Distributed Systems

(COMP41720)

Assignment 2: Group Project

Instructor: Professor Rem Collier

Project ID: 22203803

Group Name: IT_Elites_Distributed_System

UCD School of Computer Science

University College Dublin

IT_Elites_Distributed_System

Pranit Rale	Kaustubh Aphale	Shailab Chapagain
22200857	22200802	22203803

Synopsis:

The system presents a solution to the problem of remote file execution. The application is designed to allow clients to run .java (Java) and .py (Python) files remotely on a server and to receive their outputs. To do this, the client is connected to one of the chosen servers of a specific file format. i.e., python service or java service. Once the file is uploaded and submitted, the respective service which was selected by the client, will execute the file and return the output of the file as a response i.e., a string. Thus the client would be able to see the output in the window provided by the application. The process of execution is web-based, where one can interact with the immersive UI, upload files, and get the response.

The application domain for the remote execution file system is the tech industry, specifically for the industry where the client generates the files but does not have the resources or the privilege to run it on his own machine. Web-based compilers or coding assignments compilers are a fit implementation of this system.

Technology Stack

The main technologies used in our system are Spring Boot & REST for the backend and Angular 14 for the frontend.

- Spring Boot: We selected Spring Boot for the backend because JVM-based systems use it as a cutting-edge technology, highly adaptable, and trouble-free framework to create rapid, flexible, and mobile programs and systems. The features of the framework are how well it works with any Java application and how many extensions are offered to build the greatest online apps on the Java EE platform. Without Spring Boot, developers must manually deploy the application to a server after creating the project and make sure that all bean dependencies are appropriately mentioned in the context file (considered a time-consuming task by developers). Furthermore, as such coding tests are typically carried out in browsers when used with real-world apps, employing REST was a logical choice.

- Angular 14: Because it is the perfect tool for building high-quality software with less code, less time spent on debugging, and excellent scalability potential, we chose angular 14 for the front end. It is a leading framework for building online and mobile applications. Its USP is a one-page application, which makes the development and use of the web app quick by enabling dynamic URL changes without the burden of accessing the server for new pages.

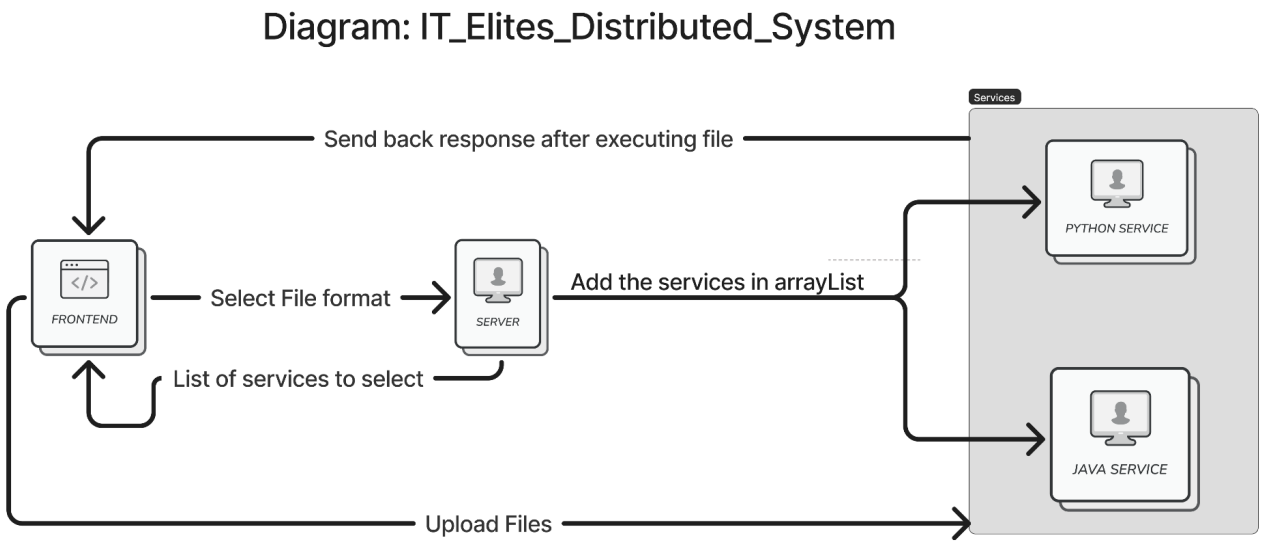
We decided to select the above set of technologies because they form a powerful tandem that works great for developing web applications. They provide a wide range of features and functionality that make it easier to build and maintain the application. Also, since they are popular and have been explored by a large group of developers it would be easier for us to explore, learn, and get support if required in any phase of development. An alternate option to choose would have been to use JMS but in that case, we would need to put the entire file in the queue resulting in making it synchronous. This would have a huge impact on performance, where REST leads due to its asynchronous nature. Moreover, the entire pipeline would fail if one of the components was to go down. After analyzing all the pros and cons, we decide to use the said technologies for the project.

System Overview

The main components of our system are as follows:

- **Server**: The server basically helps in establishing the connection between the client and the service. It is the one that registers all the existing services and maintains an array of services ready to execute files for different formats. It provides a list of selected services to the client after a client makes the selection of the format of the file via the frontend.
- **Python service**: Python service helps in executing the file which is in .py format. After the execution of the file, it sends the response back to the frontend from where the client made the request.
- **Java service**: Java service helps in executing the file which is in .java format. After the execution of the file, it sends the response back to the frontend from where the client made the request.
- **Fronted**: We have a single-page application where the client uploads the files by choosing the file format and specific service. A series of REST calls are made to get the servers and the output of the files submitted by the client.

Below is the system architecture diagram:



The system works as described below:

Initially, the client will interact with the front end where he can select the file format i.e. Java file or Python file. Depending on which type of file he chooses, the rest call will be triggered and he will be given a set of services from which he can choose one. Then the user is asked to upload the file and submit it for execution to the selected service. Basic validations are placed so that the request is not a void request.

The client on the selection of format sends a GET request to the server to get the list of selected services. The user then chooses which one he wants to connect to. It then greets the client with a file upload and output window page where the client uploads a file. On clicking the submit button, the file is sent to the service with POST request, which then processes the file and gives the output of the file as the response. The response of the typed string is then displayed in the client's output window.

Designed to support scalability and fault tolerance:

The system is designed to support scalability by allowing each new service to get registered with the server. The server is basically maintaining the list of services. Once a service is up, it automatically registers itself with the server through a REST call. That means more server uptime even when manipulating server contents. In order to support fault tolerance if one of the services goes down then a new service can be requested from the server upon refreshing the page. If the server goes down then the incoming request will be affected but the existing connection between the client and one of the services will still work because their connections are independent of the server. Once the client request is handled by the service, the failure of the server will not affect the processing of that request.

Contributions

Pranit:

- Implemented both the services i.e, Python service and java service which is registered with the server, and return a response as a list of strings to the frontend after executing the respective file. Used technologies are Spring Boot and REST.
- Performed testing and debugging of the system from time to time to ensure its performance and contributed to maintaining the system on Git.
- Contributed to making the video showcasing the system.

Kaustubh:

- Implemented client and frontend which is the starting phase in the process of file execution and sends a GET and POST request to the server and related services. Used technology like Angular 14, Spring Boot, and REST.
- Performed testing and debugging of the system from time to time to ensure its performance and contributed to maintaining the system.
- contributed to policing the report and guidance for writing better.

Shailab:

- Implemented server which maintains an array of servers ready to execute files for different formats and provides a list of selected services to the client after the client makes a selection of format of the file via frontend. Used technology like Spring Boot and REST.
- Performed testing and debugging of the system from time to time to ensure its performance and contributed to maintaining the system.
- contributed to writing the report.

Reflections

- Initially, our plan was to use JMS for fetching the list of servers but we could not resolve the proper method to get it working. Hence, we had to move to communication through REST. While developing, we had to learn and spend time on GIT to understand how to solve conflicts and how to manage the versions efficiently. Once a prototype was developed for frontend and backend we faced a strange situation while integrating where both entities would work individually, but did not work when integrated. Turns out we were facing a cross-origin resource sharing error, as it did not allow communicating to different ports while still using localhost. After researching for solutions, we found a chrome extension that resolved the issue for us.

- If we were to start again, we would have made a more detailed design to develop the system to provide more services. That being said, in the current version, there is no scope to unregister a

service if it goes out of service. Also, there is no mechanism for servers to know the health of the service. In the future, we could implement a heartbeat scenario so that the server can unregister a service if it fails a heartbeat more than 3 times. This would prevent the server from giving stale services to the client. The program is fit to execute a file which is a one-time thing but does not support a file that listens to events like creating a server. So maybe live streaming the events of the output would make sense.

- We learned new technology to develop the frontend using Angular and how to use SpringBoot to build REST API calls. What does a single page application mean and how to make components, service, and HTTP calls from the frontend, Bootstrap were the areas studied and implemented. For the backend, we applied the knowledge of REST calls and found ways by which we can make Java execute files on a machine. Both technologies provided a rapid development platform to quickly implement the functionality and set things in motion. However, this is only when one is able to adapt to a steep learning curve for both the technologies of the system. Generally, this is the choice of technologies preferred in the industry as they complete each other and spare the developers from complex integration problems. Both technologies solve the problem of hosting and handling server management problems, as the framework itself takes care of it. This reduces the workload and helps developers concentrate on creating the system rather than spending time wiring it. Since the technologies have more than the required stuff bundled, it causes an unnecessary increase in the size of the project, which unfortunately cannot be taken out.