CityHapps Documentation v1.0

Welcome to the City Happs Docs.

Stack

To get started, I've provided a list of all of the technologies used in the project, a brief description, and links to relevant documentation.

Laravel - v4.2.* - PHP framework that we are using as an internal API to handle CRUD operations. We are not using *any* of the template or view functionality baked into Laravel. Because the app is mostly single page, if it can't find a route, it defers to the front-end where they are defined. More on this later. I highly recommend laracasts.com; lots of high quality videos on everything Laravel does.

```
App::missing(function($exception) {
    return View::make('home');
});
```

Laravel Forge - We need to change the credentials for this as well to the email we get back from them.

Forge is a platform that makes deployment a breeze. We currently have it set up to deploy automatically when we merge to the master branch in the git repo. It could be time to upgrade to a machine with more RAM, but to determine if this is necessary, a server monitoring tool like New Relic could be very useful.

Digital Ocean - Hosting provider that syncs with Laravel Forge. Need to get an email to transfer this account to the client.

Angular - v1.2.8 CDN - JS MVVM framework that syncs and consumes the PHP API. The official docs are not the best, however there a lots of resources and examples throughout the web. We are leveraging a number of community created Angular directives. More on this later.

jQuery - v2.0.3 CDN - JS Library- Used for **`public/js/dom.js`** and sporadically throughout **`public/js/app.js.`** Used for category drop down fade toggles and light calendar manipulation.

SASS - CSS preprocessor that extends vanilla CSS. We are using the **`.scss`** variety so make sure to keep that in mind when reading the Sass docs. There are lots of nested rules used in CityHapps. CSS is valid Sass, but not always the other way around.

Grunt - JS build tool. This is used to watch for .scss file changes and tell it where to output the **`.css`** file. This can be substituted with Gulp or CodeKit. Project config is set in the **`Gruntfile.js`** in the root level directory. You may need to change your paths to get it compiling correctly.

MAMP - Local PHP environment. If you are using Windows, look into WAMP. This allows for an easy local environment to develop the site and modify the hosts file. If you are on a Mac, you will need to configure your **`.bash_profile`** to point your Mac to the version of PHP that comes bundled with MAMP because a later version is required for Laravel.
See THIS link for more details.

Vagrant is another option and may be easier if you are familiar with running a VM and serving the site from there.

Composer - Composer is the package manager for PHP. It is used to keep your projects dependencies located in **`composer.json`** up to date. I recommend creating a global alias so you can access it system from the command line with **`composer.`**

Sequel Pro – GUI for managing the MySQL Database. Connect via a socket with username and password : “root”

Installation

1. Clone repo from ‘https://github.com/CrispyInteractive/CityHapps.git’
2. cd into repo and run **`composer update`**
3. inside **`app/config`** copy an existing local directory and create one called local-”yourName”, if you are on Yosemite, you will need the “unix_socket” line
4. inside the /**`bootstrap`** directory, add your local-”yourName” environment with your machines name in an array. Find this in System Preferences => Sharing
5. open Sequel Pro and create a new database called **`vwd`**
6. run **`php artisan migrate`**
7. open MAMP Pro and go to ‘Hosts’. add a new entry with the plus icon on the bottom left. On the right under ‘Document Root’ click the folder icon and point it to CityHapps/public
8. Open your browser and navigate to the new server name you defined.
9. You should see the site up and running.

Adding Events to the Events table

When developing locally, we need to populate the database with event records from the 4 APIs we are consuming. Hit the following endpoints to populate the database in this order. This may take some time. On production, we have a job that runs daily to do this.

1. /storeActiveEvents
2. /storeEventbriteEvents
3. /storeEventfulEvents
4. /storeMeetupEvents
5. /storeEvents

API Accounts

We need an email address from the client so we can transfer these accounts to them.

TO DO - links to the API login page for each

Active -
Meetup -
Eventbrite -
Eventful-

Diving Deeper...

Now that we have the site installed and configured, lets dive into the code for each page and see how requests are handled.

The app starts on **`app/views/home.php`** . This page is where we include stylesheets and scripts. It is also where the header is located as well as the help section slider. You wont find much PHP at all in this page or any of the views for that matter. Starting on this page, Angular takes over and handles all routing and templating.

Any time you see an html attribute starting with **`ng`** ,that means it is handed by Angular. You will see that on the **`<body>`** tag we have the attribute **`ng-controller="appController"`**. This tells Angular that we have access to anything inside this selector and defines the scope of the controller. AppController is an app wide controller that is used to handle voting, user handling, form data etc. More on this in a moment.

Scroll down on **`home.php`** and you will see **`ng-view.`** This is the body of the application and where Angular places the template partials defined in **`public/js/app.js`**

public/js/app.js

We define the templates the app uses on line 1668. You will see all of the urls defined in the

`$scope.Config()` function.

```
.when('/example', {
      templateUrl: 'templates/example.html'
})
```

Templates are located in **`public/templates`**

I wont go too far in depth with Angular conventions, but controllers let you bind data to the DOM using **`$scope.`** Factories and Services are standalone functions that can be required when a controller is created and return values. They are very useful!!!! it makes your code exceptionally DRY and clean. app.js could benefit from a few functions broken out into factories to minimize controller bloat. The problem with this is that all data exchange is happening via AJAX, which need to exist inside the controller so that the DOM can access the data when complete. This could be mitigated by the use of **`$q`** which uses promises so you can place the return value of AJAX's success in a controllers scope, while the actual function performing the AJAX is elsewhere(in a Factory or service).

public/templates/homeView.html

Everything in the homeView is wrapped in **`eventsController.`** We also use **`ng-if=mobile()`** to check if the user is on mobile and conditionally load parts of the template.

We are leveraging UI Bootstrap for most of the animations and modal functionality. The homepage carousel uses UI Boostrap's carousel directive. Because we want 4 events per slide, we need to loop through events in groups of 4 and push each group to a slide as seen on **`app.js`** line 213. There is potential room for improvement here. To loop through in groups of 4 works, but is not as elegant as it could be.

Any time you encounter the **`nextDay()`** or **`prevDay()`** function, we are leveraging the moment.js library. JavaScript offers an internal API for dates, but Moment has a very easy API for manipulating and formatting dates.

When an event is clicked, a modal is opened and data is passed to it using **`resolve`** which is explained briefly in UI Boostrap's docs. The number is also passed so we can track which out of the group of 4 the one clicked is.

Moving through this controller, you will hopefully start to see how each of the functions chained to the controllers $scope interact with the DOM. In the template itself, you will see the actual function calls themselves. This is very powerful, but can be abused in that too much business logic is reliant / bound to the views.

public/templates/dayView.html

The day view contains just the day slider, which functions identically to the homeView day slider. It should only show events for the day listed in the slider.

which you will notice in **`app.js`** I have already began breaking out into a factory. On line 2523 you will see the factory in use. This is the way it should be done. The moment.js time manipulation is broken out into the factory and abstracted away from the controller, making the controller very thin.

A subtle feature, that is affecting the page in a substantial way is the **`<div ng-if="event.start_time | dateToISO | date: 'h:mm a' | uniqueHour">...`** div. The **`uniqueHour`** filter on the end maps to the filter on line 438 in app.js. It checks if the previous and the current hour are the same, if so, dont show the hour , if it is a unique time, insert a clearfix div to break to a new line.

`simpleVoteEvent()` refers to a vote event very similar to **`voteEvent()`** the only difference being there is no number position of where it exists in the group of 4 sliders.

public/templates/mapView.html

This is one of the most complex pages of the site. You will see the familiar **`nextDay()`** and **`prevDay()`** functions at the top, then the category dropdown. Directly after that, you will notice we are looping through the events 10 at a time. Then there is some math to show the proper index throughout the pages and the relevant functions to **`getPrevEvents()`** and **`getNextEvents()`** that works very similarly to the day slider functions, except AJAX is taking place to get events.

Further down the page we see a directive called
`<ui-gmap-google-map></ui-gmap-google-map>`
This is a community supported google map directive called Angular Google Maps It is very powerful directive, and lets you extend the Google Maps API in Angular syntax

An outstanding feature / bug that has yet to be resolved is to have the markers pop out when they have the exact same lat / long. This can be achieved by using the **`cluster`** option of the **`ui-gmap-markers`** directive. Currently I am using the singular version and looping

through the events, however the marker cluster option is not available using the singular **`ui-gmap-marker`** directive.

The relevant controller for this page is app/controllers/EventController.php, specifically the **`geoEvents()`** function where you can see we are grabbing the input lat long of the center point of the map and actually POSTing that to an instane of the Haversine object and returning the 10 relevant events.

When you drag the map, the map updates to show the 10 closest markers to the center of the map. This is handled on the server side in a model file found at, **`app/models/Haversine.php`** . This file is a modified version of Haversine.php by Doug Grubba. You will see how we are setting the radius at the top and toward the bottom we are forming the Eloquent(Laravel ORM) query to return the appropriate results.

public/templates/calView.html

The cal view is almost entirely handled by UI Calendar which is a directive that extends the Arshaw FullCalendar**.** The top month slider portion is actually the Full Calendar month toggles that have been styled to look like the controls from the other pages.

We make a GET request to "/events" with the relevant query string data and have a function call to calEvents that we pass the returned AJAX data into. Currently, we are using moment.js to get all events from the first of the month to the last day of the month.

`$scope.uiConfig` is where the calendar is configured and data is passed into each event modal when clicked.

the viewRender method is admittedly some hacky code. There is no simple way to add a three letter day prefix inside the cell for a day, but that day name is tracked inside the **`.td.fc-day-number`** as a distant relative of some kind. We loop through each day number, find that full day name, slice it to ge the first 3 letters and prepend it the day number.

There is a bunch of code for a month paginator that is not being used and can be removed as the month paginator is being handled completely by the UI Calendar directive.

When you click on the day itself, you should be taken to the day view, on the day you clicked. This is achieved by the **`dayClick`** method and the **`calDayClick`** factory. This is handled by the $routeProvider section of app.js. We are using another controller entirely to handle the calDay interaction. This could be greatly simplified.

public/templates/searchView.html

The search bar actually lives in **`app/views/home.php`** and when submitted calls a function called **`search()`** in the over-arching **`appController.`**

This view works the same way essentially as the day view except we arent clearfixing to a new line for each time, we just hide the time if it is the same as the previous entry.

Something that needs to be done still is to add a class to any event listing with no time above it and give it some margin, so they are all aligned.

Locally, even with thousands of event records, this search is very fast. On the staging server however, there are some major performance issues that need to be optimized.

The relevant controller for search is **`app/controllers/EventController.php`** at the line **`$eventParams['search] = Input::get('search')`**
The relevant DB model and function is **`app/models/EventRecord.php`** and **`scopeEventSearch().`** I actually recently asked a question about this on a laravel user group and believe this search function could be altered to use a callback function which could potentially increase performance. Post can be found here**.**