POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO DEI Departamento Engenharia Informática

Desenvolvimento de Aplicações Empresariais – 2022-23-1S

Engenharia Informática – 3.º ano – Ramo SI

Worksheet 1

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

In the laboratory classes of EAD (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 webpage.
Choose version 18 (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:
  - Install Chocolatey: https://chocolatey.org/install
  - Install packages: `$ choco install maven make`

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

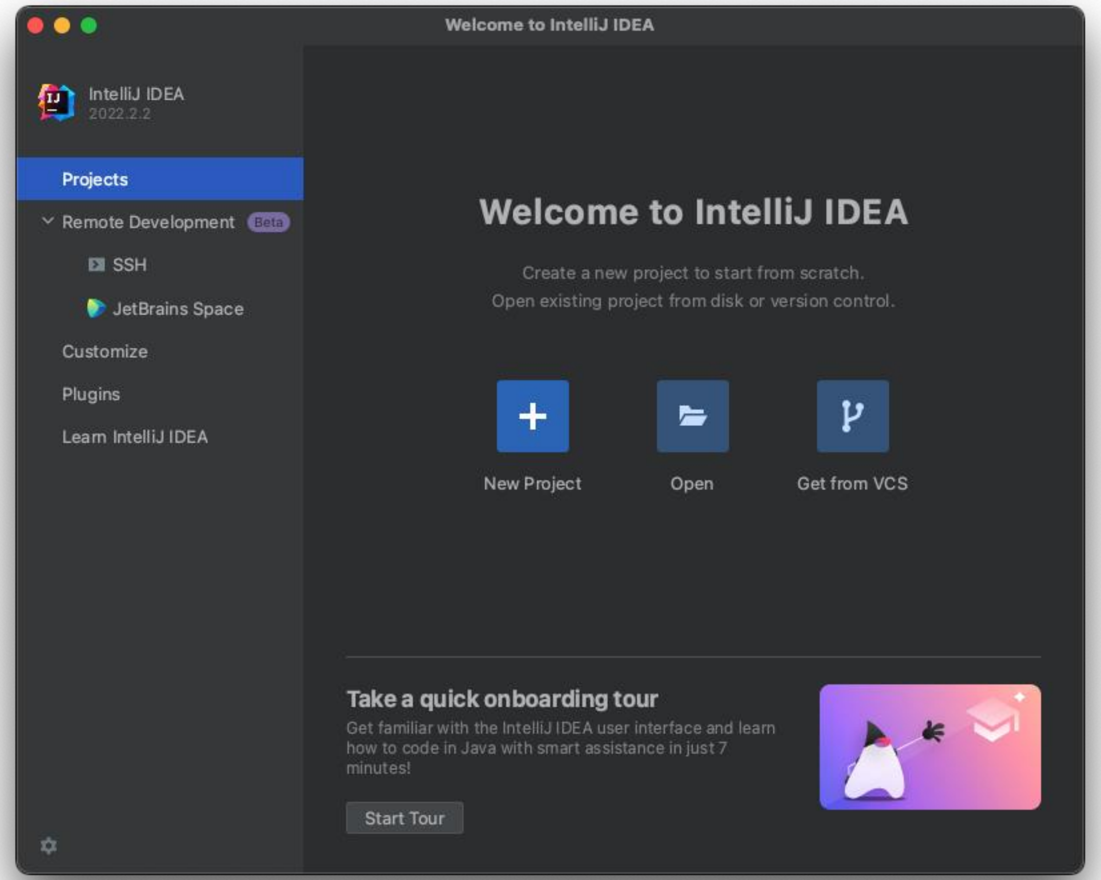
    ```
    $ choco install docker-desktop adoptopenjdk16 maven make
    ```

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.

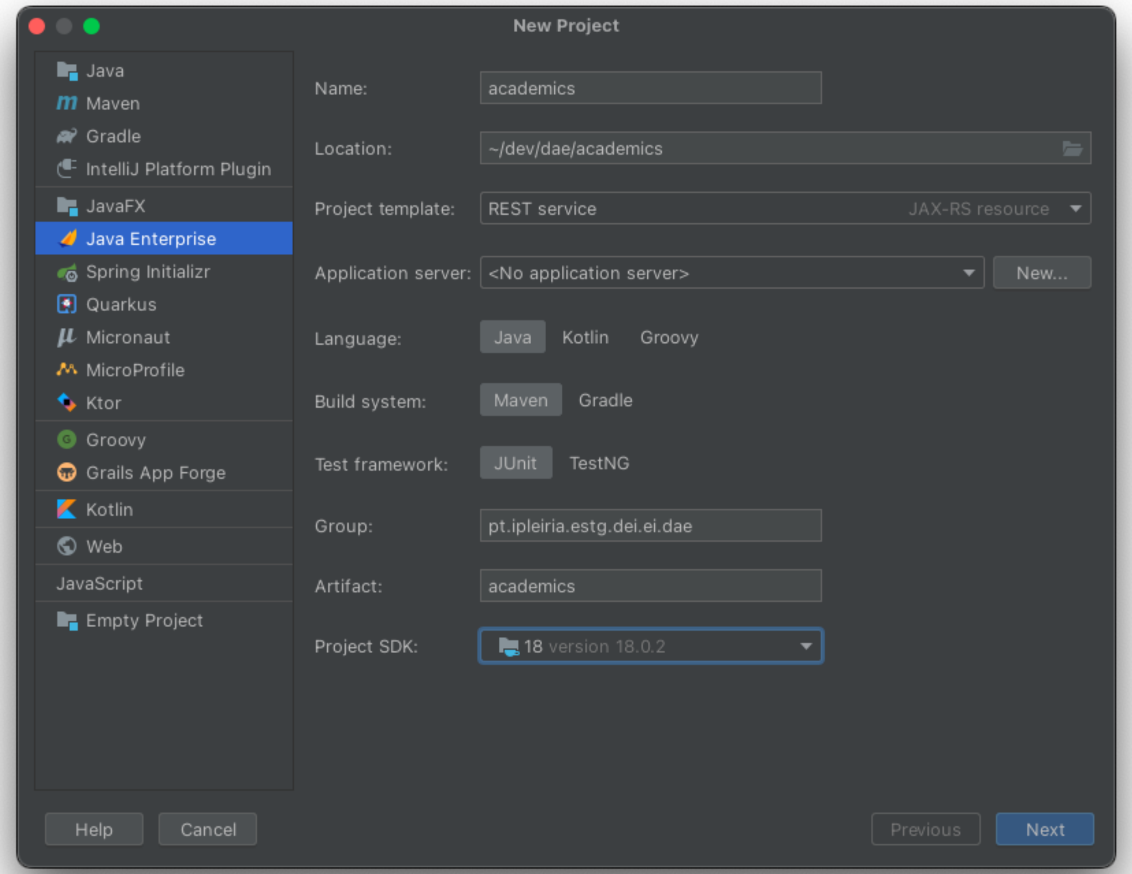
6. Open the IntelliJ IDEA Ultimate and create a new project:

- Click on "New Project"


Welcome to IntelliJ IDEA
IntelliJ IDEA
2022.2.2
Projects
Remote Development
Beta
SSH
JetBrains Space
Customize
Plugins
Learn IntelliJ IDEA
Welcome to IntelliJ IDEA
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
Take a quick onboarding tour
Get familiar with the IntelliJ IDEA user interface and learn how to code in Java with smart assistance in just 7 minutes!
Start Tour

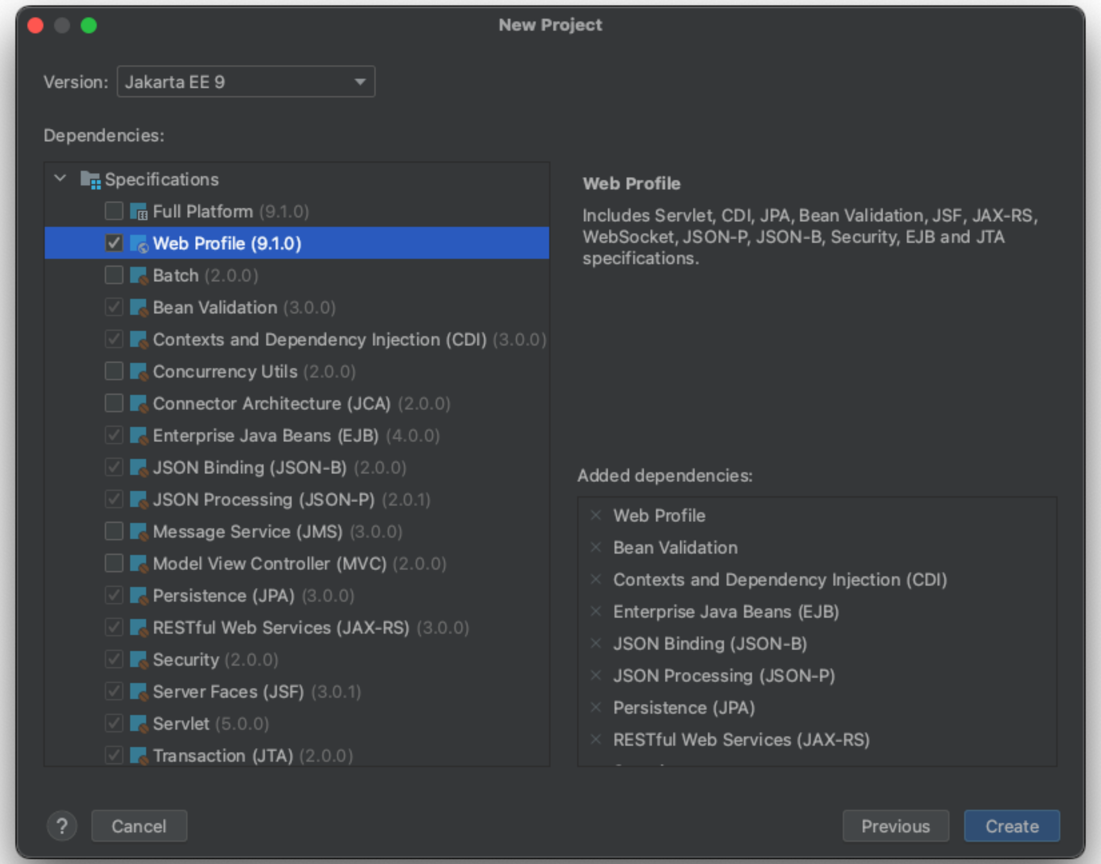
- Configure the options:
  Name: **academics**
  Location: (Choose one you prefer)
  Project template: **REST Service**
  Application Server: (Leave empty)
  Language: **Java**
  Build system: **Maven**
  Test framework: **Junit**
  Group: **pt.ipleiria.estg.dei.ei.dae**
  Artifact: **academics**
  Project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

  And click "Next".



Name: academics
Location: ~/dev/dae/academics
Project template: REST service JAX-RS resource
Application server: <No application server>
Language: [✓ Java] Kotlin Groovy
Build system: [✓ Maven] Gradle
Test framework: [✓ JUnit] TestNG
Group: pt.ipleiria.estg.dei.ei.dae
Artifact: academics
Project SDK: 18 version 18.0.2

- On the next screen, select “Jakarta EE 9” on the Version field, and “Web Profile” from “Dependencies→Specifications” (it should select a bundle of options for you). And then, on “Dependencies→Implementations” choose “Hibernate” and “Hibernate Validator”.



Version: Jakarta EE 9

Dependencies:

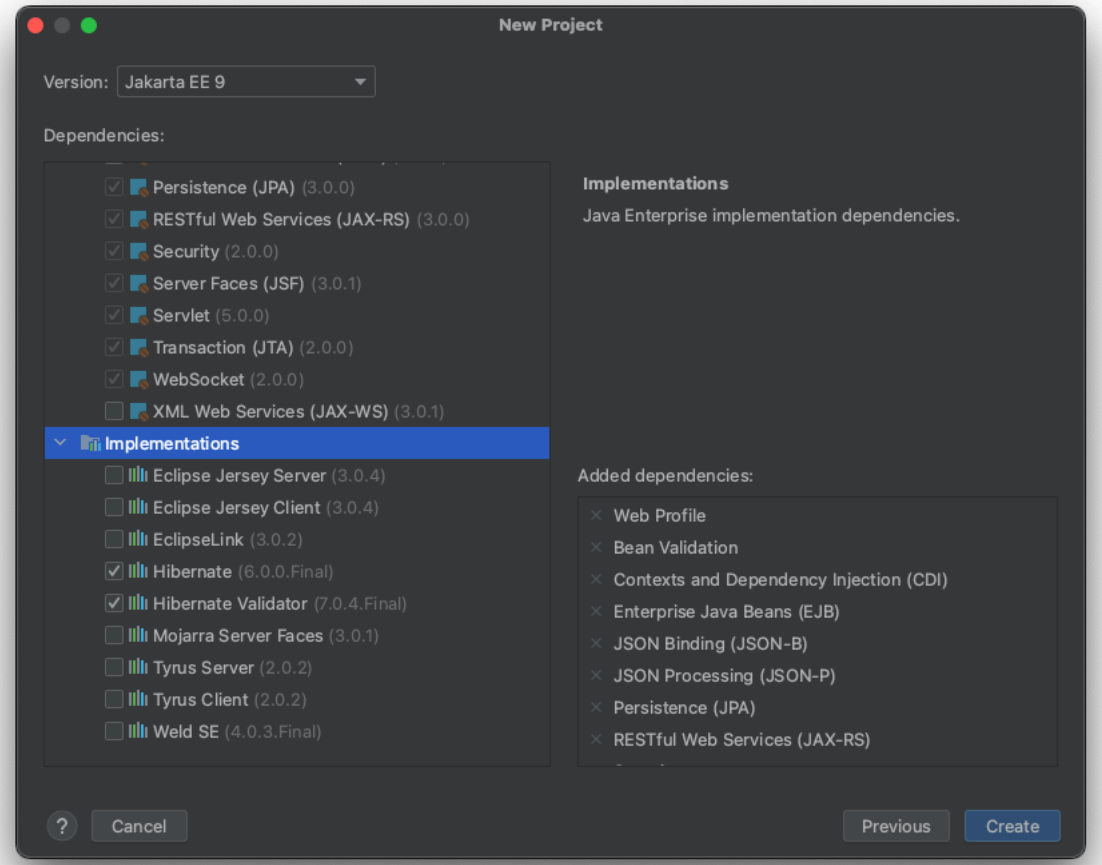
- Specifications
  - [ ] Full Platform (9.1.0)
  - [✓] Web Profile (9.1.0)
  - [ ] Batch (2.0.0)
  - [✓] Bean Validation (3.0.0)
  - [✓] Contexts and Dependency Injection (CDI) (3.0.0)
  - [ ] Concurrency Utils (2.0.0)
  - [ ] Connector Architecture (JCA) (2.0.0)
  - [✓] Enterprise Java Beans (EJB) (4.0.0)
  - [✓] JSON Binding (JSON-B) (2.0.0)
  - [✓] JSON Processing (JSON-P) (2.0.1)
  - [ ] Message Service (JMS) (3.0.0)
  - [ ] Model View Controller (MVC) (2.0.0)
  - [✓] Persistence (JPA) (3.0.0)
  - [✓] RESTful Web Services (JAX-RS) (3.0.0)
  - [✓] Security (2.0.0)
  - [✓] Server Faces (JSF) (3.0.1)
  - [✓] Servlet (5.0.0)
  - [✓] Transaction (JTA) (2.0.0)

Web Profile

Includes Servlet, CDI, JPA, Bean Validation, JSF, JAX-RS, WebSocket, JSON-P, JSON-B, Security, EJB and JTA specifications.

Added dependencies:

- Web Profile
- Bean Validation
- Contexts and Dependency Injection (CDI)
- Enterprise Java Beans (EJB)
- JSON Binding (JSON-B)
- JSON Processing (JSON-P)
- Persistence (JPA)
- RESTful Web Services (JAX-RS)



Version: Jakarta EE 9

Dependencies:

  - [✓] Persistence (JPA) (3.0.0)
  - [✓] RESTful Web Services (JAX-RS) (3.0.0)
  - [✓] Security (2.0.0)
  - [✓] Server Faces (JSF) (3.0.1)
  - [✓] Servlet (5.0.0)
  - [✓] Transaction (JTA) (2.0.0)
  - [✓] WebSocket (2.0.0)
  - [ ] XML Web Services (JAX-WS) (3.0.1)
- Implementations
  - [ ] Eclipse Jersey Server (3.0.4)
  - [ ] Eclipse Jersey Client (3.0.4)
  - [ ] EclipseLink (3.0.2)
  - [✓] Hibernate (6.0.0.Final)
  - [✓] Hibernate Validator (7.0.4.Final)
  - [ ] Mojarra Server Faces (3.0.1)
  - [ ] Tyrus Server (2.0.2)
  - [ ] Tyrus Client (2.0.2)
  - [ ] Weld SE (4.0.3.Final)

Implementations

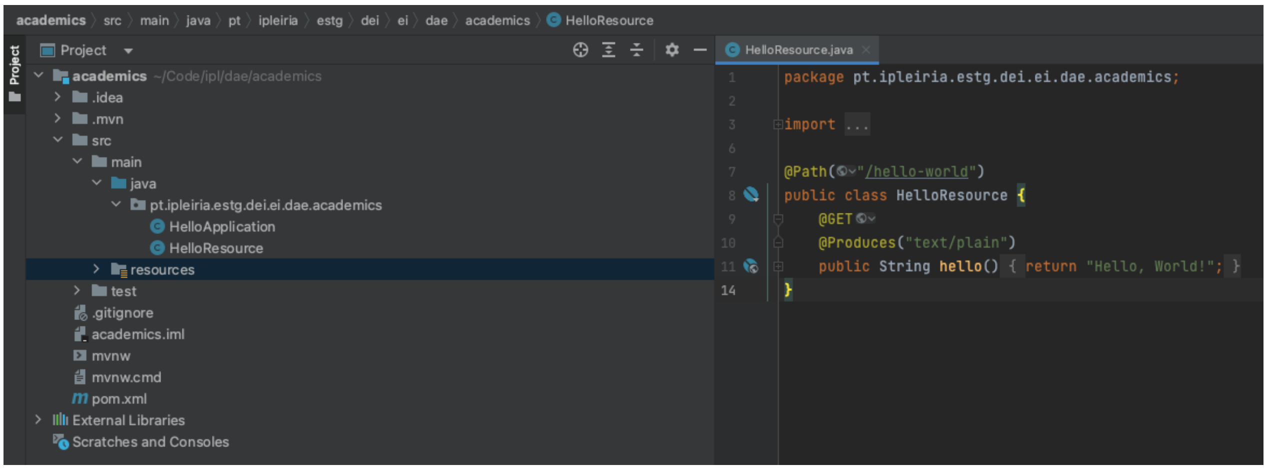
Java Enterprise implementation dependencies.

Added dependencies:

- Web Profile
- Bean Validation
- Contexts and Dependency Injection (CDI)
- Enterprise Java Beans (EJB)
- JSON Binding (JSON-B)
- JSON Processing (JSON-P)
- Persistence (JPA)
- RESTful Web Services (JAX-RS)

- Finally, click “Create”.

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



academics 〉 src 〉 main 〉 java 〉 pt 〉 ipleiria 〉 estg 〉 dei 〉 ei 〉 dae 〉 academics 〉 HelloResource

```
Project
academics ~/Code/ipl/dae/academics
  .idea
  .mvn
  src
    main
      java
        pt.ipleiria.estg.dei.ei.dae.academics
          HelloApplication
          HelloResource
      resources
    test
  .gitignore
  academics.iml
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
Scratches and Consoles
```

HelloResource.java

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

import ...

@Path("/hello-world")
public class HelloResource {
    @GET
    @Produces("text/plain")
    public String hello() { return "Hello, World!"; }
}
```

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

Open the pom.xml file and change this section:

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>3.3.2</version>
</plugin>
```

To include this (`in green`):

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>3.3.2</version>
    <configuration>
        <outputDirectory>target</outputDirectory>
        <warName>academics</warName>
    </configuration>
</plugin>
```

This will indicate that the war file (Web ARchive file we’ll be using to deploy our application) we want to build must be placed inside that directory, and that its name will be “academics.war”.

9. Open you 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

POSTGRES_DRIVER_VERSION=42.5.0

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 according to 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 Hibernate and PostgreSQL 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 only the tag <persistence-unit/> *(*not the entire content of persistence.xml)*:

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

And, where was the previous deleted tag, **replace** it with:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">
    <persistence-unit name="AcademicsPersistenceUnit" transaction-type="JTA">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</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>
</persistence>
```

This configures our project to use the PostgreSQL database, already configured in the Wildfly application server 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
```

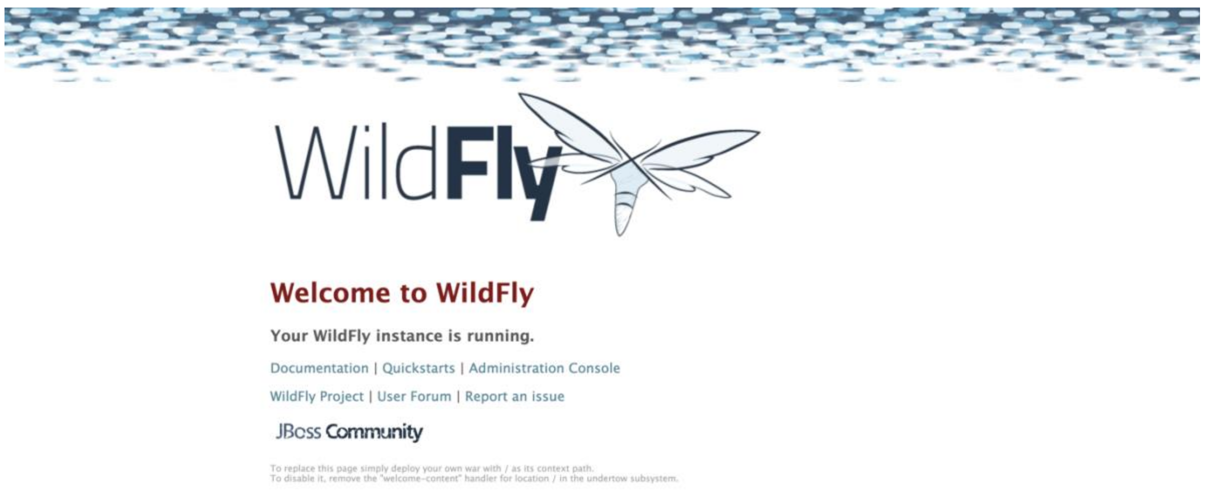
Containers should start running.

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:



Welcome to WildFly

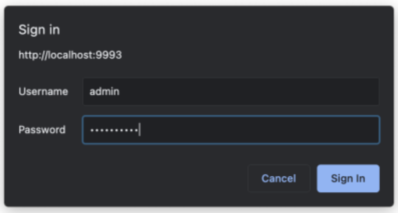
Your WildFly instance is running.

Documentation | Quickstarts | Administration Console
WildFly Project | User Forum | Report an issue

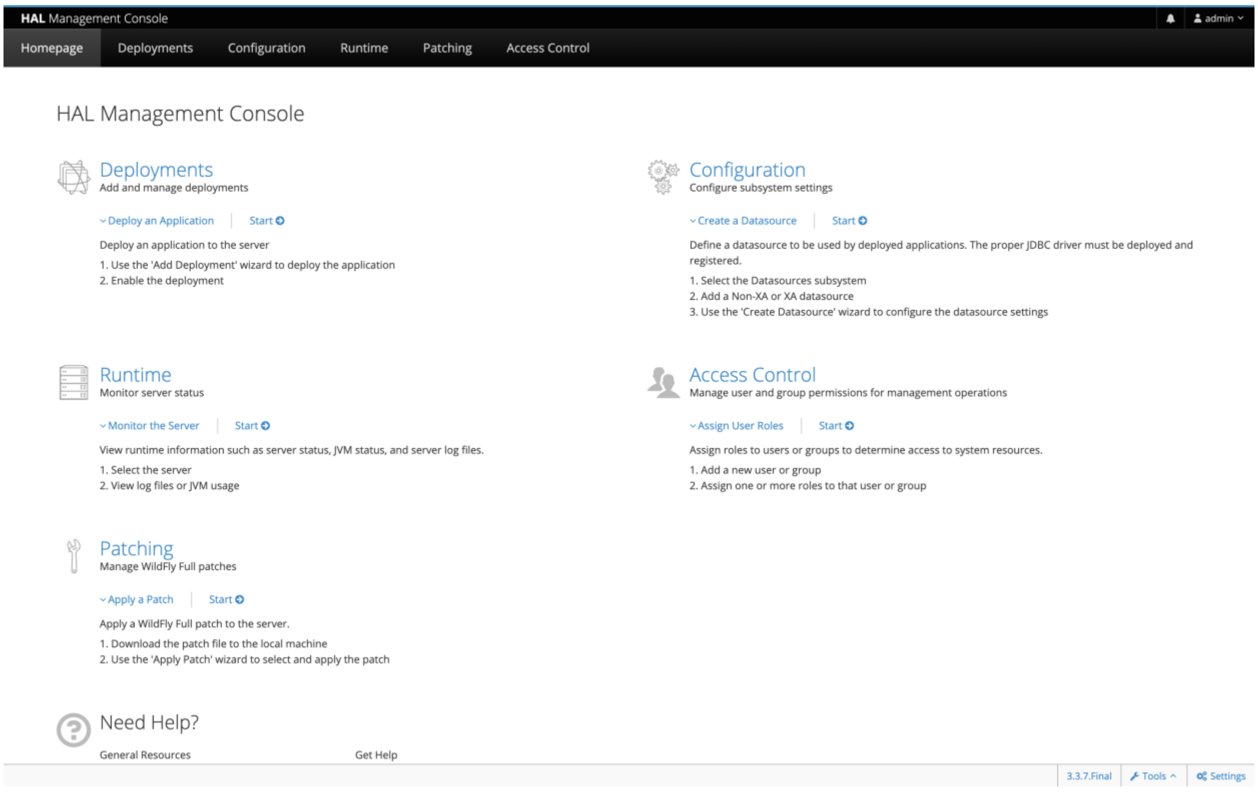
JBoss Community

To replace this page simply deploy your own war with / as its context path.
To disable it, remove the "welcome-content" handler for location / in the undertow subsystem.

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.



Sign in
http://localhost:9993
Username: admin
Password: ••••••••••



HAL Management Console

Deployments
Add and manage deployments
Deploy an Application | Start
Deploy an application to the server
1. Use the 'Add Deployment' wizard to deploy the application
2. Enable the deployment

Configuration
Configure subsystem settings
Create a Datasource | Start
Define a datasource to be used by deployed applications. The proper JDBC driver must be deployed and registered.
1. Select the Datasources subsystem
2. Add a Non-XA or XA datasource
3. Use the 'Create Datasource' wizard to configure the datasource settings

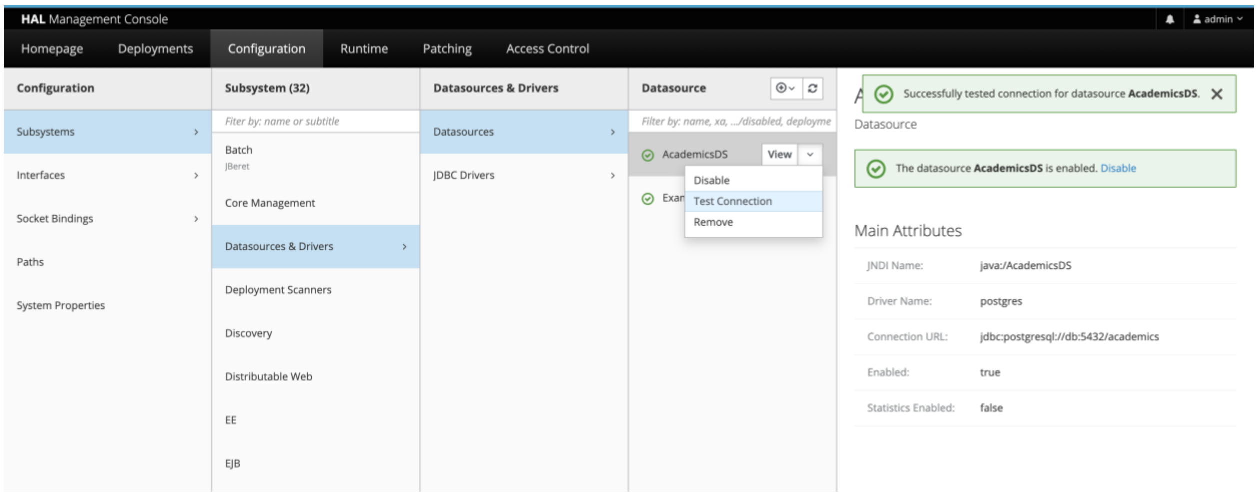
Runtime
Monitor server status
Monitor the Server | Start
View runtime information such as server status, JVM status, and server log files.
1. Select the server
2. View log files or JVM usage

Access Control
Manage user and group permissions for management operations
Assign User Roles | Start
Assign roles to users or groups to determine access to system resources.
1. Add a new user or group
2. Assign one or more roles to that user or group

Patching
Manage WildFly Full patches
Apply a Patch | Start
Apply a WildFly Full patch to the server.
1. Download the patch file to the local machine
2. Use the 'Apply Patch' wizard to select and apply the patch

Need Help?
General Resources Get Help

Finally, check if the datasource was created and it's working. Click on the "Configuration" link. Select "Subsystems → Datasources & Drivers → Datasources → AcademicsDS". Unroll the combo box next to "AcademicsDS" and click "Test connection", to check it's working.



Successfully tested connection for datasource **AcademicsDS**.

Datasource

The datasource **AcademicsDS** is enabled. Disable

Main Attributes

JNDI Name:	java:/AcademicsDS
Driver Name:	postgres
Connection URL:	jdbc:postgresql://db:5432/academics
Enabled:	true
Statistics Enabled:	false

15. Great! Now that you've started your Docker containers, and the configurations are OK, you can start developing your enterprise application. We will start by creating a Student and save it on the database, in order to test that our project can write into the database.

16. First, let's modify the Java EE version of our project, so that it can be compatible with our Wildfly application server. Head to the pom.xml file, and make sure that the Jakarta (Java EE) platform is set to version 8.0.0

```
<dependency>
    <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-web-api</artifactId>
    <version>8.0.0</version>
    <scope>provided</scope>
</dependency>
```

From here, all Java EE annotations should be supported by imports starting by `javax`…(and not by `jakarta`…)

17. On your IntelliJ project, create an `ejbs` package inside the `pt.ipleiria.estg.dei.ei.dae.academics` one.

18. Create a singleton Enterprise Java Bean (EJB - Java class) named `ConfigBean` in the `ejbs` package. This EJB should have the following Java EE annotations right above the class declaration of the EJB:

- `@Singleton` // this EJB will have only one instance in the application;
- `@Startup` //this EJB will be automatically instantiated once the application is deployed onto the Wildfly application server.

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

19. 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!");
    }
}
```

20. 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)

21. 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 `docker compose -f logs webserver` command or using our shortcut `make` command:

```
$ make logs
```

22. 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 implement the `Serializable` interface, and 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;

23. 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). Again, *all imports must come from "javax.*", not "jakarta.*"*

24. 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).

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

26. 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
  - Database: `academics`;
  - URL: `jdbc:postgresql://localhost:5432/academics`;
  - Leave the other fields as they are;
  - Click "Test Connection";
  - Click "Ok".
- Expand the `academics` 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.