# SE 3XA3: Test Report
# Node Messenger

Team #24, Node Messenger
Tasin Ahmed - ahmedm31
Shardool Patel - pates25
Omar Elemary - elemaryo

December 5, 2018

# Contents



# List of Tables



# List of Figures

This document ...

Table 1: **Revision History**

| **Date** | **Version** | **Notes** |
|---|---|---|
| Date 1 | 1.0 | Notes |
| Date 2 | 1.1 | Notes |

# 1 Functional Requirements Evaluation

## 1.1 User Inputs

We tested all the input fields in our messenger app. Through testing, we determined that the login and signup feature work as intended. The messenger returns an error if a field is empty while logging in or registering. It returns an error if the email entered is invalid, if the passwords do not match, and if the password is less than 6 characters. While logging in, the console returns errors correctly if the email and password entered are incorrect. While adding people to a new conversation, the messenger ensures that an email is entered and that it is valid. The messenger prevents the user from sending a message if the message box is empty. Pressing 'Enter' or clicking the 'Send' button both send messages correctly. By testing these user inputs we are able to ensure that the core functions of our messenger work as intended.

## 1.2 Data Storage

Testing was done to ensure that data sent from Node Messenger was being stored correctly in the firebase database. Once a user creates a new account, Node Messenger correctly stores their Name, Email, and Password in the database. After a message a sent by the user, it is sent to our database and stored. The database stores the messages, the time it was sent at, the chat it was sent to, and the user it was sent by. Our database consists of 3 collections: Messages, Users, and Conversations. By testing, we made sure that the information being sent to the database gets stored into their respective collection.

## 1.3 Data Retrieval

Node Messenger is able to look through the firebase database, and retrieve the necessary data. This is done when a new user is signing up to the website. The messenger must look through our database and return an error if the user email already exists in our database. A similar procedure occurs during login. The messenger must take the entered email and password, and allow the user to login if the credentials match that on the database. The messenger is able to display the correct messages to the correct chat, display the name of the user it was sent by, and the time it was sent at. When adding a user to

a conversation, the messenger again looks through our database and ensures the email address that was entered is valid.

# 2 Nonfunctional Requirements Evaluation

## 2.1 Look and Feel

For the look and feel of the messenger, we wanted it to look visually appealing. For the theme of the messenger, we tried analogous, complementary, triad, and many shades of color, and felt our current theme suits our messenger the best. After hosting our messenger on Github Pages, we shared the URL with many of our classmates, family, and friends. Many of them provided us with valuable feedback that helped us to further improve our messenger to suit the needs of the majority of the users. Most of the feedback we got from them was very positive and assured us that our messenger achieved the goal of being user-friendly.

## 2.2 Usability and Humanity

Node Messenger is made to be catered towards everyone. To achieve this, we used one of the main methods of software design – Abstraction. This means if a user has used any messaging apps before, they will have no problem migrating to Node Messenger. All the features in our messenger are self-intuitive and do not take much time to get used to. Node Messenger is currently hosted on Github pages making it accessible to anyone the planet. It is able to run on any platform without problems. However, during testing, we realized that our messenger does not respond well to smaller screen sizes. Our goal for our servers was to be able to hold as many people as possible. Currently, due to the limitations of our database, Node Messenger is able to serve about 50 people at once.

## 2.3 Performance

Node Messenger performs very well under normal use. The tests were performed on a Windows machine with Intel i5, 1.60GHz, and 8 GB RAM. Our performance goals for Node Messenger was for it to load the messenger, send

messages, receive messages, and load past messages quickly. We timed each of these procedures, and the results prove to be very fast as we intended.

| Description | Time |
|---|---|
| Messenger Load Time | 1.2 s |
| Message Send Time | 0.36 s |
| Message Receive Time | 0.56 s |
| Past Messages Load Time | 0.33 s |

## 2.4 Operational and Environmental

Node Messenger is able to work on all platforms. However, it performs the best on Windows. Due to the scale of this project and the lack of time, responsiveness was not fully implemented. Meaning Node Messenger has difficulty loading components properly on smaller screens. This does not mean the messenger is unusable on these platforms. All functions of the messenger perform as intended. None of our testers have reported any other errors occurring when attempting to load on platforms other than windows.

## 2.5 Maintainability and Support

All functions in Node Messenger has been tested thoroughly to ensure everything performs as intended. We want to make sure to keep the downtime and maintenance to a minimum. All of the source code is well structured and documented to help us quickly find errors, and fix them. This will also help programmers navigate through our code if they want to add any new features to Node Messenger.

## 2.6 Security

All user information is kept safely in our firebase database. Node Messenger will not save any of the personal information nor will it share them with other parties.

## 2.7 Legal

Node Messenger abides by all rules and regulations. The public will have access to the open-source project. If they would like to improve our messenger, they are free to do so.

# 3 Comparison to Existing Implementation

# 4 Unit Testing

As we stated in our test plan, the majority of the testing has been done with Jest. We tested the rendering of the messenger, as well as the calls being sent to our firebase database using Jest's snapshot feature. We tested most of the functions used by the messenger to ensure that maintenance can be kept to a minimum. Refer to the Trace to Modules table to visualize how test cases use each module.

# 5 Changes Due to Testing

Due to testing and constant feedback from our testers, we were able to make several improvements to Node Messenger. Testing helped us to uncover hidden bugs that occur through normal use of the messenger. Through testing, we discovered a bug when nothing is entered in the message box and the user presses send. The messenger would send an empty message bubble to the chat anyways. Another bug we discovered while testing was when a very long sentence is entered in the chat. The long message would not automatically go to the next line. We mixed it easily by wrapping the text with the message bubble. One of our classmates who tested the messenger notified us that he was able to login to someone else's account using a random password. We fixed it by looking through our firebase database and changing how data is searched from our messenger.

# 6 Automated Testing

# 7 Trace to Requirements

# 8 Trace to Modules

# 9 Code Coverage Metrics