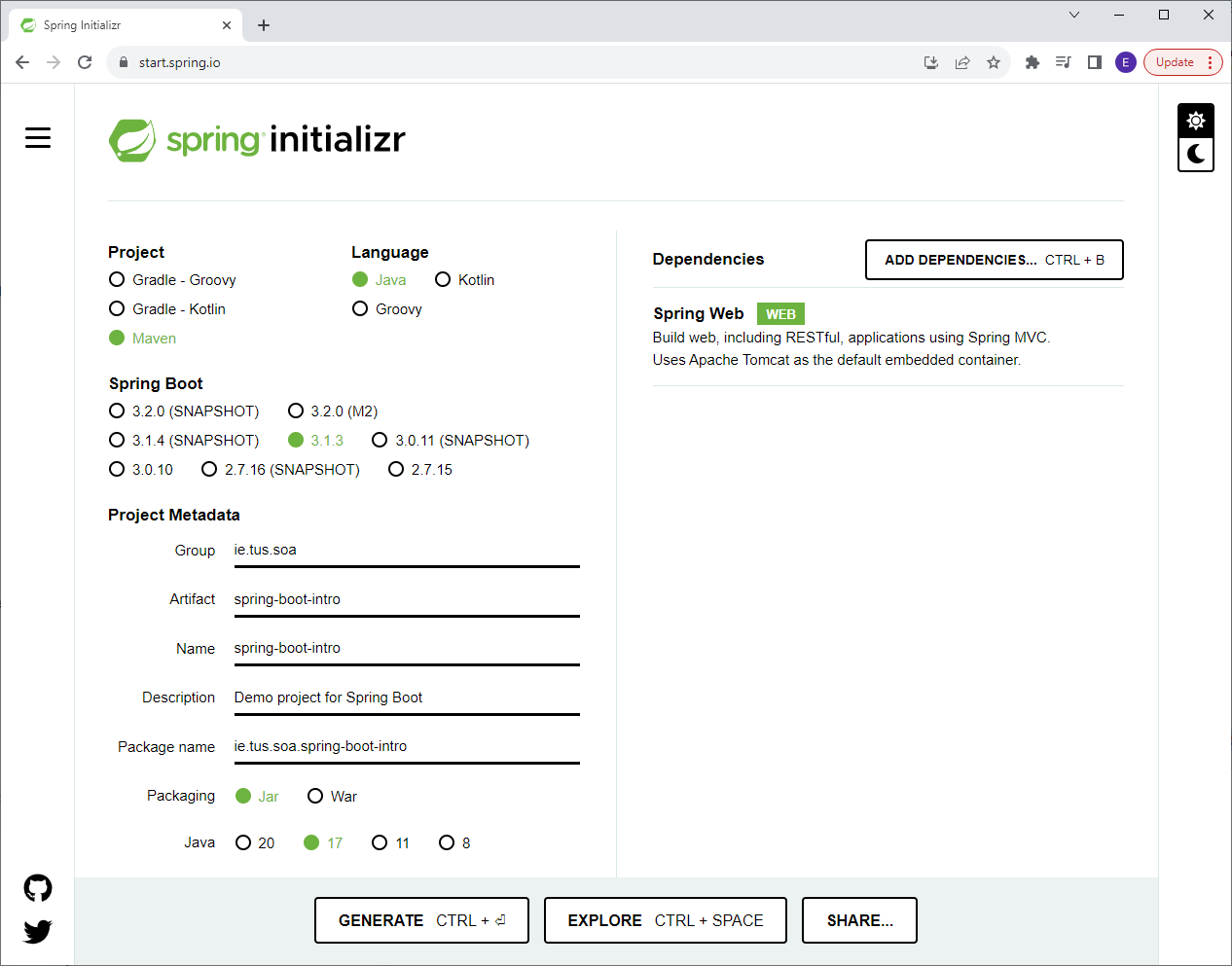
**Lab – Intro to Spring Boot**

Before Spring Boot, setting up and maintaining Java/Spring projects was difficult and took a lot of time. Many tasks are now simplified including:

* Dependency management (pom.xml)
* Web App configuration (web.xml)
* Managing beans (autowiring etc.)
* Setting up servers/logging etc

To get an introduction to Spring Boot, we’ll create a Spring Boot Project and create a simple REST API.

**Step 1: Create a project with Spring initializr**



~~Notes:~~

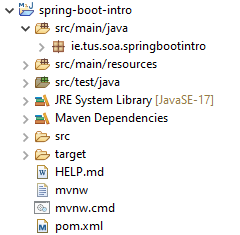
~~We need the~~ ***~~Spring Web~~*** ~~dependency to make a REST API. This also includes the Apache Tomcat server to make it easier to run our applications.~~

~~Use the latest version of Spring Boot that’s not a snapshot.~~

~~As before, click ‘Generate’ to download the starter project. Save and unzip this project (keep all your projects in one place).~~

**~~Step 2: Import the project into eclipse~~**

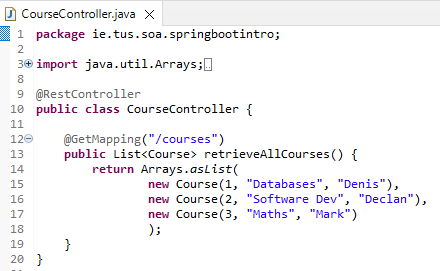
~~Import the project into Eclipse using ‘File – Import’ and choose ‘Existing Maven Projects’.~~



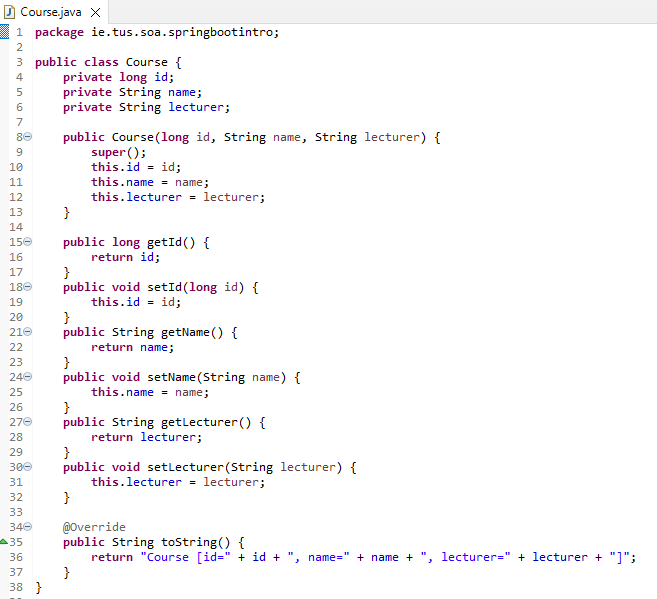
**~~Step 3: Add the following files to make a basic RESTful application~~**

~~We’ll make a simple RESTful application to return a list of courses to the user. We’ll first create a controller class – this contains a~~ ***~~@RestController~~*** ~~annotation to mark this class as a REST controller. It also contains a~~ ***~~@RequestMapping~~*** ~~to map requests to a specific function. E.g. if the request~~ ***~~localhost:8080/courses~~*** ~~is received (by the DispatcherServlet), the function~~ ***~~retrieveAllCourses~~*** ~~is executed, which returns a list of courses.~~

~~Create the following class in~~ **~~src/main/java~~**~~:~~



~~For the previous class to work, we also need to create a Course class. This is known as a plain old java object (POJO). You can create the class, type in the member variables and use Eclipse to generate the~~ ***~~constructor~~***~~,~~ ***~~getters and setters~~*** ~~and the~~ ***~~toString~~*** ~~method:~~



**~~Step 4: View the REST endpoint in the browser~~**

~~Open the following link in the browser to access the endpoint:~~



**General Spring Boot Configuration**

The following sections are general topics that we’ll use in our applications. They include devtools, logging, and profiles.

**Devtools:**

~~Spring Boot includes a~~ ***~~devtools~~*** ~~package to simplify development. This includes the ability to automatically reload the project when changes are made to the code. This means we don’t have to restart the server each time we make a change.~~

~~To add~~ ***~~devtools~~*** ~~to the project, we need to add a new dependency to the pom.xml file as follows:~~

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

~~Note: In future projects you can add this in the ‘Spring Initializr’ under dependencies (search for ‘Dev Tools’)~~

**~~Logging:~~**

~~The default level of logging in Spring is ‘INFO’ – this means that we’ll see some basic information printed out on the console when we run our applications about what port the server started on etc. However, sometimes we may want more info when debugging a project – to set the logging level to ‘DEBUG’ you should add the following line to the~~ ***~~application.properties~~*** ~~file (in src/main/resources):~~



~~The different logging levels that can be set here are shown below.~~ ***~~trace~~*** ~~is the highest level – if set to~~ ***~~trace~~*** ~~all levels below will also be printed. The same goes for the other levels – if set to~~ ***~~info~~***~~, then~~ ***~~info~~***~~,~~ ***~~warnings~~*** ~~and~~ ***~~errors~~*** ~~will also be printed.~~

* ~~trace~~
* ~~debug~~
* ~~info~~
* ~~warning~~
* ~~error~~
* ~~off~~

**Application Profiles:**

~~Spring allows for different profiles for development and production environments. For example, we may be using different databases or different web servers for development. Also, we may have logging set differently for the different environments.~~

~~To set up profiles for development and production, we create two new files: application-dev.properties and application-prod.properties. We use these files to configure the properties we want for each environment:~~





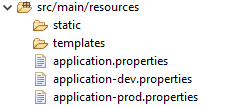
~~The above files set the development logging level to~~ ***~~trace~~*** ~~and the production logging level to~~ ***~~info~~***~~.~~

~~Next we have to tell Spring which environment we want – we do this by adding a line to the application.properties file as follows:~~



~~The ‘debug’ logging level is over-ridden by the~~ **~~dev~~** ~~profile – the resulting logging level is~~ **~~trace~~**~~.~~

~~How~~ ***~~src/main/resources~~*** ~~looks after the changes:~~



**~~Overview of Spring Boot, Spring MVC and Spring~~**

**~~Spring Framework~~**~~: Dependency Injection - @Component, @Autowired etc.~~

**~~Spring MVC~~**~~: Spring module that simplifies building web apps and REST APIs - @RestController, @RequestMapping, @GetMapping~~

**~~Spring Boot~~**~~: Simplifies configuration and makes it quicker to get an application Production-ready quicker, mainly due to the Auto Configuration.~~

~~Enables the following also:~~

* ~~Embedded Server: Tomcat is embedded so no need to setup separately~~
* ~~Logging~~
* ~~Profiles (dev/prod)~~