

Single Page Application using AngularJS

Creating a single-page application (SPA) using AngularJS involves structuring your project, defining routes, using controllers, and implementing views. SPAs provide a more fluid and responsive user experience by loading content dynamically within a single HTML page, eliminating the need for full page reloads.

- Here's a step-by-step guide to creating a simple Single Page Application (SPA) using AngularJS, including creating a module, defining a controller, embedding AngularJS script in HTML, using `ngRoute` for routing, and navigating to different pages:

1. Setting Up Your Project:

Create a new directory for your project and set up the basic structure:

```
my-spa/  
├─ index.html  
├─ js/ |  
│   ├── angular.js |  
│   └─ app.js  
├─ views/ |  
│   ├── home.html |  
│   └─ about.html
```

2. Embedding AngularJS Script in HTML:

In your `index.html` file, include the AngularJS script using a `<script>` tag:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <meta charset="UTF-8">
  <title>My SPA with AngularJS</title>
  <script src="js/angular.js"></script>
  <script src="js/app.js"></script>
</head>
<body>
  <header>
    <a href="#/">Home</a>
    <a href="#/about">About</a>
  </header>

  <main ng-view></main>

  <footer>
    <!-- Footer content here -->
  </footer>
</body>
</html>
```

3. Creating a Module and Defining a Controller:

In your `app.js` file, define the AngularJS module and controllers:

```
angular.module('myApp', ['ngRoute'])

.controller('HomeController', function($scope) {
  $scope.message = 'Welcome to the Home Page!';
})

.controller('AboutController', function($scope) {
  $scope.message = 'This is the About Page.';
});
```

4. Using `ngRoute` for Routing:

AngularJS provides the `ngRoute` module for routing. Install it using npm or include it from a CDN:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.2/angular-
route.min.js"></script>
```

5. Configuring Routes:

Configure routes in your `app.js` file using the `$routeProvider` service:

```
.config(function($routeProvider) {  
  $routeProvider  
    .when('/', {  
      templateUrl: 'views/home.html',  
      controller: 'HomeController'  
    })  
    .when('/about', {  
      templateUrl: 'views/about.html',  
      controller: 'AboutController'  
    })  
    .otherwise({  
      redirectTo: '/'  
    });  
});
```

6. Creating Templates for Views:

Create `home.html` and `about.html` templates in the `views/` directory:

```
<!-- home.html -->  
<div>  
  <h1>{{ message }}</h1>  
</div>
```

```
<!-- about.html -->  
<div>  
  <h1>{{ message }}</h1>  
</div>
```

7. Navigating Between Pages:

Use the `<a>` tag with `ng-href` to navigate between pages:

```
<a ng-href="#/">Home</a>  
<a ng-href="#/about">About</a>
```

AngularJS Routing Capability

AngularJS provides routing capabilities through the `ngRoute` module. This module allows you to create single-page applications (SPAs) where different routes correspond to different

views/templates and controllers. Here's an overview of how routing works in AngularJS using the `ngRoute` module:

1. Setting Up `ngRoute` :

Include the `angular-route.js` script in your HTML. You can download it or use a CDN link.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.2/angular-route.min.js"></script>
```

2. Configuring Routes:

In your AngularJS module's configuration block, use the `$routeProvider` service to configure routes. Specify a template URL and a controller for each route.

```
angular.module('myApp', ['ngRoute'])
  .config(function($routeProvider) {
    $routeProvider
      .when('/', {
        templateUrl: 'views/home.html',
        controller: 'HomeController'
      })
      .when('/about', {
        templateUrl: 'views/about.html',
        controller: 'AboutController'
      })
      .when('/contact', {
        templateUrl: 'views/contact.html',
        controller: 'ContactController'
      })
      .otherwise({
        redirectTo: '/'
      });
  });
```

3. Defining Views and Controllers:

Create HTML templates for each route and define controllers for them.

```
.controller('HomeController', function($scope) {
    $scope.message = 'Welcome to the Home Page!';
})

.controller('AboutController', function($scope) {
    $scope.message = 'This is the About Page.';
})

.controller('ContactController', function($scope) {
    $scope.message = 'Contact us for more information.';
});
```

4. Using `ng-view` Directive:

In your main HTML file, use the `ng-view` directive to indicate where the content of each route should be inserted.

```
<main ng-view></main>
```

5. Navigating Between Routes:

Use the `ng-href` directive to navigate between different routes.

```
<a ng-href="#/">Home</a>
<a ng-href="#/about">About</a>
<a ng-href="#/contact">Contact</a>
```

6. Run Your Application:

Start a local server and open your application in a web browser. As you click the navigation links, the content will change without the need for full page reloads.

- AngularJS's routing capabilities, provided by the `ngRoute` module, enable you to build SPAs where different sections of your application are dynamically loaded based on the route.

Modules (Application, Controller)

In AngularJS, modules play a crucial role in organizing and structuring your application. They provide a way to encapsulate different parts of your code and define dependencies between various components. Two common types of modules in AngularJS are "Application Modules" and "Controller Modules." Let's take a closer look at each:

1. Application Modules:

An application module is the top-level module that defines your entire AngularJS application. It acts as a container for various components, such as controllers, services, directives, and more. The application module is initialized using the `angular.module` function.

Here's how you can define an application module:

```
// Creating an application module named 'myApp'  
angular.module('myApp', []);
```

In this example, the application module named `'myApp'` is created, and the empty array `[]` is used to define the module's dependencies. You can include other modules as dependencies, such as `'ngRoute'` for routing or custom modules you've created.

2. Controller Modules:

Controller modules are used to define controllers, which manage the behavior and logic of specific sections of your application's UI. Each controller module is attached to the application module as a part of its dependencies.

Here's an example of defining a controller module and attaching it to the application module:

```
// Creating a controller module named 'myControllers'  
angular.module('myControllers', [])  
    .controller('HomeController', function($scope) {  
        // Controller logic for the home page  
        $scope.message = 'Welcome to the Home Page!';  
    });
```

In this example, a controller module named `'myControllers'` is created and a controller named `'HomeController'` is defined within it. The `$scope` object is used to manage data and interactions between the view and the controller.

You can then attach the `'myControllers'` module as a dependency to the `'myApp'` application module:

```
// Attaching the 'myControllers' module to the 'myApp' application module  
angular.module('myApp', ['myControllers']);
```

Now, the `'myApp'` application module includes the `'myControllers'` module and can access the controllers defined within it.

Forms (Events, Data validation, ng-click)

In AngularJS, forms are an essential part of building dynamic and interactive web applications. AngularJS provides a set of directives and features to handle form-related tasks such as capturing user input, data validation, and handling events. Let's explore some key concepts related to forms in AngularJS:

1. Form Elements and ng-model:

AngularJS provides the `ng-model` directive to bind form elements (like input fields, checkboxes, and radio buttons) to properties in your model. This enables two-way data binding, where changes in the UI are reflected in your model and vice versa.

Example:

```
<input type="text" ng-model="username">
<input type="password" ng-model="password">
```

2. Form Submission and ng-submit:

The `ng-submit` directive allows you to specify a function to be executed when the form is submitted. This function can handle form validation, data processing, and interactions.

Example:

```
<form ng-submit="submitForm()">
  <!-- Form Input fields here -->
  <button type="submit">Submit</button>
</form>
```

```
angular.module('myApp')
  .controller('FormController', function($scope) {
    $scope.submitForm = function() {
      // Handle form submission logic here
    };
  });
```

3. Data Validation:

AngularJS provides built-in form validation features using CSS classes and directives like `ng-required`, `ng-minlength`, `ng-maxlength`, and more. You can also use CSS classes like `ng-valid`, `ng-invalid`, `ng-pristine`, and `ng-dirty` to style form elements based on their validation state.

Example:

```
<input type="email" ng-model="email" ng-required="true">
  <span ng-show="myForm.email.$error.email && myForm.email.$dirty">
    Invalid email address.
  </span>
```

4. ng-click and Form Interactions:

The `ng-click` directive allows you to bind a function to the click event of an element, such as a button. This is useful for triggering actions when a user interacts with a form.

Example:

```
<button ng-click="clearForm()">Clear Form</button>
```

```
angular.module('myApp')
  .controller('FormController', function($scope) {
    $scope.clearForm = function() {
      // Reset form data and validation
      $scope.email = '';
      $scope.password = '';
      $scope.myForm.$setPristine();
    };
  });
```

AngularJS's form-related directives and features make it easier to create interactive forms with dynamic validation and smooth user interactions.