3 ways to build mobile: native apps, mobile websites, hybrid apps.

A hubrid app is a mobile app that contains an isolated browser instance, often called a webview, to run a web application inside of a native app. it uses a native app wrapper that can communicate witht he native device platform and the webview. this means web application can run on a mobile device and have access to the deive,like camera

the user taps on a button

th button calls the angular controller, which calls cordova through the JS API

cordova comunicate with the device using native SDK and request the camera app

ionic:user interface framework

angular:web application framework

it provider web developer a good application structure and the ability to write complete application quickly. ui.router, which provides better application routing and naviation than default Angular eouting module offer

cordova:hubrid app framework

bridge the gap between angular and the device to retireve that information

template: a block of HTML that's is loaded into the application when needed. a view is the use of a template to display data. a view will modify the template based on the data

<ul class= "list-group"

<li class= "list-group-item" ng-repeat= "note in notes" ng-click= "displayNote($index)"

ng-class= "{active: note.id== content.id}">{{note.title}}<br/>

<small>{{note.data| data: 'short'}}</small></li>

</ul>

//ng-class allow you to additional show a class

//{{|data: "short"}} filter

directive: a directive modifies the behavior of the element it's placed on. Some used to modify the display(ngShow, ngClass), others are used with forms(ngModel, ngForm).

conroller: controller are functions that are attached to DOM node and are used to drie the logic of your application. the scope is a shared context between the controller and view. scope has two key roles, to store data and give access to controller methods. data store on the scope is called model.

angular.module("App").controller("controller", function($scope){

$scope.notes= [{},{}];//ng-repeat will display

})

controller sets the notes model with an array of items onto a special object called $scope. everythin in this contorller is isolated from the reset of our application, except for any child scopes.

scope is hierarchical, a scope ca be attached so that only visible to html element and its children. a child scope can look upward to its parent. if you look for a value on a child scope and it doesn;t exist, it will actually check each parent.

the $rootScope is the first scope created by an AJS application to which all other scope are attached. when you put on the root scope, it will available to all scope.

you shouldn't use controller to do DOM manipulation such as changing DOM style, since it should be placed in custom directive

service:reusable obejcts with methods

data binding: sharing between controller and view

angular.module("App").controller("ctrl", function($scope){

$scope.state= {

editing: false

};

});

the scope service is injected so that you can set the state property.

angular.module('App').contorller("ctrl", functin($scope, $http){

$scope.editing= true;

$http.get("/notes").success(function(data){

$scope.notes= data;

}).error(function(err){

$scope.err= "could not load";

});

});

AJS automatically parse JSON into JS obejct. you can inject $http service into your ontroller and use it to load data, it is called dependency injection. AJS service cannot be used global instead of being injected

JS run an asychronous task, it begins with the first part of the task, and then sets it aside to continue running other task.

index.html

<div class= "col-sm-3">

<div class= "panel panel-defult">

<button class= "btn btn-primary btn-xs pull-right">New</button>My Notes</h3>

</div>

<div class= "panel-body">

<p ng-if= "!notes.length">No note</p> //ngif conditinally include or remove element

<ul class= "list-group">

<li class= "list-group-item" ng-repeat= "note in notes">

{{note.title}}<br/>

<small>{{note.data|dat:"short"}}</small></li>

</ul>

</div>

</div>

ngRepeat will loop over every item in an array and create an element for each item.

Filter: convert data to display in the view, without change its value in scope.

you can add more than one filter

{{notes| orderBy: "title"| limitTo: 10}}//limit array to 10

call $scope.view() function, which sets note data to $scope.content

<ul class= "list-group">

<li class= "list-group-item" ng-repeat= "note in notes" ng-click= "view($index)" ng-class= "{active: note.id== content.id}">

{{note.title}}<br/>

<small>{{note.data|dat:"short"}}</small>

</li>

</ul>

add ngClass to add active clas if note is selected

each note in the list can now be clicked.

$index vlaue is passed to the view function, and it is a specifal variable that ngRepeat provides. it help you to know what index of array item is.

$scope.view= function(index){

$scope.editing= false;//set editing to false since you want to just see model

$scope.content= $scope.notes[index];

}

<div class= "panel panel-default" ng-hide= "editing">

<div class= "pnel-heading">

<h3 class= "panel-title">{{content.title}}<button class= "btn btn-primary btn-xs pull-right">Edit</button></h3>

</div>

<div class= "panel-body">{{content.content}}</div>

<div class= "panel-footer">{{content.date|date: "short"}}</div>

</div>

<form name= "editor" class= "panel panel-default" ng-show= "editing">

angular.module("App", []).directive("markdown", function(){

var converter= new Showdown.converter();

return{

scope:{

markdown: "@"

},

link: function(scope, element, attrs){

$scope.$watch("markdown", function(){

var content= converter.makeHtml(attrs.markdown);

element.html(content);

});

}

}

})

link function is used by AJS as part of the rendering process, it will use the $scope.$watch feature, which allows you to listen for when the markdown content is changed.

<div class= "panel-body" markdown= "{{content.content}}"></div>

you will assign content.content model to the markdown attribute to pass the content of the model to your directive's isolated scope. this directive is used as an attribute on the element. anytime content.contetn model is changed, your scope $eatch function will fire to reoncert the new content

directive are used when we want to modigy an element in DOM. directive is easy for test, since it encapsulate function and template

it also can be reusabe, if you want to use controller, you have to write twice code or share scope

editor have two functions, to edit existing note or create a new note.

<div class= "panel-heading">

<h3 class= "panel-ttile">

<input type= 'text' class= "form-control" ng-model= "content.title" placeholder= "new note" required/>

</h3>

</div>

<div class= "panel-body">

<div class= "row">

<div class= "col-sm-6">

<h3>Editor</h3>

<textarea class= "form-control editor" rows= "10" ng-model= "content.content" palceholder= "Note content" required></textarea>

</div>

<div class= "col-sm-6">

<h3>Preview</h3>

<div class= "preview" markdown= "{{content.content}}"></div>

</div>

</div>

</div>

here you use ngModel to link the model alue to the input.

<h3 class= 'panel-title'><button class= "btn btn-primary btn-xs pull right" ng-click= "create()">New</button>My notes</h3>

<h3 class= 'panel-title'>{{content.title}}<button class= "btn btn-primary btn-xs pull right" ng-click= "edit= true">Edit</button></h3>

$scope.create= function(){

$scope.editing= true;

$scope.content= {

title: "",

content: ""

};

};

save() method using $http service again to send the note to the service. the service use post method to create a new note and the put method to update a note. $http.post() and $http.put() both accept a second parameter.

$scope.save= function(){

$scope.content.date= new Date();

if($scope.content.id)

{

$http.put("/notes/"+ $scope.content.id, $scope.content).success(function(data){

$scope.editing= false;

});

}else{

$scope.content.id= Date.now();

$http.post("/notes", $scope.content).success(function(data){

$scope.notes.push($scope.content);

$scope.editing= false;

});

}

}

<button class= "btn btn-primary" ng-click= "save()" ng-disabled= "editor.$inalid">save</button>

$scope.remove= function(){

$http.delete("/notes"+ $scope.content.id).success(functon(data){

var found= -1;

angular.forEach($scope.notes, function(note, index){

if(note.id== $scope.content.id){

found= index;

}

})

})

if(found>= 0){

$scope.notes.splice(found, 1);

}

$scope.content= {

title: "",

content: " "

};

};

<butto class= "btn btn-danger pull right" ng-click= "remove()" ng-if= "content.id">Delete</button>

we can onyl delete when it is saved, since a new node that hasn't been saved cannot be deleted

ionic core part:

naviation:

ionic is built to work only with ui-router

firstly, ionNavView and ionNavBar are the ofundation Ionic, ionNavView acts like a placeholder for different view content to be loaded into the app. if you use navigation, you need at lease one ionNavView

<body ng-app= "App">

<ion-nav-bar class= "bar-positive">

<ion-nav-back-button class= "button-clear">

<i class= "icon ion-ios-arrow-left"></i> Back

<.ion-nav-back-button>

</ion-nav-bar>

<ion-nav-view></ion-nav-view>

</body>

and then you need to declare a list of states for the application. states are a concept given to your from the ui-router. A state is the current representation of the application that's visuaully represented in the view. state would contain URL asscoiated with the view, the name of the controller for the view, and template attached to the view. you will use the $stateProvider service to declare your states, and the $urlRouterProvider to help provide a fallback in case an invalid request is made.

angular.module("app", ["ionic"]).config(functon($stateProvider, $urlRouterProvider){

$stateProvider.state("home", {

url: "/home",

tempalteUrl: 'views/home/home.html'

});

$urlRouterProvider.otherwise("/home");

}).run()

we get first state declared, it is called home, it will only load a template form the url poider in templateUrl,

home.html:

<ion-view view-title= "Aloha areda" hide-back-button="true">

</ion-view>

use ion-view to declare a view template, title is used in navbar and hide-back-button will disable back button

now you can run the code and see that it runs correctly. you should see the app running with a blue navbar and the title "Aloha resort". hide-back-button attributedoesn't want the back button to show.

the example so far has a blank view with a title, so now you need to add more content. you will add ionContent.

ionContent is the nost commonly used content container. it provides a number oof features:

size cotent area to device

aware of header and footer bars

manages scrolling

<ion-view title= "ala adw" hide-back-button= "true">

<ion-content>

<div class= "list">

<a href= "#/reservation" class= "item item-icon-left">

<i class= "icon ion-document-text"></i>see you reservation

</a>

<a href= "#/home" class= "item item-icon-left">

<i class= "icon ion-document-partlysunny"></i>see you reservation

</a>

<a href= "#/reservation" class= "item item-icon-left">

<i class= "icon ion-fork"></i>see you reservation

</a>

</div>

</ion-content>

</ion-view>

use a controller and model for the reservation view:

reservation view you need to load data.so you will attach an object with the properties describing a user's reservation details. add it to reservation.js

angular.module("App").controller("reservationController", function($scope){

$scope.reservation= {

checkin: new Date(),

checkout: new Dare(Date,now()+ 1000\* 60\*60\*24\*7),

room: 156,

rate: 121,

wifi: "resortwifi"

};

});

the controller needs a template like the home.html file to disply information. the reservation view will have a list, similar to the home view.

the major difference between this template and the home template is that you will bind data from the controller into the template.

reservation view template:

<ion-view view-title= "reservation>//declare view with title reservation

<ion-content>

<div class= "list">

<div class= "item item-icon-left">

<i class= "icon ion-key"></i>Room:{{reservation.room}}

</div>

<div class= "item item-icon-left">

<i class= "icon ion-calendar"></i>Check In:{{reservation.chechin|date: "mediumDate"}}

</div>

<div class= "item item-icon-left">

<i class= "icon ion-calendar"></i>Check Out:{{reservation.chechout|date: "mediumDate"}}

</div>

<div class= "item item-icon-left">

<i class= "icon ion-calendar"></i>total:{{reservation.rate\* 7|currency}}

</div>

</div>

</ion-content>

</ion-view>

{{}} is angular expression, it also can be math expression. such as{{reservation\* 7|currency}}

you need to add this view to the application state provider. add this new state as shown in the following listing.

.state("reservation", {

url: "/reseration",

controller: "reservationcontroller",

templateUrl: "views/reservation/reservation.html"

});

we cannot forget that our controller is declared in another file, we don't add it to the index.html to load. <scriopt src= "views/reservation/reservation.js"></script>

weather: you load weather data from an external service. there is lots of service provider this data, but now yu need Opne Weather Map. like reservation view, you need an angular controller. you use $http to handle data loading. i will show how can we use $ionicLoading service to display a loading indicator while the data loads.

you will start by adding template file, then add the controller and data-loading step.

<ion-view view-title= "current weather">

<ion-content>

<div class= "list">

<div class= "item">current conditions:{{weather.weather[0].main}}</div>

<div class= "item">current temperature:{{weather.main.temp}}&deg;</div>

<div class= "item">humidity:{{weather.main.humidity}}&deg;</div>

<div class= "item">highest temperature:{{weather.main.temp\_max}}&deg;</div>

<div class= "item">lowest temperature:{{weather.main.temp\_min}}&deg;</div>

<div class= "item">wind:{{weather.wind.speed}}maph,{{getDirection(weather.wind.deg)}}</div>

</div>

</ion-content>

</ion-view>

the data return by Open Weather Map is fromatted as JSON string that parsed in javascript object by angular.

create weather controller to load external data.

angular.module('App').controller("WeatherController", function($scope, $http){

var directions= ["N", "NE", "E", "SE"];

$http.get("https: //ionic-in-action-api.herokuapp.com/weather").success(function(weather){

$scope.weather= weather

}).error(function(err){

});

$scope.getDirection= function(degree){

if(degress> 338){

degree= 360- degree;

}

var index= Math.floor((degree+ 22)/45);

return directions[index];// {{gerDirection(degree)}} need return value s it can show

};

});

.state("weather", {

url: "/weather",

controller: "WeatherController",

templateUrl: "views/weather/weather/html"

});

then also add <script src= "views/weather/weather.js"></script> to index.html

adding a loading indicator:

loading indicator prevent a user from using the app until the loading has finished. it will be imortant when you need to the data has been loaded and then interact with app.

the loading component has method show() and hide(). you will want to show the loading indicateor while the http is waiting to finish. you need to tell it to hide when the response is return.

angular.module('App').controller("WeatherController", function($scope, $http, $ionicLoading){

var directions= ["N", "NE", "E", "SE"];

$ionicLoading.show();

$http.get("https: //ionic-in-action-api.herokuapp.com/weather").success(function(weather){

$scope.weather= weather;

$ionicLoading.hide();

}).error(function(err){

$ionicLoading.show({

template:"could not load weather"

duration: 3000;//close after three seconds.

});

});

$scope.getDirection= function(degree){

if(degress> 338){

degree= 360- degree;

}

var index= Math.floor((degree+ 22)/45);

return directions[index];// {{gerDirection(degree)}} need return value s it can show

};

});

show() is used before an asychronous command is executed. the second show() method will automatically close after three seconds. it accepts an object containing configuration values.

there only one loading component that can exist. even if you call the show method twice, it just update the already visible component with a new configuration.

infinite scroll with cards for the restaruants view:

in restaurants view, you want to show a list of local restaurant that resort guest might enjoy. you will load a list of restaurant from an external website, dislay the name and image of each restaurant using the cards component. and use infinite scroll to allow more places to be load as a user reaches the bottom of the list.

cards are just a variation on list. it is be used for displaying a piece of data with some visual separation from the rest of the content.

when you scroll to the bottom, the infinite scroll loading indicator will appear while more items are loaded.

each restuaurant will display inside of its own card.

restaurant.html

<ion-view view-title= "local restaurant">

<ion-content>

<div class= "list card" ng-repeat= "restaurant in restaurants">

<div class= "item">

<h2>{{restaurant.name}}</h2>

<p>{{restaurant.adress}}, {{restaurant.city}}</p>

</div>

<div class= "item item-image">

<img ng-src= "{{restaurant.img\_url}}"/>

</div>

</div>

<ion-infinite-scroll on -infinite= "getRestaurants()" ng-if= "total >page" immediate-check= "false"></ion-infinte-scroll>

</ion-content>

</ion-view>

infinte scroll element will call getRestaurant when within a certain distance of bottom.

by adding the card class to the list item, you are able to create the car display. each card is its own list, but you use ngRepeat to create new card for each restaurant.

infinite scroll: when the component is within a certain distance o the view area(default 1%), then it will call the method declared witht he on-infinite attribute.

initially the view won't have any restaurant, it will show a loading ndicator and make a call to the getRestaurants() method to load the initial set. to prevent the loading twice, setting the immediate-check attribute to false, and you will have the controller load the data.

when it doesn't have data to load, you use ng-if to stop infinite scroll. API retur a value that indicate how many pages of data are avilable so you can check that against how many have already loaded.

now you need to add a controller, that handle the loading of restaurant data, and tell infinifte scroll when to stop and hide.

angular.module("App").contorller("restaurantcontroller", function($scope, $http){

$scope.page= 0;

$scope.total= 1;

$scope.restaurants= [];

$scope.getRestaurants= function(){

$scope.page++;

$http.get("https://ionic-in-action-api.herokuapp.com/restaurant?age='+ $scope.page").success(function(response){

angular.forEach(response.restaurants, function(restaurant){

$scope.restaurants.push(restaurant);

});

$scope.total= response.totalPages;

$scope.$broadcast("scroll.infiniteScrollComplete");

}).error(function(err){

$scope.$broadcase("scroll.infiniteScrollComplete");

console.log("err");

});

};

$cope.getRestaurants();//load first page of restaurant from API

});

by using $scope.$bradcast("scroll.infiniteScrollComplete") call, which sends a message to the component to complete. if this event is called, infinite will stop shwng the loading animation

the last thing you need to do is kick off the initial load of data by calling getRestaurants()

app.js

.state("restaurants", {

url: "/restaurants",

contorller: "restaurantsContorller",

templateUrl; "views/restaurants/restaurants.html"

});

and add new contorller gile

<script src= "views/restaurants/restaurants.js"></script>

using slidebox component for app:

see a quick tour of waht the app can do. you can use ionSlideBox component

sildebox: show items in a list tha you can swip between.

$ionSlideBoxDelegate service can be used to programmaticaly control the slidebox. you could have a button that could use the slidebox service to force the slidebo to go to particular slide.

in this examples, the tour will show three slidee suing th slidebox.

tour.html:

<ion-view view-title= "welcome to alo resort" id= "tour-view">

<ion-nav-content side= "right">

<a class= "button button-clear" href= "#/home" nav-clear>Start</a>

</ion-nav-butons>

<ion-slide-box show-pager= "true">

<ion-slide>// each ionSlide is automatically added as a new slide to slidebox

<span class= "icon icon-slide ion-document-text"></span>

<h3>see your reservation</h3>

</ion-slide>

<ion-slide>// each ionSlide is automatically added as a new slide to slidebox

<span class= "icon icon-slide ion-fork"></span>

<h3>see your retaurant</h3>

</ion-slide>

<ion-slide>// each ionSlide is automatically added as a new slide to slidebox

<span class= "icon icon-slide ion-ios-sunny"></span>

<h3>see your weather</h3>

</ion-slide>

</ion-slide-box>

</ion-view>

slidebox have three slides, the slidebox will only be as large a sthe content inside of it, and right now because of the icon and header tags are standard HTML elements, they will only make the slidebox as tall as the text itself.

you need css:

#tour-view .slide{

height: 100%;

}

#tour-view .slider-slide{

padding-top: 100px;

text-align: center

}

#tour-view .icon-slide{

font-szie: 20em;

display: inlink-block; //make icons large and display them as inline block

}

<link href= "views/tour/tour.css" rel= "stylesheet">

and you want to open app and set tour view as default. so change $urlRouterProvider.otherwise

.state("tour", {

url: "/tour",

templateUrl: "views/tour/tour.html"

});

$urlouterProvider.otherwise("/tour");

now you have learn how to build a navigable interface for your mobile app. ionic app are build around the idea of states. states are declared in the config()method using $stateProvider.

ionic load your template inside of the ionNavView component whent he state changes.

the ionNavbar tag automatically can update the title of the navbar based on the current view

the list and card component are mobile firednly way to display lists of content

you can load data using the $http service into your controller, and use the $ionicLoading service to show a loading indicator while it loads

the $ionicSlideBox is a fully featured slideshow component for a mobile interface, and you used it in this chapter as an app introductory tour.

in next chapter we lean how using tabs for your app navigation

three tabs: show current market, to view a chart of historical rates, and for currency management

you will learn how to have a navigation inside each tab. also when you load from an external source, the data is cached even if you change between tabs.

you first need to add the base navigation elements: ionNavBar, ionnavView.

index.html

<body ng-app= "App">

<ion-nav-bar class= "bar-positive">

<ion-nav-back-button class= "button-clear">

<i class= "ion-chevron-left"></i>Back

</ion-nav-back-button>

</ion-nav-bar>

<ion-nav-view></ion-nav-view>

</body>

js/app.js

angular.module("App", ["ionic"]).config(function($stateprovider, $urlRouterProvider){

$stateProvider.state("tabs", {

url: "/tabs",

templateurl: "views/tabs/tabs.html"

});

$urlRouterProvider.otherwise("/tabs");

})

adding tabs contaner and three tabs to app:

tabs are commonly used to show a visual connection between serveral views. tabs can be used in anywhere, exccept in ionContent

ionic provider you ionTabs and ionTab, much like ionSlideBox.

tabs.html

<ion-tabs class= "tabs-icon-top tabs-positive">

<ion-tab title= "rates" icon-on= "ion-social-bitcoin" icon-off= "ion-social-bitcoin-outline">//give active and inactive states

</ion-tab>

<ion-tab title= "rates" icon-on= "ion-social-bitcoin" icon-off= "ion-social-bitcoin-outline">

</ion-tab>

<ion-tab title= "rates" icon-on= "ion-social-bitcoin" icon-off= "ion-social-bitcoin-outline">

</ion-tab>

</ion-tabs>

add ionNaview for each tab:

your tabs are empty, and you want to use ionNavView component to load your componenet. each tab should have its own navigational history

tabs don't require individual views, is one large view that has only one tab displayed at once. with other tabs hidden. the benefit of using the ionNavView element inside of each tab gives you one major benefit. each tab is able to have its own history

when using tabs for navigation, adding individual views is useful. thie chapter exmaple will demnstrate this use case.

you will start by adding the ionNavView components into your tabs. you will have to give each one a name so that can be identified later. each tab will also be given a ui-sref attribute that will turn tab icons into buttons to navigate.

tabs.html

<ion-tabs class= "tabs-icon-top tabs-positive">

<ion-tab title= "rates" icon-on= "ion-social-bitcoin" icon-off= "ion-social-bitcoin-outline" ui-sref= "tabs.rates">//give active and inactive states

<ion-nav-view name= "rates-tab"></ion-nav-view>

</ion-tab>

<ion-tab title= "rates" icon-on= "ion-social-bitcoin" icon-off= "ion-social-bitcoin-outline">

<ion-nav-view name= "rates-tab"></ion-nav-view>

</ion-tab>

<ion-tab title= "rates" icon-on= "ion-social-bitcoin" icon-off= "ion-social-bitcoin-outline">

<ion-nav-view name= "rates-tab"></ion-nav-view>

</ion-tab>

</ion-tabs>

this adds three new ionNavView component with a different name. the ui-sref attribute act like a normal href attribute to link to a particular state based on name, so instead of havinf a URL, you have a state name.

//each tab has its own ionNavView, and the tabs container is the parent, each tab have its own navigation history because of it's unique ionNavView.

even if only one of these three views will be visible, all three will be part of the same parent tabs view.

ui-router has a featured called nested states that allow you to declare states with a hierarchical. the tabs route that display the tabs is like the root state, each tab is a child state underneath it. you need to update the app config() with the new states and modify the tabs state as well.

.config(function($stateProvider, $urlRouterProvider){

$stateProvider.state("tabs", {

url: "/tabs",

abstract: true,//update abstract to true, since you always use a child

templateUrl: "views/tabs/tabs.html"

}).state("tabs.rates", {//.means parent.child relationship

url: "/rates",//it is a child route, so it appends this to URL of parent

views: {

"rates-tab":{

templateUrl: "views/rates/rates.html"

}

}

})

.state("tabs.history", {

url: "/history",

views: {

"history-tab":{

templateUrl: "views/history/history.html"

}

}

});

$urlRouterProvider.otherwise("/tabs/rates");

});

abstract: true property make it possible to declare it as a parent but doesn't allow it to be active state. subview of the state are declared, the view must be named the same as the name you gave to the ionNavView earlier. it should inject the speciiced tmplate into that view.

the following taks to is to add template for thre tabs.

<ion-view view-title= "current rates">

<ion-contetn>

</ion-content>

</ion-view>

.factory("currentcies"), function(){

return[{code: "aud", text: "australina", selected: true},{},{}]

});

the currecy is an array containing a list of oejcts containing information about currency.

//tab has ionNavView, allow it to use a back button that only for a given tab instead of the whole app

the first place you will use currencies service is in a controller for the rates views. the controller will take care of loading the curent rates from the BitcoinAverage API.

rates.js

angular.module("App").controller("ratecontroller", function($scope, $http, currencies){

$scope.currencies= currencies;

$scope.load= function(){

$http.get("https://api.bitcoinaverge.com").success(function(tickers){

agular.forEach($scope.currencies, function(currency){

currency.ticker= tickers[currency.code];

curency.ticker.timestamp= new Date(currency.ticker.timestamp);

});

});

};

$scope.load();

});

this controller taks care of loading data. you store current rates on the currencies service. this is a single data object that you will pass around and use multiple times in other places.

now, you need to display it on the screen. it is time to update your template to loop over the currencies and display the data loaded

rates.html

<ion-view view-title= "current rates">

<ion-content>

<ion-list>

<ion-item ng-repeat= "currency in currencies|filter: {{selected:true}}">

{{currency.code}}- {{currency.text}}

<span class= "price" ng-if= "currency.ticker.last== currency.ticker['24h\_avg']">

{{currency.ticker.last||"0.00"}}<br/>0.00

</span>//price box shown when current price equal to 24-hour average

<span class= "price negative" ng-if="currency.ticker.last<currency.ticker['24h\_avg']">

{{currency.ticker.last}}<br/>

<span class= "icon ion-arrow-down-b"></span>{{currency.ticker['24h\_avg']-currency.ticker.last| number: 2}}

</span>//when decrease

<span class= "price positive" ng-if="currency.ticker.last>currency.ticker['24h\_avg']">

{{currency.ticker.last}}<br/>

<span class= "icon ion-arrow-up-b"></span>{{currency.ticker['24h\_avg']-currency.ticker.last| number: 2}}

</span>//when increase

</ion-item>

</ion-list>

</ion-content>

<ion-footer-bar class= "bar-dark">

<h1 class= "title">Updated{{currencies[0].ticker.tmestampe| date: "mediumTime"}}</h1>

</ion-footer-bar>

</ion-view>

"currency|filter:{{selected: true}}"

there are for displaying the current price and the trend compared to past 24hour average. there are three possible situation. equal, heigh, lower than average. only one of three span will display.

style.css

.item .price{

font-weight:bold;

font-size:12px;

color:#fff;

position: absolute;

right;15px;

height: 42px;

top: 5px;

width: 80px;

text-align: center;

padding: 6px;

line-height: 1.2em;

}

<script src= "views/rates/rates.js"></script>

app.js

.state("tabs.rates", {

url: "/rates",

views{

"rates-tab":{

templateUrl: "views/rates/rates.html",

contorller: "ratescontroller"

}

}

})

display currency's details in the same tab view:

you can create a new view that the rates tab can navigate to. and use back button on that tab to main list. if you navigate to another tab and return, then the detail view will still be active with the back button to return to rates.

details.js

angular.module("App").controller("DetailController", function($scope, $stateParams, $state, currencies){

angular.forEach(currencies, function(currency){

if(currency.code== $stateParams.currency){

$scope.currency= currency;//loop over each currency to find requested currency and store it on scope

}

});

if(angular.isUndefined($scope.currency.ticker){

$state.go("tabs.rates");

})

})

when you declare this state, you add a parameter called currency, and the controler use the $stateParams service to access the value of that parameter.

now you need detail.html view, at the bottom there will be a link to the historical data for that currency, that link to another tab.

detail.html

<ion-view view=title= "detial for {{currency.code}}">

<ion-content>

<ion-list>

<ion-item>Last<span class= "badge badge-stable">{{currency.ticker.last}}</span></ion-item>

<ion-item>ask<span class= "badge badge-balanced">{{currency.ticker.ask}}</span></ion-item>

<ion-item>bid<span class= "badge badge-assertive">{{currency.ticker.bid}}</span></ion-item>

<ion-item ui-sref= "tabs.history({currency: currency.code})" class= "item-icon-right">View history<span class= "icon in-arrow-right-b"></span></ion-item>

//sref link to tabs.history state and passes a currency code parameter

</ion-list>

</ion-content>

<ion-footer-bar class= "bar-dark">

<h1 class= "title">Updated{{currency.ticker.itemstamp|date: "mediumTIme"}}</h1>

</ion-footer-bar>

</ion-view>

the last item has ui-sref attribute, here you use it like a function and pass an object, which is the way your can pass parameters to antoher state.

<script src= ".../detail.js"></script>

app.js

.state("tabs.detail",{

url: "/detial/:currency",//:currency indicate a parameter that will be currency code

views: {

"rates-tab":{

templateUrl: "views/detail.html",

//reuse same rates tab view since this state is designed to be displayed there

contorller: "detailcontroller"

}

}

})

the :currency paraemter will be set to a currency code, and passed to the state so it knwos which currency to use. this value will be made available to $stateParams int he detail contorller.

rates.html

<ion-view view=title= "current rates" hide-back-button= "true">

<ion-content>

<ion-list>

<ion-item ng-repeat= "currency in currencies| filter: {{selected: true}}" ur-sref= "tabs.detail(currency: currency.code})">

</ion-list>

</ion-content>

</ion-view>

//add ui-sref and target tabs.detail state,

you use ui-sref to link to tabs.detail state . you pass currency code as a parameter so the detail view knows which currency was selcted.

refresh the bitcoin rates:

add ionRefresher component, pull down and release to triggger a refresh

you also want to make user have a quick help panel taht explain the information they're looking at. use ionPopoverView component.

ionRefresher:

you have to upload bith the rates template and controller to support ionRefresher.

<ion-view view-title= "current rates" hide-back-button= "true">

<ion-content>

<ion-refresher on-refresh= "load()" pulling-text= "pull to refresh">

</ion-refresher>

</ion-content>

</ion-view>

it will inject the hiiden ionRefresher component above the content, and when the user pulls, the component will appear. it also shwon an icon, you can configure which icon are used.

when ionRefreshen compoent is pulled far enough and released, it will call load() method. and you need to tell component when the data has loaded. if component doesn't know when the data is done loading and will never hide. so you have to update the load() method and broadcase an event that will tell the ion0Refresher component to complete.

rates.js

$scope.load= function(){

$http.get("http://api.btconvencie").sucess(function(tickers){

angular.forEach($scope.currencies, function(currency){

currency.ticker= tickers[currency.code];

currency.ticker.timestamp= new Date(currency.ticker.timestamp);

});

}).finally(function(){

$scope.$broadcast("scroll.refreshComplete");//after http request has completed, it will call finally()

});

};

you will use the finally() method, which is part of the angular promises API, to broadcase the scroll.refreshComplete.

you don't want the refresher to continue showing even if there was was an error. so the finally() method is abled to execute no matter what.

$ionicPopover: showing help in a popover

$ionicPopver component is typically used by having a button in the header that opens the popover. you aren't limited to what you can put into.

you will start by adding a new template file witht the content of your popover. i like it be a subview.

/views/rates/help-popover.html

<ion-popover-view>

<ion-header-bar>

<h1 class= "title">about bitcoin</h1>

</ion-header-bar>

<ion-content>

....

....

....

</ion-content>

</ion-popover-view>

rates.js

angular.module("App").controller("ratescontroller", function($cope, $http, $ionicPopover, currencies){

$scope.currencies= currencies;

$ionicPopover.fromTemplateUrl("views/rates/help-popover.html", {

//declare a popover from template URL and assign parent scope as scope

scope: $scope,

}).then(function(popover){

$scope.popover= popover;// when template has loaded, assign popover to scope

})

$scope.openHelp= function($event){//scope method to open popover

$scope.popover.show($event);

};

$scope.$on= function("$destroy", function(){//$destroy event, which is when view is destroyed, clean up popover

$scope.popover.remove();

});

})

you can use $ionicPopover to create a new popover from template URL. connect the scope by passing on object with{scope: $scope}, often you need this to access parent scope.

you use $scope popover.show($event) to show the popover, you'll need to add an ngClick to a button to trigger and hide

//some component need to be manually removed. modals and popovers require the app to remove the component when the scope is destroyed. $destroy fired when the current scope has been deleted from memory

rates.html

<ion-nav-button side= "primary">

<button class= "button"ng-click= "openHelp($event)">About</button>

</ion-nav-button>

angular.module('App', ["ionic", "hightcharts-ng"])

history.html

you need to add selected box. you will use insert list with just one item to create the select box container.

<ion-view view-title= "hourly average price" hide-back-button= "true">

<ion-cntent>

<div class= "list list-inset">//use inset list to contain select box

<label class= "item item-input item-select">

<div class= "input-label">

currency

</div>

<select ng-change= "changeCurrency()" ng-model= "history.currency">

<option ng-repeat= "currency in currencies| filter: {selected: true}" value= "{{currency.code}}" ng-selected= "history.currency== currency.code">{{currency.code}}- {{currency.text}}</option>

</select>

</label>

</div>

<highchart cnfig= "chart"></highchart>//it accept a config attribute with a chart object

</ion-content>

</ion-view>

when value change, ngChange trigger the changeCurrency() method, which update the view to display the chart for that currency

history.js

angular.module('App')

.controller('HistoryController', function ($scope, $http, $state, $stateParams, Currencies) {

$scope.history = {

currency: $stateParams.currency || 'USD'

};

$scope.currencies = Currencies;

$scope.changeCurrency = function () {

$state.go('tabs.history', { currency: $scope.history.currency });

};

$scope.chart = {

options: {

chart: {

type: 'line'

},

legend: {

enabled: false

}

},

title: {

text: null

},

yAxis: {

title: null

},

xAxis: {

type: 'datetime'

},

series: []

};

$http.get('https://api.bitcoinaverage.com/history/' + $scope.history.currency + '/per\_hour\_monthly\_sliding\_window.csv').success(function (prices) {

prices = prices.split(/\n/);

var series = {

data: []

};

angular.forEach(prices, function (price, index) {

price = price.split(',');

var date = new Date(price[0].replace(' ', 'T')).getTime();

var value = parseFloat(price[3]);

if (date && value > 0) {

series.data.push([date, value]);

}

});

$scope.chart.series.push(series);

});

$scope.$on('$ionicView.beforeEnter', function() {

$scope.history = {

currency: $stateParams.currency || 'USD'

};

});

});

the history model contain currency value form $stateParams. if no currency was provider, you then default to US dollars. then you store the currencies onto $scope for the template.

changeCurrencie() method takes the value of the select box and updates the current state to use it. it calls the $state.go method, which is equivlent to ui-sref in template.

the last bock is an event listener to listen for ionic's built in navigation events. ionic has a nice feature that allow a state to be cached in memory.

use ng-change to trigger the app to navigate to another view. when user change the value of the select, value is stored on model and change view.

<script src= "history.js"></script>

app.js

.state("tabs.history", {

url: "/history?currnecy",//add currency parameter for this state

views: {

"history-tab" :{

templateUrl: "history.html",

contorller: "historycontroller"

}

}

})

currencies tab with lsit reoreder and toggle:

ionReorderButton: adding reordering to a list:

reodering can only work with the ionList directive. it works by setting a reordering state to true or false, and reordering handles appear or hide. when it shows, you can drag it, your contorller will reflect new ordering

currencies.html

<ion-view view-title= "currencies">

<ion-nav-buttons side= "primary">

<button class= "button" ng-clikc= "state.reordering= !state.reordering">reorder</button>

</ion-nav-buttons>

<ion-content>

<ion-list show-reorder= "state.reordering">//use show-reorder to declare list can be reordered

<ion-item ng-repeat= "currency in currencies">

{{currency.code}}-{{currency.text}}

<ion-reorder-button class= "ion-navion" on-reorder= "mve(currency, $fromIndex, $toIndex)"></ion-reorder-button>

//ionreorderbutton must be include anc all a method after an item is moved

</ion-item>

</ion-lsit>

</ion-content>

</ion-view>

the ionList component uses show-reorder to evaluate if the ionReorderButoon should be shown or not.

state.reordering property, which will triger the reordering to show or hide.

on-reorder method allows you to write a method that handles what to do when reordering is complete. it give you $fromIndex, $toIndex index values for the item in array to move it from and to.

currencies.js

angular.module("App").controller("currenciesController", function($scope, currencies){

$scope.currencies= currencies;

$scope.state= {

rerdering: false

};

$scope.$on("$stateChangeStart", function(){//listen for state changes and turn off reordering when leaving tab

$scope.state.reordering= false;

});

$scope.move= function(currency, formIndex, toIndex){

$scope.currencyies.splice(fromIndex, 1);

#scope.currencies.splice(toIndex, 0, currency);

}

})

you listen for $stateChangeStart event because you want to disable reordering any time the currencies tab loses focus. this event will fire when tab changed. and this is power of shared serive currencies, change order on one place will change it in all tabs.

toggle:

ionToggle component, but is just an abstraction of the CSS version, it doesn't provide any extra features.

<ion-item class= "item-toggle" ng-repeat= "currency in currencies">

{{currency.code}}-{{currency.text}}

<label class= "toggle toggle-balanced">//add item-toggle to get toggle styling

<input type= "checkbox" ng-model= "currency.selcted">

<div class= "track">

<div class= "handle"></div>

</div>

</label>

<ion-reorder-button class= "ion-navion" on-reorder= "mve(currency, $fromIndex, $toIndex)"></ion-reorder-button>

//ionreorderbutton must be include anc all a method after an item is moved

</ion-item>

you use currency.selected for each currency to filter out items that aren't enabled in other tabs. the power of shared currencies work again.

tabs are a great way to provide a navigation structure in you app. sometime you need basic tabs, sometimes you need tabs with its own history

list have ability to reordered, badges, able to include toggle

using a shared service like the currencies make it possible to share data between views.

weather app:

side menu component, paginated scrolling pane to view the weather information.

the side menu will be the basis for your app navigation. you will use a single left menu that an nagivate around app.

such as to note a favorite location. using a modal, you'll display the next year's chart of surise an sunset values.

you will use two different services for loading data in this app. forecast.io and google's geolocation services

side menus: ionSideMenus components,

inedx.html

<body ng-app= "App">

<ion-side-menus>

<ion-nav-menu-content>//declare ionSideMenu container to wrap side menu and content area

<ion-nav-bar class= 'bar-positive'>

<ion-nav-buttons side= "left">

<button class= "button button-clear" menu-toggle= "left">

<span class= "icon ion-navcion"></span>

</button>

</ion-nav-buttons>

</ion-nav-bar>

<ion-nav-view></ion-nav-view>

</ion-nav-menu-content>

<ion-side-menu side= "left">

//declare a side menu, assigning it to left

<ion-header-bar class= "bar-dark">

<h1 class= "title"> My Weather</h1>

</ion-header-bar>

<ion-content>

<ion-list>

<ion-item class= "item-icon-left" ui-sref= "search" menu-close><span class= "icon in-search"></span>

Find a city</ion-item>

<ion-item class= "item-icon-left" ui-sref= "setting" menu-close><span class= "icon ion-ios-cog"></span>

Setting</ion-item>

</ion-list>

</ion-content>

</ion-side-menu>

</ion-side-menus>

</body>

side menu need ionSideMenus, ionSideMenuContent, ionSideMenu directive. you first declare ionSideMenus, and inside it, you add two child element, ionSidemenuContent and ionSidemenu.

menu-toggle= "left". the menuTggle directive is used to take care to toggleing the side menu open or close when the button is activeated. in the side menu, item is menu-close, the menuClose directive will close any open side menu when the item is activeated. when cick find a city, it will automatic close left sidemenu

you should think ionSideMenu and ionSideMenuContent like their own views.

you can include navigation line ui-sref in the side menu

search for location:

app.js

angular,module("App", ['ionic']).config(function($stateProvider, $urlRouteProvider){

$staeProvider.state("search", {

url: "/search",

controller: "searchController",

templateUrl: "views/search.html"

});

$urlRouterProvider.otherwise("/search");

})

search.html

<ion-view view-title= "find location">

<ion-content>

<div class= "list">

<div class= "item item-input-inset">

<label class= "item-input-wrapper">

<input type= "search" ng-model= "model.term">

</label>

<button class= "button button-small btton-positive" ng-click= "search()">Submit</button>

</div>

<div class= "item" ng-repeat= "result in results" ui-sref= "weather({city: result.formatted\_address}, lat: result.geometry.locatio.lat, lng: result.geometry.location.lng})"{{result.formatted\_address}}</div

</div>

//repreat over list of results when you available to display address and link to wheather view

</ion-contetn>

</ion-view>

the first item is search box, if any result exist, it will display below it. this box uses the inset input style to give it a slightly grayed appearance.

the first list item is the search box, and then if any results exist, they'll be displayed below it. you see the link is added to the state using ui-sref. in this case, you pas the city, latitude, longitude values from the result.

<script src= "views/...search.js>

angular.module("App").contorller("searchcontroller", function($scope, $http){

$scope.model= {term: ''};

$scope.search= function(){

$http.get("http://...", {params: {address: $scope.model.term}}).success(function(response){

$scope.results= response.results;

});

};

});

search method is tapped when button is clicked.

setting view and data services:

you add a new state with controller and template for your setting view. the, to manage you app, you need two services can be used to share data and methods netween views.

create services for locations and settings

one to keep track of favoriate locations and another for setting. setting service will be just a simple javascript object, locations service will contain methods.

app.js

.factory("Setting", function(){

var Setting= {

units: "us",

days: 8

};

return Setting;

})

.factory("location", function(){

var locations= {

data: [{

city: "chicago, IL, USA",

lat: 41.8781136,

lng: -87.6297982

}],

//method to determine index value of a location

getIndex: function(item){

var index= -1;

angular.forEach(locations.data, function(location, i){

if(item.lat== location.lat&& item.lng== location.lng){

index= i;

}

});

return index;

},

//toggle method add or remove an item

toggle: function(item){

var index= locations.getIndex(item);

if(index>= 0){

locations.data.splice(index, 1);

}ele{

locations.data.push(item);

}

},

//primary method move item to top position or add it to top if new

primary: function(item){

var index= locations.getIndx(item);

if(index>= 0){

locations.data.splice(index, 1);

locations.data.splice(0, 0, item);

}else{

location.data.unshift(item);

}

}

};

return loactions;

});

any change to this service can be add to each views. you'll use locations.data as the array that stores.

show favorites in side menu list:

you have location service, you can display the lsit of favorite locatons in the side menu. To do this, you need to add a controller for your side menu so can can inject locations service into the scope, and then add a new item to the navigation list with ngRepeat to display.

app.js

.contorller("LeftMenuController", function($scope, location){

$scope.locations= locations.data;

})

update ionSide Menu and add ngController

<ion-side-menu side= "left" ng-controller= "leftMenuController">

index.html

<ion-list>

<ion-item class= "item-icon-left" ui-sref= "search" menu-close><span class= "icon -ion-search"></span>Find a city

<ion-item class= "item-divider">Fvorite</ion-item>

<ion-item class= "item-icon-left" ui-sref= "weather({city: location.city, lat: loaction.lat, lng: location.lng})"

menu-close ng-repeat= "location in locations"><span class= "icon ion-ios-location">

</span>{{location.city}}

</ion-item>

</ion-list>

loop over list of loations, links them to weather state, and display city name

when you open side menu, the default Chicago location should appear under favorties. when more locations have been added by users.

adding the setting template

your setting template will contain three primary areas: a radio list to choose betwee imperial or metric units. a range input to configure the number of dats to show in forecase. and delete

<ion-view view-ttile= "setting">

<ion-content>

<ion-list>

<ion-item class= "item-divider">unites</ion-item>

<ion-radio ng-model= "setting.units" ng-value= "'us'">imperial</ion-radio>//it will pass ng-value to unit

<ion-radio ng-model= "setting.units" ng-value= "'si'">metric</ion-radio>

<div class= item item-divider>days in forecase<span class= "badge badge-darl">{{setting.days- 1}}</span>

</div>

<div class= "item range range positive">

2 <input type= "range" name= "days" ng-model= "setting.days" min= "2" max= "8" value= "8">8

</div>

<div class= "item-button-right">favorites

<button class= "button button-small" ng-click= "candelete= !candelete">{{candelete? "done": "edit"}}</button></div>

//create a divider with a button that toggle candelete state

</ion-list>

<ion-list show-delete= "candelete">//create a list and shows delete button based on value of candelete

<ion-item ng-repeat= "location in locations">

<ion-delete-button class= "ion-minus-circled" ng-click= "remove($index)"></ion-delete-button>

{{location.city}}

//delete button only display when delete is active

</ion-item>

</ion-list>

</ion-content>

</ion-view>

<p class= "padding">Weather data powered by <a href= "http"//developer.forecase.io>Forecase.io</a>

ionRadio component is a wrapped-up radio button. wen you click radio button, the ng-value will be passed to ng-model

to use the delete feature, you use show-delete= "candelete", attribute.

setting controller

angular.module("App").controller("Settingcontroller", function($scope, setting, location){

$scope.setting= Settings;//anything change to $scope.setting will reflect on all service loader

$scope.locations= Locations.data;

$scope.candelete= false;

$scope.remove= function(index){

locations.toggle(locations.data[index]);//move to handle removing an item from list

}

})

app.js

.state("setting", {

url: "/settings",

controller: "statecontroller",

templateUrl: "views/setting/settings.html"

})

forecase.io doesn't support CORS, which mean you cannot request for data from brower, so you us CLI

declare proxt in ionic.projecy file

{

"name": "chapter6",

"app\_id": "",

"proxies": [

{

"path": "/api/forecast",

"proxyUrl": "http://api.forecast.io/forecast/your\_key"

}

]

}

weather.html

<ion-view view-titl= "{{params.city}}">

<ion-content>

<h3>current condition</h3>

<p>{{foreasr.currently.temperature|number: 0}}&deg;</p>

</ion-content>

</ion-view>

weather.js

angular.module("App").contorller("weathercontroller", function($scope, $http, $stateParams, Setting){

$scope.params= $stateParams;

$scope.settings= srttings;

$http.get("/apo/forecast/"+ $stateParams.lat+ $stateParams.lng, {params: {units: settings.units}}).success)function(forecast){

$scope.forecast= forecast;

};

});

the $stateParams are assigned to the $scope.params and used to get location name for the header bar.

app.js

.state("weather", {

url: "/weather/:city/:lat/: lng",

})

ionScroll:

you will use ionScroll to create a pagination vertical scroller. ionScroll gives you a little more controll over how the scrolling content area functions. ionScrol need to given width and height. you need to calculate the size o ionScroll based on the size of the screen.

the content area is a large container that is the size of all pages. the entire content container is the size of all three pages. ionScroll will work by creating an area three times the height of the device that will scroll and stop at each page.

you declare div eleemtn inside ionScrll with three tmes te height of ionScrll. image ionSroll was 500 pixels tall, the div inside would be 1500. it has three pages. it will always stop at boundary.

weather.html

<ion-view view-title= "{{params.city}}">

<ion-content>

<ion-scroll direction= "y" paging= "true" ng-style= "{width: getWidth(), height: getHeight()}">

<div ng-style= "{height: getTotalheight()}">// create inner div and give it a height equal to all three pages

<div class= "sroll-page page1" ng-style= "{width: getWidth(), hegiht: getHeight()}">

page1

</div>

<div class= "sroll-page page2" ng-style= "{width: getWidth(), hegiht: getHeight()}">

page2

</div>

<div class= "sroll-page page3" ng-style= "{width: getWidth(), hegiht: getHeight()}">

page3

</div>

</div>

</ion-content>

</ion-view>

weather.js

var barHeight= document.getElementByTagName("ion-header-bar")[0].clientHeight;//get first bar's height

$scope.getWidth= function(){

return window.innerWidth+ "px";

};

4scope.getTotalheight= function(){

return parseInt(parseInt($scope.getHeight()\*3))+ "px";

};

$scope.getHeight= function(){

return parseInt(window.innerHeight- barHeight)+ "px";//

};

the getHeight of div of every page is size of window minus bar height.

action sheet:

to show a list of options, the action sheet is a list of buttons that slides up from the bottom of the screen. ususally there's a cnacel button. and destructing action like deleting.

you use $ionicActionSheet service. you ned to declare the lsit of buttons, and what should happen when each button is selected

<ion-view view-title= "{{para,s.city}}">

<ion-nav-buttons side= "left">

<button class= "button button-clear" menu-toggle= "left"><span class= "icon ion-navicon"></span></button>

</ion-nav-button>

<ion-nav-buttons side= "right">//add new nav button to right side

<button class= "button button-icon" ng-click= "showOptions()"><span class= "icon ion-more"></span></button>

//add a nav button to call showOptions

</ion-nav-buttons>

</ion-view>

showOptions method will open action sheet.

weather.js

$scope.showOptions= function(){

var sheet= $ionicActionSheet.show({//use show method to set up and show an action sheet.

buttons: [

{text: "toogle favorite"},

{text: "set as primary"},

{text: "sunrise sunset"}//create three buttons

],

cancelText: "cancel",//show optional cancel button and give it text

buttonClicked: function(inedx){

if(index=== 0){

Locations.toggle($stateParams);//use locations service to toggle current as favorite

}

if(index===1){

locations.primary($stateParams);

}

if(index==2){

index.showModel();

}

return true;//close sheet after click button

}

})

}

ionModal:

you click on the third button of sheet and it will show a modal that display more information.

ionModal:dispaly the sunrise and sunset chart

a modal is a temporary view that's layered above the current view.

in this example, the modal will show the sunrise and sunset chart for the location. you'll start by using the $ionicModal service to create a new modal instance. much like popovr. and clean modal when scope is destroyed

weather.js

$scope.showModal= function(){

if($scope.modal){//if modal already exist, show it again

$scope.modal.show();

}else{

$ionicModal.fromTemplateUrl("views/weather/modal-chart.html", {

scope: $scope

}).then(function(modal){//when template loads, store modal instance on scope

$scope.modal= modal;

$scope.modal.show()

});

};

$scope.hideModal= function(){

$scope.modal.hide();

}

$scope.$on("$destroy", function(){//current view is destroyed

$scope.modal.remove();

})

};

the syntax identical to the ionPopver syntax you saw in chapter 5. modal are isolated views. meaning they need a new template. modal create an isolated child scope, and th scope parameter tell the modal which scope should be the parent for the modal. modal cannot clean itself

modal-chart.html:

<ion-modal-view>

<ion-header-bar class= "bar-dark">

<h1 class= "title">sunrise, sunset chart</h1>

<button class= "button button-clear" ng-click= "hideModal()">close</button>//since scope: $scope, you can access to weather.js method

</ion-header-bar>

<ion-content>

</ion-content>

</ion-modal-view>

ion-modal-view is a specialized version if ionView. the header bar has a close button that calls the hideModal() method. which is from parent scope(weather scope).

collection repeat; making the sunrise and sunset list fast.

if can use ng-repeat, but it will create all 365 item in a list. it will cause scroll unsmoothly

we use collection repeat feature. instead of creating 365 items, it will just 365 items, it will create just enough to display on the screen. when we scroll view, we destroy items go out view and create new items in view.

it only works for arrays of items, which you cannot have an object and use collection repeat

it will define all item size is same as the first item. images with a collection repeat can cause performance issue, so when possible, cache the images ahead of time

weather.js

$scope.showModal= function(){

if($scope.modal){//if modal already exist, show it again

$scope.modal.show();

}else{

$ionicModal.fromTemplateUrl("views/weather/modal-chart.html", {

scope: $scope

}).then(function(modal){

$scope.modal= modal;

var days= [];

var day= Date.now();

for(var i= 0; i< 365; i++){

day+= 1000\*60\*60\*24;

days.push(SunCalc.getTimes(day, $scope.params.lat, $scope.params.lng));

}

$scope.chart= days;

$scope.modal.show()

});

}

}

};

here you create an array with the times for each day of the year, starting from tomorrow for a year.

modal-chart.html

<ion-content>

<div class= "list">

<div class= "item" collection-repeat= "day in chart">

{{day.subrise| date: 'MM d'}}: {{day.subrise| date: "shortTime"}},

{{day.subset| date: "shortTime"}}

</div>

</div>

</ion-content>

popup: alert and confirm changes to favorite

ionic provide three type of popup, alert, confirm and prompt. you can add the popup into Locations service toggle method.

app.js

toggle: function(item){

var index= Locations.getIndex(item);

if(index>= 0){

$ionicPopup.confirm({

title: "are you shure",

template: "this will remove "+ Locations.data[index].city

}).then(function(res){//res will be true when OK clicked

if(res){

Locations.data.splice(index, 1);

}

});

}else{

Locatons.data.push(item);

$ionicPopup.alert({

title: "location saved"

});

}

}

advance technique for professional apps: