[Frontend Developer Guide](http://itwiki/display/EA/Frontend+Developer+Guide)

[Skip to end of metadata](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#page-metadata-end)

* Created by [Olayinka Adetoye (Temp)](http://itwiki/display/~oadetoye), last modified by [Matthew Morgan](http://itwiki/display/~mmorgan) on [Aug 12, 2014](http://itwiki/pages/diffpagesbyversion.action?pageId=283901956&selectedPageVersions=47&selectedPageVersions=48)

[Go to start of metadata](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#page-metadata-start)

* [Introduction](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Introduction)
* [Installation](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Installation)
  + [System Requirements](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-SystemRequirements)
* [ngAAMC Application Generator](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-ngAAMCApplicationGenerator)
  + [Installing the Generator](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-InstallingtheGenerator)
  + [Updating the Generator](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-UpdatingtheGenerator)
  + [Initializing your new App](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-InitializingyournewApp)
  + [ngAAMC Sub-Generators](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-ngAAMCSub-Generators)
    - [State](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-State)
    - [Directive](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Directive)
    - [Filter](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Filter)
    - [Service](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Service)
* [Application Structure](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-ApplicationStructure)
  + [Naming Conventions and Best Practices](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-NamingConventionsandBestPractices)
* [Development Workflow and Grunt](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-DevelopmentWorkflowandGrunt)
  + [Grunt](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Grunt)
  + [Starting your Dev Server](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-StartingyourDevServer)
  + [Running Unit Tests](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-RunningUnitTests)
  + [Running End-To-End Tests](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-RunningEnd-To-EndTests)
  + [Running JSHint](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-RunningJSHint)
  + [Directives](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Directives)
  + [How to create a Directive](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-HowtocreateaDirective)
  + [When do I need a Directive?](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-WhendoIneedaDirective?)
* [Unit Tests](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-UnitTests)
  + [The Test framework](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-TheTestframework)
  + [What do I include in my tests?](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-WhatdoIincludeinmytests?)
  + [Testing element directives](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-Testingelementdirectives)
* [End-to-End Tests](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-End-to-EndTests)
  + [The Test Framework](http://itwiki/pages/viewpage.action?title=Frontend+Developer+Guide&spaceKey=EA#FrontendDeveloperGuide-TheTestFramework)

Introduction

 This document provides guidance on how to effectively develop web applications using AAMC's new User Interface Architecture. The primary objective of this project and technology stack is to empower developers to develop high-quality, stable and robust web applications based on established best practices in a consistent and predictable manner. These applications should be fully tested and provide a high level of Quality Assurance reducing and hopefully eliminating the the use of valuable developer resources in bug-fixes and thereby giving developers more time to be innovative and efficient in building new solutions

Installation

To simplify the installation process, a Yeoman Generator (AAMC Angular App Generator) has been developed to bootstrap your new application. The AAMC Angular App Generator is an interactive installer that will request some basic information about your new application. Based on the information you provide, the Generator will:

1. Create the application directory structure and intial project files
2. Install all required NodeJS modules for the project
3. Install all initial JavaScript dependencies and libraries for your project
4. Install selected AAMC AngularJS modules and configure your new app to use them
5. Write project-specific configuration files for your project

The project skeleton generated by the Generator will have the file and directories described in the **Application Structure** section below.

Before using the generator for the first time, it will need to be installed on your local machine.

**System Requirements**

* NodeJS
* Node Package Manager (NPM)
* Git
* Yeoman
  + To install: npm install -g yo

For detailed instructions on installing Node and NPM, please read [ngAAMC](http://itwiki/display/EA/ngAAMC)

For detailed instructions on using Git SCM, please read [Git](http://itwiki/display/EA/Git)

ngAAMC Application Generator

A customized yeoman generator has been created to aid in the development of AngularJS applications at AAMC. This generator will speed development by not requiring the developer to worry about repetitive tasks when creating Directives, Filters, States or Services.

**Installing the Generator**

The recommended way to use the AAMC Angular App Generator is to install it globally as a NodeJS module using NPM. This method will help keep your generator up-to-date just likie any other module installed using NPM.

To install the generator using NPM,

|  |
| --- |
| $ npm install -g git+https://gitlab.aamc.org/ngaamc/generator-ngaamc.git |

**Updating the Generator**

If you've already installed the generator and want to update to the latest version, execute the following NPM command:

|  |
| --- |
| $ npm update -g git+https://gitlab.aamc.org/ngaamc/generator-ngaamc.git |

**Initializing your new App**

To initialize your new app,

1. Create a new directory for your app
2. Change into the app directory you just created
3. Initialize a new app in that directory using the AAMC Angular App Generator

|  |
| --- |
| $ mkdir new-app    $ cd new-app    $ yo ngaamc         \_-----\_      |       |      |--(o)--|   .--------------------------.     `---------´  |    Welcome to Yeoman,    |      ( \_´U`\_ )   |   ladies and gentlemen!  |      /\_\_\_A\_\_\_\   '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'       |  ~  |     \_\_'.\_\_\_.'\_\_   ´   `  |° ´ Y `  You're using the fantastic AAMC Angular App generator.  [?] Application Name: |

Make your selections at the prompts. Pressing `Enter` or `Return` will also select the default presented for a question. Once all the information has been collected, the Generator will install all NodeJS modules and javascript packages required for your app development as well as write an initial basice configuration file.

Voila! You're ready to start developing your new app. To test your new application installed correctly,

|  |
| --- |
| $ grunt serve |

Open your web browser and navigate to: **https://<computer-name>.adm.aamc.org:8443**

**ngAAMC Sub-Generators**

The ngAAMC Yeoman generator has several sub-generators. The command syntax of the sub-generators take the form of 'yo ngaamc:<command> <name>'. Where command would be state, directive, filter, service and name is the name of the new component.

**State**

This sub-generator will create a new State, update the route file and manage all module injections.

|  |
| --- |
| $ yo ngaamc:state State1 |

**Directive**

This sub-generator will create a new Directive

|  |
| --- |
| $yo ngaamc:directive Directive1 |

**Filter**

This sub-generator will create a new Filter

|  |
| --- |
| $yo ngaamc:filter Filter1 |

**Service**

This sub-generator will create a new Service

|  |
| --- |
| $yo ngaamc:service Service1 |

Application Structure

 The application structure was designed based on and informed by recommendations in the [Best Practice Recommendations for Angular App Structure](https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/pub)and the development needs at AAMC.

|  |
| --- |
| .bowerrc  .jshintrc  Gruntfile.js  bower.json  env.json  package.json  conf/  -- aamc-cov.js  -- e2e.html  -- karma.conf.js  -- protractor-phantomjs.conf.js  -- protractor-conf.js  -- users.conf.js  src/  -- assets/  -- fonts/  -- less/  -- vendor/  -- index.html  -- app/  ---- app.js  ---- app-controller.js  ---- app-controller\_test.js  ---- main.js  ---- components/  ------ filters/  ------ services/  ------ aamc-component/  -------- aamc-component-controller.js  -------- aamc-component-controller\_test.js  -------- aamc-component-directive.js  -------- aamc-component.html  ---- states/  ------ new-state/  -------- new-state.html  -------- new-state-controller.js  -------- new-state-controller\_test.js  -------- new-state\_e2e\_test.js  -------- new-state-child-state.html  -------- new-state-child-state-controller.js  -------- new-state-child-state-controller\_test.js |

The files installed by the generator are starting points for your new application. The core generator (when you run `yo aamc-angular-app`)  initializes your app in the current directory. Once the app is initialized and you have started development, there is typically no need to use the core generator ever again. However, sub-generators are provided to help you get started with building components of your application and are described in sections below.

The main application files installed are:

* **.bowerrc -**The configuration file for bower, contains the destination path for third-party libraries installed using Bower
* **.jshintrc -**The main configuration file for JSHint
* **Gruntfile.js -**The main configuration file for Grunt, contains all the task definitions and configurations to get you started
* **package.json -**contains the list of modules and their dependencies used by the Node Package Manager (NPM)
* **bower.json -**contains the thrid-party JavaScript libraries and their dependencies used by your application
* env.json - Configuration data for your local development environment. This serves the data that will be served by the config API on deployment. Data in this file should be limited to environment-specific data only
* **karma.conf.js -** configuration file for the Karma unit tests
* **protractor.conf.js -** configuration file for the Protractor end-to-end tests
* **src -** your web application root
* **src/app -** your Angular app root - all your custom development is in this folder and its sub folders
* **src/vendor -** contains third party libraries installed using Bower
* **src/fonts -** contains all non-web fonts used in your app
* **src/less -** contains the general LESS scripts for your app. Individual, view-specific LESS scripts can also be developed  with directives or states. All LESS files are compiled into CSS for development and deployment
* **src/assets -** contains static assets for the application. Assets include images, CSS (compiled from LESS scripts), etc

Some of the files in the src/app directory are critical to the proper functioning of your app:

* **app.js -** The module definition file for your app. This is where your app is initialized. Note that your app is not automatically bootstrapped and this file only defines your app, bootstrapping happens manually in a different file
* **main.js -** manually bootstraps your app
* **app-controller.js -** The main controller for your app, helps define an application scope and also provides a place for application startup logic
* **app-controller\_test.js -** unit tests for the app controller

**Naming Conventions and Best Practices**

One of the main objectives of this project is to provide organization-wide standards for developing web applications increasing the scalability and portability of applications across the various orgs while also fostering increased collaboration, creativity and code quality. To this end, the following guidelines on file and directory naming should be followed as closely as possible

1. Use lowercase filenames with dashes separating words e.g my-sample-view.html rather than mySampleView.html
2. Show the purpose of the script in the filename with a leading hyphen. Controllers should be named my-sample-controller.js, my-sample-directive.js, my-sample-filter.js, my-sample-service.js
3. Unit tests must have the \_spec.js suffix and exist alongside the scripts they test, named directly after the script. E.g. the tests for `my-main-controller.js` would be named `my-main-controller\_spec.js`
4. End-to-end tests must have the \_e2e\_spec.js suffix
5. Precede directive names with the `aamc` prefix. This is to prevent conflicts with future additions to the HTML language. E.g. use aamc-navbar rather than just navbar
6. When creating directives, keep the directive definition, controller and template in 3 separate files. Include the tests for the directive and any special styles in the same folder named with the name of the directive  
   The folder structure for an example directive will be:  
   aamc-navbar/  
   -- aamc-navbar-controller.js  
   -- aamc-navbar-directive.js  
   -- aamc-navbar.html  
   -- aamc-navbar-controller\_spec.js
7. Place all directives in properly named directorys under **app/components**
8. Place filters in **app/components/filters**
9. Place services in **app/components/services**
10. Place states and child-states in appropriately named directories under **app/states**
11. Name files after the state. State scripts and tests are placed in the states directory eg.  
    new-state/  
    -- new-state.html  
    -- new-state-controller.js  
    -- new-state-controller\_spec.js  
    -- new-state\_e2e\_spec.js

To simplify the process of creating components and states conforming to the above best practices, sub-generators have been provided to jump-start the creation following predefined standards and patterns. More detail is provided in each appropriate section below.

Development Workflow and Grunt

**Grunt**

Grunt is a JavaScript task runner and is used actively in developing applications using Pink Elephant. For more information on Grunt, please visit the [Grunt Website](http://gruntjs.com/). In a typical application development scenario, you would perform the following tasks:

1. Start your development server
2. Write your code
3. Preview your changes in the browser
4. Run tests
5. Deploy to a staging or production server

 Appropriate Grunt tasks have been provided to assist in each of the tasks above. Some of the tasks will actually automatically perform other steps of the above process for you

**Starting your Dev Server**

 To start up your local development environment, open up a terminal or shell and type in the following command:

|  |
| --- |
| $ grunt serve |

 This command sets up a Continuous Integration development environment and performs the following tasks:

1. It compiles all LESS scripts into CSS and places the compiled CSS file in the assets foldet using the [grunt:less](http://gruntless/) task
2. It configures rewrite rules for your development server
3. It starts a mock backend ExpressJS server for mocking a REST API for development. This is not required, but is an available convenience if the API you're developing against is unavailable This uses the grunt server task
4. It starts up a small webserver to serve your new application. This typically runs at port 8443. This task and the next one both happen using the grunt [connect:dev](http://connectdev/) task
5. It opens up your browser to your local machines fully-qualified domain name and port 8443. The default browser setup in the Gruntfile is firefox.
6. It runs the unit test suite with Karma. This uses the grunt [karma:unit](http://karmaunit/) task
7. It starts the watch task. This task watches your files for changes and if any change is found, it runs jshint and [karma:unit](http://karmaunit/) This is done using grunt watch

Each of the tasks mentioned above can be run directly at the terminal at will. For example, to manually run the unit tests using karma, do the following:

### How to create a Directive

The AAMC Angular App Generator provides a sub-generator to make creating directives easy and painless. To create a directive, do the following:

|  |
| --- |
| $ yo ngaamc:directive aamc-my-directive-name |

This will perform the following tasks for you:

1. It will create a folder called aamc-my-directive-name following the best practices mentioned earlier
2. It will create a template file named aamc-my-directive-name.html
3. It will create a controller named aamc-my-directive-name-controller.js
4. It will create a unit test for the directive controller named aamc-my-directive-name-controller\_test.js

### When do I need a Directive?

Since directives are  used to encapsulate functionality in a reusable and testable manner, consider using a directive in your application if you have markup and logic that function very similarly or identically across your application. For example, if you have an application with 5 tables that all have sortable columns and all provide pagination and an export button, you should consider making that table abd all the associated functionality a directive  and using that directive in all 5 instances

Another good case for using directives is for simplifying view and state templates. In the example below, a product search results view has been simplified by using a directive to replace the individual products and their associated functionality:

Before:

products.html

|  |
| --- |
| <div id="product-catalog">    <label>Search: <input ng-model="search" type="text" /></label>    <div ng-repeat="product in products | search">      <h3>{{product.name}}</h3>      <div class="product-thumb thumbnail">        <img ng-src="{{product.image}}" class="thumb" ng-alt="{{product.name}}" />      </div>      <div class="brief-description">{{product.description}}</div>      <div class="product-offer">        <span class="product-list-price">{{product.listPrice}}</span>        <span class="product-discount">{{product-discount}}</span>        <span class="product-selling-price">{{product.sellingPrice}}</span>      </div>      <div class="call-to-action">        <button class="btn success pull-right">Buy Now!</button>        <button class="btn default pull-left">Like</button>      </div>    </div>  </div> |

After:

products.html

|  |
| --- |
| <div id="product-catalog">    <label>Search: <input ng-model="search" type="text" /></label>    <div data-aamc-product-list-item ng-repeat="product in products | search"></div>  </div> |

app/components/aamc-product-list-item/aamc-product-list-item.html

|  |
| --- |
| <div class="panel product">      <h3>{{product.name}}</h3>      <div class="product-thumb thumbnail">        <img ng-src="{{product.image}}" class="thumb" ng-alt="{{product.name}}" />      </div>      <div class="brief-description">{{product.description}}</div>      <div class="product-offer">        <span class="product-list-price">{{product.listPrice}}</span>        <span class="product-discount">{{product-discount}}</span>        <span class="product-selling-price">{{product.sellingPrice}}</span>      </div>      <div class="call-to-action">        <button class="btn success pull-right">Buy Now!</button>        <button class="btn default pull-left">Like</button>      </div>    </div> |

 app/components/aamc-product-list-item/aamc-product-list-item-directive.js

|  |
| --- |
| app.directive('productListItem', function() {    return {      restrict: 'A',      'replace': true,      'templateUrl': '/app/components/aamc-product-list-item/aamc-product-list-item.html'    };  }); |

 With the above approach, you can easily modify the layout of the view without having to worry about where the product starts and stops. It is very easy to read the view markup and see where elemts are relative to each other without having to track the nestedness of complex UI elements. It would also make managing the features of that element or widget across all implementations simple and seamless.

Directives can therefore be thought of as client-side includes if you will.

## Unit Tests

Unit tests are required for all business logic written for the application frontend. The target for applications is high level of coverage of all statements, branches and functions. Especially with a concentration on high-value business logic.

Unit tests should exist side-by-side with the javascript that is being tested.Test files must have a \_test.js suffix to be picked up by the test runner.

### The Test framework

The unit testing framework in Pink Elephant consists of the following:

* **Karma** - The test runner rns through all the test suits and provides output on test progress and stats
* **Mocha** - The test scripts are written in Mocha
* **Chai** - The assertions are written using [Chai's BDD library](http://chaijs.com/api/bdd/)
* **Sinon** - Used to spy on and observe functions, more info at <http://sinonjs.org/docs/>
* **Istanbul**- Generates the test coverage reports as HTML

### What do I include in my tests?

When writing unit tests, defining the actual extent and scope of your unit tests can  get confusing. While it is important to aim for high test coverage numbers, it is even more important to ensure that the behavior, logic and flow of your application is throughly tested. Having 100% coverage does not ncessarily indicate that a piece of your application is fully tested. While you typically do not want to run your unit tests against live data (because you want full control over your test data), you should test pieces of your application in the context or environment in which they operate. E.g. it might work better to test a controller using an Angular service with that service injected rather than with a mock of the service. Some other recommendations for writing good unit tests are:

* **Do** mock any  backend APIs you're using with $httpBackend
* **Do** assert data from the backend is assigned correctly to the neessary scope properties
* **Do** assert that functions are called using Sinon Spy
* **Do** assert the parameters passed into called functions
* **Do** use the actual custom Angular services you built in your tests. This also helps to confirm good integration of the script you're testing with that service
* **Don't** assert the functionality of services used in a script, only assert the use of the service in the context of the application unit being tested
* **Do** use good Separation of Concerns in your code design. Each function should perform one task and perform it well
* **Do** name your test cases well. If test cases are being written based on test cases provided by a QA team, title your test cases to match the wording provided by QA. If available, also use the test case numbers
* **Do** have each test case test one unit of functionality only. The test for that may involve multiple assertions though. E.g test that a function that squares number actually returns a square, and also confirm it returns a positive number
* **Don't** test the quality of data received from external sources, those should be covered by the tests for those sources.
* **Do** test failing conditions as well as passing conditions. Assert that exceptions are thrown and handled correctly. Assert that your applicatio behaves in a predictable way when failing conditions occur (e.g if an API endpoint returns a 500 error)

### [Testing element directives](http://itwiki/display/EA/Testing+element+directives)

## End-to-End Tests

 End-to-end tests are designed to test the user workflow through the complete application. As such, the tests should be written in which they mimick user activities on the application in a proper working environment (local development, dev, staging, production, etc). These tests should use actual API endpoint implementations and should replicate user behavior on the website based on test cases provided by QA.

### The Test Framework

The End-to-End test framework runs the test cases in actual browsers on an actual application URL.The End-to-End test framework in Pink Elephant consists of:

* Protractor - a wrapper around Selenium Webdriver for testing Angular Applications. Protractor is aware of the presence of AngularJS and allows selection of DOM elements by AngularJS-specific properties eg. by binding or by repeater
* Mocha - Test framework for writing the tests
* Chai-as-promised - Assertion library used by Mocha. It comes with support for promises
* AAMC-Cov Reporter - Custom test reports generator for  Pink Elephant, generates a JSON data file whick is then included in the END-to-End Test Report