INDIVIDUAL ASSIGNMENT PROJECT REPORT

Universidade de Aveiro - Software Engineering

To Do List app

LAFAGE Robin - 123811

Table of content

Table of acronyms

Acronym	Meaning
AWS	Amazon Web Services
US	User Story
VPC	Virtual Private Cloud
API	Application Programming Interface
UI	User Interface
IDP	Identity Provider
ALB	Application Load Balancer
ELB	Elastic Load Balancing

Acronym	Meaning
EC2	Elastic Compute Cloud
RDS	Relational Database Service
AZ	Availability Zone
ORM	Object-Relational Mapping
JWT	JSON Web Token

Table 1 : Table of acronyms

Table of figures

I - Introduction

The objective of this project is to develop a to-do list web application using microservices via Amazon AWS, while following an agile methodology to manage the project effectively.

In this report, we will first detail the use of the agile methodology, then examine the architecture of the implemented system, and then the implementation of the solution. After that, we will look at the difficulties encountered during the project, and what could have been done better, before concluding.

II - Agile methodology

1 - Agile definitions

Definition of ready	Definition of done
Explicit title (As a …, I want to …, in order to …)	Implemented

Definition of ready	Definition of done
Estimation of workload with Story Points	Tested
No blocking tasks to start working on it	Documented

Table 2 : Agile definitions

2 - Epics

In order to structure the project, it has been broken down into several main features, the following epics :

- **Launch of the project**: All tasks necessary for starting the project, including selecting and configuring the various tools and defining the overall architecture of the project.
- **Authentication**: User stories that allow users to log into the application and prevent access to other users' information.
- **Tasks management**: User stories related to task management, such as adding, editing, deleting, marking tasks as completed, and adding various details like deadlines or priorities.
- **Display management**: All aspects related to displaying tasks, including the ability to filter and sort them.
- **Deployment**: Tasks related to deploying the application on AWS.

3 - User stories

To successfully complete the project, user stories were used to break down the epics into specific actions to be performed, thus facilitating their planning, prioritization, and execution.

Each user story follows the format: *As a …, I want to …, in order to …*, to clearly define the added value from the user's perspective. To quantify this value, each US is assigned a story point value.

The following figure illustrates an example of a user story from the project:

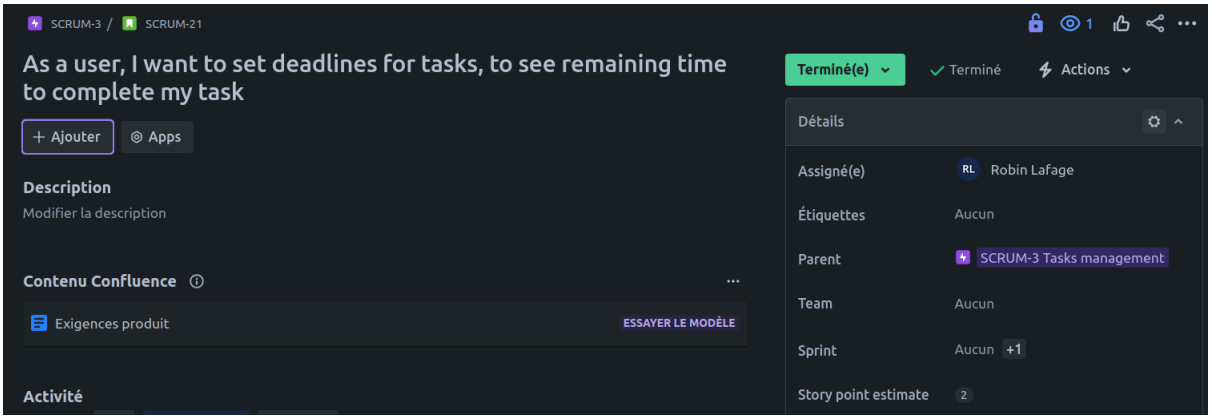

Figure 1 : Example of user story

4 - Sprints

The development of the project was divided into multiple sprints to achieve it iteratively. The progress of each sprint was tracked using Jira's Kanban board.

Sprint 1 : *Start the project* - 14/10/2024 - 21/10/2024

The purpose of this first sprint was to initiate the project before starting the development, ensuring a solid foundation and a clear direction for the development phase.

The work carried out during this sprint included choosing the tools to be used: programming languages, frameworks, the relational database management system, and AWS services for application deployment. The system architecture design was also within the scope of this sprint, covering both the overall architecture and the relational data model.

By the end of the sprint, almost all project variables were clearly defined, as will be detailed later in this report.

Sprint 2 : *Finish the project setup and start implementing endpoints and UI* - 21/10/2024 - 28/10/2024

The objective of this second sprint was first to finalize the project setup and then to implement the API endpoints along with their corresponding UI.

The first objective was achieved with the creation of the VPC architecture containing all AWS components. The scope of the second objective included the following features:

- Adding a task with a title and description
- Editing a task
- Deleting a task
- Marking a task as complete
- Adding a deadline to a task
- Adding a priority (high/medium/low) to a task
- Sorting tasks
- Filtering tasks

By the end of the sprint, all functionalities related to the to-do list itself were implemented locally, without authentication or AWS hosting.

Sprint 3 : *Set up authentication, deploy, and finish the project* - 28/10/2024 - 11/11/2024

The objective of this final sprint was to complete the project by adding authentication via an IDP and hosting the application on AWS to make it accessible from the internet.

Since this was my first real experience with AWS (beyond introductory modules), I struggled to estimate the time required for managing authentication and hosting. That's why I decided to define a two-week sprint instead of a one-week sprint, to avoid unfinished tickets at the end of the sprint while avoiding wasting time if a ticket took less time than expected.

III - Architecture

1 - Global architecture

I chose to design my system according to the following architecture:

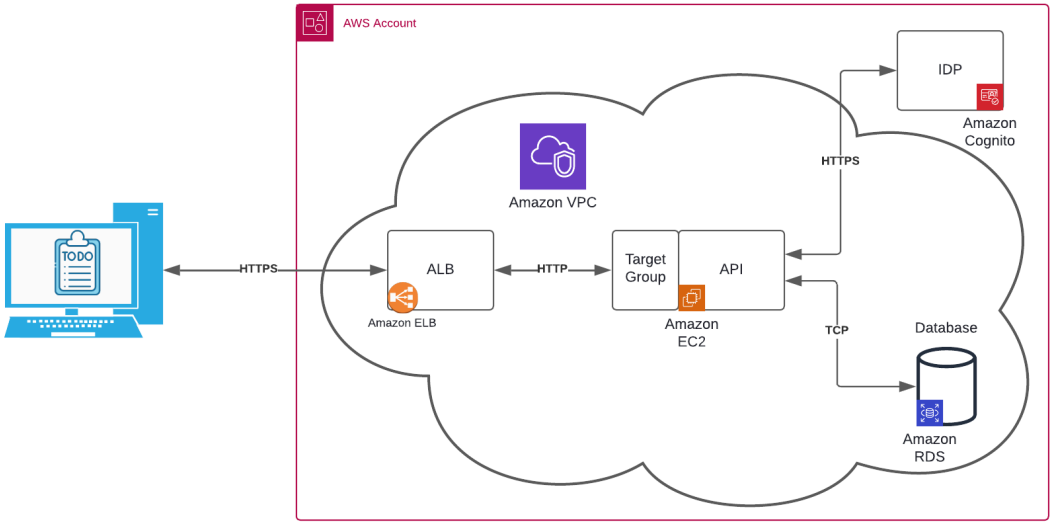

Figure 2 : Global architecture

Thus, the application (API and website) is accessible through an ALB (Amazon ELB) that distributes the load between two EC2 instances (via a target group) hosting the API and the website. Communication between the client and the ALB is secured using SSL/TLS, while HTTP is sufficient for communication between the ALB and the target group. I chose to use EC2 for its simplicity in deployment and easy integration with other services, such as the database.

For authentication, I use Cognito as the IDP. The API redirects users to the Cognito login page if they are not authenticated. Communication between the EC2 instance and Cognito occurs over HTTPS.

Finally, the database is hosted on RDS MySQL and is only accessible from the EC2 instances. The connection is established over TCP. The choice of an SQL solution, particularly MySQL, is justified by my prior experience with this tool compared to other database management systems or NoSQL solutions.

2 - Network architecture

The VPC is built as follows :

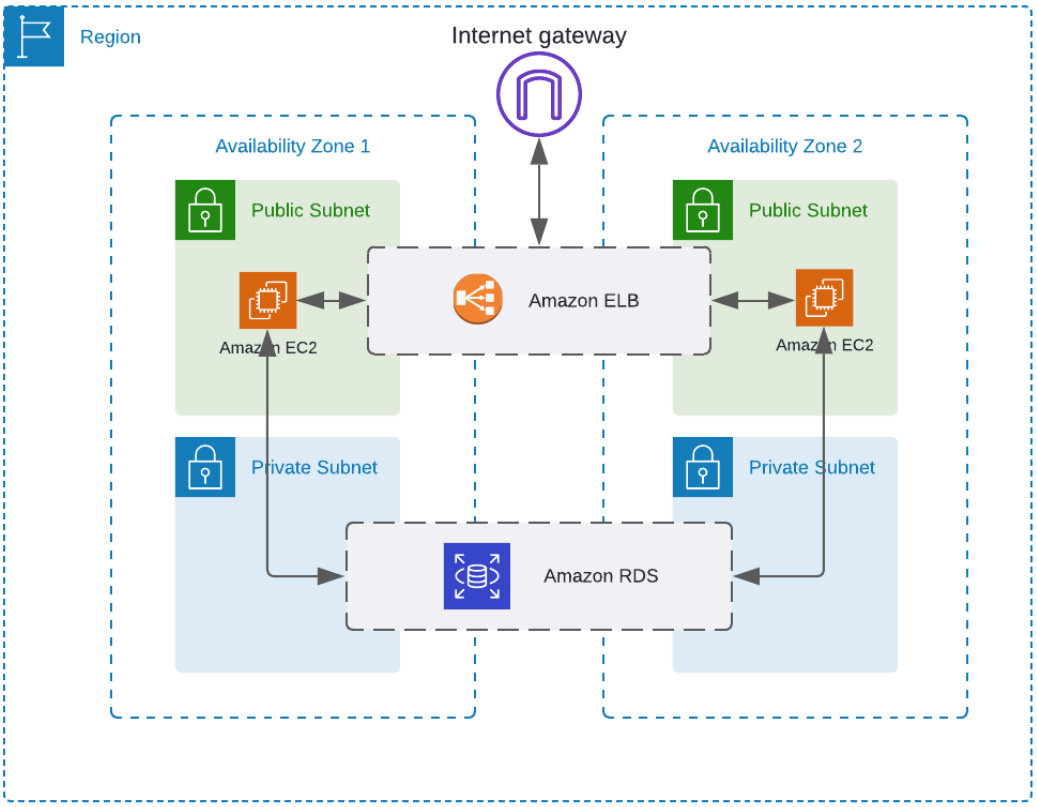

Figure 3 : VPC diagram

We have two EC2 instances in two separate availability zones, with an ALB accessible from the internet via an internet gateway, distributing the load. This setup provides high availability and fault tolerance, which, although not crucial for this project, is always a good practice for larger-scale projects. The two EC2 instances are in public subnets to allow access to Cognito, but, through a security rule, they are only accessible via the ALB.

Regarding the database, a multi-AZ configuration, as shown in Figure 3, would have been preferable for high availability and fault tolerance. However, for budgetary reasons, it is configured as a single-AZ, but it remains accessible by both EC2 instances, which does not pose an issue for our project, as it doesn't have high

availability requirements. Additionally, through a security rule, it is only accessible from these instances for security reasons.

3 - Database model

Initially, the data model schema was as follows:

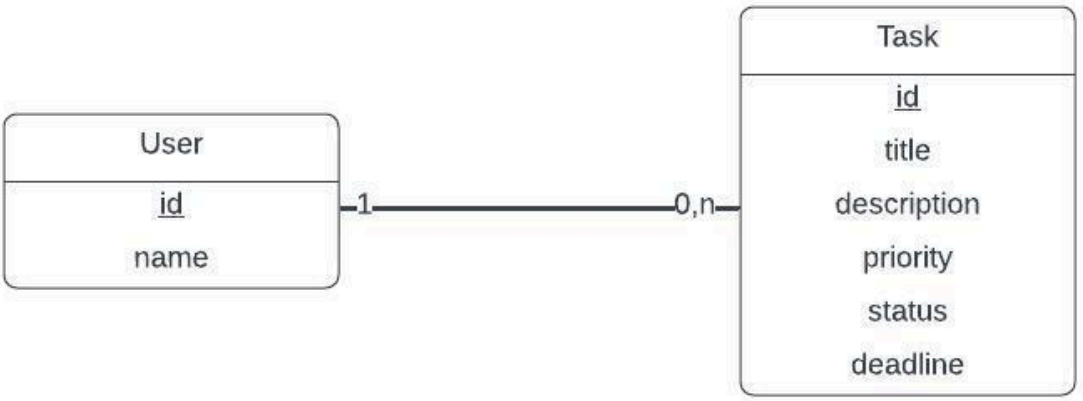

Figure 4 : First database model

However, during development, I realized that it was unnecessary to store user information since it is stored and provided by Cognito. As a result, the schema was simplified as follows:

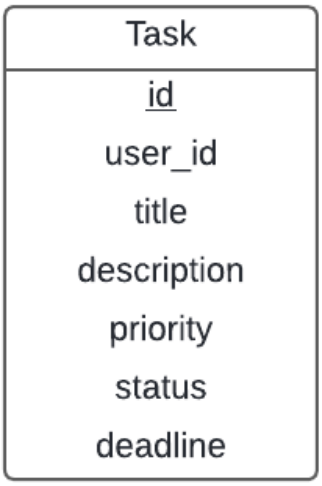

Figure 5 : Final database model

Tasks are simply stored with a user ID (provided by Cognito) to display only the tasks belonging to the correct user. The necessary information for each task is also stored.

IV - Implementation

1 - API

To develop the API, I chose to use Python with the FastAPI library for its ease of use and simplicity to learn. Each API feature (retrieve, add, update, delete, or mark tasks as complete) is implemented in a distinct function, each associated with a route, such as `/add_task`. These routes return a JSON response indicating whether the operation was successful or not. If necessary, the task ID to be processed is retrieved from the body of the HTTP request.

To execute the required functionality, the application interacts with the database. For this, the ORM SQLAlchemy is used, which establishes the connection to the database and performs read and write operations. Using an ORM eliminates the need to write raw SQL, offering several advantages:

- Simplifies data manipulation
- Reduces errors caused by incorrect SQL queries
- Makes the code more readable and maintainable
- Enhances security by preventing SQL injection attacks

Finally, to ensure the program starts automatically when the instance boots up, I added a service to handle this action, which also restarts the API in case it stops unexpectedly.

2 - Website

The UI was implemented using Jinja HTML templates to define the page structure and inject data dynamically with Jinja. When the user accesses the homepage, the API retrieves tasks from the database and injects them into the HTML page for display. In terms of code, this corresponds to a route in the API that returns the HTML page for the homepage.

The interface itself was built with Bootstrap to avoid manually handling CSS, saving time in the process.

To perform API functions, the interface is equipped with various buttons that send the appropriate request to the API along with the necessary information. The API processes the request and returns the updated homepage displaying the updated information.

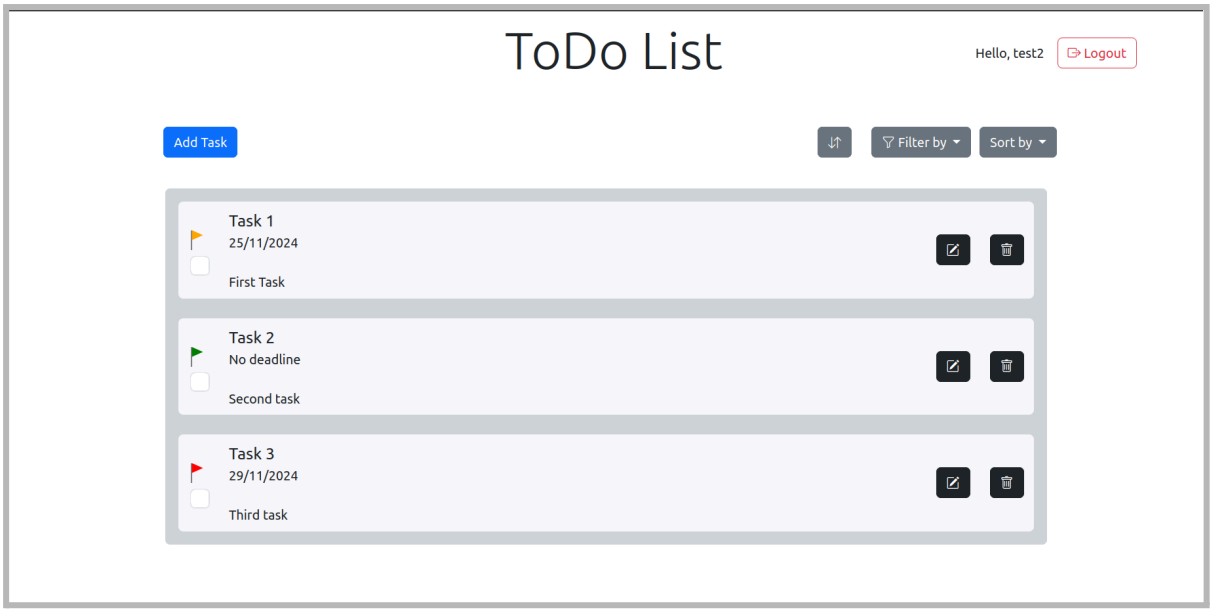

Figure 6 : ToDo List interface

3 - Authentication

As mentioned earlier, authentication is managed using the Cognito IDP. To access the website and API, the user must be logged in, meaning they must have a valid JWT token (stored in a cookie). If not, the user is automatically redirected to the Cognito login page. If someone tries to send a request directly to the API without going through the website, the API will return a 401 or 403 status code if the token is invalid or missing.

It is also possible to log out via a button in the interface, which deletes the session cookie and redirects the user to the Cognito page.

4 - Logging

To identify errors that may occur during program development, a logging system was implemented. This was done using syslog, so whenever an error occurs, it is recorded in the `/var/log/syslog` file of the EC2 instances. For added clarity, errors can also be viewed using the command `sudo systemctl status todolist`, where *todolist* is the name of the previously mentioned service.

```
Nov 11 15:45:36 ip-10-0-22-186 gunicorn[2788]: Error while decoding JWT: Signature has expired.
Nov 11 15:49:09 ip-10-0-22-186 gunicorn[2788]: Error while decoding JWT: Signature has expired.
```

Figure 7 : Example of logging

5 - Documentation

The project documentation, i.e. the launch of the project and the user story tests, was done using Docusorus, hosted on GitHub Pages at the following address `https://robinlafage.github.io/es-todolist/` .

V - Overcome difficulties

During the development of this project, I had to overcome several challenges. First, it was difficult to estimate the workload for each task, as it was my first time using these technologies.

Similarly, it was also challenging to estimate how much work could be completed within a sprint, as this depended on the previous point as well as the other subjects I had to work on in parallel.

Finally, the last challenge was using AWS, as I had never received any prior training on cloud microservices, but I was able to get by when I was stuck thanks to the well-supplied documentation, and the help of LLMs.

VI - What could have been done better

With hindsight and the knowledge gained from the courses and the Group Assignment Project, some aspects could have been better executed. This is particularly true for the distribution of work across sprints. I later realized that each sprint should deliver tangible value to the product from the user's perspective. It was therefore not ideal to dedicate an entire sprint to locally developing all the features, followed by a sprint solely focused on deployment. A better approach would have been to develop a limited set of features that were immediately accessible online from the start and then add and complete the remaining features in subsequent sprints.

VII - How to access the ToDo List

The ToDo List can be accessed at `es-ua.ddns.net` using the certificates supplied by the teacher. In the `/etc/hosts` file, this domain must be linked to the IP address associated with the ALB `todolist-alb-491005032.us-east-1.elb.amazonaws.com`.

VIII - Conclusion

To conclude, the project went quite well despite some areas that could be improved. This project and the course in general allowed me to discover the fundamentals of microservice architecture and cloud services, fields I was previously unfamiliar with, despite their widespread use in recent years.

Moreover, it helped me better understand the agile methodology and philosophy, which is widely used in companies, making it highly valuable for my future career.