ShortStack

A COMP 324 Final Project by Nicholas Synovic

Table of Contents

– Conclusion

Note

This document only includes information relevant to the final report submission. For a condensed version of the document, please read this project's README.md.

Project Source Code and Published Link

GitHub Repository Link

- https://github.com/NicholasSynovic/shortstack

GitHub Pages Link

- https://nicholassynovic.github.io/shortstack

List of Project Participants

Nicholas Synovic

Lead (and only) developer and designer of the application. Literally did every part of the assignment on his own.

Project Abstract

ShortStack is a simple chat application that allows users to have one-on-one chats with other users. Chats can contain only contain textual information, but users can send both Markdown and HTML code to be rendered in the chat window. This allows for hyperlinks, documents, images, video, audio, etc. to be shared without having to upload multimedia to a database.

The main draw of the application is the built in Markdown renderer, allowing for Markdown and HTML content to be shared and rendered within the chat window. Furthermore, ShortStack supports the cloud synchronization of messages (via a Firebase Firestore collection), allowing users to view their messages on any browser. Since all messages are textual, Markdown and HTML content that is sent on one browser will be rendered on another in a similar and timely manner.

The application is designed and functions around the less-is-more ethos. The app only has the components and wiring necessary for it to run, both on the client and Firebase side. It presents all of the availble client options to the user unobstructed by side or hidden menus. The singular Firebase Firestore collection schema is as simple as possible too.

I chose to build a chat application to learn the following:

- Firebase
- Markdown rendering in browser

- JavaScript templating
- Progressive Web Applications (PWAs)
- Pretty Good Privacy (PGP) encryption

While not all of what I learned is included within this application, due to iterative design all of these components were at once a part of ShortStack. Through this application, I learned more about the underlying web components and services that power today's top websites and applications.

Project Narrative

What Was I Trying To Achieve?

Following the midterm presentations, I wanted to build a chat application that allowed for users to share all sorts of content and media via text formatting with Markdown and HTML. I did not, however, want to store mulitmedia content in Firebase. Rather, I wanted to treat text as a first class and only citizen of my Firebase Firestore collection.

I wanted to build ShortStack as a fast, simple, and unbloated chat application for my users. I wanted as much of the application to be dynamically rendered based on the current view (reducing the size of the distributed HTML files), have all availible options for a view to be presented to the user without hiding them behind hamburger, obtrusive, or hidden menus (think the opposite of Facebook), and to only present a view's completed features so as to avoid users from using broken or half-complete components.

I want ShortStack to look as good as possible across as many devices as possible. I don't want complex CSS styling or animations as they can lead to bad user interfaces when trying to distribute across a number of devices unless properly tested for. Furthermore, I suffer from red-green colorblindness, so the less that I have to color elements, the better.

Why Did I Want To Achieve These Goals?

I believe that modern chat applications, such as Facebook, have gotten so large that they are unusable. Whether it be implementing half-complete features, hiding user options, or taking substantial amounts of time to load on the client side, these applications are no longer user friendly unless you have been with the application since day one. I wanted to take the core component of these applications (chatting with others) and focus ShortStack on accomplishing that singlular goal with an interface that is as simple as possible.

To do so, I reduced the scope of what a chat feature can or should do to just the bare minimum: send text content between users. This was both to accomplish my goals and to reduce the Firebase backend development that I would have to do to host multimedia content. However, I still wanted ShortStack to support multimedia content. To do so, I implemented a Markdown renderer

API called marked. `marked` implements the GitHub Flavored Markdown, which also supports HTML rendering. Using this, users can send multimedia content between one another so long as it is packaged in HTML markup. They can also do text formatting via GitHub Flavored Markdown.

Design Considerations

Original Project ideas and Concepts

The original goal of this application was to build a chat application with some additional nuts and bolts. Particularly, I wanted to make the application a PWA that implemented PGP encryption on all messages.

The reason to make the application a PWA was to experiment with media queries as well as to learn how to build mobile applications that are distributed via websites rather than centralized sources such as the Apple App Store and the Google Play Store. As a PWA, my application can be installed onto a users device via their browser. Furthermore, a PWA would let this application would be able to cache messages offline and utilize the device notification API.

The reason why I wanted to implement PGP encryption was to both learn how to do it, as well as to offer a chat option to those focussed on protecting their digital privacy and security. My implementation would have stored the users public key in aFirebase Firestore document associated to their account, but encrypt messages on the client side before sending the message into the Firebase Firestore *messages* collection as a document. The private key would have to be inputted into the application everytime the user returned to it, but Firebase Firestore wouldn't know the user's private key. This way, messages are encrypted before Firebase Firestore even recieves them, preventing me (the applicaiton owner) or any bad actor from peering into someone's messages without knowing that individual's private key.

As this was a chat application, no premade datasets were to be used in the application. Rather, the dataset that would be used would be the user generated chats.

Example projects that I looked at were Facebook, Twitter, Tumblr, Android Messages, and Proton Mail (a PGP secured email client) for design and technical considerations.

Primary Goal of the Homepage

The primary goal of the homepage of ShortStack is to host the sign in and up buttons. The only content on the page is meant to direct users to the GitHub repository (via the ribbon that appears on all pages on the desktop version of the site), present the title, logo, and other metadata about the site.

The true homepage of ShortStack is the chat interface. This is only accessible in a working state once the user logs in. This interface presents the chat window,

an option to sign out, a clickable header that will take you back to the page described above, a button to send messages, and a textbox and textarea element. The textbox is to input the recipients email address, and the textarea is to input the message to send to the recipient.

Consistency Considerations

I choose the following colors for my color scheme:

- #55a1ff (a light blue color)
- #ffc861 (a light green color)
- #ffab61 (a light red color)

These colors were selected because they were the colors of ShortStack's icon provided by bukeicon from flaticon.

I tried to avoid including any graphics in the views, with the exception being the chat view. There a robot icon can be found at the top of the screen which is unique to each user. This is meant to be a distinguishing icon that can be used to track who sent what message. This icon is generated using the Dicebear Avatars API.

Each page is as simple as it can be in terms of functionality. There is nothing extra or unnescary on each page as part of the design ethos of this application was to keep it as simple as possible.

Each page follows the same width, height, content placement, and containerization of elements style guide. This is enforced via component specific CSS that is loaded into every page, as well as by keeping common elements as components that are loaded in after the HTML file is rendered. By utilizing components I can ensure that a component will look the same across multiple pages. Components are handled as HTML templates that are appended as children to the document.

Response to Midterm Feedback

After the midterm presentation, I rescoped the project into something that I feel is more manageable. I removed the PWA functionality, and put the PGP encryption of messages on the backburner. I then strongly focussed on creating as simple of a chat app as possible that included the following features:

- Authentication with Firebase Authentication
- One-to-one messaging
- Unique user avatars with the Dicebear Avatars API
- Markdown and HTML rendering of messages with the `marked` API
- Cloud storage via Firebase Firestore

With the project rescoped to include only whats listed above, I was able to create a project that is both easily extensible and can be used as a template to others as to what a chat application can be.

These changes reduced the technical debt that the site would take on as well as to allow me to iterate quickly through ideas and prototypes with little concern that I would have to redevelop the site. As someone who worked on this project alone, quick iteration was key to the success of ShortStack.

Interaction Patterns

There are two types of interation that occur within ShortStack:

1. Automated interaction

This is interaction that is done by the site at runtime. This includes:

- Rending components
- Firebase Authentication
- Firebase Firestore document storage and request calls
- Markdown and HTML of messages rendering

All of these interactions in this catagory benefit from the Deferred pattern. Specifically:

- Markdown and HTML message rendering doesn't occur until there is a message to render
- Firebase calls are all asynchronus
- All components are rendered on the page after the HTML file has been rendered in the browser
- I utilized webpack to bundle npm packages for this application and it loads the bundled bundle.js file after the page has been rendered

2. User interaction

This is interaction that the user does wihtin ShortStack. This includes:

- Clicking buttons
- Inputting text

These interactions are all handled by standard HTML elements including:

- `form`
- `button`
- `input`
- `textarea`

Regarding the input elements on the sign in and sign up pages: the user has to input an email and password to log into the application. To enforce that an email address is typed into the email `input` element and a password is typed into the password `input` element, I set the `type` of those `input` elements to *email* and *password* respectively. Also, both in my application and in the application's Firebase Authentication, a password of 6 charachters or more has to be inputted for it to be a valid password.

Should a user input a wrong value or encounter an error with the application (either on the client or Firebase side of the application), an error report is shown to the user via a JavaScript alert element.

Design, Design Process, and Specification

Design

I went with a less is more approach. I am a huge fan of the lynx browser and how it presents information with the barest amount of styling. Now, I can't expect people to access ShortStack through a niche browser, so I did add some basic styling that allows the gives the site some color while also nicely spacing out elements.

The design of chat messages is handled via marked. I didn't configure the API to style the text in any particular manner as I prefer the default styling that it applies to content.

Design Process

The general design philosophy looked like this algorithm:

1. Research a feature that I want to add
2. Implement feature
3. Contemplate morally if that feature adds value to the application
4. If true, reduce the code complexity of the feature to its simplest form (or what I am capable of doing)
5. Else, scrap feature
6. Finally, jump to 1

This algorithm is a genralization of the design process as every feature required a slight tweak to complete it.

Specification

I decided to format chat messages with Markdown for the sole reason that I love Markdown. I think it provides one of the most intuitive and easy to approach methods of styling text. I also believe that the applications that support Markdown rendering of messages such as Reddit and GitHub Issues provide users with a way to specify blocks of content easily without having the application build it in as an insertable widget. Furthermore, Markdown allows for HTML to be rendered as well. Therfore, users can share custom HTML snippets with one another via the Markdown renderer that is included in ShortStack

Testing and Iterative Design

Testing

There was very little testing done with unit tests for ShortStack. The only tests that I had were webpack error reports.

User stories were written out and those were tested by running through test cases that I designed.

One of the problems with working alone is that I didn't recieve any feedback about the design of ShortStack until my final presentation. Since then, I have implemented a better color scheme than what I presented. That is my only regret with deciding to work alone for this project.

Iterative Design

This section will be described in terms of version numbers with version 1.0 being the version presented in class on 12/7/2021:

v0.1

- Inital version
- Created GitHub repository and started messing around with creating PWAs

v0.2

- Decided to focus on creating a chat application
- Finalized basic installable PWA that ran on a client's device
- Started work on learning how to access device notifications with a PWA
- Started looking at sites like Facebook, Twitter, Tumblr, Android Messages, and Proton Mail
-

v0.3

- Midterm presentation
- Started learning how to implement OpenPGP.js into the application

v0.4

- Scrapped work due to anxiety that was driven by the sheer scope of the HTML files
- Began researching JavaScript HTML templating and components
- Refocused efforts on just creating a chat application first and bolting on PGP and PWA support later

v0.5

- Reworked HTML and JavaScript to utilize components
- Started learning webpack
- Started learning Firebase

v0.6

- Learned enough webpack to deploy site
- Wrote GitHub Action to publish site on `push` to GitHub Pages
- Implemented basic Firebase Authentication with email and password

v0.7

- Implemented basic Firebase document publisher pattern
- Implemented basic Firebase collection observer pattern

v0.8

- Started work on the overall design on the site
- Revisted Android Messages and Facebook for inspiration and to see what not to do respectively
- Started lookign at logos with a good color scheme that I could match the site to

v0.9

- Completed the overall design of the site
- Started work on styling the chat component
- Simplified JS to be as simple as possible

v1.0

- Final presentation of application
- Completed styling the chat component
- Implented color scheme based off of logo (post presentation)

Restrictions, Limitations, and Constraints

Restrictions

I didn't encounter any restrictions with the technologies that I planned to use or did use in developing ShortStack.

Limitations

I was limited by time and other responsibilities in developing ShortStack. Again this is a challenge that arose purely from working alone and if I worked in a team I'm sure that the time constrained issues would have been resolved.

Most of the time constrained issues had to do with the style of the application. However, the one non-style limitation that I ran into was developing a proper Firebase query that only retrieved a specific users messages. Sadly, I just didn't have time to either work out the database schema or how to implement the query. To resolve this, the application fetches all documents from the collection and uses client side logic to resolve which documents should be presented to the end user.

Constraints

I didn't run into any technical constraints while working on ShortStack.

Features That I Failed To Complete

One of the features I wanted to add was a contact picker or a friends list. The reason why this was canned and didn't make it into the application was because there wasn't enough time to properly test it before the final presentation.

Another feature was the PWA functionality. I canned this feature not because it was difficult to implement, but it added too much technical debt that I had to look out for while developing this application.

A third proposed feature that I wanted to include but got canned was the PGP encryption of messages. Similar to the PWA feature, this added too much technical debt that I was unprepared to take on, so I canned it.

Conclusion

I think my application, ShortStack, achieved all of the post-midterm goals it set out to meet. I think that this is probably the best website I have built, both technically and stylistically, using nothing but vanilla HTML, CSS, and JavaScript.

If I were to go back and redo this application, I would use a CSS theme like BootStrap to style the page as it takes all of the pressure off of my to create my own custom styles.

I would also spend more time looking into the PWA feature. I do believe that in my applications current state, it is still possible for it to be a PWA. However, in order to meet the deadlines of the course, I decided to can the feature in favor of the core functionality which is to send one-to-one text messages to other users.

The Markdown and HTML chat rendering that I built into the application is pretty cool as well. Without it, I would've had to implement all of that from scratch which would've been too much technical debt and probably would've been scrapped in favor of something else.

Furthermore, the JavaScript templating and components that I came up with saved my butt and reduced my size anxiety whenever I look at HTML files.

Having these repeated elements be loaded in on the fly I think is a pretty cool feature that sets my project apart from other work that I've done.

I am happy with how my application turned out. It isn't perfect by any means, but it gets the job done. I look forward to expanding on it after this course ends.