MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 495

CLOUD COMPUTING

FALL 2024 – Final Report
ClearTasks

Team Members:

Raed H. Manna – 2550911

Omar A. Mourad - 2487080

`

Contents

1.Introduction

1.1 Brief overview:
The ClearTasks project is a cloud-based task management solution designed to provide seamless task tracking, secure authentication, and scalable backend services. This report highlights the work we completed so far, the challenges faced, and showcasing the ability to leverage cloud-based services to our advantage. The project is implemented using AWS (Amazon Web Services) as the cloud platform, adopting a Platform as a Service (PaaS) model.

1.2 Benefits of the Project:
ClearTasks goal is productivity, by making it easy to assign tasks to the appropriate individuals we introduce organization that can limit typical confusion experienced when being tasked a big project. We strive to help you feel accomplished after clearing your tasks and get you productive.

1.3 Novel Features/Ideas:

Integration of scalable Cloud Services:

- With ClearTasks we utilize AWS services such as Cognito, S3, and SNS for seamless integration, by combining such services we are open for scalability, security and real-time management of tasks.

User Friendly Interface through React:

- We leverage React to provide a clean and intuitive interface for user onboarding, as it is important, we make abstract complexity and simplify functionality.

Platform as a Service (PaaS) Adoption:

- ClearTasks is built on the PaaS model so that teams can focus on their development without worrying about any type of underlying infrastructure.

Security through Cognito:

- Thanks to AWS Cognito service we, can ensure a heightened security for user accounts, setting it apart from basic authentication services.

Our project’s contribution goes to show the value of using cloud services, as from now we also can advocate and have proof of its utility.

1.4 Similar Applications:

ClickUp: A productivity all-in-one platform that supports collaborative editing, task assignment, and time tracking, their project management service that has a wide variety of functionalities that help elevate and boost productivity. (ClickUp, n.d.).

Similarities-ClearTasks: Cloud based storage and management

Todo (by Microsoft):

A simple task management app integrated with Microsoft 365, allowing users to set various reminders. (Microsoft, n.d.).

Similarities-ClearTasks: Clou-based data synchronization.

1.5 GitHub Link: https://github.com/llTimell/ClearTasks

2. Project Structure:

Clear Tasks at its' core consists of three main parts: Frontend, Backend, and Cloud Services, each designed to work seamlessly to be operated smoothly.

2.1 Overview:

The project implements a task management system with admin and user functionalities. It features FASTAPI backend and a frontend developed using React.

Project Structure:

- Frontend (React):
  Mainly JavaScript heavy side with CSS to smooth out the visuals, the structure belongs in the client folder as you will find in our repository
  Dashboards, Login, Registration, and Task Management all is shown in the Frontend.

- Backend (FASTAPI):
  Python, along with FASTAPI which was structured so that we split and manage API endpoints, along with cloud interactions.
  Handling authentication, creation of task objects, updates and notification.

- Cloud Services (AWS):
  AWS services for storage, notifications, and user management.

Backend Components

Authentications Module (auth.py):

- Handling user login and registration
  with the use of AWS Cognito for secure authentication.
- Functions:
  - register(): registration for new users (normal users)
  - login (): Authenticates users and returns the access tokens for each user that is longlining in

Task Management Module(task.py):

- Manages task creation, retrieval, updates, and deletion, along with storing tasks in AWS s3 and SNS for notifying users for necessary operations.
- Functions:
  - Create_task (): for setting up the task object and notifying user that is receiving the task.
  - Get_tasks (): retrieve tasks.
  - delete_task(title): delete specific task.
  - Update_task_status (): updating status of a specified task.

Frontend Components

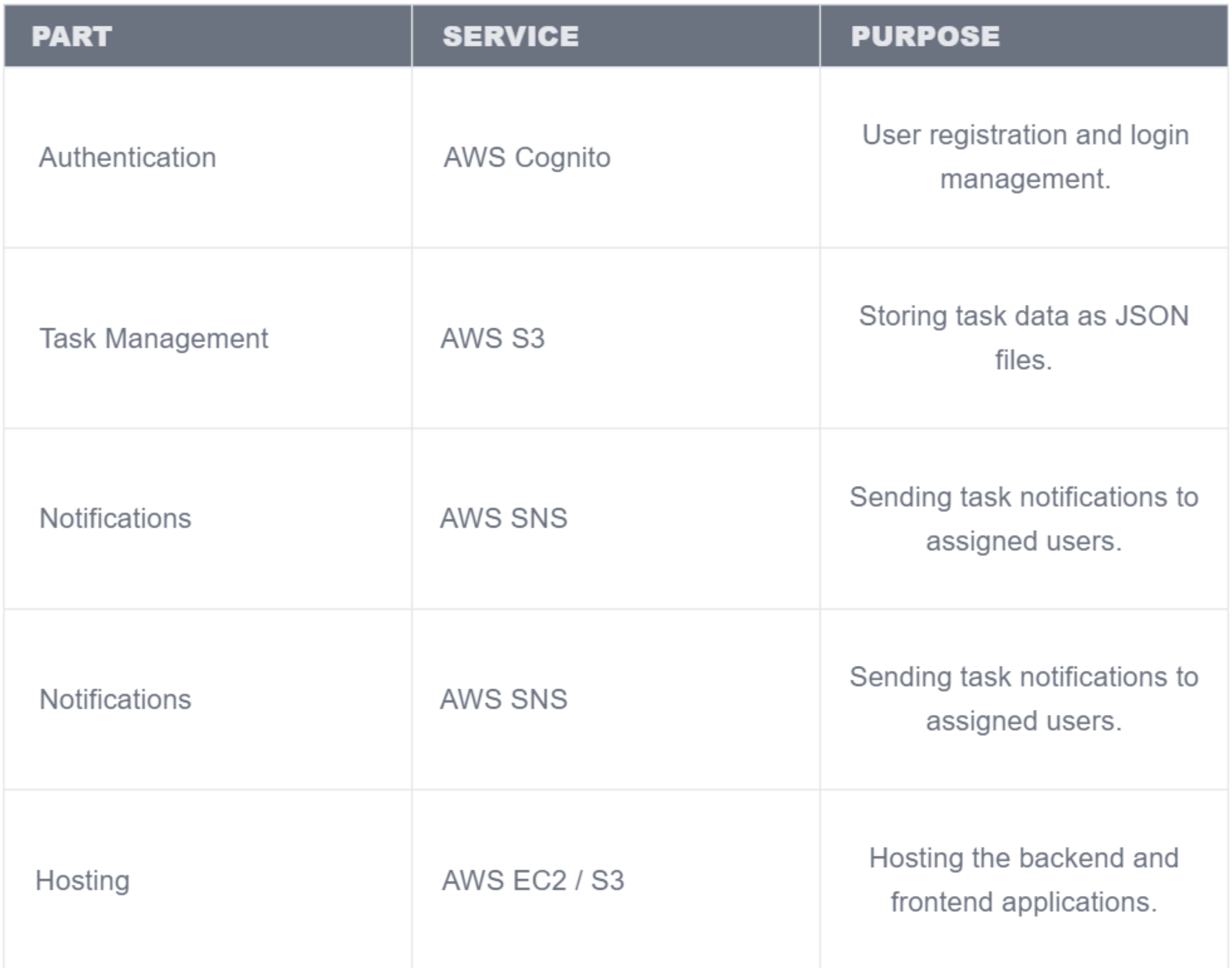
- Login Component (Login.js):
  - Allows users to login and store the tokens that get generated.

- Register Component (Register.js):
  - Facilitating the user's registration.

- Admin Dashboard (AdminDashboard.js):
  - Enables the creation of tasks and viewing created tasks.

- User Dashboard (UserDashboard.js):
  - Displays the tasks the user received and allow basic operations on task objects like update status.

2.2 Cloud Services Table



PART	SERVICE	PURPOSE
Authentication	AWS Cognito	User registration and login management.
Task Management	AWS S3	Storing task data as JSON files.
Notifications	AWS SNS	Sending task notifications to assigned users.
Notifications	AWS SNS	Sending task notifications to assigned users.
Hosting	AWS EC2 / S3	Hosting the backend and frontend applications.

2.3 User Manual:

In the User Manual, we will show examples for using both the endpoints and frontend.

Main Function Guide for Backend Endpoints:

1.Register new user using AWS Cognito.

In Endpoint: POST /auth/register

Example Input for Testing:

Example input

```
{
  "username": "raed_m",
  "password": "CoolPassword",
  "email": "raed@example.com"
}
```

2.Create new task and send notification.

In Endpoint: POST /tasks/create

Example Input for Testing:

Example input

```
{
  "title": "Finish Frontend",
  "description": "finish the dashboard",
  "deadline": "2024-12-19T23:59:59",
  "assigned_to": "raedmanna@yahoo.com",
  "created_by": "omar123@example.com",
  "priority": "High"
}
```

Main Function Guide for Frontend:

1.Login Page:

Straight forward input fields; it’s the only step between the user and Dashboards.

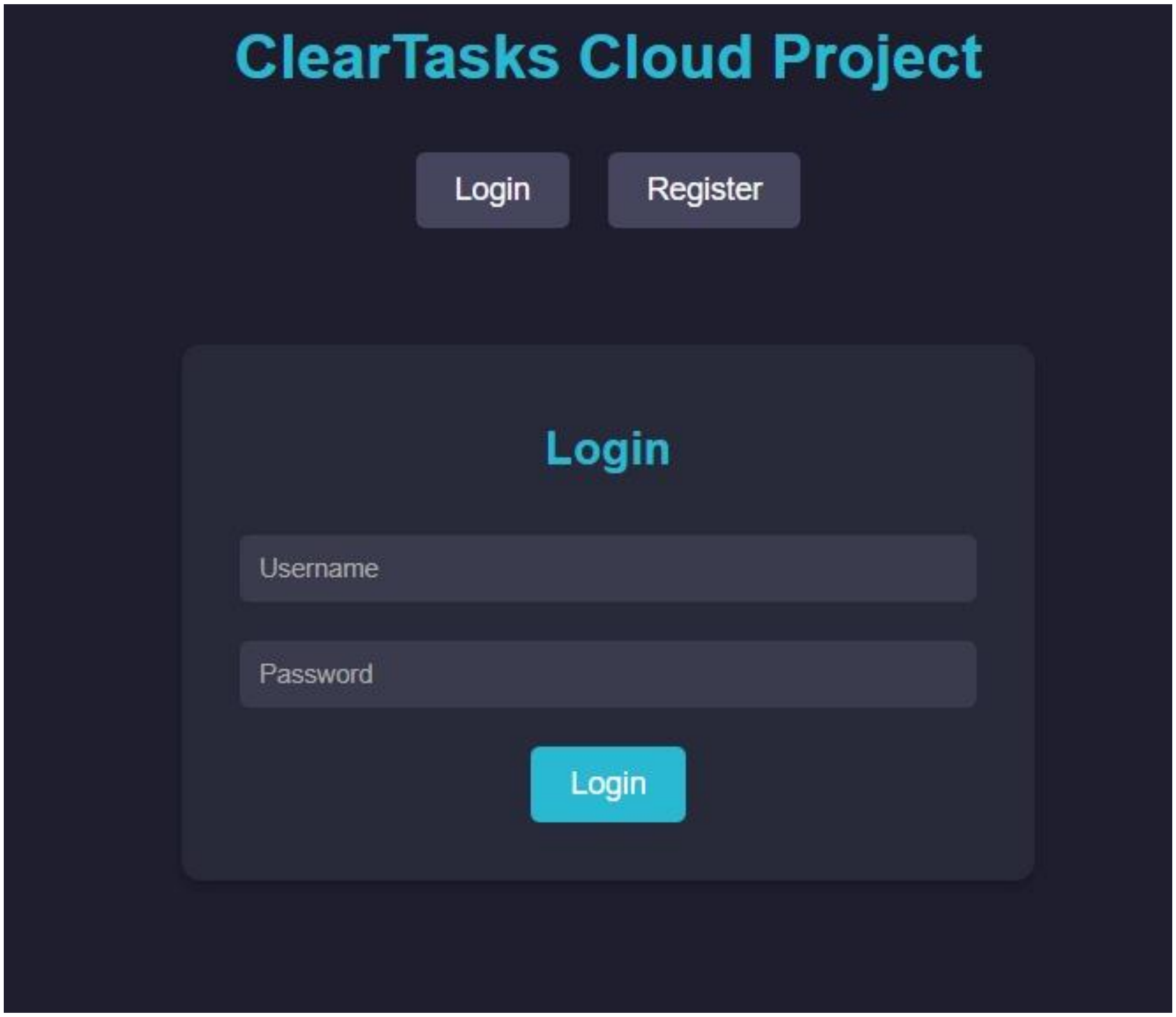
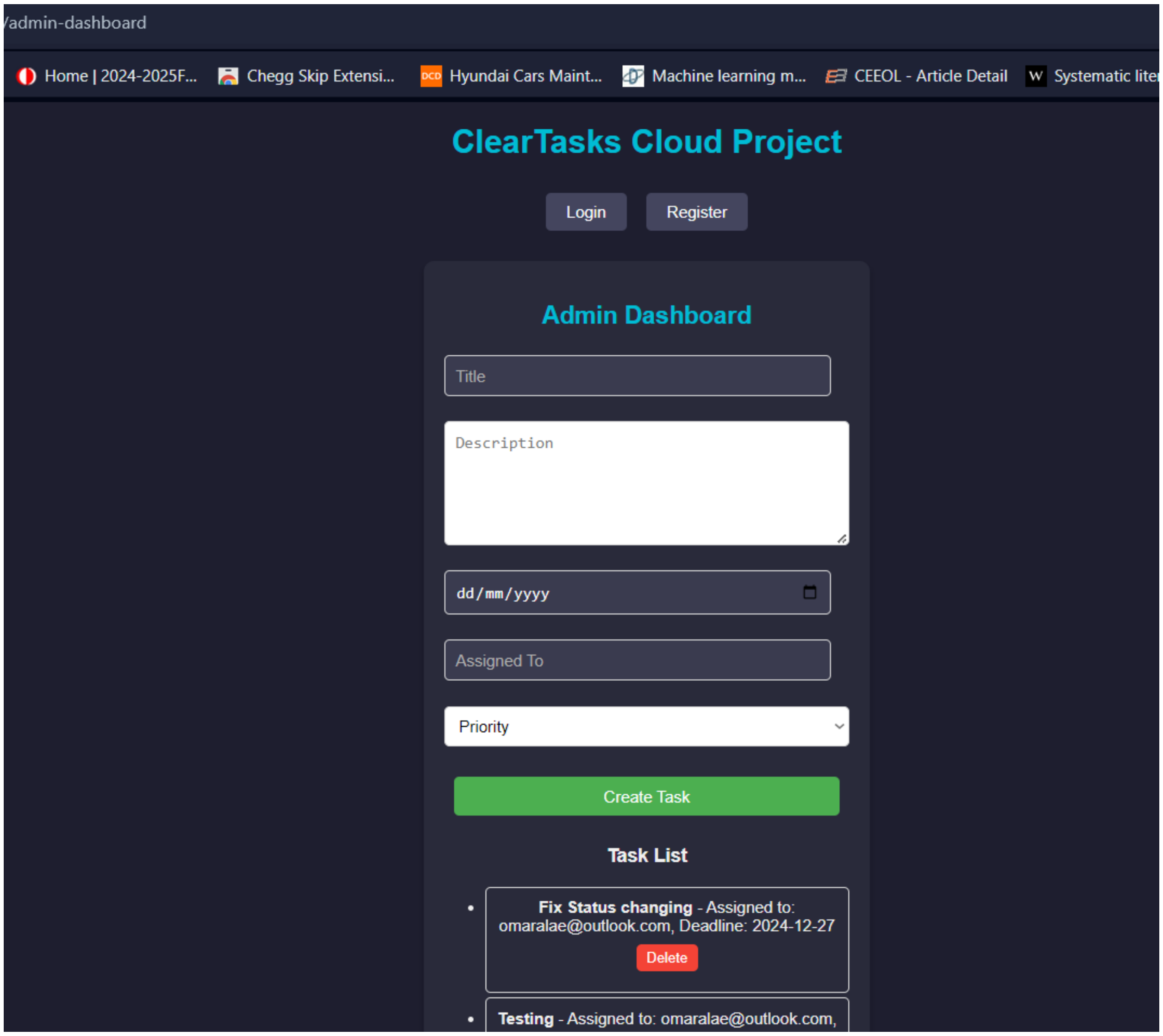


Figure 1: Login Page

2.Admin Dashboard:

Relevant to the Backend “Create new task and send notification”. Same input fields are used. The example used in backend can be used as long as the admin is logged in



/admin-dashboard

Home | 2024-2025F... Chegg Skip Extensi... Hyundai Cars Maint... Machine learning m... CEEOL - Article Detail Systematic lite

ClearTasks Cloud Project

Login Register

Admin Dashboard

Title

Description

dd/mm/yyyy

Assigned To

Priority

Create Task

Task List

- **Fix Status changing** - Assigned to: omaralae@outlook.com, Deadline: 2024-12-27 Delete
- **Testing** - Assigned to: omaralae@outlook.com,

Figure 2: Admin Dashboard Page

Admin Features Summary

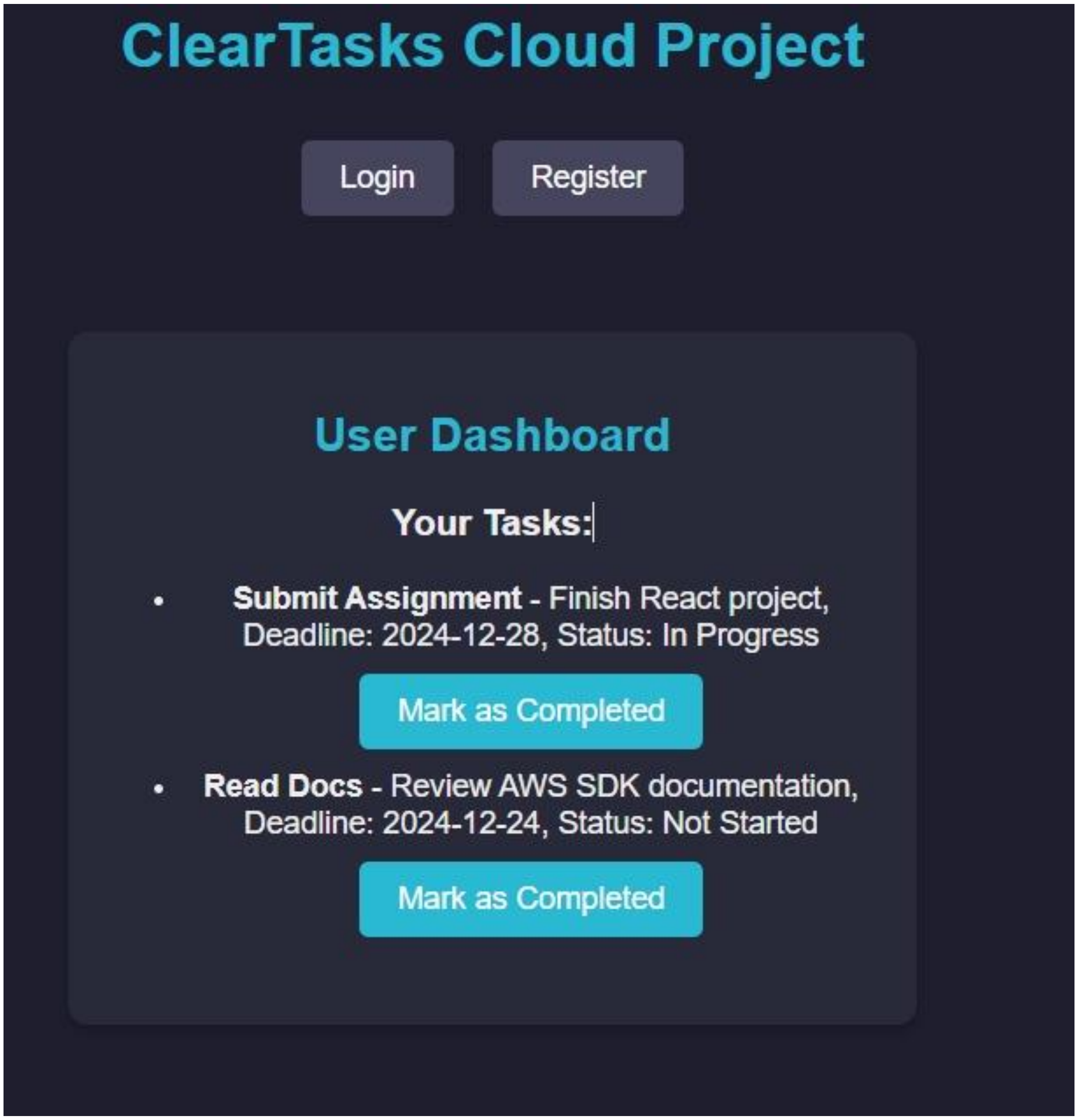
Feature	**Endpoint**	**Description**
Create Task	POST /admin/tasks/create	Admin creates tasks and stores them in S3.
Assign Task	PUT /admin/tasks/assign/{task_id}	Admin assigns a task to a user and sends notifications.
Monitor Progress	GET /admin/tasks/progress	Admin retrieves all tasks and monitor's progress.

User Features Summary

View Assigned Tasks	GET /tasks/tasks?user_email={email}	Fetches all tasks assigned to the user.
Update Task Status	PUT /tasks/update/{title}	Allows a user to update the status of their task.

3.User Dashboard:

- Lists tasks assigned for logged in user
- Allows for updates on status for tasks

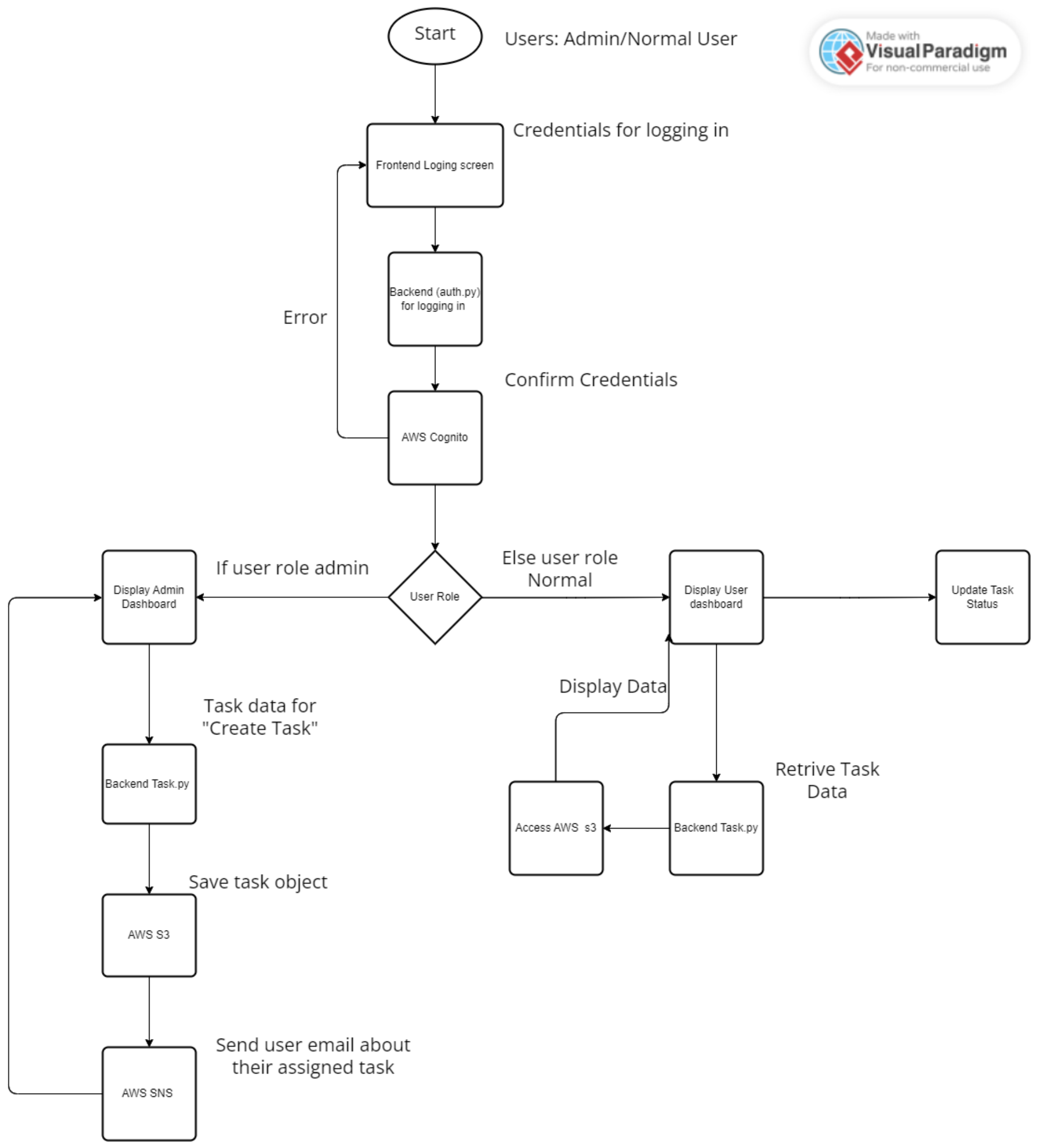


- **Submit Assignment** - Finish React project, Deadline: 2024-12-28, Status: In Progress

- **Read Docs** - Review AWS SDK documentation, Deadline: 2024-12-24, Status: Not Started

Figure 3: User Dashboard Page

2.4 Flow Chart Diagram:



2.5 Technologies Used:

2.5.1 Programming Language:

Backend:

Python was used with the FASTAPI framework to build API’s and backend logic.

Efficient and Fast, framework arguably was the least difficult to deal with since we have previous experience with setting up endpoints and coding with python, overall difficulties were at minimum.

Frontend:

JavaScript (with React): popular for building dynamic, responsive user interface, it also proved to have seamless integration with the Backend Api.

2.5.2. Cloud services:

- AWS Cognito: Service based for user authentication and authorization, used features were the User Pools, token generation mainly.

- AWS S3: Scalable storage service offered by amazon which was used to store the “Tasks” assigned to the users, CORS policies were needed for Frontend and Backend communication.

- AWS SNS: Notification service, to notify users to verify their emails after registering.

2.5.3. Developmental Tools:

IDEs:

PyCharm is the main one we are using as it has some convenient GitHub plugins and widely known for python development and package handling.

GitHub:

Repository version control and collaboration between developers made easy using GitHub.

2.5.4. Deployment:

Platform as a Service (PaaS):

- Using AWS based deployment meant we are providing a platform with scalable design as the way of deployment of ClearTasks.

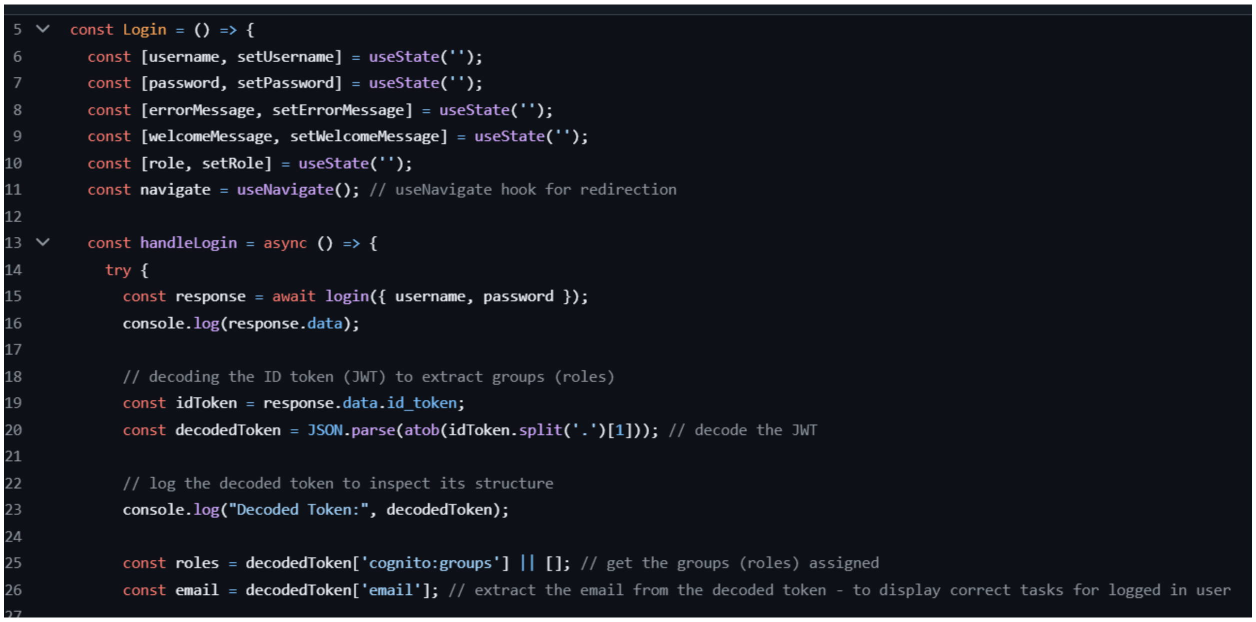
Security and Accessibility:

- **HTTPS with AWS Certificate Manager**:
    - Ensures secure communication between users and the application.
- **Load Balancer**:
    - Optional: AWS Application Load Balancer to distribute traffic evenly and ensure high availability.

2.6 Challenges:

General Issues:

a. **Login authentication:** While implementing login, the system needed to store and retrieve, the tokens (id_token, access_token, and refresh_token), which firstly wouldn't properly login to the admin or user.



```javascript
const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [errorMessage, setErrorMessage] = useState('');
  const [welcomeMessage, setWelcomeMessage] = useState('');
  const [role, setRole] = useState('');
  const navigate = useNavigate(); // useNavigate hook for redirection

  const handleLogin = async () => {
    try {
      const response = await login({ username, password });
      console.log(response.data);

      // decoding the ID token (JWT) to extract groups (roles)
      const idToken = response.data.id_token;
      const decodedToken = JSON.parse(atob(idToken.split('.')[1])); // decode the JWT

      // log the decoded token to inspect its structure
      console.log("Decoded Token:", decodedToken);

      const roles = decodedToken['cognito:groups'] || []; // get the groups (roles) assigned
      const email = decodedToken['email']; // extract the email from the decoded token - to display correct tasks for logged in user
```

Figure 4: saving tokens using JWT

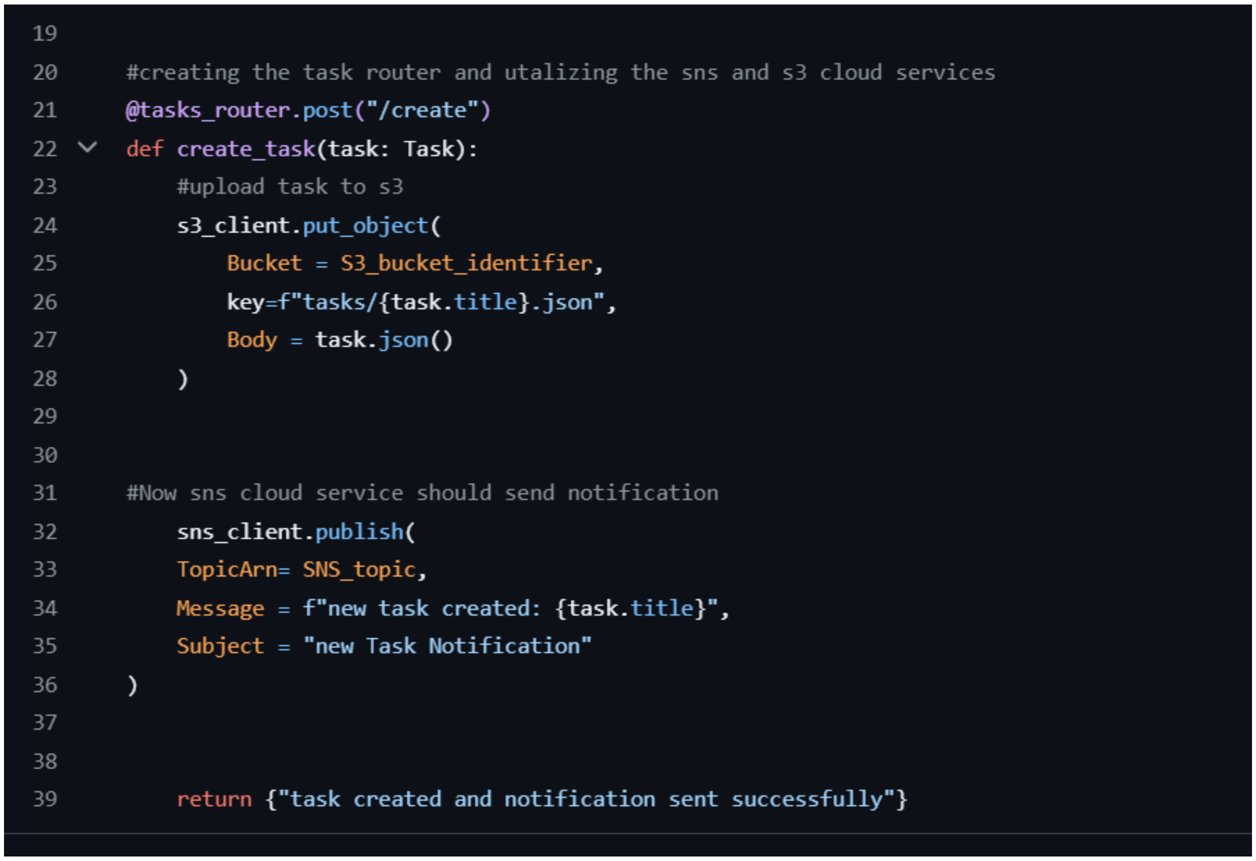
By decoding the Id, we can extract the information about the roles of who is logging in and properly route them to the right screens.

b. **Cloud services:**

Major hardship was had using these cloud services, we had clashing policies on our services that wouldn't allow us the Root users to access our own S3 buckets, and our lack of knowledge on the matter didn't help so we opted to using Root user credentials instead of IAM credentials to do our bucket operations.

The access key generated by Root User have zero restrictions which got us starting the project.

AWS SNS: We had to do a lot of code fixing to be able to receive emails. It took a good portion of our time to figure out the right credentials for such operation.



```

    #creating the task router and utalizing the sns and s3 cloud services
    @tasks_router.post("/create")
    def create_task(task: Task):
        #upload task to s3
        s3_client.put_object(
            Bucket = S3_bucket_identifier,
            key=f"tasks/{task.title}.json",
            Body = task.json()
        )


    #Now sns cloud service should send notification
        sns_client.publish(
        TopicArn= SNS_topic,
        Message = f"new task created: {task.title}",
        Subject = "new Task Notification"
    )


        return {"task created and notification sent successfully"}
```

Figure 5: SNS_topic solution

When we properly used the right Topic ARN from the AWS console, we were able to receive emails regarding the creation of tasks.

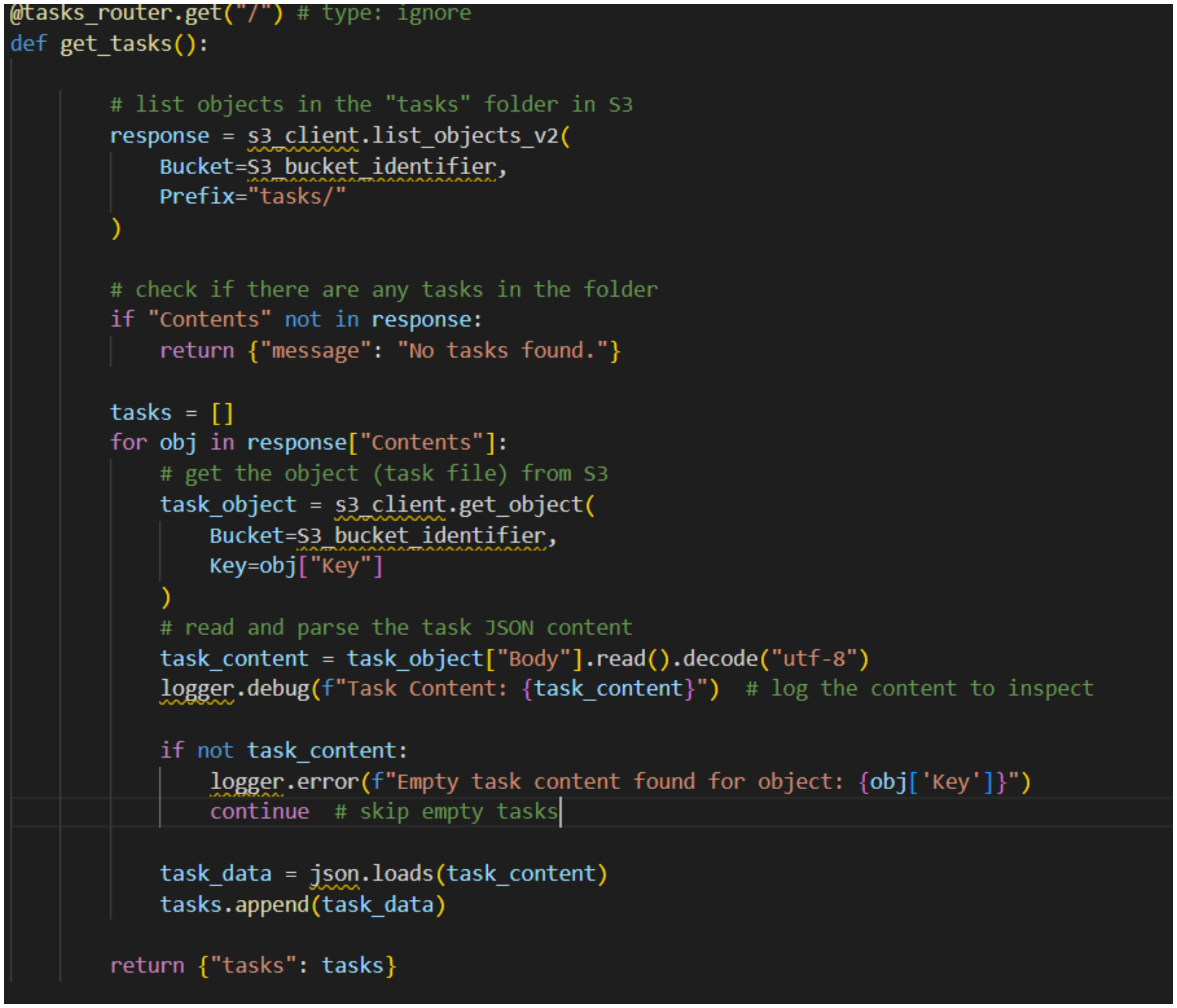
Difficulties of Each Part:

Backend:

Tasks.py:

This proved to be difficult since it was the heart of the cloud service operations communications. We had to separate functions outside our endpoints to help us, especially in get_tasks since that function had many connections to the services.

Firstly, we have to retrieve the tasks from the S3 bucket, and check the objects are not empty nor have wrong formatting.



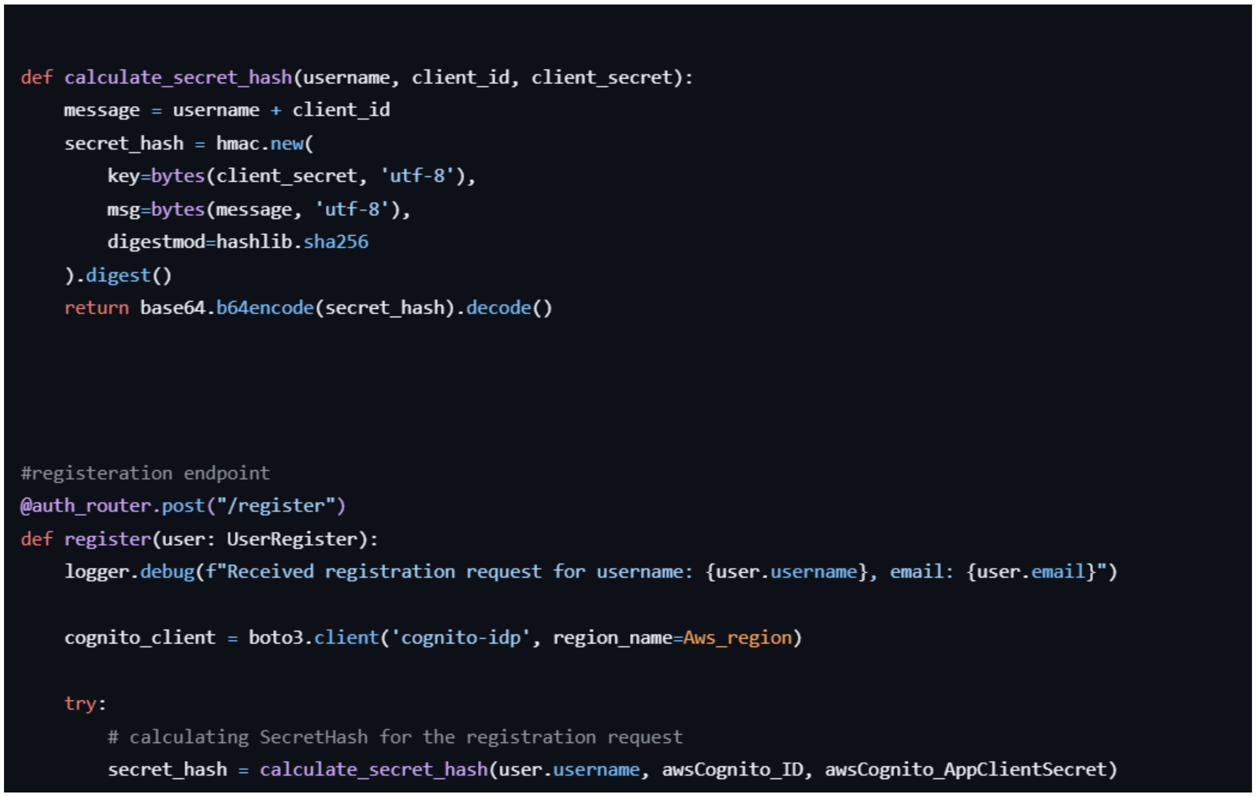
```
@tasks_router.get("/") # type: ignore
def get_tasks():

        # list objects in the "tasks" folder in S3
        response = s3_client.list_objects_v2(
            Bucket=S3_bucket_identifier,
            Prefix="tasks/"
        )

        # check if there are any tasks in the folder
        if "Contents" not in response:
            return {"message": "No tasks found."}

        tasks = []
        for obj in response["Contents"]:
            # get the object (task file) from S3
            task_object = s3_client.get_object(
                Bucket=S3_bucket_identifier,
                Key=obj["Key"]
            )
            # read and parse the task JSON content
            task_content = task_object["Body"].read().decode("utf-8")
            logger.debug(f"Task Content: {task_content}")  # log the content to inspect

            if not task_content:
                logger.error(f"Empty task content found for object: {obj['Key']}")
                continue  # skip empty tasks

            task_data = json.loads(task_content)
            tasks.append(task_data)

        return {"tasks": tasks}
```

Figure 6: get_task function

Auth.py:

We had to calculate the secret hash for when a user decides to register as part of the attributes that makes a user is a secure hash composed of their username and the client id

This took some research to figure and some interesting documentation reading.



```python
def calculate_secret_hash(username, client_id, client_secret):
    message = username + client_id
    secret_hash = hmac.new(
        key=bytes(client_secret, 'utf-8'),
        msg=bytes(message, 'utf-8'),
        digestmod=hashlib.sha256
    ).digest()
    return base64.b64encode(secret_hash).decode()




#registeration endpoint
@auth_router.post("/register")
def register(user: UserRegister):
    logger.debug(f"Received registration request for username: {user.username}, email: {user.email}")

    cognito_client = boto3.client('cognito-idp', region_name=Aws_region)

    try:
        # calculating SecretHash for the registration request
        secret_hash = calculate_secret_hash(user.username, awsCognito_ID, awsCognito_AppClientSecret)
```

Figure 7: Hashing function

Frontend

Problem: The user dashboard needed to display tasks assigned to the logged-in user on their email addresses. Handling API calls for proper token usage and data fetching was a challenge.

Solution:

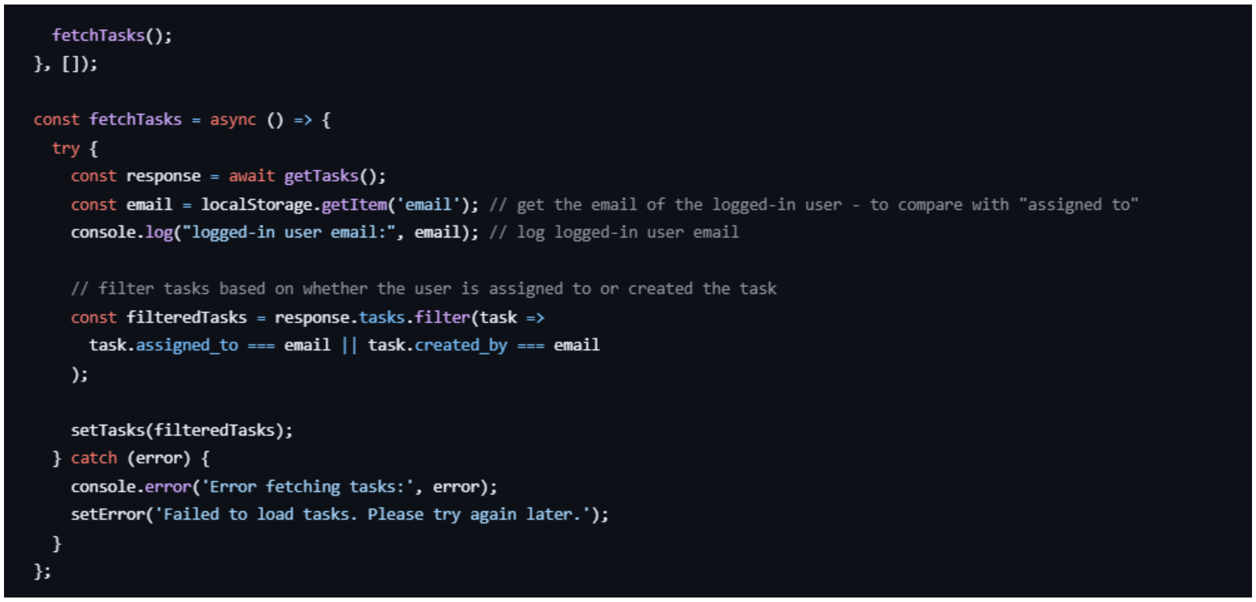

```
  fetchTasks();
}, []);

const fetchTasks = async () => {
  try {
    const response = await getTasks();
    const email = localStorage.getItem('email'); // get the email of the logged-in user - to compare with "assigned to"
    console.log("logged-in user email:", email); // log logged-in user email

    // filter tasks based on whether the user is assigned to or created the task
    const filteredTasks = response.tasks.filter(task =>
      task.assigned_to === email || task.created_by === email
    );

    setTasks(filteredTasks);
  } catch (error) {
    console.error('Error fetching tasks:', error);
    setError('Failed to load tasks. Please try again later.');
  }
};
```

Figure 8: Fetch Tasks

The fetchTasks function was our solution to properly filter based on the logged in user, localStorage.getitems('email') is how we previously stored the user's email and we retrieve it on the UserDashboard.js to dynamically display the tasks.

3. Project Statistics

3.1 Development Time and Delays

Planned Tasks and Responsibilities Previous report:

Date/Week December 2 – December 8:

Milestone Add missing backend Endpoints for the main features

December 9 - December 15:

Complete the main features in task.py + refine FrontEnd

December 16 - December 22:

Final integration, testing and bug fixes

Due to delays we had to push:

Complete the main features in task.py + refine FrontEnd → **December 11 - December 20**

Task.py and AWS configurations took longer than expected

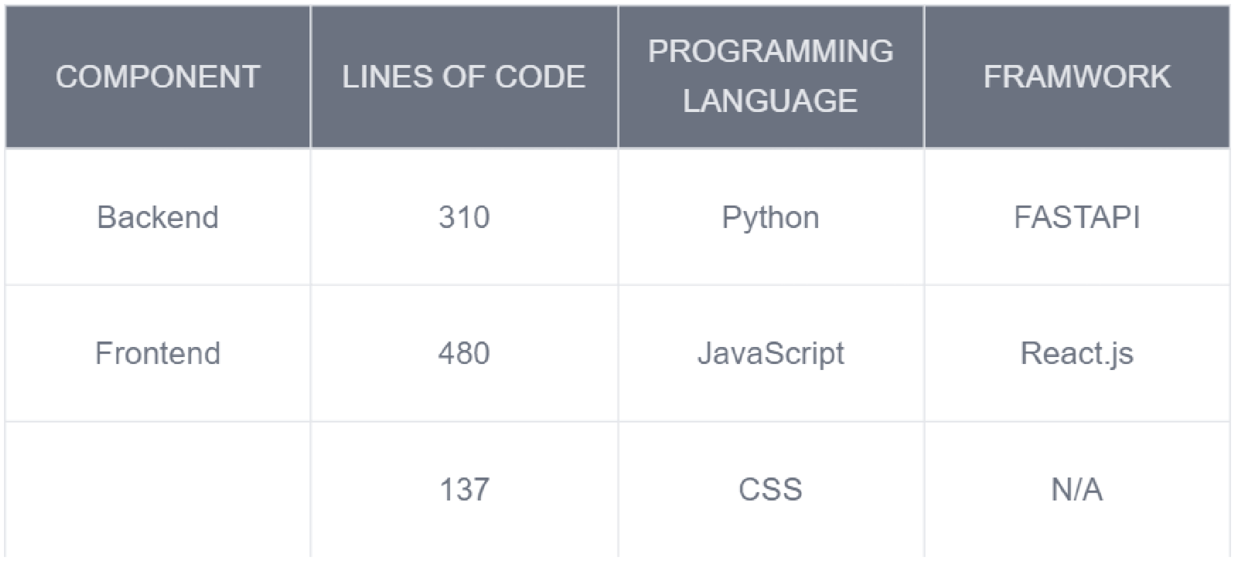
Final integration, testing and bug fixes → **December 20 - December 24**

Bugs were across multiple frontend components and some of the API calls had issues

Frontend and API calls Plus auth.py files were Raed H. Manna

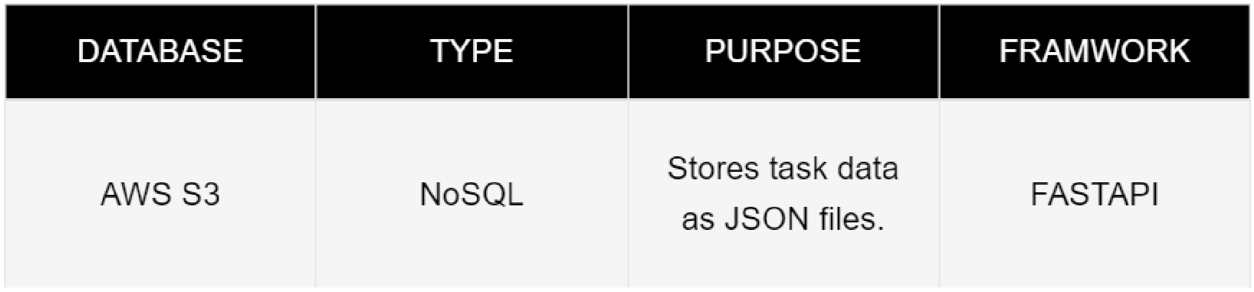
Tasks.py and plus relative API calls were Omar A. Mourad

3.2 Lines of Code:



COMPONENT	LINES OF CODE	PROGRAMMING LANGUAGE	FRAMWORK
Backend	310	Python	FASTAPI
Frontend	480	JavaScript	React.js
	137	CSS	N/A

3.3 Database Information:



DATABASE	TYPE	PURPOSE	FRAMWORK
AWS S3	NoSQL	Stores task data as JSON files.	FASTAPI

- For AWS S3 we are allowed to store up to 5 GB standard storage
    - Requests: 20,000 GET requests & 2000 PUT requests
    - Data Transfer 15 GB transfer Per/month

- For AWS SNS:
    - Requests: 1 million publishes per month
    - HTTP/S Deliveries: 100,000 deliveries per month
    - Email Deliveries: 1000 deliveries per month
    - SMS Deliveries: 100 per month

References

Amazon Web Services. (n.d.). *Amazon Cognito documentation*. AWS Documentation. Retrieved from https://docs.aws.amazon.com/cognito

Amazon Web Services. (n.d.). *Amazon S3 documentation*. AWS Documentation. Retrieved from https://docs.aws.amazon.com/s3

Amazon Web Services. (n.d.). *Amazon SNS documentation*. AWS Documentation. Retrieved from https://docs.aws.amazon.com/sns

Amazon Web Services. (n.d.). *Amazon CloudShell limits*. AWS Documentation. Retrieved from https://docs.aws.amazon.com/cloudshell/latest/userguide/limits.html

Newsletter AWS Fundamentals. (n.d.). SNS unpacked: From pub/sub patterns to pricing. Retrieved from https://newsletter.awsfundamentals.com/posts/sns-unpacked-from-pub-sub-patterns-to-pricing

ClickUp. (n.d.). *ClickUp app documentation*. Retrieved from https://clickup.com

Microsoft. (n.d.). *Microsoft To Do: Stay organized*. Retrieved from https://todo.microsoft.com

React. (n.d.). *ReactJS documentation*. Retrieved from https://reactjs.org

Tiangolo, S. (n.d.). *FastAPI documentation*. Retrieved from https://fastapi.tiangolo.com

JetBrains. (n.d.). *PyCharm IDE*. Retrieved from https://www.jetbrains.com/pycharm

Microsoft. (n.d.). *Visual Studio Code documentation*. Retrieved from https://code.visualstudio.com