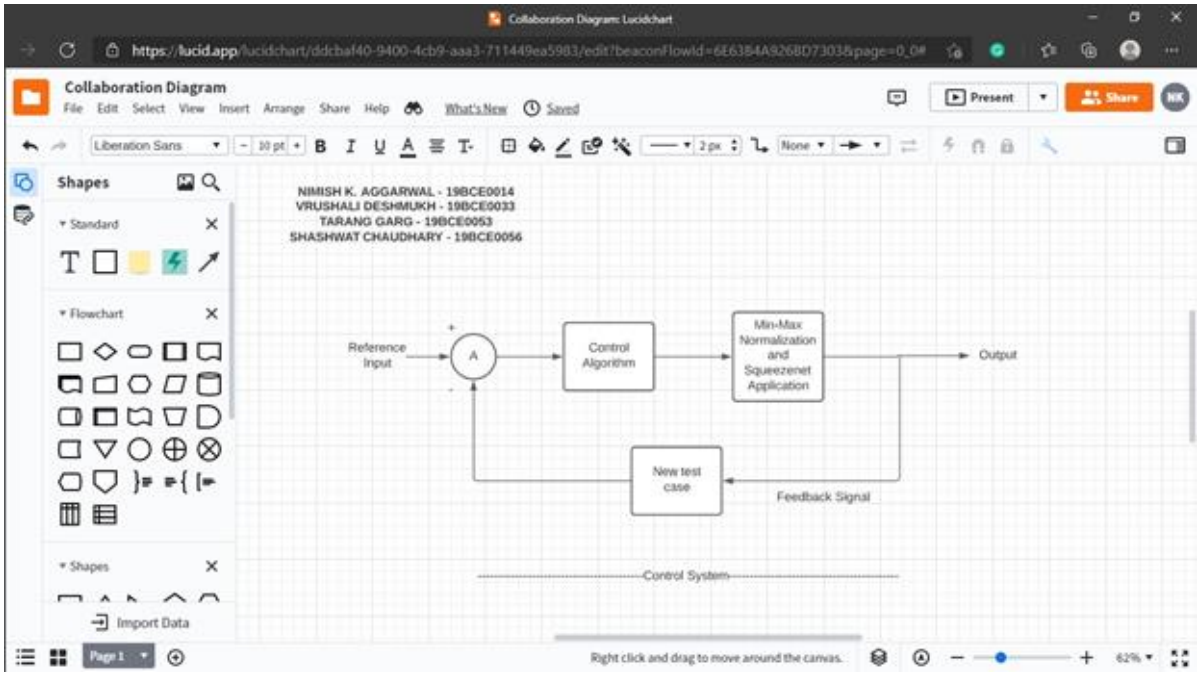

Figure 11, Control Systems Diagram

3. sequence Diagram

1. Log In

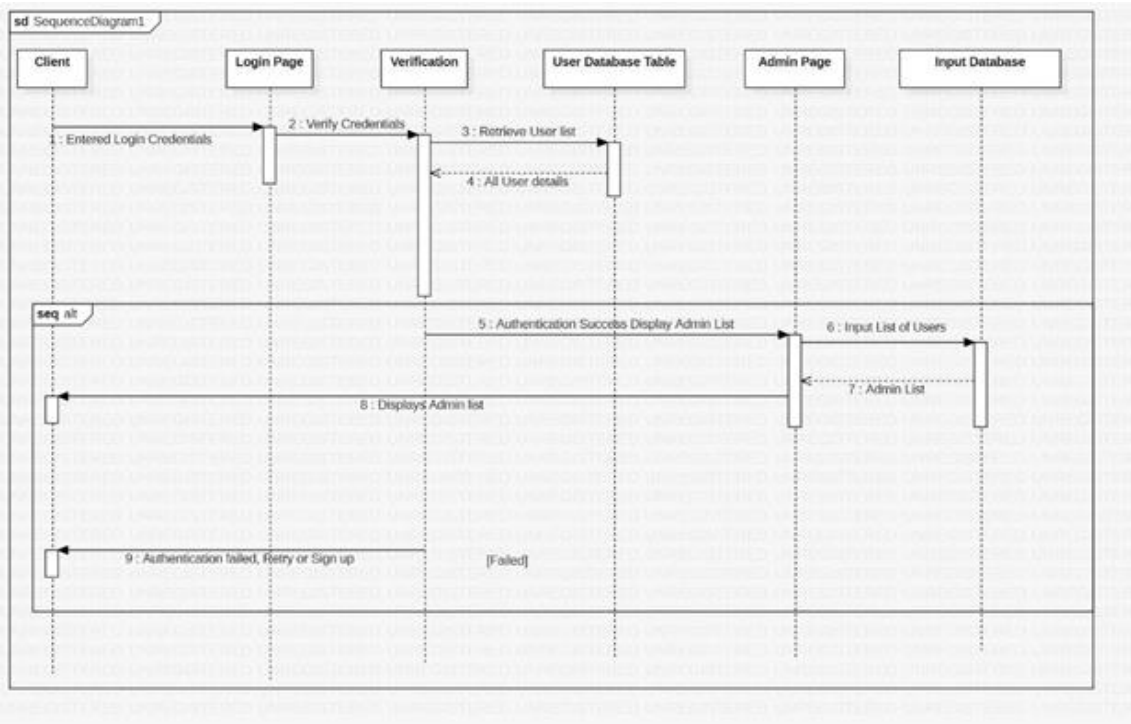

Figure 12, Sequence Diagram for login

2. Sign In

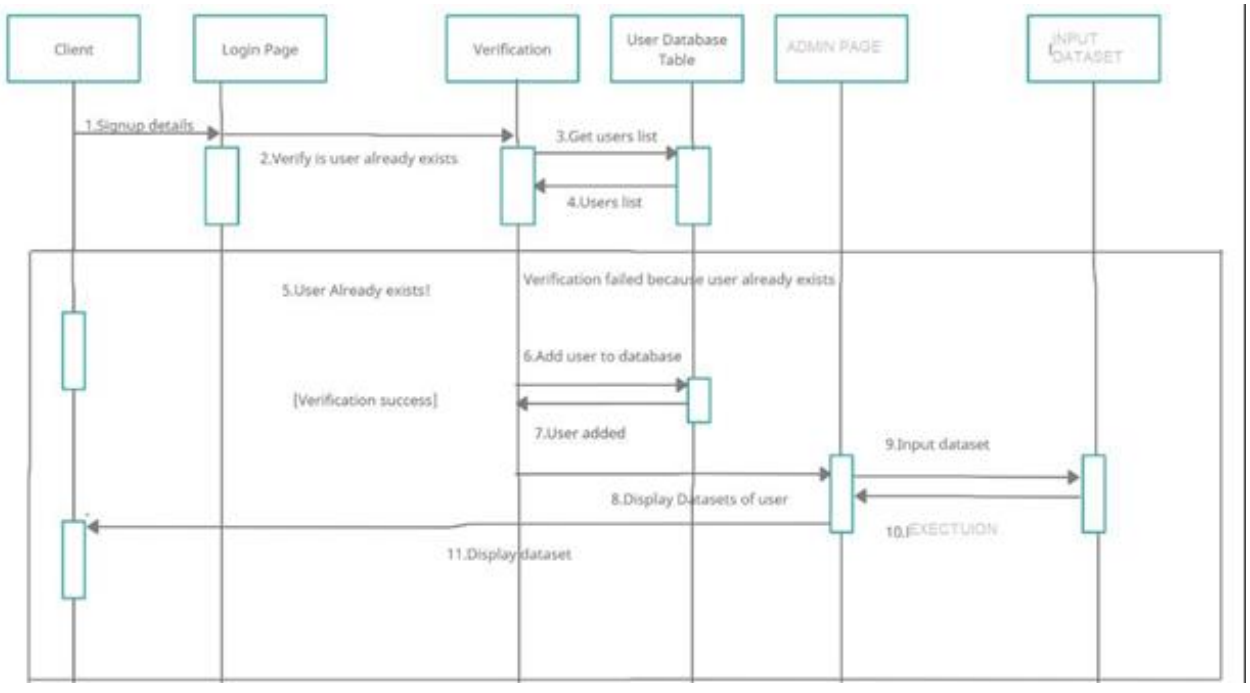

Figure 13, Sequence Diagram for sign in

3. Execution

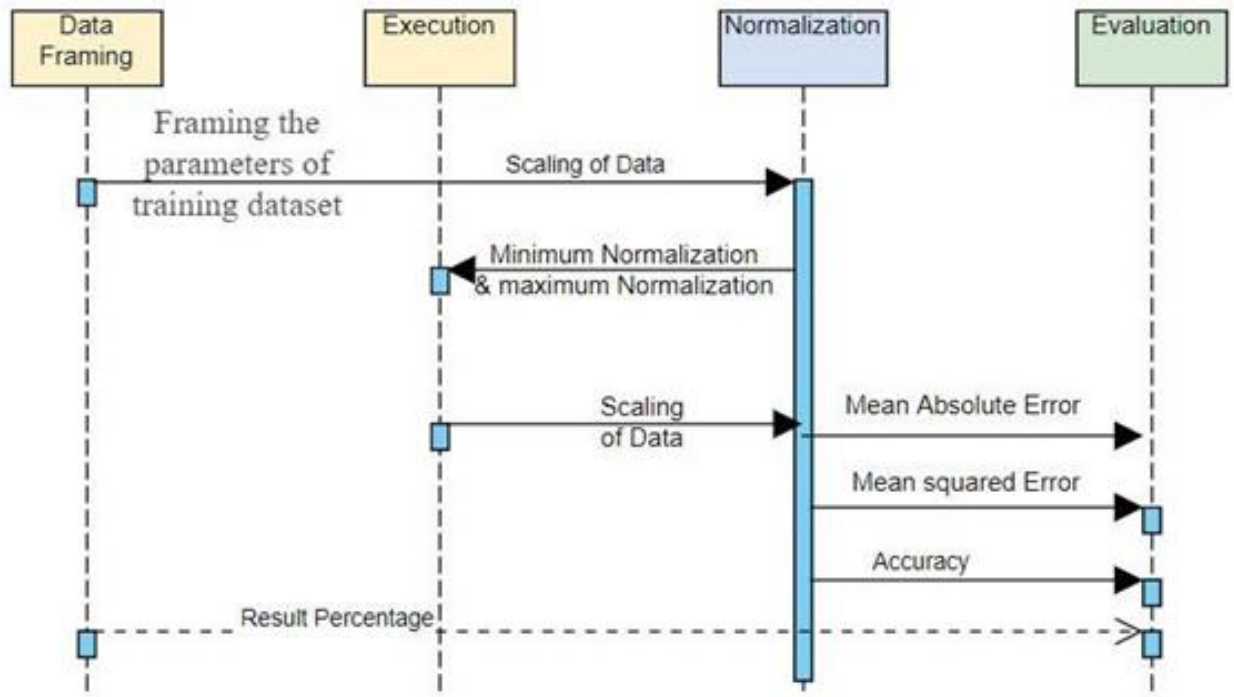

Figure 14, Sequence Diagram for Execution

*All representations have been done using Lucid Chart

6.PROJECT DEMONSTRATION

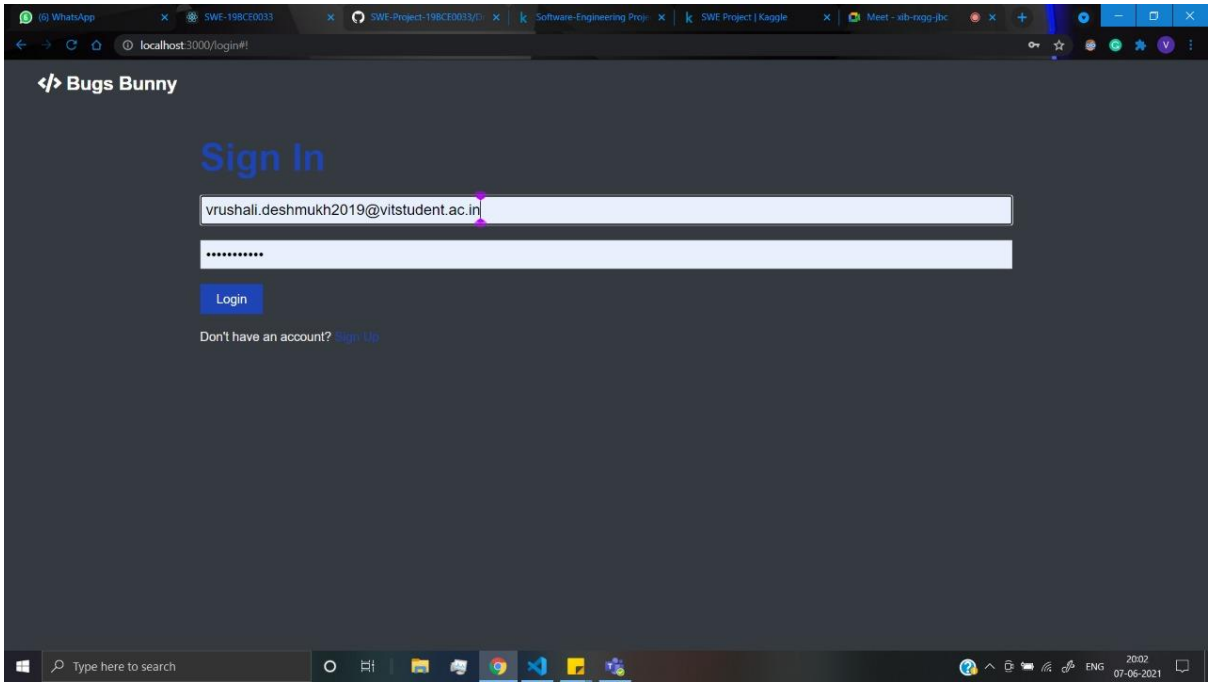

Figure 15, Login Module

Test Case ID	Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
1.1.1	Checking login by an existing user	Email Id and Password	Login Successful	Login Successful	Pass
1.1.2	Incorrect credentials entered by a user	Email Id and Password	Login Failed	Login Failed	Pass

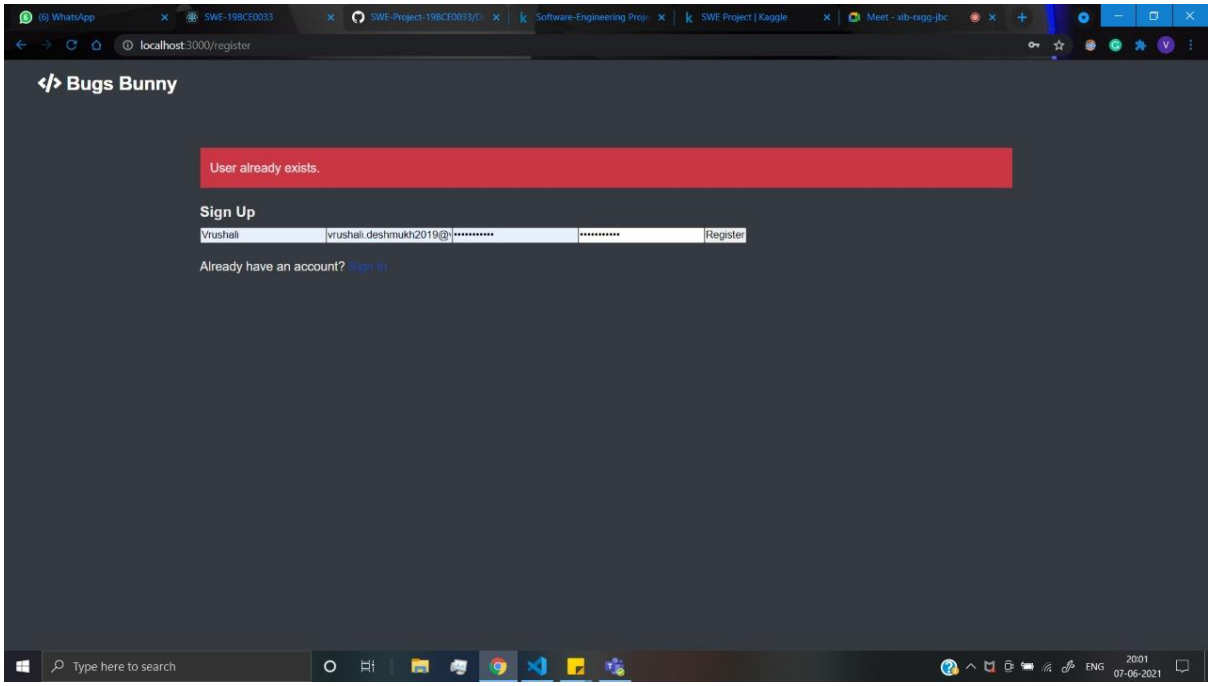

Figure 16, Sign Up Module

Test Case ID	Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
1.2.1	Sign Up by a new user	Username, Email Id and Password is entered by the new user	Login Successful	Login Successful	Pass

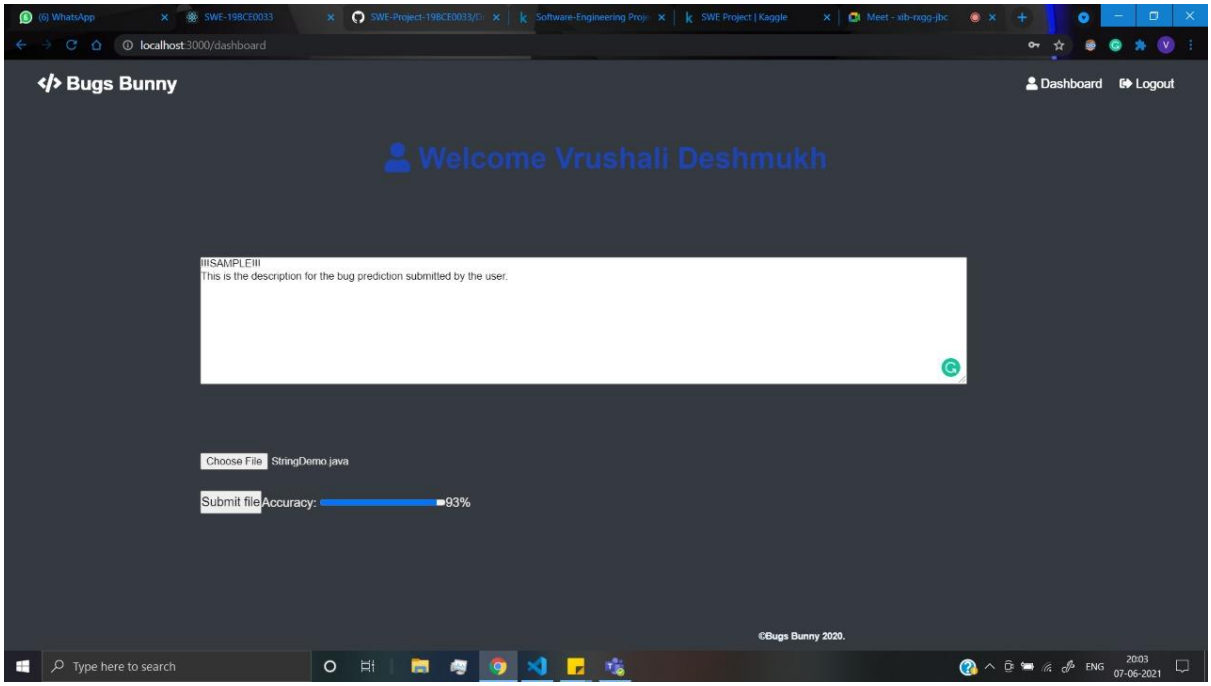

Figure 17, Input Module

Test Case ID	Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
2.1	Testing the IDE	Input code	Text box taking the input	Text box taking the input	Pass
2.2	Submitting the code	Input code	Submission successful	Nil	Fail
2.3	Deleting the entered code	Input code	Blank text box	Blank text box	Pass
2.4	Load an input file of code by using the file button	Code file	Entire code is displayed in the text box	Entire code is displayed in the text box	Pass

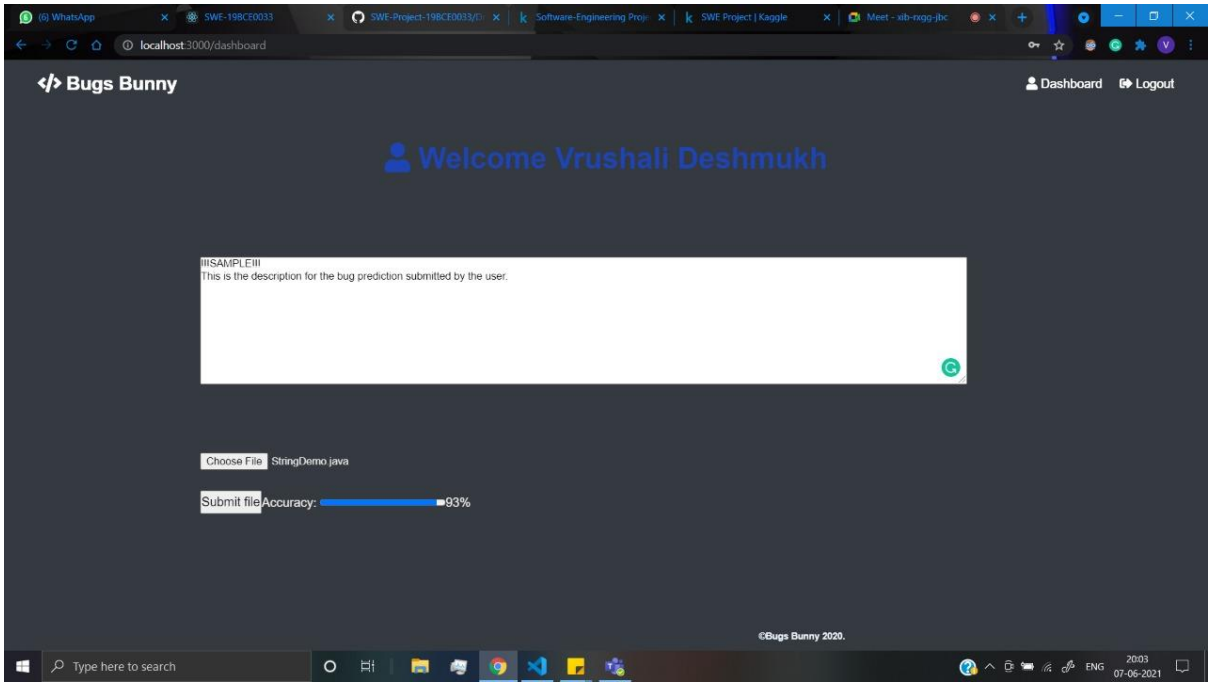

Figure 18, Output Module – Adding notes

Test Case ID	Test Objective	Test Data	Expected Results	Actual Results	Test Pass/Fail
3.1.1	Writing additional details	Notes	Text box working	Text box working	Pass
3.1.2	Attachment of files	Code file	File loaded successfully	nil	Fail
3.1.3	Percentage Output	Input code	Percentage of code that is bug free	Percentage of code that is bug free	Pass

7. COST ANALYSIS

It saves a lot of man power as well as money because of bug prediction at an initial stage. Cost reduction is extremely important in order to make the project efficient. This is why our project is important since it proves to instrumental in reducing tremendous costs.

RESULT & DISCUSSION

The result is given in the form of Accuracy, Mean Absolute Error and Mean Squared Error. This ML Model is highly usable in order to determine the accuracy of the code and test its efficiency in the initial stages itself.

The GITHUB link of the project has been attached below.

GITHUB Link:

https://github.com/vruuushali/SWE-Project-19BCE0033

ML Model:

https://www.kaggle.com/vruushali/software-engineering-project/

Video:

https://drive.google.com/file/d/1MuMZcAwaSqCyE2RREKeCAw4rUw6VaAGo/view?usp=sharing