Modern Funding

Colin Schuh, Francesco Ventura

University of Kentucky

Computer Science 498; Software Engineering for Senior Project

Kevin Eby

Modern Funding

Modern Funding is a potential startup, based off of the idea that it can be difficult for many small businesses and certain individuals to get funding to complete projects and expand. Modern Funding is a platform with the potential of helping to solve this problem, while also allowing the general public to invest similarly to venture capitalists. Typically, banks are where people go when they need funding, however our aim is to begin to change that. Aside from the current financial system being limiting to certain parties, it's also not as efficient as it could be.

For this project we created a web application that can connect private investors and borrowers and unite them towards a common goal. Our application includes a public forum where borrowers can make requests for a loan and a private email-type messaging system where investors and borrowers can connect privately, among a few other tools.

Project

Our team chose to complete our own project task which was creating a full stack web application using ReactJS, NextJs, and Supabase for the back end. We chose to do this project because we had a mutual interest in Fintech. During a Financial Technology class, Franco was inspired by one of the earlier lectures in the semester, and we both agreed that he may have a pretty good idea for a business. From there we started to brainstorm and expand on how we wanted to approach this.

In retrospect, after encountering many challenges trying to implement our application, we believe that we still would have chosen to complete this project task, but it would be in our best interests to reduce the scope of the project. Perhaps reducing the scope in the form of choosing to not implement an authentication service as that was one of the hardest user stories to implement.

User Stories

The user stories that we have selected to be in our final implementation are as follows:

1. As a user, if I do not have an account, then I should be able to create an account by clicking on the sign-up button on the login page, or the top navbar, so that I will be able to use the app's functionalities.

2. As an account holder, I should be able to login by clicking on the login option on either the sidebar or in the top navbar. Once directed to the login page, I should be able to sign in by using their username/email and password. This way I will be directed to their account page securely so nobody can act on their behalf.

3. As an account holder, if they type in an email/username that doesn't exist in the system, then they should be prompted so that they know that they either don't have an account or typed in the email/username wrong.

4. As an account holder, if the username/email does exist as part of an account but the password entered is wrong, the user should be prompted that the password is incorrect. This way they will have another opportunity to type it in again, and their account is secure against anyone without the password.

5. As an account holder, I would like to be able to change my username to whatever I'd like and be able to update my account information.

6. As a standard user, I would like to be able to click the website logo to return to the homepage no matter where in the application I am.

7. As an account holder, I would like to be able to message/email other account holders in real time.

8. As an account holder, I would like to see investment opportunities, and information about them to decide if they would be a good fit for me. I'd also like to be able to request a secure conversation with the other party.

9. As a user, when I go to the website, I would like to see a home page.

10. As a user, I want the home page to have a background, navbar at the top, sidebar with links to different pages of the website, a mission statement, recent website activity, and company / company relevant news/promotions. At the bottom I would like for there to be a common footer with contact us, about us.

11. As an account holder, I want to be able to post a request for funds and be able to give details to entice potential investors.

12. As a potential investor, I want to be able to see a list of all requests for funds and be able to review information provided by the fund seeker to help determine if I want to proceed.

13. As an account holder, I want to be able to negotiate contracts.

Upon completing the *Project Milestone 2* requirement we had only implemented the first four user stories listed above. Since then, we have refined the authentication user interface and implemented user stories five through thirteen. Stories six through thirteen generally involve the inbox and dashboard pages, where users can instantly message other users in real time or create new investment listings, respectively. Lastly, user story five was completed upon the implementation of the settings page, where a user can change nearly all of the user information they input when they first signed up, including username, phone number, etc.

Source Code

The source code for our project can be found in our git repository here. (https://github.com/colins-uky/CS498-Project). Our production branch is called *master* and it is where you will find the most up to date version of our application. Most of our changes are located within the /src/ directory of the app. *Vercel*, the creator of NextJS, allows anyone to create an account and deploy their NextJS project to their website. Our finished project can be found and tested here. (https://nextjs-vercel-omega-one.vercel.app/).

Design Elements

The components that we decided to implement relate directly to the user stories that we selected. They can be broadly classified into the following categories:

1. **User Interface (UI) Components:** These components ensure a visually appealing and intuitive user experience. They include a responsive navbar at the top of each page, a sidebar with links to different pages of the website, and a common footer with contact information.
2. **Authentication Components:** To ensure a secure environment, we implemented a robust authentication system, allowing users to create accounts, sign in, and change their credentials if needed. This involved creating a sign-up form, a login form, and a *Supabase* authentication session which holds all of the user authentication information once they have signed in/up.

3. **Dashboard Components:** The dashboard is the central hub for users to view investment opportunities, and post funding requests. We designed an intuitive layout that presents information in a clear and organized manner.
4. **Messaging Components:** To facilitate communication between account holders, we integrated a real-time messaging system that allows users to send and receive messages instantly. This feature includes an inbox where users can view their conversations and a messaging interface for seamless interaction.

All of these component groups interact with at least one component from another group while in production. For example, both the dashboard and messaging components contain the UI component *Topbar*. They also make use of the authentication component by fetching the authentication data of the user *before* displaying any sensitive or user specific data to the inbox or dashboard pages.

The result of having our components nested into an inheritance “tree” is that the component will only “see” and have access to the information that it needs. This will hopefully result in less bad data redundancy, and ultimately less memory usage on our end user’s machines.

Sources of Additional Info

The sources of additional and helpful information that we sought out were primarily the official documentation pages from ReactJS, NextJS, and Supabase. To construct a solid foundation for our project to be built on, we began by extensively studying the official documentation for ReactJS, NextJS, and Supabase. The ReactJS documentation provided us with

comprehensive information on creating components, managing state, and handling events, which helped us develop a highly interactive and responsive user interface. The NextJS documentation played a crucial role in understanding server-side rendering, static site generation, and API routes, enabling us to create a high-performance, full stack web application. Finally, shortly after we began the development process, the Supabase documentation became invaluable in setting up and managing our backend infrastructure. We learned how to integrate Supabase's real-time messaging capabilities, authentication services, and database management, which allowed us to build a secure and efficient application.

Software Engineering Techniques

Our team decided to use the pair programming technique at first which helped us share the knowledge that we learned about each framework individually during our group meetings. Later on in development we switched over to an agile sprint approach where we wrote specific tickets that we would complete to bring us closer to implementing our user stories. In retrospect, we believe that we should have used 2-week agile sprints for a longer duration of the development lifecycle as we believe we worked the most efficiently under this technique.

Technologies, Techniques, Frameworks

Creating a modern and responsive web application in 2023 is a little different than it was 10 years ago. Gone are the days of pure html webpages importing JavaScript and CSS. Today, we have frameworks developed in JavaScript to help organize our file tree and create single page applications (SPAs).

The most popular of these frameworks is ReactJS, created by *Facebook/Meta* in 2013, otherwise known as just React. React is a JavaScript framework that specializes in user interface (UI) and responsiveness using what it calls “state”. The state of a React component can be stored in a variable and updated using a function. Upon updating the state variable, any react component that uses it or is passed the state variable is re-rendered and updated with the latest information from that state. On its own, React is only capable of creating SPAs but with the addition of another framework, NextJS, we can travel and store information between SPAs.

NextJS is a framework for React developed by *Vercel* in 2016. It allows React SPAs to be rendered server side and the resultant webpage is static. We will not be using this functionality; however, we will be using the *router* function built into NextJS that allows us to travel or “route” between pages. NextJS is open source and is used by many large companies including *Walmart, Apple, Nike, Netflix* to name a few, and was contributed to by *Google* in 2019 with 43 pull requests to its git repository.

Then, we had to become familiar with Supabase and its JavaScript API. Luckily, Supabase, also being open source, has an extensive and descriptive documentation with easy to read and understand API examples that made interacting with our database from our application very seamless and easy to implement.

Finally, our previous experience developing websites from CS 316; Web Programming was very useful when styling the UI of the application as the vanilla CSS format was the same in NextJS.

Non-Technical Lessons Learned

While working on our project, we gained invaluable non-technical lessons that contributed to the project development and towards our own personal development. One big takeaway was the importance of communication and teamwork. As we dove into ReactJS, NextJS, and Supabase, we found that sharing our individual discoveries and insights made it much easier to tackle the challenges we faced. We noticed this especially during our initial pair programming sessions and when we switched to the agile sprint approach later on.

Another lesson we learned was how crucial it is to manage our time well and set realistic expectations for ourselves. Throughout the project, we ran into tough moments trying to implement features like the authentication service, which made us rethink the scope of our project. In hindsight, we could have saved ourselves time and headaches if we'd recognized our limitations and adjusted our goals accordingly.

Lastly, we discovered that being flexible in our development process can lead to better efficiency and better code. By incorporating agile sprints into the development process, we discovered that we worked and implemented user stories a lot quicker and more efficiently.

Conclusion

In conclusion, our experience in developing the Modern Funding web application has been both challenging and rewarding. As a team, we have gained experience with cutting-edge technologies such as React, NextJS, and Supabase to develop full stack web application that connects private investors and borrowers, offering them a secure and user-friendly platform for their financial interactions.

Throughout the process, we learned the importance of communication, teamwork, time management, and adaptability, which not only contributed to the project's completion but also to our own personal growth. Despite the challenges we faced, we believe that our experiences have provided us with valuable knowledge that will serve us in our future endeavors as computer science students and future graduates of the University of Kentucky.