ADInt – MEEC

2021/2022 - Project

Access control system – Intermediate project

In this project students should develop a system (IST-GATE) that will allow the deployment of access control gates (such as the one in SCDEEC), control the opening of the gates, authenticate and control users that what to enter the gates and provide access to usage statistics and lists.

The final system will have two types of human users: administrators and regular.

Regular users will access the system using s smartphone when entering a gate. In the smartphone, users will authenticate using the FENIX username and password. Before trying to access a gate.

Administrators will define the available gates and will see the listings of the gates usage.

In this work students should

- Define the architecture of the systems (services, pages, gates)
- Define the set of resources to be made available by the various components
- Define the relevant information (attributes) of such resources to be stored in a Database
- Define the interfaces (WEB and REST) to access such resources
- Implement simple prototypes (in python and Javascript) the replicate the behavior of a real gate
- Implement a simple web server for access and management of resources

This document will only describe the functionalities to implement in the intermediate version. All decisions here made can be reused in the final project or changed if issues are found

1 Users

The intermediate system will be operated by two classes of users:

- Regular users that use a mobile application (simple web page) to open the gates
- Administrator that will access the admin pages to configure gates and see listings and statistics

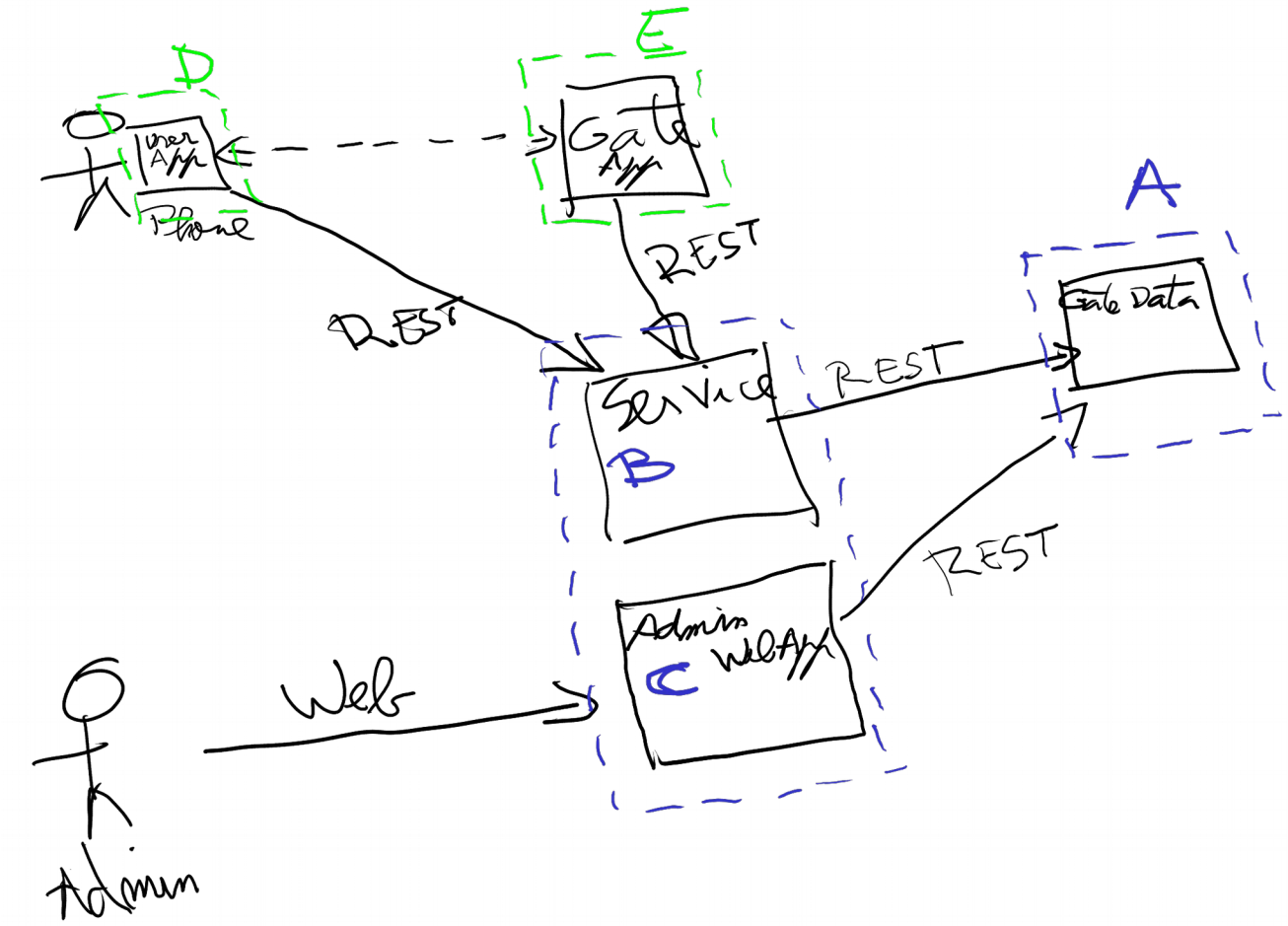
Gates are not regular users, but will have an internal representation in the system and will interact with it to read numeric codes/QRCodes and receive authorization to let users pass.

In the intermediary system there will only be one user, but the implemented functionalities will be extended to various concurrent users.

Student should take this into consideration when implementing the endpoints.

2 Architecture

The following picture shows the various components that make the **IST-GATE** system, along with the various types of interactions.



The components in green correspond to independent applications executed close to the user and to the gate. In the final version they will be coded in JavaScript, while in the intermediate version will be developed in python.

The components represented in blue will be coded in Python+Flask and will implement a set of REST APIs and administration web pages.

The administrator will interact with the system using a set of web pages, while all other interactions will be done using REST.

The **GateData** (**A**) will store the information pertaining all available gates (id, secret, location) it will allow the management of such information using a set of REST endpoints.

The **Service** (**B**) and **AdminWebApp** (**C**) components will be implemented in the same Flask application and will provide the web pages for the administration and a set of REST endpoints to be called from the **GateApp** (**E**) and **UserApp** (**D**).

2.1 UserApp

When a user whats to enter a gate, he needs to execute a simple application.

On the final version this application will run on the browser and after authentication will show a QRCode.

On the intermediate version this application will be developed in python and will only show a

simple alfanumeric code. This code is generated by the server and retrieved by the application using a REST API.

The application will work as follows:

> **python userapp.py**	> **python userapp.py** Contacting Server …	> **python userapp.py** Contacting Server … Code received >>> 56611X <<< Please type the code in the Gate >

Every time the application is executed a new code is generated and retrieved.

A code becomes invalid after 1 minute of creation or when a new code is generated.

2.2 GateApp

Every gate will have an application running continuously. The system should allow multiple gates simultaneously.

The **GateApp** in the final version will be developed in JavaScript and will take advantage of a camera to read a QRCode and send its content to the server for validation.

In the intermediate version the **GateApp** code will be developed in python and the QRCode reading will be replaced by the code keyboard input.

After reading the code, the **GateApp** will send it to the server that verifies if it is correct and valid. If the server validates the code, the gate will be opened for 6 seconds, after which it will be ready to read a new code.

When launching the application it is necessary to provide the **gate_id** and the **gate_secret**. as application arguments.

Wrong gate information:

> **python gateapp.py 1 2657**	> **python gateapp.py 1 2657** Contacting Server for …	> **python gateapp.py 1 2657** Contacting Server … The secret is not valid for this gate Exiting…. >

Regular usage:

> **python gateapp.py 1 2667**	> **python gateapp.py 1 2667** Contacting Server for …	> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code :
> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code : **56612**	> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code : **56612** Contacting Server …	> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code : **56612** Contacting Server … !!! Code Not valid !!! Type the user code :
> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code : **56612** Contacting Server … !!! Code Not valid !!! Type the user code : **56611X** Contacting Server …	> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code : **56612** Contacting Server … !!! Code Not valid !!! Type the user code : **56611X** Contacting Server … !!! Code valid !!! !!! The gate will close in 5 s !!!	> **python gateapp.py 1 2667** Contacting Server … The secret is valid for this gate Type the user code : **56612** Contacting Server … !!! Code Not valid !!! Type the user code : **56611X** Contacting Server … !!! Code valid !!! !!! The gate will close in 5 s !!! >

2.3 Service

The service is composed of a set of REST endpoints that allow both applications (**UserApp** and **GateApp**) to interact with the system.

Taking into consideration the functionalities of the applications students should define two

endpoints:

- one for the **UserApp** to retrieve a new user code
- one for the **GateApp** to verify is a code is valid

This the intermediary system version will only allow one user to operate the system, but students can assume an identity to such user in order to ease the future extension to multiple users.

2.4 AdminWebApp

This application will be composed of a set of pages that allow:

- the registration of a new gate
- the listing of registered gates

In order to register a new gate the administrator should supply the following information:

- **gateID** – unique identifier that allow the univocal access to a specific gate
- **gateLocation** – String that describes the gate

After successful registration of the gate on the system, the corresponding secret is shown in the browser.

The gates listing page will show a table with the following information related to each gate:

- id
- location
- secret
- number of activation (gate opens)

This web application will not store any information. All the data related to gates is stored in the **GateData** service and is access by a set of REST calls.

2.5 GateData Service

This service will store every gate information in a database and will provide a set of REST APIs to access, manipulate such data and to implement any additional operations.

3 Implementation

Students should follow the proposed steps in order to implement the intermediate project:

1. Define and implement the data-model of the database used by the **GateData** Service
2. Implement and test the REST API of the **GateData** so that new gates can be registers and listed.
3. Implement the **AdminWebApp** to allow a administrator user to easily register new gates.

4. Implement and test the REST API that allows a user to get a new **UserCode**
5. Implement and test the REST API that allows a gate to verify if a **UserCode** is valid
6. Implement the **userapp.py**
7. Implement the **gateapp.py**

Students should make decisions about some things not covered in this assignment.

4 Error validation

All endpoints should guarantee that incorrect call will deliver and error and not do incorrect behavior.

In case of invalid input the endpoints should return a suitable error code

5 Delivery

Students should deliver the code of all the components along with a simple **pdf** document that describes the data model and implemented REST API, and any decision made during the development.

6 Evaluation

The intermediate project will be evaluated taking into consideration the number of functionalities implemented, the correction of the API, error validations and the code structure.