**Chapter 7 – Testing**

# 7.1 Chapter Overview

All the details regarding how successfully implemented the project have been addressed in detail in the preceding chapters of our group project Zeta. The testing phase of Project Zeta is discussed in this chapter. The testing criteria will be described when the goals and objectives of testing have been discussed. It will then go into how to test Zeta's functional and non-functional requirements to ensure that it meets all the requirements. This chapter will conclude with a full analysis of the limits encountered during the testing procedure, as well as an assessment of the test results.

# 7.2 Goals and objectives

The testing stage was used to ensure that the prototype operates as expected based on the requirements obtained during the requirement elicitation phase.

The following are the testing objectives for Zeta's implemented prototype:

* Verify functional requirements
* Verify non-functional requirements
* Identify any defects and bugs in the system to guarantee they are resolved before the final product
* Enhance and improve the system based on test findings

# 7.3 Testing Criteria

To carry out the testing stage for this project prototype, a solid test plan table is established and worked on; hence, the two functional and non-functional requirements utilized are displayed in detail. To validate the test coverage of this application, test cases for the system's fundamental functional and nonfunctional requirements were identified. These test cases will specify a collection of conditions that will determine whether the developed system Zeta is functioning properly as expected.

# 7.4 Testing Functional Requirements

The functional requirement testing was carried out to verify the requirements for Zeta Application incorporated using the white box testing approach. As per the successful development/implementation of our software application detailed in the previous chapter more test cases are provided to support our development as shown in the figure below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case No.** | **Feature Tested** | **Test Case Description** | **Test Case Condition** | **Expected Result** | **Actual Result** | **Status of the result** |
| 1 | Feature 1 | The ability for the user to log in | Successfully was able to log in to the web application | The ability for the user to log in | Successfully was able to log in to the web application | Pass (100%) |
| 2 | Feature 2 | Email Clustering Feature | Made it successful to add emails into the application including Spam folders | The ability to add get emails and categorize them into the Spam and the main inbox | Successfully was able to make this feature display on our application | Pass (98.5%) |
| 3 | Feature 3 | Automated reply suggestions to emails | This feature enables help to suggest finishing emails | The ability to suggest emails during the composition | Successfully was able to display and suggest lines to emails | Pass  (-%) |
| 4 | Feature 4 | Email prioritization using the system of color code system | This feature enables the user not to miss any important deadlines | The ability to remind about upcoming meetings and deadlines | Successfully added this feature | Pass (98.5%) |
| 5 | Feature 5 | Email summarization | In progression to develop this feature | Successfully implement this feature | In the progress of developing | In Process |
| 6 | Feature 6 | Read Aloud Feature | Consists of Read Aloud Feature | Make it visible to use this feature on the body of the email | Implemented 50% and yet to develop it further | In progress  Pass (90%) |

# 

# 7.5 Testing Non-Functional Requirement

The system's non-functional requirements are detailed in this section, and testing for them will be conducted below, based on accuracy, performance, reliability, usability, and scalability. More non-functional test cases were conducted, and all relevant details and information are detailed in this section for accessibility.

Testing for this application was done backend of the system to determine the used algorithm's accuracy, execution time, precision, and recall for the web application. In detail, the Zeta web application consists of two main algorithms, the Spam classification function and the Automated reply suggestions for the email during the email composition process.

An overview of this terminology that is used in the Zeta web application will be detailed in this section and the reason why the two selected algorithms are implemented is because of featuring the best output for our end-user as expected.

Starting with the first algorithm used in this implementation, the Spam classification. The ultimate theory behind why the Zeta application consists of this is that as an individual or a group of individuals, it is mandatory for them to keep their inboxes as clean as possible without filling their ultimate communication tool with unwanted but to fill in with necessary and most important emails. This makes everyone’s work-life easier and much more saving time and stress for them.

The concept behind Spam classification consists of creating a spam classifier using BERT. The following few steps to implement this in the project and this is described as follows.

First and foremost, loading the data in CSV files that are categorized into folders namely ham and spam files. Secondly, performing the EDA model helped to identify what the data collected looked like. Thirdly, Pre-processing is going to happen, and this step is undertaken to identify and create it more model friendly. As the final step is the functional API is implemented to develop the deep learning model. The functional API capability is to create models with both flexibility and complexity in traditional sequential models.

Furthermore, the implementation of spam detection using the BERT concept includes downloading a suitable dataset few libraries that are known as TensorFlow Hub where the TensorFlow pre-trained model is stored, TensorFlow for model creation, Pandas for data loading, manipulation, and wrangling, TensorFlow test addition of NLP test processing, SkeLarn for performing the data splitting, Matplotlib for visualization support. For the first time running the collected dataset, it has shown a 15% of the data needs to balance. Moving forward with collecting two new datasets namely ham and spam the filtrations turned out to be successful.

During the preprocessors the one got encoding category cannot be understood in text formats hence to resolve this an assignment of integer label to the class of ham and spam can be referred to as 0 and 1 accordingly and storing this information new column spam is where the Hot-Encoding is performed.

* 1 – If a category is a ham/ not spam
* 0 – if the category is spam

After done with a successful data procession incrementing this to the model is left but in this case, it is identified that the developed model will take place to learn the patterns of the data, and during the evaluation process it is predicted to have the accurate results. Moving forward with this Train Test Split Strategy is detailed It states that a large portion of all data (population) is used for training (approximately 80%), with inputs (Messages) and labels (1/,0), and the remaining 20% is not used. We can just predict that 20% and see how it performs when we do further evaluation because the model is no longer biased.

Now since the preprocessing comes to an end look at the model creation that has been developed. The usage of Keras Functional API to build our model after downloading the Bert model is processed. After this step, the compilation of the training model is started this is done the Adam as the optimizer and binary cross-entropy as the loss function. The final stage is evaluating a model. To evaluate the model performed, simply feeding it testing data and labels and used the model's evaluation method to get a rough estimate of how it performed. Looking at the output for the following It's like training results, which can lead to incorrect model interpretation.

So, we'll need a better way to see how our model is doing, and classification reports and confusion matrices are usually the way to go. The use of sklearn confusion matrix and classification report, which takes in actual labels (y test) and predicted labels (y pred) and returns an array of numbers that we will plot using seaborn's heatmap and matplotlib in the Plotting Confusion Matrix and Classification.

Furthermore, looking at the model prediction everything worked out fine as expected shown in a shortcode.

array([0, 0, 0, 1, 1, 1], dtype=int64)

In this section, the second algorithm and data to get and perform the automated reply function in the email application are discussed.

NLTK known as the Natural Language Toolkit is used because of its greatest efficiency wrapped in teaching and working in computational linguistics implemented using the python programming language. After succeeding in installing the NLTK package the NLKT packages also must be installed. The text pre-processing with NLTK is to make it ideal for the work in which the preprocessing consists of conversation of uppercase or lowercase also tokenization.

A pre-trained Punkt tokenizer for English is included in the NLTK data package and they are Removing Noise, Removing Stop words, Stemming, Lemmatization.

After the start of the preprocessing phase is done the next step was to transform the text into a meaning vector or array of numbers and that is known to be the Bag of words stage undertaken. Due to the problem of this approach that was found the TF-IDF Approach was considered. This is to determine the frequency of words by how often they appear in a particular document. For example, “they” are the most frequently used in the documentary.

In information retrieval and text mining, the Tf-IDF weight is commonly used. This weight is a statistical metric for determining the importance of a word in a collection or corpus of documents.

The TF-IDF transformation is a vector space transformation that is applied to texts to produce two real-valued vectors. The Cosine similarity of any pair of vectors could then be calculated by dividing their dot product by the product of their norms.

Using this formula, we can find out the similarity between any two documents d1 and d2.

Cosine Similarity (d1, d2) = Dot product(d1, d2) / ||d1|| \* ||d2||

After this process, the reading data takes place along with preprocessing of the raw information by defining a function called LemTokens which is intended to input the token and return the normalized token. For keyword match function for an automated response's greeting is tested. Finally, Creating a Responses

The concept of document similarity will be used to generate a response from our automated responses for input questions. As a result, we begin by importing the required modules.

* Import the TFidf vectorizer from the sci-kit learn library to convert a collection of raw documents into a matrix of TF-IDF features.

import sklearn.feature extraction.text from sklearn.feature extraction.text TfidfVectorizer

* Also, from sklearn.metrics.pairwise import cosine similarity import cosine similarity import cosine similarity import cosine similarity import cosine similarity import cosine similarity import cosine similarity import cosine similarity import cosine\_

Finally at this stage, depending on the user's input, the lines that are wanted will be automated responses to say when starting and ending a conversation.

# 7.6 Unit Testing

Unit tests were performed during the development stage, and any bugs discovered during testing were addressed.

# 7.7 Performance Testing

The developed web application was tested on our internet browsers such as Microsoft Edge and Google Chrome this practice was undertaken to ensure the selected browser compatibility and responsiveness among and across all the devices. In performance testing loading times of our web application, Zeta was tested and checked whether it is performing as expected.

# 7.8 Usability Testing

Usability testing determines how user-friendly the system is and how well the user interface design and experience are retained. The Zeta web application was created with user experience guidelines in mind, with a focus on simplicity. It is one of a kind. From a fully responsive web GUI that works across all devices to a logo that was built specifically for devices and browsers that is enabled to work on any selected browser or smart devices.

# 7.9 Compatibility Testing

Compatibility ensures that the system works with the web browsers of your choice. Google Chrome and Microsoft Edge were used to test the Zeta email web application. All the features worked flawlessly in all browsers.

# 7.10 Chapter Summary

This chapter covered the testing phase of our web application Zeta, including the system's goals and objectives, testing the functional and non-functional requirements of Zeta, and the limitations encountered during the testing process. The system's critical evaluations are discussed in the following chapter.