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

Table 1: **Revision History**

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

This document ...

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

Node Messenger was based on the open-source web messaging app, Tinode. Tinode aims to be a replacement for XMPP. It compares itself to other open-source messaging apps like WhatsApp and Telegram. There are a lot of similarities between Tinode and Node Messenger. They were both created using React. Both allow users to sign up and log in to begin using the messenger. Both allow the user to create conversations with multiple other users using emails. However, Node Messenger uses a different database than Tinode. Node Messenger and Tinode both have a unique theme that distinguishes them from other messaging apps on the market. Despite the differences, Node Messenger and Tinode both share a common goal – To create a fast, efficient, and user-friendly messaging environment.

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

6.1 Validation of User Input

6.1.1 Control

Checks if user information entered is valid

6.1.2 Inputs

Empty String, invalid emails and valid emails

6.1.3 Outputs

Appropriate Error shown if input invalid. Chat screen if valid input.

6.1.4 Procedures

The test will be performed using the Jest framework. Based on the input, the render tree will be tested for appropriate output.

6.2 Successful sign up

6.2.1 Control

Checks if sign up is successful

6.2.2 Inputs

Email address and password

6.2.3 Outputs

Pop up indicating user successfully signed up. User redirected to chat screen

6.2.4 Procedures

The test will be automated with the testing framework jest. The user data will be used for sign up and will be checked against the list of users in the database.

6.3 Successful Login

6.3.1 Control

Checks if login is successful

6.3.2 Inputs

email address and password used for login

6.3.3 Outputs

Redirect user to the chat screen

6.3.4 Procedures

The test will be automated with the testing framework jest. The user data will be used for login and the authentication state will be checked to see if the login was successful.

6.4 Authentication State Persistence

6.4.1 Control

Check if the computer is trusted

6.4.2 Inputs

User reloads the site

6.4.3 Outputs

If the computer is listed as a user trusted computer, keep the user logged in.

6.4.4 Procedures

The test will be automated with the testing framework jest. The change in authentication state will be used to determine if the user is automatically signed if the page is reloaded.

6.5 User message renders on input box

6.5.1 Control

Check if the messages render properly on the input box

6.5.2 Inputs

User starts entering message in input box

6.5.3 Outputs

Message shows up in the input box

6.5.4 Procedures

The test will be done using jest framework. The react tree will be checked for the rendered message in the input box.

6.6 User message renders on screen

6.6.1 Control

Check if user message renders on the right side of the screen

6.6.2 Inputs

User enters message in the input box

6.6.3 Outputs

Message shows up on the right side in the conversation windows

6.6.4 Procedures

The test will be done using jest framework. The react tree will be checked for the rendered message.

6.7 Received message renders on screen

6.7.1 Control

Check if the received messages render on the left side of the screen

6.7.2 Inputs

N/A

6.7.3 Outputs

Message received shows up on the right side in the conversation windows

6.7.4 Procedures

The test will be done using jest framework. The react tree will be checked for the rendered message.

6.8 Instant Messaging

6.8.1 Control

Check if message send and receive features are working as intended

6.8.2 Inputs

N/A

6.8.3 Outputs

Send/Receive messages

6.8.4 Procedures

This manual test will be performed using two instances of the application. The test will check if the sending and receiving of messages happen within a few seconds of pressing the send button.

6.9 Contacts

6.9.1 Control

Check if contacts are properly being rendered on the contacts list

6.9.2 Inputs

Add a contact

6.9.3 Outputs

Contact is rendered on user's contacts list

6.9.4 Procedures

The test will be performed using Jest. The rendered contact list will be tested for the updated state.

6.10 Contacts

6.10.1 Control

Check if the database stores the contact information after they are added

6.10.2 Inputs

Add a contact

6.10.3 Outputs

Upon adding a contact the database should add the contact to user's contact list

6.10.4 Procedures

This manual test will be performed using jest. The user's contact list will be tested before and after the addition of few contacts. Edge cases will cover the testing of adding contacts that are not in the system.

6.11 Chat history

6.11.1 Control

Check if chat history is rendered properly

6.11.2 Inputs

Log-out and Log-in back to the chat

6.11.3 Outputs

The chat history should be rendered and the message history should be accessible.

6.11.4 Procedures

This manual test will be performed using jest. The user's contact list will be tested before and after the addition of few contacts. Edge cases will cover the testing of adding contacts that are not in the system.

6.12 To test the look and feel of the application

6.12.1 Control

Ensure the app is user-friendly and eye catching

6.12.2 Inputs

N/A

6.12.3 Outputs

N/A

6.12.4 Procedures

The UI/UX tests will subjectively give a qualitative result of how the system feels in terms for ease of use ,appearance and responsiveness. This test will carried out by using the application for a certain amount of time.

6.13 Time taken to send and receive messages (acceptable: $<$ 2 sec)

6.13.1 Control

Ensure that messages transfer quickly between users

6.13.2 Inputs

Message sent

6.13.3 Outputs

Server callback with approximated time.

6.13.4 Procedures

This test will be automated by the jest framework. The server will send a ping once a message is received, which will then used to estimate the average time taken to send and receive messages.

6.14 Scrolling through a large amount of messages

6.14.1 Control

Check that the messenger scrolls through a large number of messages properly

6.14.2 Inputs

Scroll action

6.14.3 Outputs

Scrolling message area

6.14.4 Procedures

This test will be performed manually to see if the scrolling behaviour and responsiveness is affected by large number of messages being loaded in on the client side

6.15 Server responsiveness when large group is active

6.15.1 Control

Check that the server is stable when there are a lot of people using the messenger at once

6.15.2 Inputs

Messages being sent by everyone

6.15.3 Outputs

Receiving messages

6.15.4 Procedures

This test will be performed using jest. The time from the message being sent to the server and receiving a callback will be tested to see the responsiveness of the server when under heavy load

6.16 Tests users access level

6.16.1 Control

Check if the user has access to the contact list

6.16.2 Inputs

User tries to access the page that routes to a different user's contact list

6.16.3 Outputs

Unauthorized access error

6.16.4 Procedures

The test will be automated using jest. jest will try to access unauthorized pages under a user and check if the appropriate error is thrown back.

6.17 Mobile/Tablet UI/UX

6.17.1 Control

Check if messages send as intended on mobiles or tablets

6.17.2 Inputs

Log-in and send a message

6.17.3 Outputs

Rendered message on screen

6.17.4 Procedures

The Mobile/Tablet UI/UX tests will subjectively give a qualitative result of how the system performs on different devices. Google Chrome Dev Tools along with our personal devices will be used for this test.

7 Trace to Requirements

8 Trace to Modules

9 Code Coverage Metrics

We tested all the functions in our messaging app and that they function as intended. We tested for all the requirements and made sure they pass all the tests. We had exactly 92% test coverage after all the testing was done. The functions we covered during testing carry out all the main functions of Node Messenger. Testing these functions will prove to be beneficial in keeping Node Messenger bug free.

Test	Requirements
Functional Requirements Testing	
auth-test-1	M2, M3, M4
auth-test-2	M2, M3, M4
auth-test-3	M3, M2
auth-test-4	M4, M2
chat-test-1	M5
chat-test-2	M5
chat-test-3	M5
chat-test-4	M5
chat-test-5	M5
db-test-1	M5, M6
db-test-2	M5
Non-functional Requirements Testing	
ui-test-1	M2, M3, M4, M5, M6
pf-test-1	M5
pf-stress-test-2	M5
pf-stress-test-3	M2, M3, M4, M5
sr-test-1	M2, M3, M4, M5, M6
pr-test-1	M2, M3, M4, M5
Automated Testing	
ut-test-1	M6, M3, M2
ut-test-2	M6, M4, M2
ut-test-3	M5, M2, M3
ut-test-4	M5
ut-test-5	M5
ut-test-6	M5

Table 2: Trace Between Tests and Requirements