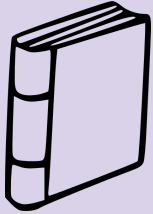# Coding & Cocktails Session 14:
Intro to Single Page Applications
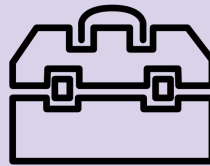
a

Overview

Basics of how to create a Single Page Application with Angular2.

Don't know a term? **Check out our glossary** bit.ly/CnCgloss	**First time at C&C ?** **Install recommended tools** bit.ly/CnCTools

---

Project

Single page applications provide a more native app-like experience to users. They can also be a lot of fun to develop. There are a lot of frameworks available to make this process easier, but choosing one and learning how to use it can be overwhelming. Angular 2 is very opinionated, which means you have less architecture decisions to make than some of the other frameworks. This also means there will be more consistency among Angular 2 projects, so if you've worked on one, you can easily jump into a different team's project.

In this session, we will become familiar with how to use Angular 2 to create a single page trivia app.

Open Slack! At home or setting-up before the session starts? Open Slack. Mentors are available to help & we post important updates to the whole group! https://kcwit.slack.com	**Check your work** Compare your work to the final... Project Files: http://bit.ly/spa-project & App: http://bit.ly/spa-cnc

Part 0A: Set-up | Verify NodeJS & npm is installed & Install Angular CLI

In order to use Angular CLI later in this lesson, you have to install it via npm, which requires NodeJS. (We won't be writing a NodeJS application, but the two are essentially installed together.)

1. Open your terminal / command line

2. In Git Bash (windows) or iTerm2 (macs), type: **node --version && npm --version**
3. If you get version numbers, then NodeJS and NPM are already installed. Proceed to Step 4.

4. In Git Bash (windows) or iTerm2 (macs), type: **npm install -g angular-cli**
5. To confirm Angular CLI was installed, while still in your Command Line tool, type: **ng**
   If you get version numbers, you're done with setup!

Part 0B: Setup | Add TypeScript Package to Sublime Editor

Using TypeScript with Angular2, provides us a lot of shortcuts. However, to make sure the correct words highlight for us in Sublime (remember the pretty colors you've seen in HTML & CSS files?), and that we get some of the autosuggest features of TypeScript, we have to install the TypeScript package for Sublime.

1. Open Sublime Text Editor and hold down these 3 keys to open the Command Palette:
   Mac: `cmd + shift + p` | Windows: `ctrl + shift + p`
2. In the Command Palette input field, type: **install**
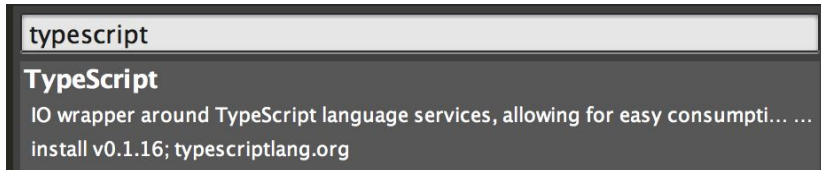3. Find and select: **Package Control: Install Package**

```
install
Package Control: Install Package
```

4. The Package Manager pops open another input field. In this one type: **TypeScript**
5. Find & select the TypeScript package that matches this description:

```
typescript

TypeScript
IO wrapper around TypeScript language services, allowing for easy consumpti... ...
install v0.1.16; typescriptlang.org
```
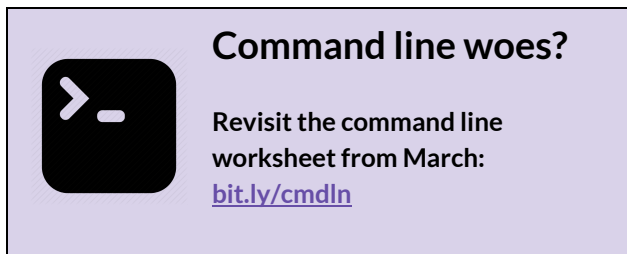
6. Setup is done! Give high fives to those around you.

Part 1: Starting our Single Page App

Let's start with creating our app's foundation. Open Git Bash (windows) or iTerm2 (macs) and rock & roll.

1. Navigate to your CodingAndCocktails folder: **cd <your home directory>/CodingAndCocktails**

2. Use Angular CLI to scaffold the project for us. Type: **ng new trivia**
3. Change the directory to the new one Angular CLI created for us. Type: **cd trivia**
4. Let's run our new app. Type: **ng serve**
5. In Chrome, visit the URL http://localhost:4200 to see your working (basic) app.

6. Let's make some updates to the app to see the live reload we get from **ng serve**! Open your *trivia* folder in SublimeText.
7. Open the *src/app/app.component.ts* file in SublimeText.
8. Edit the title variable to whatever text you want (like: *have a cocktail!*)
   Note: This *title* will end up being the title of your trivia app.
   See: http://bit.ly/spa-1a

```
◀ ▶   app.component.ts   ✕
1    import { Component } from '@angular/core';
2
3    @Component({
4        selector: 'app-root',
5        templateUrl: './app.component.html',
6        styleUrls: ['./app.component.css']
7    })
8    export class AppComponent {
9        title = 'app works!';
10   }
11
```

9. Save your file & go back to the tab in Chrome that has your app running.
   You should see your updated text!
   See: http://bit.ly/spa-title

   ⓘ localhost:4200

   ## app works!

   Nice work!

   Take a break and grab another drink. You've earned it!

Part 2: Creating a component

Our app works, but it's not very exciting. Let's add some components!

1. Start in your CLI tool and generate a component named *quiz* with Angular CLI.
   Type: **ng g component quiz**

   > **Helpful tip:**
   > ● This creates a *src/app/quiz* directory with 4 files in it: *html, css, typescript & spec*
   >   *spec* files are to write tests (to make sure your code works). We won't be using these tonight, so you can ignore this file.

2. Let's add some styles to our app, so it won't look bland as we develop it.
   a. Copy the CSS styles here [http://bit.ly/spa-css-a] & paste into your *src/styles.css* file.
      These are global styles that apply to the whole app.
   b. Copy the CSS styles here [http://bit.ly/spa-css-b] & paste into your *src/app/app.component.css* file.
      These styles apply only to the markup in *app.component.html*.

c. Copy the CSS styles here [http://bit.ly/spa-css-c] & paste into *src/app/quiz/quiz.component.css*. These styles apply only to the markup in *quiz.component.html*.

> ### What is this HTML & CSS stuff?
> In an effort to keep the project focused on SPA's, we're providing the HTML & CSS code for you to copy/paste. If you have questions about any of it, or HTML/CSS in general, grab a mentor!

3. Now that we have styles, let's add the HTML markup for our templates.
   a. Copy the HTML here [http://bit.ly/spa-fonts] & paste into your *src/index.html* file, above the closing **</head>** tag.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">
<link href="//fonts.googleapis.com/css?family=Playfair+Display|Raleway" rel="stylesheet">
</head>
```

This adds some Google Web Fonts, so we have some pretty fonts in our app.

   b. Let's add our *quiz* component html to our *app* component.
      Copy the HTML here [http://bit.ly/spa-html-a] & paste into *src/app/app.component.html*.

```
app.component.html ×
1  <div class="quiz-wrapper">
2    <h1>{{title}}</h1>
3    <app-quiz></app-quiz>
4  </div>
5  |
```
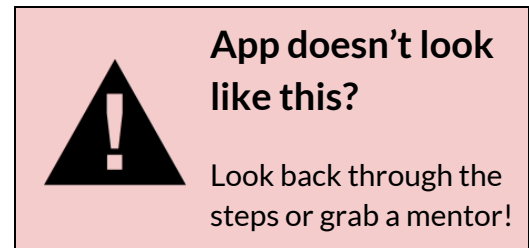
Helpful tip:
- **<app-quiz></app-quiz>** is the HTML selector for our *quiz* component. So this tag will be replaced (automagically) by the HTML in our *src/app/quiz/quiz.component.html* file.
- Where does **<app-quiz>** come from? In the Component metadata in our *src/app/quiz/quiz.component.ts* file, the *selector* is set to **app-quiz**.

```
@Component({
  selector: 'app-quiz',
  templateUrl: './quiz.component.html',
  styleUrls: ['./quiz.component.css']
})
```

   c. Copy the HTML here [http://bit.ly/spa-html3c] & paste into *src/app/quiz/quiz.component.html*.

```
quiz.component.html ×
1  <div class="quiz-wrapper">
2
3    <div class="quiz">
4
5      <div class="each-question">
6        <div class="q-and-a">
7          <h3 class="question">question here</h3>
8
9          <ul class="answers">
10            <li>answer here</li>
11         </ul>
12       </div>
13     </div>
14
15   </div>
16
17 </div>
18
```

d. Head on over to Chrome to check out how your app looks.

App doesn't look like this?

Look back through the steps or grab a mentor!
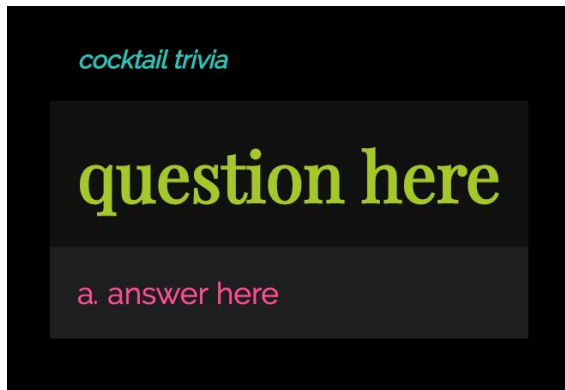
Part 3A: Adding Data to Our Template

Our template is working and looks pretty, but it's hardcoded. Let's add a couple questions and answers to our component, and make the template render our quiz questions dynamically. In *Part A*, we'll do some set-up work for data to use in our template. *Part B* will be to integrate the data with our component's template (HTML).

Later, we'll be using an API to provide us the questions and answers for our quiz. In order to make that transition go smoothly, we're going to set-up our data in the same format that will come from our API.

1. Open your *src/app/quiz/quiz.component.ts* file. In the **ngOnInit()** method, between the **{}**, paste the code from here [http://bit.ly/spa-qa].
    Your file should look like the one here [http://bit.ly/spa-3_1].
2. Since we're using TypeScript, we need to set types for the data that we just copied and pasted. This is one file *Angular CLI* won't generate for us.
    Create a new file in *src/app/quiz* & name it **quiz.model.ts**.
3. In *quiz.model.ts*, add the code found here [http://bit.ly/spa-model]. Feel free to manually type it out or copy/paste.

> ### What does this code do?
> ● We're defining an *Answer* & a *Question* Model, and we're **export**-ing them, so we can **import** them into the file where we want to use them.
> ● Our **Answer** Model has 2 properties: *correct* & *text*.
>   The *type* of **correct** is a *boolean* (true/false) & **text** is a *string*.
>   ```
>   export class Answer {
>     correct: boolean;
>     text: string;
>   }
>   ```
> ● Our **Question** Model has 2 properties: *text* & *answers*.
>   The *type* of **text** is a *string* & **answers** is an *array* ([]) of our *Answer* Model.
>   ```
>   export class Question {
>     text: string;
>     answers: Answer[];
>   }
>   ```

4. Now we're going to import these into *quiz.component.ts*.
    a. Open *src/app/quiz/quiz.component.ts* & below the *import* at the top, add: **import {} from './quiz.model';**

b. Place your cursor in the middle of {}. And type **A**.
Sublime should give you an autocomplete suggestion of **Answer**. [Hit *tab* or *enter/return* key for it to autocomplete **Answer** for you.
c. Add a comma after *Answer* & repeat process by typing **Q** (*Question*).
See: http://bit.ly/spa-ts

```
quiz.component.ts    ×
1   import { Component, OnInit } from '@angular/core';
2   import {} from './quiz.model';
3
4   @Component({
5     selector: 'app-quiz',
6     templateUrl: './quiz.component.html',
7     styleUrls: ['./quiz.component.css']
8   })
```

5. In the *quiz.component.ts* file, within the *OnInit* method, add: **questions: Question[];**

```
@Component({
  selector: 'app-quiz',
  templateUrl: './quiz.component.html',
  styleUrls: ['./quiz.component.css']
})
export class QuizComponent implements OnInit {
  questions: Question[];

  constructor() { }

  ngOnInit() {
```

This uses the imported *Question* Model to set the *type* for *questions*, (the data we set earlier in *ngOnInit*).

Part 3B: Adding Data to Our Template

Now that we have our data ready, let's wire it up to our component's *template*. Open *src/app/quiz/quiz.component.html* & let's get to it.

1. In <div class="quiz-wrapper">, at the top of the file, add an *nglf* attribute:
   ***ngIf="questions"**
   See: http://bit.ly/spa-if1

```
quiz.component.html    ×
1   <div class="quiz-wrapper">
```

   nglf* is an Angular *attribute directive* that will only render our template **IF *this.questions* is defined in our component.
2. In the markup for **<div class="each-question">**, add the following as an attribute:
   ***ngFor="let question of questions; let i = index;"**

See:

```
quiz.component.html  ×
1   <div class="quiz-wrapper">
2
3     <div class="quiz">
4
5       <div class="each-question">
6         <div class="q-and-a">
7           <h3 class="question">question here</h3>
8
9           <ul class="answers">
10             <li>answer here</li>
11           </ul>
12         </div>
13       </div>
14
15     </div>
16
17   </div>
18
```

What does this code do?

- This is an *ngFor directive, that will repeat the markup (including the div.each-question), for every question in our component's this.questions.)
- We're also setting a variable named i to index. This helps us keep track of what question is currently being rendered in the template, by giving us the index of that question. Question 1 is index 0. Question 2 is index 1. This will come into play later in the tutorial.
- To help make the code easier to read, I start each attribute on it's own line [see the gif above to see that in action].

3. Replace **question here** with **{{question.text}}**

```
<h3 class="question">{{question.text}}</h3>
```

This will use the *question's text* from our data to populate our template.

4. Let's loop through our *answers*. In the **<li>** tag, add the following *ngFor* attribute:
   *ngFor="let answer of question.answers"
   See:

```
<ul class="answers">
  <li>answer here</li>
</ul>
</div>
div>
```

5. Replace **answer here** with **{{answer.text}}**

```
<ul class="answers">
  <li
    *ngFor="let answer of question.answers">{{answer.text}}</li>
</ul>
```

6. We now have our template rendering questions and answers, but nothing happens when a user selects an answer. Let's ad="ond a *click* event so we can start tracking this.

a. In that same **<li>** tag, add a *(click)* attribute:
   (click)="onSelect(answer)"

```html
<ul class="answers">
  <li
    *ngFor="let answer of question.answers"
    (click)="onSelect(answer)">{{answer.text}}</li>
</ul>
```

> ## What does this code do?
> - When a user clicks on the **<li>** (*answer*), it executes a method in the Component's TypeScript file, named *onSelect*, and passes the *answer* data thru to the method. [We don't have the *onSelect* method defined in our quiz.component.ts file, so we'll do that next.]

b. Open *src/app/quiz/quiz.component.ts*.
   Copy the code here [http://bit.ly/spa-select] & paste after the **ngOnInit() {}** method.

```typescript
onSelect(answer: Answer){
  if(answer.correct) {
    console.log('answer correct');
  }
  else {
    console.log('answer wrong');
  }
}
```

> ## What does this code do?
> - We're declaring that the *answer* parameter is type *Answer*
> - If the *answer* is *correct*, we log **answer correct** to the dev console. Otherwise, we log **answer wrong**.

Your *src/app/quiz/quiz.component.ts* file should look like the one here [http://bit.ly/spa-6_b].

Part 4: Tally Correct Answers & Show Results

We now have questions and answers to create a quiz, but at the end of the quiz, the user won't know how many they answered correctly. Let's add a counter that keeps track of the correct answers throughout a quiz and displays the results at the end!

1. Copy the code here [http://bit.ly/spa-4_1] & paste into your *src/app/quiz/quiz.component.html* before the final **</div>** tag.

```html
<div class="results" *ngIf="quizIsOver">
  <div class="result-message">
    You answered {{correctAnswers}} out of {{questions.length}} questions correctly.
  </div>
  <div class="score">That's {{correctAnswers / questions.length * 100}}%</div>
  <div class="result-action">Grab a cocktail &amp; celebrate!</div>
</div>
```

> ## What does this code do?
> - *div.results*: only shows *if* the *quizIsOver*
> - *div.result-message*: tells the user how many *correctAnswers* they had out of the *total # of questions*
> - *div.score*: calculates & displays *percentage* correct
> - *div.result-action*: displays a fun message

2. Open *src/app/quiz/quiz.component.ts.*
   a. Below **questions: Question[];** let's add 3 new variables and their types:
      correctAnswers: number;
      currentQuestionIndex: number;
      quizIsOver: boolean;

```
export class QuizComponent implements OnInit {
  questions: Question[];
  correctAnswers: number;
  currentQuestionIndex: number;
  quizIsOver: boolean;
```

   b. Then in *ngOnInit,* before this.questions, let's set default values for these 3 variables:
      this.correctAnswers = 0;
      this.currentQuestionIndex = 0;
      this.quizIsOver = false;

```
15    constructor() { }
16
17    ngOnInit() {
18      this.correctAnswers = 0;
19      this.currentQuestionIndex = 0;
20      this.quizIsOver = false;
21      this.questions = [
22        {
```

   c. We need to increment our *correctAnswers* variable, every time an answer is correct.
      In the *onSelect* method, within the **if(answer.correct) {...}** add:
      this.correctAnswers ++;

```
if(answer.correct) {
  this.correctAnswers ++;
  console.log('answer correct');
}
```

   This will update the value of *correctAnswers + 1*, when the *answer* is *correct*.
   d. We need to increment *currentQuestionIndex*, every time an answer is selected (*correct or not*).
      Below *onSelect*'s **else {...}** add:
      this.currentQuestionIndex ++;

```
onSelect(answer: Answer){
  if(answer.correct) {
    this.correctAnswers ++;
    console.log('answer correct');
  }
  else {
    console.log('answer wrong');
  }
  this.currentQuestionIndex ++;
}
```

   e. Below this.current add:
      if (this.currentQuestionIndex === this.questions.length) {
        this.quizIsOver = true;
      }

```
    this.currentQuestionIndex ++;

    if (this.currentQuestionIndex === this.questions.length) {
        this.quizIsOver = true;
    }
}
```

When the *currentQuestionIndex* value is the same as the *questions.length* value, **quizIsOver** is set to true.

3. Right now, you see all the quiz questions at once. And even when the results display, the questions/answers are still visible. Let's add an *ngIf*, so we only see 1 question at a time.
   a. Open *src/app/quiz/quiz.component.html*. In **<div class="q-and-a">**, add attribute:
      *ngIf="currentQuestionIndex === i"
      See: http://bit.ly/spa-if2

```
<div
  class="each-question"
  *ngFor="let question of questions; let i = index;">
  <div class="q-and-a">
    <h3 class="question">{{question.text}}</h3>

    <ul class="answers">
      <li
        *ngFor="let answer of question.answers"
        (click)="onSelect(answer)">{{answer.text}}</li>
    </ul>
  </div>
</div>
```

4. **BONUS**: Add a tracker to the top of the quiz that tells the user which question they're viewing.
   a. In *src/app/quiz/quiz.component.html*, before *div.quiz* but after opening *div.quiz-wrapper* tag, paste the code from here [http://bit.ly/spa-track].

```
<div class="quiz-wrapper" *ngIf="questions">
  <div
    class="question-tracker"
    *ngIf="!quizIsOver">Question {{currentQuestionIndex + 1}} of {{questions.length}}</div>
  <div class="results-header" *ngIf="quizIsOver">Results</div>

  <div class="quiz">
```

Challenge: Can you explain to your neighbor what this is doing?

Nice work!

**Check out your app in Chrome.
Take a break and grab another
drink.
You've earned it!**

Part 5: Creating a Service & Fetching Data

We have our component set-up to display trivia questions and answers, but we only have a couple hardcoded questions & answers. Let's use an API to populate our quiz questions and possible answers.

1. First, use Angular CLI to generate a service named *quiz* within the *quiz* directory. Open your CLI tool.
   Type: **ng g service quiz/quiz**

Helpful tip:

2. Open *src/app/quiz/quiz.service.ts*. We need to import some methods from the HTTP Module & RxJS [Angular CLI installed both of these for us so we can simply import to use them]. Add these imports below the *@angular/core* import:
   import { Http, Response } from '@angular/http';
   import 'rxjs/add/operator/map';
3. In the parenthesis for constructor(), add:
   private http: Http

```
 6   export class QuizService {
 7
 8      constructor(private http: Http) { }
 9
10   }
```

   This creates an instance of the *Http* service that we imported and assigns it to *http*. It's private because we don't want to access it from outside the *QuizService* class.
4. Now, we're going to add our API request to **//cocktail-trivia-api.herokuapp.com/api/sample**.
   Copy the code from here [http://bit.ly/spa-http] & paste below **constructor(private http: Http) { }**

What does this code do?

- We create a method named *getQuestions*, that makes the *http* request to our API URL.
- We use the *map* method we imported from *RxJS* to grab the *json* from the response (which we set to *res* of type *Response*). The *json* response is our questions and answers.

5. Open *src/app/quiz/quiz.component.ts*.
   a. We need to import the *QuizService*, so that we can use it. Add the following to the list of your other imports: **import { QuizService } from './quiz.service';**
   b. In our *Component metadata*, we need to add *QuizService* as a *Provider*.
      Add a comma & a new line after **styleUrls: ['./quiz.component.css']** and add:
      providers: [QuizService]

```
quiz.component.ts

 1   import { Component, OnInit } from '@angular/core';
 2   import {Answer, Question} from './quiz.model';
 3   import { QuizService } from './quiz.service';
 4
 5   @Component({
 6      selector: 'app-quiz',
 7      templateUrl: './quiz.component.html',
 8      styleUrls: ['./quiz.component.css'],
 9      providers: [QuizService]
10   })
```

   c. In the parenthesis for **constructor() { }** add:
      private

```
11   export class QuizComponent implements OnInit {
12      questions: Question[];
13      correctAnswers: number;
14      currentQuestionIndex: number;
15      quizIsOver: boolean;
16
17      constructor(private quizService: QuizService) { }
18
19      ngOnInit() {
20         this.correctAnswers = 0;
```

   Now, we can access the *QuizService* methods via *this.quizService*.

d. Replace **this.questions = […];** with:
   **this.quizService.getQuestions()
      .subscribe(questions => this.questions = questions);**

```
19    ngOnInit() {
20      this.correctAnswers = 0;
21      this.currentQuestionIndex = 0;
22      this.quizIsOver = false;
23      this.quizService.getQuestions()
24        .subscribe(questions => this.questions = questions);
25    }
26
27    onSelect(answer: Answer){
```

> ## What does this code do?
> - This *subscribes* to the *getQuestions* method we defined in *QuizService*, since HTTP requests can take a little or a long time to return us a response.
> - When the HTTP request comes back, the *getQuestions* method returns the data to our subscriber & we set that data to our component's *this.questions*, so we can use it in our template.

Homework

The more you practice, the better you'll get. Reinforce what you've learned tonight with the following tutorial.

Hey Slacker!

Remember, we're here to help. Join the KCWiT codingandcocktails Slack Channel: **kcwit.slack.com**

Don't remember git?

See the git version control worksheet from April: **bit.ly/CnCvers**

Part 1: Git - Deploy App to GitHub Pages

Angular CLI initialized a git repository for you in your source folder. Open your CLI tool & check git by running **git status**. Angular CLI provides us a command to deploy our app to GitHub Pages.

1. Add all your files to git. Type: **git add .**

```
[~/C/t/trivia [master] > git add .
```

2. Commit all your files. Type: **git commit -m "Add your own message here"**

```
~/C/t/trivia [master] > git commit -m "trivia app"
```

3. Deploy your app to GitHub Pages.
   Type: **ng github-pages:deploy --message "Add your own message here"**

```
~/C/t/trivia [master] > ng github-pages:deploy --message "deploy to pages"
```

   It may prompt you to generate a token on GitHub. Follow the instructions as provided.

4. **View your App from Chrome or from your phone!**
   When the deployment to GitHub Pages is completed, it will provide the URL for your app.
   Format of the URL should be something like https://username.github.io/trivia/

Part 2: Add a router

It is recommended to watch this video [http://bit.ly/router-video] (*sign-up for free account to view*) or read this article [http://bit.ly/angular2-routing] to see how routers work in Angular 2 & how to configure them.

1. Create a new component named about
2. Create a new file in *src/app* and name it **app.routes.ts**
3. In this new file, we need to import a couple things from the Router module.
   Add: **import { Routes, RouterModule } from '@angular/router';**
4. Now we need to set our variable that will hold our routes. Add: **const routes: Routes = [];**
   This sets a *constant* variable named **routes** [of imported type *Routes*] to an empty array. We'll add our routes to this array.
5. In your array, add 2 routes:
   a. **''** that loads our *QuizComponent* [this is the root or homepage path '/']
   b. **'about'** that loads our new *AboutComponent* [this will load for the path **'/about'**]

> ## Helpful Tip:
> - The format for a route looks like this [replace the italicized words with your own]:
>   **{ path: *'urlpath'*, component: My*Component* }**
> - When defining a route, don't include the `` `/` `` at the beginning

6. We need to **import** our 2 components, so add those 2 import statements to the top of the file.
7. Now we need to export our routes, so at the bottom of *app.routes.ts*, add this:
   export const QuizAppRoutes = RouterModule.forRoot(routes);
8. Open the *src/app/app.module.ts* file & add an import. Import **QuizAppRoutes** from *app.routes.ts* [hint: that needs minor modifications to make that the actual import statement.]
9. We also need to add **QuizAppRoutes** in this file's *@NgModule imports*.
10. Last, but not least, we need to add set where our router will display the relevant component.
    In *src/app/app.component.html*, replace **<app-quiz></app-quiz>** with **<router-outlet></router-outlet>**
11. Visit your app in Chrome to checkout your routes! http://localhost:4200/ & http://localhost:4200/about

> ## Check your work
> **Compare your homework to the files here:**
> http://bit.ly/spa-homework