

Voluntold Design

Underlying Platforms and Frameworks

Ionic and Cordova

I wanted Voluntold to be available across all devices, and I felt that programming the app more than 3 times would be out of the scope of my CS50 Final Project, and its allotted time. For this reason, I looked into Hybrid App Development: an application running in the mobile browser of a phone or tablet.

The problem of this is that running apps in the browser are slow, have latency issues and offer no options to leverage native APIs, such as emailing or camera.

The only framework that stands above the rest in terms of capability and usability is Ionic Framework, developed by Drifty Co. Ionic is built on top of Cordova, which is able to add native capabilities to apps, as well as provide the ability to build and run the apps natively on mobile platforms. In addition, Ionic allows for development in JavaScript, HTML and CSS, so the learning curve wasn't as big.

Nevertheless, the learning curve was still big. Ionic relies on AngularJS to power its interactions: a performance-obsessed framework which allows for the creation single-paged web-apps. Combined with Cordova, AngularJS and Ionic's front-end components creates a multi-platform, yet native feeling application.

A large part of my time was spent learning and getting comfortable with AngularJS and its capabilities, but Ionic offered great documentation (I even recommend it for future years of CS50!) so it's learning process was strait forward.

Firebase

I wanted Voluntold to be extremely scalable, and this required a database/backend that was fast, readily available, and most importantly: reactive. Features I intend on implementing in future iterations of the app include instant messaging, where reactivity is crucial.

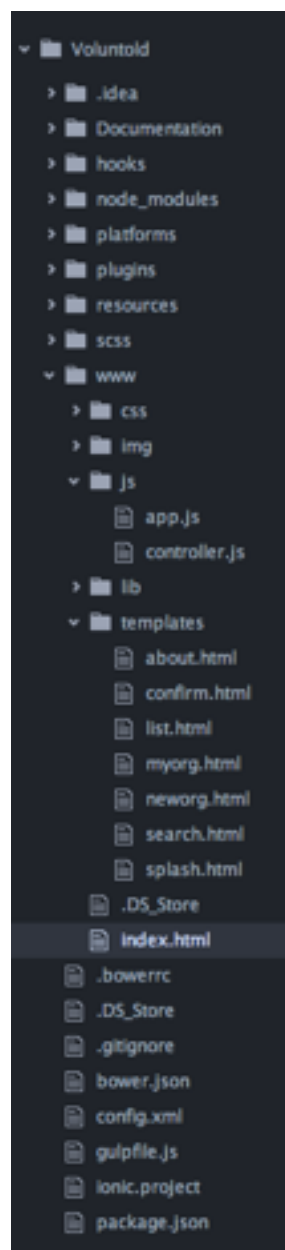
Due to SQL's limitations with native applications, and its lack of reactivity/scalability, I opted for a MBaaS (Mobile Service as a Backend). After toying with MBaaS's such as Parse (backed by Facebook), I decided on Firebase (backed by Google).

Firebase is extremely useful since it allowed for me to use the same code to implement the database across all platforms, natively. It also offers large capability for free, with affordable pricing. It took less than a day to fully learn, since Firebase allows developers to simply interact with standard JSON objects, instead of using proprietary formats.

Implementation

Structure

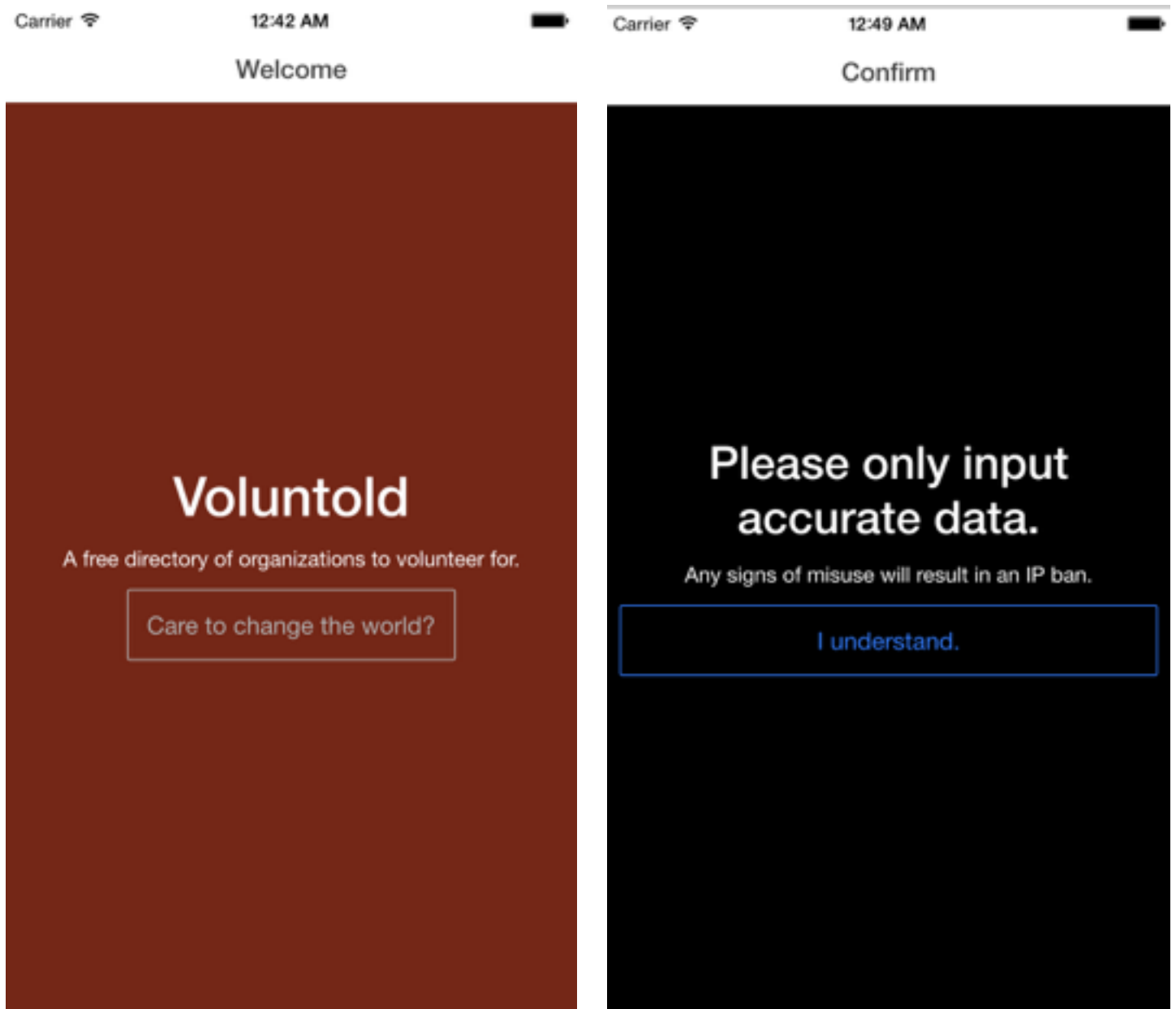
Voluntold loosely follows MVC, since I wanted the benefits of MVC, while still obliing the Ionic's standards. The Model is Firebase, which is available in the cloud, and is an array of JSON objects, containing the various organizations. The views of Voluntold are separate HTML files which are all linked using the `<ion-nav-view>` component, that are available in the `www/templates` folder. Finally, the controller are the various controllers for each view, defined `www/js/app.js`, and are available in `www/js/controller.js`.



Splash & Permissions page

The Splash and Permissions page of Voluntold are rather simply implemented. The screens were structured using Ionic's HTML components. Using a custom CSS class, I centred the text and the button, and set the background to various colours.

Pressing the button in the centre of the screen loads the next screen. This is done using AngularJS's `$state` component, which allows the web-app to transition between views and their respective controllers. Pressing the button would transition the state using the `$state.go('state')` function.

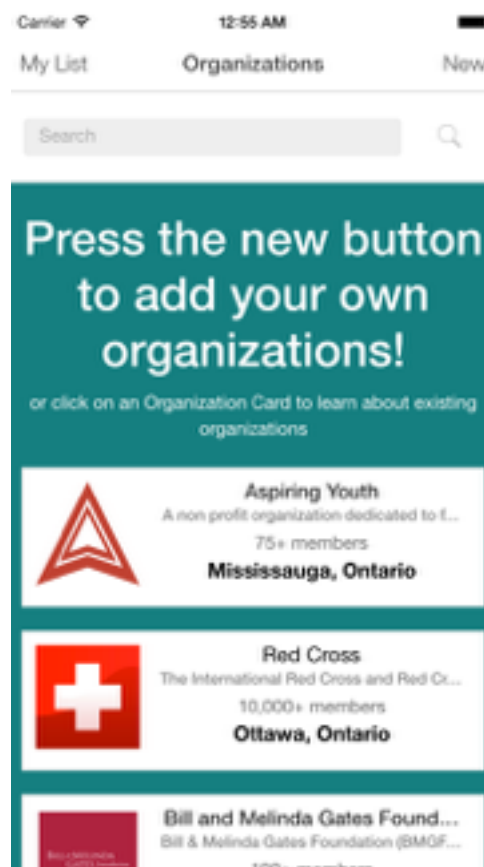


Organizations page

The Organizations page is the heart of the app. Once again, it is structured using Ionic Framework's components in HTML, using custom CSS classes to style its appearing, margins, colour etc. Helper text is also set to NULL after user visits a different page, or interacts with the view in any way.

The view uses AngularJS's *foreach* attribute to generate the cards in the list. The array used is the array of JSON objects from the Firebase database backend. Each JSON object contains information about the organization, such as its name, location, size, description, avatar, image and contact info. When any of the Organization cards are tapped, it changes the state to the About page, and passes the JSON object of the respective organization with it, in order to display the information. Also, a search bar is available, which, using AngularJS's data binding abilities, saves a query for the database.

Since the app should have offline capabilities as well, I used Firebase to create a local array, and did searches and manipulations on the local array instead of directly using Firebase. The query is then ran through a *for loop*, iterating through all the organizations to see if any organizations' names match, and then passes the organization's JSON object to a global array, so the search results can be accessed from a different view: the Search/Results page.



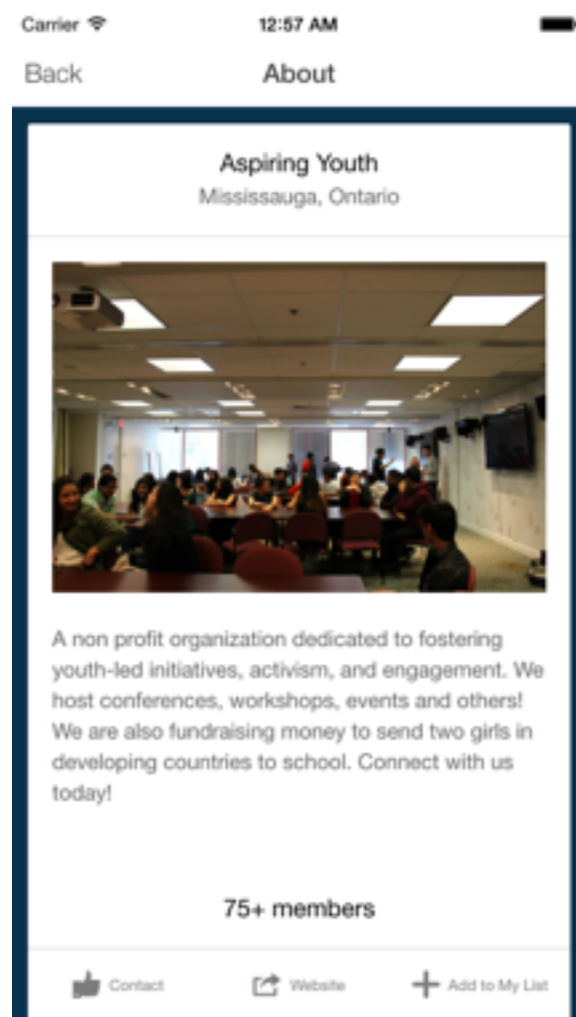
About Page

Pressing on an organization in either the My List page, or the Organizations page will open the organization's About page (each organization has one). Like every other view, it is structured using HTML, and uses custom CSS classes to alter its appearance. The back button would transition it's state to its previous state.

While the app transitions from either the Organizations page, the My List page, or the Search/ Results page to the About page, the app passes on the JSON object of the organization, containing all the necessary information for the view. In order to minimize storage size on the Firebase database, the app will dynamically generate the view by auto-populating the view with information passed from the JSON object of the organization.

The bottom of the screen leverages native API calls in order to open the device's web browser and email client, using buttons.

In addition, the user can add the organization to his/her list of organizations to apply for, using the "Add to My List" button. This pushes the JSON object of the organization to a global array of organizations to form a list, so it can be accessible across views.

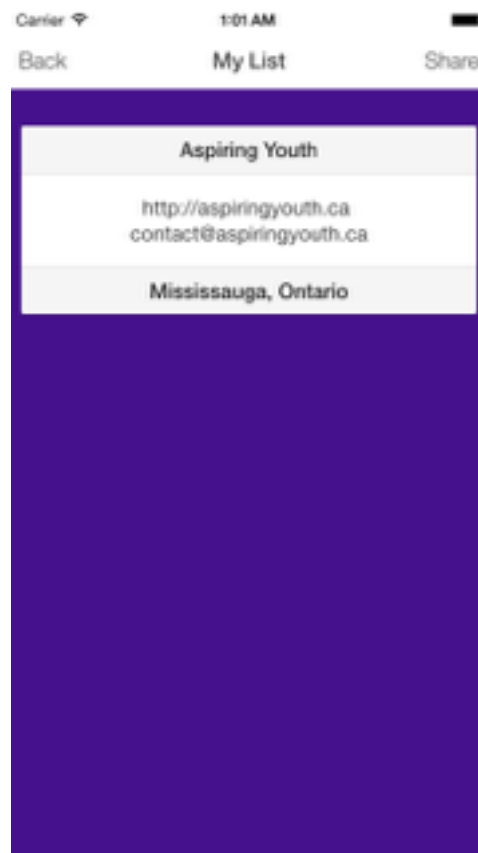


My List page

The hardest portion of the app to implement was the My List view, since it required the usage of 3rd party libraries on top of the app framework in order to implement a sharing feature, which is extremely restrictive by nature, since it can only be demoed on devices that are running the app natively and have an email client. The view is structured using Ionic's HTML components, and custom CSS classes, but also required an Apache Cordova plugin in order to implement the sharing feature. This view can be accessed by pressing the "My List" button in the Organizations page, or automatically after adding an organization to "My List". The back button would transition it's state to its previous state.

The List page didn't have to implement its own List array, since it would be generated whenever the user pressed the "Add to My List" button, in any organization's About page. Using AngularJS's *foreach* attribute, it would generate the appropriate cards, using the array of JSON objects to populate its information. Pressing the card would transition to the About page of the organization, by passing its respective JSON object used to generate the card.

The sharing feature was tricky to implement since the body of the email would have to be automatically generated, depending on the array of "My List". A standard "mailto:" href attribute wouldn't work in this scenario. Using an Apache Cordova plugin named "Email Composer" (<https://github.com/jcjee/email-composer.git>), I had the option of rigidly structuring the email using JSON, and the body of the email using HTML. I used a *for loop* to add to the email's body string the name of the organizations, their locations and their contact information (all formatted using h1, h2 and h3) in a conveniently organized structure.

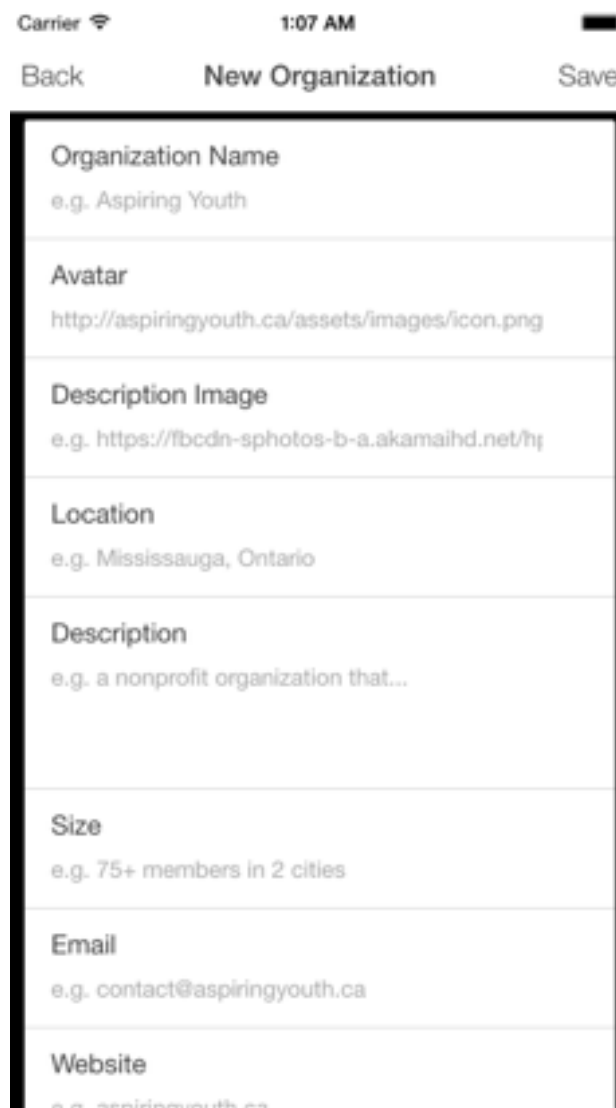


New Organization page

New organizations can be added to the directory through the New Organization page, which can be accessed by pressing the New button in the header of the Organizations page. The back button transitions to the Organizations page. This view was structure using Ionic's HTML components.

This view is a form, which requires all crucial information about an organization to be viewed, which is then stored in a JSON object and pushed directly to the Firebase array. I chose to use image addresses for images instead of uploading in order to save storage space in the database, and for the app to run faster, since the main functionality of the app would be from retrieving information from online, which would then be later cached by creating a local database afterwards for possible offline use.

Pressing the save button would generate a JSON object and push the object to the Firebase array, finally transitioning the state to the Organizations page.



The screenshot shows a mobile application interface for adding a new organization. At the top, the status bar displays 'Carrier', signal strength, '1:07 AM', and battery level. Below the status bar is a header with three buttons: 'Back', 'New Organization' (highlighted), and 'Save'. The main content area is a form with the following fields:

- Organization Name**: e.g. Aspiring Youth
- Avatar**: <http://aspiringyouth.ca/assets/images/icon.png>
- Description Image**: e.g. <https://fbcdn-sphotos-b-a.akamaihd.net/hj>
- Location**: e.g. Mississauga, Ontario
- Description**: e.g. a nonprofit organization that...
- Size**: e.g. 75+ members in 2 cities
- Email**: e.g. contact@aspiringyouth.ca
- Website**: e.g. aspiringyouth.ca

And that's all!