{EPITECH}

ACTION-REACTION

CREATE AN AUTOMATION PLATFORM

ACTION-REACTION

General considerations

As part of this project, you will assume the role of a Software Architect team.

Your main goal is neither reinventing the wheel nor writing numerous lines of code. On the contrary, your main goal is to understand, select and integrate a wide range of existing libraries.

The code you write will only implement the so-called *business* logic. In other words, your main job will be to write *glue* between selected software components to complete the requested project.

Before embarking on the creation of such a project, we suggest you take the time to analyze and understand the operation of each software brick. In this theory, we will talk about **state of the art** and **POC**:

- ✓ **State of the art**: Study the different possible solutions and choose the right component based on the requirements.
- ✓ **POC** (*Proof Of Concept*): Make a quick demo program that proves the proper functioning of a component or algorithm.

This project is longer than your usual ones. It will be **essential** for the success of this project that you put in place real project management and not rush headlong.

The project

The goal of this project is to discover, as a whole, the software platform that you have chosen through the creation of a business application.

To do this, you must implement a software suite that functions similar to that of IFTTT and/or Zapier.

This software suite will be broken into three parts :

- ✓ An **application server** to implement all the features listed below (see *Features*)
- ✓ A **web client** to use the application from your browser by querying the **application server**
- ✓ A **mobile client** to use the application from your phone by querying the **application server**

No business process will be performed on the **web & mobile** client side, which only serves as a user interface and redirects requests from/to the **application server**.

While developing the **web client** and **mobile client**, consider accessibility guidelines and best practices. Make your application usable by as many people as possible, including those with disabilities.
Click here for more information

Functions

The application will offer the following funtionalities:

- ✓ The user registers on the application in order to obtain an account (cf User management)
- ✓ The registered user then confirms their enrollment on the application before being able to use it (see Authentication / Identification)
- ✓ The application then asks the authenticated user to subscribe to **Services** (cf Services)
- ✓ Each **service** offers the following components:
  - type **Action** (see *Action Components*)
  - type **REAction** (see *REAction Components*)
- ✓ The authenticated user composes **AREA** by interconnecting an **Action** to a **REAction** previously configured (see AREA)
- ✓ The application triggers **AREA** automatically thanks to **triggers** (cf. Trigger)

The **application server** is only exploited for **web and / or mobile clients**, which will have to expose all of its functionalities through a REST API.

Work group

The project is to be done in a group. Validation of the associated module will take into account not only the quality of the work performed but also the quantity of available features.

Here is the minimum expected configuration for a group of **X students**:

- ✓ Either **NBS** the number of **services** supported by the **application server** and available from **clients**
- ✓ Or **NBA** the total number of **Actions** supported by all **services** and **customers** available
- ✓ Or **NBR** the total number of **REActions** supported by all **services** and **customers** available

Here's whats expected of you:

- ✓ **NBS** >= 1 + **X**
- ✓ **NBA** + **NBR** >= 3 * **X**

In the case where one of the two **web or mobile clients** offers less functions than the other, this calculation will be based on the least successful client.

User Management

As the application is centered on the digital life of users, it must therefore offer a management of the latter.

To do this, you must create a user management module.

The **client** asks non-identified users to register via an online form. Take inspiration by what you already know about this step (eg fill in an email address, registration via a third party service like Google, Facebook, X, etc.).

When the form is submitted from the **client**, a request is made to the **application server** to validate this account creation step.

An administration section would be useful to manage site users.

Authentication / Identification

Using the application requires knowing the user in question.

To do this, it is necessary to implement the following options:

- ✓ A method of user authentication via a username / password. In this case, the **client** transmits the request to the **application server** which will process it.
- ✓ A method of identifying users via OAuth2 (eg Google / X / Facebook / etc.). In this case, the **client** processes itself this identification and warns the **application server** if successful.

Regarding the method of identification, remember to connect the third party account to a system user.

Services

The purpose of the application being to interconnect **services** between them (Outlook 365, Google, OneDrive, X, etc.), it is first necessary to propose to the authenticated user to select the **services** for which he has an account.

In this part, it will therefore be necessary to ask users to subscribe to these **services** (eg from their profile page the user links their X / Google account / etc via an authentication OAuth2).

When it is necessary, the **client** will have to manage the identification before transmitting to the **application server** the subscription to this **service** (eg the user subscribes to the Facebook service which requires them to log in via OAuth2).

The available **services** offered to the user from their **client** will be retrieved from the **services** list implemented on the **application server** side.

Composants Action

Each **service** may offer **Action** components. These components enable the activation of a **trigger** (see *Trigger*) when the condition is detected.

Some examples :

- ✓ Google **service** / Facebook / X / Instagram / etc.
  - A new message is posted in group G
  - A new message containing a #hashtag is posted
  - A new private message is received by the user
  - One of the user's messages gets a like
  - The user gains a Follower
- ✓ **service** RSS / Feedly / etc.
  - A new article is available
  - An article is added to their favorites by the user
- ✓ **service** OneDrive / Dropbox
  - A new file is present in the X directory
  - A user shares a file
- ✓ **service** Outlook 365 / Mail / Gmail / etc.
  - Receipt of a message from a user X
  - Receipt of a message whose title contains the word X
- ✓ **service** Timer
  - The current date is of the type DD / MM
  - The current time is of the type HH: MM
  - In X days it will be Y (ex In 3 days, it will be Friday).

Composants REAction

Each **service** may offer **REAction** components. These components perform a specific task by activating a **trigger** (see *Trigger*).

Some examples :

- ✓ **service** Google / Facebook / X / Instagram / etc.
- ✓ The user posts a message in group G
- ✓ The user is a new person P
- ✓ **service** OneDrive / Dropbox
- ✓ The user adds the file F in the directory R
- ✓ The user shares the F file with another U user
- ✓ **service** Outlook 365 / Mail / Gmail / etc.
- ✓ The user sends a message M to a recipient D
- ✓ **service** Scripting
- ✓ The user checks his ‘pickup’ rights on the project P

AREA

After subscribing to different **services** (see *Units*), the authenticated user can create some **AREA** in order to execute a **REAction** when an **Action** is found.

Some examples :

- ✓ Gmail / OneDrive
- ✓ **Action**: A received email containing an attachment
- ✓ **REAction**: The attachement is stored in a directory in OneDrive
- ✓ GitHub / Teams
- ✓ **Action**: An issue is created on a repository
- ✓ **REAction**: A message is sent on teams

Trigger

This essential element of the application aims to trigger **REActions** when the effects of **Actions** linked via an **AREA** are recorded.

To do this, for each **Action** used in the system, it will retrieve the necessary elements to determine whether or not this **Action** occurred.

For example, for the **Action** "*A project P is to be completed in less than H hours*":

- ✓ The **trigger** will list the user projects
- ✓ The **trigger** will find that the project P ends in less than H hours
- ✓ The **trigger** checks that this is the first time they see it for this project P
- ✓ The **trigger** will activate the **REAction** components present in the **AREA** of the user that are linked to this **Action** component

Refer to the IFTTT or Zapier website for a better understanding of this feature if necessary.

Architecture

Mobile Client

The **mobile client** should be available on Android or Windows Mobile. It will only be responsible for displaying screens and forwarding requests from the user to the **application server**.

The **mobile client** must provide a way to configure the network location of the **application server**.

Web Client

The **web client** will only be responsible for displaying screens and forwarding requests from the user to the **application server**.

Application Server

The **application server** is the only one to embed the *business* logic of the project. It will offer its services to **web & module clients** through a REST API.

No business model process will be carried out on the **web & mobile** client side. This only serves as a user interface and redirects requests from / to the **application server**.

Project Construction

As part of this project, you will have to develop several more or less independent parts. Each of these parts will be free to use the technologies of your choice.

In order to homogenize the construction of such a project, it will be based on the use of Docker Compose

Please read the following points carefully

docker-compose build

You will have to make a docker-compose.yml file at the root of your project, which will describe the different docker services used.

This file must include at least the following three docker services:

- ✓ A **server** service to launch the **application server** on port 8080
- ✓ A **client_mobile** service to build the **mobile client**
- ✓ A **client_web** service to launch the **web client** on port 8081

The **client_web** depends on **client_mobile** AND **server** .For more information, see the documentation about depends_on

docker-compose up

Validation of the integrity of your images will be done when launching the *docker-compose up* command.

The following points should be respected :

- ✓ The services **client_mobile** and **client_web** will share a common volume
- ✓ The **client_mobile** service will edit the associated binary and put it on the common volume with the **client_web**
- ✓ The **server** service will run by exposing the port 8080
- ✓ The **server** service will respond to the request http://localhost:8080/about.json (see File about.json)
- ✓ The **client_web** service will run by exposing the port 8081
- ✓ The **client_web** service will respond to one of the following queries:
- ✓ http://localhost:8081/client.apk to provide the Android version of **mobile client**

File about.json

The **application server** should answer the call http://localhost:8080/about.json.

```
{
    "client": {
        "host": "10.101.53.35"
    },
    "server": {
        "current_time": 1531680780,
        "services": [{
            "name": "facebook",
            "actions": [{
                "name": "new_message_in_group",
                "description": "A new message is posted in the group"
            }, {
                "name": "new_message_inbox",
                "description": "A new private message is received by the user"
            }, {
                "name": "new_like",
                "description": "The user gains a like from one of their messages"
            }],
            "reactions": [{
                "name": "like_message",
                "description": "The user likes a message"
            }]
        }]
    }
}
```

The following properties are required:

- ✓ **client.host** indicates the IP address of the client performing the HTTP request
- ✓ **server.current_time** indicates the server time in the Epoch Unix Time Stamp format
- ✓ **server.services** indicates the list of **services** supported by the server
- ✓ **server.services[].name** indicates the name of the **service**
- ✓ **server.services[].actions** indicates the list of **Actions** supported by this **service**
- ✓ **server.services[].actions[].name** indicates the identifier of this **Action**
- ✓ **server.services[].actions[].description** indicates the description of this **Action**
- ✓ **server.services[].reactions** indicates the list of **REActions** supported by this **service**
- ✓ **server.services[].actions[].name** indicates the identifier of this **REAction**
- ✓ **server.services[].actions[].description** indicates the description of this **REAction**

Documentation

It is important to take the time to define a simple architecture, without code duplication and that is scalable.

You are asked to provide clear and simple documentation of your project. You can integrate some design schemes: class diagram, sequence diagram …

The goal is to have a document that serves as a working support to easily understand the project in order to facilitate communication in the work teams and facilitate the development of skills of new developers.

Thus, there is no need to make class or sequence diagrams of the entire project, but rather to choose the important parts to understand what needs to be documented.

You will need to provide a README.md file describing the API of your **application server**. This file must be written using the markdown format.

Timeline of the project

As explained during the kick-off, this project will be divided into 3 milestones.

- ✓ The first defense - **Planning**:
  The goal here is to have experimented with various technological stack and languages and choosen the best one to do the job for your team.

  For your benchmarks, we recommend separating the **back-end** and **front-end**. This will allow you to evaluate each part of your application independently and choose the best technologies for each. However, if you decide to use a **full-stack** solution, you will need to justify your choice.

  Furthermore it is expected that you separated the project into multiple tasks, that task are scheduled in time (in a manner of your choosing!) and that the work organization between the members of the group is defined.
  A PoC of the project in the selected language wil be appreciated.

After planning your project and choosing your tech stack, don't forget to consider **security** and **database design (BDD)**. Security is crucial in any application to protect data and maintain trust with your users. Similarly, a well-designed database can improve performance, accuracy, and speed of your application. Consider these aspects while moving forward with your project.

- ✓ The second defense - **Minimum Viable Product**:
  The goal here is to demonstrate that you were able to realise the core architecture of the project and implemented the base concepts (make different APIs interact with each other). Furthermore, your planning must be adjusted to the reality of where you're at in the project and an analysis the changes will be asked.
- ✓ Final defense - **Final Product**:
  The goal here is to have finished the project with the most features possible (different types of services, intersting interactions, good user interface, deployment using docker, ...).
  The presentation of your project will also be evaluated along with an analysis of what went right and wrong during the development of the project and what you learned along the way.

v6