**Cloud EPG with Angular JS**

Contents

[OOPS patterns 2](#_Toc426839222)

[Angular.js 2](#_Toc426839223)

[ Inheritance: 3](#_Toc426839224)

[ Polymorphism: 3](#_Toc426839225)

[ OOPs Pattern: 3](#_Toc426839226)

[How to design projects in Angular.js 3](#_Toc426839227)

[Below points may help you while breaking down project modules with respective angular.js features 3](#_Toc426839228)

[Our approach is to have the following folder structure: 4](#_Toc426839229)

[Other URLs: 5](#_Toc426839230)

[Modules: 5](#_Toc426839231)

[Services: 5](#_Toc426839232)

[Factory: 6](#_Toc426839233)

[Model – view – whatever: 6](#_Toc426839234)

[How to integrate model with controller in angular.js 6](#_Toc426839235)

[The difference between factory and service 6](#_Toc426839236)

[Reading XML data by using angular.js 6](#_Toc426839237)

[Animations: 6](#_Toc426839238)

[How to implement animations using angular.js 6](#_Toc426839239)

[UI Template: 7](#_Toc426839240)

[Create Template: 7](#_Toc426839241)

[Business logic: 7](#_Toc426839242)

[How to implement stateful modules 7](#_Toc426839243)

[How to manager Video and Audio tags by using angular.js 7](#_Toc426839244)

[Voice Driven web application 7](#_Toc426839245)

[Search functionality with Angular.js 7](#_Toc426839246)

[How to development angular.js web application in 5 min 8](#_Toc426839247)

[Directives: 8](#_Toc426839248)

[Messages or Custom Events: 8](#_Toc426839249)

[Localization: 8](#_Toc426839250)

[Build tools: 8](#_Toc426839251)

[JQuery library: 8](#_Toc426839252)

[Suggestions: 8](#_Toc426839253)

[Implementation: 9](#_Toc426839254)

[Model 9](#_Toc426839255)

[View 9](#_Toc426839256)

[Controller 9](#_Toc426839257)

[Filters 9](#_Toc426839258)

[Templates 9](#_Toc426839259)

[Page Routing 9](#_Toc426839260)

# OOPS patterns

Always OOPs patterns will solve many of our design problems. These design patterns are always like best practices for project design. While design phase if we raw materialized the required components or modules what we need for the project we can implement ground level design by using OOPs patterns.

There are couple of OOPs patter are available, but if we ask the below questions then we can use the better way of using patterns.

* What is the problem ?
* What is the use of design pattern ?
* What are all the problems / limitations with this pattern ?

## Angular.js

Even angular.js also provides you Object Oriented Programming support.

Create Module level Classes. Identify where we should use inheritance concept and where we should go for Polymorphism.

For Inheritance first identify base class which is parent class for all the classes. Create Abstract classes or interfaces to implement abstract type of components.

Define all classes with respective member variables and methods.

Example Of classes:

Application – Abstract Classs

* Show
* Hide
* Init
* onEnter
* onExit

**MainMenu** extends **Application**

Better approach is creating façade pattern to have Application class as interface for MVC pattern.

TODO: List of Classes and relation diagrams between classes

DLD

* Inheritance: <http://blog.mgechev.com/2013/12/18/inheritance-services-controllers-in-angularjs/>

<http://blog.revolunet.com/blog/2014/02/14/angularjs-services-inheritance/>

* Polymorphism: <http://stackoverflow.com/questions/21150125/angular-dynamic-polymorphic-directives>

<https://gist.github.com/jelbourn/8015527>

* OOPs Pattern: <https://github.com/mgechev/angularjs-in-patterns>

# How to design projects in Angular.js

* First understand all the features available in Angular.js
* List down all the requirements and break down them into modules or components
* Define the overall architecture of you project
* Understand the dependencies between modules. Here you can define whatever the modules are required reusable components and define that also in the architecture.
* Identified components and modules map with the angular.js features.
* Before this, make sure you are gone thought the best practices of angular.js
* In order to maintain project in best way we can look for project build tools like bower.js, grant.js and etc.

## [Below points may help you while breaking down project modules with respective angular.js features](http://stackoverflow.com/questions/25874809/large-angularjs-application-design)

* A Module could have
  + Partials
  + Controllers
  + Directive
  + Services
* Exception handling (HTTP Status code or Business errors)
* Logging (With line number, from which function)
* May also need to save the logged info in the server
* Should have the ability to turn on and off logging
* custom widgets via factory class (Reused in other modules)
* Shared Directives (isolated scope)
* Shared Modules
* Shared Utilities (sorting, filtering, etc.)
* Enumerators as per master data
* Constants via singleton
* Authentication (CSRF)
* offline storage
* REST services
* Event handling for dispatching from one module and handling it in other

## Our approach is to have the following folder structure:

* app
  + assets
    - css
    - lib js
    - images
  + common components
    - directives
    - utilities
    - authentication
    - Service proxy to hold the $resource calls
    - Enums
    - Constants
  + Model
    - entity json (example customer, product, etc.)
  + business Module A
    - Partials
    - Directives
    - Services
    - Controllers
  + business Module B
  + business Module C
  + index.html
  + Requirejs config file

## Other URLs:

<http://www.toptal.com/angular-js/a-step-by-step-guide-to-your-first-angularjs-app>

# Modules:

Angular support modular pattern to create all the sub screens as modules. If we create all the sub screens with modules we can have internal communication within application.

* Event Manager
* State Manager – Main Controller
* Main Menu
* Full Screen
* Profiles
* EPG
* Favorites
* Recordings
* VOD
* Channels
* Channel List
* Action Menu

## Services:

Services must be singleton objects . For example if event manager need to get access in each application we should make use of Event Manager as service create one instance and get the access thought out the application to control each event in proper way.

## Factory:

Work as utilities. Animation behavior we can create as factory. This factory can make use in different instances to avoid same behavior for all the screen. If you all to create that in factory we can update instance of factory service according to the current behavior

## Model – view – whatever:

We need well controlled animations for each and every screen. Even data required to bind in two way of binding. Different XML data need to be bind with each screen.

## How to integrate model with controller in angular.js

<http://joelhooks.com/blog/2013/04/24/modeling-data-and-state-in-your-angularjs-application/>

Model we should keep always as JavaScript Object. Service should be like only providing the business logic

One best example of using model data

<https://github.com/pablodenadai/angular-sync-data/tree/master/app>

<http://jonathancreamer.com/the-state-of-angularjs-controllers/>

# The difference between factory and service

<http://stackoverflow.com/questions/23074875/factory-and-service-in-angularjs>

## Reading XML data by using angular.js

<http://motyar.blogspot.in/2014/08/how-to-parse-xml-in-angular-js.html>

<http://forums.asp.net/t/2001228.aspx?Angular+JS+Loading+an+xml+file>

## Animations:

Animations Directives should be implemented for common animations

Like fade-in, fade-out with different type of behavior

## How to implement animations using angular.js

<http://www.nganimate.org/>

Here we can see how to handle enter, leave animations and css3 animations

<https://thinkster.io/angular-animations-tutorial/>

<http://www.yearofmoo.com/2013/04/animation-in-angularjs.html>

## UI Template:

This code should be good enough to scale different templates with the little modifications. We should able to split UI templates till the child level. So that we can use the same template in multiple screens.

## Create Template:

<http://tutorialzine.com/2013/08/learn-angularjs-5-examples/>

Layout creation and identify required module level controller and models

## Business logic:

This should be different from the UI template or data binding. So that we will implement functionality in business logic. Always Business logic should have functionality of each component. If functionality changes only business logic will change. Our logic should be good enough to scale business logic without effecting UI template and Model.

Adding to business logic responsibility like each Controller should not know his model source or data flow. Controller is good enough to scale the different models and views. So that we can support multiple model as well as themes also.

Dependency Injection is always good for best practice because the each component does not required to know the in-flow or out-flow inputs or outputs. So that We can depend on current business logic to make sure you are not dependent with the current object. Example Logger type which logger type need to be created inside business logic. That should get resolved by dependency injection.

* Voice command support with angular.js
* Service and Factory both will serve the same functionality.
* Factory will be used to support complex logic
* Service will be used to support simple logic

## How to implement stateful modules

<http://www.sitepoint.com/creating-stateful-modals-angularjs-angular-ui-router/>

We should use all data models as angular service or factory

## How to manager Video and Audio tags by using angular.js

<https://github.com/mrgamer/angular-media-player?utm_source=angular-js.in&utm_medium=website&utm_campaign=content-curation>

## Voice Driven web application

<https://github.com/angular-adaptive/adaptive-speech>

## Search functionality with Angular.js

<http://jsfiddle.net/r26xm/1/>

## How to development angular.js web application in 5 min

<https://www.airpair.com/angularjs/building-angularjs-app-tutorial>

## Directives:

Element : Element based animations if we want to do directive behavior so we can make use of element directive in that case

Attribute: If the particular element has the attribute called *data-module: epg* and if you want the different behavior for that particular element with the corresponding attribute. We can make use of the attribute Directive

Class – If we add Animation class to particular element we can directive with the class name for Animations fade-in, fade-out

Commenter:

Filters:

## Messages or Custom Events:

If we need to communication or send any information to other screen we can make use of Customer events to raise or send messages. Like there a update in EPG data that need to be communicate with other dependent apps so we can send custom event to register observers and update them with the new data.

## Localization:

Localization required for all the time for different regions. If we defined each UI string in JSON object and always reading from JSON object to create a screen if is easy to create localization solution

## Build tools:

If we use Bower.js or grunt.js is helpful to create a build package for web based projects. Advantage of this is will be well organized dependent 3rd party libraries, to build Production code, Easy to Integration dependency modules.

# JQuery library:

By using angular it gives most of the functionalities whatever it required for UI development. But some of them like selectors, animations related basic stuff we can make use of JQuery libraries. Like Carousal for List display with animations.

# Suggestions:

We always look for best practices or best code reviews to avoid dirty code to be in production code

We can make use of core types of objects and components:

* Value
* Factory
* Service
* Provider
* Constant

# Implementation:

Model: Here XML files data as model data. We are reading XML files and parsing as JSON object. JSON object is passing to Angular template by using Scope object. Model implementation we have done using services or factory pattern in angular’s one of the feature. First we defined the Model Object structure and try to get the data with respective Object structure. Even I have the initial Model Object with respective Model object structure.

View: Angular.js is supporting View layer with Angular Templates. Angular templates support HTML Markup code as reusable structure. Even Model and Template scope variables can be two way binding . Once model get the updated XML object we will translate to JSON and update the scope object which is referred in Templates. So that templates will auto update the data.

Controller: As of now controller is more responsible to get request from User and invoke the same request to corresponding model to get the required data.

Filters: I have created Filters for Text display, upper case, lower case, trim the unwanted string from the input string.

Templates: We have created template based on screens. According to current requirement EPG, Favorites, Recordings follows the same UI design. So those screens can be implemented with the same template.

Page Routing: Based on request I am routing to different screens. In Angular.js there is a concept called $*routeProvider*. Ng-app based one route provide will be there. While loading application we can configure route provider request configurations. All requests and corresponding redirect controller and templates will be defined in the application initial configuration time.

Key Event Handler:

Key Event handler is implemented with the help of the below URL:

<http://plnkr.co/edit/C61Gnn?p=preview>

Only key event propagation is done as of now. We have to handle all the key events depends upon the screen and current state of the screen. Event it is more over focus management need to be implemented.