SE 3XA3: Test Plan
Node Messenger

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

November 28, 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

1 General Information

1.1 Purpose

This purpose of this document is to outline areas of testing from tools to methodology and plan that will verify both functional and non-functional requirements of the product and ensure that we are delivering a valid product to our clients designed in accordance to the outlined requirements.

1.2 Scope

As this is web-messenger product, testing will be focused on elements that are integral for mobile communication. Test plan shall cover both front-end and back-end implementations of functions to ensure validity. Front-end testing will consist of user input and output correction as well and proper rendering. Back-end testing will ensure that proper input and output collection is correct as well as user authentication, data transfer and stress testing firebase database system. If the product passes all the tests outlined in this document, it will be considered a valid implementation of the requirements listed in the traceability matrix outlined in the SRS document.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	Definition
auth	authentication
UI/UX	User Interface/ User-Experience
async calls	Asynchronous requests that client makes to retrieve information

1.4 Overview of Document

This document will outline a proper test plan for functional and non-functional requirement as well as tests for our current proof of concept demonstration.

Table 3: **Table of Definitions**

Term	Definition
Firebase	Firebase is a backend service provider
Jest	React Testing framework with Async Testing Support
React	JavaScript UI Library

The document will also break down the testing process into multiple divided section and elaborating on them in great detail to provide context to our testing methodologies. Finally this implementation of the product will be compared to a successful one based on a list of usability questions and through this an outlined unit test plan will be formed. Any abstract symbols can be explored in the attached appendix for clarification on any unclear document elements.

2 Plan

2.1 Software Description

The software is a fully functioning web-app that allows users to communicate with each other through messages. The users are able to create an account using their preferred email address and specified display name and password. Once they are clients of Node Messenger, they are able to utilize its messaging features. The web-page displays a list of previous conversations and selected ones. Users can initiate conversations with other Node Messenger clients and converse through messages in a local chat window. All conversation information is received and rendered for the user in constant time to create a seamless and naturally flowing dialogue.

2.2 Test Team

The test team consists of all three members of the development team: Tasin Ahmed, Shardool Patel, and Omar Elemary. The testing process will be divided into both front-end and back-end testing. Each member is responsible for conducting unit tests for each developed components to ensure proper functionality. The team will then begin integration testing once all modules

are complete to examine the system working as a whole and additionally perform a system test to verify that all outlined functional and non-functional requirements of the product are successfully met.

2.3 Automated Testing Approach

The team plans to automate most of the testing using Jest framework. ~~Jest and Mocha.~~ UI components will be tested based on the snapshot testing approach that jest implements, allowing us to see if things are rendered accurately. ~~Everything other than the UI will be unit tested on mocha~~. Black Box testing will be performed using Jest for all modules. Back-end testing and system testing will be done through jest using its async testing features.
Since the system is relying on firebase servers, stress tests as indicated in performance section will be undertaken to ensure the servers can perform under heavy load.

2.4 Testing Tools

~~We will be utilizing Jest in order to test React our code, and Mocha to test JavaScript.~~ The system testing will be done using Jest. This will include client side rendering and async calls to the database. The client side rendering is tested using Jest's snapshot feature which parses the render tree to tests if the render matches the previous state one. This will be done in order to keep bugs to a minimum, and to better the user experience. There will be a sections on our website where users can leave their reviews, and suggestions to improve our product.

2.5 Testing Schedule

See Gantt Chart at the following url:
https://gitlab.cas.mcmaster.ca/pates25/NodeMessenger/blob/master/ProjectSchedule/ProjectSchedule.gan

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Authentication

Login / Sign up Forms

1. auth-test-1 : Validation of User Input

   Type:
   Functional, Dynamic, Automated

   Initial State:
   Empty Forms

   Input:
   Empty String, invalid emails and valid emails

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

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

2. auth-test-2 : Successful sign up

   Type:
   Functional, Dynamic, Automated

   Initial State:
   Email not signed up

   Input:
   email address and password

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

   How test will be performed:
   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.

3. auth-test-3 : Successful Login

   Type:
   Functional, Dynamic, Automated

   Initial State:
   User not logged in

   Input: email address and password used for login

   Output:
   Redirect user to the chat screen

   How test will be performed:
   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.

4. auth-test-4 : Authentication State Persistence

   Type:
   Functional, Dynamic, Automated

   Initial State:
   User logged in

   Input:
   User reloads the site

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

   How test will be performed:
   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.

3.1.2 Chat Input/Output

1. chat-test-1 : User message renders on input box

Type:
Functional, Dynamic, Automated

Initial State:
Message not rendered in the input box

Input:
User starts entering message in input box

Output:
Message shows up in the input box

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

2. chat-test-2 : User message renders on screen

Type:
Functional, Dynamic, Automated

Initial State:
Message not rendered on the chat box

Input:
User enters message in the input box

Output:
Message shows up on the right side in the conversation windows

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

3. chat-test-3 : Received message renders on screen

Type:
Functional, Dynamic, Automated

Initial State:
No received messages

Input:
N/A

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

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

4. chat-test-4 : Instant Messaging

   Type:
   Functional, Dynamic, Manual

   Initial State:
   No sent or received messages

   Input:
   N/A

   Output:
   Send/Receive messages

   How test will be performed:
   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.

5. chat-test-4 : Contacts

   Type:
   Functional, Dynamic, Manual

   Initial State:
   No contacts

   Input:
   Add a contact

   Output:
   Contact is rendered on user's contacts list

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

3.1.3 Database

1. db-test-1 : Contacts

   Type:
   Functional, Dynamic, Automatic

   Initial State:
   No contacts

   Input:
   Add a contact

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

   How test will be performed:
   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.

2. db-test-2 : Chat history

   Type:
   Functional, Dynamic, Automatic

   Initial State:
   Chat initiated with a contact

   Input:
   Log-out and Log-in back to the chat

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

   How test will be performed:
   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.

3.2 Tests for Nonfunctional Requirements

3.2.1 UI/UX Tests

1. ui-test-1 : To test the look and feel of the application

   Type:
   Static, Manual

   Initial State: Chat Box with messages rendered in.

   Input: N/A

   Output: N/A

   How test will be performed:
   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.

3.2.2 Performance

1. pf-test-1 : Time taken to send and receive messages (acceptable: < 2 sec)

   Type:
   Dynamic, Automated

   Initial State: Message typed in chat window

   Input:
   Message sent

   Output:
   Server callback with approximated time.

   How test will be performed:
   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.

2. pf-stress-test-2 : Scrolling through a large amount of messages

   Type:
   Static, Manual

   Initial State: 300-500 Messages loaded on client side

   Input:
   Scroll action

   Output:
   Scrolling message area

   How test will be performed:
   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.

1. pf-server-stress-test-3 : Server responsiveness when large group is active

   Type:
   Automated, Dynamic

   Initial State: ¿20 Member group is active.

   Input:
   Messages being sent by everyone

   Output:
   Receiving messages

   How test will be performed:
   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.

3.2.3 Security Testing

1. sr-test-1 : Tests users access level

   Type:
   Dynamic, Automated

Initial State:
User logged in

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

Output:
Unauthorized access error

How test will be performed:
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.

3.2.4 Portability

1. pr-test-1 : Mobile/Tablet UI/UX

Type:
Static, Manual

Initial State: Log-in page

Input: Log-in and send a message

Output: Rendered message on screen

How test will be performed:
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.

4 Tests for Proof of Concept

The purpose of the proof of concept was to demonstrate the general functionality of a chat application. It included log-in/sign-up and real-time chat built within a simple user interface.
The proof of concept will include system testing so that the basic functionality of the application is tested. This will include automated test from following sections:
3.1.1 Authentication, 3.1.2 Chat Input/Output along with trivial non-functional

tests from 3.2.1 Performance and 3.2.4 Portability.
Aside from the system tests, the proof of concept will be unit tested to see if the methods function correctly. This will be done through the jest framework.

5 Comparison to Existing Implementation

Node Messenger is based upon Tinode, an existing implantation of a web messaging app. Tinode implements a successful messaging app with the inclusion of 3 main functions: Correct display of messages being sent and received, displays a list of past messages with other users, and stores the messages of a user, allowing them to logout or login with a different account. We will be testing to see if the messages being sent, and the messages being received are formatted properly, and they display on the correct side of the screen. Node Messenger will also be tested to ensure it can create different messaging rooms to let the user chat with other users flawlessly. We will be creating multiple accounts to test if Node Messenger is able to store, access, and show data specific to each user.

6 Unit Testing Plan

6.1 Unit testing of internal functions

We will be using Jest to test the functions governing Node Messenger. We will be ensure that the functions carry out their task properly and return their expected output. Messages should display properly if it is entered correctly, e.g. if a blank message is entered, it must not send an empty message to the message box. Our group will try to cover as much ground as possible in order to ensure a better user experience.
The test will apply the black box testing to see if the output is in compliance with whats listed in the MIS and will also include white box testing to cover edges cases that might be missed due to rare occurrences. ~~Unit testing using Jest will be mainly used for testing internal functions in Node Messenger. We will be testing the functions above, as well as other key internal functions to ensure Node Messenger functions as expected. Unit testing these internal functions will help us to determine hidden errors, or loopholes in the code that cannot be easily spotted.~~

6.2 Unit testing of output files

Not Applicable.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.