Final Update

CSCI 4805-10 Spring 2024

## Introduction

HobbyPro is a web-based, mobile-friendly project management application created for individual hobbyists. We created this application to help hobbyists better organize and better visualize their projects in a more structured way. This application hopes to solve problems for hobbyists who might have trouble keeping track of their projects or do not plan and fall behind on completing something that they might have needed done sooner. We hope that with this application we can make hobbyists have a smoother time managing their projects ongoing or not.

Technology

* HyperText Markup Language (HTML): our project is website based seemed logical
  + Most of us knew how html worked and did not have too much of a problem using it or having to relearn HTML
  + It is very basic so having to work around with it was challenging but other technologies helped speed up the process of creating navigations displays and other items that are present on our webpage.
  + This was standard and other technologies would not have been the same as this as this language was made for webpage applications.
* Cascade Style Sheet (CSS): this kind of goes hand in hand with HTML so was a must.
  + A few of us understood CSS right away, but we implemented Bootstrap to save time using CSS.
  + This language was basic, so with no cool shortcuts we ended up implementing Bootstrap to speed up CSS work.
  + We swapped to bootstrap right away as it saved us time using it as shortcutting CSS calls made using CSS simpler.
* Vue.js: is an all-in-one Website tool as it combines HTML, CSS and JavaScript into components that can be used and implemented on views which allows us to have seamless page traversal.
  + Sam fletcher: I studied this with the help of tutorials and documents and was brought up to speed and knew how to both manipulate the data using Vue built in calls and functions and assisted those who needed help with Some of Vue's v commands in the html part. I also got assistance whenever there was something I did not understand that I found out later.
  + It made the technology easy to manipulate with the help of Router calls and components was a bit difficult to set up but makes traversal through the webpages seamless.
  + We did not propose an alternative to Vue.js.
* Vuex is a state management pattern and library for Vue.js applications.
  + Vuex was added to the project a little too late. We had begun designing how information was passed around the different components and were having trouble managing state effectively. This led to using this library for Vue state management and it has a learning curve to it.
  + I (Dustin) am certain that at least one bug is a result of my limited knowledge of Vuex implementation, and I am incorrectly handling system state.
* Bootstrap: A CSS library used to create responsive mobile-friendly web applications.
  + Learning the bootstrap commands was simple, remembering the bootstrap commands is a different story.
  + It made resizing and scaling of the websites structure easier to manipulate with the help of its row and column tags
  + This was the alternative to CSS we had. Although we still had to perform hard coded changes here and there to ensure the structure was accurately changed.
* Firestore/Firebase is a comprehensive platform provided by Google for developing mobile and web applications.
  + Authentication: It is a well know Google API that was pain free to get use. It is an industry standard, and it shows in its ease of implementation.
  + Firestore: The biggest problem in the beginning with Firestore was not understanding how NoSQL worked. In the beginning we debated using the real-time database from firebase and decided its design of depth loading subcollections was going to be a problem eventually. Because of this We chose firestore which offers real-time updating with a shallow query system making it a perfect chose for this project.
  + Cloud Storage: We needed a way to store images and other files and google offers the perfect solution to this issue. The growing pains here where large, where firestore was easy to use once I(Dustin) understood NoSQL, cloud Storage was not a simple implementation. A lot of the fun of using firestore was ruined by cloud Storage.

Design

Class design

This project was not so much class based but component based, so I will go through all the components of our website’s design. For the first the Views. These were put in place by Vue to layout components on the screen for the user to see. There are six views each with their own looks, two of them similar in style due to main screen navigation and the need for the nav bars. The only ones that are different are our onboarding pages which are different for each one. For this section we will go over what each view is used for, its functions and its variables included in the views.

Login view

For Onboarding we will start with the Login View which this page is your typical login screen simple box with input values. We have five constant values of which 3 are by reference: Email, Password, and Error Message. and the rest are auth which is gotten GetAuth, and router which is gotten by useRouter. We also have four imports which are ref from vue, getAuth and signInWithEmailAndPassword from Firebase/auth which help us get our values that relate to useAuth. And finally, we have the import useRouter from vue-router. Which connects to the value using useRouter. For functions included in this we have one which is Login which watches for a click event and uses the function calls SignInWithEmailAndPassword which will take the email and password values the user used and check if a user exists in the system. If they do, they will be redirected to the main dashboard. If not, they will receive an error message which will empty the password field and ask them to retry.

Register view

Another onboarding view screen is Register. The Register view has 5 input fields: First Name, Last Name, Email, Password, and Confirm Password. We only handle the validation of the fields to ensure none are left blank, and they meet the character count. All password hashing and storage is handled by firebase. Any errors displayed are firebase errors formatted to be presented in a more user-friendly manner.

The final view for onboarding is the Forgot-Password view. This view consists of one input field that takes an email. Firebase uses this email to send a link to the users email to reset the password.

Dashboard view

Project Dashboard view and dashboard view are our two main pages where information is distributed from page to page via routing calls and commands. The views themselves do not have a lot of script within but let us go over each to ensure we discuss everything. Dashboard view is our first one which the user gets routed to upon logging into the system itself we have one import line which is the navigation bar component in which by doing this we can now make a html element tag that calls the navigation bar component. Along with the navigation bar component, there is the router-view tag, which calls components to the screen when a route has been selected on the navigation bar. The only styling we did was style the content which was our router-view tag to fit within the confines of the navigation bar on both normal size and small screen sizes.

Project Dashboard View

Next is the Project Dashboard view this view goes over similar stuff to Dashboard but instead of the import in script being Navigation Bar its ProjectNavigationBar component which has a different set up from the normal Navigation bar due to the navigation being reliant on a project's stages updating the ProjectNavigationBar with more paths the more stages a user wants per Project. The rest of this view page is like that of the Dashboard view, but the html tag used to be that of a Project Navigation Bar instead of just navigation Bar.

Navigation Bar

The navigation Bar is our navigation component present on every page of the website which is attached to the view of dashboard view. This component is aligned along the top and left sides of the screen that can also be exaggerated and changed as the screen shrinks down to size. This navigation bar has three main nav links present of the page in which you can navigate to these three pages: main Dashboard, The Library, and completed projects. And the bottom of the navigation bar on the left side of the screen is a button which allows the user to navigate to the Create project page. And there are some hidden links added in as well when the user clicks on the hobby pro Title name it takes the user to the main dashboard from any page location. And at the far top-right corner of the screen the user can hover over a profile icon and se two other buttons one is a profile navigation link leading to the profile page. The other one is a sign out button which allows the user to sign out.

Script for this component isnt as extensive as other pages mainly we have four imports getAuth, and signout from firebase/auth, useRouter from Vue-router, and finally useStore from vuex we have datavalues that are isOpen for our offCanvas that appears when the page is small enough. We have a profile url which is initally null, a dropdownOpen value which has two separate values which is 1 and 2. the auth which used getAuth(), router which used useRouter(), display Name which uses the getAuth and get the current user’s display name, and finally vueStore which uses useStore(). The methods on this page are toggle dropdown which takes a dropdown id which is 1 or 2 to show a dropdown menu to the screen. When a dropdown id is swapped to true. Toggle is the method used for our offCanvas in which it also toggles its boolean variable isOpen to true to show its visible. And then we have a logout method that listens for an event in which once the logout button is pressed it logs the user out of the system. We also have a mounted method which sets up any elements that are set up on inital page load which is the profile image which make a vueStore call that looks for a current user photoURL. Once that is done in computed a profileUrl returns the profile Url for that mounted method.

Main Dashboard

The Vue template of the Main Dashboard starts with a Bootstrap div class .container-fluid for a full width container, spanning the entire width of the viewport within the Navigation Bar. In this div class the first row contains the header and buttons for sorting and filtering. The next row contains the Project Icons and uses the Bootstrap card class for its layout. The information displayed in the Project Icons is using Vue.js. For example v-for is used to render a user’s project from the database using the project id.

Script the imports for the mainDashboard page are onMounted, and ref from Vue, getAuth from firebase/auth. The last two imports are useRouter from vue-Router, and useStore from vuex. We then create 8 consts whch are Store that uses useStore() to save state, projects which is set up to be a refrence that well use within the template area. we then have auth which uses getAuth(), router which uses useRourter(), and docPath which uses auth to get the current user by email to get their projects. Then we have three consts that are fetchProjects, fetchStages, and selectProject these three would do store dispatches to obtain the data they needed and for fetchProject will store them in their refrence values. For selectProject would reroute the user to the projects overview page when the project had been selected. And for our on mounted it would fetch our projects for use on the page for the project cardds. there are some methods besides the values used on this page that help with displaying project information that is the project days Left method that takes the current deadline. If there is no deadline value, then the deadline days left value is not calculated. If the start date is closer to the deadline date than to the current date, if this sticks then the current days left will calculate as the total days between the start date compared to the deadline. But if today's date is passed the start date it will calculate the days left as the days change over time. But if the deadline is passed at any point, it will present the days left as 0. For the next method this one is simple as it sorts the project cards by their name or start dates for the users' convenience. This can also be sorted in ascending or descending order.

Vuex’s index.js page is where most of the functionality of calling the database exists, the end goal is to move all database functionality to the file to keep it all together. Right now, calls reading the database are done through the Vuex file. The file is set up in several different areas. The first is the objects that the state is being tracked. This is simply just a list of variables that are referenced at every later step. The second portion of this file is the “mutators” these are just getters and setters with some different terminology. They are implemented in a way that is very similar to getters and setters, I (Dustin) do not know what is going on under the hood. The next section is the actions. The actions are methods that affect the state variables. In our project all of these methods are aysnc calls. The first is a simple fetch projects method where the database is queried, and the user's projects are loaded into the page for the user. Because Firestore is a shallow query database each collection has to be retrieved on its own. This means when a user selects a project from their list that project is then loaded into Vuex’s state, and its information is tracked. We do this because to retrieve the sub collections within this document we need to be able to reference this document when querying the database.

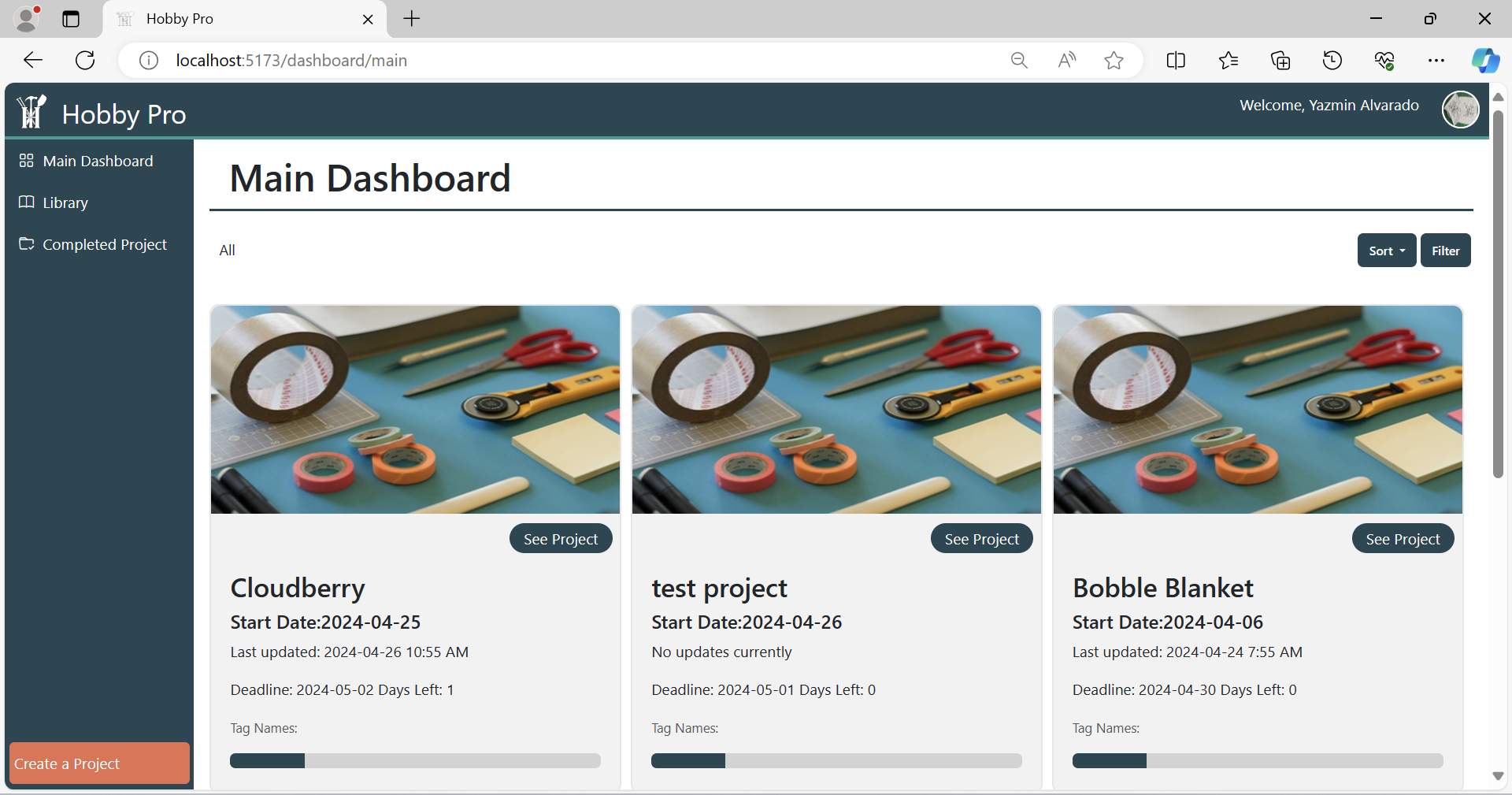


Fig 1 shows the layout of the main dashboard

Library

The library is implemented to simulate a file structure, like the ones that are found on most computers. Users can create and delete folders, subfolders, and upload files. Files that can be uploaded are pdf’s, .png’s , and. jpg’s. On initial page load, a function to fetch all the files and folders from the root directory is called. A stack is used to maintain the current directory the user is in. When a user clicks on a folder, a new path is appended to the directory stack, and another request is made to firebase to load all the resources within the new directory. The UI state is maintained by using arrays. If a user adds or deletes a folder/file, then the respective array is updated to reflect the changes. Because Vue is reactive, those changes are immediately rendered in the DOM.

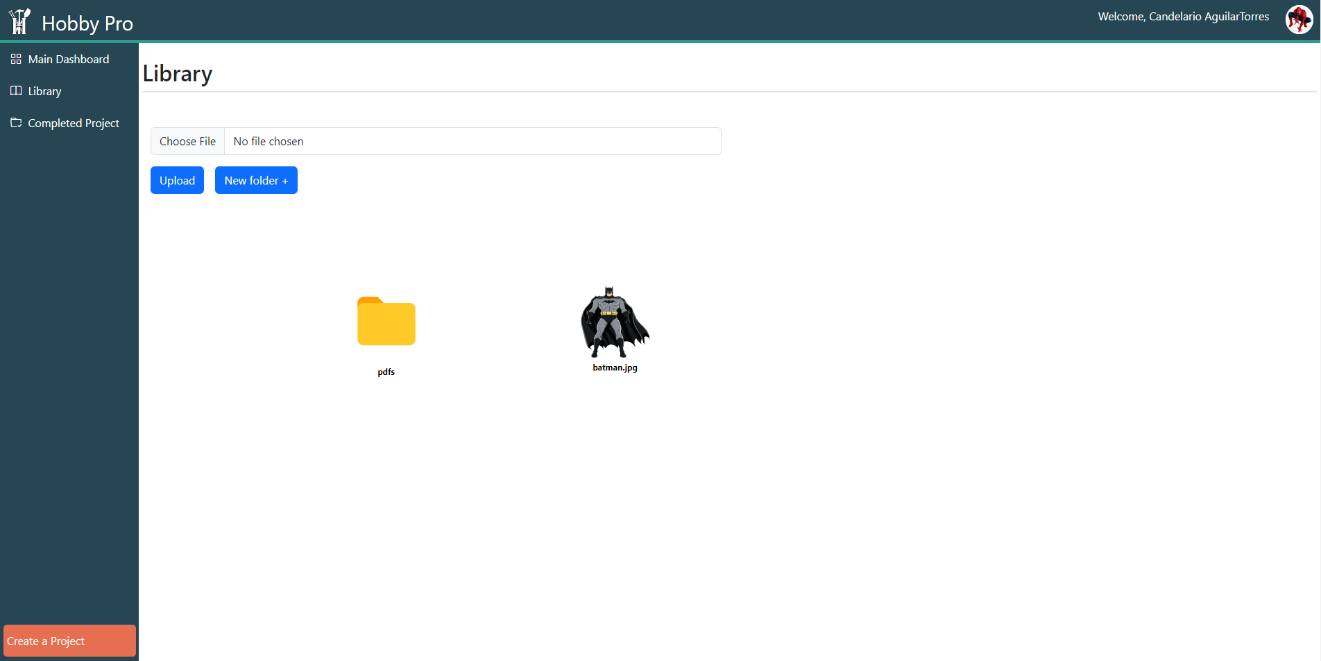


Fig 2 shows the library page

Completed Project

In the template section of the CompletedProject.vue, it starts with a div class container fluid with margin top set to 3. The next div class row contains the header and buttons for sorting and filtering. The next div class contains the project icons for the completed projects. The project Icon uses the card class of Bootstrap which uses a combination of grid and utility classes. Bootstrap also allows the cards to be made horizontal when the cards are the full width of screen and change to vertical when the cards no longer fit the viewport.

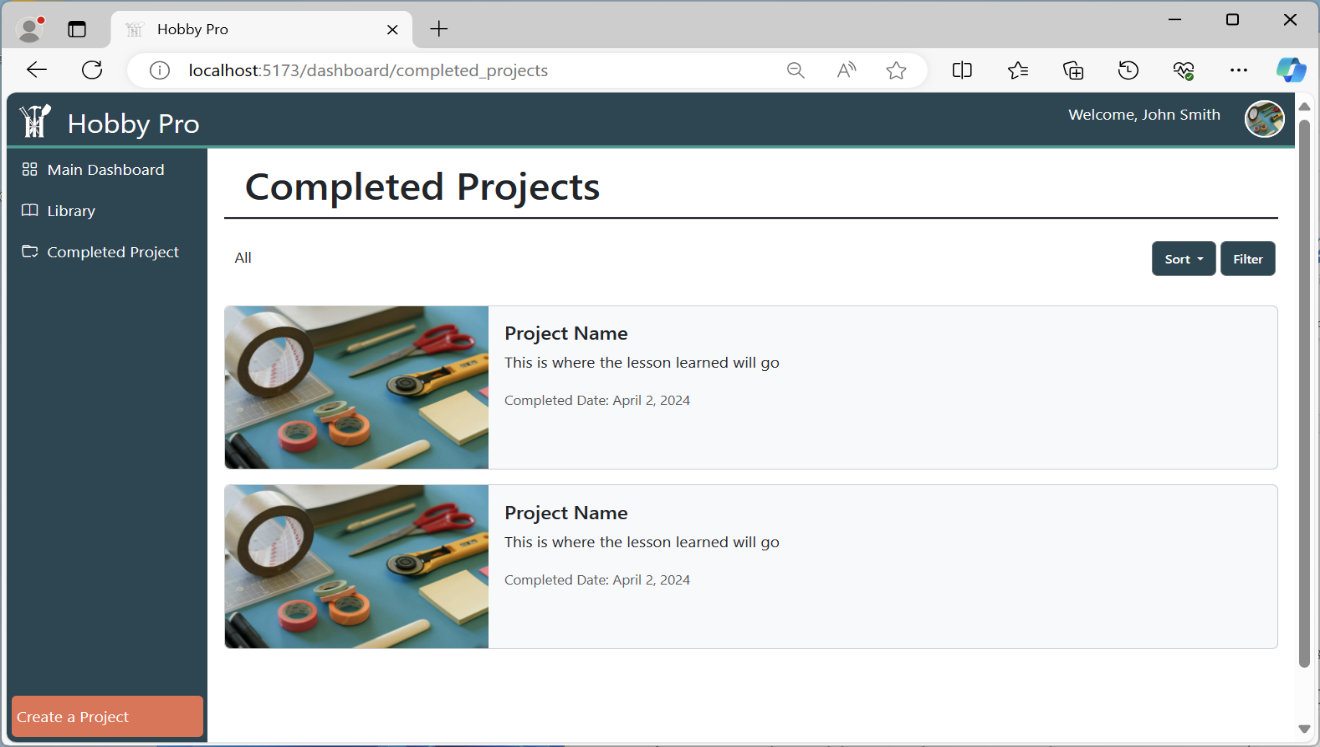


Fig 3 shows the layout of the completed projects page

Create Project

In the template section of the CreateProject.vue, it starts with a div class container fluid with margin top set to 3. The next div has a class row and within it has three class columns: 2, 8, 2. The div class containing col-2 is designed to be the gutter or spacing next to the form and center the form to the middle of page essentially being responsive. The div class containing col-8 is designed to contain the form itself. The form has its own rows and columns with the necessary labels linked to various input types. Each input type uses Bootstraps class called form-control that makes each input type have a cohesive look.

Script

In the create project component we begin by declaring all of our variables so that the user’s input can be read in real time. The component has two helper functions. The main function of the component is in the larger save function.

The first is a function that prevents the user from setting the project deadline before the project's start date. The input to this function is the value of the start date. An if statement is used to check if the user has selected a deadline date that is smaller than the start date. If they have the system gets the value of the start date and does not allow the user to select a previous date. This is done in real time, to the effect that you cannot select a date before your start date if the user has selected one.

The second helper is an add tag function. It allows the user to add multiple tags to their project during creation. Thats all it does.

The main function of this component is a save to database function. It is called “saveToFireStore” and it takes on event as input. Inside the function there is a try/catch block. Inside the try block the values the user input into the various fields is collected and saved to a document, that document is saved into the user’s projects collection. After the document is created the three initial stages are created and added to the document. If everything is successful the user is then rerouted to the main dashboard, where their new project is awaiting them. Inside the catch block an error isdisplayed to the console and the page reloads.

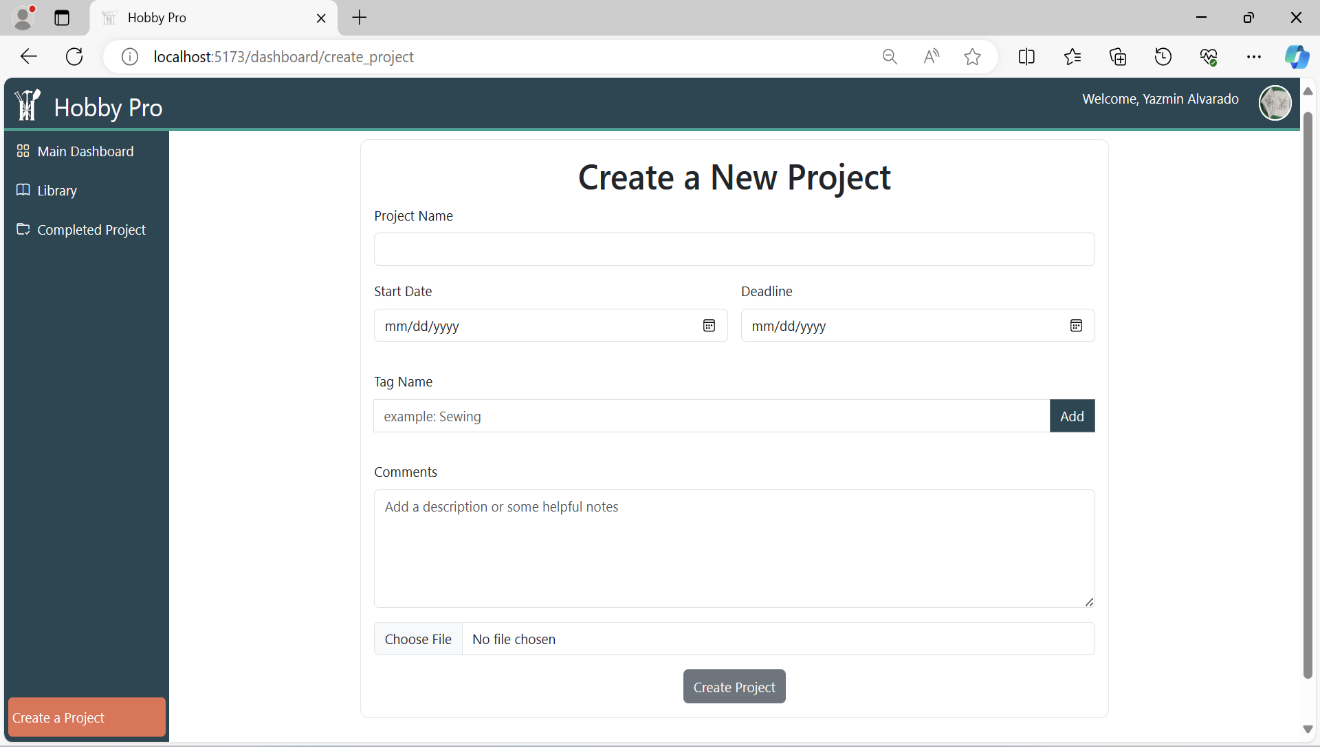


Fig 4 shows the create new project page

Profile

The Profile.vue template section starts with a div class container fluid with margin top set to 3. The next div is set to class row and within it three column classes of 2, 8 and 2. The col-2 serves as the gutters or spacing next to the profile and centers it to the middle of page essentially being responsive. Within the col-8 is the main layout of the profile. It has two row div classes to separate the profile icon image and the user’s information and statistics. The profile icon image and Change profile image button take up about 4 columns while the rest of the 8 columns are taken up by the user’s info. Labels and read-only input text using the form-control class are used to contain the user’s info. All the user's information including FirstName Lastname and Email which is loaded in by

Script

The script on this page is getting values from the current selected user we have a total of 4 imports from this page that include: computed, onMounted, ref from vue, getAuth and updateProfile from firebase/auth, useStore from vuex, and finally getDownloadURL, getStorage, ref as storageRef, and uploadBytes from firebase/storage. We have a const for vueStroe that uses the useStore() call, and we then have 6 lets that are: email that gets the current users email as a string, displayName that gets the users display name and sets it as a string selectedFile as a reference value. And a display name array that takes the display name and separates it by the space in between the first and last name. Then we have the first and last name that are set to 0 and 1 from the displayNameArray respectively which will be used in the main template. We now have four more consts which are store which uses useStore. Projects which is saved in a refrence value which will be used in a function to get project count. DocPath which gets the projects by user. And the const that fetches projects which awaits for a store dispatch to get the projects and store them in the reference value that's used in the onMounted section along with another function that gets a profile image later mentioned. We not have our const methods that get our total projects, and total ongoing projects. We then have our last two values which are completed projects since we never got our project page fully functional they are dummy values reflecting how the completed project page looks so we have 2 and a refrence for our profileImageURI. We then have two methods finishing off scrypt which are GetProfileImage() which gets the image from the currrent users photoURL and if that value is not null it will set the ProfileImageURI.value to equal to userProfileURI. Our final method is uploading an image which acts as a way for the user to upload an image to their account for profile image purposes in which they choose an image from their computer and once selected will set the url of that image into storage of the user in which it will use for future purposes of displaying to the screen.

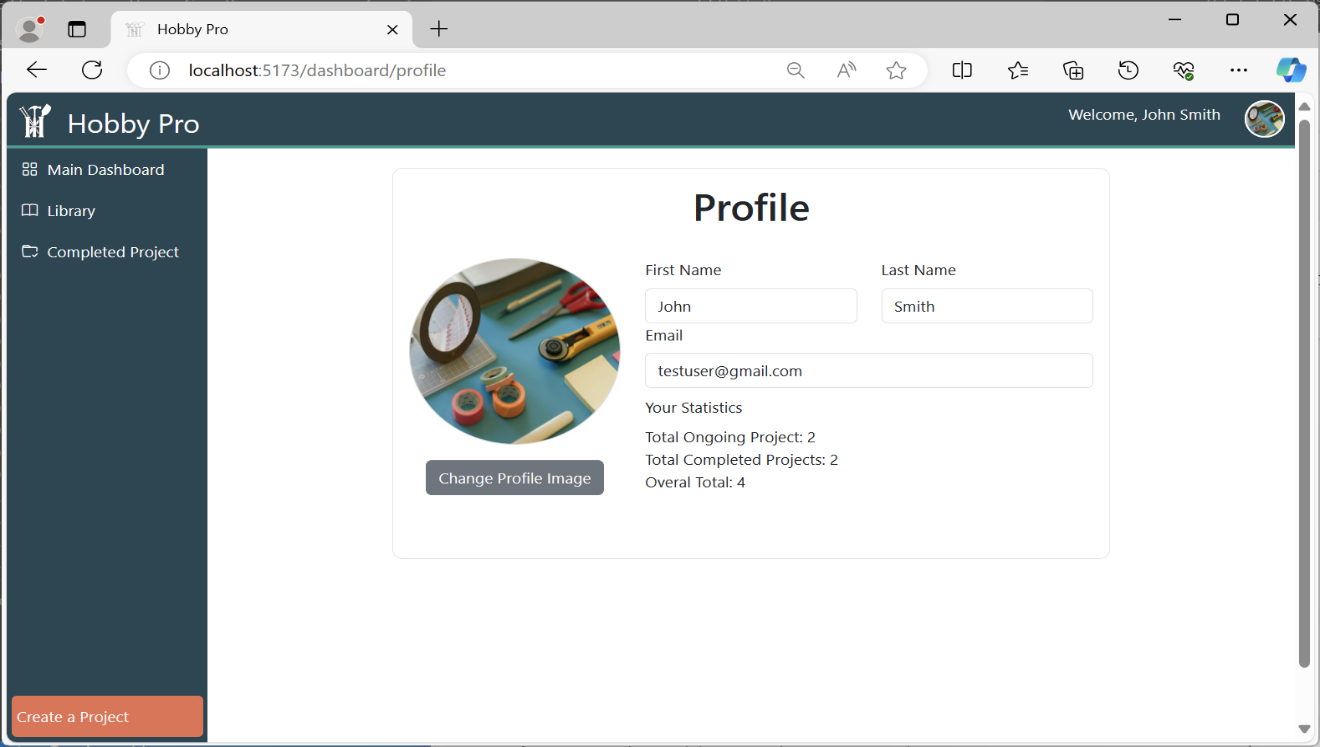


Fig 5 shows the layout of the profile page

Overview Page

The Overview.vue template section starts with a div class container fluid. The first row contains the header in a class column of 12 with a text center so that it is centered no matter the screen size. The next div row contains class columns 2, 8, and 2. The col-2 serves as the gutters or spacing next to the overview and centers it to the middle of page essentially being responsive. The div col-8 contains all the project information within paragraph tags and is divided into “two columns” except for the text area which is at the full twelve columns.

Script

The Overview page has two functions, a function that returns the number of days left until a project's deadline happens and a call to the onMounted() function that sets the state of the variable project. Information is read of this viable to display information to the user. The function to get the remaining days left of a project is called “daysLeft” and it has two input variables a start date and an end date. If the user does not have a deadline date the function ends. The two dates are then compared on a granular level to find the delta of time between them. That data gets formatted and output to the user.

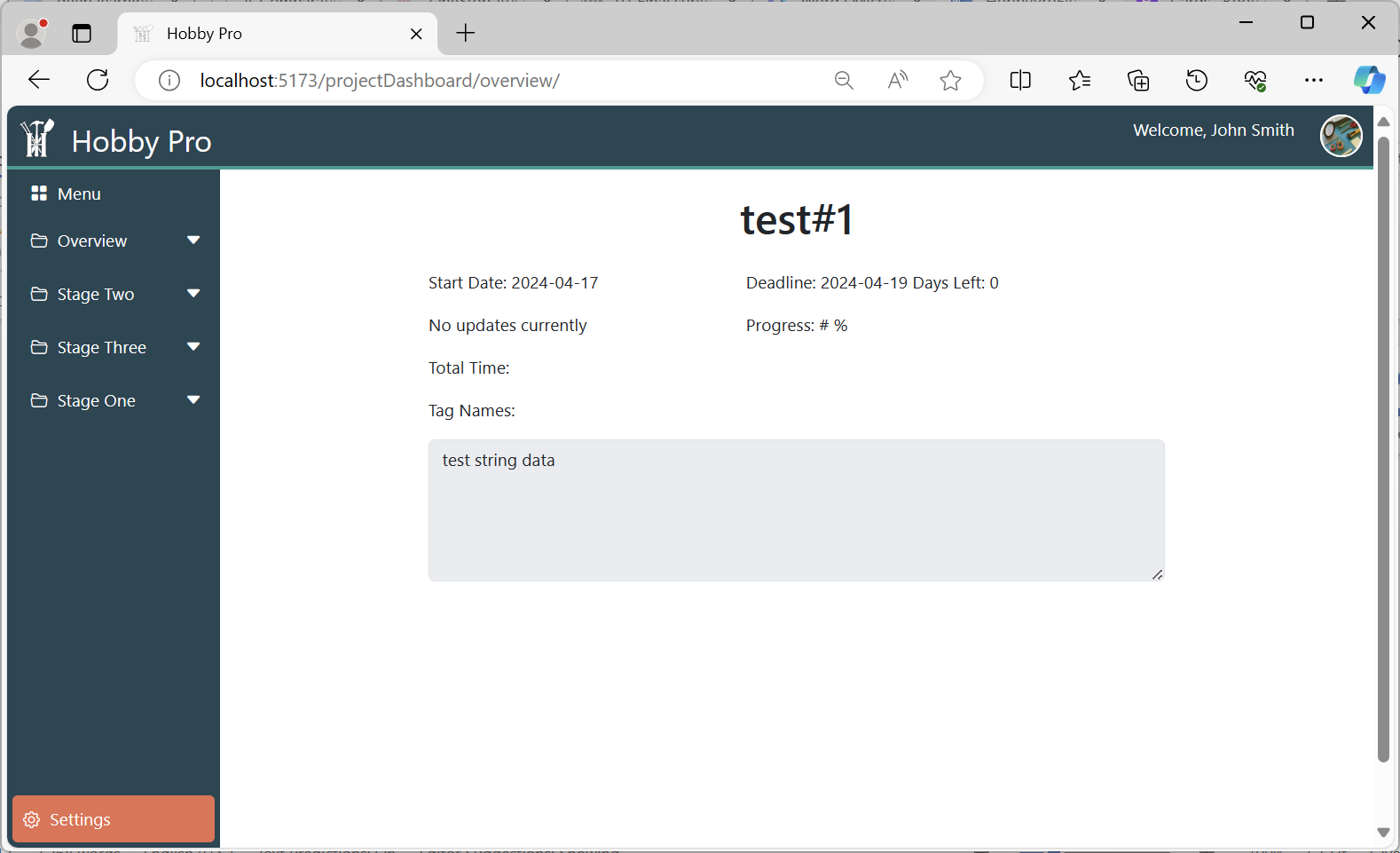


Fig 6 shows the overview page

Edit Overview Page

The EditOverview.vue template section starts with a div class container fluid. The following div row contains two columns divided at 5 and 7. The col-5 contains two rows with the project icon image in the first and the buttons to change the image in the second. The col-7 contains several rows to format everything correctly. For example, the row containing the start and deadline dates shown is using col-5 and col-7, respectively. Each label and input type uses Bootstraps' form-control or form-label for the cohesive look and style. At the end of the first row mentioned in the beginning there is the final row containing the save and delete buttons. Each button is centered taking up six columns each with Bootstraps’ modal component attached to each button respectively. Not many changes are applied to the modal component except for the modal title and content.

Script

The functionality inside the edit overview page is all designed around editing the “top level” information inside a project document. The top level is all information not contained in a sub-collection inside the document. Once the page component is loaded an object representing the top-level of a project is filled with all of the data of the loaded project. The only other function is an update function called “updateProject”. This function then compares the loaded project against any changes the user may have made and updates the according values.

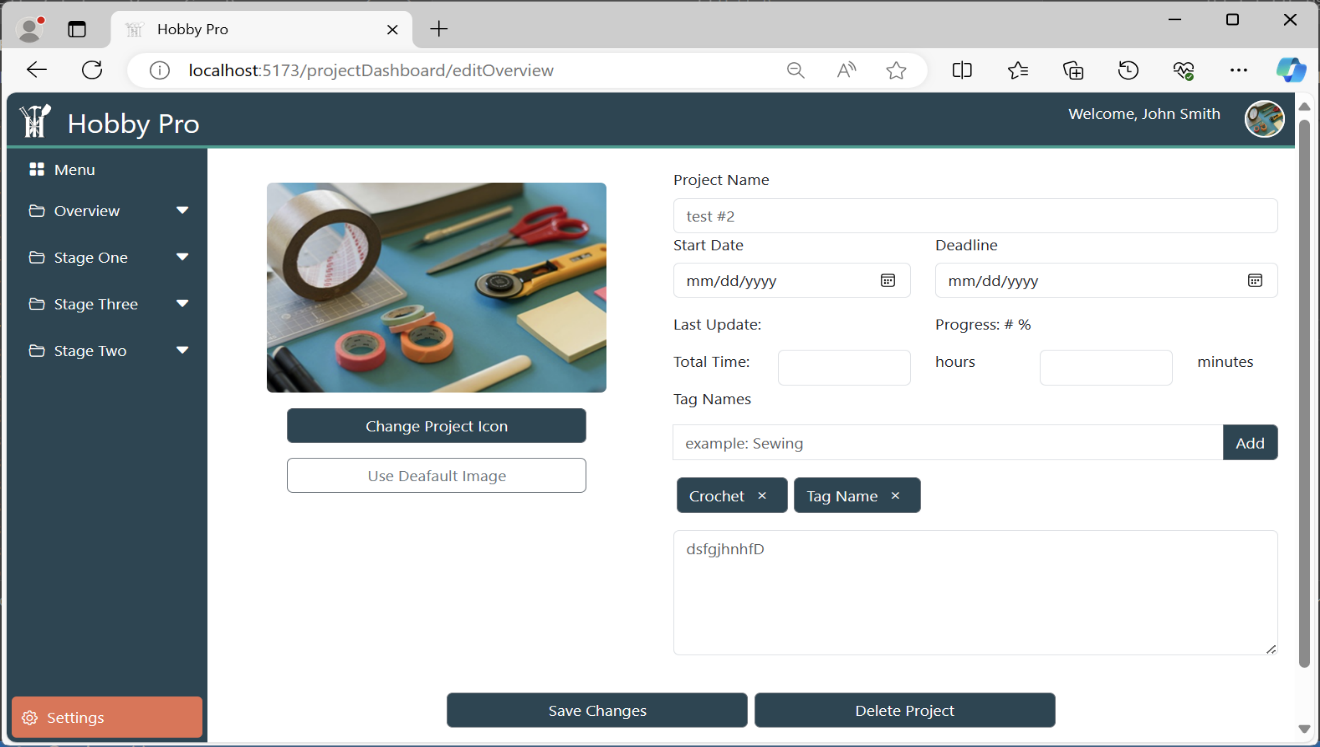


Fig 7 shows the edit overview page

Stage Page

The StageDetails.vue template section starts with a div class container fluid. The first row contains the header, the stage name, within its own div of class col-12. The next row contains the Notes and text area in two columns and are divided using class col-6. The text area has a header where each button contains an icon image from Bootstraps Icon collection. The next row contains the tasks section, and it is displayed using Bootstraps card class. Each card is set to a height of 100 and has a header with a button to add a new task.

Script

This page does not currently have any function.

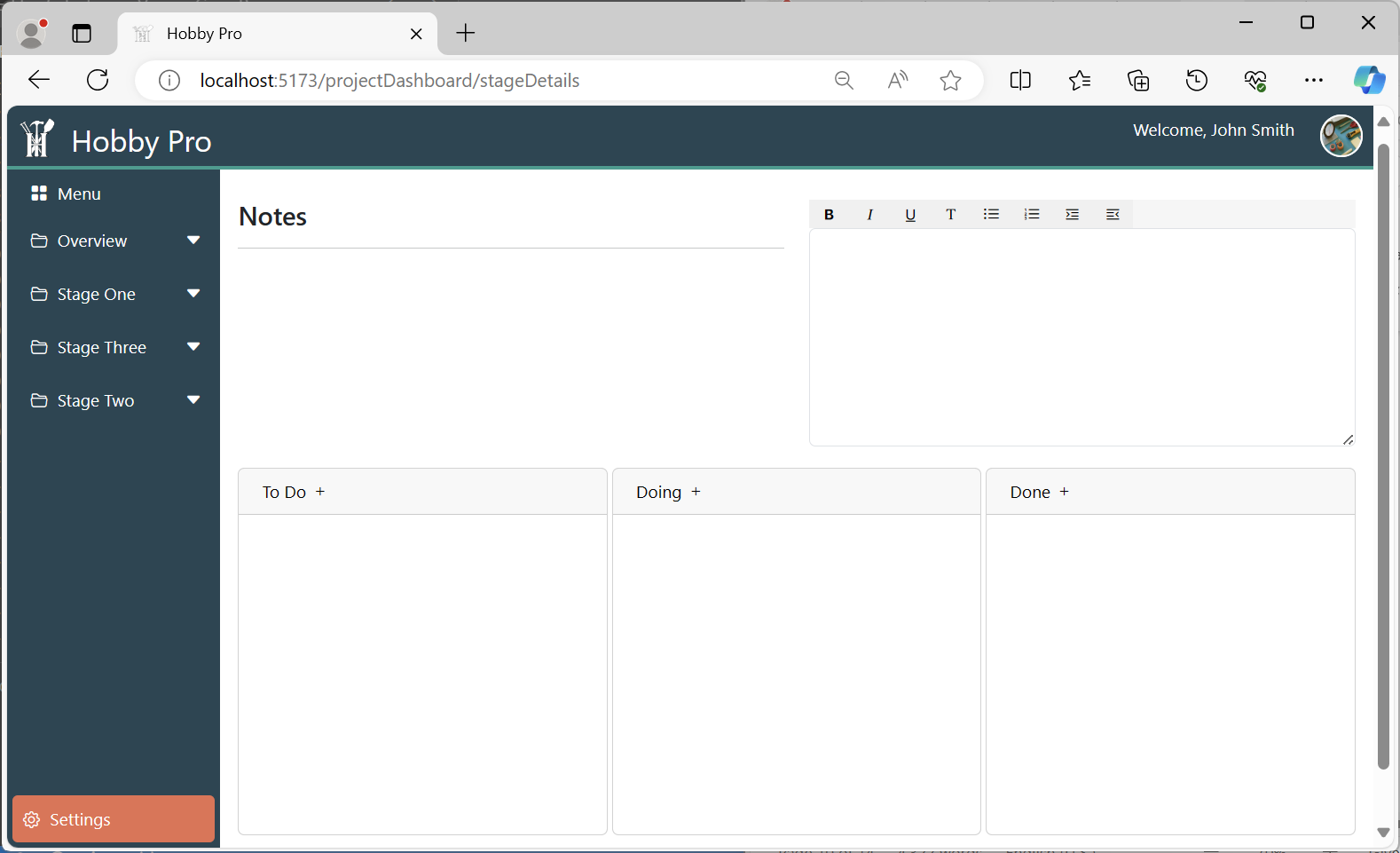


Fig 8 shows the edit overview page

### Database Design

As stated above the database this project used is Google’s Firestore database. This is a NoSQL database based on documents and collections. Documents belong to collections and every document contains key-value pairs. Whereas the structure of documents within collections is a common occurrence with NoSQL based databases Firestore shines in its real-time synchronization. Firestore uses a distributed data model and has a built-in synchronization protocol. This allows for a change made on one device to be propagated to all other instances of that database in real-time without the need for manually requesting the data. Firestore also has built in support through its service of Firebase.

Hobby Pro’s database design is a simple self-contained collection of projects per user. At the creation of a new account and the users first project a collection of documents is created using some user information to isolate collections the signed in user has access to. This collection conations only project documents. The main item the database is built around is the project. The project is a collection of several key-value pairs and two sub collections. The key-value pairs identify information about the project such as name, and description. The sub collections are a collection of stage documents, and a sub collection within each stage. The stage documents contain descriptive information about a particular stage in the project and a collection of smaller documents called tasks. Tasks are the granular level of project description and functionality. Below is a Json reorientation of a user project. The overall design includes a smaller document for each user. Store in this document is the users name both first and last and their login credentials. Thier login credentials are just the user’s password, the system now longer saves passwords, hashed or not.

A computer screen shot of a computer code

Description automatically generated

Fig 10 shows a layout of the user's documents (update: passwords are no longer saved)



Fig 11 shows the design of a user project document with included subdocument

### Clone, Build, and Deploy

Run locally

In the terminal-

1. Clone the repository:  *git clone* [*https://github.com/Caguilar92/HobbyPro.git*](https://github.com/Caguilar92/HobbyPro.git)

2. Navigate into the hobbypro directory: *cd hobbypro*

*3.* Install project dependencies: *npm install*

3. Run the development server: *npm run dev*

4. Go to the URL the server is running on (should be something like *http://localhost:PORT\_NUMBER/)*

Set Up project for deployment (only needs to be done once):

In the terminal-

\*\*\*\* run command before initializing deployment: *npm run build*

1. Install the firebase CLI by running: *npm i firebase-tools –D*
2. Check to make sure your logged into the correct firebase account: *node\_modules/.bin/firebase login:list*
3. If not logged in, log in using: *node\_modules/.bin/firebase login*
4. If you have multiple projects, select the project you want using: *node\_modules/.bin/firebase use <project id>*
5. Initialize deployment: *node\_modules/.bin/firebase deploy --*o*nly hosting*
6. When asked use public folder, type: */dist*
7. When asked to configure as single page app type: *yes*
8. Then asked to set up with git hub automatic builds: *no*
9. When asked to overwrite index.html, type: *no*

## Deploy Project:

*npm run build*

*node\_modules/.bin/firebase deploy --only hosting*

Known bugs  
a. Your system probably has bugs that you are aware of but have run out of time and  
cannot fix. Explain each known bug and provide an explanation of the approach the  
team will take to fix the bug for the next release

* There is a bug in which iOS Iphone's navigation bar will not appear when the navigation bar menu button is pressed.
  + We will proceed with a general fix in which we plan to accommodate the page navigation on both main dashboards' navigation and project with setting to detect iOS devices so that visibility will be fixed when the button is pressed.
* There is a known bug in which the data on all the project details pages including overview and edit overview will have data disappear upon page reload.
  + A fix for this could be implementation of a local DataPort for the data to remain persistent on those pages so that upon page load or any subsequent reloads the page will not experience loss of data values state.
* There is a known bug in which stages will most of the time when navigating to the project page will either show: no stages, one initial stage, or stage 1 – 3 in the navigation bar. Which is inconsistently changing, unintended, and not an intended behavior, supposed to be the one initial stage that shows up only.
  + Unknown I am scratching my head myself. We might have to check the database for old information possible sharing ids with other stages info. Or do a deep scan of the problem to produce a proper solution to fix this bug for the next release.

## Link to Licenses

Images

Photo by Jo Szczepanska on Unsplash (https://unsplash.com/photos/assorted-color-office-items-on-table-9OKGEVJiTKk)

<https://unsplash.com/license>

Libraries

CSS: <https://github.com/primer/css/blob/main/LICENSE>

Bootstrap: <https://github.com/twbs/bootstrap/blob/v5.0.2/LICENSE>

Vuex: <https://github.com/vuejs/vuex/blob/main/LICENSE>

Nodejs: <https://github.com/nodejs/node/blob/main/LICENSE>

Services

Vue: <https://github.com/vuejs/vuex/blob/main/LICENSE>

VS-Code: <https://github.com/microsoft/vscode/blob/main/LICENSE.txt>

Firebase/Firestore: <https://cloud.google.com/terms/> <https://developers.google.com/terms/> <https://firebase.google.com/terms/data-processing-terms> <https://firebase.google.com/terms/crashlytics> <https://firebase.google.com/terms/billing>

Languages

HTML: <https://github.com/whatwg/html/blob/main/LICENSE>

JavaScript: <https://github.com/GitbookIO/javascript/blob/master/LICENSE>