Desenvolvimento de Aplicações Empresariais – 2021-22-1S

Engenharia Informática – 3.º ano – Ramo SI

Worksheet 8

Topics: Client login and logout forms, Documents upload/download.

Client login and logout forms

1. In the last worksheet, we implemented the authentication technology using JAX-RS REST web services and JWT tokens. Now, we need to create a Vue.js login page for our users to login. In the NUXT project, add both `auth` and `toast` modules by running these commands in a Terminal window, in your NUXT project root:

   ```
   $ npm i --save @nuxtjs/auth
   $ npm i --save @nuxtjs/toast # if not installed yet
   ```

2. Activate the Vue Store. This step is required, because the auth module uses the store to save the user data. For that, you only need to create an empty file called "index.js" under the folder "store", that already exists in your NUXT project.
3. Open you NUXT project in IntelliJ and configure the `nuxt.config.js` file by adding the `auth` and `toast` modules.

   ```
   modules: [
         'bootstrap-vue/nuxt',
         '@nuxtjs/axios',
         '@nuxtjs/toast',
         '@nuxtjs/auth'
   ],
   ```

4. Add new entries in the `nuxt.config.js` file:

   ```
   ssr: false, // Disable Server Side rendering

   // Auth module configuration (https://auth.nuxtjs.org/)
   auth: {
     redirect: {
       login: '/auth/login',
       logout: '/',
       home: '/'
     },
     watchLoggedIn: true,
     strategies: {
       local: {
         endpoints: {
           login: {
             url: '/api/auth/login',
   ```

```
          method: 'post',
          propertyName: 'token'
        },
        logout: false,
        user: {
          url: '/api/auth/user',
          method: 'get',
          propertyName: ''
        }
      },
      // tokenRequired: true, -> default
      // tokenType: 'bearer' -> default
    }
  }
},

router: {
  middleware: [
    'auth'
  ]
},
```

5. Create a new directory under pages, called `auth` and a new Vue.js component called `login`. In that page, create the code that shows a login form and authenticates the user. Please, take a look at the documentation of the auth module here.

```
// An example of a login form for pages/auth/login.vue:
```

```
<template>
  <b-container>
    <h3>Login into Academics Management</h3>
    <b-form @submit.prevent="onSubmit" @reset="onReset">
      <b-form-group label="Username" description="Enter your username">
        <b-input
          name="username"
          placeholder="Your username"
          v-model.trim="username"
          required />
      </b-form-group>
      <b-form-group label="Password" description="Enter your password">
        <b-input
          name="password"
          type="password"
          placeholder="Your password"
          v-model="password"
          required />
      </b-form-group>
      <b-button type="reset" class="btn-warning">Reset</b-button>
      <b-button type="submit" class="btn-success">Submit</b-button>
    </b-form>
  </b-container>
</template>

<script>
export default {
  auth: false,
  data() {
    return {
      username: null,
      password: null
    }
  },
  methods: {
    onSubmit() {
      let promise = this.$auth.loginWith('local', {
        data: {
          username: this.username,
```

```
          password: this.password
        }
      })

      promise.then(() => {
        this.$toast.success('You are logged in!').goAway(3000)

        // check if the user $auth.user object is set
        console.log(this.$auth.user)

        // TODO redirect based on the user role
        // eg:
        if (this.$auth.user.groups.includes('Teacher')) {
          this.$router.push('/students')
        } else if (this.$auth.user.groups.includes('Student')) {
          this.$router.push('/students/' + this.username)
        }
      })

      promise.catch(() => {
        this.$toast.error('Sorry,   you   cant   login.   Ensure   your   credentials   are
correct').goAway(3000)
      })
    },
    onReset() {
      this.username = null
      this.password = null
    }
  }
}
</script>
```

6. At the Java EE application, change the `LoginService.authenticateUser(…)` method to Consume JSON instead. Also, change the parameters to receive an `AuthDTO` object, with a username and password.
7. Run and test both Java EE and NUXT applications. You should be able to login.
8. Now, take a look at the `this.$auth.user` object. You can access the user type (Student, Teacher or Administrator) by checking this object. Depending on the user role, after login, redirect the user accordingly (see commented code above). For example, if you authenticate as a Student or a Teacher, redirect to your list of subjects. For an Administrator, you can redirect to the current home page that we've already developed, so that s/he can access whatever s/he wants (s/he can view/edit everything: students, teacher, courses and subjects).
9. To log out a user, you just need to call `this.$auth.logout()`. This should be applied globally, only once, to avoid writing the same code repeatedly. To do so, we will change the default layout of our application, adding a <navbar/> that will have a button to do logout. In the file layouts/default.vue, remove the entire code and paste the following:

```
<template>
  <div id="app">
    <b-navbar toggleable="lg">
      <b-navbar-brand href="#">Academics</b-navbar-brand>
      <b-navbar-toggle target="nav-collapse"></b-navbar-toggle>
      <b-collapse id="nav-collapse" is-nav>
        <b-navbar-nav>
          <li class="nav-item">
            <nuxt-link class="nav-link" to="students">Students</nuxt-link>
          </li>
        </b-navbar-nav>
        <!-- Right aligned nav items -->
        <b-navbar-nav class="ml-auto">
          <b-nav-item-dropdown v-if="$auth.loggedIn" right>
            <!-- Using 'button-content' slot -->
            <template #button-content>
              <em>{{ $auth.user.sub }}</em>
            </template>
            <b-dropdown-item @click.prevent="signOut">Sign Out</b-dropdown-item>
          </b-nav-item-dropdown>
          <li class="nav-item" v-else>
            <nuxt-link class="nav-link" to="/auth/login">Sign In</nuxt-link>
          </li>
        </b-navbar-nav>
      </b-collapse>
    </b-navbar>
    <main>
      <Nuxt/>
    </main>
  </div>
</template>
<script>
export default {
  methods: {
    signOut() {
      this.$auth.logout()
      this.$router.push('/')
    }
  }
}
</script>
```

Documents upload/download

10. Create the Document entity with attributes `filepath`, `filename` (both strings) and `student` (`Student`); Add the `id` attribute, which will be the automatically generated primary key (use the `@GeneratedValue(strategy = GenerationType.AUTO)` annotation); Add a list of documents to the `Student` entity and the respective getters, setters and methods to add and remove documents; Add the necessary mappings in both the `Document` and `Student` entities; Add constructors, getters, setters and the following named query to the `Document` entity:

```
@NamedQuery(name = "getStudentDocuments", query = "SELECT doc FROM Document doc
WHERE doc.student.username = :username")
```

11. Create the `DocumentBean` stateless EJB and code the following methods:

```
public void create(String username, String filepath, String filename){…}
public Document findDocument(int id){…}
public List<Document> getStudentDocuments(String username){…}
```

12. Create the `DocumentDTO` with the `id`, `filepath` and `filename` attributes;

13. Replace your pom.xml with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>pt.ipleiria.estg.dei.ei.dae</groupId>
    <artifactId>academics</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>academics</name>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.compiler.source>1.8</maven.compiler.source>
        <junit.version>5.7.1</junit.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>8.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>eclipselink</artifactId>
            <version>2.7.9</version>
        </dependency>
        <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-servlet-initializer</artifactId>
            <version>4.7.3.Final</version>
        </dependency>
        <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-jackson2-provider</artifactId>
            <version>4.7.2.Final</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>javax.mail</groupId>
            <artifactId>javax.mail-api</artifactId>
            <version>1.6.2</version>
        </dependency>
```

```
        <dependency>
            <groupId>com.nimbusds</groupId>
            <artifactId>nimbus-jose-jwt</artifactId>
            <version>9.15.2</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-multipart-provider</artifactId>
            <version>5.0.0.Final</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.1</version>
                <configuration>
                    <warName>academics</warName>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

14. Create the `DocumentsService` class within the `ws` package as below:

```
@Path("documents")
public class DocumentsService {

    @EJB
    private StudentBean studentBean;

    @EJB
    private DocumentBean documentBean;

    @POST
    @Path("upload")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    public Response upload(MultipartFormDataInput input) throws MyEntityNotFoundException,
IOException {
        Map<String, List<InputPart>> uploadForm = input.getFormDataMap();

        // Get file data to save
        String username = uploadForm.get("username").get(0).getBodyAsString();
        Student student = studentBean.findStudent(username);

        if(student == null) {
            throw new MyEntityNotFoundException("Student with username " + username + " not
found.");
        }

        List<InputPart> inputParts = uploadForm.get("file");

        for (InputPart inputPart : inputParts) {
            try {
                MultivaluedMap<String, String> header = inputPart.getHeaders();
                String filename = getFilename(header);

                // convert the uploaded file to inputstream
```

```
                InputStream inputStream = inputPart.getBody(InputStream.class, null);

                byte[] bytes = IOUtils.toByteArray(inputStream);

                String path = System.getProperty("user.home") + File.separator + "uploads";
                File customDir = new File(path);

                if (!customDir.exists()) {
                    customDir.mkdir();
                }

                String filepath =  customDir.getCanonicalPath() + File.separator + filename;
                writeFile(bytes, filepath);

                documentBean.create(student.getUsername(), path, filename);

                return Response.status(200).entity("Uploaded file name : " +
filename).build();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    @GET
    @Path("download/{id}")
    @Produces(MediaType.APPLICATION_OCTET_STREAM)
    public Response download(@PathParam("id") Integer id) throws MyEntityNotFoundException {
        Document document = documentBean.findDocument(id);

        File fileDownload = new File(document.getFilepath() + File.separator +
document.getFilename());

        ResponseBuilder response = Response.ok((Object) fileDownload);
        response.header("Content-Disposition", "attachment;filename=" +
document.getFilename());

        return response.build();
    }

    @GET
    @Path("{username}")
    @Produces(MediaType.APPLICATION_JSON)
    public List<DocumentDTO> getDocuments(@PathParam("username") String username) throws
MyEntityNotFoundException {
        Student student = studentBean.findStudent(username);
        if(student == null) {
            throw new MyEntityNotFoundException("Student with username " + username + " not
found.");
        }
        return documentsToDTOs(documentBean.getStudentDocuments(username));
    }

    @GET
    @Path("{username}/exists")
    public Response hasDocuments(@PathParam("username") String username) throws
MyEntityNotFoundException {
        Student student = studentBean.findStudent(username);
        if(student == null) {
            throw new MyEntityNotFoundException("Student with username " + username + " not
found.");
        }

        return Response.status(Response.Status.OK).entity(new
Boolean(!student.getDocuments().isEmpty())).build();
    }
```

```
    DocumentDTO toDTO(Document document) {
        return new DocumentDTO(
                document.getId(),
                document.getFilepath(),
                document.getFilename());
    }

    List<DocumentDTO> documentsToDTOs(List<Document> documents) {
        return documents.stream().map(this::toDTO).collect(Collectors.toList());
    }

    private String getFilename(MultivaluedMap<String, String> header) {
        String[] contentDisposition = header.getFirst("Content-Disposition").split(";");
        for (String filename : contentDisposition) {
            if ((filename.trim().startsWith("filename"))) {
                String[] name = filename.split("=");
                String finalFileName = name[1].trim().replaceAll("\"", "");
                return finalFileName;
            }
        }
        return "unknown";
    }
    private void writeFile(byte[] content, String filename) throws IOException {
        File file = new File(filename);

        if (!file.exists()) {
            file.createNewFile();
        }
        FileOutputStream fop = new FileOutputStream(file);
        fop.write(content);
        fop.flush();
        fop.close();
        System.out.println("Written: " + filename);
    }
}
```

15. Run your application (`make monitor`).

16. Test the file upload and download services with the following HTTP requests:

```
###
POST http://localhost:8080/academics/api/documents/upload HTTP/1.1
Authorization: Bearer <replace by the authentication token here>
Content-Type: multipart/form-data; boundary=boundary

--boundary
Content-Disposition: form-data; name="file"; filename="<filename goes here>"

< <full path including filename goes here – do not remove the first '<' sign>

--boundary
Content-Disposition: form-data; name="username"

<username of the token goes here>

--boundary

###
GET http://localhost:8080/academics/api/documents/download/1 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
```

17. In your NUXT project, create a form page that allows your user to upload a file. You need to send the request with the media type "`multipart/form-data`". Also, you need to create a `FormData` object, to send the binary file using the library `axios`, already installed in your NUXT project.

An example could be a page in `students/upload.vue` with the following code:

```
<template>
  <form @submit.prevent="upload">
    <!-- Styled -->
    <b-form-file
      v-model="file"
      :state="hasFile"
      placeholder="Choose a file or drop it here..."
      drop-placeholder="Drop file here..."
    ></b-form-file>
    <div class="mt-3">
      Selected file: {{ file ? file.name : '' }}
    </div>

    <nuxt-link class="btn btn-link" :to="`/students/${this.username}`">Back</nuxt-link>

    <b-button type="submit" :disabled="!hasFile">Upload</b-button>
  </form>
</template>

<script>
export default {
  auth: false,
  data() {
    return {
      username: this.$auth.user.sub,
      file: null
    }
  },

  computed: {
    hasFile () {
      return this.file != null
    },
    formData () {
      let formData = new FormData()

      formData.append('username', this.$auth.user.sub)

      if (this.file) {
        formData.append('file', this.file)
      }

      return formData
    }
  },

  methods: {
    upload() {
      if (!this.hasFile) {
        return
      }

      let promisse = this.$axios.$post('/api/documents/upload', this.formData, {
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      })

      promisse.then(() => {
        this.$toast.success('File uploaded!').goAway(3000)
      })
      promisse.catch(() => {
        this.$toast.error('Sorry, could no upload file!').goAway(3000)
      })
    }
  }
}
```

```
</script>
```

18. Create the download page too, that allows the user to download the previous uploaded file. Let's change the page students/_username/index.vue to allow to download a document of some student. Before paste the code below, make sure your StudentDTO returns a list of documents (DocumentDTOs, of course).

```
<template>
  <b-container>
    <h4>Student Details</h4>
    <p>Username: {{ student.username }}</p>
    <p>Name: {{ student.name }}</p>
    <p>Email: {{ student.email }}</p>
    <p>Course: {{ student.courseName }}</p>

    <h4>Subjects</h4>
    <b-table v-if="subjects.length" striped over :items="subjects" :fields="subjectFields" />
    <p v-else>No subjects enrolled.</p>

    <h4>Documents</h4>
    <b-table v-if="documents.length" striped over :items="documents"
   :fields="documentsFields">
      <template v-slot:cell(actions)="row">
        <b-btn class="btn btn-link" @click.prevent="download(row.item)"
   target="_blank">Download</b-btn>
      </template>
    </b-table>
    <p v-else>No documents.</p>

    <nuxt-link to="/students">Back</nuxt-link>
     
    <nuxt-link :to="`/students/${this.username}/send-email`">Send e-mail</nuxt-link>
     
    <nuxt-link :to="`/students/upload`">Upload</nuxt-link>
  </b-container>
</template>

<script>
export default {
  data() {
    return {
      student: {},
      subjectFields: [ 'code', 'name', 'courseCode', 'courseYear', 'scholarYear' ],
      documentsFields: [ 'filename', 'actions' ],
    }
  },
  computed: {
    username() {
      return this.$route.params.username
    },
    subjects() {
      return this.student.subjects || []
    },
    documents() {
      return this.student.documents || []
    }
  },
  created() {
    this.$axios.$get(`/api/students/${this.username}`)
      .then((student) => {
        this.student = student || {}
      })
  },

  methods: {
    download(fileToDownload) {
```

```
      const documentId = fileToDownload.id

      this.$axios.$get('/api/documents/download/' + documentId, { responseType:
   'arraybuffer'})
        .then(file => {
          const url = window.URL.createObjectURL(new Blob([file]))
          const link = document.createElement('a')
          link.href = url
          link.setAttribute('download', fileToDownload.filename)
          document.body.appendChild(link)
          link.click()
        })
    }
  }
}
</script>
```

Bibliography

Java EE Tutorial 8 (all of it)

NUXT