POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Online services for continuous evaluation - Kafka

Networked Software for Distributed Systems

Matteo Mormile, 10666565

---

1. Introduction

The project is based on the implementation of an online application that support university courses adopting continuous evaluation. The programme is made up of a frontend that receives user requests and a backend that handles them. Three categories of users interact with the service: professors, admin and students. The four services that compose the backend communicate with each other via Kafka and are:

- the user service which is responsible for managing registered users' personal information;
- the courses service manages courses, which are made up of multiple projects;
- the projects service handles the submission and grading of individual projects;
- the registration service registers final grades for courses that have been completed (a course is considered completed for a student if they have completed all of the projects).

Each user, depending on their role, will access a different dashboard that will allow them to perform different functions such as register for courses, submit solutions for projects, create or remove courses and others.

2. Architecture

The project, which is written in Java and includes support for the Kafka libraries, is divided into two major components: front-end and back-end. The front-end component handles client interaction, specifically data visualisation and user interaction. The back-end part, on the other hand, is in charge of ensuring that the four different services mentioned in the introduction communicate each other through Kafka. Remote Method Invocation (RMI) technology will be used for communication between the front-end, specifically the HTTP server that will handle client requests, and the back-end, which is composed by the individual services. In fact, each service will also expose a set of functions that the server can call as needed.

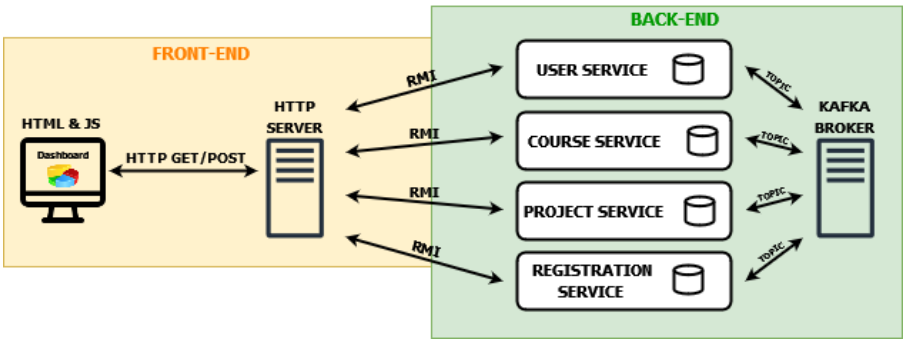

Figure 1: Font-end and Back-end architecture of the project.

2.1. Frontend - Javascript and Java HTTP Server

As previously stated, the front-end, which is responsible for user interaction, is made up of html pages with css styles that communicate with a server via ajax requests implemented in javascript. The http server, which is written in Java and listens on port 8600, sends requests to the appropriate http handlers based on the requested path. Once processed and sent, the response is parsed by javascript and displayed appropriately.

Listing 1: Create request and analize response for showing submission details

```
function showSubmissionDetails(project_name) {
  let param = /* create string of parameters */
  makeCall("GET",'http://'+http_server_address+':'+http_port+'/'+http_project_path,
    param, function (req) {
      if (req.readyState === XMLHttpRequest.DONE) {
        switch (req.status) {
          case 200:
            document.getElementById("assignment_details").innerHTML = req.responseText;
          /* handle other status */
        }
      }
    });
}
```

2.2. Frontend and Backend communication - RMI

The communication between the handler classes of the http server and the services that make up the backend is via RMI. In fact, each service exposes several remote functions and we find them indicated in the `Remote` interfaces.

Listing 2: Remote functions exposed by Course service

```
public interface CourseRemote extends Remote {
    boolean addCourse(String course_name) throws RemoteException;
    boolean enrollStudent(String username,String course_name) throws RemoteException;
    boolean addProject(String course_name,String project_name) throws RemoteException;
    List<String> getProjectsName(String course_name) throws RemoteException;
    /* ... */
}
```

Each of these functions, implemented by the services, will be responsible for retrieving the information saved in their respective databases. In fact, each service will have its own database with the structure that is most appropriate for storing the data it handles.

2.3. Backend - Kafka

When the information saved in the databases changes, such as when a course is created, the service informs the other services by publishing a message on a specific topic. At the same time, it will publish the new value to its own private topic so that it can be retrieved in the event of a service crash. The two messages are published through a transaction to ensure their presence on the various topics.

Listing 3: Publish message on update and recovery topics

```
@Override
public boolean addCourse(String course_name) throws RemoteException {
  try {
    [...] /* course added to the database */
    course_producer.beginTransaction();
    // publish course for recovery
    ProducerRecord<String, String> record_course =
            new ProducerRecord<>(courseTopic, course_name, gson.toJson(course));
    course_producer.send(record_course);
    // publish course for update other services
    record_course = new ProducerRecord<>(courseUpdatesTopic, "new_course", course_name);
    course_producer.send(record_course);
    course_producer.commitTransaction();
  } catch (IllegalArgumentException e) {
    // abort transaction
    course_producer.flush();
    course_producer.abortTransaction();
    return false;
  }
  return true;
}
```

2.3.1 User Service

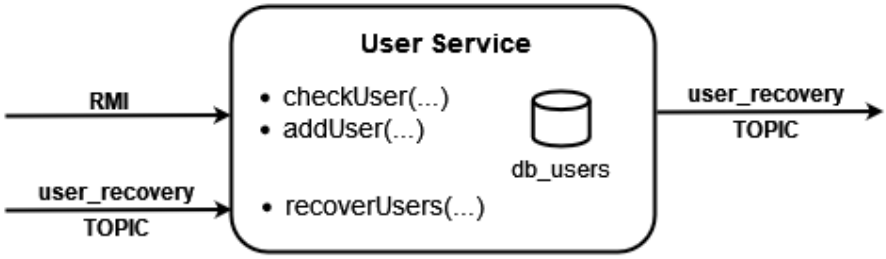

Figure 2: Detailed view of user service

The User microservice manages users and their roles. It includes functions for creating new users and verifying their presence. Because it does not need to communicate with other services, it will only read and write the recovery topic `user_recovery`. Sending a message on that topic every time the database is modified will allow it to rebuild the data in the event of a failure. The database is a hashmap with the user's username as the key and the User object as the value, which contains the required information like name, role, and password.

2.3.2 Course Service

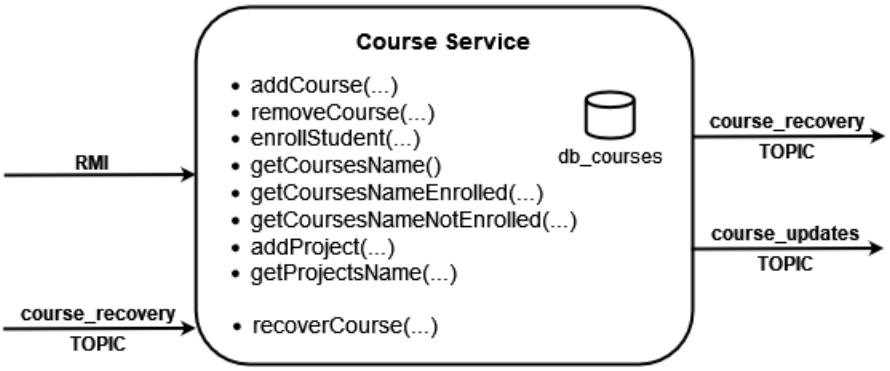

Figure 3: Detailed view of course service

The course service will be in charge of keeping track of the courses created by the admins and the projects available for each course managed by the professors. The service will also keep track of the students enrolled in each course, while the project service will handle delivery and evaluation. Because the project service will need to know what projects are present in the various courses, the course service will publish all changes on a special topic called `course_updates`. As previously stated, there will be a topic that will only use the course service to retrieve data in the event of a failure. The course data is again saved within a map that has the course name as key and the course object as value. This will contain lists on available projects and enrolled students.

2.3.3 Project Service

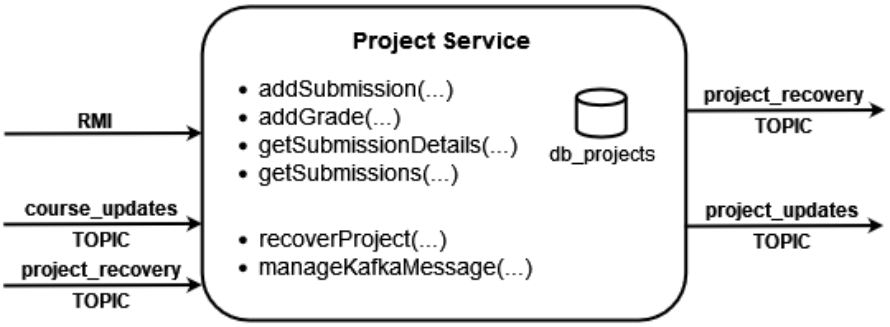

Figure 4: Detailed view of project service

This service will manage project delivery and grading. To keep its database up to date with the information stored by the course service, the project service will always listen for new messages on the `course_updates` topic. Because the registration service requires data on submissions, particularly grades assigned, in order to assign final grades, the project service will post both on the recovery topic `project_recovery` and on the project updates one `project_updates`. The service database will be a map with as key the name of the course and as value the list of projects in that course. Each project will contain the submissions. These will be saved as a map with as key the name of the student and as value the delivery data as file and grade assigned.

2.3.4 Registration Service

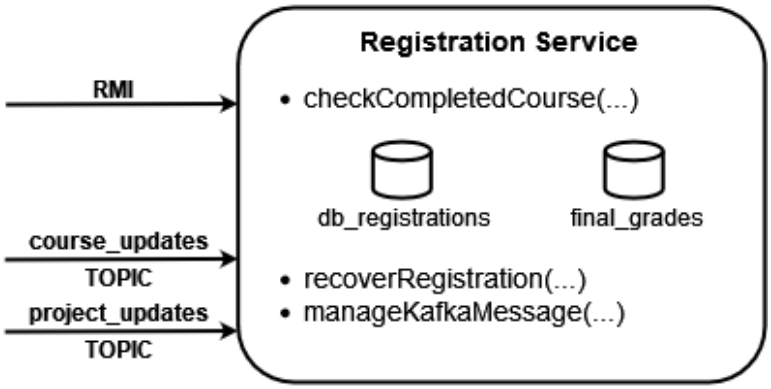

Figure 5: Detailed view of registration service

The final service will only record the final grade for a course if the student has completed all projects and the sum of the delivery grades is positive. Because the only function of the service that communicates with the http server is to check if the student has completed a course, the stored data is determined only by the messages received from the course and project services. In this way, there is no need to create a private topic of the registration service, but the status of the databases can be reconstructed directly from the messages of the two previous topics. The databases of the registration service are two: the first `db_registration` will contain the grades assigned to the various projects following a structure similar to `db_projects`; the second one instead `final_grades` will be a list containing the tuples `<course,username,grade>`.

3. Testing and Conclusions

In order to test the application, there are some parameters that need to be checked. In particular, within the `HTTPServer` class we find the declaration of the addresses of the individual services to which we will connect via RMI. In case these do not reside on the same machine it is necessary to update them.

Listing 4: Example having two services on the 192.168.1.33 machine

```
public class ServerHTTP {
    [...]
    // ip addresses rmi services
    private static final String userServiceAddress = "localhost";
    private static final String courseServiceAddress = "localhost";
    private static final String projectServiceAddress = "192.168.1.33";
    private static final String registrationServiceAddress = "192.168.1.33";
}
```

Another parameter that needs to be inserted is the address of the Kafka broker (it can be local or on another machine), which is specified as parameter when the services are run. If no parameter is specified, the broker will be searched locally.

```
java -jar Course_service.jar <KAFKA_BROKER_IP:PORT>

java -jar Course_service.jar
java -jar Course_service.jar 192.168.1.20:9092
```

Listing 5: Configuration with local and remote Kafka Broker

Also the HTTP server address to which our HTML pages will send requests, indicated in the `utils.js` file, must be equal to the one indicated in the `ServerHTTP.java` file.

Listing 6: Configuration with remote Http Server

```
const http_server_address = "192.168.1.20";
const http_port = 8600;
```

The last thing to check is the Kafka broker configuration. In particular I chose to use a `compose.yaml` file to make it easier to deploy the server. In the compose file we find indicated that the Kafka broker will respond to local requests on port 9091, while it will be available for external requests to its IP address via port 9092. It is necessary, if the compose file is used, to verify and change the IP address used.

To test if everything was working correctly, we tested the application first locally, using all services on the same machine (in this case all addresses will be 127.0.0.1), and then in a distributed environment connecting multiple machines in the same network. In both scenarios everything worked as expected and here there are some pictures of a sample test:

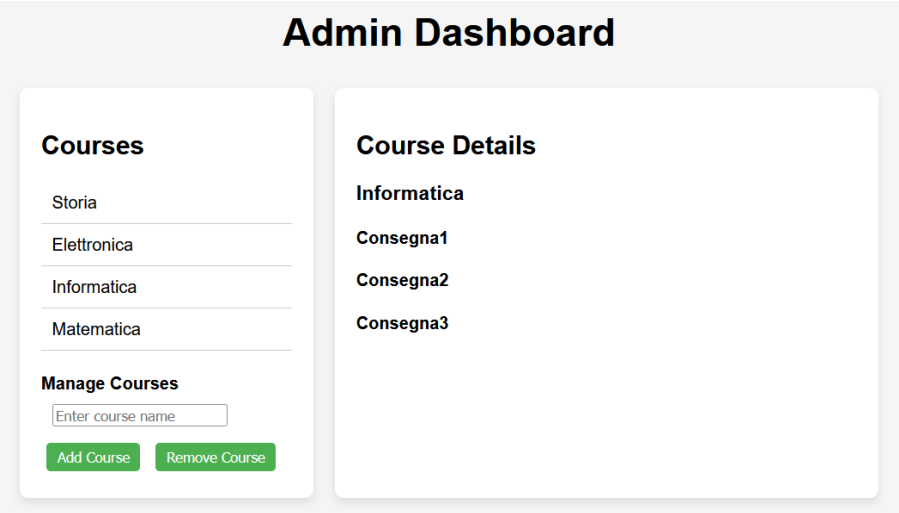

Figure 6: Admin Dashboard to add and remove courses

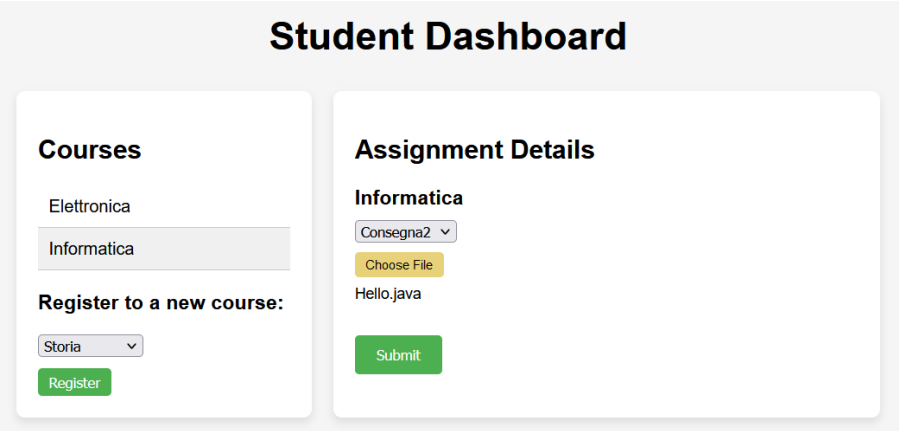

Figure 7: Student Dashboard for new course registration and project delivery

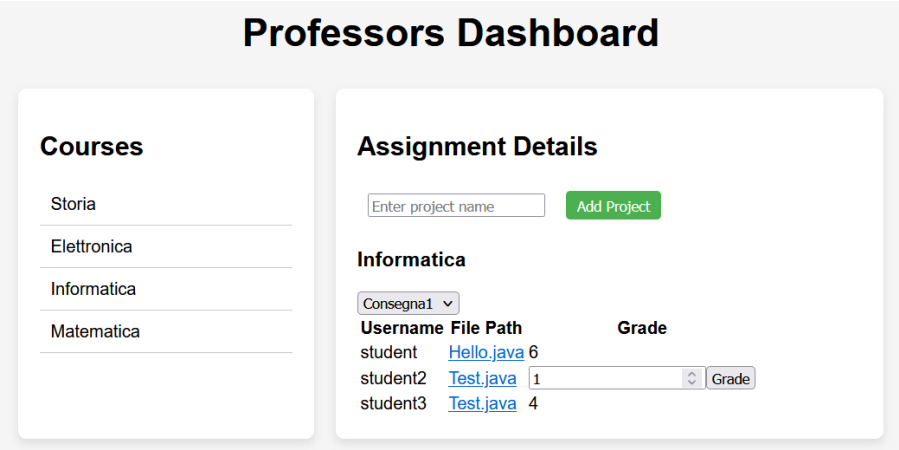

Figure 8: Professor Dashboard for adding new projects and evaluating submissions

```
CURRENT ADDRESS 192.168.56.1:2002
>>>> START RECOVERY PHASE
KAFKA MESSAGE: Created course <Informatica>
KAFKA MESSAGE: Created course <Matematica>
KAFKA MESSAGE: Created course <Storia>
KAFKA MESSAGE: Project <Consegna1> added to course <Informatica>
KAFKA MESSAGE: Project <Consegna2> added to course <Informatica>
KAFKA MESSAGE: Project <Consegna3> added to course <Matematica>
END RECOVERY PHASE<<<
Project RMI Server online
KAFKA MESSAGE: Created course <Geografia>
KAFKA MESSAGE: Created course <Elettronica>
HTTP MESSAGE: Submission added for student <student> to course <Informatica> for project <Consegna1>
HTTP MESSAGE: Grade <6> added for student <student> to course <Informatica> for project <Consegna1>
KAFKA MESSAGE: Project <Consegna3> added to course <Informatica>
HTTP MESSAGE: Submission added for student <student3> to course <Informatica> for project <Consegna1>
HTTP MESSAGE: Submission added for student <student2> to course <Informatica> for project <Consegna1>
HTTP MESSAGE: Grade <4> added for student <student3> to course <Informatica> for project <Consegna1>
```

Figure 9: Project service logs with recovery phase and message receptions via Kafka topics