8. Integration of UI with API

8.1 Introduction

This chapter briefly explores how models, views and controllers are implemented in an Ionic application. The chapter also delves into the steps taken to accomplish the integration, focusing on “logging in”, “selecting a pathway” and “booking a session” processes. It also mentions steps taken to improve the coachee’s user experience.

8.1.1 Overview

After implementing and testing a couple of API calls, the author sought to integrate the UI with the API using Ajax. Subsequently, a sequence of steps was taken to accomplish the integration. They are as follows:

* implementing the API
* testing the API requests using the “curl” command
* formatting JSON response using JSON Editor (an online JSON editing tool)
* implementing Ajax requests in the UI
* testing and debugging using “pry-remote”, “rails console” and inspecting server logs on server side; and using $log.log (a wrapper around console.log) on the client side – the UI
* consuming the Ajax response

8.2 The Routes

The routes in an Ionic application are defined by the $stateProvider module.



Fig 8.1: Screenshot of $stateProvider

In fig 8.1, the url ‘/sign-in’ is bound to a ‘SignInCtrl’ controller and a ‘templates/sign-in-html’ view. The route is defined in www/js/app.js.

8.3 The Controller

*.controller('SignInCtrl', ["$scope", "$state", "$http", "sessionService", "pathwayService",*

*function($scope, $state, $http, sessionService, pathwayService) {*

*}*

Fig 8.2: SignInCtrl controller

Fig 8.2 shows how controllers are defined. The name of the controller is “SignInCtrl” and modules like $scope, $state are injected using dependency injection. The controller is defined in www/js/app.js.

8.4 The View

*<script id="templates/sign-in.html" type="text/ng-template">*

*</script>*

Fig 8.3: The ‘sign-in’ view

Fig 8.3 shows the “sign-in.html” view and, as can be seen, the id attribute of the script tag is assigned the value of “templates/sign-in.html”. All views are located in www/index.html and can be located by their ids. The views could be separated into separate files but the decision to include them in one file was taken so as to improve the loading time of the app i.e. faster to load one file than load more than one files.

8.5 The Model

The services and factories represent the model in an Ionic application as stated in the Software Architecture section – data is also stored in the $scope module so $scope is also seen as a model. Services and factories are defined similarly as controllers.

*.service('authenticationService', ["sessionService", "$state", function(sessionService, $state) {*

*}*

Fig 8.4: authenticationService

The service defined in fig 8.4 is named “authenticationService”. Factories are defined in a similar manner with a “.factory” notation. Services and factories can be used to pass data from one controller to another and are also used to fetch data from an external service. Services and factories are defined in www/js/app.js.

8.6 Directives

*.directive("preLoader", function() {*

*return {*

*restrict: 'E',*

*templateUrl: 'loading.html',*

*replace: true*

*}*

*})*

Fig 8.5: preLoader directive

*<div class="cssload-container padding" ng-show="loading">*

*<div class="cssload-whirlpool"></div>*

*</div>*

Fig 8.6: loading.html

Directives can be used to define reusable UI components. In fig 8.5, a directive named “preLoader” is defined in www/js/app.js. The templateUrl attribute indicates the location of the template of the directive. In the index.html, the directive is used as shown below:

*<pre-loader> </pre-loader>*

Fig 8.7: Use of preLoader directive in index.html

8.7 The Navigation Tabs

*.state('tabs', {*

*url: '/tabs',*

*abstract: true,*

*templateUrl: 'templates/tabs.html'*

*})*

*.state('tabs.home', {*

*url: '/home',*

*views: {*

*'home-tab': {*

*templateUrl: 'templates/home.html',*

*controller: 'HomeTabCtrl'*

*}*

*}*

*})*

*.state('tabs.coaches', {*

*url: '/coaches',*

*views: {*

*'coaches-tab': {*

*templateUrl: 'templates/coaches.html',*

*controller: 'CoachesTabCtrl'*

*}*

*}*

*})*

*.state('tabs.settings', {*

*url: '/settings',*

*views: {*

*'settings-tab': {*

*templateUrl: 'templates/settings.html',*

*controller: 'SettingsTabCtrl'*

*}*

*}*

*})*

Fig 8.8: Defining the navigation tabs

In fig 8.8, the ‘tabs’ state is defined as abstract meaning it doesn’t have an actual url – it serves as a template. Subsequent states “tabs.home”, “tabs.coaches” and “tabs.settings” are sub-states of the “tabs” state, each defining its own view and controller. The navigation tabs are defined in the index.html as shown in Fig 8.9



Fig 8.9: Defining the navigation tabs in index.html

8.8 Log in

As seen from Fig 8.11, the design of the login page is unchanged. Important sections of the login form are shown in the figure below.

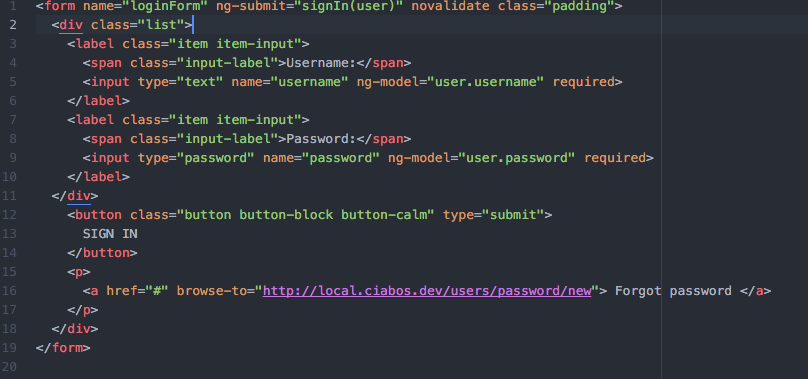


Fig 8.10: Login form

In Fig 8.10 on line 5 and line 9, assigning user.username and user.password to the username and password fields creates a user object. This is accomplished by using Ionic’s “ng-model” directive i.e. ng-model=”user.username” and ng-model=”user.password”. Logging in by submitting the form calls the signIn(user) method in the “SignInCtrl” controller in app.js.

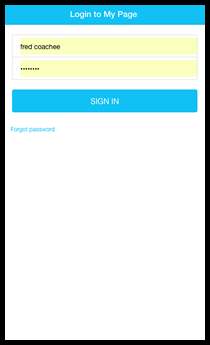


Fig 8.11: Login Page

The signIn method in the “SignInCtrl” controller, as shown in fig 8.12, shows

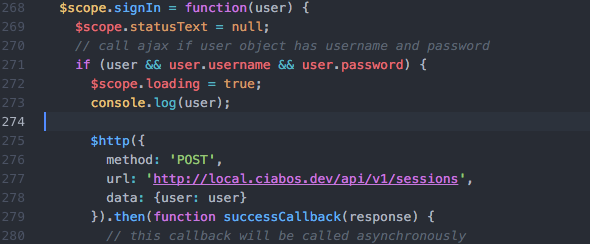


Fig 8.12: signIn function

the function receiving the user object and making a POST request to <http://local.ciabos.dev/api/v1/sessions> with the user object sent in the params. The POST request is made only if the user object is not empty and the user object has a username and password: this prevents the app from making unnecessary API requests.

*Started POST "/api/v1/sessions" for 127.0.0.1 at 2016-04-18 15:13:22 +0100*

*Processing by Api::V1::SessionsController#create as JSON*

*Parameters: {"user"=>{"username"=>"fred coachee", "password"=>"[FILTERED]"}, "session"=>{"user"=>{"username"=>"fred coachee", "password"=>"[FILTERED]"}}}*

Fig 8.13: Server logs

The server logs show the parameters sent and indicate that the request will be processed by the Api::V1::SessionsController#create method. Remember this routing to the controller is possible because of the route (fig 8.14) defined in the config/routes.rb of the API.

*resources :sessions, only: [:create]*

Fig 8.14: login route

"id": 24372,

"username": "fred coachee",

"token": "c5qdL-t12C11CGMdxUbc",

"pathway\_attributes": [

[

28599,

"New Programme for mypage",

"Dummy Test Cohort One"

],

[

28594,

"Sample programme for mypage",

"Dummy Test Cohort One"

],

[

28593,

"sort self learning programme",

"Dummy Test Cohort One"

]

]

}

}

Fig 8.15: response after authentication

The JSON response sent back has an attribute called “pathway\_attributes” that contains details of the coachee’s pathways; these details will be used in the “Home Page”. The JSON response also contains the authentication token, which will be used to authenticate subsequent requests.

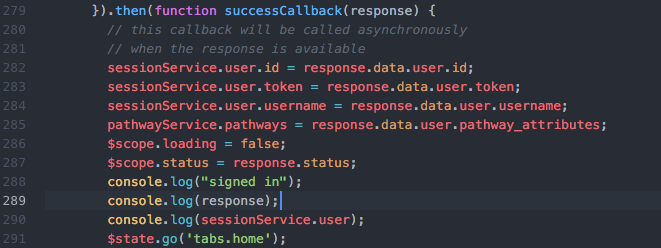


Fig 8.16: onSuccess Callback of signIn method

Upon successful authentication of the coachee, the coachee’s ID, username and token are stored in the sessionService’s user object (lines 282 to 284) so that it can be reused in subsequent Ajax requests. The pathwayService ‘s pathways object (array) is used to store details of the coachee’s pathways (line 285). The details of each pathway in the pathways are its ID, full name and cohort’s full name.

Recall in fig 8.8 the second state is called “tabs.home”, which has a controller called ‘HomeTabCtrl’ and a view called ‘templates/home.html’. Line 291 of fig 8.16 navigates the app from the login page to the home page.

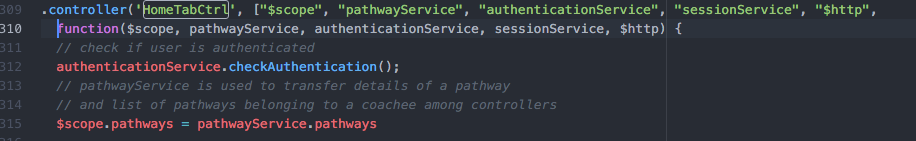


Fig 8.17: HomeTabCtrl

In the “HomeTabCtrl” controller on line 315 (fig 8.17), the pathways that were stored in the “SignInCtrl” controller, are bound to the view by assigning the pathways to $scope.pathways.

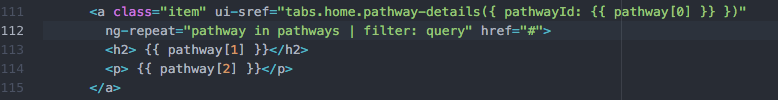


Fig 8.18: anchor tag in templates/home.html

In fig 8.18 on line 112, the “ng-repeat=”pathway in pathways” loops over the pathways array object, setting the pathway’s full name (pathway[1]) and pathway’s cohort’s full name (pathways[2]). The result can be seen in fig 8.19.

On the home page, you can search for a pathway in the top nav bar. This is possible because of the “ng-model=’query’” att”ribute in the search field tag in fig 8.20. The value entered is bound to the “query variable and it’s used to filter the pathway object, as shown in line 112 of fig 8.18 (ng-repeat="pathway in pathways | filter: query").

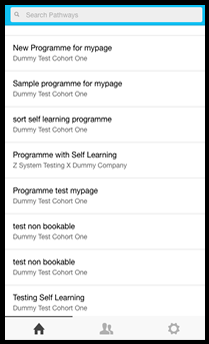


Fig 8.19: Home Page

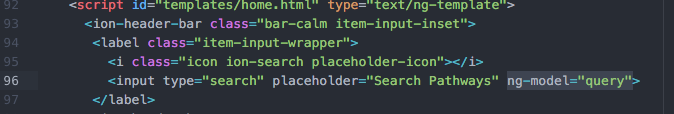


Fig 8.20

8.9 Select a pathway

Selecting the “Sample programme for mypage” pathway (fig 8.19) shows the coaching sessions for the pathway. The ui-sref attribute in fig 8.18 (line 111) navigates the page from the home page to the coaching sessions of a pathway. A closer look at the ui-sref attribute reveals its value is the name of a route or state in app.js.

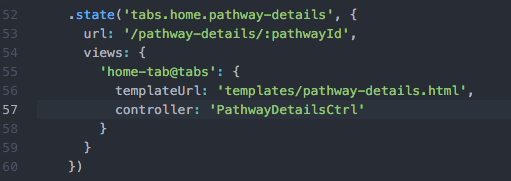


Fig 8.21: tab.home.pathway-details

In fig 8.21, the “tabs.home.pathway-details” route is hooked to a “PathwayDetailsCtrl” controller and “templates/pathway-details.html” view. The route is also expecting the pathwayId to be sent in the params (line 53). The value of the pathwayId sent in the params is stored in $stateParams, an Ionic module.

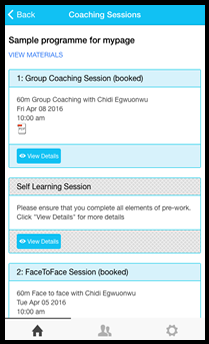


Fig 8.22: Coaching Sessions

Fig 8.22 shows the coaching sessions of the “Sample programme for mypage”. Each type of coaching sessions (Phone Call, Group Coaching, Non-Bookable, Self learning etc) has a different styling depending on its type and booking status (booked or not yet booked). The first coaching session in fig 8.22 is a booked Group Coaching Session whilst the second coaching session is a “Self Learning Coaching Session”, which coachees don’t book. In fig 8.23, the “LockedCall Session by Specialist” is booked and the coachee can delete the booking.

8.10: Book a coaching session

When a coachee books a session, the coachee is presented with the schedule of the coach -- assigned to the coaching session, in the coachee’s timezone. This is made possible by the “around\_action :set\_time\_zone” (fig 8.24) in the Api::V1::ApplicationController class: the timezone of the Rails API is set to the timezone of the logged-in coachee.

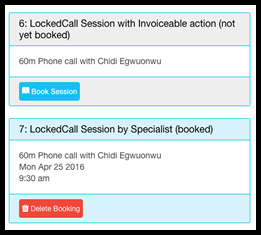


Fig 8.23: More coaching sessions

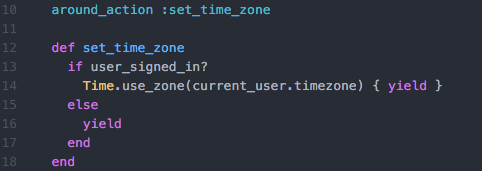


Fig 8.24: set\_time\_zone in Api::V1::ApplicationController

Fig 8.25 shows the dates and times the coach is available. The format of date was chosen to accommodate coachees in countries like USA, who use a different date format than those in countries like England: using ‘04/05/2014’ is confusing.

Upon booking a session the app navigates back to the “Coaching Sessions” page, where it usually makes an Ajax call to fetch the current details of the coaching sessions. But when a coachee books a session, the app fetches the details of the coaching sessions from the pathwayService’s pathway object. This decision improves the user experience of the coachee as the “Coaching Sessions” page is rendered faster. Fig 8.26 shows how this was implemented.

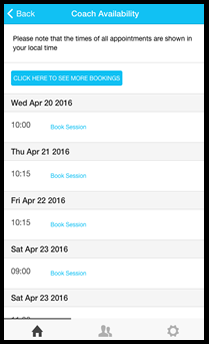


Fig 8.25: Coach Availability

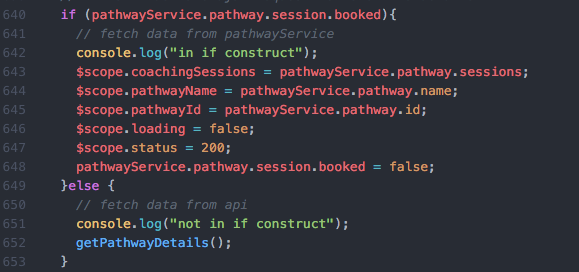


Fig 8.26: check if session was booked

8.11: Refresh lists

Pulling down the page can refresh the list of pathways and coaching sessions. When this is done, the app fetches up-to-date details from the API using Ajax. Refreshing the list of pathways is made possible by adding the “ion-refresher” tag.

*<ion-refresher on-refresh="refreshPathwayList()"></ion-refresher>*

Fig 8.27: ion-refresher

The ion-refresher in fig 8.27 calls the refreshPathwayList() method when the ‘on-refresh’ event is triggered.

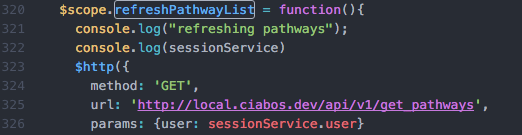


Fig 8.27: refreshPathwayList in HomeTabCtrl

The refreshPathwayList method is defined in the “HomeTabCtrl” controller and makes a GET request to fetch the current list of pathways.

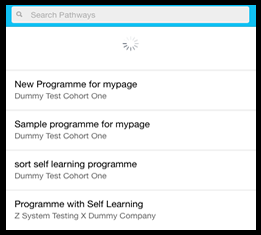


Fig 8.28: Refreshing the list of pathways

When the app receives a response from the API, the app broadcasts a message: “$scope.$broadcast('scroll.refreshComplete')”, to inform the UI the refreshing is complete. This feature adds to the user experience of the coachee.

8.12: Edit photo

All of the processes listed above don’t require a Cordova plugin. When a coachee edits his photo, the coachee selects a photo from his/her phone’s gallery. This feature can only be tested on a device and requires installation of the ngCordova plugin (<http://ngcordova.com/docs/install/>). The $cordovaCamera is used to select the picture from the gallery and $cordovaFileTransfer is used to upload the selected picture to the API. The sourceType option in fig 8.31 (line 784) indicates pictures should be selected from the photo gallery.

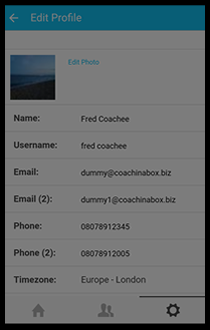
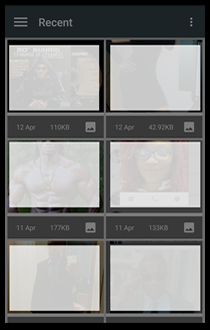
 

Fig 8.29: Edit Photo Fig 8.30: Photo Gallery

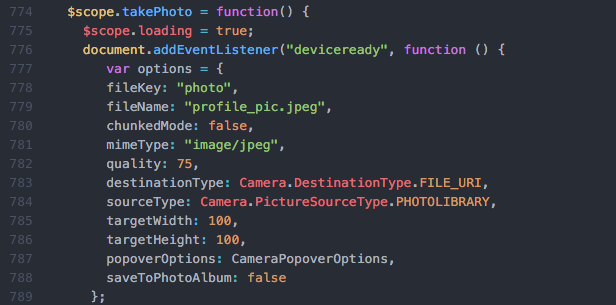


Fig 8.31: setting options of $cordovaCamera

8.13 Conclusion

The integration was not without its issues but with the ability to inspect the server logs; debug the API using pry-remote and the rails console; debug the UI using “console.log” and Ionic’s “$log.log”, issues encountered were resolved. Of course, the online resources at Stack Overflow and Ionic forums were very helpful in resolving issues.

At this stage of the development, the author didn’t adhere to the principle of placing Ajax requests in Services and/or Factories; instead the author placed them in the controllers. This made the reuse of Ajax requests among controllers difficult but the author addressed the reuse of Ajax results in a controller by wrapping the Ajax code in a method. There also wasn’t much divergence with the final high-fidelity design. Git was used for version control.

The author progressed from implementing the list of pathways and their associated actions e.g. selecting a pathway, booking a coaching session etc. to implementing the list of coaches to the display of the coachee’s profile. The author also ensured that the requirements of the client with a high priority were implemented in version 1 of the app.

In the beginning the author returned JSON objects from the API as arrays of arrays e.g. pathway\_attributes in fig 8.15; subsequently the author returned JSON objects as arrays of objects. Selecting values by their keys, as done with objects, proved to be far more reliable and effective than selecting values based on their order in an array.

Testing and debugging was done in the Opera browser and the performance of the app was fast. Switching from using Chrome browser to Opera browser was just due to convenience – the author had many tabs opened in the Chrome browser and reserved the Chrome browser for browsing for resources. The author used both Chrome and Opera browsers in the course of developing this project.