PING — Subject

version #

Yet Another Kind of Assistants 2025 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.forge.epita.fr

Obligations

*Obligations are **fundamental** rules shared by all subjects. They are non-negotiable and to not apply them means to face sanctions. Therefore, do not hesitate to ask for explanations if you do not understand one of these rules.*

Obligation #0: **Cheating**, as well as sharing source code, tests, test tools or coding-style correction tools is **strictly forbidden** and penalized by not being graded, being flagged as a cheater and reported to the academic staff.

Obligation #1: Your submission repository must be **clean**. Except for special cases, which (if any) are **explicitely** mentioned in this document, an *unclean* repository may contain:

- binary files;[1]
- files with inappropriate privileges;
- forbidden files: `*~`, `*.swp`, `*.o`, `*.a`, `*.so`, `*.class`, `*.log`, `*.core`, etc.;
- a file tree that does not follow our specifications.

Obligation #2: If you do not submit your work before the deadline, it will not be graded.

Obligation #3: All your files must be encoded in ASCII or UTF-8 without BOM.

[1] If an executable file is required, please provide its sources **only**. We will compile it ourselves.

All files in the file tree are to submit. Be careful, you **must** have a root `ping/` directory.

1 Introduction

PING is a project designed to mobilize all the skills you have learned throughout this intense and beautiful year. This project aims to implement your own IDE according to your coach's expectations. It is composed of two major parts: the backend and the frontend. This subject presents only the mandatory features your backend must implement.

2 Setup

2.1 Git

2.1.1 Repository

As usual, you will be provided with an assistant repository. However, to work appropriately in a group, we recommend you use your own git repository. This will give you more flexibility and get you used to the **must-have** tools developers need.

We recommend using the Gitlab of epita.

Be careful!

Your repository must be private! A public repository will be considered cheating. Also, remember to push on your assistant repository to be graded.

2.1.2 Provided files

Some files are provided to you through the Intranet. The `@Given` annotation above Java classes, interfaces, and enums can be found. Thanks to this annotation, you can identify which files will be overwritten by the moulinette. Keep in mind that the `pom.xml` will also be overwritten!

You are now ready to start!

3 Goal

Your goal here is to implement the backend part of your IDE. The backend will handle the different actions your IDE can perform, like editing files, compiling a project, and so forth... You must implement all the given interfaces to be graded.

Tips

The given interfaces are well documented. Take your time reading them all and understanding what is expected from you. This document is only a summary of it.

Be careful!

Remember, you **must** follow the layered architecture for this project. You are free to implement your backend as you want as long as you respect those constraints.

3.1 Project Service

You **must** handle your projects through the `ProjectService` interface. This service will manipulate the entities that implement the `Project` interface.

3.1.1 Project

The `Project` entity is the logical representation of the projects your IDE works with. A project is at least represented by an architecture of files and by its `aspects` and `features`.

3.1.2 Aspect

The `Aspect` interface represents the different forms that a project can take. Your IDE must be able to handle Git and Maven projects. Thus, `Maven` and `Git` could be valid aspects. A project can have multiple aspects at the same time, and every project has `Any` aspect.

For example, a Maven project in a Git repository would have `Maven`, `Git`, and `Any` aspect at the same time.

You **must** at least implement the aspects defined in the `Mandatory.java` file.

3.1.3 Feature

Depending on its aspects, your project will have different `Features` available. For example, if the project is using Maven, you must be able to `Compile`, `Package` or `Execute` it.

You **must** at least implement the features defined in the `Mandatory.java` file.

Your features must be called this way:

```
project.getFeature(Mandatory.Features.Maven.COMPILE).get().execute(project);
```

Tips

To help you, some dependencies are included in the given `pom.xml`. You will need them to handle all features correctly.

Executing a feature return an `ExecutionReport` as described in the `Feature.java` file, this report only have a boolean representing whether or not the execution have been successful.

However, some features need extra informations in theirs reports, such as the given one `SearchFeatureReport` in the `SearchFeatureReport.java` file.

Tips

We encourage you modifying these given classes to fit your implementation. However, you should not change the given getters such as getResults.

3.2 Node Service

The node service is the interface used to manipulate the `nodes` in your project. A `node` can either be a file or a folder. You are free to add any other kind of `node` if you need to.

3.3 MyIde

In order to grade your work, you **must** implement the `MyIde` class.

Be careful!

Read the javadoc **carefully** as it describes the expected behavior for all the given interfaces.

3.4 MyIdeEndpoint

To make the link between your `React` frontend and your `Java` backend you must implement the `MyIdeEndpoint` class.

To do so, you will use the framework `Quarkus` that you already used during JWS. A `Hello World !` sample is given, we encourage you to take a look at the documentation to have a better understanding on what is happening. You should neither remove or change this sample endpoint.

You should at least implement these endpoints as post requests :

- /api/open/project
- /api/open/file
- /api/create/file
- /api/create/folder
- /api/delete/file
- /api/delete/folder

All these endpoints take a single string parameter `path` in the body of the request. You also need to implement these ones with different parameters :

- **/api/execFeature :**
  - feature : string
  - params : list of string
  - project : string
- **/api/move :**
  - src : string

    - dst : string
- **/api/update :**
    - path : string
    - from : int
    - to : int
    - content : string

As in JWS, you'll need to be using JAVA 17 in order to run properly quarkus. You can use these commands (on the PIE and at home) to achieve this :

```
42sh$ nix profile install nixpkgs#openjdk17
42sh$ export JAVA_HOME="$HOME/.nix-profile/lib/openjdk"
```

You can test your program by running your server with :

```
42sh$ mvn quarkus:dev
```

And doing a request with :

```
42sh$ curl http://localhost:8080/api/hello
```

Be careful!

Speaking of Quarkus, what you've learned during JWS might be useful for this project.

3.4.1 Logger

A logger is provided in the `Logger.java` file with minimal logger function.

This class is here so that you can easily log what is happening in the different endpoints. That way it will be easy for you to prove that the backend is connected to the frontend during your defense for the frontend.

We encourage you to add more functions in this class to fit your needs.

You will need to log whenever something happen in each endpoint.

You also need to respect the format `<optional_color> [HH:mm:ss] <log_message> <optional_color_reset>`. Your logs have to be clean and understandable in order to pass the tests and help you debug. Therefore you will at least need to have all parameters of the endpoint where a log happen inside the log message.

4 Conclusion

4.1 Final Words

You may be questioning yourself about the utilities of this backend part in JAVA, as your frontend will be in React and support more features but keep in mind that you will also be graded later on the backend of your final project and that its presence is mandatory throughout the entire ping project.

As you could encounter some difficulties starting the project or implementing some features, we provided you with useful links to documentation.

4.2 Helpful Links

4.2.1 Documentation

- Apache Lucene
- Maven lifecyles
- Zip files
- RandomAccessFile
- JGit
- Quarkus - JSON REST Services

Success is granted to those who seek the light and fight the shadows.