Project Description

In this project, you'll use the React, Java, Spring Boot knowledge you've gained along with a MySQL database to implement a talent recruitment system. You will be implementing both the front and back end of the application. The application has may aspects to it. To complete it the class will need to pool their efforts. Students will be assigned to groups. Each group will be assigned a portion of the whole project that they need to complete. The finished application will be made up of the contributions from all the groups.

This document provides a plan for the entire end to end application. Although your group will only be responsible for a portion of the final application you should read through this entire document to understand how everything fits together.

The overall application can be broken down into the following parts:

- Part 1: Understanding the Project
- Part 2: Setting up the Development Environment
- Part 3: Creating the Database Schema and Sample Data
- Part 4: Creating the Back-End API Project
- Part 5: Connecting to Database from Java
- Part 6: Coding the API Endpoints
- Part 7: Creating the Front-End React Project
- Part 8: Implementing Front-End Features
- Part 9: Putting it all Together
- Part 10: Wrapping Up

Collaborating with your group and other groups

The class will use GitHub for collaboration. Two repositories will be set up: one for the React front-end project and the other for the Java back-end API project.

Be careful:

- Do not push code to the shared repository unless you know that its working
- Notify other groups when you push code to the shared repository
- When other groups push changes, don't wait to pull them to your local environment
- Groups should work together to resolve any issues that come up

Part 1: Understanding the Project

The project contains a back-end API and a front-end client application. Before you start working you need to read through all the provided documents.

When working on a large project, design decisions need to be made for each group's work to fit together with the work of the others. The provided documents include suggestions like SQL Scripts, a list of API endpoints and a list of screens for the client application which you can use to get started. If you find that you want to do some things differently, that's ok, but remember you need to communicate with the other groups about any changes you're making so they can incorporate it into what they're doing as well.

Part 2: Setup the Development Environment

Development Machine
Each student will be provided with a remote virtual machine to work in. The VMs will be configured like with the following:

- Ubuntu 22.04 LTS
- 8GB RAM
- Chrome
- Git
- Java 22.0.1
- Eclipse IDE 2024-06
- NodeJS 18.20.2
- Microsoft Visual Studio Code
- Postman REST client
- MySQL server and CLI
- MySQL Workbench

Source Code Management
The Git SCM tool will be used for source code management. Each student will clone two repositories from GitHub to their local machine:

- talent-api
- talent-client

Students will need to create an account on GitHub before they can push code up to the shared repository.

Part 3: Setting up the Solution Projects

To better understand the overall application structure and data flow, each team member will setup the solution projects and get the full-stack partial-solution application running. The high-level steps are listed below:

1. Setup the Database
2. Setup the REST API
3. Setup the Front-End application
4. Check out the solution application

Detailed instructions for these steps are available in the **solution-setup.pdf** file. Open that file now and follow the steps shown there to set up the full-stack solution.

When you are done setting up and checking out the solution you should come back here and continue with the next part.

Part 4: Creating the Back-End API Project

Tech involved: Java, Spring Boot, Gradle, Git

Next, you'll need to create the Spring Boot server project.

- Create a new Spring Boot project using the spring initializer site: **https://start.spring.io/**
  - See settings below for generating the project zip
  - Download the project zip file and unzip into a local directory ("~\ProjectWork")
  - Git enable the local directory and sync it up with the GitHub repo you created
  - Open the project directory in the eclipse IDE

- The following settings should be used when generating the Spring Boot project:
  - Project Build: Gradle – Groovy
  - Language: Java
  - Spring Boot version: 3.3.1
  - group - com.example
  - artifact – talent-api
  - name – talent-api
  - description - Capstone back-end API project
  - package: com.example.talent-api
  - Packaging: Jar
  - Java version: 22
  - Dependencies:
    - Spring Web
    - Spring Data JPA
    - MySQL Driver

- You'll be using the Eclipse IDE when coding the API project. Follow these steps to open the "talent-api" project in Eclipse:
  - From the Linux "Files" app, navigate to ~/ eclipse/jee-2024-06/eclipse
  - Right-click on "eclipse" and choose run
  - Wait for Eclipse to load
  - Select: File/ Import/ Gradle/ Existing Gradle Project
  - Click Next
  - For "Project root directory" browse and select ~/ProjectWork/talent-api
  - Click Finish
  - Wait for the import to complete
  - The project is now ready to work on

- Commands for initializing a Git repo in the local project directory:
  - git init
  - git add .
  - git commit -m "initial commit"

- Commands to upload the project to an existing GitHub repository (adjust as needed):
  - git remote add origin {github repo url}
  - git push origin {branch-name}
  - branch-name is typically "main" or "master". Check your local repo by executing "git branches" that will show you the current branch name.

- Get started developing the API Project
  - Start by creating a GET route for the path "/" that returns a simple text string: "This is the Talent-API!"
  - Compile and run the project
  - Using a browser or the Postman rest client, test out the endpoint by making a GET request to "/"
  - Make sure the text string is returned as expected
  - Once the "/" endpoint has been coded and tested, do a local commit and then a "git push" up to the shared repo on GitHub
  - The project is now ready for the rest of the groups to copy down from GitHub using "git clone"

Part 5: Connecting to the Database from Java

Tech involved: Java, Spring Boot, MySQL, Gradle and Git

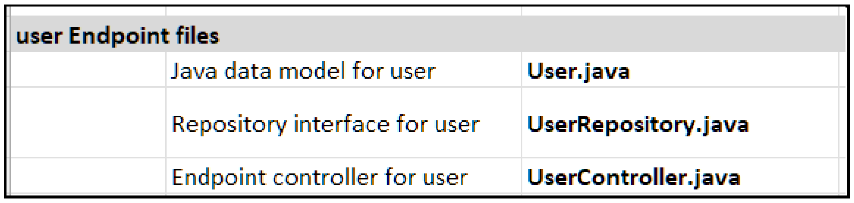
The following were created earlier when you setup the solution project:

- The "talent" database
- The "user", "manager", "candidate", "job" and "application" tables

- The user "dbuser" with password "passpass"

You can continue to use the setup you created earlier with the "talent-api" you are coding now. Be mindful though that changes you make (if any) to the database tables may affect the functioning of the solution application. If this becomes a problem, you can always create the tables again under a different database name ("talent2...").

You will need to set up the API project so that it can interact with MySQL tables. This typically involves creating several Java classes for each table. For example, the following classes should be created to interact with the "user" table:



user Endpoint files		
	Java data model for user	**User.java**
	Repository interface for user	**UserRepository.java**
	Endpoint controller for user	**UserController.java**

The above screenshot comes from the "**REST-API-Artifacts**" tab of the "**project-artifacts-list.xlsx**" file.

A similar set of classes/interfaces will have to be created for the other API endpoints:

- login
- managers,
- candidates,
- jobs,
- applications

Information for these endpoints (like what is listed above for "users") is available in the file:

"**project-artifacts-list.xlsx**"

Don't forget to set the following db related properties in the project's "application.properties" file:

```
spring.datasource.url = jdbc:mysql://localhost:3306/talent
spring.datasource.username = dbuser
spring.datasource.password = passpass
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto = none
spring.sql.init.mode=never
```

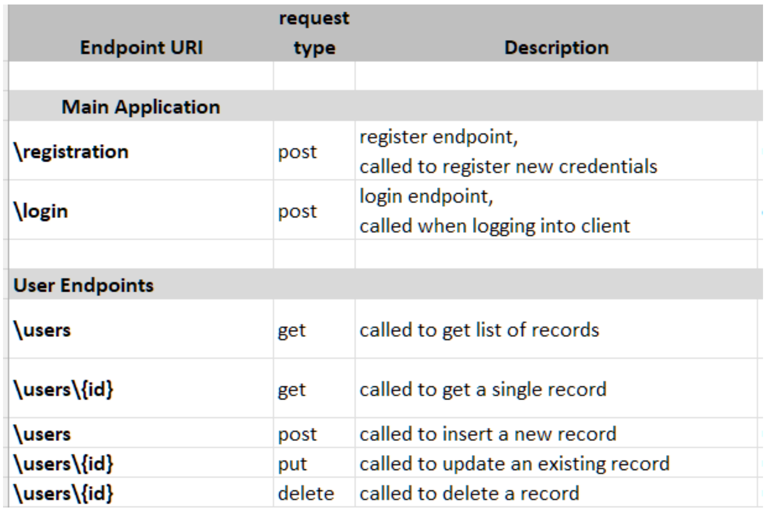
(make sure there are no space characters after the username and password values!)

For more information on MySQL integration see:

- https://www.freecodecamp.org/news/how-to-build-a-rest-api-with-spring-boot-using-mysql-and-jpa-f931e348734b/
- https://spring.io/guides/gs/accessing-data-mysql

Part 6: Coding the API Endpoints

The application makes use of many endpoints. The instructor will divide up the endpoints so that each group can work on them at the same time. The full list of endpoints is listed in the "**project-artifacts-list.xlsx**" file under the "**Required API Endpoints**" tab. The screenshot below shows some of the information that can be found there:



Endpoint URI	request type	Description
Main Application		
\registration	post	register endpoint, called to register new credentials
\login	post	login endpoint, called when logging into client
User Endpoints		
\users	get	called to get list of records
\users\\{id}	get	called to get a single record
\users	post	called to insert a new record
\users\\{id}	put	called to update an existing record
\users\\{id}	delete	called to delete a record

Manual Testing

As each endpoint is completed it should be manually tested using one or more of the following methods:

- Browser,
- At the command line using Curl
- Using a dedicated REST client utility like Postman

Automated Junit Testing

For this capstone, automated JUnit tests should be created for the following endpoints:

- POST /login
- POST /registration
- GET, POST /users
- GET, PUT, DELETE /users/{id}
- GET /jobs
- GET /jobs/{id}

- GET /jobs/{manager_id}

Example Unit Tests are available in the solution API project. Note. For a production application, ALL endpoints should be covered by Unit tests.

Tests can be run from:

- Inside the Eclipse IDE
- Or, from a terminal with the command: ./gradlew test

An Html test report should also be generated with this command:

- ./gradlew testReport

The Html testReport task needs to be added to build.gradle like this:

```
task testReport(type: TestReport) {
  reportOn test
  destinationDir = file("$buildDir/reports")
}
```

Html test reports are output here:

```
build/reports/index.html
```

Information on creating Junit tests for Spring Boot REST services is available here:

https://spring.io/guides/gs/testing-web
https://www.baeldung.com/spring-boot-testing

Configuring API for CORS

In order for the front-end browser client application to be able to access the REST API server you will also need to configure the API to handle CORS (Cross Origin Request Sharing) requests. One configuration method involves creating a Java class that implements "WebMvcConfigurer".

An article on implementing WebMvcConfigurer can be found here:

https://www.geeksforgeeks.org/spring-mvc-webmvcconfigure/

Remember to commit and push completed endpoint code as your group finishes and tests the endpoints they've been assigned. At the same time, it would be good to notify other groups that you've updated the code on GitHub so they can pull your updates.

Part 7: Creating the Front-End React Project

Tech involved: React, Git

In this part you create the front-end client project. This only needs to be done once. Whichever group is assigned to it should:

- Create the project
- Push it up to the GitHub client project: "talent-client"
- Notify other groups so they can Git Clone the GitHub project.

Before starting, make sure you are working with NodeJS version 18.x (not 19.x). The initial commands required include:

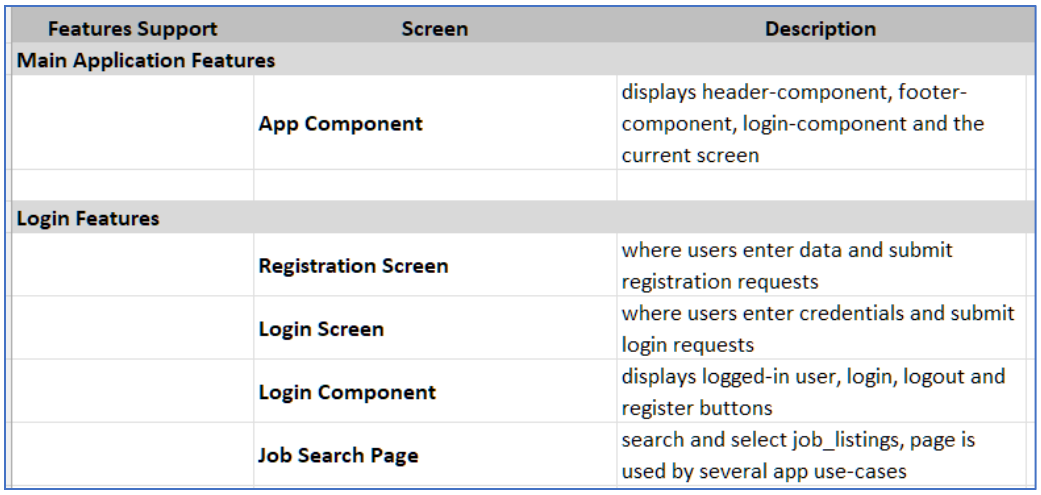
```
npm install -g vite
npm create vite@latest talent-client -- --template react
```

Part 8: Implementing Front-End Features

Tech involved: React, Java, Spring Boot, Git and MySQL

The client application includes many screens and pages, each of which need to be developed. The instructor will assign a subset of the overall front-end tasks for each group to complete.

A tentative list of "**Screens**" appears on the " **Front-End-App Screens**" tab in the "**project-artifacts-list.xlsx**" file. The following screenshot shows a portion of that list:



Features Support	Screen	Description
Main Application Features		
	App Component	displays header-component, footer-component, login-component and the current screen
Login Features		
	Registration Screen	where users enter data and submit registration requests
	Login Screen	where users enter credentials and submit login requests
	Login Component	displays logged-in user, login, logout and register buttons
	Job Search Page	search and select job_listings, page is used by several app use-cases

The list is based on the suggested implementation. You may wind up creating more or fewer screens based on how your own application design is configured.

Remember, the capstone is a group effort. Any design decisions you make that differ from the suggested implementation should be communicated to everyone working on the project.

Many client screens will require working copies of the back-end API server endpoints. Make sure your local copy of the back-end API server project, "talent-api", is up to date.

The API server can be started from its project directory with the following command:

```
./gradlew bootRun
```

Keep in mind that you'll need implementations of both the client and the server working together to complete the capstone.

Automated Unit Testing

For this capstone unit tests should be created for the following:

- Login Screen
- Login Component
- User Management Page
- Job Search Page

The unit tests should verify that the components can be instantiated and test that individual methods in the components are functioning.

Note: For production applications all screens are typically covered by unit tests.

Creating unit tests will require that you add these packages to the talent-client project:

```
// basic testing packages
npm install --save-dev vitest@1.6
npm install --save-dev jsdom@24

// react testing library packages
npm install --save-dev @testing-library/react@14.3
npm install --save-dev "@testing-library/jest-dom@5.17
npm install --save-dev @testing-library/user-event@14.5

// mocking packages
npm install fetch-mock@9.11 --save-dev
npm install node-fetch@2.7 --save-dev
npm install --save-dev lodash@4.17
npm install --save-dev lodash.isequalwith@4.4.0
```

Don't forget to add a test **`setup.js`** and update the **`vite.config.js`** as well.

The following links should be of help when setting up and creating unit tests:

- https://victorbruce82.medium.com/vitest-with-react-testing-library-in-react-created-with-vite-3552f0a9a19a

- https://www.npmjs.com/package/@testing-library/react#installation
- https://testing-library.com/docs/react-testing-library/cheatsheet/

Part 9: Putting it all Together

Tech involved: React, Java, Spring Boot, Git and MongoDB

This last major step is to get the final-implementations of the talent-client app and the talent-api server working together.

While it would be great if everything worked right away it's more likely that there will be some issues along the way. Some issues may require last minute coding changes.

Also, some groups may not have completed all the features assigned to them when it comes time to put together the final implementation.

Steps:

- An hour before the final integration, the instructor will ask the groups to:
  - Stop working on any new features
  - If needed, roll-back any half-completed features
  - Check in all completed and working code

- The final implementation will include:
  - Stopping any existing front and back-end projects
  - Cloning the final versions of the front and back-end client to a fresh directory (~/ProjectFinal).
  - Checking that MySQL is running
  - Running the talent-api project (./gradlew bootRun)
  - Running the talent-client project (npm run dev)
  - Manually testing all features.
  - Fixing anything that's not working
  - Committing and pushing fixes to GitHub
  - Pulling changes down to the front and back-end project directories
  - Restarting and retesting the applications

Part 10: Wrapping Up

In this part you will:

- Assemble project deliverables
- Get ready to present your work

Assemble Project Deliverables

- Create a directory ~/LabWork/Deliverables
- Add WorkingCode:
  - Clone the two final Git repos to a "temp" directory
  - Zip up each project directory,
  - Copy the zip files ("talent-client.zip", "talent-api.zip") to the Deliverables directory

- *Add SQL Scripts*
  - Export the schema and creation scripts for the application's tables in the "talent" db (try "sudo mysqldump talent > sqlscripts.sql"...)
  - Clean up the script, delete unnecessary comments etc.
  - Rename the file to "sql-create-scripts.sql"
  - Copy the renamed file to the Deliverables directory

- Create a wrap-up document
  - Named: "fs-capstone-wrap-up_{your-name}.docx"
    - Example: " fs-capstone-wrap-up_jack-smith.docx
  - *Add to the document an **Application Overview** section with:*
    - Problem statement outlining the *needs that the application addresses*,
      - Example: Too much time is being spent managing the recruitment process, a solution is needed that provides self-service capabilities for managers and candidates in the areas of: creation of job-openings, submitting and following up on applications, and eases communication of application and job-openings status ...
    - Overview of the technology stack that was used
    - Description of the implemented solution
      - Example: This solution provides these major benefits ...
  - *Add to the document a Status section that lists:*
    - Planned features – what's included in the feature list
    - Features that were completed – what's complete and ready to go now
    - Features that still need to be completed – what's features are planned for a future release
  - *Instructions for building/running the application with the provided code:*
    - Include all instructions someone might need to get the application running based on the code and SQL script deliverables.
  - *Comments on the student's personal experience regarding the project.*
    - Provide a personal look into your experience during the development project. What worked well, what would you change, what did you learn? How was the experience of working in groups? Were there any problems with communication? Any technical issues along the way? Etc. ...

Get Ready for the Presentation

In the end, each group will make a presentation related to their experience in working on the project. Take an active role when your group is discussing what to present. You may want to suggest some of the personal comments you provided in your wrap-up document. If your group added or modified the application design to make it better, that would be something to talk about. If you thought of a way to streamline the development process that would be a good thing to discuss. An overview of what the group accomplished is also good.