Project Architecture:

Main:

app.js:
Contains the javascript for the core user interface. It is based off an Ionic.
app.configure creates the pages of the applicatio
app.run launches

index.html:
Contains all of the html for the app
Dependencies:
The dependencies are listed at the top of the file. Index requires ng-Cordova and Ionic, as well as references to each of the controllers and services:

Buttons, text boxes, and menus:
Every button is contained within the html body and passes information to a js file.

Services:

locationService:
locationService.js is responsible for watching the device location and reporting changes

State:

- watchId: a uid returned by geolocation.watchPosition used to identify the watch.

Methods:

1) watchLocation: called by settings.js in toggleGPSnotification. Sets a geolocation watch and notifies the user when their zip code changes. For use in future implementation of geo-fenced polls. The watch is cleared and watchId set to null when toggleGPSnotifications is set to false in settings.js
2) getZipCode: takes a position object as a parameter and parses it for postal code. Called by watchLocation in locationService.js

notificationService:
notificationService.js

State:

- URL: url of cityzen core database
- TIMEOUT: present integer representing timeout on calls to the cityzen database

Methods:

1) notificationListEquality: returns false if the new list of polls is longer than the previous, alerting that a new polls has been added
2) getNotifications: polls the database for all polls
3) updateNotificationsConstantly: calls getNotifications and notificationListEquality and adds a notification when it detects a new poll on the database
4) addNotification: generates a local notification with a title and message passed in as arguments

pollService:

pollService is responsible for watching for the release of new polls in the database

State:

- URL: url of table containing polls
- TIMEOUT: integer value for exiting database poll

Methods:

1) getPolls: called from polls.js, checks the database for all polls and returns them to a callback in polls.js

userService:

userService.js is responsible for all interactions between the mobile app and the web service involved in log in, log out, and register, as well as maintaining user settings.

State:

- userID: an integer value representing the current user's userToken which is used for authentication and as an ID for that user's settings choices.
- settings: an object which stores the user's settings, divided into two sub-objects, notifications and user
    - notifications: contains state regarding what the user wishes to be notified of
        - areOn - boolean, whether or not user wishes to receive notifications
        - gpsOn - boolean, whether or not the user has enabled geolocation
        - cactegories - list, contains every category the user has marked interest in
        - cityID - integer, represents the location in the list of possible cities, corresponding to the user's declared city of interest
    - user: the user's personal information
        - first name
        - last name
        - address

Methods:

1) authenticate: called by login.js in the login function. Takes a username and password and authenticates it on the browser using a php script
2) register: called by register function in login.js. Creates new user account in the database.
3) socialLogin: uses OneAll token to authenticate user on the browser using a a php script
4) getNotificationCategories: *not implemented* checks the database for a list of tags
5) getAvailableCities: *not implemented* checks the database for supported cities
6) resetDefaultSettings: called by logout function in settings.js, returns app to its default state

:

Controllers:

(javascript files in www/js/controllers folder)

LoginCtrl - binded to '#/' [login html page]
This controller manages user login, registration, and social media integration. It is the first controller to be initiated. It begins by checking if any user login data has persisted (meaning the user previously didn't log out). If it does exist, we bypass the login screen and leave the user logged in.

Methods:

7) login: grabs the information from the username and password html input boxes and calls the userService authenticate method which will return the user's authenticated userID (if it exists)
8) loginWithUserID: this is the callback function which the userService's authenticate method will return to. It checks to see if the userID is valid and if not returns a log-in fail. If successful, the user can continue to the polls page and their user information is stored to local storage.
9) showRegisterDialog: asks the user to enter a username, password, and email for their new account. This is then passed to userService where we try to register it
10) registerNewUser: this is the callback function which userService's register method returns to. It checks to see if the user registration was successful, and if not asks the user to re-enter their information
11) socialLogin: gets the userID from oneAll's mobile interface
12) showLogin: uses the ionicLoading module to display "Attempting to Log In" after the user has submitted a username and password and we are comparing it against the backend
13) hideLogin: hides this login dialog

NotificationsCtrl - binded to '#/tabs/notifcations'

This controller has a “watch” function that constantly looks for notification updates. If a new notification is added to notificationService’s session data, the new notification is displayed.

Methods:

1) openNotifURL: when a notification is selected, this method redirects the window to the URL associated with the notification. It also sets the notification as “read” (vs. unread)

PollsCtrl - binded to ‘#/tabs/polls’
When this controller is initiated, it immediately calls pollService’s getAllPolls method which gathers all of the poll information from our backend.

Methods:

1) openPollURL: given the poll URL, this method first redirects the user to a php script hosted on cityzen’s backend, which will embed the user’s cookie in the browser, and then redirects the user to the poll. Because the cookie has been added, the user will not need to login on the web application side. Signing into the mobile application is enough. (see the php documentation below for more details)
2) show: displays a loading screen while the polls load from our database
3) hide: hides the loading screen

SettingsCtrl - binded to ‘#/tabs/settings’
This controller manages all User Settings/Preferences

Methods:

1) toggleNotifications: updates user notification preference in userService
2) toggleGPSNotifications: if toggled on, this method watches the user’s zipcode using location service and will output to the user when it changes
3) toggleCategory(id): updates user preference in userService that they want to receive notifications for this specific category
4) logout: returns user to home page and deletes all local storage containing user preferences

DB Tables:

All new tables were created within the *citizena_style21* table. We will only describe tables that were added during the development of this project.

notification:

- *id (int):* The unique id of the notification. Also primary key.
- *title (varchar):* The title of the notification (most important content).

- *description (varchar):* A longer description of the notification.
- *type (varchar)*: The type of the notification.
- *trackback (varchar)*: The URL to open once the notification is clicked (inside the app).
- *read (tinyint)*: A boolean (1=true or 0=false) whether the notification was read.

Note: The app continuously polls the notification table for new notifications, so adding a new notification to the table will send a notification to the client. The category logic is done on the client side.

category: *stores all possible categories for a topic.*
- *id (int):* The unique id of the category. Also primary key.
- *name (varchar):* The name of the category (roads, health etc…)

UserCategories: *join table for users and categories (many-to-many relationship).*
- *UserId (int):* The unique id of the category.
- *CategoryId (int):* The name of the category (roads, health etc…)

Note: UserId and CategoryId are composite keys. A combination is unique.
PollCategories: *join table for polls and categories (many-to-many relationship).*
- *PollId (int):* The unique id of the category.
- *CategoryId (int):* The name of the category (roads, health etc…)

Note: PollId and CategoryId are composite keys. A combination is unique.

Back-end (PHP and HTML):

/home/citizena/public_html/core/poll/listpolls2.php:
A PHP script that allows CORS and responds to AJAX GET requests.
Arguments: None.
Side Effects: None.
Returns: A JSON string representing a list of polls. The polls have been joined using the *UserCategories* table and the *category* table to also include a nested categories object.

example:

GET http://cityzenapp.us/core/poll/listpolls2.php


```
[
    {
        PollId: "1",
        name: "Should LA citizens be paid to vote in city elections?",
        description: "Some believe the incentives could help turnout, others say
        this would send the wrong message.",
        customerid: "2",
        trackback: null,
        socialpost: null,
        CategoryId: "1",
        categories:
```

```
                [
                        "road",
                        "safety",
                        "leisure"
                ]
},
{
                PollId: "2",
                name: "Gateway Improvement",
                description: "Raleigh seeks to improve the city's Southern Gateway along
                South Saunders Street. Which one of these could help most?",
                customerid: "1",
                trackback: "http://southerngatewaystudy.com",
                socialpost: "727876970594007",
                CategoryId: "1",
                categories:
                [
                        "road",
                        "leisure"
                ]
}]
```


/home/citizena/public_html/core/mobile/auth.php:
A PHP script used to authenticate a user by injecting the necessary cookies into their browser. Once a user logs into the mobile app their default browser is redirected to this page so that they do not have to be logged in again.
Arguments: *userId*: The id of the user you would like to login.
Side Effects: Injects authentication cookies based on the userId into your browser. Does nothing if the userId is not valid.
Returns: Nothing.

example:
Redirect browser to:
http://cityzenapp.us/core/auth.php?userId=1
And your browser will be authenticated as user with ID 1.

/home/citizena/public_html/core/mobile/about.html:
An HTML page that is displayed in the about page of the app through an iFrame. Any changes that are made to this page are also displayed in the app. This makes editing the about page extremely easy.

See at: http://cityzenapp.us/core/mobile/about.html