
Desenvolvimento de Aplicações Empresariais – 2021-22-1S

Engenharia Informática – 3.º ano – Ramo SI

Worksheet 1

Topics: Java EE first Cup ☺ – Configuring a new Java EE project using a Wildfly Web Server and a PostgreSQL database on top of Docker and Docker Compose.

In the laboratory classes of DAE we will develop an enterprise application for academic management. In this initial worksheet, please execute the following steps:

1. Install Docker Desktop: <https://www.docker.com/products/docker-desktop>

Optional (for Windows users **only**):

You can install now or skip this step and install it on the step 3.

2. If you don't have the OpenJDK 16 (HotSpot) installed on your machine, you can download it and follow the steps described on the [AdoptOpenJDK](https://adoptopenjdk.net/) webpage.
Choose version 16 (latest, last option). For the JVM choose HotSpot (first option).

Optional (for Windows users **only**):

You can install now or skip this step and install it on the step 3.

3. Download Apache Maven CLI. We will use this to build the war files.

- If you are a **Mac** user, just hit:

```
$ brew install maven
```

- If you are a **Linux** user (Ubuntu-based):

```
$ sudo apt install maven
```

- If you use **Windows**, with administration privileges:

- o Install Chocolatey: <https://chocolatey.org/install>

- o Install packages:

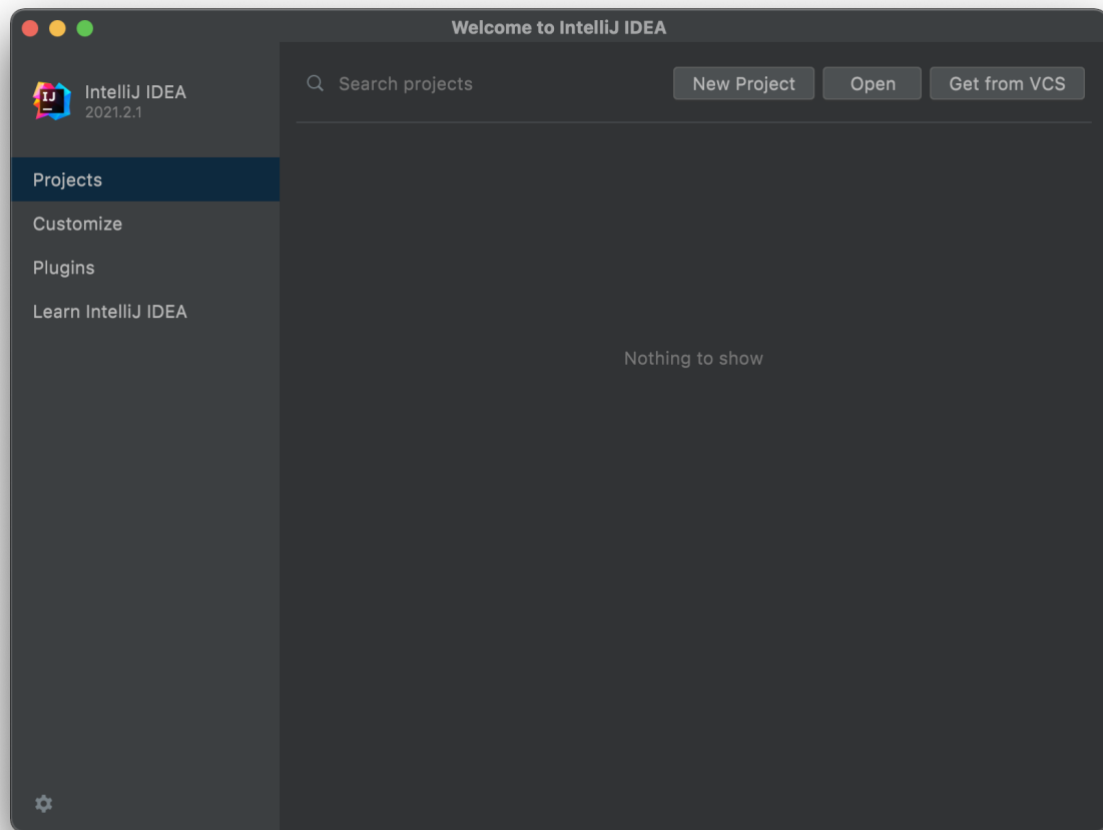
```
$ choco install maven make
```

NOTE: If you are a Windows user **and** skipped previous steps:

```
$ choco install docker-desktop adoptopenjdk16 maven make
```

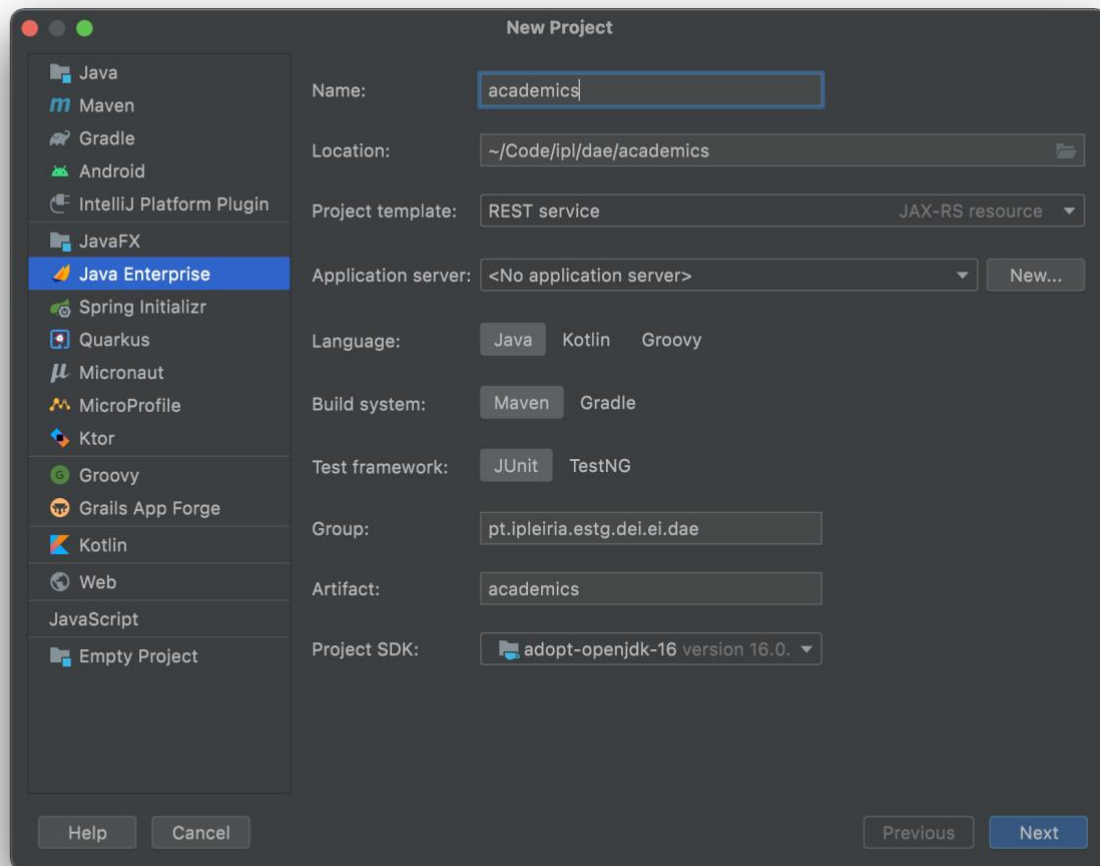
4. [Download](#) and install the **latest** version of IntelliJ IDEA **Ultimate** edition;
5. Clone the project: <https://github.com/dmp593/docker-wildfly-postgres> into a folder on your machine you may choose.

6. Open the IntelliJ IDEA Ultimate and create a new project:
- Click on “New Project”

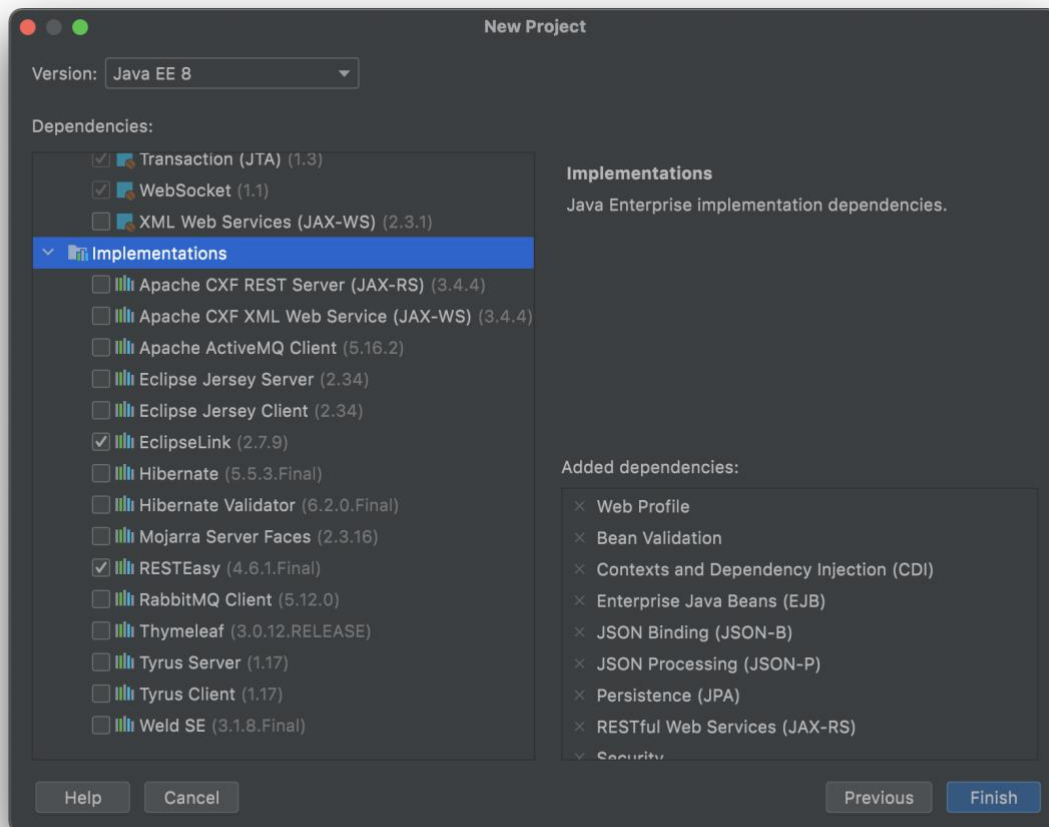
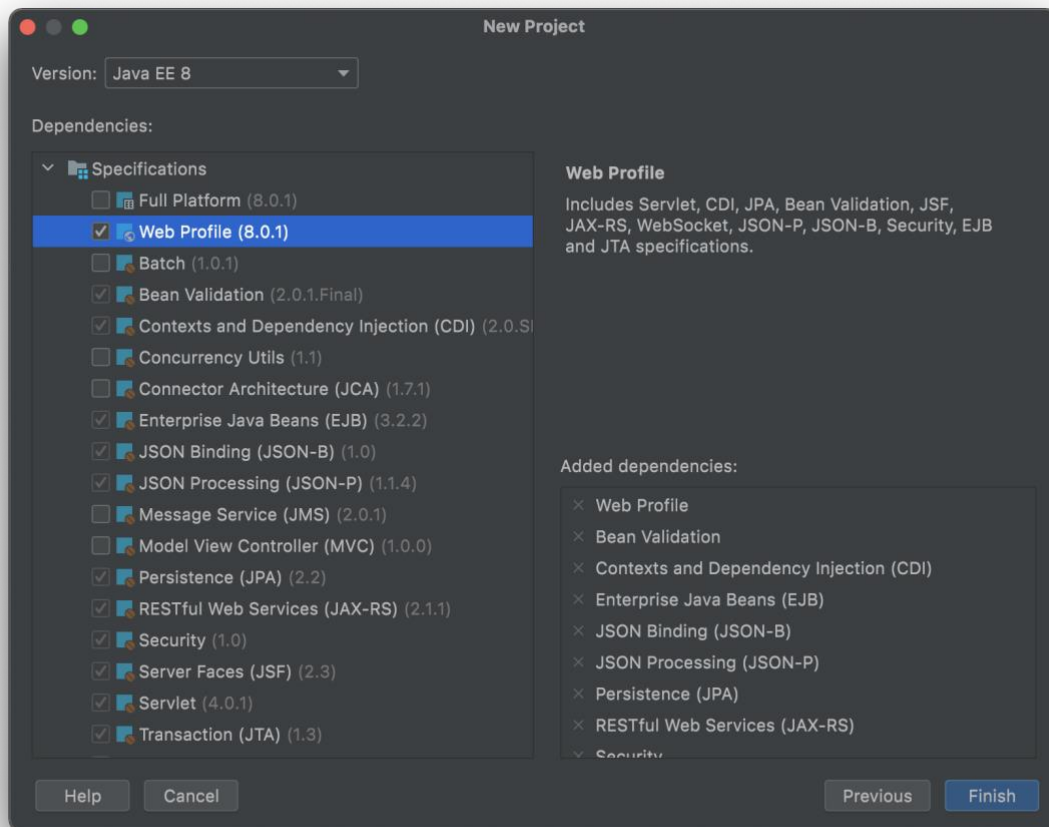


- Configure the options:
 - Name: **academics**
 - Location: (Choose one you prefer)
 - Project template: **REST service**
 - Application Server: (Leave empty)
 - Language: **Java**
 - Build system: **Maven**
 - Test framework: **Junit**
 - Group: **pt.ipleiria.estg.dei.ei.dae**
 - Artifact: **academics**
 - Project SDK: **adopt-openjdk-16**

And click “Next”.

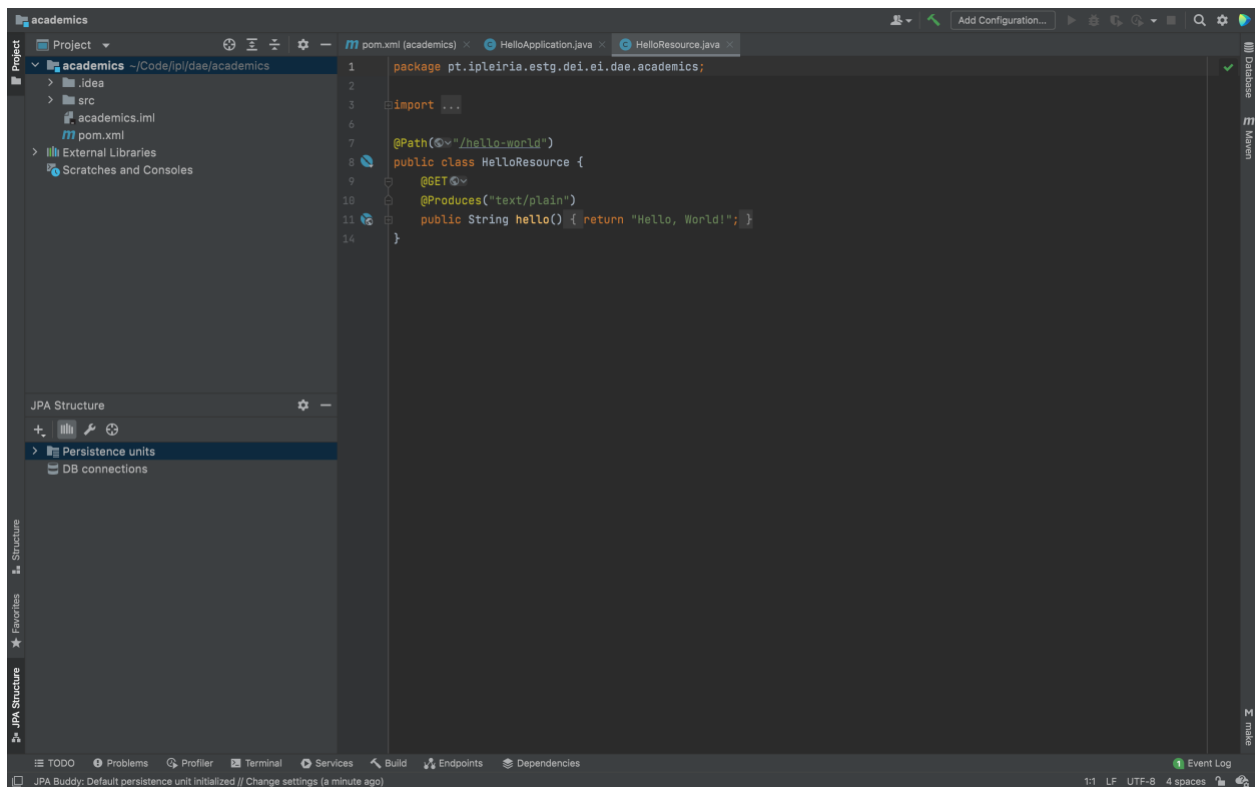


- On the “Specifications” select “Web Profile” (it should select a bundle of options for you). And then, on “Implementations” choose “EclipseLink” and “REST Easy”.



- Finally, click “Finish”.

7. Now you should end up with something like this:



8. Let's first alter some configurations of our project.

Open the pom.xml file and change this section:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.3.1</version>
</plugin>
```

To include this:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.3.1</version>
  <configuration>
    <outputDirectory>target</outputDirectory>
    <warName>academics</warName>
  </configuration>
</plugin>
```

This will indicate that the war file that we want to build must be placed inside that directory, and that the filename of our war file will be called “academics.war”.

9. Open your File Explorer, go to the project you've cloned at step 5 and copy the files and folders:

- Dockerfile
- docker-compose.yaml
- scripts/
- .env.example
- Makefile

into your project root directory.

10. Yet inside the project root directory, create another file called “.env” (**with no extension**) and copy/paste the values from “.env.example”.

11. Edit the .env file and replace the variable values to match this configuration:

```
DATASOURCE_JNDI=java:/AcademicsDS  
DATASOURCE_NAME=AcademicsDS
```

```
DB_USER=postgres  
DB_PASS=dbsecret  
DB_HOST=db  
DB_PORT=5432  
DB_NAME=academics
```

```
ECLIPSELINK_VERSION=2.7.9
```

```
POSTGRES_DRIVER_VERSION=42.2.23
```

```
WILDFLY_ADMIN_PASSWORD=wildsecret
```

Our project uses a Wildfly Web Server container and a PostgreSQL database container, on top of Docker and Docker Compose. Everything is already configured and ready to be integrated with your Java EE project. Your “.env” contains all the configurations you need to set, in order to run the containers and to accord with your academics application.

These configurations will create a database called “academics”, with a default user called “postgres” and a password “dbsecret”. The wildfly admin password is set to be “wildsecret”. This will be helpful to access the Wildfly Admin Console later.

Finally, the datasource is configured for your java project, with a datasource JNDI “java:/AcademicsDS”. Your persistence.xml file must reference this datasource JNDI in order to access the database. Finally, you can also define the eclipselink and postgres driver versions.

12. Return to your project in IntelliJ IDEA Ultimate.

13. Go to “src/main/resources/META-INF” folder and open the “persistence.xml” file.

Remove the tag:

```
<persistence-unit> ... </persistence-unit>
```

And **replace** it with:

```
<persistence-unit name="AcademicsPersistenceUnit">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <jta-data-source>java:/AcademicsDS</jta-data-source>
  <properties>
    <property
      name="javax.persistence.schema-generation.database.action"
      value="drop-and-create"/>
  </properties>
</persistence-unit>
```

This configures our project to use the PostgreSQL database, already configured in wildfly with a datasource name “java:/AcademicsDS”. The configuration (host, port, db, user, pass, etc.) is already defined in that datasource. We only need to reference the datasource, to use that configuration on our project.

14. Now we are ready to start. You can try to start the containers and check if they are running. Open a terminal at the project root directory (you can use the “terminal” tab on your IDE) and hit the command:

```
$ make up
```

Containers should start running.

You can check their status any time with:

```
$ docker compose ps
```

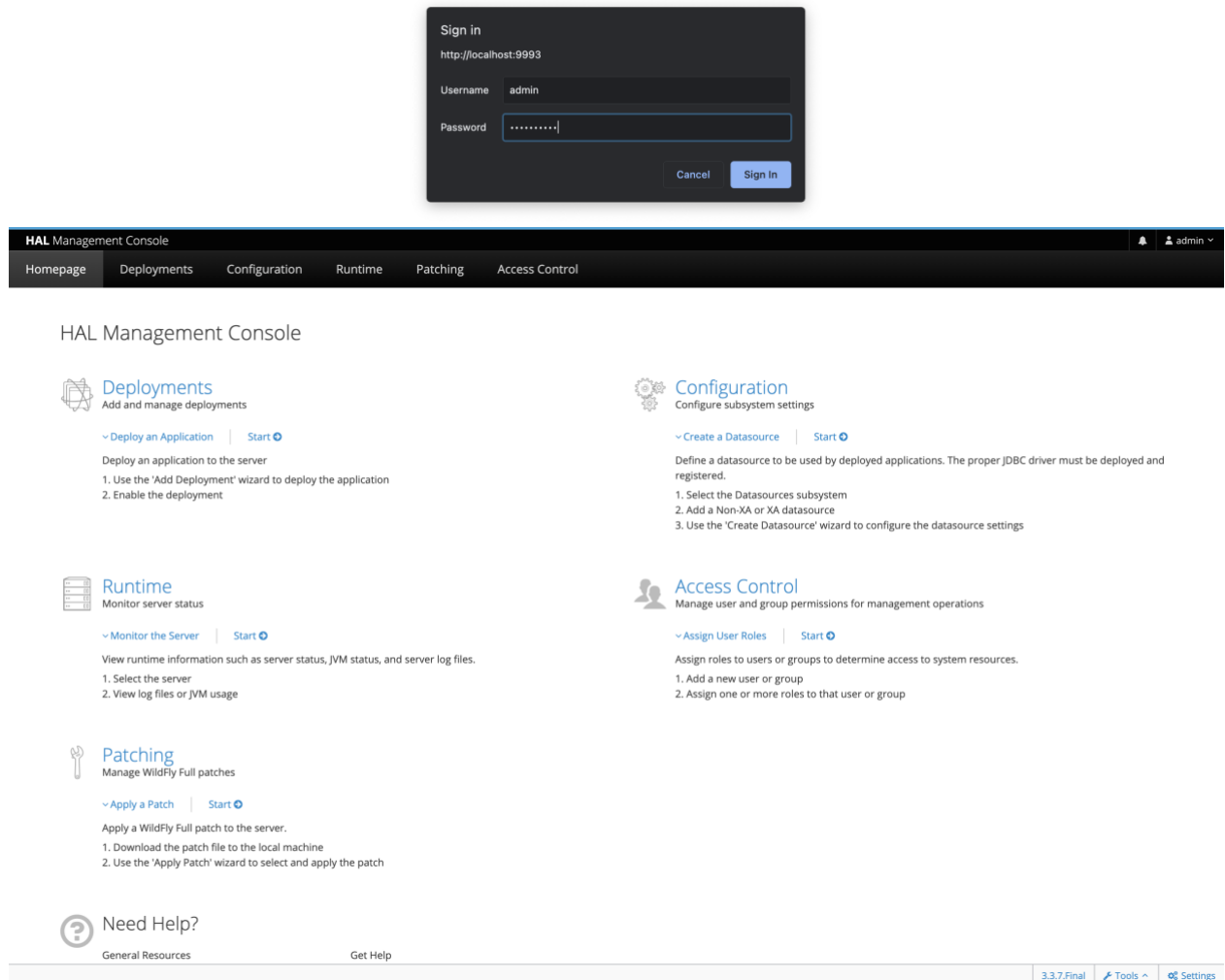
Access the URL: localhost:8080

(When you start the container, may need to wait more less 1 or 2 minutes in order to the webserver can serve the page)

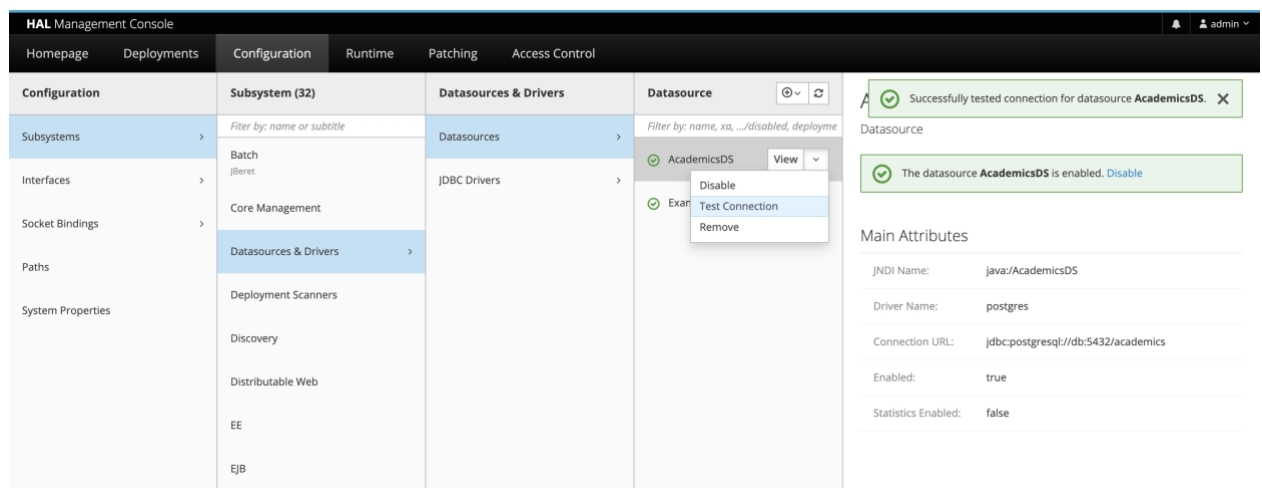
You should see the following welcome page:



You can try access the Administration Console by clicking on the link you or just changing the URL to: localhost:9990. Test if the administration is also running and you can access with the credentials you've defined in your .env file.



Finally, check if the datasource was created and it's working. Click on “Configuration” link. Select “Subsystems > Datasources & Drivers > AcademicsDS”. And click on test, to check it's working.



15. Great! Now that you’ve started the containers, and the configurations are ok, you can start developing. We will create a student and save on the database, in order to test that our project can write into the database.

16. Create a singleton EJB named `ConfigBean` in the `ejbs` package, on the source (src) directory of your project. This EJB should have the following Java EE annotations right above the class declaration of the EJB:

- `@Singleton` // type and name for the EJB;
- `@Startup` //EJB will be automatically instantiated once the application is deployed onto the server.

NOTE: if any of these annotations cannot be recognized by IntelliJ, resolve it’s Maven dependencies, and then reload your project’s Maven (IntelliJ should suggest this to you);

17. Create a method `populateDB()`, which should be called by the server right after it instantiates the `ConfigBean` EJB (annotate the method with the `@PostConstruct` annotation). For now, this method should just print some welcome message in the console (we will be back here later). The full code for this EJB should be similar to this one:

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

import javax.annotation.PostConstruct;
import javax.ejb.Singleton;
import javax.ejb.Startup;

@Startup
@Singleton
public class ConfigBean {

    @PostConstruct
    public void populateDB() {
        System.out.println("Hello Java EE!");
    }
}
```

18. To deploy your application into the webserver, access to the terminal (either a separated console application or using the integrated one in IntelliJ) and hit the command inside the project root directory:

```
$ make deploy
```

(if you get a “Duplicate mount point” error, just try to execute the `make down` (equivalent to `docker compose down`) and then `make deploy` afterwards)

19. Run the application and pay attention to the Wildfly’s server log (you’ll be looking into it most of the time in this course). Be sure that the application is deployed, and that you can see the “Hello Java EE!” output from the `populateDB()` method; You can check the logs via the shortcut command `make logs` or directly through docker compose:

```
$ docker compose -f logs webserver
```

20. Create the `Student` entity in the `entities` package (use the `@Entity` annotation above the class declaration). It should have the following attributes: `username` (which is the entity’s ID – use the `@Id` annotation above this attribute), `password`, `name` and `email`. The entity must have at least a default empty constructor and a getter and setter method for each attribute. Also write a constructor that receives and sets all attributes’ values;

21. Create the stateless Enterprise Java Bean (EJB) (`@Stateless` annotation right above the class declaration) named `StudentBean` in the `ejbs` package and write the method `create(...)` with all student attributes as parameters. This method should create and persist a student in the database. Do not forget to declare and use an `EntityManager` in the EJB (use the `@PersistenceContext` annotation right above the entity manager declaration).

22. Modify the populated method of the `ConfigBean` EJB so that it creates students and persists them in the database. You need to call the `create(...)` method of the `StudentBean` EJB. To do so, you need to declare a `StudentBean` variable in the `ConfigBean` EJB and annotate it with the `@EJB` annotation. Remember that the `populatedDB()` method is called by the server right after it instantiates the `ConfigBean` EJB (due to the `@PostConstruct` annotation).

23. Run the application (make deploy) and verify the server logs for any issue (make logs);

24. In order to see the generated data table, do the following:

```
$ make sql
```

Enter the password you've defined in your `.env` file.

Verify that there is a student at the student table, by hitting:

```
<db_name>=# SELECT * FROM student;
```

Notice that you can also see the database contents by configuring your datasource in IntelliJ:

- On the right side of the main editor, click on the “Database” tab to expand it;
- Click on + -> Data Source -> PostgreSQL;
- On the dialog box that opens:
 - Name: `academics`; # or other if you changed your `.env` file
 - User: `postgres`; # or other if you changed your `.env` file
 - Password: `dbsecret`; # or other if you changed your `.env` file
 - URL: `jdbc:postgresql://localhost:5432/academics`;
 - Leave the other fields as they are;
 - Click “Test Connection”;
 - Click “Ok”.
- Expand the `dae` datasource until you see the student table icon (if you cannot see it, click on the refresh button in the window toolbar).
Double click on the student table to show its contents.
- Click the “Download missing drivers” link if it is available in this form. s