Create a Spring Boot Project

Estimated time needed: 20 minutes

Overview

In this lab, you will use `Spring Initializr` to generate a project to create a Job Search Spring Boot web application. You will use Java Collections to maintain a database of the jobs.

Learning objectives

After completing this lab, you will be able to:

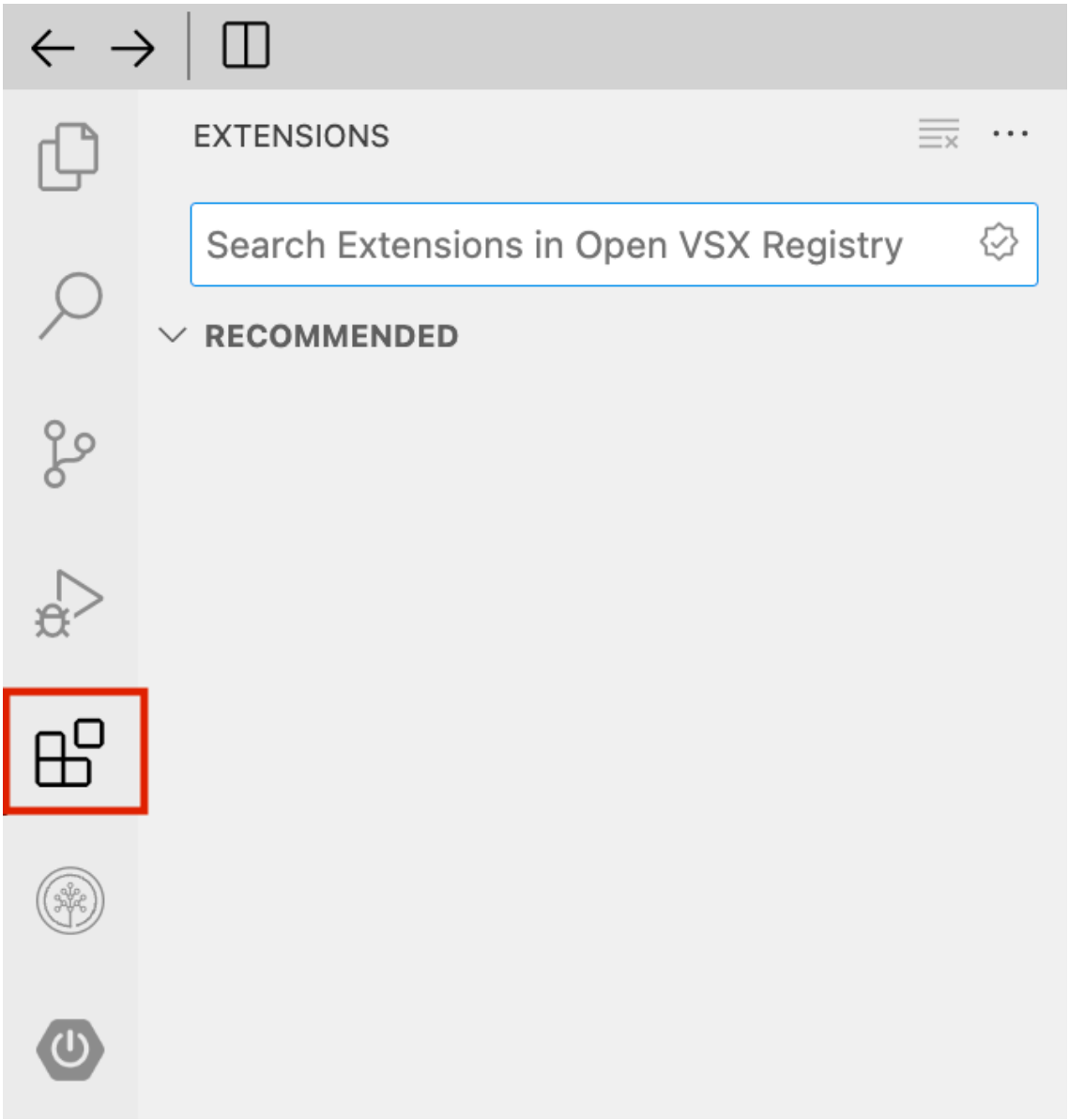
- Use Spring Initializr to generate the project
- Set up a Spring Boot project structure
- Edit the core configuration contained in `pom.xml`
- Create a model with getters and setters
- Create a controller that uses the model and handles http requests
- Build and run the Spring Boot application

Prerequisites (optional)

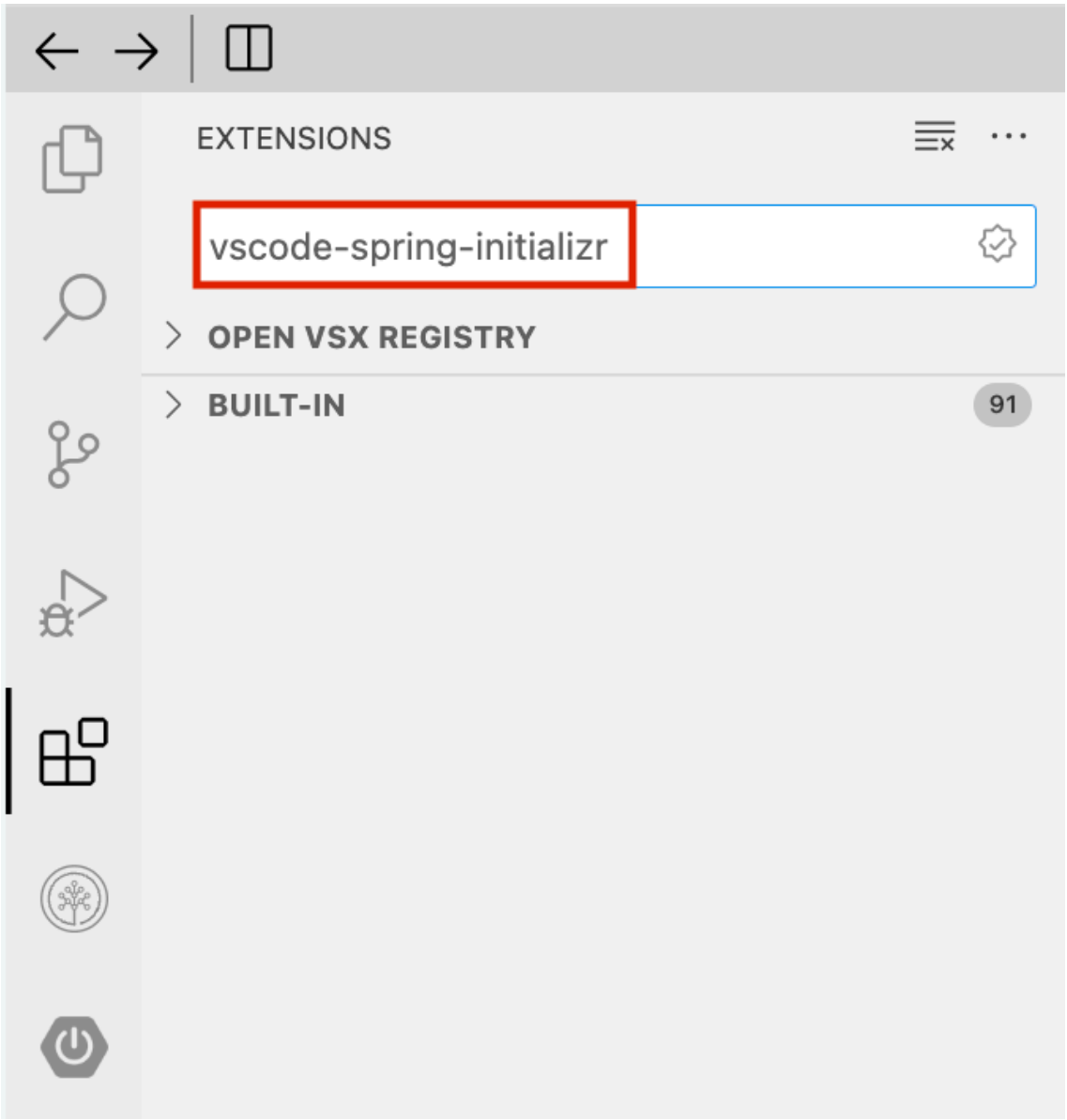
You should know basic Java programming before you get started with this lab. Please ensure that you complete the labs sequentially as they are constructed progressively.

Set up Cloud IDE for Maven Project

1. Click the extensions icon to start installing the extensions.



2. In the search bar for the extensions type `vscode-spring-initializr`.

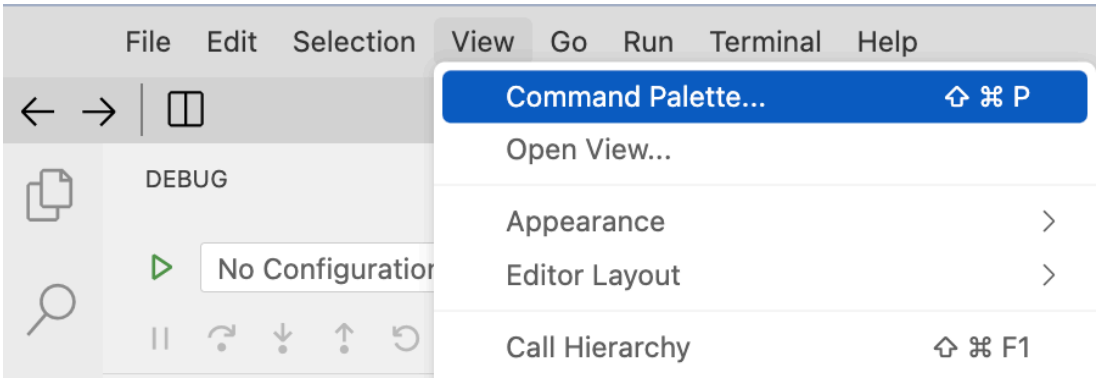


This will list all the extensions you will need for Spring Initializr.

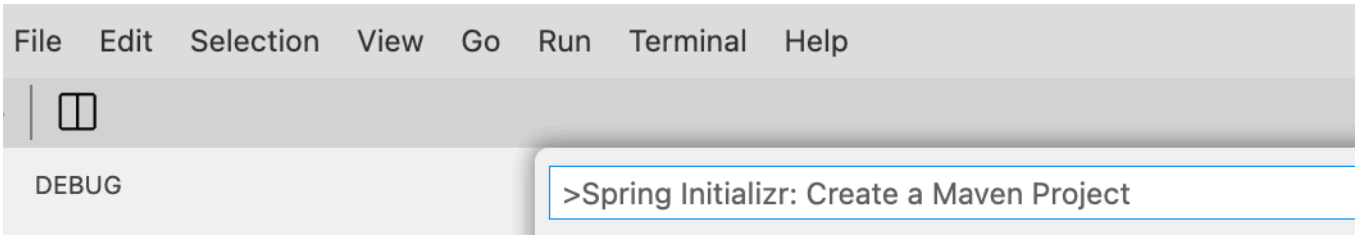
3. Install the `Spring Initializr Java Support` extension.

Create Maven Project

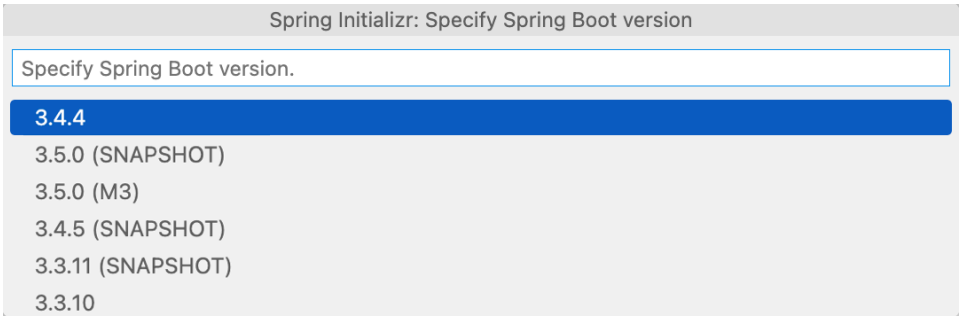
1. Open command palette.



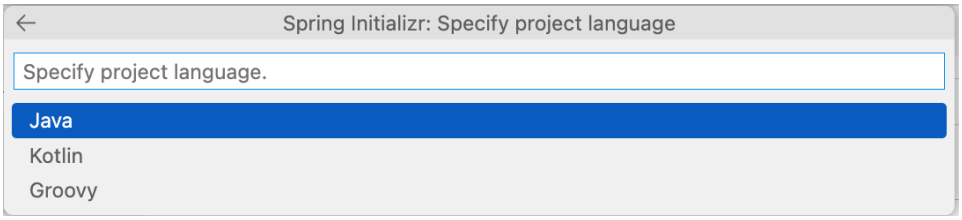
2. In the Command Palette text entry box, type `Spring Initializr: Create a Maven Project`.



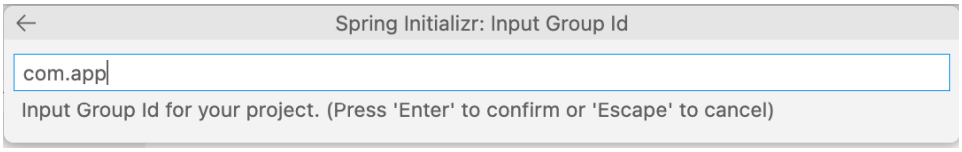
3. Specify the Spring Boot version `3.4.4`.



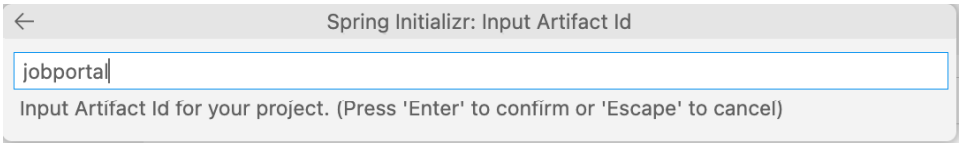
4. Select the language you want to use, in your case`Java`.



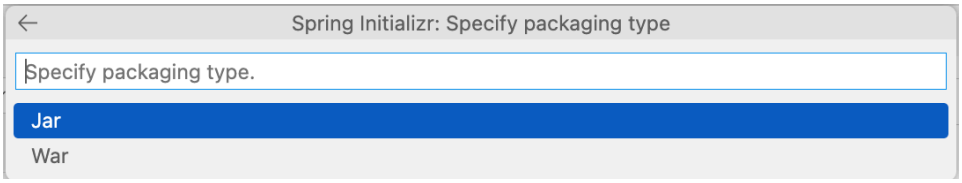
5. Specify the Group Id such as `com.app`.



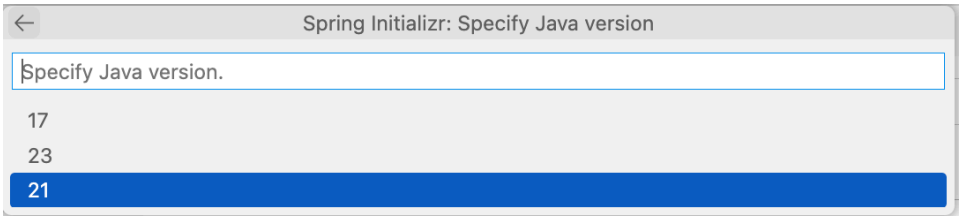
6. Specify the Artifact Id such as `jobportal`.



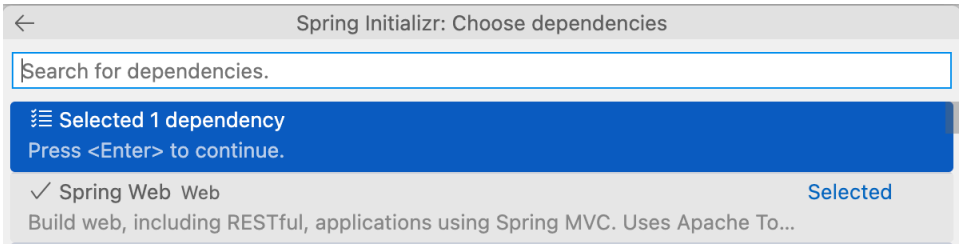
7. Specify the packaging you want to use as `Jar` (for Java archives).



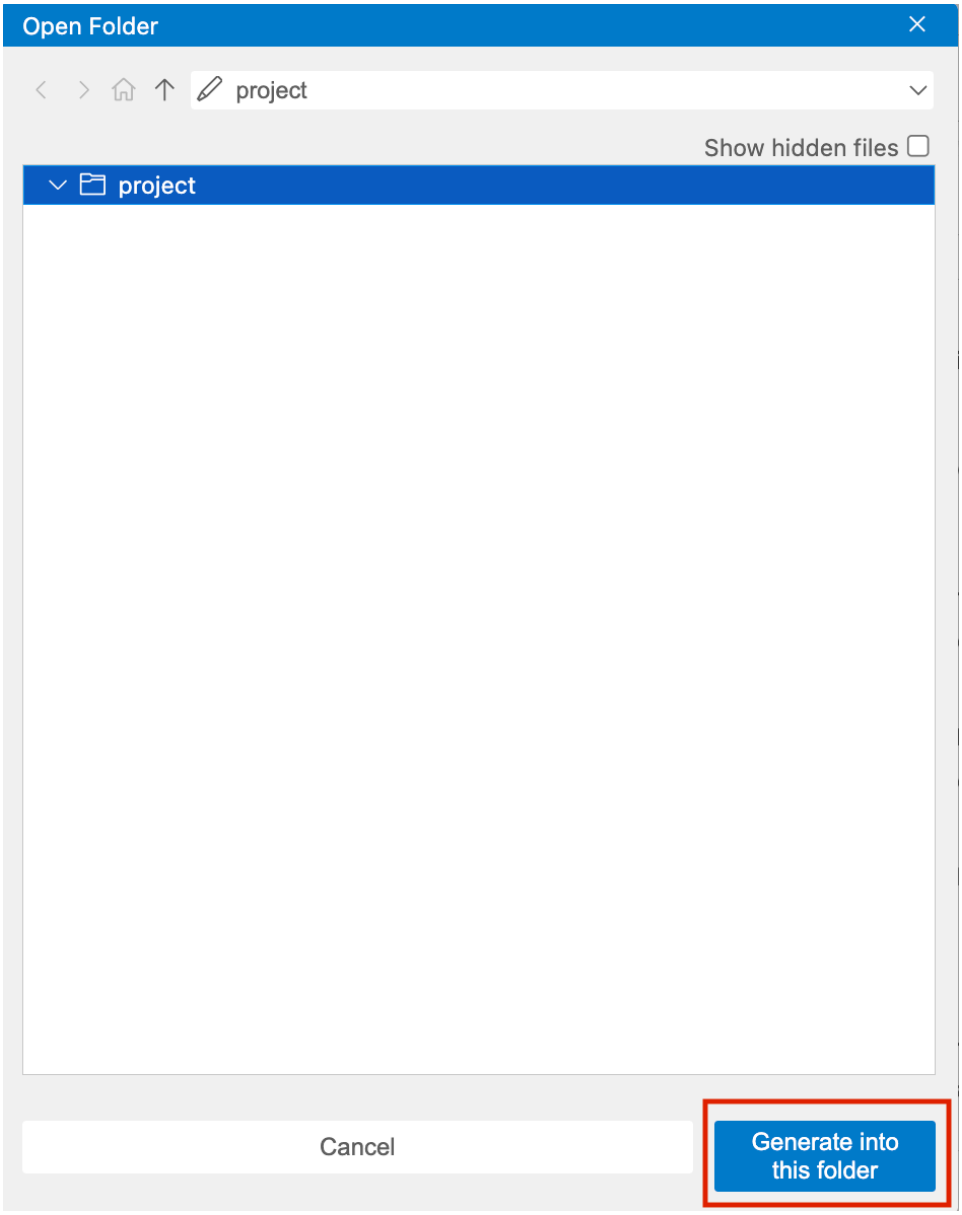
8. Specify the Java version you want to use, `21` in the case of the IDE.



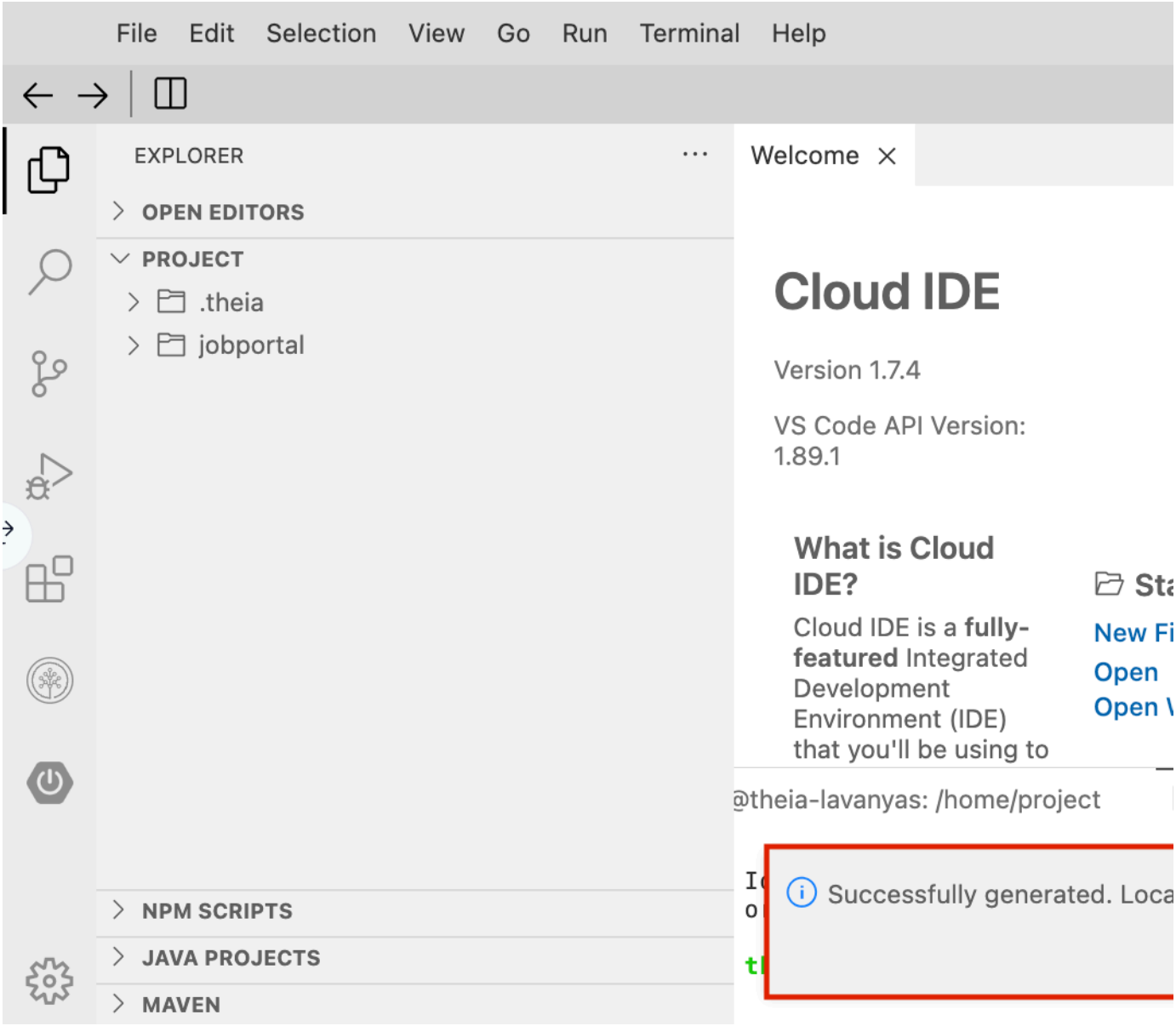
9. Select `Spring Web` dependency for the project and press Enter to add it



10. The project is generated by default in the `/home/project` folder. Click `Generate` to generate the project in the desired folder.



11. Add the project to the workspace.



Configure project and add code

1. Click the button below to open `pom.xml` to edit the project settings.

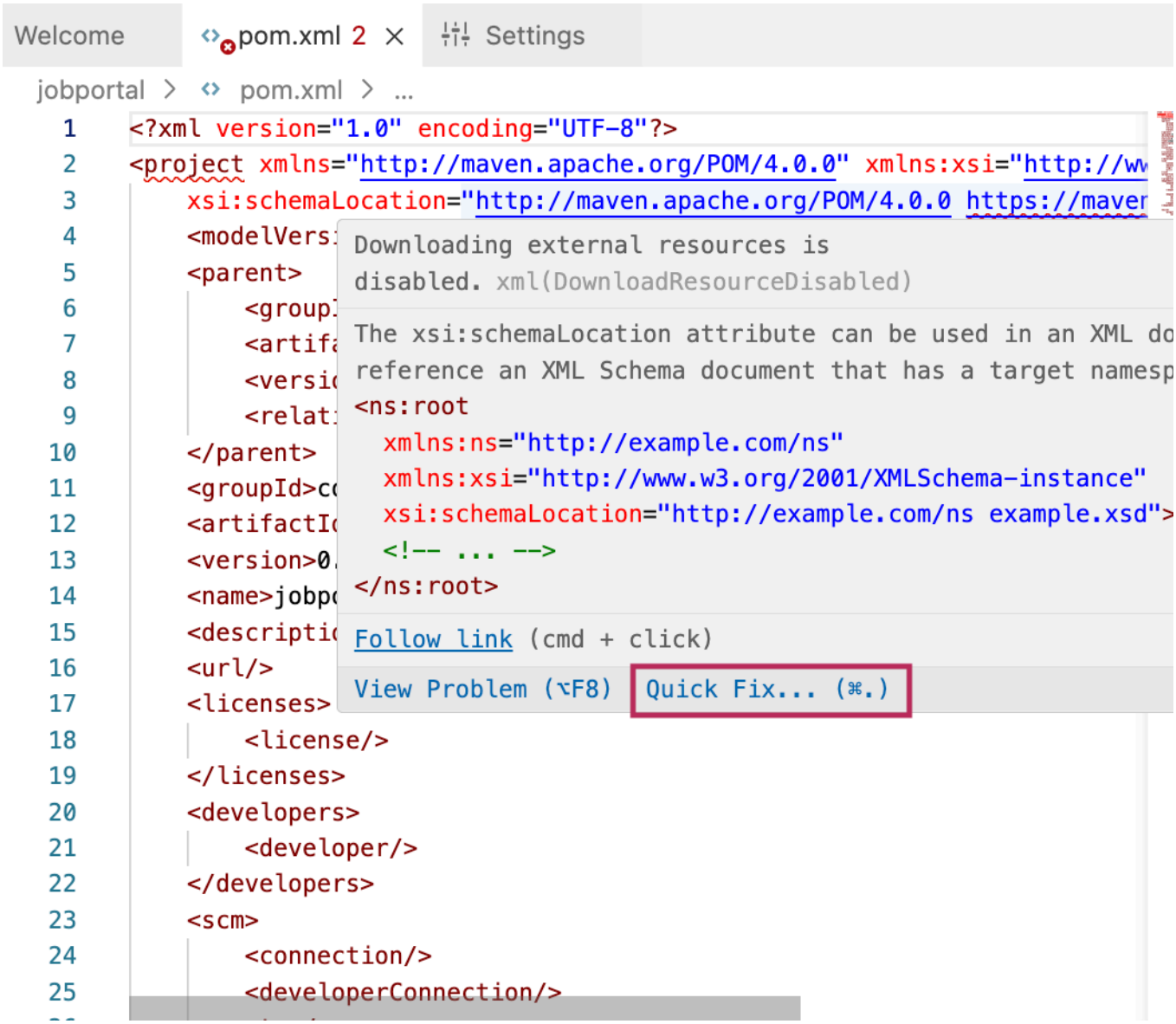
Open **pom.xml** in IDE

Ensure that the required dependencies are included.

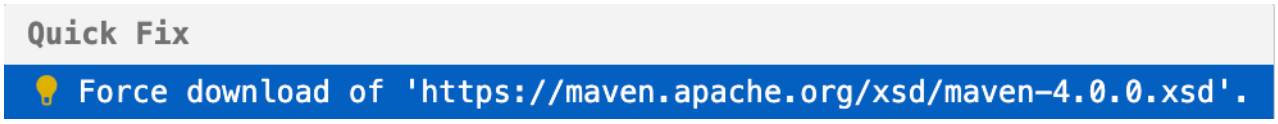
- spring-boot-starter-web
- spring-boot-starter-test
- spring-boot-maven-plugin

With these you get a framework for building web applications using the Model-View-Controller (MVC) pattern with an embedded Tomcat server, which allows you to run your web application without the need for a separate application server.

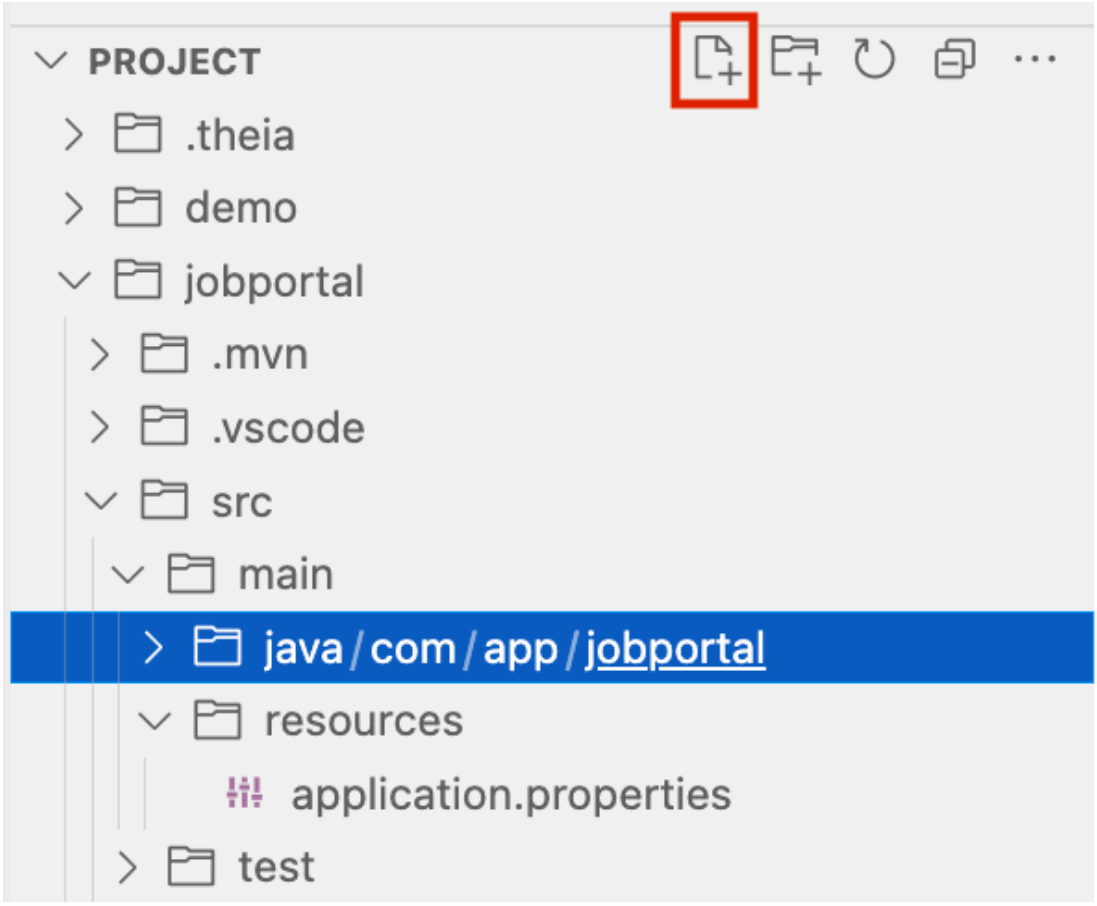
2. Hover over the highlighted links on `pom.xml` and click. A suggestion comes up. Click on `Quick Fix`.



3. Download the `xsd` file.



4. Create a new file named `JobController.java` in the `com/app/jobportal` folder.



5. Add the following code in `JobController.java` to create the `JobSearchController` class.

```
package com.app.jobportal;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class JobController {
    @GetMapping("/")
    public String listJobs() {
        return "<h1>Here are the list of jobs</h1>";
    }
}
```

4. On the terminal, change to the project directory.

```
cd /home/project/jobportal
```

5. Run the following command to maven clean, maven build and run the application.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.app.jobportal.JobportalApplication"
```

`mvn clean install`

- clean:
  Cleans/removes all files generated by the previous build (like compiled .class files in the target directory).
- install:
  Compiles your Java code. Runs tests (if any). Packages your code into a .jar file.

mvn exec:java -Dexec.mainClass=”com.app.jobportal.JobportalApplication”

`mvn exec`

A Maven plugin goal that executes a Java program. `-Dexec.mainClass="..."` Specifies which Java class contains the main() method to run. Here, it’s telling Maven to run JobportalApplication’s main method.

The server will start in port 8080.

6. Click the button below to open the browser page to access the end point.

Job Search Application

You should see a page like this:

6. Press `Ctrl+C` to stop the server.

Add Job model

1. Create a file named `model/Job.java`, which contains the Job Model in `com.app.jobportal.model`package.

```
package com.app.jobportal.model;
import java.time.LocalDate;
import java.time.LocalDateTime;
public class Job {
    private String id;
    private String title;
    private String description;
    private LocalDate postingDate;
    private LocalDateTime lastAccessDate;
    private int numViews = 0;
    public Job(String id, String title, String description, LocalDate postingDate) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.postingDate = postingDate;
        this.lastAccessDate = postingDate.atStartOfDay();
        this.numViews = 0;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public LocalDate getPostingDate() {
        return postingDate;
    }
    public void setPostingDate(LocalDate postingDate) {
        this.postingDate = postingDate;
    }
    public LocalDateTime getLastAccessDate() {
    return lastAccessDate;
    }
    public void setLastAccessDate(LocalDateTime dateTimeLastAccess) {
        this.lastAccessDate = dateTimeLastAccess;
    }
    public void addNumViews() {
        numViews++;
    }
    public String toString() {
```

```
    return this.id + " "+ this.title + "\n"+
    this.description + "\n" + "Posted on " + this.postingDate + "\n" +
    "Last access on " + this.lastAccessDate + "\n" +
    "Total views " + this.numViews;
}
}
```

2. Open `JobController.java` and replace the content with the following code. This will provide the end point for retrieving the jobs list.

```
package com.app.jobportal;
import com.app.jobportal.model.Job;
import jakarta.annotation.PostConstruct;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
@RestController
public class JobController {
    private ArrayList<Job> jobsList;
    @PostConstruct
    public void init() {
        Job job1 = new Job("1", "Software Engineer", "Develop software applications", LocalDate.now());
        Job job2 = new Job("2", "Data Scientist", "Analyze data and develop models", LocalDate.now());
        jobsList = new ArrayList<Job>();
        jobsList.add(job1);
        jobsList.add(job2);
    }
    @GetMapping("/jobs")
    public ArrayList<Job> getJobs() {
        for(Job job: jobsList) {
            job.setLastAccessDate(LocalDateTime.now());
            job.addNumViews();
        }
        return jobsList;
    }
}
```

3. Run the following command to maven clean, maven build and run the application.

```
mvn clean install && mvn exec:java -Dexec.mainClass="com.app.jobportal.JobportalApplication"
```

The server will start in port 8080.

4. Click the below button to open the microservice through the browser.

Job Search Application

You should see a page like this.

Pretty print ☑


```
[
  {
    "id": "1",
    "title": "Software Engineer",
    "description": "Develop software applications",
    "postingDate": "2025-04-04",
    "lastAccessDate": "2025-04-04T06:37:50.233186331"
  },
  {
    "id": "2",
    "title": "Data Scientist",
    "description": "Analyze data and develop models",
    "postingDate": "2025-04-04",
    "lastAccessDate": "2025-04-04T06:37:50.233236641"
  }
]
```


Practice exercise

1. Add an import statement in `Jobcontroller.java`.

   ```
   import org.springframework.web.bind.annotation.PathVariable;
   ```

2. Add another end point in `JobController.java` with the following code. Clean, compile, and run the app and check the end point in the browser.

   ```
   @GetMapping("/jobs/search/{query}")
   public ArrayList<Job> searchJobs(@PathVariable String query) {
       ArrayList<Job> matchingJobs = new ArrayList<>();
       for (Job job : jobsList) {
           if (job.getTitle().toLowerCase().contains(query.toLowerCase()) ||
               job.getDescription().toLowerCase().contains(query.toLowerCase())) {
               matchingJobs.add(job);
           }
       }
       return matchingJobs;
   }
   ```

Conclusion

In this lab, you have:

- Used Spring Initializr in VSCode to generate the project
- Set up a Spring Boot project structure
- Created a controller that handles http requests
- Created a model with getters and setters
- Modified the controller that uses the model and handles http requests
- Built and ran the Spring Boot application

Author(s)

Lavanya