

**Navy Project**

**GovAppStore Prototype**

**Software Design**

**CS337**

**Software Design**

**Prepared By:**

**Brandon Ung**

**Maximiliano Barragan**

**Andrew Aquino**

**Kevin Parton**

**Richard Thome**

**Faculty Advisor: Jiang Guo**

**October 24, 2015**

Table of Contents

* Introduction
  + Audience
  + References
  + Overview
* Design Considerations
  + Objectives
  + Strategies
* Software Design
  + Data-flow Diagram
  + Process Diagram
  + Context Diagram
* Architectural Patterns
* System Architecture
* MVC
* REST
* Full Web Stack
  + Server
    - Cloud Service with strong NodeJS support (AWS, RedHat, Bitnami, etc)
      * MongoDB
      * NodeJS
  + Client
    - Modern Browsers
    - Mobile
* User Interface Design
* Database Schema/Models
* Collections

**1. Introduction**

NAVSEA GovApp Store is a website dedicated to delivering military-developed applications designed for the Navy. With the rapid evolution of computer machinery, GovApp utilizes cloud-based technology that allows for flexible and dynamic updates that meets the demands of the time, and a system where its maintenance is low-cost and efficient.

**Audience**

The intended audience of this application is for the California State University, Los Angeles.

**References**

...

**Overview**

The user will find this application an easy way to search for popular or specifics apps found on the server. Users will be able to create their own accounts dedicated to this website. The account holds personal information pertaining to the user as well as uploads and download history of applications. Users can search via the search bar to query specific items found on the database.

Utilizing front-end libraries such as Polymer, the website’s design creates an easier translation from computer to smartphone displays. Polymer’s flexible html tags allows developers to create their own layouts that can embed other templates on top another.

MongoDB is a NoSQL database that focuses heavily on storage and retrieval of data. This type of database is good to use since GovApp’s primary goal is to gather the appropriate data, retrieve it, write it, and then save it as fast as it can.

**2. Design Consideration**

**Objectives**

GovApp’s three primary goals are…

1. Create a system in which the Navy is able to access the GovApp website, create an account, download apps, and upload apps.
2. Manage a flexible computing system that can dynamically make adjustment and instantly meet the demand changes.
3. Use reliable computing services and decrease maintenance cost by supporting automatic software updates.

Overall, GovApp’s goal is to create a system in which the Navy can easily access and understand the interface of the store. The store must use a front-end library that easy for the user to understand its mechanics, such as registering, logging in, searching for apps, downloading, and etc. If the requirements are not met, then the objectives of the system are not met. The back-end implementation must also function properly in how it handles data-flow and data-write. The system must preserve the integrity of the user’s personal information and history as well. Any damage or unwarranted change to the user’s information occurs, then the objectives are not met.

**Strategies**

The development of the GovApp store will be created in the orderly manner:

Requirements -> Design -> Implementation -> Verification -> Maintenance

This strategy utilizes the waterfall and incremental models of software development. These models allow for easier implementation of the system. The incremental process works in a cycle where GovApp would first increment the index page, then the registration/login, then the search and so on.

UML diagrams will be created to visualize the functions of GovApp. Such diagrams used are Use-Cases, Activity, State, Sequence, and Data Trees. These diagrams display the process and layout of GovApp’s functionality both on the front-end and back-end.

**3. Software Design**

Deployment of the GovApp website.

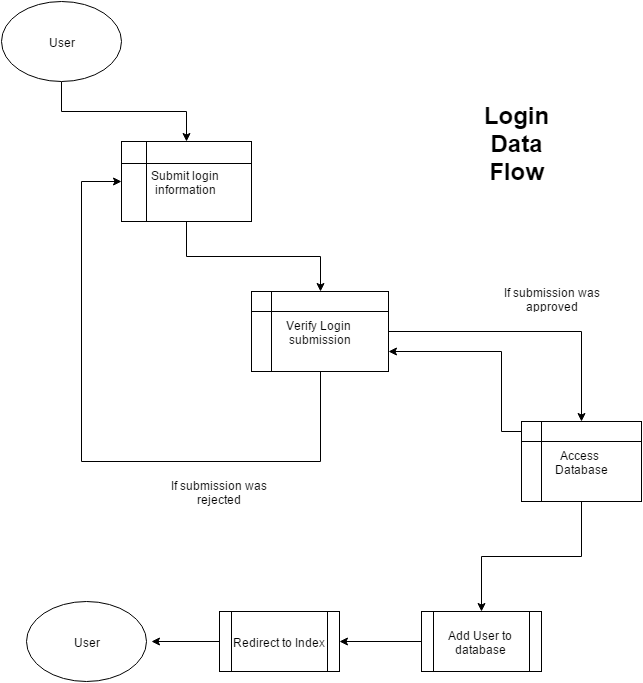


The GovApp store will use the sails.js MVC framework. Sails.js easily creates an instance of a MVC work layout and allows easy implementation of other libraries. Communication between view engine and controller is simple.

HTML5 and Polymer will act as the interface for displaying information while Jquery will be the glue that connects the front-end to the back-end.

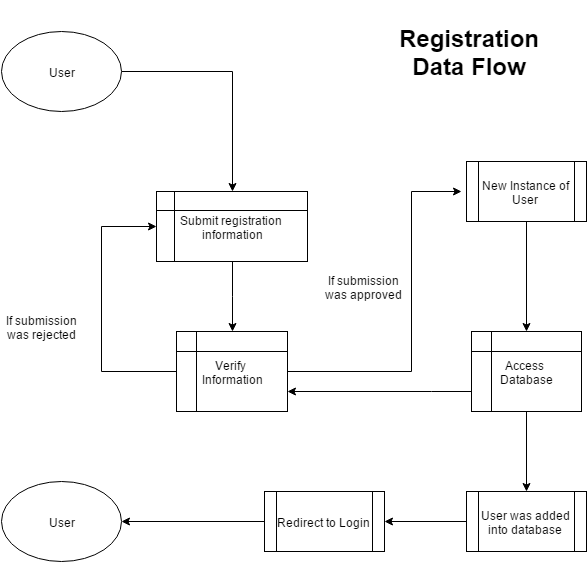
**Data Flow**

**Login Data Flow**



When considering the flow of data in login, the controller must be able to erase any empty spaces from the form. It must also be able to identify whether or not the user’s inputs already match an existing profile. The expected order of events eventually leads to the creation of the user in the database and a redirect to the homepage.

**Registration Data Flow**



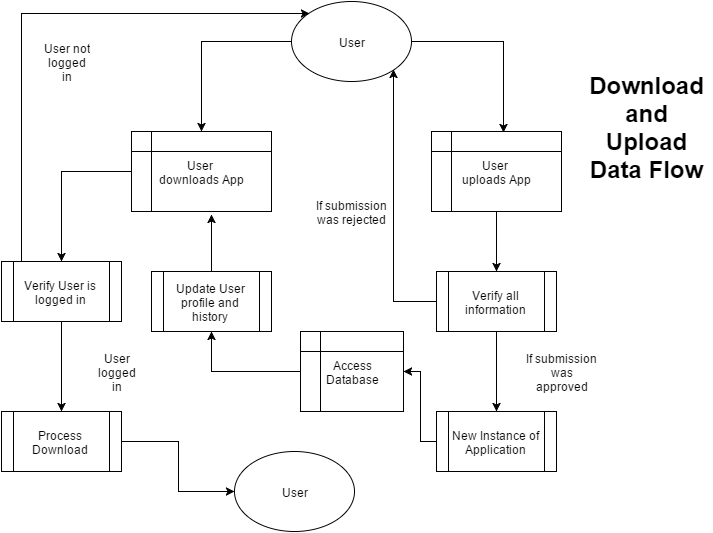
Data flow in the Registration module works very similarly to the Login Data Flow. The controller must also shave off any white spaces found in the form submission. If part of the form is missing or that a user already exist, then return a rejection. Otherwise, the form data will become an instance of a User model and will be added to the database. The user will then be redirected to the login page.

**Search Data Flow**



Initially, the user can request the page to search from 10, 20 or 30 results at a time. The user can submit a search string into the search form and the controller will retrieve the data based on the string. The controller will filter out any apps whose key terms do not contain the user’s submitted string. JQuery will then manage the JSON data from the Controller and render the information for the User to see. Results will be rendered within the GUI with predefined PolymerJS templates. Additional functionality will include sorting functions. These will be accessible using the column headings as buttons. A click on these buttons will trigger events sorting the results by the respective attribute. Through native JavaScript and the core API of PolymerJS, the results will reappear in the sorted order dictated by the column header clicked.

**Download and Upload Data Flow**

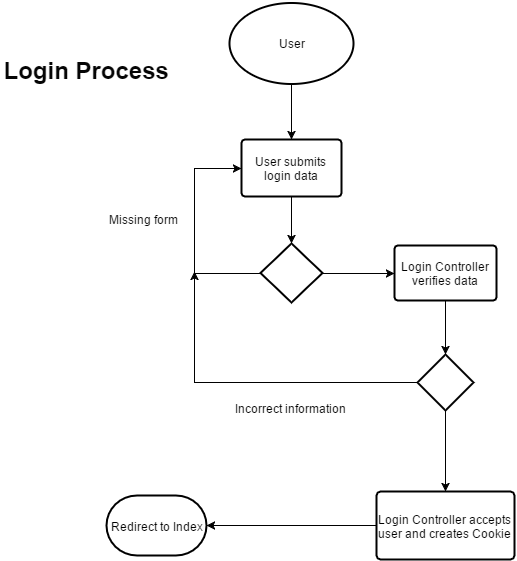


GovApp Store utilizes a single controller when handling the download and upload of Apps. If the User chooses to download, the controller will retrieve the app for the user, but a verification step requires another password submission. This is to ensure that the correct user is logged in.

In upload, a form submission requires the user to submit: name, author, imageLink, price, and description. For this version, the field price will be ignored. The application will request a file submission for the app itself. The app data will be uploaded to the server and the user’s profile will be updated.

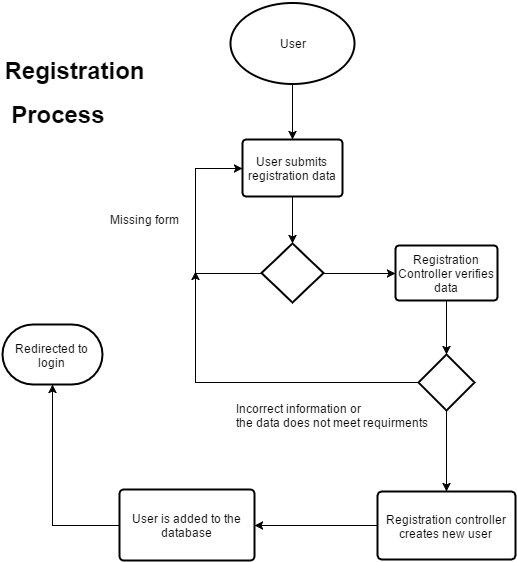
**Process Diagram**

**Login Process**



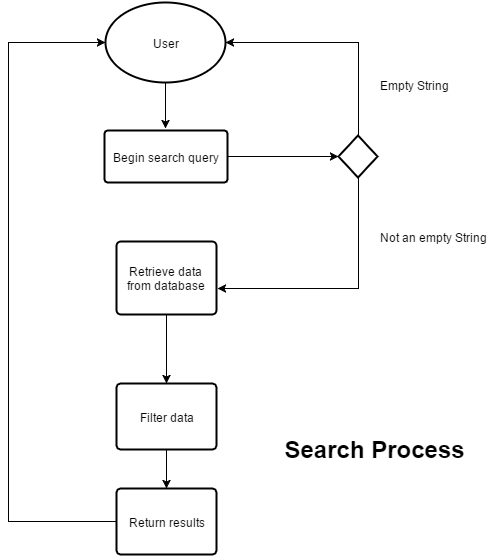
When analyzing the process of login, the webapp must consider the form data. The app must check to see if the user submitted blank data. It must also check to see if the data submitted matches existing data in the database. When this fails, the user will be prompted to resubmit the form data. A reasonable amount of tries will be permitted within a given interval of time before applying a lockout timer.

**Registration Process**



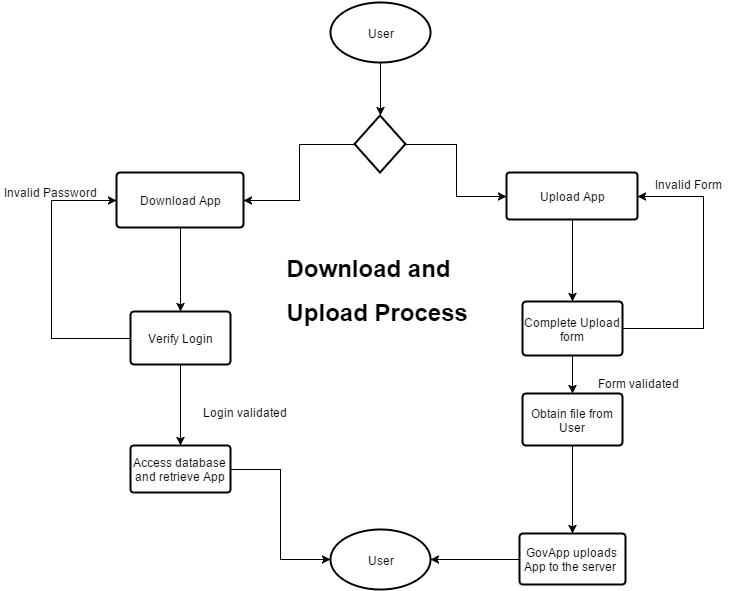
The diagram above looks very similar to the Login since they both share similar form verification processes. In this process, however, submitted strings will be tested against appropriate regular-expression patterns to determine their validity for the database, as well as the presence of any JavaScript, such as eval() statements which can be malicious. The final processing parts of registration puts the form data into the database as a new registered user.

**Search Process**



Search process for the GovApp Store application. This module requires the user to submit a string in the search query in order to interact with this module. A generic search will perform a test of the query string against all attributes in the data model for apps in the collections. The records will be displayed as usual, with all attributes in sensibly ordered rows, but the displayed results will highlight (or bold) the matching strings within the resulting records.

**Download and Upload Process**



Both the download and upload process are available for the User on the homepage. Any user is able to access this functionality without involvement of any admins, but logic to guide the user experience and maintain system security must be implemented with utmost scrutiny. As the number of simultaneous users increases, the system will execute measures for load-balancing and caching to maintain performance, gracefully degrading in worst-case scenarios. Users must always be able to request the service, and the system will manage its resources to best respond to all simultaneous users.

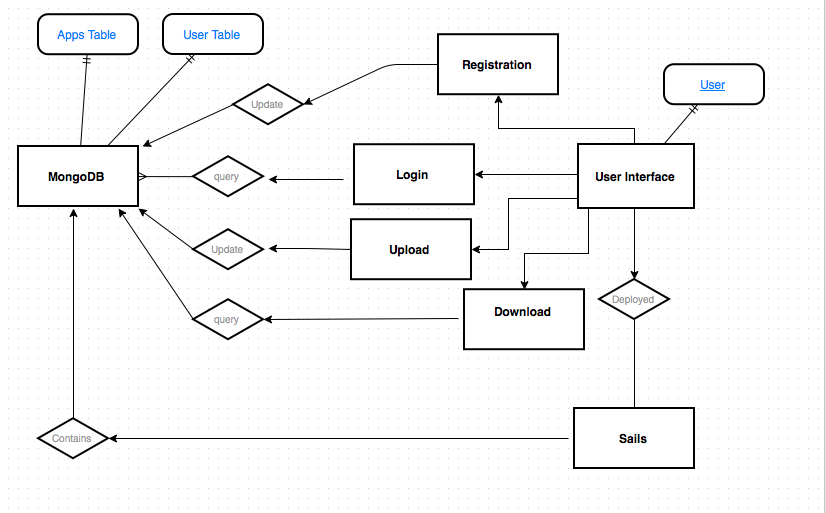
*Downloading*

If the User requests an App download, they must verify once more with their password. If the password is invalid, then they are denied the download. Otherwise, the server will push the app to the user. This event will trigger another database interaction, appending to the user's list of downloaded apps the primary key of the application.

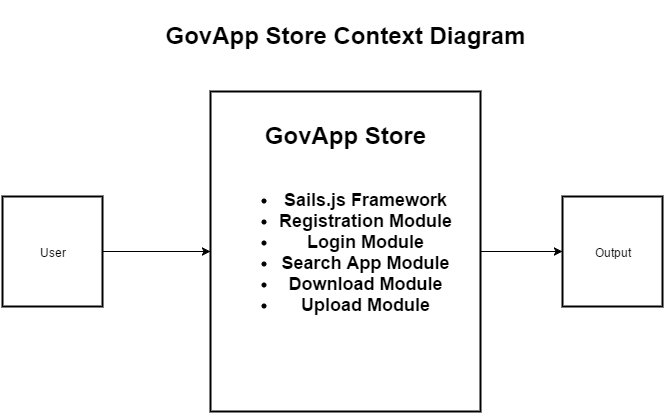
Although MongoDB is capable of storing the binary within the data-store itself, this will be implemented using a string representing the path to the file on the server's hard drive. When this app eventually scales to a distributed architecture, node information or routing logic may also be required to locate the binary.

*Uploading*

In upload, the User is required to be logged in, and fill a form about the App they are uploading. This will include a module whose GUI will prompt the user to locate their file using the standard file explorer conventions on modern PCs. This will allow the module to produce its output: a path to the source code on the user's machine. Additional compilation may be necessary for certain applications. A source file may consist of binary (.exe, .msi, etc), or a complete source tree for certain languages that should be built on the server in accordance with the platform of future users requesting download. Potentially malicious software must be prevented from upload. This may require additional scanning or the prevention of use of potentially malicious code and unrecognized low-level frameworks and utilities. If the form data is invalid (as in, a nonexistent path), then the upload is canceled. Otherwise, the user will then have their file submitted to the server for approval.



**Context Diagram**



The overall design of the GovApp store uses Sails.js as the central framework. Sails.js has an opinionated approach to Node.js modular development, guiding our developers to follow the Model-View-Controller pattern. This framework will be used to create all the modules and assets needed for this webapp. Each module exposes its own back-end API as well as the front-end component which exclusively calls the methods available. The back-end component is in charge of calls to the MongoDB API and any Sails.js funcionality. Front-end controllers will optimally be thin, performing basic validation and mostly data-binding tasks. Direct access of Node.js core utilities is discouraged as it undermines the use of Sails.js. The five main modules (registration, login, search, download, and upload) are the centerpieces of this program.

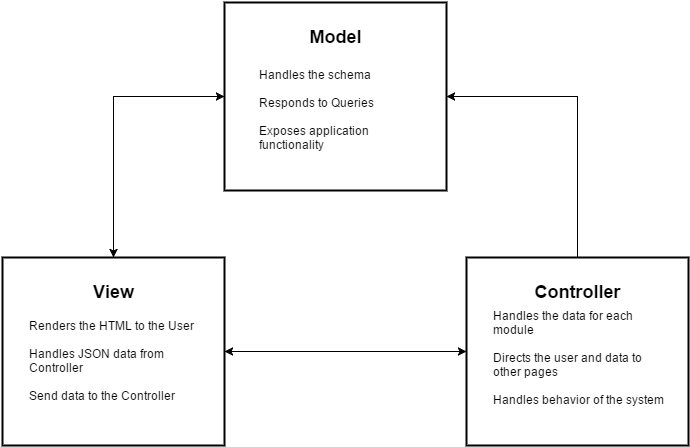
**Architectural Patterns**

GovApp store uses the existing libraries and designs based off of other similar applications. The MVC (Model-View-Controller) pattern creates a separation of concerns which greatly simplifies the process of making updates the the code-base for future contributors.

JQuery is a massive and widely-supported JavaScript library that works exceptionally well in web development. It's main advantages lie in its simplicity and universally-useful abstractions. While many front-end libraries are strongly opinionated (BackboneJS, AngularJS, ReactJS), JQuery is essentially just a collection of plugins which a developer can use without invoking any higher level design patterns dictated by a framework. The headache of keeping code cross-browser compatible with graceful degradation is also handled by JQuery, which does any universally-required heavy lifting to deal with idiosyncrasies of obscure or outdated browsers not supporting ES6 JavaScript. Finally, with wide support, the site can be optimized with parallel-loading of this library by an early request to an external CDN from the lightning-fast Google servers, among others. JQuery is used as the glue that connects the front-end to the back-end of GovApp, and native JS will be discouraged in module creation when a JQuery-based solution exists.

PolymerJS is library that helps create custom tags and unique effects to the HTML. PolymerJS also allows easier data-binding without introducing a heavy framework like AnguarJS. With a community actively developing the library at Google and open-source contributors, this choice is also motivated by the potential for improvement and continued long-term support. PolymerJS is also useful for its flexibility in displaying from either a computer monitor or a phone device.

**MVC**



Most web browsers will be able to support the GovApp website. However, it is recommended that the latest version of each browser is installed.

This application is developed to be able to run on any operable browsing program such as Internet Explorer, Firefox, Google Chrome, and Opera. The latest versions of each browsers is recommended for better access to the website’s interface and implementation.

1. The default view will be on the index.html where the user has access to 4 pages:
   1. Homepage
   2. Download Page
   3. Upload Page
   4. Login Page
2. Each individual page is designed to work with MVC. So that means each page is its own view along with a model and a controller.
3. The controller of each page responds to the request made by the view page and handles the data sent by it.
4. The controllers handles the data by using Models to accurately respond to queries and expose the its functionality to the user.
5. The view will update its content according to the response data sent by the controller.

**REST**

REST or Representational State Transfer is a software architectural design that focuses coordinating restraints to the system so that the resulting application provides high-end performance and a more maintainable system.

It provides a constraint on the following aspects:

1. Performance - To be able to perform by the User’s standards
2. Scalability - The application is capable of handling any amount of implementation
3. Simplicity - Easy to use interface
4. Modifiable - Functional modules should be updated to meet the changing demands
5. Visibility - Define a clear communication between the User and the Service
6. Portability - Should be able to display on many devices
7. Reliability - Resistance to crashes, bugs, and disconnections

**Full Web Stack**

**Server**

Proper use of the GovApp store is that it needs to run on a server that is compatible with Nodejs. GovApp Store utilizes a lot of the packages provided by Nodejs. The following servers are able to run with Nodejs:

1. AWS (Amazon Web Services)
2. RedHat
3. Bitnami

Not only do these servers require the usage of Nodejs, but it must also be able to run MongoDB. MongoDB is a NoSQL database that utilizes heavy read/write of data. SQL databases focuses on relational data. Relational data is not necessary in an application such as GovApp Store.

**Client**

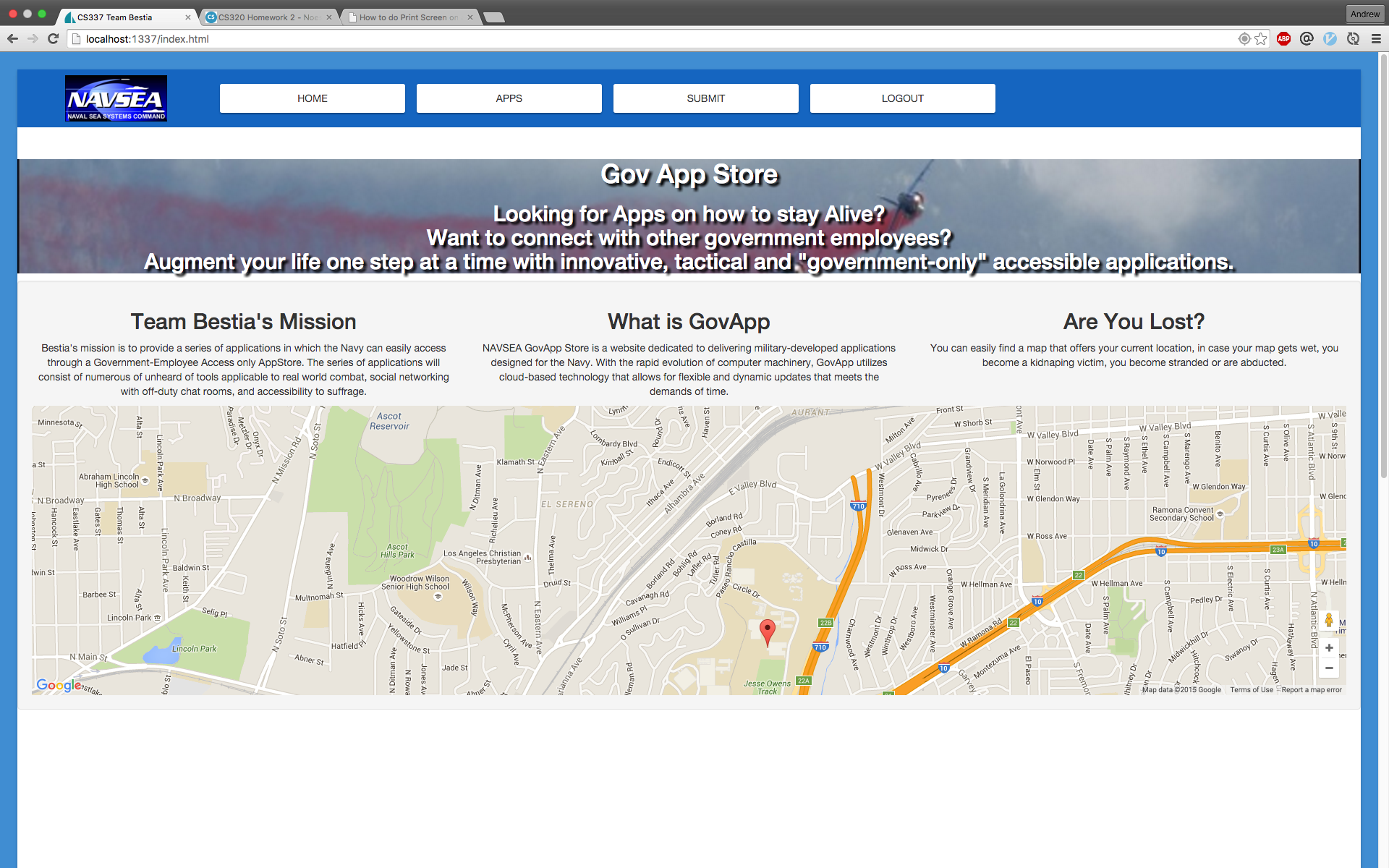
**Modern Browsers**

Modern browsers are compatible with GovApp store. Such browsers like Microsoft Edge, Firefox, Chrome, and Opera are able to work with GovApp Store.

**Mobile**

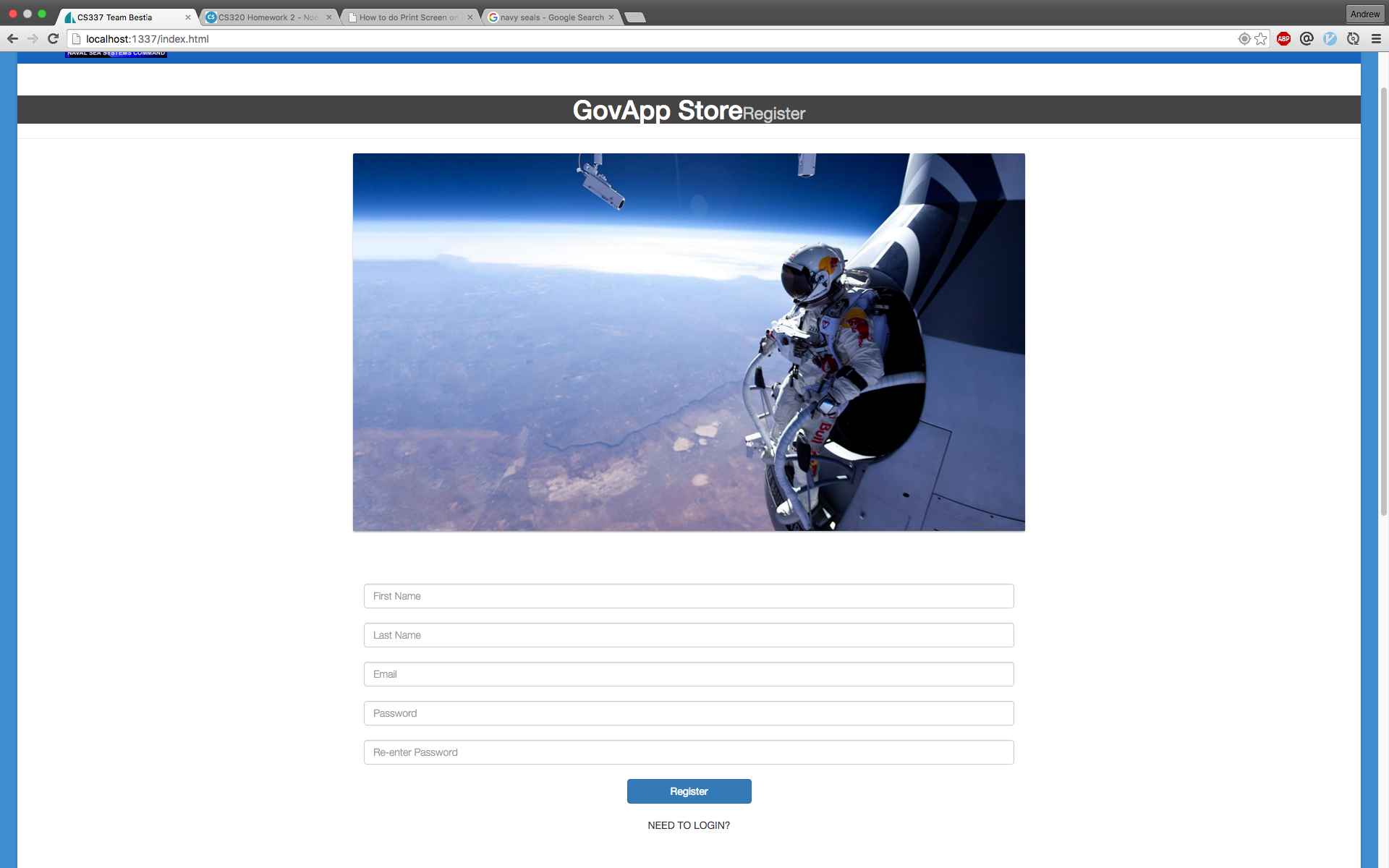
Polymer is a front-end library that is able to create customizable tags that allows for unique displays. Such capabilities allow for mobile interactions as well.

**User Interface Design**

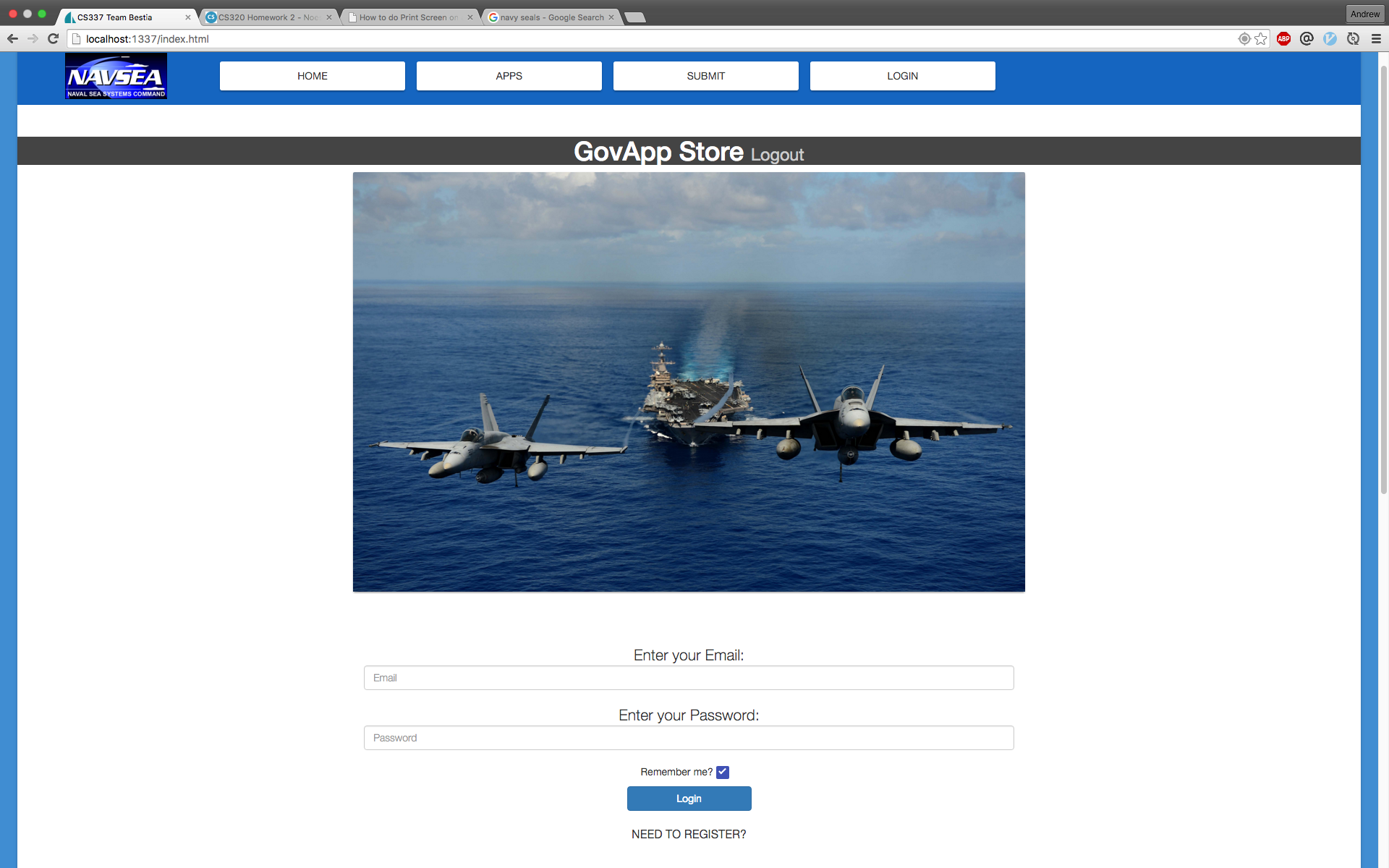


The user interface design was defined specifically to ensure easy interaction between the web application and the end-user. The design of our interface make the complexity for the end-user minimal making it very simple for different “actors” in our ideal environment to interact with the web application.

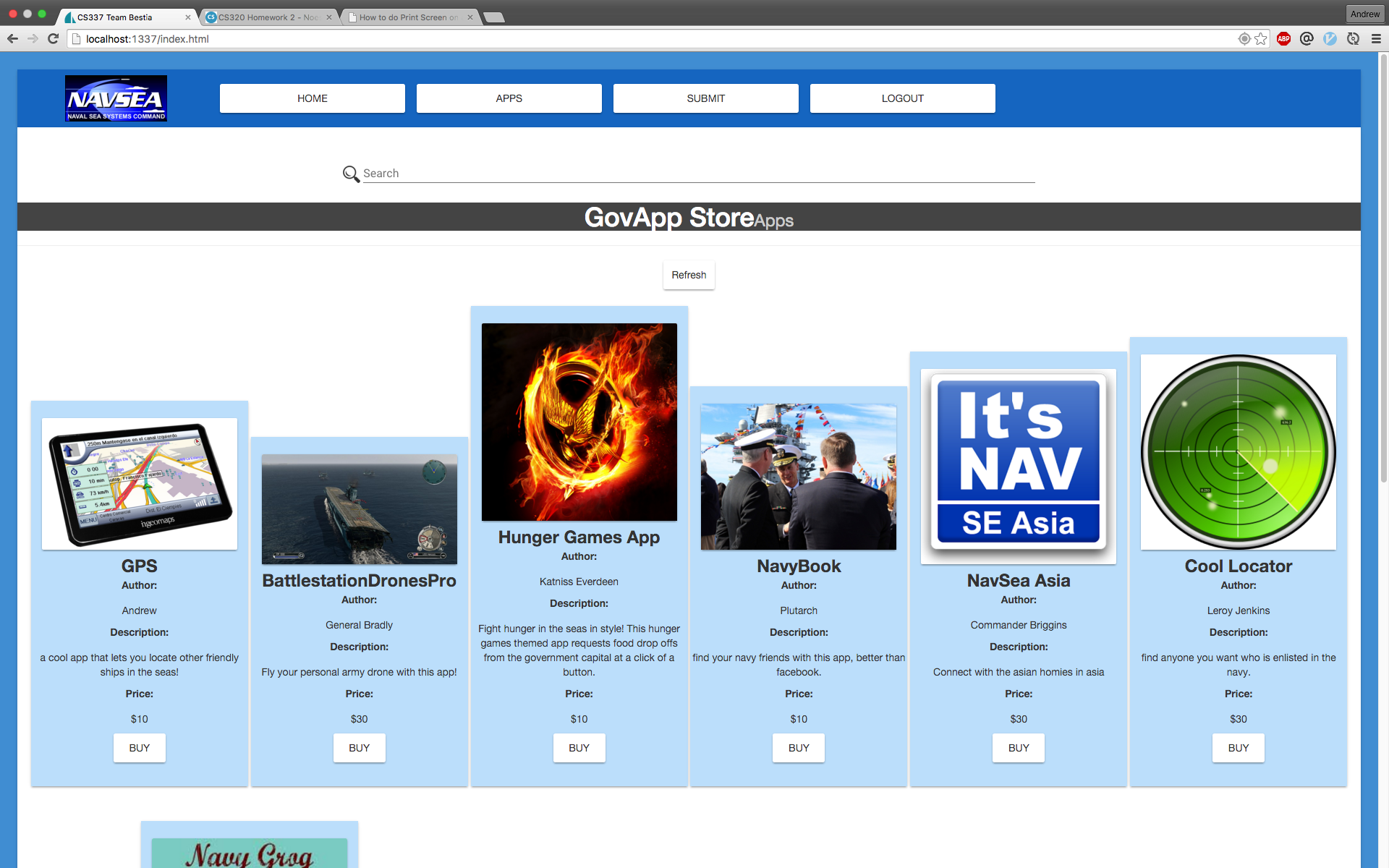
Different components of the user interface do not differ from any other existing web applications when it comes to uploading and downloading applications, registration, and logging in. It is not necessary for the web application to make such activities be done in an unconventional way, in other words we are not reinventing the wheel when there standard ways that work well to accomplish what the web application needs to do.



The user interface for the registration page is straightforward and is not all that different from any other registration page that other web applications use. The “actor” in this scenario will be a potential end-user, most likely a navy official who will either be downloading apps from the app store or a navy software developer who will be uploading apps to the app store. In either case, the potential user will need to enter his/her information in the provided text fields, which will be displayed on the interface when the user navigates to the registration page. Each text field will be labeled so the user knows exactly what information to enter in the text fields. The data the user must provide is his/her first name, last name, email address, and password. Behind the scenes and unseen by the user, the registration controller will determine if the given information is valid and will either make a new member in the app store database or notify the user that the information provided is not valid and will make the user attempt the registration process again until the information entered is valid in order to create a new member. Once the validation procedures are executed, the user interface will need to communicate the result of the validation to the user. The web page will either indicate to the use that the registration was successful or the registration failed due to invalid inputs or if inputs match that of a current user.



The user interface for the login process is just as straightforward as the registration user interface. The “actor” in this case would be an end-user who is already a member of the app store and they have a corresponding email and password that will allow them to login to their account. In the user interface, there will be text fields for the user to enter their email and password. Once they enter the email and password, the login controller will query the database to make sure that the email and password entered match any current members in the database. If the email and password entered do match a user in the database then the user will complete the login process and be redirected to the application where they can upload and download apps. If the information entered does not match any current user in the database then a message telling the user that they need to register will appear.



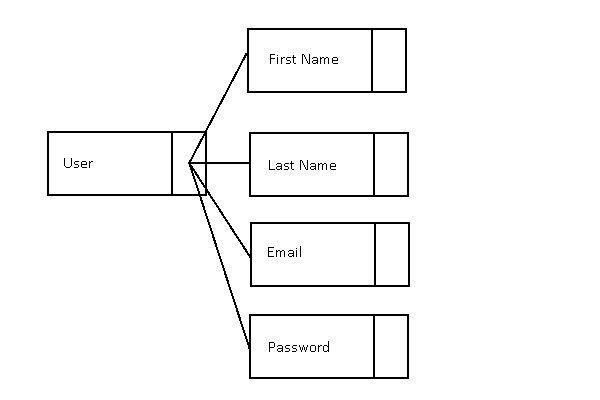
The user interface for downloading apps is more involved when it comes to reducing complexity for the end-user. When the user is at this web page, they will be able to see all the apps that have been uploaded and the information indicating the apps author, name of the app, a photo, and a description of the app. At the top of the web page will be a text field labeled search where the user can enter a keyword and the web application will perform a query to return apps that match the keyword. When the user decides to download a particular app, there will be a button to begin downloading the app. The design of this user interface is not difficult for the user to understand and does not create any unnecessary complexity. There is no great deal of navigation that needs to be done by the user to find an app that he/she would like to download.

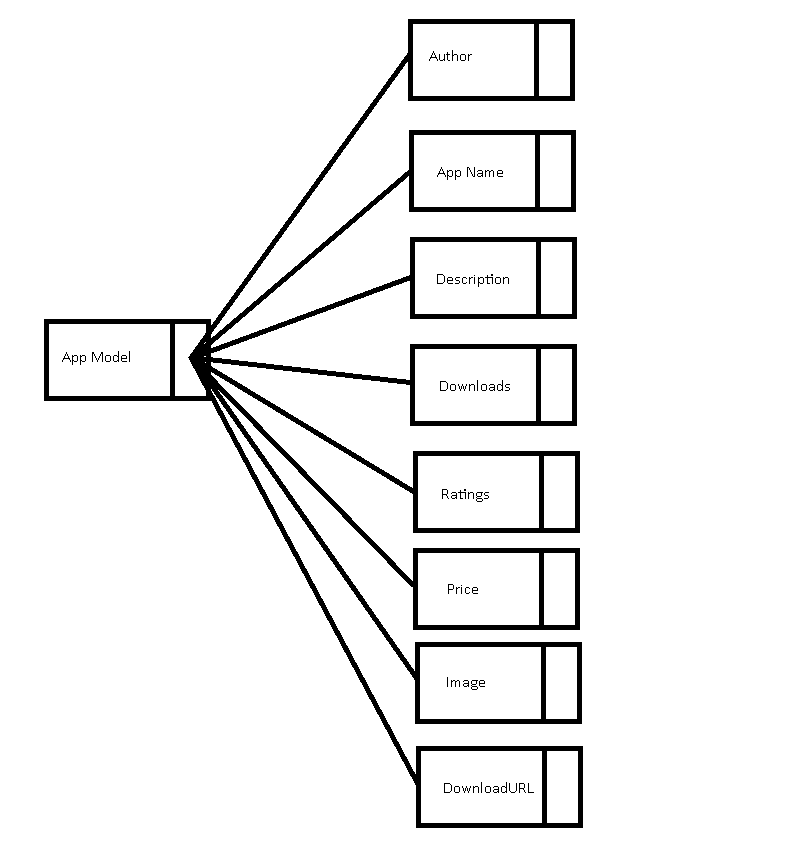


The user interface for uploading is made easy for developers to deploy their apps on to the app store. When a developer is at the upload web page they will see text fields that they must fill out in order to deploy the app. The text fields will require the developer to specify an author, which would be themselves, a name for the app, a description for the app, a photo for the app, and the app itself. For uploading the app, there will be a button that the developer clicks in order to choose a file of his/her device to upload to the app store.

**Database Schema/Models**

When working with the Models, GovApp needs to make sure the data the front-end is sending matches the models how the back-end will read it. JavaScript fortunately is able to read object in chucks, but we are required to have the following data for each individual model. When searching for the Models within the database, the two primary models will be represented in the following fashion:





**Collections**

*User Data Model*

The User model is defined to instantiate and update a user of the app. Basic data members serve to identify the user, while simple data structures are used to personalize the experience. An example of such a structure is the AppsDownloaded list. This list will implememnt a subset of the standard set interface, disallowing the “remove” function, as an app cannot be “undownloaded”. This list serves to customize the user experience in a couple of ways. Firstly, a user can directly view a list of apps they have downloaded for the purpose of downloading a new copy of a desired app on a new machine, or merely for reference. Secondly, this list of apps can be passed to a routine which will return a list of recommended apps for the user, based on the apparent preferences evidenced by their download history. A potential future feature could, for example, either omit or differently display the apps returned in a call to the Search module to make a clear denotation to the user of his/her previous download of the app.

This data member could be extended or refactored to allow for a “Like” or “Rate” functionality, further improving GovApp’s ability to provide recommendations (aside from potentially collecting these data across users to provide general feedback to developers and other users). Such an implementation will require additional business logic performed on the server, and caching will then become a crucial component.

*App Data Model*

The central schema for application metadata will contain any applicable attributes needed for business and UX logic. Basic attributes will include the developer’s name, the app’s name, date created/uploaded, brief description, a path to an image on the server’s hard drive, and potentially a price attribute, should management decide to apply a pricing structure in the future. Attributes related to business logic will include a rating, number of downloads across user base, a path to the download file, and, for future use, a reference ID for a node in a distributed system.

Collections of data are stored in the sails’ MongoDB. Two primary Collections that the database stores are also the two primary models: User and App.

The Collections is updated whenever the user interacts with the forms to update information to the User collections, whenever they desire to download or upload an app to the App collections.