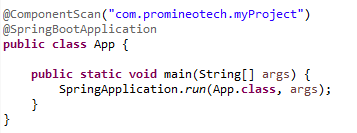
Lesson 14

* Maven – As the applications we are building start to grow in complexity, there are many features that have already been built by others that we can reuse instead of reinventing the wheel. These preexisting features or libraries can be downloaded and added to our classpath for use in our projects. Manually downloading these dependencies can introduce an unwelcome overhead because each developer who works on the project, and each server the project is deployed on, will have to ensure that all dependencies are manually downloaded, or they won’t be able to compile or run the application. Maven is an Apache tool that allows us to list the dependencies our project needs and then it handles the downloading on each computer. This makes managing dependencies as simple as everyone just having a copy of the file where the dependencies are listed, and we can put that file in our project git repository so that everyone has it when they clone the project to start working on it.
  + Maven also helps manage our build and some additional plugins. However, we will mostly be focusing on the dependency management it provides.
  + To start using Maven, we need to download the tool and add it to our Path variable. To do this, we’ll go to <https://maven.apache.org/download.cgi> and download the binary zip.
  + Once downloaded, let’s extract the files to a directory that makes sense. For me, I extracted the files to C:\Program Files\Apache
  + Next, let’s add this directory to our path variable by….
  + To test that it was successfully added, open command prompt and type in mvn space dash dash v and hit enter. You should see this. If you get a message saying mvn is not a recognized command, it was not successfully added to your path variable and you need to go back over the steps and troubleshoot or reach out for help.
  + Now, let’s create a new maven project in Eclipse….
  + The group id will make the base of your packages, I will use com.promineotech
  + The artifact id is the name of the project
  + Once you click finish, you’ll see the pom.xml file. This is where maven manages our dependencies. And if you open it and look at the pom.xml tab, you can see we have one default dependency for our junit library that we can use for unit tests.
  + As we need more external libraries, this is where we will list them, and maven will pull the necessary files for us so that we don’t have to manually manage them.
* To build our web APIs, we will use the SpringBoot framework. This comes with a bunch of premade classes and annotations that will allow us to create web APIs without having to reinvent the wheel. SpringBoot is heavily used in the industry, so the skillset to use it is transferable.
  + To start, we need to add some spring dependencies to our project’s pom.xml file….
  + We will also need to add a parent element outside of the dependencies for our newly list dependencies to pull the version from….
  + Now, in our command prompt, let’s navigate to our project root directory and run mvn clean install. You should see it install successfully and you will see it downloading some of the dependencies we listed.
  + Let’s go back to the project and we can see the jars for the dependencies we added are in our project under Maven Dependencies.
  + Now we can start using those jars’ code in our code!
* We can now start building applications using our SpringBoot dependencies! To start, let’s modify the main method in our App class.
  + …
  + The componentscan and springbootapplication pieces of code, preceded by the at symbol, may look different than what you’ve seen so far. These are called annotations. And they provide meta data to our application to help with some configuration behind-the-scenes. The ComponentScan tells our application what the base package is to scan for components to include in our web API, which we will add a few in a minute. Because I chose com.promineotech.myProject, SpringBoot knows to look for all components we create that are in this package, or any subpackage.
  + The SpringBootApplication does a few things behind the scenes, but most importantly, it tells springboot that this is where our application starts.
  + Using command prompt, let’s navigate to our project directory and run mvn spring-boot:run
  + This will start the application, and if we have no errors will pause after a while, at which point we are running a local SpingBoot server. Now, there is much we can do with this server yet as we haven’t built out the API endpoints, but now that we have the base of our application working, we can start adding our endpoints!
  + To do this, we will create a new sub package called controllers. In this package we will create the controller layer of our application, which houses the endpoints for our API. This layer will be where we accept incoming HTTP requests and return responses.
  + Inside this package, let’s create a new class called FriendsController. We will set up an endpoint in here that allows us to send HTTP requests to add new friends to a list of friends, and view the already added friends as well.
  + Add the RestController annotation right above the class declaration. This annotation will tell SpringBoot that this a component to add to the endpoints that will be made available when the server is running.
  + Next, let’s create a list of string to hold friend names, and two methods. One to return the list of names and the other to add a new name to the list.
  + Now, let’s add the RequestMapping annotation above each method. This annotation will allow us to map a URI or endpoint to each method. That means that if an HTTP request comes in to our API server, and matches any of the mappings we set up, the method annotated by that specific mapping will be executed.
  + Notice that for the addFriend method we explicitly say that the HTTP request must be a POST request.
  + On the getFriends method, we don’t explicitly point out what HTTP verb we are listening for, this will listen for any verb method, but if it’s a POST it will go to our other method because it is more specific and takes precedence.
  + Also notice that when we don’t have more than one attributes in the parenthesis of our RequestMapping annotation we don’t have to specify the value attribute, the default String we use is implicitly assigned to the value attribute.
  + We have one more annotation we need to add. On our addFriend method, we need to add the @RequestBody annotation to our parameter. This tells SpringBoot that the HTTP request payload will be the same data type that our parameter is, which in this case is a String.
  + Now, if we restart our server with mvn spring-boot:run, we should see these endpoints come up in the startup logs, then we can open postman and test them out.
  + By default, the server will run at localhost port 8080, so that’s where we’ll send our requests.
  + …. Talk through the postman process and mention that data doesn’t persist right now