

**Pairings**

Find Your Match.

Project Team 3:

Brandon Banner

Francisco Serratos

Scott King

Sogol Ghotbi Taheri

Suqoya Rhodes

UNCG Computer Science SPR2022

Senior Capstone (CSC490-02)

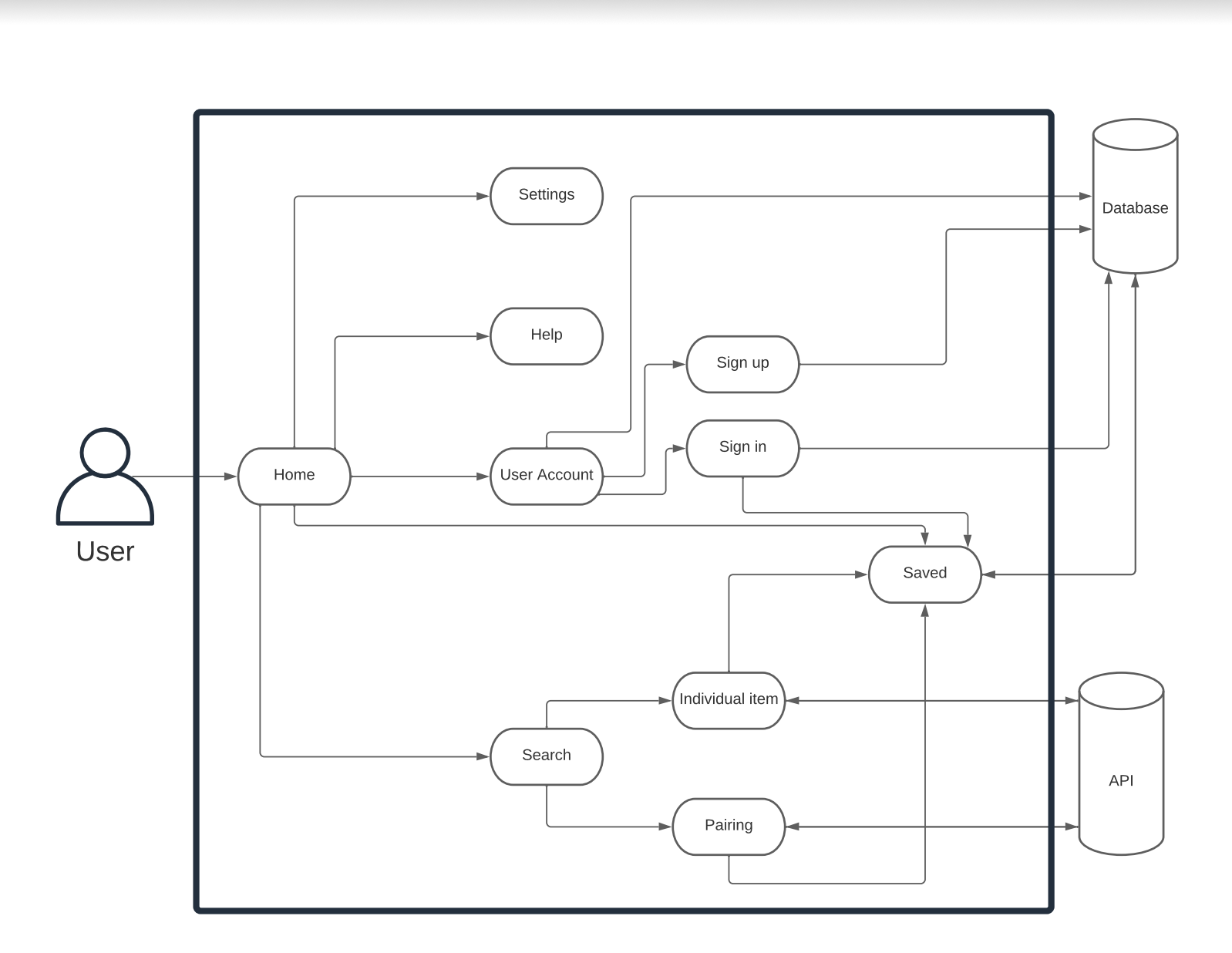
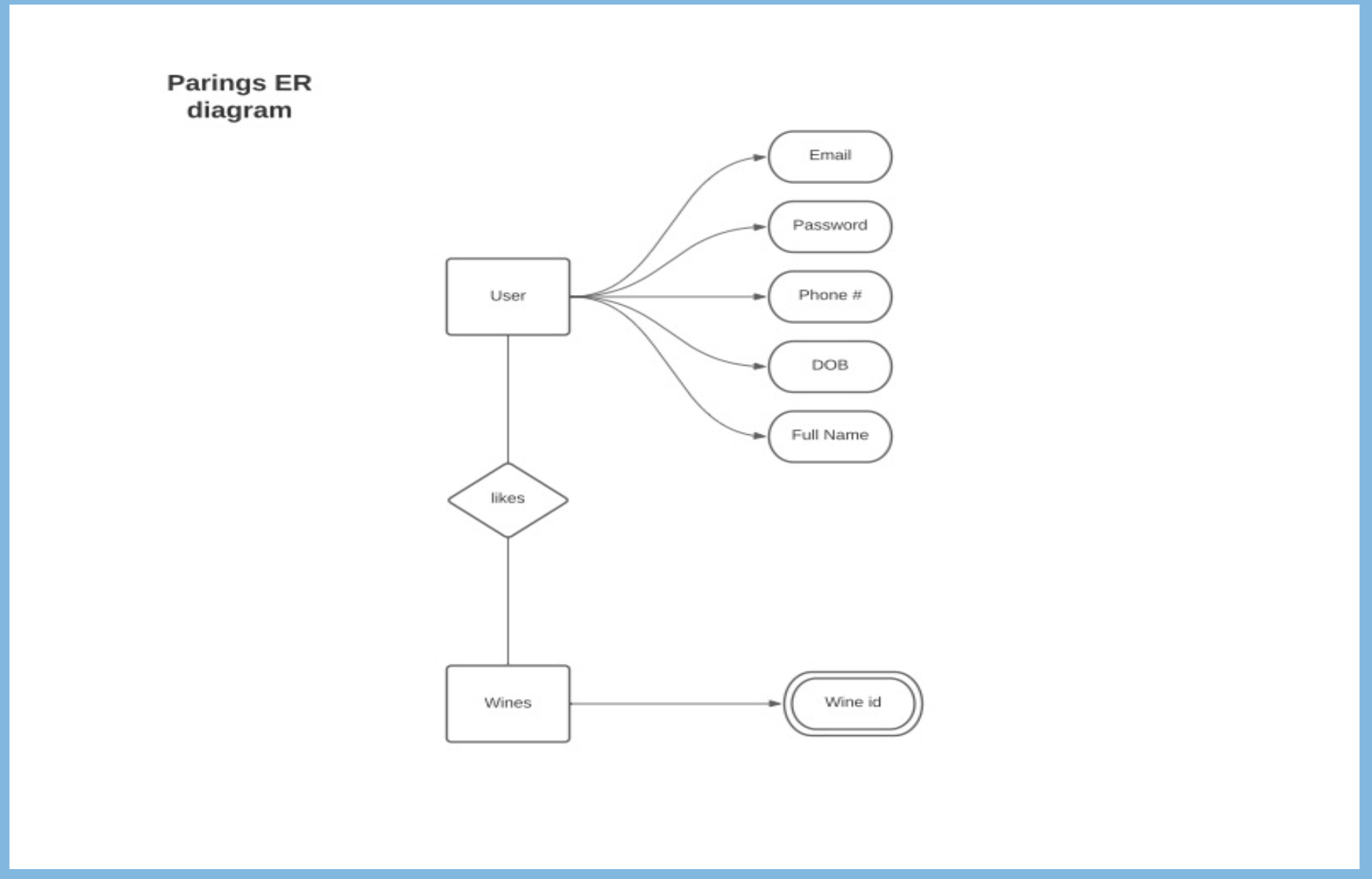
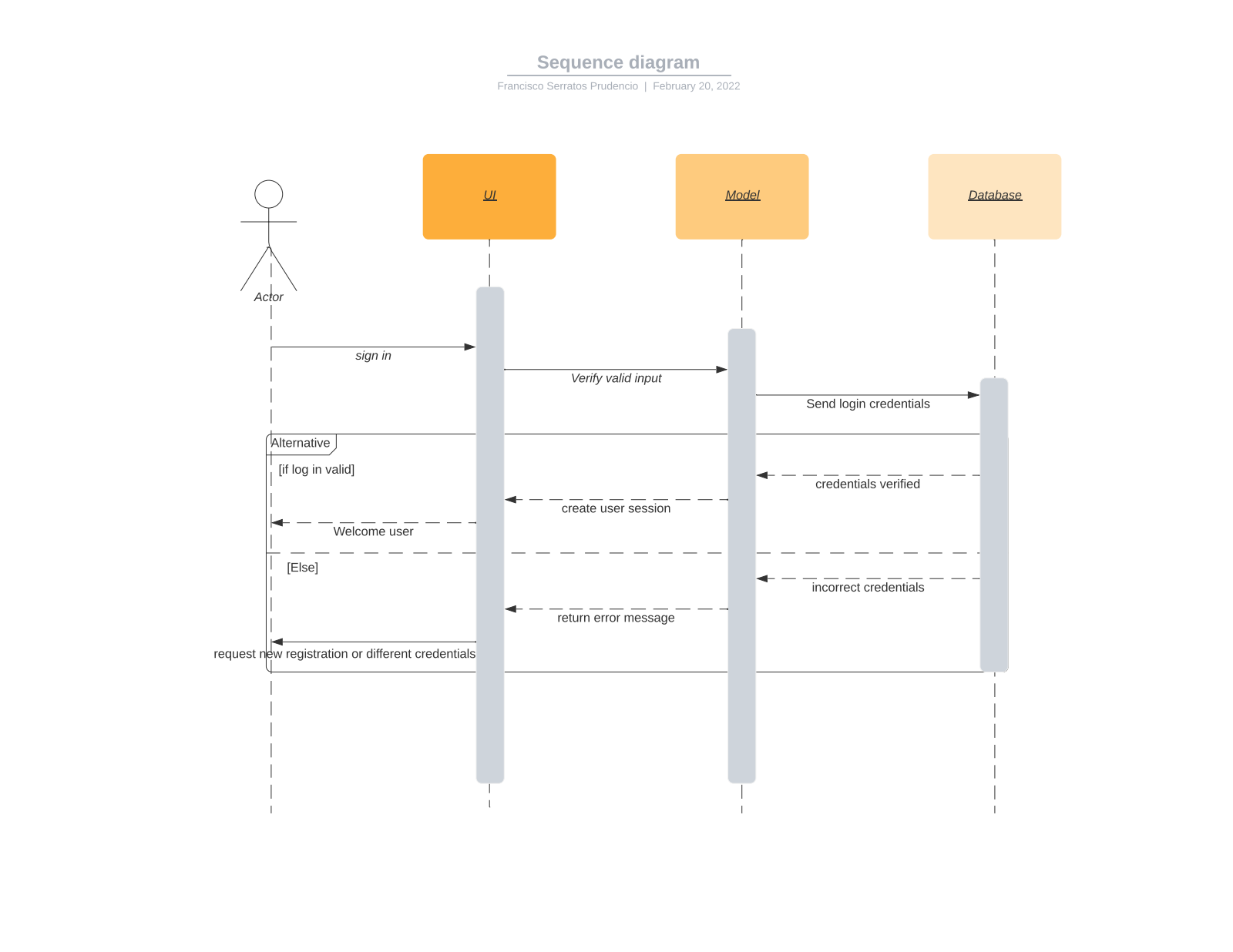
Table of Contents

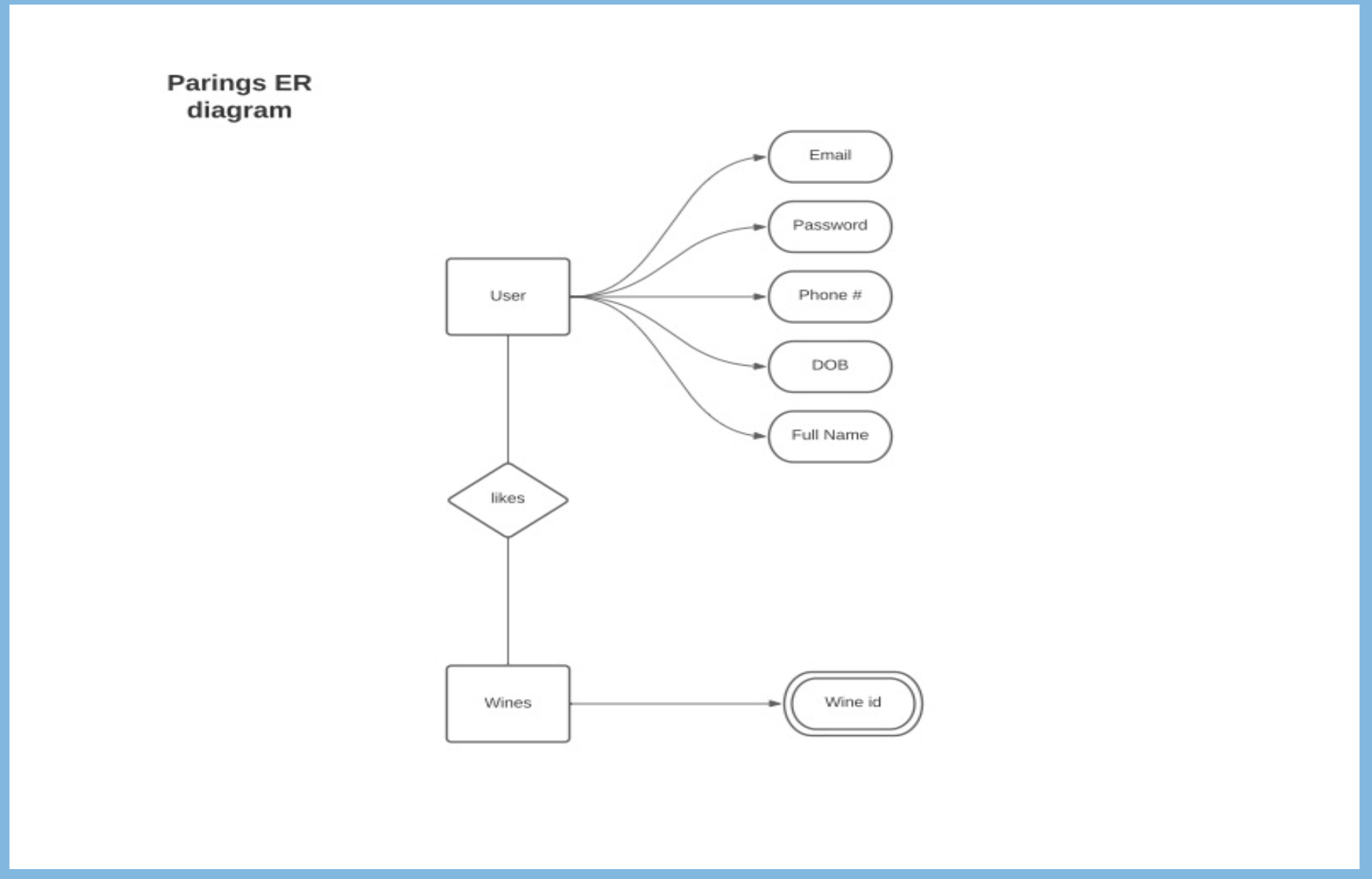
1. **Project Definition**
   1. Why
   2. What
   3. How
2. **Project Requirements**
   1. Functional
   2. Usability
      1. User interface
      2. Performance
   3. System
      1. Hardware
      2. Software
      3. Database
   4. Security
3. **Project Specification** 
   1. Focus / Domain / Area
   2. Libraries / Frameworks / Development Environment
   3. Platform
   4. Genre
4. **System – Design Perspective**
   1. Identify subsystems – design point of view
      1. Illustrate with class, use-case, UML, sequence ..... diagrams
      2. Design choices
   2. Sub-System Communication (Diagram and Description)
      1. Controls
      2. I/O
      3. DataFlow
   3. Entity Relationship Model (E-R Model)
   4. Overall operation - System Model
   5. Simplified Sub-system to System interaction
5. **System – Analysis Perspective**
   1. Identify subsystems – analysis point of view
   2. System (Tables and Description)
      1. Data analysis
         1. Data dictionary (Table - Name, Data Type, Description)
      2. Process models
   3. Algorithm Analysis
      1. Big - O analysis of overall System and subsystems
6. **Project Scrum Report**
   1. Product Backlog (Table / Diagram)
   2. Sprint Backlog (Table / Diagram)
   3. Burndown Chart
7. **Subsystems**
   1. **Subsystem 1** – Name 1
      1. Initial design and model
         1. Illustrate with class, use-case, UML, sequence ..... diagrams
         2. Design choices
      2. Data dictionary
      3. If refined (changed over the course of project)
         1. Reason for refinement (Pro versus Con)
         2. Changes from initial model
         3. Refined model analysis
         4. Refined design (Diagram and Description)
      4. Scrum Backlog (Product and Sprint - Link to Section 6)
      5. Coding
         1. Approach (Functional, OOP)
         2. Language
      6. User training
         1. Training / User manual (needed for final report)
      7. Testing
8. **Complete System**
   1. Final software/hardware product
   2. Source code and user manual – screenshots as needed - Technical report
      1. Github Link
   3. Evaluation by client and instructor
   4. Team Member Descriptions
9. **Project Definition**

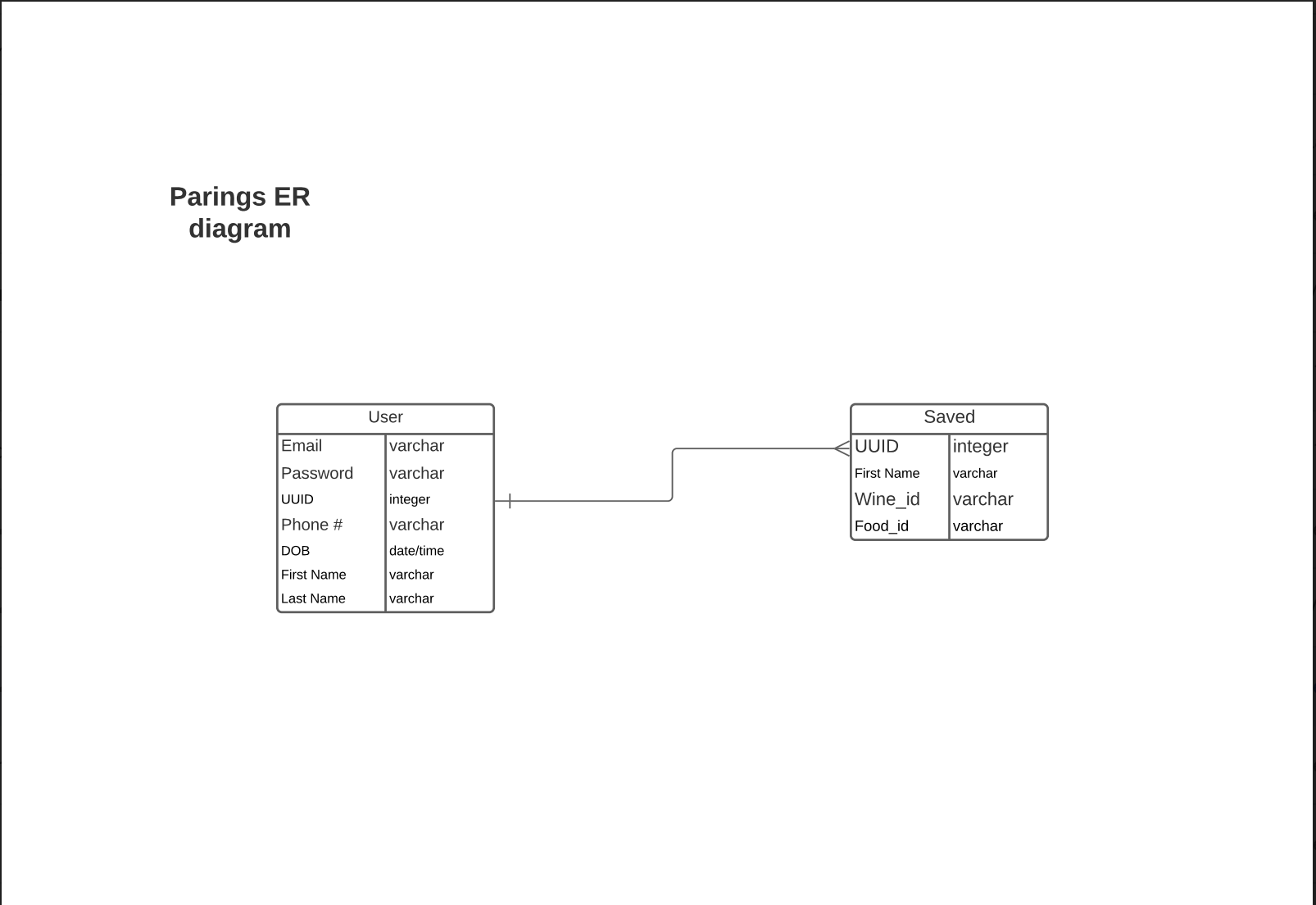
Introduction

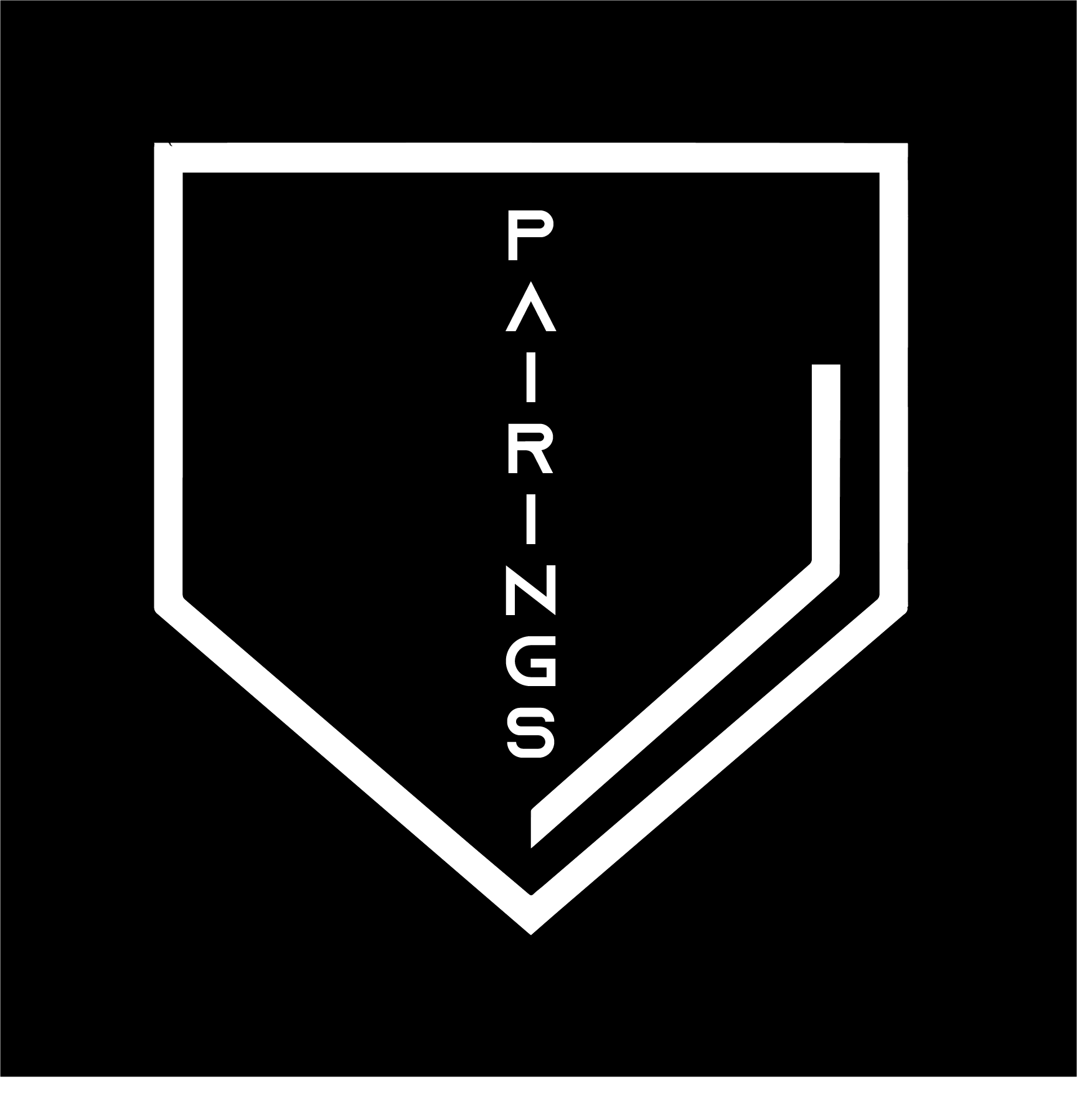
Wine has been produced for thousands of years. In today’s global marketplace, there are numerous sources of wines available from all regions of the world. Reds, whites, sparkling, mead, and dessert wines offer different experiences. The variety of grape used, the climatic growing conditions, and the strain of yeast used in the fermentation process all have a significant impact on the resulting wine. These differences can yield changes in acidity, sweetness, color, and finish, just to name a few. Almost every restaurant today offers a wine selection menu. However, the list of choices can be overwhelming, especially in the case of a non-enthusiast. A properly paired wine can complement the characteristics of a protein or dish. Similarly, a poorly paired wine can degrade or even ruin a meal. It’s these complexities and subtleties that serve as the motivation behind our application, Pairings. At its core, Pairings is a resource to improve the dining experience.

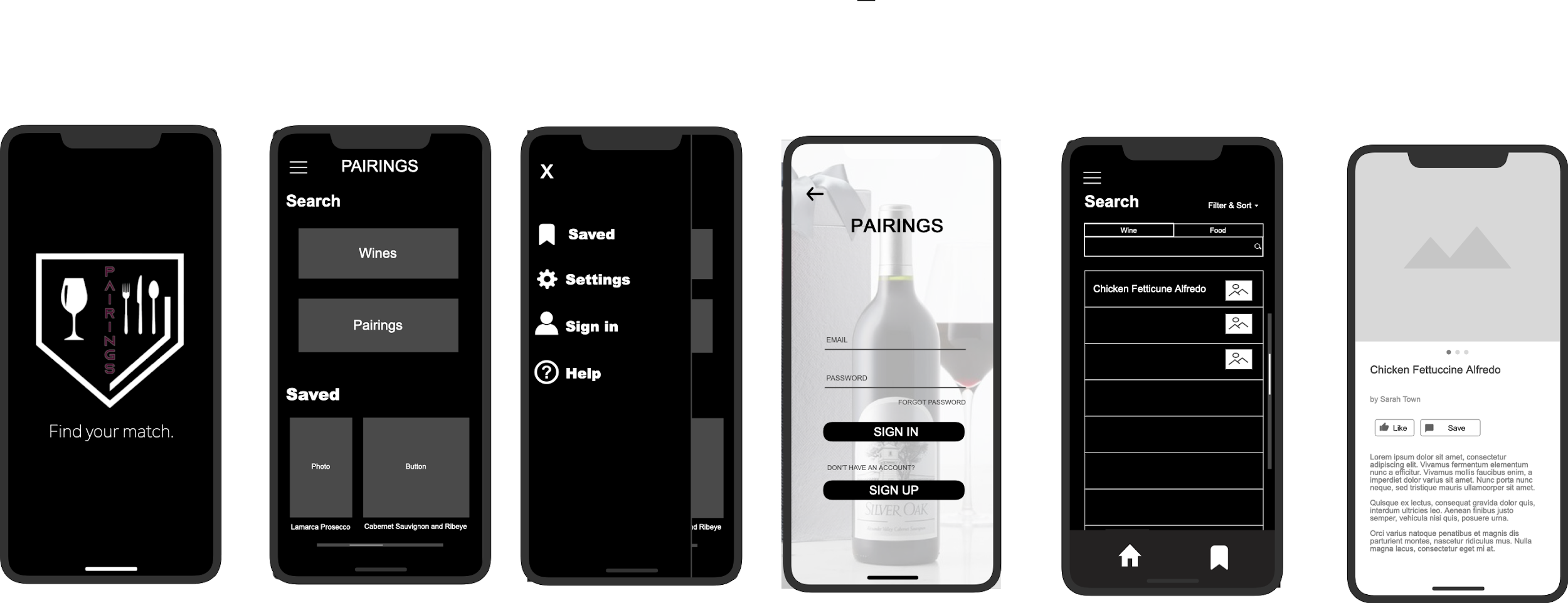
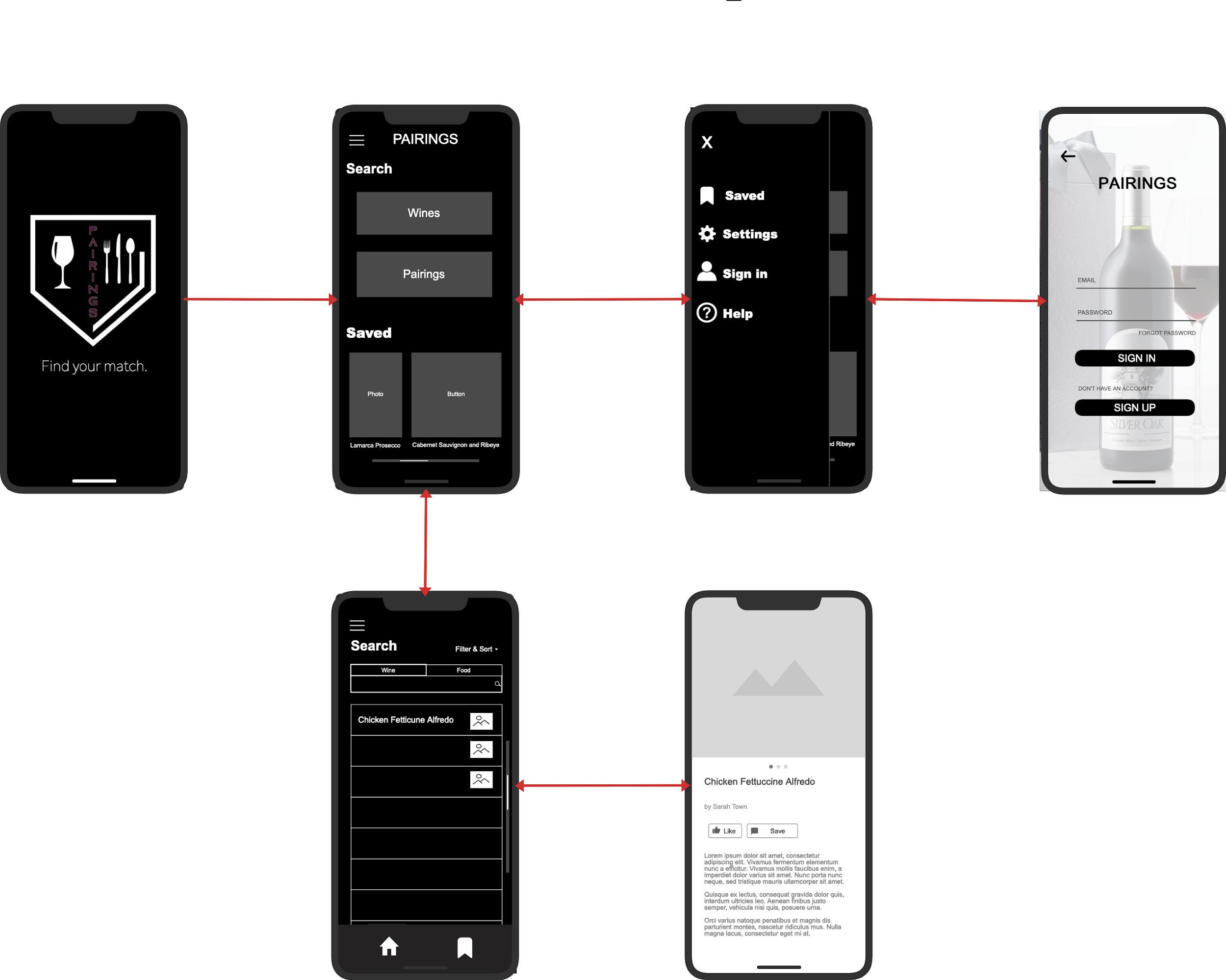
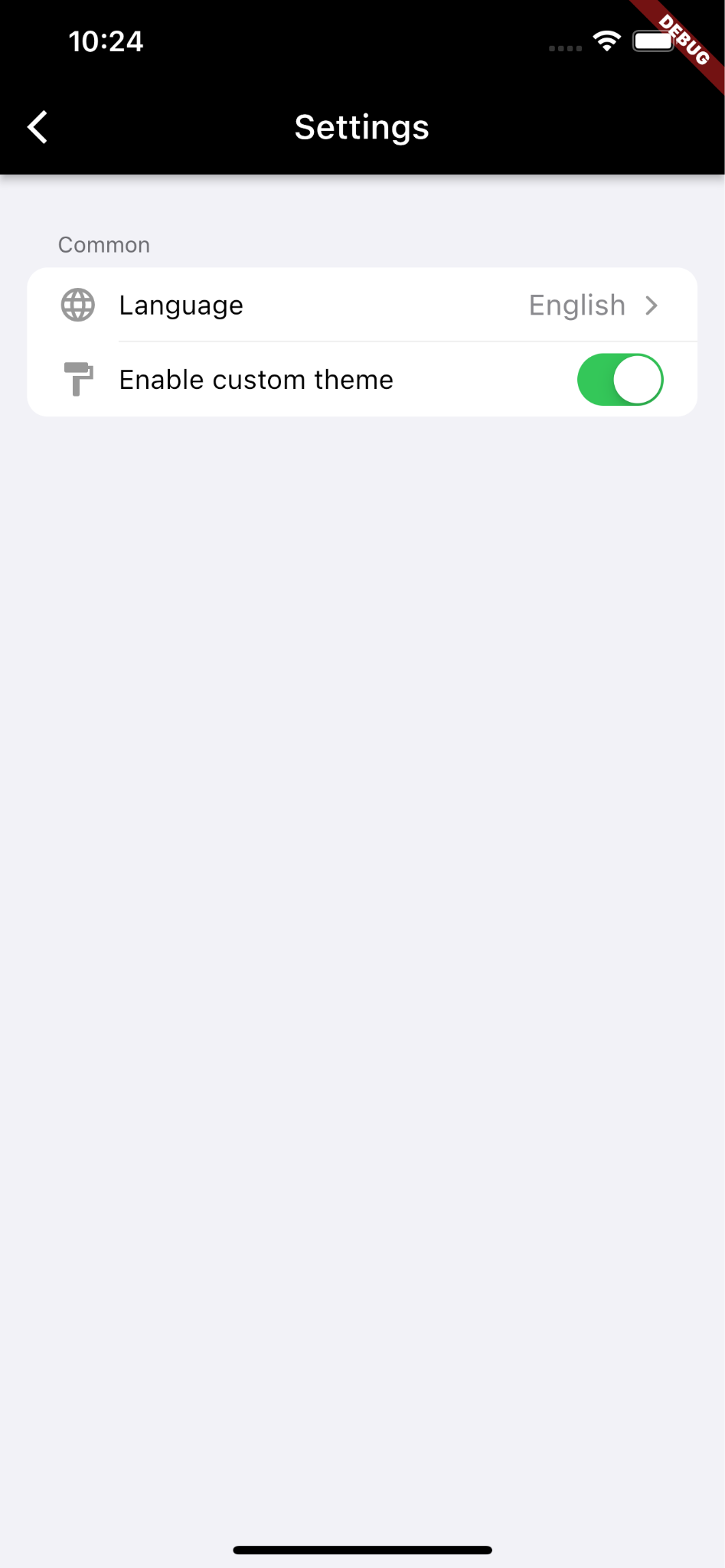
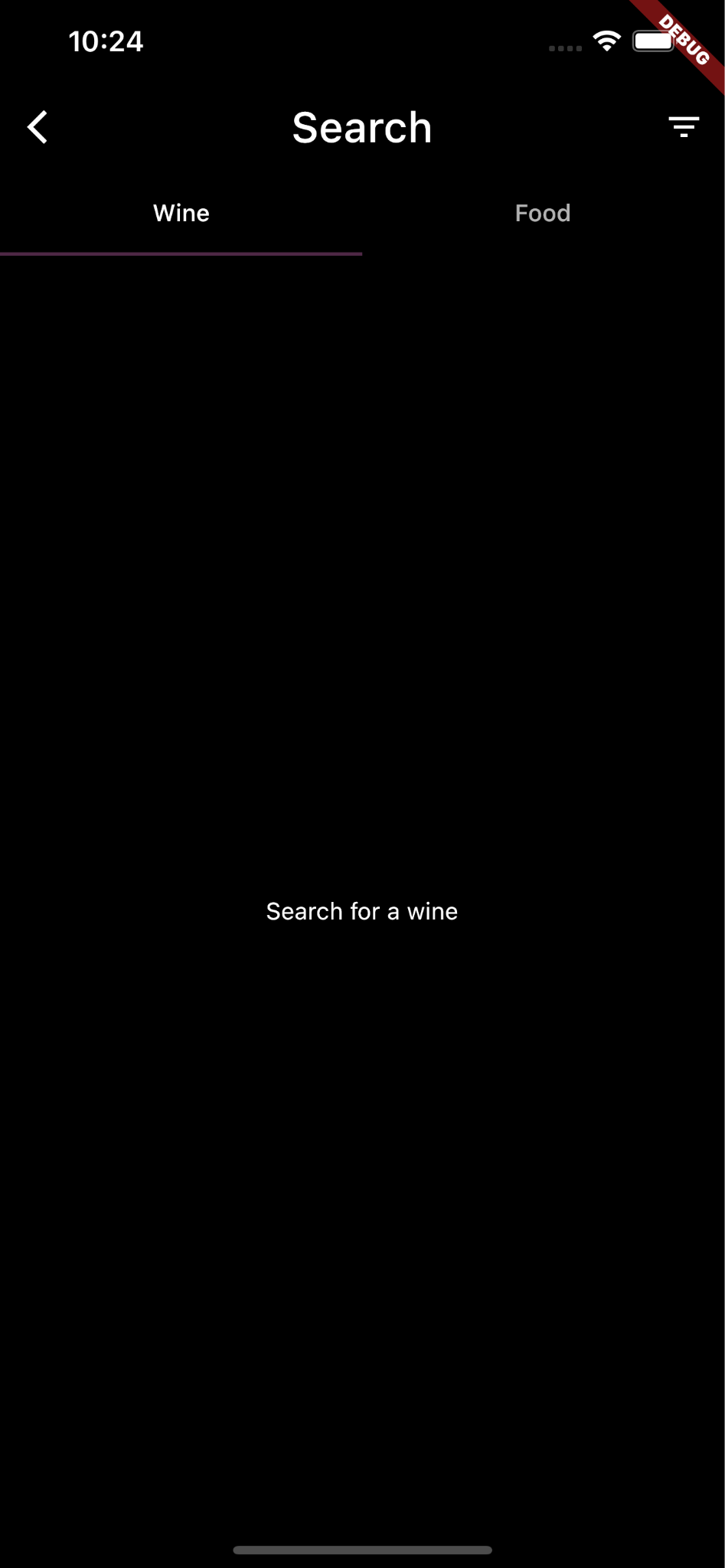
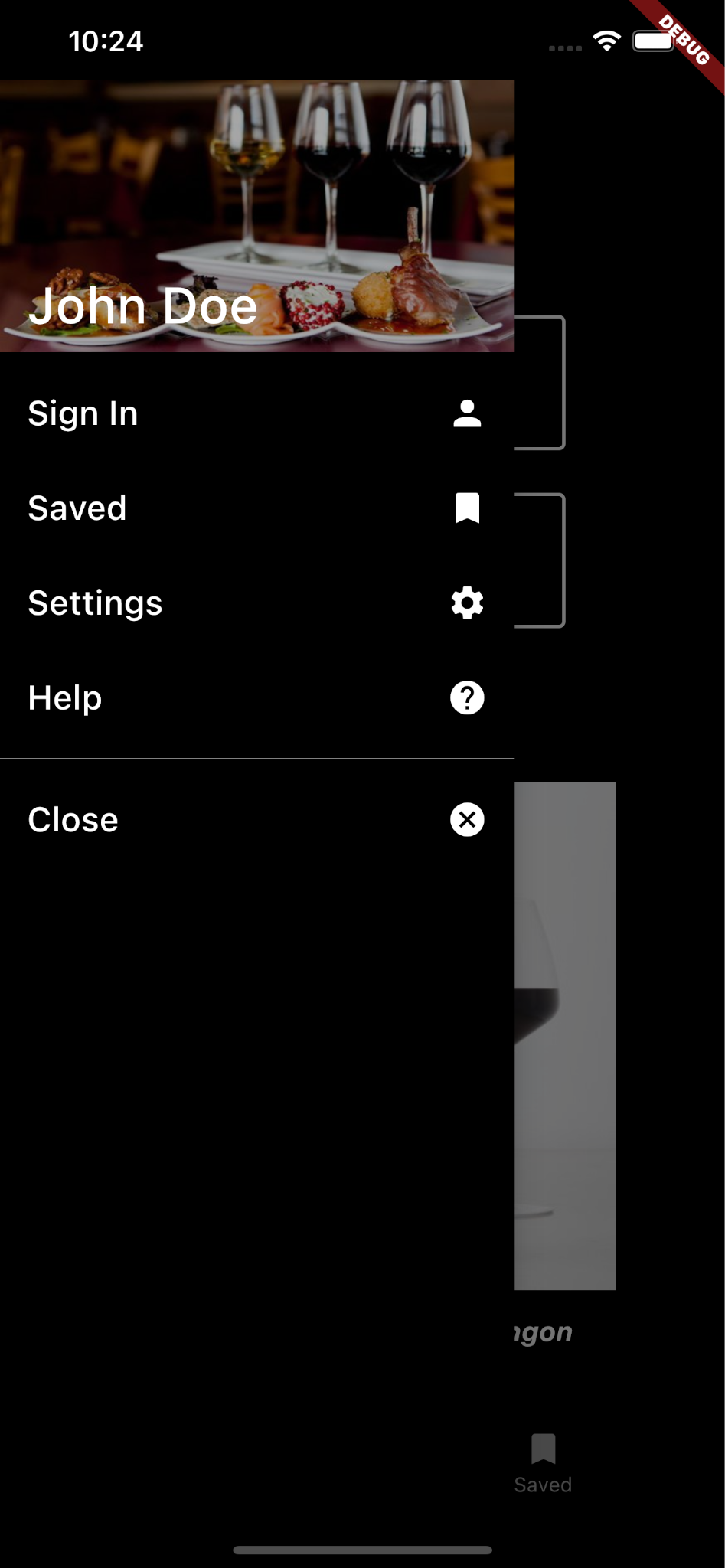
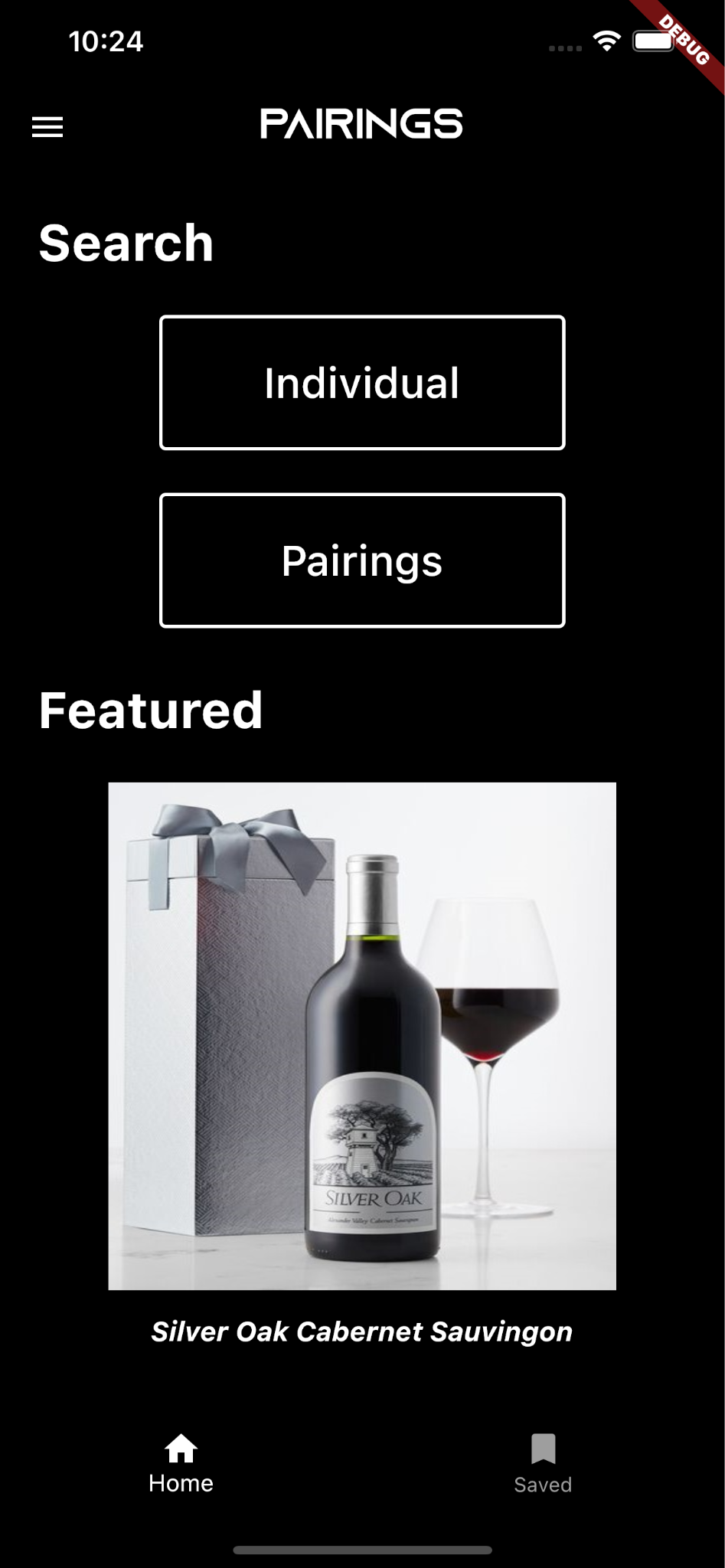
* 1. Why is the application needed?
     1. The Pairings application is needed because there are little to no applications that will generate information about wine that users are drinking and match the type of wine to food pairings and vice versa. Pairings is a quick and efficient way for users to find the right wine pairings to fit their mood and or meal. It will take the guesswork out of choosing a pairing, and make it simple even for novice wine drinkers.
  2. What is the goal of this application?
     1. The goal of the Pairings application is to create a simple, seamless experience that will allow users to discover wine that is best matched to their meal and to find food best matched to their wine. The application will be able to store favorites through the use of a user created account.
  3. How will the application be achieved?
     1. Through the use of the Spoonacular API, the Pairings application will make calls to the API to match an inputted wine to a food pairing and to also match an inputted food to a wine. The application will store and access data using AWS as the backend service.

1. **Project Requirements**
2. Functional Requirements
   1. The core functions of the application will be to find food and wine that should be paired with each other using the Spoonacular API to get the information. Users will input either a food or a wine and will receive results in the form of pairings that will match the users to paired wines or foods. Users will be able to store favorites under their account.
3. Usability
   1. User Interface - The UI will be a simple, clean design. Users need to access a mobile device, therefore the user interface will be created to be on a mobile device. The application will incorporate responsive design and the application will adjust to system accessibility settings. Those settings being font size and reduced animation.
   2. Performance - A simple user interface that is responsive, intuitive, and efficient.
4. System
   1. Hardware - Mobile device (iOS or Android)
   2. Software - Latest version of iOS or Android
   3. Database - mySQL and AWS for user authentication
5. Security
   1. Database encryption
   2. User login
6. **Project Specification**
7. Focus / Domain / Area
   1. Wine consumers (all levels of expertise)
8. Libraries / Frameworks / Development Environment
   1. Dart / Flutter
   2. VS Code
9. Platform
   1. Mobile (iOS and Android)
10. Genre
    1. Food and Drink Application
11. **System-Design Perspective**
12. Subsystems
    1. Systems
       1. Home
       2. Sign in / Sign up
       3. Search
    2. Design choices
       1. The home subsystem houses the main screen and is the hub for all other functions. The home subsystem also includes help and settings.
       2. The sign in / sign up function connects to the database to either check for authentication if the user already has an account, if not, the system to send user information to the database for registration. This will be used for the verification of the user. Once a user is signed in, the database will store the user’s information and saved wine and or food pairings.
       3. The search subsystem will take user input and search for a food or a wine to get search results from the API. If a user chooses to save an item, that saved item will be stored in the saved section under the user’s profile in the database.
       4. Use Case Diagram - [Pairings Use Case Diagram.pdf](https://drive.google.com/file/d/1VjucQTwMJvNosuA8M0WUedbDfSuF6om0/view?usp=sharing)
       5. Subsystem communication - [Sequence diagram.pdf](https://drive.google.com/file/d/1DY5dwb1s5CVeC5qFJWL-vUgJI2OHJt1l/view?usp=sharing)
13. Entity Relationship Model (E-R Model) - [Pairings ER diagram.pdf](https://drive.google.com/file/d/1R0fHuIiXJWJ2Ku0vTsM1CtmVXcqaRPye/view?usp=sharing)
14. **System-Analysis Perspective**
    1. Subsystems
    2. Home - this is the central subsystem which is the way the other subsystems are accessed.
    3. Sign in / Sign up - this is the subsystem that will connect to the database to store and retrieve user information.
    4. Search - this is the subsystem that will connect to the API to get search results, then the search subsystem will connect to the database for saving search results.
    5. System
    6. Data analysis
       1. Data dictionary - [Data Dictionary](https://docs.google.com/spreadsheets/d/1ezd7KYaBuMCnwomgOvly1q7_W8Mr0s1jB5cO4RNkO2o/edit?usp=sharing)
       2. Process models
    7. Algorithm analysis
    8. In this current iteration of our application, we are not planning to use any complex algorithms. With the omission of any complex algorithms our overall system will run as fast as the user can make a request to the application. This means that overall runtime is dependent on how the user chooses to interact with our system.The majority of our sub-systems will run at O(n), runtime is dependent on the amount of requests the user makes at any given time.
15. **Project Scrum Report**
    1. Product Backlog - [Product/Sprint Backlog](https://docs.google.com/spreadsheets/d/1OmfJj55uJtBVtONvZLoXAgkpCwutHMl4z-d0Dl_dL6M/edit?usp=sharing)
    2. Sprint Backlog - [Product/Sprint Backlog](https://docs.google.com/spreadsheets/d/1OmfJj55uJtBVtONvZLoXAgkpCwutHMl4z-d0Dl_dL6M/edit?usp=sharing)
    3. Burndown Chart - [Product/Sprint Backlog](https://docs.google.com/spreadsheets/d/1OmfJj55uJtBVtONvZLoXAgkpCwutHMl4z-d0Dl_dL6M/edit?usp=sharing)
16. **Subsystems**
    1. Database - Francisco Serratos
       1. Initial design of the database



* + 1. Refinements
       1. The first refinement was the switch to an actual ER diagram which is referenced earlier in the report. This issue occurred because of our lack of database design knowledge and so more research was needed to create a proper design we could implement for our mobile application
       2. The second refinement was the choice to create an api to communicate with the database. Initially the idea was to create a class in our application to communicate directly with the database. This was a bad idea for security reasons, so the first iteration of the api was built with Lumen, a php micro-framework. This new implementation was much easier to work with and get a working prototype.
       3. The third refinement was the switch to a more robust framework. When trying to deploy my api on Amazon web services, the lumen framework was not robust enough to handle the transition. So the move to Laravel was a no brainer. Laravel is the framework from which lumen came from so the transition was fairly easy. This change was integral in deploying the api on AWS.
    2. Coding
       1. Approach - The initial thought was to code everything from scratch but upon discovering the different frameworks, our focus switched to using a framework to help alleviate some of the stress from the time constraints. By creating an api to connect to our database, it provides us modularity if for some reason we must switch to a new database.
       2. Language - PHP is an object oriented language that is mainly used for web development. The Laravel framework which is written in PHP was made for developing web applications with ease.
       3. User manual - pending completion
    3. Testing
       1. The majority of testing for the api has been done with the help of the postman app,which has enabled us to test all our api endpoints.
  1. Front End - Brandon Banner
     1. Responsibilities
        1. Overall design language
        2. Logo creation
        3. Loading page
        4. Home page
        5. Settings page
        6. Help Page
        7. Search Page
        8. Item Page
     2. Initial design
        1. Logo



* + - 1. Wireframes
      2. Flow of interaction
      3. Implemented Design



* + - 1. Design choices
         1. Overall

Black and white, simplistic, mature theme

Minimalistic and squared off buttons and elements

* + - * 1. Logo

A logo that reflected the theme and the idea of the application.

* + - * 1. Loading

Show Pairings logo along with slogan, while app loads

* + - * 1. Home

Search options are displayed first to highlight main feature of the application

Featured section to show random wine / food of the day

App Drawer

Show options for users without overcrowding the home screen. This gives users a familiar way to access application features.

Settings

Gives users a way to adjust theme and other accessibility options (pending)

Help

Gives users suggestions and instruction on how to operate the application and provides access to get technical support (pending)

* + - * 1. Search

Select from two kinds of search: Individual and Pairings.

Then select wine or food as the basis for search

Upon searching, results will be displayed below and users can click on a list item and go to the item page to receive more information and the ability to save that item or pairing.

* + - 1. Refinements
         1. Logo

The original logo seemed too simplistic and monochromatic. The decision was made (by Brandon), to add an accent color (burgundy) and add a wine glass and silverware to reflect the dual nature of the application.

* + - * 1. Home Screen

Upon looking at the entire UI, there became a realization that there were too many ways to get to the saved page. I needed to figure out a way to fill that space that would replace the saved preview with something visually appealing. So I thought that a featured item section would be a perfect addition. This would entice users and would also give them more information about different wines and foods.

* + - * 1. Slide-over drawer

The decision was made to add a wine background and the user’s name (once signed in) to the top of the app drawer. The drawer was too plain and I figured a personalized approach is always something that the user loves.

I added a close button to the lost of items in the drawer because I had not yet figured out a way to add a close button to the top right, but it’s a great idea to have it at the bottom in the list, its functional, noticeable, and it does not blend in with the rest of the items, due to the divider.

* + - * 1. Search Page

The decision was made to implement a tab bar to first choose food or wine, then just have a search bar underneath (pending implementation), so that it would be more streamlined for the user and additionally, it looks more sophisticated.

* + - 1. Coding
         1. Approach: Object Oriented
         2. Dart and Flutter work in unison to create the application and its functionality. However, the majority of the front end coding is done in Flutter. The line between Flutter and Dart is not very apparent, but flutter handles UI, and Dart handles functionality.
      2. User training
         1. Training and guidance will be provided under the help page, which is located in the slide-over drawer
      3. Testing
         1. Testing Style

Testing was completed running device emulators so that the Pairings team could see what the application would look like to the user, on different devices and operating systems. All functions and elements were tested in a semi-random format, so that it would be realistic to how a user would use the application.

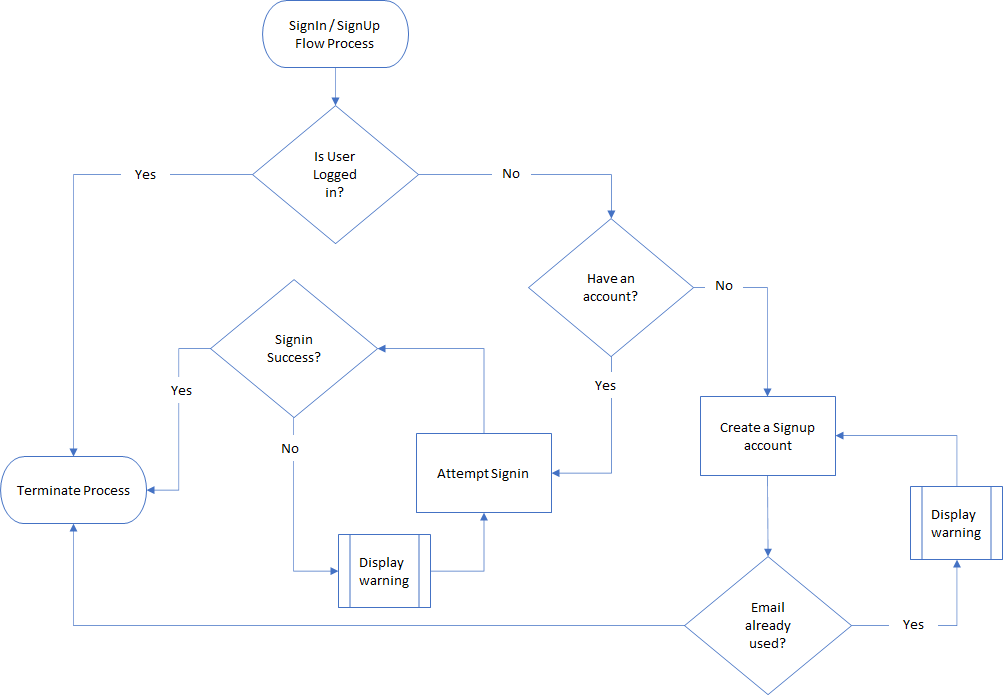
* + - * 1. Devices using emulators

iPhone 13 Pro Max

Google Pixel 5

* 1. **Front End Maintenance Screens, Controllers, and More - Scott King**
     1. Maintenance Screens

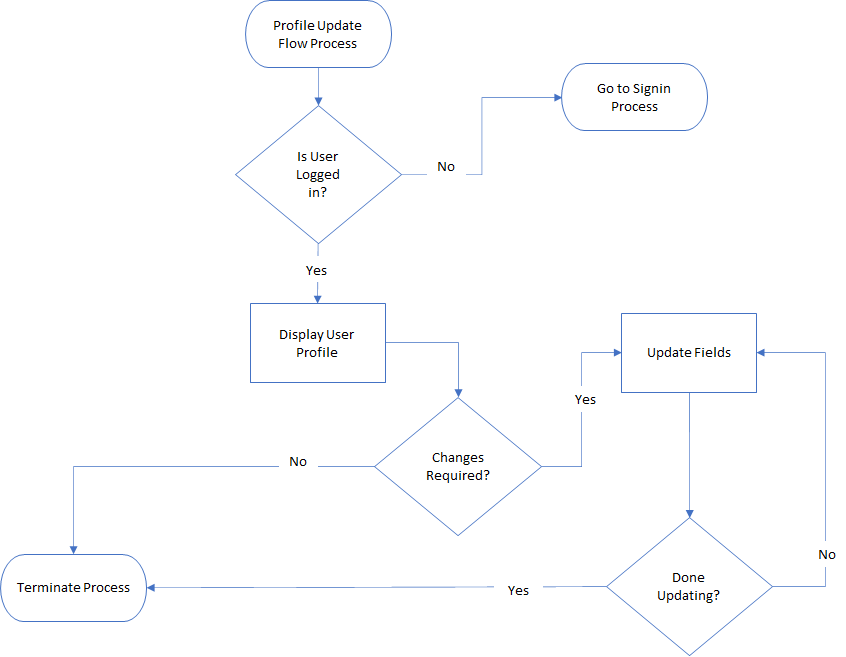
Maintenance screens are extremely important to the UI and the overall user experience with the application. Initial design and subsystem flow was derived based on expected user experience. The user is typically looking for a clean, uncluttered interface. The user experience must include access control to ensure that the particular user session is unique to them. Furthermore, data retention is required to provide the user with a means to recall previous interests. These requirements and expectations specify that the application has a robust set of maintenance screens. The account sign up (new user without an existing account) and sign in (returning user that already has an existing account) was designed around the following process flow:



Once the decision logic was determined, the next step was to design the user interface that would collect the pertinent information and ultimately forward to the controller and database for authentication. The following design elements were identified for the the sign up and signin screens:

* Sign up (only) - New user account creation and initial profile data collection.
  + Base visual widgets required:
    - Header Text
    - Email address text form box
    - Password text form box
    - Password confirmation text form box
    - First name
    - Last name
    - Phone number
    - Birthdate
    - Create account button
  + Confirmation of account creation:
    - Snackbar confirmation at bottom of screen
* Sign in (with sign up option) - Return user view to save and retrieve user specific information
  + Base visual widgets required:
    - Header Text
    - Email address text form box
    - Password text form box
    - Sign in button
    - Create new account hypertext option
    - Forgot password hypertext option
  + Confirmation of account sign in
    - Snackbar confirmation at bottom of screen

Supplemental screens are also required as a byproduct of the sign in and sign up process. A user also needs the ability to to change or reset password as well as the ability to update profile information. Profile management logic is dictated by the following flowchart:

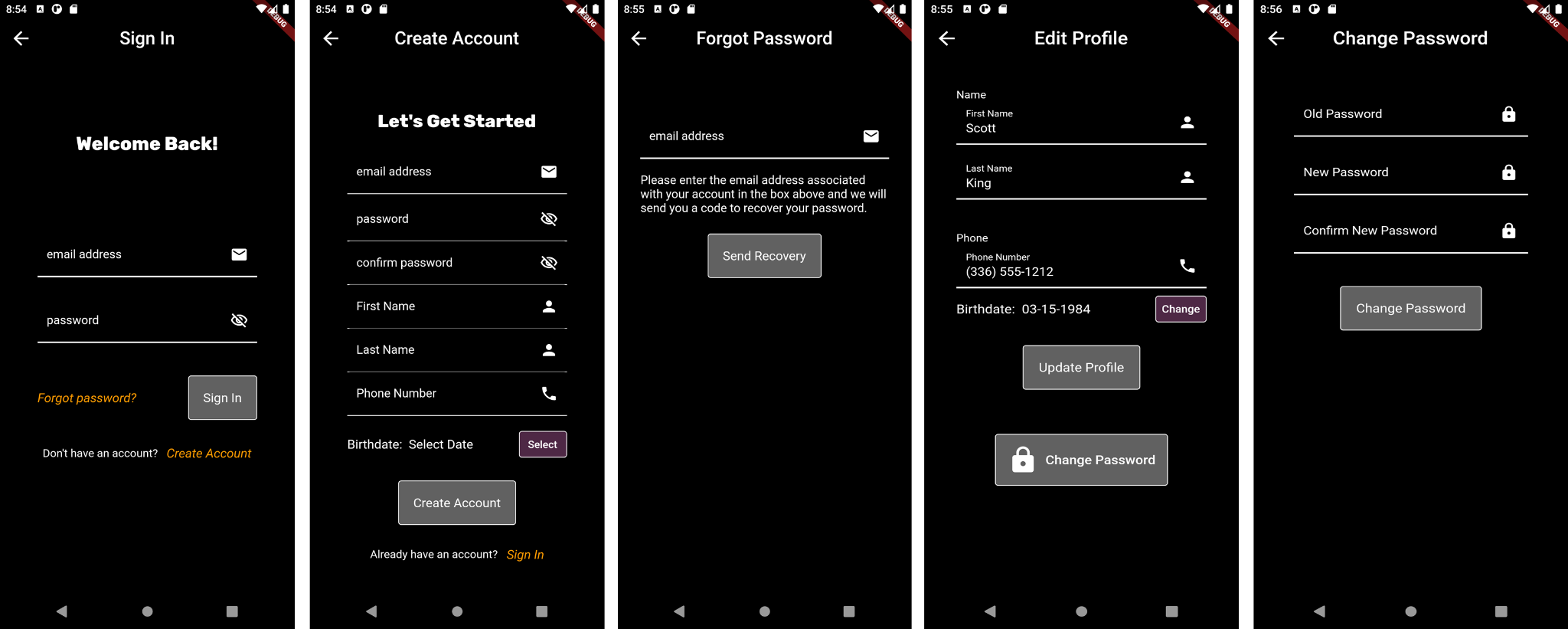


Next the design elements for these screens were determined by identifying the major visual and interface elements required.

* View/Edit Profile - User can view and change edit profile fields
  + Populate screen with current user profile information:
    - First name text box
    - Last name text box
    - Phone number text box
    - Birthdate text
    - Birthdate change button
    - Update profile button
    - Change password button
  + Alert dialog response confirming change or error message
* Change Password - User can change a known password
  + Base widgets required:
    - Current password text box
    - New password text box
    - Confirm password text box
    - Save changes button
  + Alert dialog response confirming change or error message
* Reset Password - User can change an unknown password
  + Base widgets required:
    - Email address text box
    - Information text message
    - Send recovery text button
  + Alert dialog response confirming next step in reset process

The initial wireframes were based on concept discussions that included heavy fonts, sharp edges, and rectangular elements. However, later refinements included softening some of the edges and the use of lighter line weights and fonts for the screens.

The final screen design for the maintenance screens are shown below:



After the design of the visual elements on each of the screens was completed, the next step was to develop the code. This is where the decision to code using flutter was extremely helpful. The core principles of flutter coding are:

* Everythings a widget
* Declare a property then define a widget that satisfies the property
* The widget child/parent relationship

The child/parent widget relationship means that the ancestry hierarchy can be used to percolate notifications up the widget tree. This is especially useful in passive notifications on the current view to update the user session. However, more complex notifications were required for updating future and non-active views. These notifications were accomplished through the use of global variables. Form keys and global variables allowed updates to current and future pages without having to actively passed these variables as arguments to each view and controller.

For fields where user input was required, controllers and validators were used to ensure that the input met the correct format before passing the inputted values to the controllers. This pre-checking validation method prevented the unnecessary passing of corrupt values to the controllers and database. It also reduced the amount of error handling programming required for the controllers and database.

For complex fields, such as email, birthdate, and phone number, simple controllers could not capture all of the possible acceptable responses. The solution was to use regex patterns and a utility function to verify the inputted data met the appropriate format. Verifying and controlling the program direction while still on view screens made it easier to provide immediate feedback on erroneous values while reducing backend controller calls.

* + 1. Controllers

Although not part of my initially assigned responsibilities, I contributed to the development of the controllers for the maintenance screens. These controllers were through multiple iterations. Initially, the signin and signup controllers were designed to return a user object. However, with the final refinement, they were changed to update the global variables and then merely return a boolean value indicating whether the controller was successful. Similarly, the profile and password controllers only returned a boolean value pertaining to the successful update status because the user data was already available in the active user session. The only additional step required by the controllers was to update the global user object with the temporary object that held the values until the database confirmation was received.

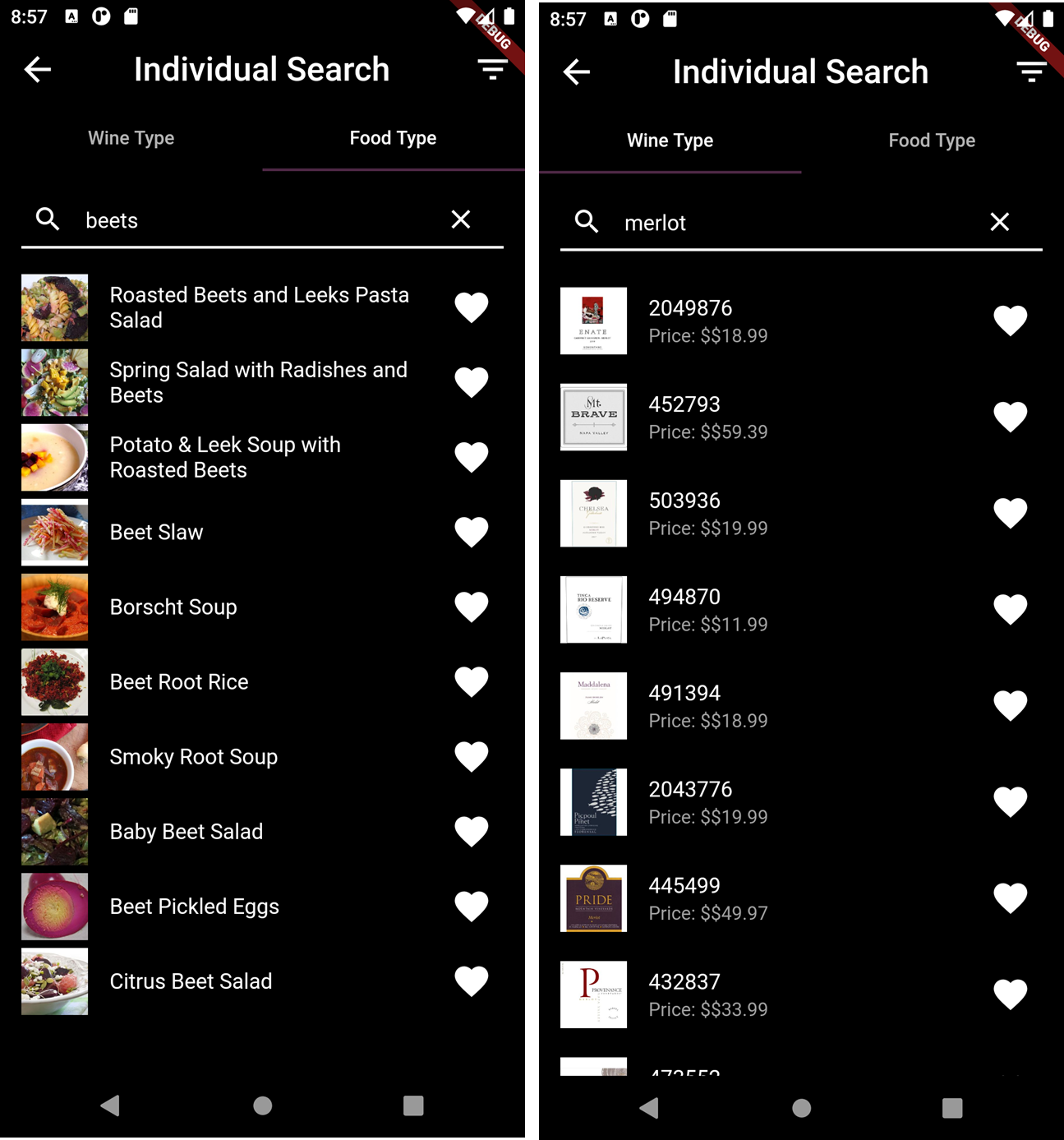
* + 1. Individual Search and Spoonacular API

Although not part of my initially assigned responsibilities, I completed the individual search view page. The individual search is a core functionality of the application. The design criteria included allowing the user to input a text string associated with a food or wine type and then querying the Spoonacular API for matching response data. This data was presented in the form of a json, which was mapped to the appropriate model type and then passed back to the view for presentation to the user. The interactive list also allowed the user to select any items and apply them to a favorites list for future recall.

* Individual Search - Interactive user interface query of wine and food searches
  + Base visual widgets required:
    - Header Text
    - Tab bar for separate wine and food searches
    - search text form box
    - Listview response
    - Favorites icon

The food quarry required significant change from the initial coding. Initially, the API query returned a FoodSearchModel object. This object was far too complicated with multiple layers of objects nested within objects and the majority of the information wasn’t useful to our application. As a fix, I edited the API to include a recipe search function call that returned a more applicable json response. This also necessitated that I create a recipe result model and food model to handle new API call json response.

The final design screens are as shown below:



* 1. Backend (Middleware) - Sogol Taheri

7.4.1 Responsibility - application functionality and making the connection between frontend and backend.

7.4.2 Refinement

7.4.2.1 at first I coded for application functionality such as validation email, validation password, authentication, etc.

7.4.2.2 After that I tried to help and learned about the API.

7.4.2.3 Then I got the user model done.

7.4.2.4 and now I’m coding in controllers in order to connect our frontend to the backend and my main focus is to get the signin controller to work.

7.4.3 Coding - we use dart (flutter) for this project and I write the required code in vs code for my part.

7.4.4 User Training - pending

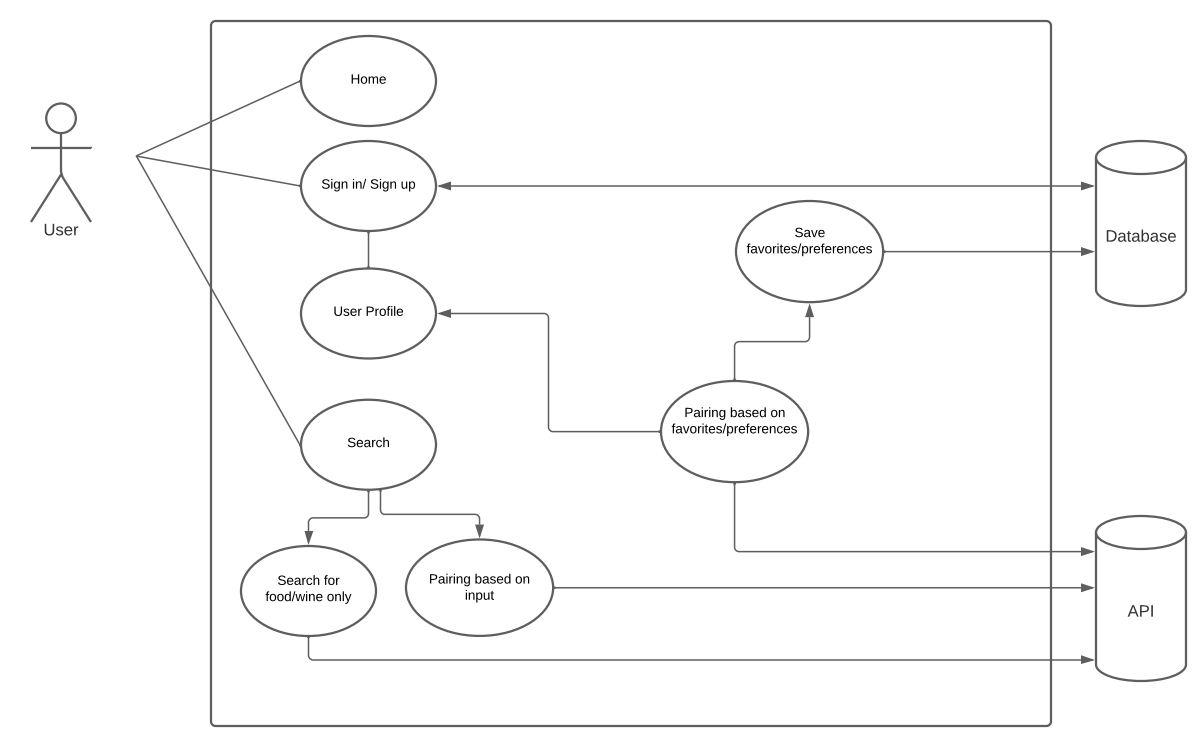
7.4.5 Testing - at the moment I’m not able to test my code in my own PC since I get errors for some reason in all files but after I complete the signin controller and fix those errors (in all files) I should be able to test them too in the application flow.

* 1. API - Suqoya Rhodes

Responsibilities

* Wine Search/Recommendation
* Food Search
* Wine Pairing
* Dish Pairing

7.5.1 Initial Design

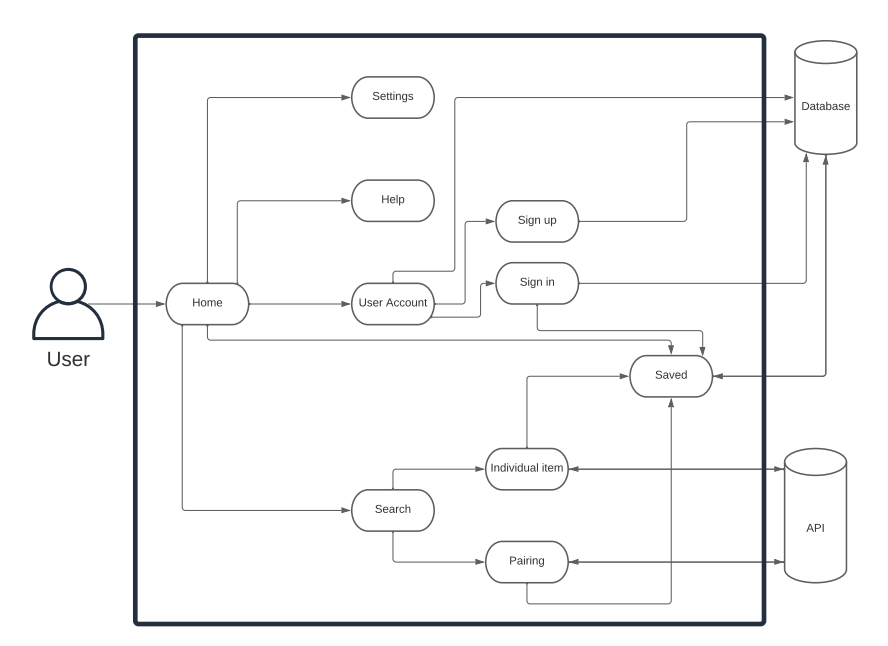


7.5.2 Refinement

7.5.2.1 The initial design for the use case diagram was very early on and before I had a clear idea of what was needed from the API (Spoonacular) and how it would be implemented. It took some time to have a clear picture of what the final product would look like with some back and forth between the team to come up with the final design.

7.5.2.2 Changes made from the initial design to the final design in regards to the api are that the user can search for an individual item or a pairing. This information will be coming from the api as initially planned of course. One big difference that was needed was saving the information and how that would need to be done so that the user can be provided with better recommendations and there is a database that keeps track of what they liked most. Also the final design has a much cleaner look and is much easier to follow.

7.5.2.3



7.5.3 Coding

7.5.3.1 Approach- The initial approach was to get the code done that would return wine pairings and recommendations. Once that was complete I moved to dish pairings. Upon first glance it seemed simple enough to be able to search the foods, but there was no clear path to only being able to search foods without returning a lot more information than we needed. Searching foods doesn’t just provide one item with an id and other attached information. It returns various id’s for various things the api provides like videos of that food and where to buy it from etc. The issue with that was that there could be multiple ids that go along with that one search. So I had to narrow down what was the most important and useful information for the data we needed. Once that issue was clear my next task was to figure out how the user favorites needed to be stored so that they could be easily referenced. Going from food to wine recommendation we can store the (food\_id, recommended\_wine\_id).

7.5.3.2 Language- The language used was Dart/Flutter and the environment was VS Code

7.5.4 User Training- pending

7.5.5 Testing

7.5.5.1 Postman used for API (Spoonacular) testing