# Frameworks

## Angularjs

**Angularjs** also called **Angular** is *100%* ***JavaScript****, 100% client side* and compatible with both *desktop and mobile browsers*.  
Its goal is to simplify development and testing of single page applications (**SPA**) applications by providing a framework for model view controller (**MVC)** architecture, along with components commonly used in rich Internet applications (**RIA**). It’s lightweight and fast, it uses an extremely simple syntax (at least compared to **Backbone** or **Knockout** 2 popular **JavaScript** **libraries**) and like **jQuery**, it has a large number of 3rd party plugins and extensions. **Angular** comes with **jqLite** (a lightweight simplified version of **jQuery**) if there’s a need for accessing the Document Object Model (**DOM**), even if **jQuery** can still be loaded.

Data transmission model

When first loading an Angular **SPA** application a few server – client communications occurs as illustrated on figure 2.1.

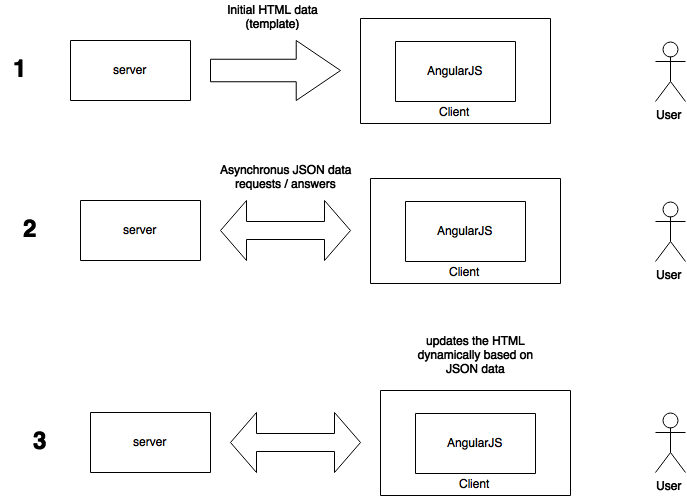


FIGURE **Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**.1 – INITIAL ANGULARJS APPLICATION DATA TRANSMISSION

This web model offers some great benefits:

* The server doesn’t have to send out the same set of HTML/css over and over again, which saves bandwidth consumption. Part of the HTML is sent when the view is changed only as explained in the figure 2.2.
* The client receives the HTML right away without any execution time preceding (usually seen with **php**) which reduces server workload.

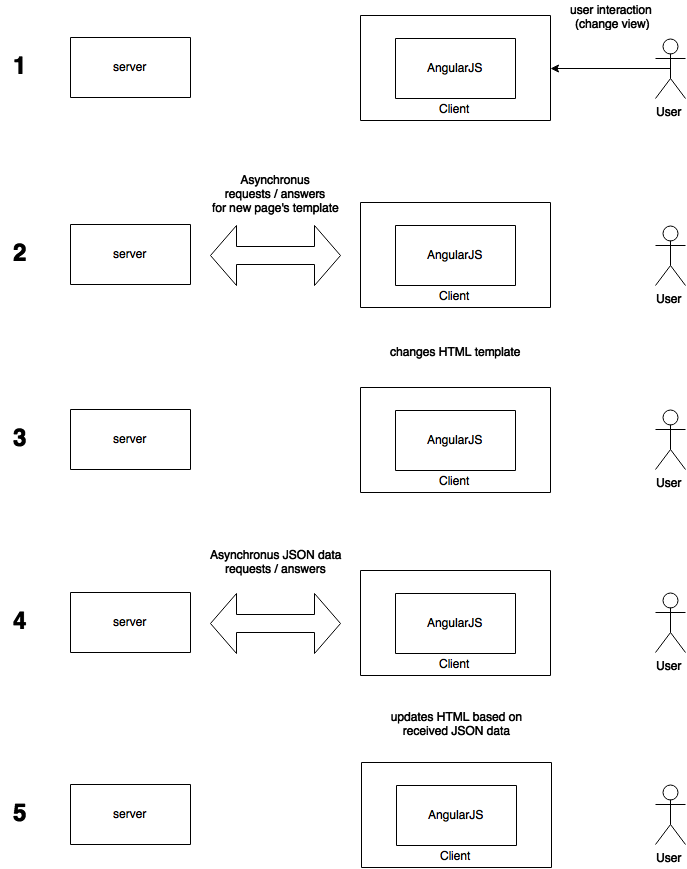


FIGURE **Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**.2 – DATA TRANSMISSION ON VIEW CHANGE

This web model fits right in the HTML5, AngularJS, and thin-serversstack (**HAT**),  
the **HAT** stack moves all the application logic in the browser letting the server handle data related operations such as sharing and storage.

Advantages

Since **Web 2.0** and asynchronous **JavaScript** and **XML** (**AJAX**) became popular, Frameworks like **jQuery** added dynamic behavior to pages.

Unfortunately, the result was a lot of duplicated logic on server and client side.  
Weaker browsers required the server to be the master thus forcing browser side logic to be written progressively as browsers improved.

This has been a trend since "Web 2.0" and AJAX applications started becoming popular. JavaScript and frameworks like jQuery added dynamic behavior to pages, but the net result was a lot of duplicate logic on the server and on the client. However, there was no resolution to this costly duplication problem- weaker browsers required that the server logic was the master and the browser enhancements had to be written to progressively enhance a server-side foundation.

In 2013, almost all users were using recent browsers with good implementation of **HTML5** and **JavaScript,** meaning developers could afford putting the client as the master and the server as it’s servant.

This meant a revolution in the application development process.  
Previously an application should consider 4 versions:

* **Web**
* **Mobile web**
* **IOS**
* **Android**

Suddenly, developers could writer browser-applications that almost all users could run on any platform or device with almost half the code of the traditional server + client duplicated logic.

**HTML5**, **responsive design** and application logic worked seamlessly together on the **web**, **mobile web** and even **IOS** / **Android** via frameworks like **Apache Cordova**.

**Angular**-like frameworks allow a clean client-side application architecture which helps building much larger applications as the mess of jQuery and CSS hacks to coordinate with server-side code is avoided. It scales way better as the server is less solicited, even better, server side code could almost be totally avoided using services such as **mongoLab** or **Firebase** (backend-as-a-service).

## Ionic

Ionic is a framework based on **Angularjs,** an open source **MVC**   
(model - view - controller) JavaScript framework maintained by **Google** and supports **Apache Cordova,** a platform for building native mobile (**Android**, **IOS** and much more…) application using **HTML, CSS** and **JavaScript.**

**Ionic** framework enables development of hybrid native mobile applications.  
Its particularities are its **Angularjs** coreand support for **SASS** (Syntactically Awesome Style Sheets) **CSS** extension.

**Ionic** is heavily optimized for touch devices, allowing touch events and gestures recognition but isn’t adapted at all for desktop use.

It comes with a lot of **CSS** components and a **JavaScript** **UI Library** that allows fast interface building and brings a feel of native application.

Gestures and events

Gestures and events are handled by **[$ionicGesture]** an **Angular** **service** implementing different methods like

* **[on(eventType,callback,$element,options)] –** add an event listener
* **[off(gesture,eventType,callback)] –** remove an event listener

The different type of events and gestures available are :

* [on-hold](http://ionicframework.com/docs/api/directive/onHold/)
* [on-tap](http://ionicframework.com/docs/api/directive/onTap/)
* [on-double-tap](http://ionicframework.com/docs/api/directive/onDoubleTap/)
* [on-touch](http://ionicframework.com/docs/api/directive/onTouch/)
* [on-release](http://ionicframework.com/docs/api/directive/onRelease/)
* [on-drag](http://ionicframework.com/docs/api/directive/onDrag/)
* [on-drag-up](http://ionicframework.com/docs/api/directive/onDragUp/)
* [on-drag-right](http://ionicframework.com/docs/api/directive/onDragRight/)
* [on-drag-down](http://ionicframework.com/docs/api/directive/onDragDown/)
* [on-drag-left](http://ionicframework.com/docs/api/directive/onDragLeft/)
* [on-swipe](http://ionicframework.com/docs/api/directive/onSwipe/)
* [on-swipe-up](http://ionicframework.com/docs/api/directive/onSwipeUp/)
* [on-swipe-right](http://ionicframework.com/docs/api/directive/onSwipeRight/)
* [on-swipe-down](http://ionicframework.com/docs/api/directive/onSwipeDown/)
* [on-swipe-left](http://ionicframework.com/docs/api/directive/onSwipeLeft/)

They are illustrated on figure 1.6.

Ui-router and application’ structure

As **Ionic** uses **ui-router,** a routing framework for **Angular** (<https://github.com/angular-ui/ui-router>)**,** the developer has to separate his application into different parts. Each part of an interface should be viewed as a block or as they call it, state.

The frame being the main level state it contains several 2nd level states that all inherits from the main level state. Those 2nd level states can themselves contain 3rd level state without any deep limit.

Each state is independent and can react differently depending on the interaction of the user.

The user navigates between main states and can interact with its children but never navigate from a child to another as they’re not main states.

Putting one secondary state into a main state is called nesting and it presents a lot of advantages, which will be detailed and compared with **AngularJS ngRoute function** below.

Just adding a dot after a state name tells ionic that the next state is a child.  
The other way to indicate that is by using the parameter parent and quoting the parent state there.

Each template (part of a page, often HTML, that can be on another file) is loaded into a <ui-view> or a <ion-nav-view> tag within the parent state template.

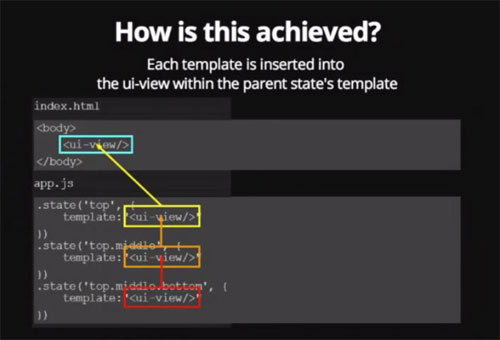


FIGURE **Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**.3 - UI-ROUTER AND UI-VIEW

How does inheritance work within nested state?

A nested state inherits the scope and the methods of its parent (thanks to **Angular**).  
To make the scope inheritance work the views must be nested exactly like the state are nested.

Child states also inherit resolved dependencies and custom data:

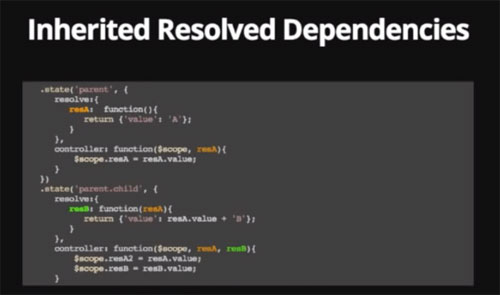


FIGURE **Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**.4 - UI-ROUTER AND INHERITED RESOLVED DEPENDENCIES

Here we can see that something resolved in the parent state is still accessible in it’s child.

**What is [resolve] ?**

Resolve can be used to provide the controller with content or data that is custom to the state. **[resolve]** is an optional map of dependencies which should be injected into the controller.

If any of these dependencies are promises, they will be resolved and converted to a value *before*the controller is instantiated and the **[$stateChangeSuccess]** event is fired.

**What about custom data ?**

.state('contacts.list', {

templateUrl: 'contacts.list.html',

data: {

customData1: 44,

customData2: "red"

}

}) ;

Simply use the property data to set those custom data specific to a state and it’s children.

It then can be accessed using this variable **[$state.current.data.customData1]**

You can change state with a custom directive **[ui-sref]**, which stands for   
state reference => smart anchors. It’s to be used instead of the usual **[href]**.

This allows changing the **url** (Uniform Ressource Locator) without changing them in the application.  
It will generate the corresponding **[href]** during compile.

**[Ui-sref] also accepts parameters**

<div ng-repeat contact in contacts>  
 <a ui-sref='contacts.detail({ id : contact.id})'>link 1</a>  
</div>

**urls with [ui-router]**

The property **[url]** exists to allow us to tie a state to a specific url.  
if state ‘sample’ has url ‘/sample’ and I create a state sample.child and define it’s url to /child then the url of the child will really be /sample/child.  
This can be avoided by using the ‘^’ before the url => ‘^/child.

**Angular** usually uses the **ngRoute** service however **ui-router** is more efficient in many aspects.

|  |  |
| --- | --- |
| **Angular with ngRoute** | **ionic with ui.router (state machine)** |
| Flat hierarchy  To get more precise details from an initial view data display a new URI needs to be created:   * /interventions {view intervention} * /interventions/details {view details intervention}   Each URI theoretically reloads the whole view (maybe some fixed content wont change but most of the dynamic part will reload). | Nesting and inheritance To get more precise details from an initial view data display a sub-state can be created:   * /interventions – {view intervention, view interventions.details}   Only loads what needs to be changed. |
| The name of a route is it’s url. | The name of a state is an actual name |
| Routes are reachable only by their **url**. | States are reachable by their name with the command $state.go(stateName). |
| Single view using ng-view (lots of reloading). | Several views nested into each other, which means more focused reloading. |
| Only populates the current view. | A state can populate any view within its hierarchy. |