Mark Vong

ITP 368

[mvong@usc.edu](mailto:mvong@usc.edu)

**Final GUI Project Design Document: Charity Finder**

**Purpose of the application:**

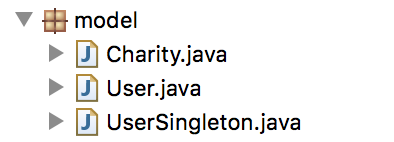
The purpose of the application is to create the opportunity for users to locate and filter nearby charities to donate items to. Individuals will want to use the application because it will facilitate the access of charity information as well as allow these users to store their favorite charities for later access. The application will reveal information about charities including their name, mission, foundation status, category, location and in order to aid the user in picking the appropriate charity to donate to.

**Design Patterns:**

The Charity Finder application adheres to the MVC design pattern. The controllers are responsible for updating the views if there are any changes to the model. They are also responsible for updating the data on the database should a user choose to update a model. The views are responsible for displaying the model attributes and the models are responsible for encapsulating data and information. A singleton was used for the User class to prevent the passing of User instances between views.

**Models:**

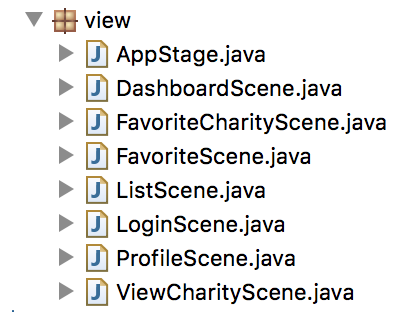
In order to model the data, a **Charity** and **User** class is used to encapsulate the information pertaining to the charities and users. Instead of a database singleton, I incorporated REST API calls through Java’s http client: **HttpsURLConnection** to retrieve data from my Firebase instance. In addition, the same client was used to get data from the Charity Navigator REST API. Data structures include **Arrays**, **ArrayLists** and **HashMaps**. In order to retrieve the active user’s information between the views, I created a **UserSingleton** to avoid passing the user instance between views. This allowed for cleaner code as well as further abstracting views and the user model.



***Figure 1.*** *Model package in Charity Finder: Charity, User, UserSingleton.*

**Views:**

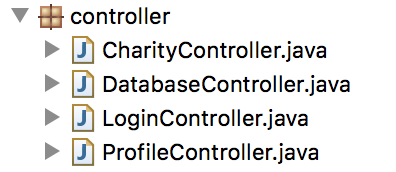
In order to create a more cohesive application, the **Items scene** was removed because it did not serve a purpose as the Charity Navigator’s REST API did not provide a functionality for direct donations to the given charity. In addition, the **Map scene** was converted to a **Charity List scene** due to the lack of resources to implement a map view in JavaFX. The **Charity List scene** allows users to search for charities based on a given state, selected through a combobox of state abbreviations. Also, a **Charity Information scene** was added to allow users to see more information on a given charity they double click on once on the **Charity List scene**. In addition, the **“Add Favorite” button** was removed from the **Favorite Charity scene** and moved to the **Charity Information scene** for better design and data control. Additional text fields and a checkbox were added to the **Login scene** to account for new user creation.



***Figure 2.*** *View package in order: Main stage, Dashboard scene, Favorite Charity Info scene, Favorite List scene, List of Charities scene, Login scene, User profile scene, View generic charity info scene.*

**Controllers:**

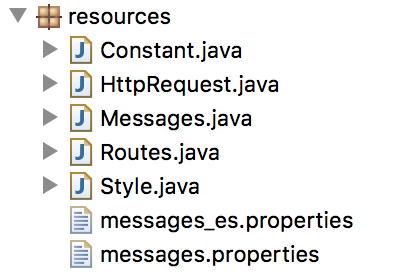
In order to maintain data flow between the view and the model, multiple controllers were used. In order to retrieve data from the Firebase database, **DatabaseController** was used. The controller converted the user data into a JSON object to be sent to the Firebase as well as converting the JSON object back to Strings to read into the models. The **LoginController** maintained user authentication in conjunction with the **DatabaseController**. The **CharityController** was used to convert the Charity information to JSON objects to be stored in the Firebase database. It was also used to make GET requests to the Charity Navigator’s REST API. In addition, the **ProfileController** was implemented to communicate the user profile information changes to the views as well as to the database. Each controller was responsible for their own respective views that contained controls capable of updating the user’s information as well as the user’s favorite charities. They worked in conjunction with the **DatabaseController** to parse the JSON data appropriately for Firebase’s REST API calls.

****

***Figure 3.*** *Controller package in order: Charity controller, Database controller, Login controller, Profile controller.*

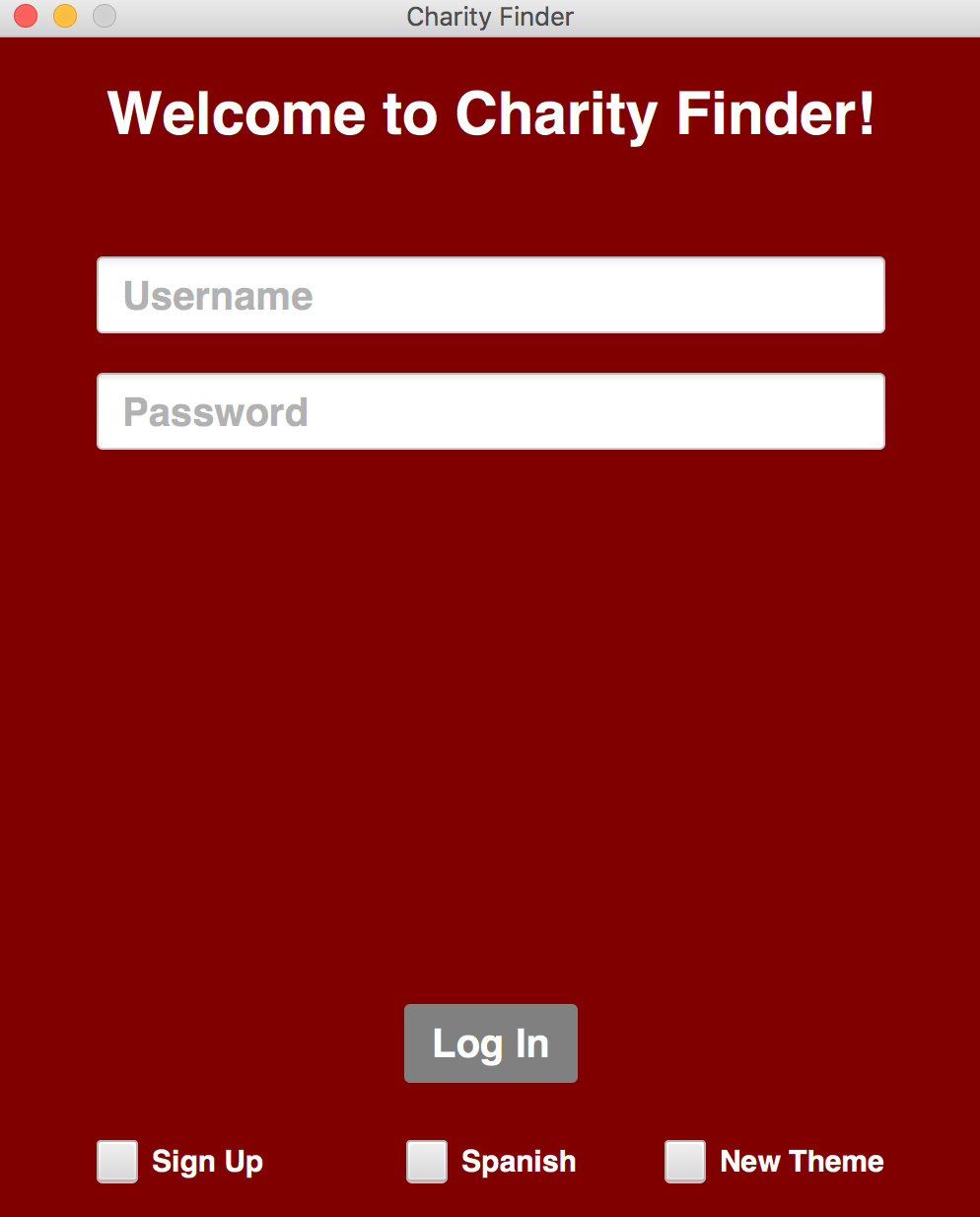
**Resources:**

Finally, I created a resources package to abstract the styling and strings from the UI views/controls. The **Constant** class housed the string resources used in internationalization/localization as well as throughout the application’s scenes. The **Messages** class hosted the resourcebundle and updated it according to the user’s volition, calling either **messages\_es.properties** for Spanish or **messages.properties** for English. **HttpRequest** housed the main API call procedures: GET, PATCH, POST, PUT, DELETE for both the Charity Navigator and Firebase REST API’s. The **Routes** class contains the necessary routes for the API calls. **Style** contains the control and view styles as well as various fonts and paints.

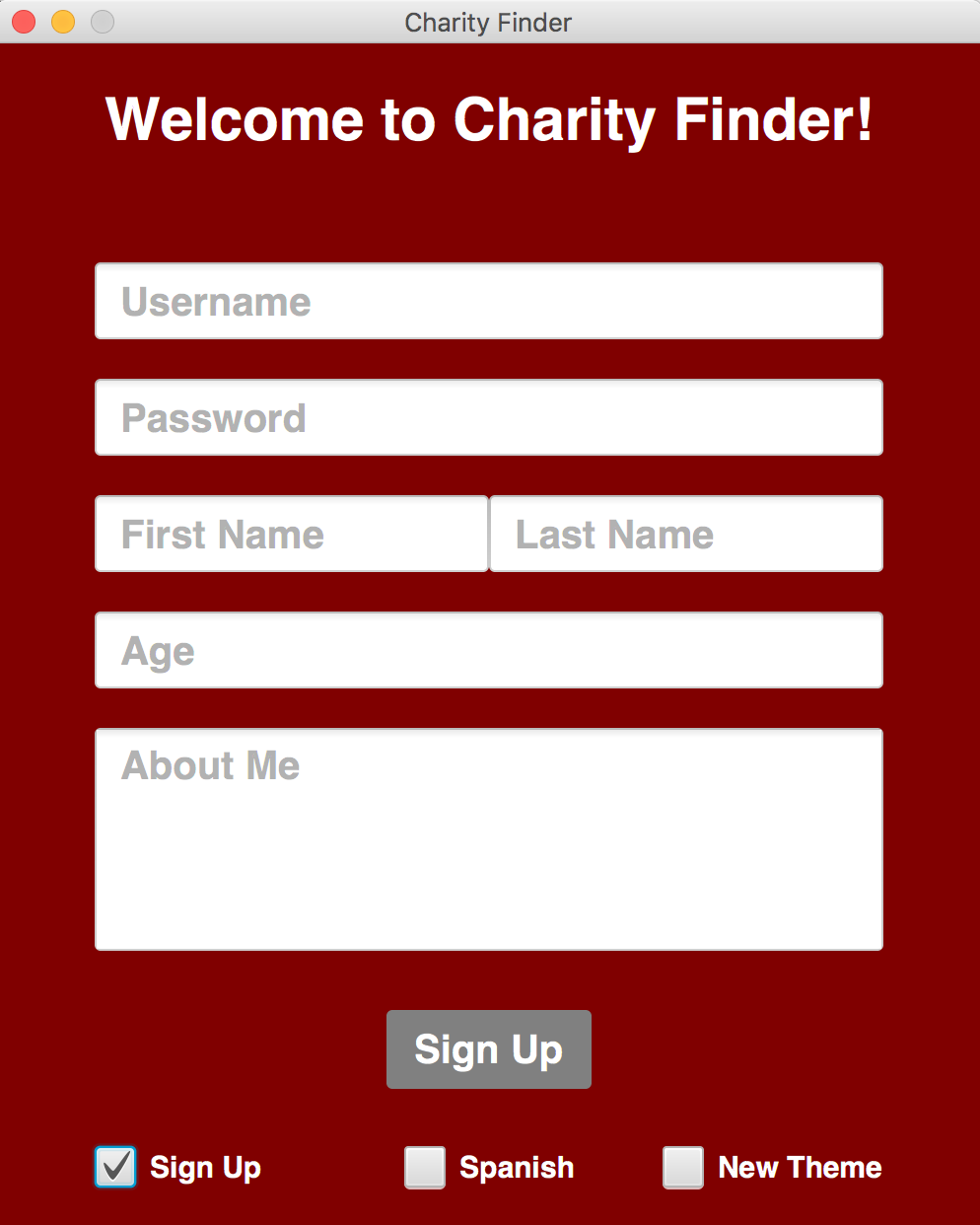


***Figure 4.*** *Resources package in order: Constant, HttpRequest, Messages, Routes, Style, messages\_es.properties, messages.properties.*

**Program Flow & Applications Screenshots:**

****

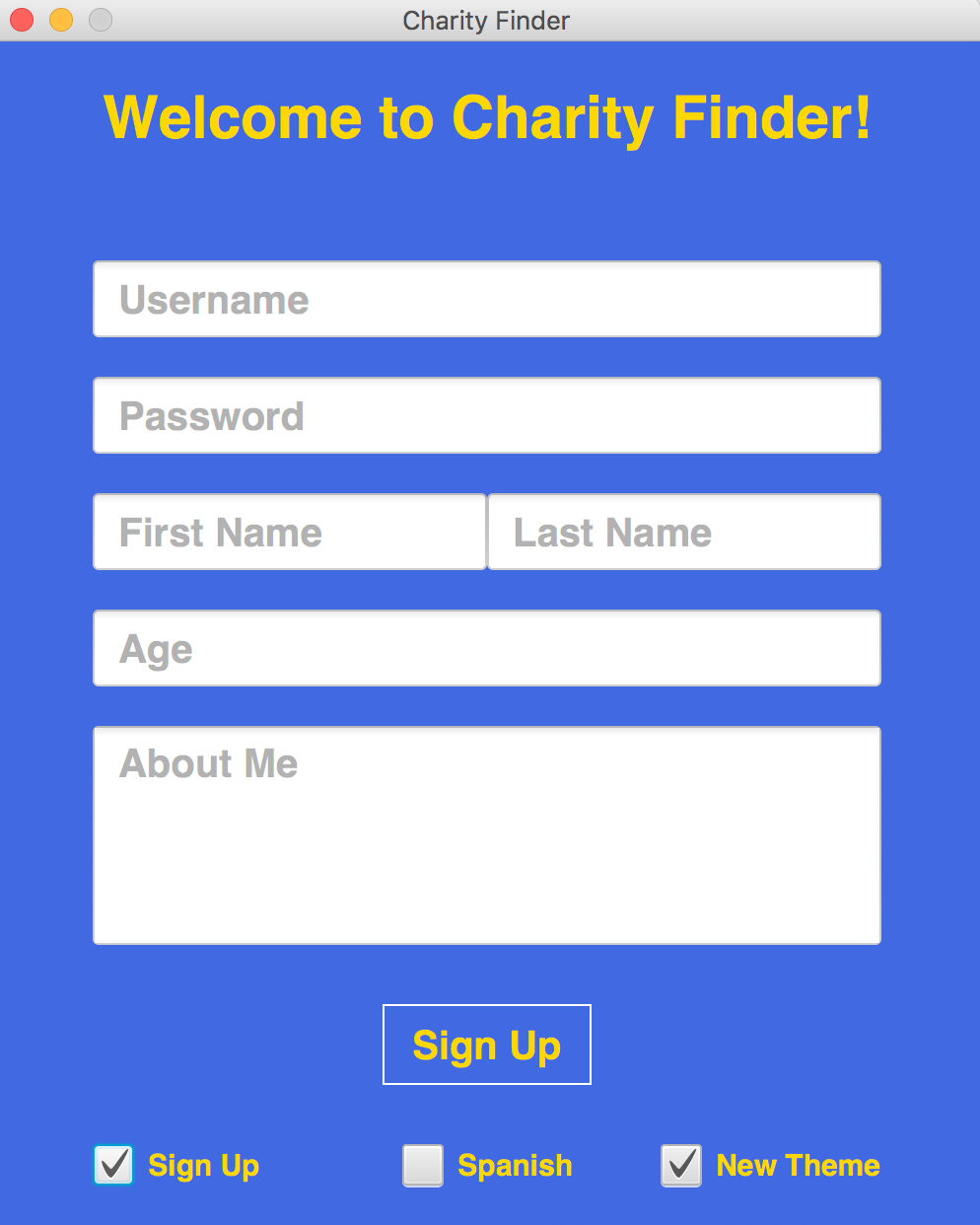
***Figure 5. Login Scene***

****

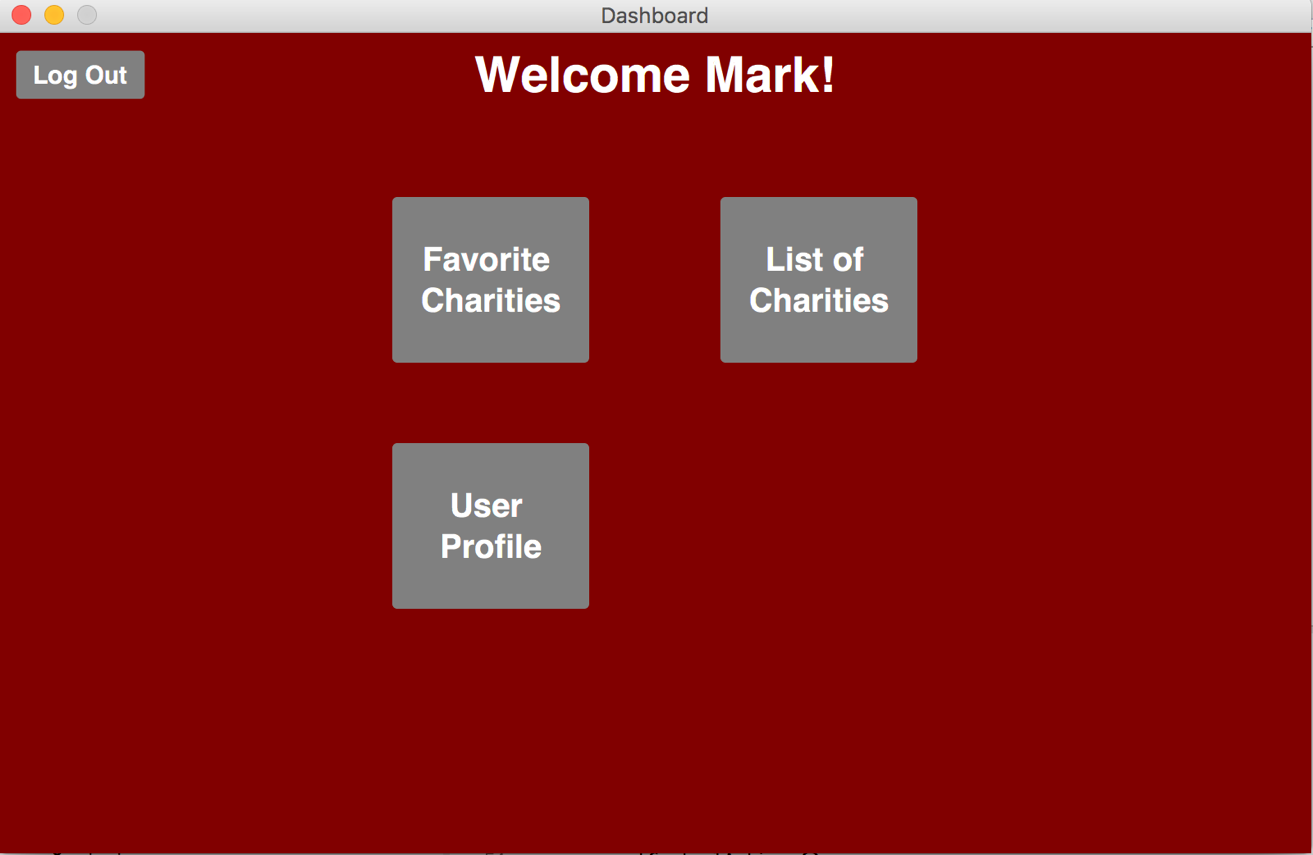
***Figure 6. Login Scene: Sign Up checked***

****

***Figure 7. Login Scene: Sign Up and Spanish Checked***

****

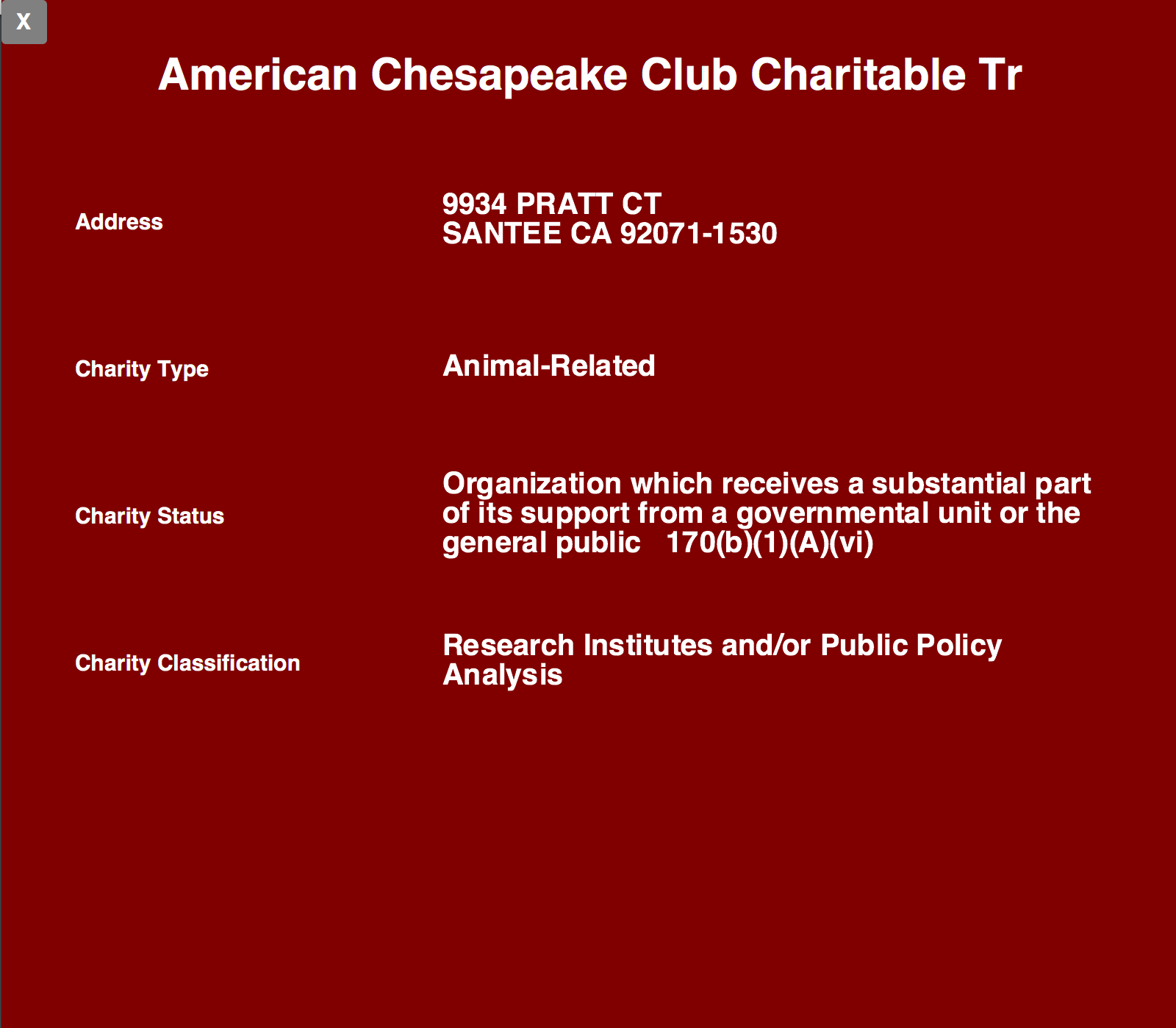
***Figure 8. Login Scene: Sign Up and New Theme checked***

****

***Figure 9. User dashboard scene. Displays options for Favorite Charities, List of Charities, and User profile.***

****

***Figure 10. User’s list of favorite charities scene. Can double click on each charity to view information from a pop up window.***

****

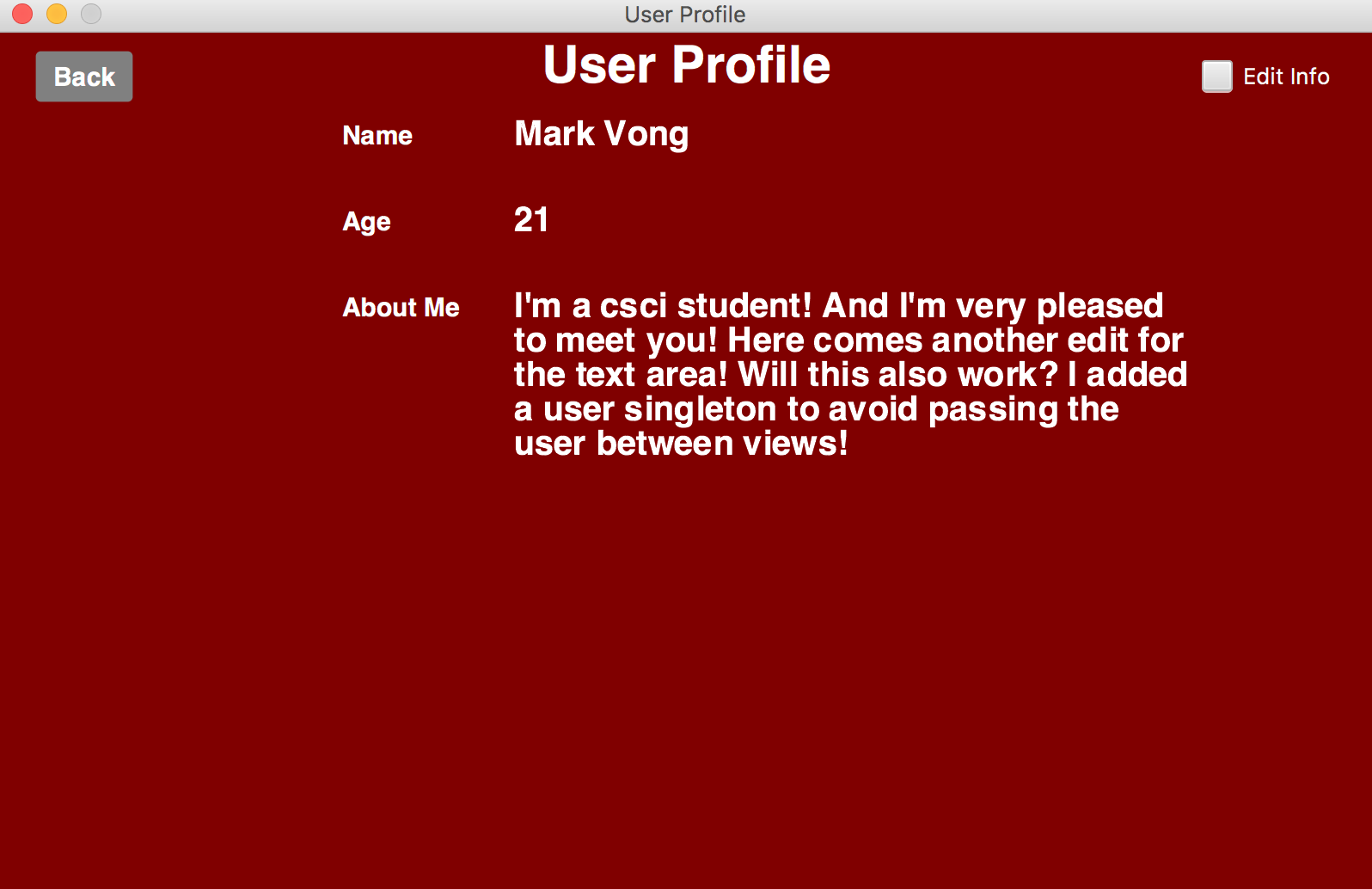
***Figure 11. When a user double clicks a charity from the list of favorite charities, they are able to view the selected charity’s information: address, type, status, and classification.***

****

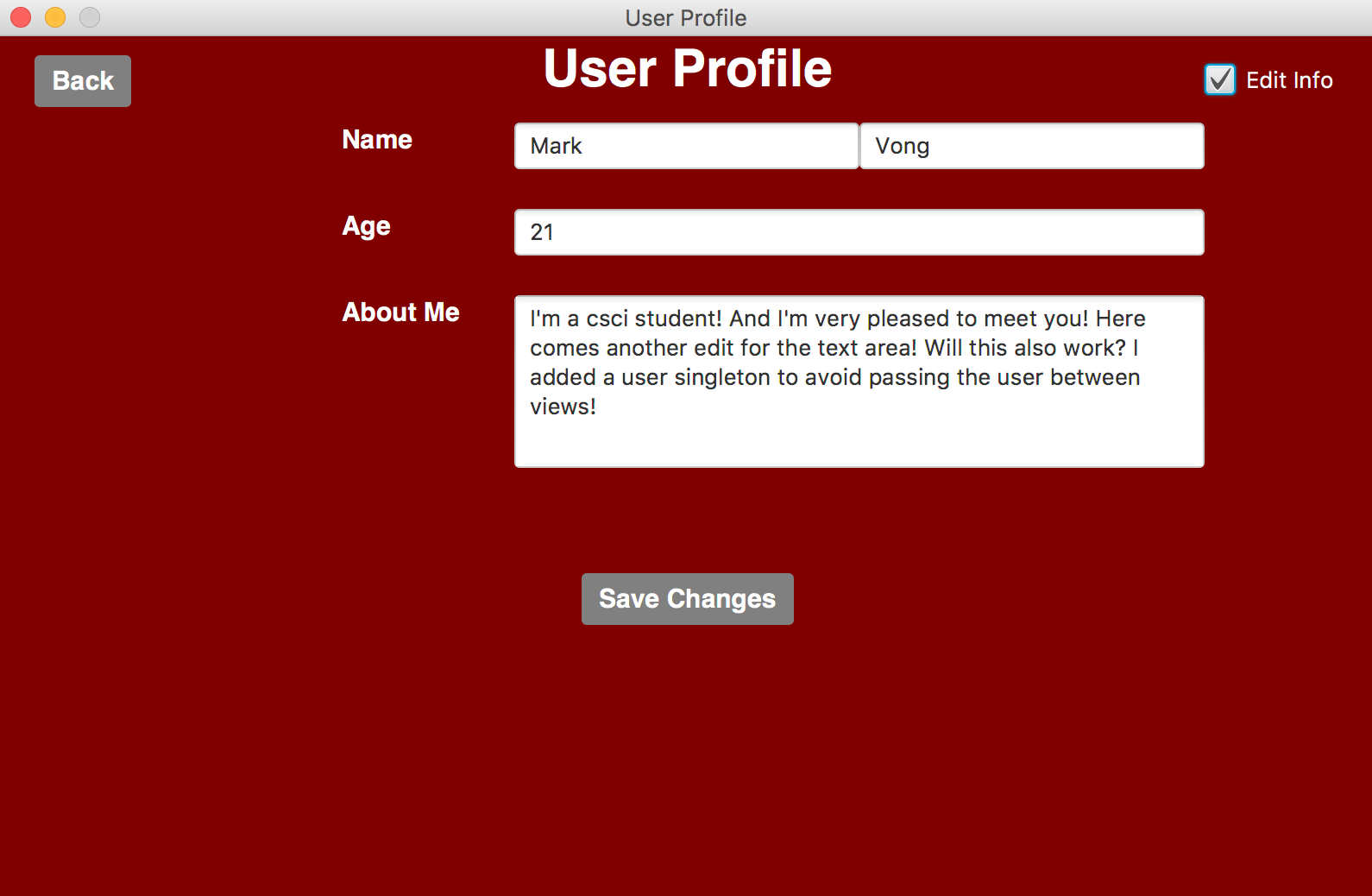
***Figure 12. The list of charities scene. User may search by state selected through the combo box. User may double click a charity to learn more about it and favorite it.***

****

***Figure 13. The information pop up view scene of a clicked charity from the list of charities scene. Similar to Figure 11, however, users may favorite this charity and store it in their list of favorites.***

****

***Figure 14. User’s profile scene. Can view the user’s name, age and about me information. User may also edit their information by selecting the check box in the upper right.***

****

***Figure 15. User’s profile scene: edit info check box selected. User may edit the information and the changes will be pushed locally as well as to the database.***

**Extra Credit:**

I used Firebase’s REST API for data persistence and Charity Navigator’s REST API for data retrieval of each charity. The integration of the API calls was the most difficult part of the assignment and I believe this could be a project that could be easily transferred to a mobile application.