Desenvolvimento de Aplicações Empresariais – 2021-22-1S

Engenharia Informática – 3.º ano – Ramo SI

Worksheet 2

Topics: Web Services; Presentation Logic Layer (PLL)

Requirements: Make sure you have the latest Node.js LTS version. If not or unsure, download and install/upgrade it.

In the previous worksheet we created an application that had a JPA Entity called `Student` and a stateless EJB called `StudentBean`, to manage the Business Logic for the CRUD (Create, Read, Update, Delete) operations related to students. By doing so, we developed the `create(...)` method that receives the properties and saves a student in the database, thanks to an Entity Manager. Also, using the Singleton EJB `ConfigBean`, we populated the database in order to see some results.

PART I: Creating a (Web) Service Layer

Today, we are going to add a Service Layer (Web Service Layer) that stays on top of the Business Logic Layer (BLL) and serves as an Application Programming Interface (API) between our Java EE Enterprise Application' Business Logic Layer (BLL) and the components of the Presentation Logic Layer (PLL).

1. Your application was already created with an Application Java Class. It's your entry point for your (Web) Service Layer. It's called `HelloApplication`. Let's rename it to `AcademicsApplication` to match our project name (right-click on the HelloApplication.java file and do Refactor->Rename...).

```
package pt.ipleiria.estg.dei.ei.dae.academics;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/api")
public class AcademicsApplication extends Application {}
```

2. You can delete the class `HelloResource`. We will cover this and create ours later.

3. Let's create a Data Transfer Object (DTO) that serves as a contract between the Service Layer API components the (web) client applications in order to establish a "way of interaction".
Create a new Java class called `StudentDTO` in a package "dtos". This class must have the properties we want to make public to the "world". For the purpose of the worksheet, the `StudentDTO` is a class that has the

same properties that the Student Entity, namely: username, password, name, email. It is a simple Plain Old Java Object (POJO) class, so make sure **not** to annotate it as an Entity. Make sure this class implements the Serializable interface. Generate the default no parameters constructor, and a constructor that receives all the properties values and the getters and setters for each property (you can/**should** right-click on the code editor and choose "Generate…" to automatically generate this kind of code).

4. Now let's make a (web) service that handles the requests from client applications. Create a class called `StudentService` in the package "ws", and annotate it with the `Path`, `Produces` and `Consumes` Java EE annotations. Also inject a StudentBean `EJB`:

```java
package pt.ipleiria.estg.dei.ei.dae.academics.ws;

import pt.ipleiria.estg.dei.ei.dae.academics.ejbs.StudentBean;

import javax.ejb.EJB;
import javax.ws.rs.Consumes;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("students") // relative url web path for this service
@Produces({MediaType.APPLICATION_JSON}) // injects header "Content-Type: application/json"
@Consumes({MediaType.APPLICATION_JSON}) // injects header "Accept: application/json"
public class StudentService {
    @EJB
    private StudentBean studentBean;
}
```

5. Make sure to import all needed classes (`Path`, `Consumes`, `Produces` and `MediaType` from the `javax.ws.rs` package, `EJB` from the `javax.ejb` package – please double check that you imported the correct classes).
`StudentBean` and `StudentService` do not need a constructor. The annotation `@EJB` already tells the Java EE that this is an EJB that should be resolved using Dependency Injection (another example is the `@PersistenceContext` that you added in the previous worksheet, to resolve an Entity Manager instance).

6. Now let's tell our Java EE application how we want to fetch all the students from the database. To do so, we need to create a named query in the Student (entity) class. A named query means: associate an unique name in the entire application, so that when we ask for a query with this name (e.g.: "getAllCars"), the application should resolve this name as the following query "SELECT c FROM Cars c WHERE c.status = 'active' AND c.country = 'pt-PT' AND c.deletedAt IS NULL ORDER BY c.brand". Translating this into code, it means you must add an annotation, called `@NamedQuery` in the `Student` Entity. Often in a typical enterprise solution, an Entity needs to have many queries. Thus, we wrap the `@NamedQuery` annotation in the `@NamedQueries` annotation. The result in the Student (entity) class should be this:

```java
@Entity
@NamedQueries({
        @NamedQuery(
                name = "getAllStudents",
                query = "SELECT s FROM Student s ORDER BY s.name" // JPQL
        )
})
public class Student implements Serializable {
    // here go properties, default constructor, getters and setters...
}
```

NOTE: JPQL (Java Persistence Query Language) is an object-oriented query language for the Java Persistence API (JPA) specification. As you can see, it is SQL alike, but also has some differences that are important to pay attention. To learn and explore more, use the official documentation: https://javaee.github.io/tutorial/persistence-querylanguage.html

7. Return to the `StudentService` class. Add the following methods:

```java
@GET // means: to call this endpoint, we need to use the HTTP GET method
@Path("/") // means: the relative url path is "/api/students/"
public List<StudentDTO> getAllStudentsWS() {
    return toDTOs(studentBean.getAllStudents());
}

// Converts an entity Student to a DTO Student class
private StudentDTO toDTO(Student student) {
    return new StudentDTO(
            student.getUsername(),
            student.getPassword(),
            student.getName(),
            student.getEmail()
    );
}

// converts an entire list of entities into a list of DTOs
private List<StudentDTO> toDTOs(List<Student> students) {
    return students.stream().map(this::toDTO).collect(Collectors.toList());
}
```

8. You can see that this code is not working. Let's create the method `getAllStudents()` in the `StudentBean`, that retrieves all the students as defined in our previous named query.

```java
public List<Student> getAllStudents() {
    // remember, maps to: "SELECT s FROM Student s ORDER BY s.name"
    return (List<Student>) em.createNamedQuery("getAllStudents").getResultList();
}
```

9. Make sure your Docker Desktop is running, deploy your application (`make deploy`) and access `http://localhost:8080/academics`. You **should** get the result "**Forbidden**".

10. Change the URL to `http://localhost:8080/academics/api/students/`. You should get all the students you populated in your database, ordered by name.

NOTE: Your web service lives in the relative URL "/api", just like defined in the Academics class, that extends an Application Web Service. The rest of the path refers to students calls in the `StudentService`, because you told to your `StudentService` that all of its methods must live inside the URL pattern `"/students"` relative path. Finally, we instructed the method to obtain all the students to be in the `"/"` URL relative path, so that `http://localhost:8080/academics/api/students` retrieves all the students. Feel free to change all the URL paths to understand better its way of working behind the scenes.
For instance, in the `StudentService`, for the method get`AllStudents()`, change the `@Path("/")` to `@Path("/all")` and re-deploy the application. Now, to get all the students, you should enter URL
`http://localhost:8080/academics/api/students/all`.

PART II: Creating a Presentation Logic Layer application

11. Now let's create an independent (client) application to consume the Service Layer API we've just developed. We will create a new project that represents our Presentation Logic Layer components, according to the 3-layer architectural pattern you learned in theory classes. You will create another **independent** application, so make sure you **do not create it** inside your current project root directory. They shall be 2 separated projects. For that, open a terminal, go to a location where you want to save this new project and hit the following command:

```
$ npx create-nuxt-app academics-client
```

Answer the questions:

```
? Project name (academics-client) # or other name you may prefer
? Programming Language (Javascript)
? Package manager (Npm)
? UI framework (Bootstrap Vue) # or another you may prefer
? Nuxt.js modules (Press <space> to select Axios)
? Linting tools (None. If you like it, Press <space> to select one) # eg: ESLint
? Testing framework (None)
? Rendering mode (Universal (SSR))
? Deployment target (Server (Node.js hosting))
? Development tools (None. If you use VS Code, you may use the jsconfig.json)
What is your GitHub username? (None, or leave default)
? Version control system (None. If you know what Git is, you are free to choose)
```

12. Using the terminal, go to the created directory (academics-client) and execute the following command to start the NUXT application in development mode:

```
$ npm run dev
```

The application may ask you to help them by providing anonymous data usage. Feel free accept or reject. The command launches the app in the port 3000 and opens it in your default browser (or not…; if not, point your browser to `http://localhost:3000`). The application does hot-reload, which mean that in most of the cases, when you change a file, you don't need to re-run the application. It listens for changes and reloads the application.

13. Open the project in the with an IDE/Text Editor you feel more comfortable with. For instance, you can open with: WebStorm (recommended, also created by JetBrains), IntelliJ Idea Ultimate, Visual Studio Code, Sublime Text or Atom.

NOTE: To have a better Vue.js integration using IntelliJ IDEA Ultimate or WebStorm, make sure you have the following plugins installed: Vue.js, NodeJS and IntelliVue. To verify this, go to the IDE settings/preferences, on the left panel, select "Plugins". Verify if they're already installed, and if not, search for them in the marketplace and install them.

14. Remove all the files inside the "components" folder: NuxtLogo.vue and Tutorial.vue.

15. In the `index.vue` page (under the "pages" directory) replace the entire file contents with this code:

```
<template>
  <!-- easy components usage, already shipped with bootstrap css-->
  <b-container>
    <!-- try to remove :fields=”fields” to see the magic -->
    <b-table striped over :items="students" :fields="fields" />
  </b-container>
</template>

<script>
export default {
  data () {
    return {
      fields: ['username', 'name', 'email'],
      students: []
    }
  },
  created () {
    this.$axios.$get('/api/students')
      .then((students) => {
        this.students = students
      })
  }
}
</script>

<style></style>
```

16. Stop the NUXT server in the terminal (Ctrl + C). We are going to edit the NUXT configuration file.

17. Open the `nuxt.config.js` file (it is in the root of the academics-client project). In this file, find an array called "modules". Ensure that it has the "`@nuxtjs/axios`" module (if you have chosen correctly all the answers on the app creation, you should have already this module added in the modules array). If not present, install it with the command `npm i nuxtjs/axios` and **then** add it to the array.

18. In order to configure an "alias" for the path `http://localhost:8080/academics/api/<the-rest-of-the-url>`, you can add a proxy that tells Axios to do this for you. To do this, add these settings:

```
  modules: [
    // Doc: https://bootstrap-vue.js.org/docs/
    'bootstrap-vue/nuxt', # if you choosed Bootstrap Vue...
    '@nuxtjs/axios', # if you enabled axios module upon the app generation
  ],
  axios: {
    proxy: true,
    credentials: true
  },

  proxy: {
    '/api/': {
      target: 'http://localhost:8080/academics/api/',
      pathRewrite: {
        '^/api/': ''
      }
    }
  },
```

As you can see, we define the relative path "`/api/`" and map it to be replaced by the target URL `http://localhost:8080/academics/api/`.
Now, the code you saw at `index.vue`:

```
// index.vue
// ...
  created () {
    this.$axios.$get('/api/students').then((students) => { this.students = students })
  }
```

will be translated to:

```
 created () {
    this.$axios.$get('http://localhost:8080/academics/api/students')
      // ...
 }
```

19. Finally, we need to allow communications between two servers (CORS). If you do not know what CORS is, please check the MDN web docs. To do that, we need to add a filter that intercepts the incoming requests and allows the communications on the requested server (our Java EE enterprise application deployed in Wildfly). Return to the Java EE project and create a new file, under a package named "providers", named `CorsFilter`, and paste the following code:

```java
package pt.ipleiria.estg.dei.ei.dae.academics.providers;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.ext.Provider;
import java.io.IOException;

@Provider
public class CorsFilter implements ContainerResponseFilter {
    @Override
    public   void   filter(ContainerRequestContext   requestContext,   ContainerResponseContext
responseContext) throws IOException {
        // Allows this server to be called by any other server.
        // For development, it can be set to '*'.
        // In a production environment, it's a security risk.
        // You may only specify the servers you allow to communicate.
        // Eg: to communicate only with NUXT ("Access-Control-Allow-Origin", "localhost:3000")
        responseContext.getHeaders().add("Access-Control-Allow-Origin", "*");

        // important to pass in the preflight browser request
        // note: you enabled this option in the nuxt.config.js (NUXT Web App)
        // axios: {
        //   proxy: true, # tells axios to use a proxy
        //   credentials: true # CORS
        // },
        responseContext.getHeaders().add("Access-Control-Allow-Credentials", "true");

        // defines what headers you authorize that can be present in a request
        //   responseContext.getHeaders().add("Access-Control-Allow-Headers","origin,   content-
type, accept, authorization");
        // defines the verbs you authorize
        responseContext.getHeaders().add("Access-Control-Allow-Methods","GET,     POST,     PUT,
DELETE, OPTIONS, HEAD");
    }
}
```

20. Re-deploy the Java EE academics management app **and** then re-start the NUXT server (`$ npm run dev`). Point your browser to `http://localhost:3000` and you should see a table with the students list (you may need to wait a few minutes before the Wildfly container is ready to serve your JavaEE application).

Worksheet 2 Documentation:

We strongly recommend you read the basic documentation of the NUXT and other frameworks to better understand the concepts skipped here, that are not the focus of this course.

About Vue and its ecosystem:

- https://vuejs.org/
- https://nuxtjs.org/
- https://axios.nuxtjs.org/
- https://bootstrap-vue.js.org/

or other CSS framework you prefer... e.g.: check for Buefy based on Bulma. It is gaining adepts and it has a very intuitive usage approach - https://buefy.org/ (here's an example of a Vue CSS framework shipped with Bulma – https://github.com/vuejs/awesome-vue)

Important for your workgroup project: You have a curated list of great Vue-js-based projects you can integrate in your project that will speed up your frontend (Presentation Logic Layer) application development. This list has some libraries you can use for the frontend, so that you spend less hours developing the "beauty" of the website, focusing mainly on the robustness and quality of your Enterprise Application as a whole. That is what, in the end, matters and is expected in an enterprise context.

About JPQL:

- https://javaee.github.io/tutorial/persistence-querylanguage.html

About REST

- https://developer.mozilla.org/en-US/docs/Glossary/REST
- https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/
- https://www.restapitutorial.com/
- https://restfulapi.net/
- https://restfulapi.net/resource-naming/

About CORS

- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- https://www.codecademy.com/articles/what-is-cors