



PRACTICAL TRAINING-I

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD
OF
DEGREE OF BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE
ENGINEERING



Submitted By:

Name: Pulkit

University/College Roll No. 25127

**SUBMITTED TO: Dr. Ashima Mehta
(HOD)**

Department of Computer Science & Engineering



DRONACHARYA COLLEGE OF ENGINEERING

KHENTAWAS, GURGAON, HARYANA

DRONACHARYA

College of Engineering

PRACTICAL TRAINING-1

Full Stack Engineer

Submitted in partial fulfillment of the
Requirements for the award of
Degree of Bachelor of Technology in Computer Science Engineering



Submitted By

Name: **Pulkit**

University/College Roll No: **25127**

Submitted To

Dr. Ashima Mehta

(HOD)

Department of Computer Science & Engineering

GURUGRAM UNIVERSITY, GURUGRAM

(HARYANA)



STUDENT DECLARATION

I hereby declare that the Practical Report entitled PRACTICAL TRAINING-I is an authentic record of my own work as requirements of 5th/6th/7th semester during the period from 11.06.2024 to 11.07.2024 for the award of degree of B.Tech. (Computer Science & Engineering), Dronacharya College of Engineering.

Pulkit
25127

Date:

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Signatures

Examined by:

Head of Department
(Signature and Seal)



Acknowledgement

I would like to express my profound gratitude to Mr. Prateek Dubey, Founder and CTO, for his exceptional guidance and valuable on-call support throughout my internship. His insights and advice were instrumental in the successful completion of this project. I am also deeply thankful to my teachers, Prof. Dr. Ashima Mehta and Prof. Ms. Vimmi Malhotra of Dronacharya College of Engineering, for their exemplary guidance, unwavering support, and encouragement throughout the course of this project. Their blessings and assistance have greatly contributed to my development and will continue to inspire me as I move forward in my career. Additionally, I must acknowledge the faculties and staff of Dronacharya College of Engineering for their continuous guidance and teaching support, which were crucial in the successful completion of this training report. Finally, I extend my heartfelt thanks to my colleagues and friends for their constant support and motivation. Their encouragement and help were invaluable in completing this report

Pulkit

Computer Science Engineering (CSE)

Roll No: 25127



About the Company

Datawave Labs, established in 2024 by a team of seasoned technologists, is driven by a singular vision: to harness the transformative power of AI, Cloud, and Data. We are passionate about unlocking the immense potential embedded in the vast volumes of information that businesses gather daily. Our mission is to empower organizations to turn this data into actionable insights, driving growth, innovation, and competitive advantage. In today's fast-paced and ever-evolving technological landscape, businesses face the daunting task of keeping up with emerging technologies. At Datawave Labs, we pride ourselves on being more than just a technology provider; we are your strategic partner in navigating the complexities of the modern digital era. Our expertise spans across key areas such as Data Engineering, Machine Learning, Artificial Intelligence, Large Language Models (LLMs), Cloud Computing, and DevOps. By staying at the forefront of these technologies, we ensure that our clients are equipped with the tools and knowledge they need to thrive in their respective industries. Our approach is rooted in a deep love for technology. However, what truly excites us is the application of these technologies to real-world problems, particularly when they drive tangible improvements in our clients' businesses and overall well-being. We believe that the intersection of emerging technologies is where the most groundbreaking and disruptive solutions are born. Whether it's streamlining operations through advanced data engineering practices, enhancing decision-making with machine learning models, or ensuring seamless scalability with cloud solutions, our goal is to deliver innovation that makes a difference. At Datawave Labs, we recognize that every business is unique. That's why we offer tailored solutions designed to meet the specific needs and challenges of each client. Our collaborative approach ensures that we work closely with you to understand your goals, identify opportunities, and implement solutions that deliver measurable results. We are committed to providing the highest level of service, leveraging our technical expertise to help you achieve success in a rapidly changing world. We invite you to explore how Datawave Labs can help your organization harness the power of AI, Cloud, and Data to achieve your strategic objectives. Contact us today to start a conversation about how we can partner with you to drive innovation and create value in your business. Together, we can turn the potential of technology into a powerful force for growth and success.

DRONACHARYA

College of Engineering

Certificate



11th August 2024

Dear Pulkit,

CERTIFICATE OF EMPLOYMENT

This is to certify that Pulkit (PAN No.: [REDACTED]) is employed at Datawave Labs Pvt. Ltd. since 7 April 2024 as a Full Stack Engineer (Intern).

Roles & Responsibilities

- Assisted in the development of Datawave Labs product and its features.
- Resolved bugs and issues as reported by the team and users.
- Showcased problem-solving, communication, and time-management skills.
- Showcased interest, willingness, and ability to learn new programming languages.
- Used JavaScript, Python for backend development.
- Use Node.js, React for frontend development.
- Explored Cloud (AWS), Docker, Kubernetes, DevOps, etc.

Should you require further information, please do not hesitate to contact us at [REDACTED] or you may email us at [REDACTED]

Yours sincerely,

.....
Prateek Dubey
Founder & CTO
Datawave Labs Pvt. Ltd.

Contents

1. Introduction.....	2
1.1 Overview.....	3
1.2 Project Overview.....	4
1.3 Technologies Used.....	6
2. Authentication and Authorization.....	9
2.1 User Sign-up and Login.....	10
2.2 Google Sign-in Integration.....	13
3. User Account Management.....	18
3.1 Username Management.....	19
3.2 Password Management.....	20
4. Cloud Account Integration.....	22
4.1 AWS Integration.....	23
4.2 Azure Integration.....	24
4.3 GCP Integration.....	26
5. Cluster Management.....	28
5.1 Cluster Creation Workflow.....	29
5.2 Authentication & Authorization.....	32
5.3 Real-Time Status Updates.....	33
6. Server Development	36
6.1 Database Development with Psycpg2.....	37
6.2 API Testing with Postman.....	38

7. Notification System	39
7.1 Overview of Notification Mechanism	40
7.2 Integration with Services	40
8. Conclusion	42
8.1 Summary of Work Done	43
8.2 Lessons Learned	44
8.3 Challenges Faced	44
9. References	46

List of Figures

2.1 Login Page.....	10
2.2 User Schema.....	11
2.3 Authentication Cookies.....	13
2.4 Authentication Token Object.....	14
2.5 Google User Information.....	14
2.6 Using cookies with useCookies hook.....	15
2.7 Sending headers from client.....	16
2.8 Creation of Authentication token.....	16
2.9 Removing Cookies for logout.....	17
3.1 Database User Schema.....	20
4.1 Verifying AWS account.....	24
4.2 Verifying Azure account.....	25
4.3 Verifying GCP account.....	26
5.1 Add Cluster Page.....	30
5.2 Initializing a Background Process.....	31
5.3 Using python sdk to Run Docker Container.....	31
5.4 Using Tokens from Auth Headers.....	33
5.5 Implementation of SSE.....	34
5.6 Client Side handing SSE with EventSourcePolyfill.....	34
5.7 Extracting messages from event source.....	35
5.8 Handling Errors.....	35
6.1 Creating Database Connection with Psycpg2.....	37
6.2 Postman API Collection.....	38



Chapter 1

Introduction

1.1 Overview

1.2 Project Objectives

1.3 Technologies Used

1. Introduction

1.1 Overview

During my internship at Datawave Labs, I had the opportunity to gain hands-on experience as a Full Stack Engineer, immersing myself in a dynamic and collaborative environment. The internship provided a perfect balance between learning and practical application, allowing me to engage in meaningful tasks while also managing my academic commitments. I was required to contribute 20 hours weekly, which was manageable due to the supportive nature of my team and the flexible work environment. Each day we had daily standup calls, a cornerstone of our workflow that kept the team aligned with project goals and individual progress. These meetings were brief yet effective, allowing each team member to provide updates on their tasks, discuss any blockers, and plan the day's work. The collaborative nature of these meetings ensured that everyone was on the same page and helped to foster a sense of accountability and teamwork. Additionally, weekly group meetings were held to discuss broader project objectives, review milestones, and plan upcoming sprints. These meetings provided a platform for sharing ideas, discussing challenges, and aligning on long-term goals. One of the key aspects of my internship was the autonomy I was given in completing my tasks. While I was assigned specific responsibilities, the level of monitoring was minimal, which allowed me to take ownership of my work. This approach not only boosted my confidence but also encouraged me to develop problem-solving skills, as I was often required to find solutions independently before seeking help. However, the lack of constant supervision did not mean that support was unavailable. My manager was incredibly approachable and understanding, offering guidance whenever needed. His leadership style was one of empowerment, trusting that I would manage my time effectively and produce quality work. This trust was particularly important as it allowed me to balance my college studies with the demands of the internship. The supportive environment extended beyond just the tasks at hand. My manager's flexibility in accommodating my academic schedule was crucial in ensuring that I could meet my obligations both at school and at work. This understanding created a work atmosphere where I felt valued not just as an intern, but as an integral part of the team. This experience taught me the importance of time management and prioritization, skills that are essential in any professional setting. Throughout my internship, I was introduced to various industry-standard tools that enhanced our productivity and collaboration. Slack, a messaging platform widely used for team communication, became an essential part of our daily operations. It facilitated real-time communication, allowing for quick resolution of issues and fostering a collaborative team environment. I also gained experience with Notion,

a versatile tool used for project management and documentation. Notion allowed us to organize our tasks, track progress, and maintain project documentation in a centralized location. These tools not only streamlined our workflow but also provided a glimpse into the efficiency tools that drive modern workplaces. In summary, my internship at DWL was a comprehensive learning experience that combined practical skill development with professional growth. The blend of daily standup calls, group meetings, and the autonomy to manage my tasks provided a well-rounded introduction to the workflow of a tech company. The supportive and flexible environment, coupled with the use of industry-leading tools like Slack and Notion, made this internship an invaluable step in my professional journey.

1.2 Project Overview

The project at Datawave Labs was structured to align with the company's mission of leveraging cutting-edge technologies to drive business innovation. My role as a Full Stack Engineer was pivotal in developing and integrating various systems that are integral to modern data-driven operations. Below are the key objectives that guided the project:

1. Enhance User Authentication and Authorization Systems

- **Implement Login/Signup Flows:** Develop a secure authentication system using both traditional email/password methods and modern OAuth2 with Google Sign-In, ensuring users have multiple, secure entry points.
- **Session and Token Management:** Employ JWT for session management, coupled with cookie-based authentication, to ensure security and scalability in user sessions.
- **User Verification Process:** Streamline the user verification process with email verification and features like resending verification links and authorized only access to resources.
- **Role-Based Access Control:** Implement middleware to enforce role-based access, ensuring that users only access resources appropriate to their roles within the application.

2. Seamlessly Integrate Cloud Accounts

- **AWS, GCP, and Azure Integration:** Build interfaces for securely adding cloud accounts using various credentials, such as AWS Access/Secret Keys, GCP JSON data, and Azure Tenant IDs. This allows for versatile and secure cloud resource management.

- **Connector Implementation:** Develop and validate connectors for essential cloud services like AWS S3, Azure Storage Accounts, and Google Cloud Storage. This integration enhances the system's ability to interact with various cloud services, providing a unified management platform.

3. Automate Infrastructure Management

- **Cluster Management Automation:** Automate the creation and deletion of clusters using Docker images, reducing manual effort and minimizing the risk of human error in infrastructure management.
- **Real-Time Status Updates:** Implement live status updates for cluster operations using Event Source connections, providing continuous feedback on infrastructure states, which is critical for timely decision-making.

4. Develop a Comprehensive Notification System

- **Redis Pub/Sub for Notifications:** Integrate a notification system using Redis Pub/Sub to handle real-time notifications efficiently. This system supports functionalities like marking notifications as read and deleting them, enhancing user experience.
- **Notification Management:** Create APIs for managing notifications, ensuring that users are kept informed about critical system events and updates.

5. Enhance System Security

- **Implement Advanced Security Features:** Focus on strengthening security through the implementation of secret encryption, token protection mechanisms, and regular vulnerability scanning. This ensures that the system is resilient against potential threats and maintains user trust.

6. Optimize the Client User Experience

- **Responsive and Accessible UI:** Develop a responsive client using ReactJS and Tailwind, ensuring the application is accessible across different devices and user environments.
- **Integration with Cloud Services:** Implement seamless UI interactions with cloud routes and notification systems, ensuring that users can manage their resources efficiently and receive timely updates.

These objectives were designed to align with Datawave Labs' commitment to leveraging emerging technologies to drive innovation and efficiency in business operations. The project not only provided me with practical experience in handling real-world challenges but also contributed significantly to the company's goal of creating impactful technological solutions for its clients.

1.3 Technologies Used

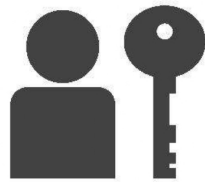
During my internship at Datawave Labs, I had the opportunity to work with a wide array of technologies that are fundamental to modern full-stack development and DevOps. Below is a breakdown of the key technologies and their purposes within the project:

- Client
 - **ReactJS**: A powerful JavaScript library for building user interfaces, especially single-page applications. ReactJS was utilized to create dynamic and responsive web components, ensuring a seamless user experience.
 - **Recoil**: This state management library for React allowed efficient handling of the application's state. It provided a flexible and scalable solution for managing global state across various components.
 - **Tailwind CSS**: A utility-first CSS framework that helped in designing and styling the application. Tailwind enabled the rapid creation of visually appealing and responsive interfaces with minimal custom CSS.
 - **EventSourcePolyfill**: This library was essential in managing Event Source connections, particularly when dealing with authorization headers. It ensured that real-time updates from the server were handled efficiently.
 - **react-cookie**: A library used for managing cookies within React. It was particularly useful for handling user authentication states and session management on the client side.
- Server
 - **FastAPI**: A modern, fast (high-performance) web framework for building APIs with Python. FastAPI was chosen for its simplicity and speed, enabling the rapid development of robust server services.

- **Event Source:** This was used for server-sent events (SSE), allowing the server to push real-time updates to the client. It played a crucial role in keeping the client updated with the latest status changes without needing to refresh the page.
- **Poetry:** A dependency manager for Python that streamlined the process of managing project dependencies, packaging, and publishing. Poetry ensured that the project had consistent and reliable dependency management.
- **boto3:** The AWS SDK for Python was used to interact with AWS services like S3 and EKS. It facilitated the seamless integration of AWS services into the application, enabling features like cloud storage and computing.
- **azure-sdk:** Similar to boto3, the Azure SDK was employed to manage and interact with Azure services. It allowed the application to integrate with Azure's cloud offerings, such as storage and virtual machines.
- **GCP-sdk:** This SDK was used to interact with Google Cloud Platform (GCP) services, enabling functionalities like cloud storage management and computing resources.
- **SQLAlchemy:** A SQL toolkit and Object-Relational Mapping (ORM) library for Python. SQLAlchemy was instrumental in managing the database schema and interactions in a more Pythonic and object-oriented manner.
- **Alembic:** A lightweight database migration tool for usage with SQLAlchemy. Alembic was used to manage and apply database schema changes over time, ensuring consistency and version control of the database structure.
- Database
 - PostgreSQL: An advanced, open-source relational database system. PostgreSQL was chosen for its robustness, scalability, and support for complex queries and data types, making it ideal for managing the application's data.
- DevOps
 - **Docker:** A platform that enabled the packaging of applications into containers. Docker ensured that the application could run consistently across different environments, reducing the "it works on my machine" problem.
 - **Kubernetes:** A container orchestration system used to automate the deployment, scaling, and management of containerized applications. Kubernetes was crucial in managing the clusters, ensuring high availability, and scaling the application according to demand.

- **Redis Pub/Sub:** Redis was used as a message broker for the notification system, enabling real-time push notifications. The Pub/Sub model ensured that notifications were delivered promptly and could be managed efficiently.

These technologies collectively empowered the development and deployment of a robust, scalable, and efficient full-stack application at Datawave Labs. Each tool and framework were selected for its specific strengths, contributing to the overall success of the project.



Chapter 2

Authentication and Authorization

2.1 User Sign-up and Login

2.2 Google Sign-in Integration

2. Authentication and Authorization

Authentication and authorization are critical components of any application, ensuring that users can securely access the system and that their actions are properly controlled based on their identity and permissions. In my role at Datawave Labs, I was responsible for implementing a comprehensive authentication and authorization system that included user sign-up, login, and verification processes, as well as secure session management using JWT (JSON Web Tokens) and role-based access control (RBAC). These mechanisms were essential in protecting the application from unauthorized access and ensuring that each user had the appropriate level of access to the system's resources.

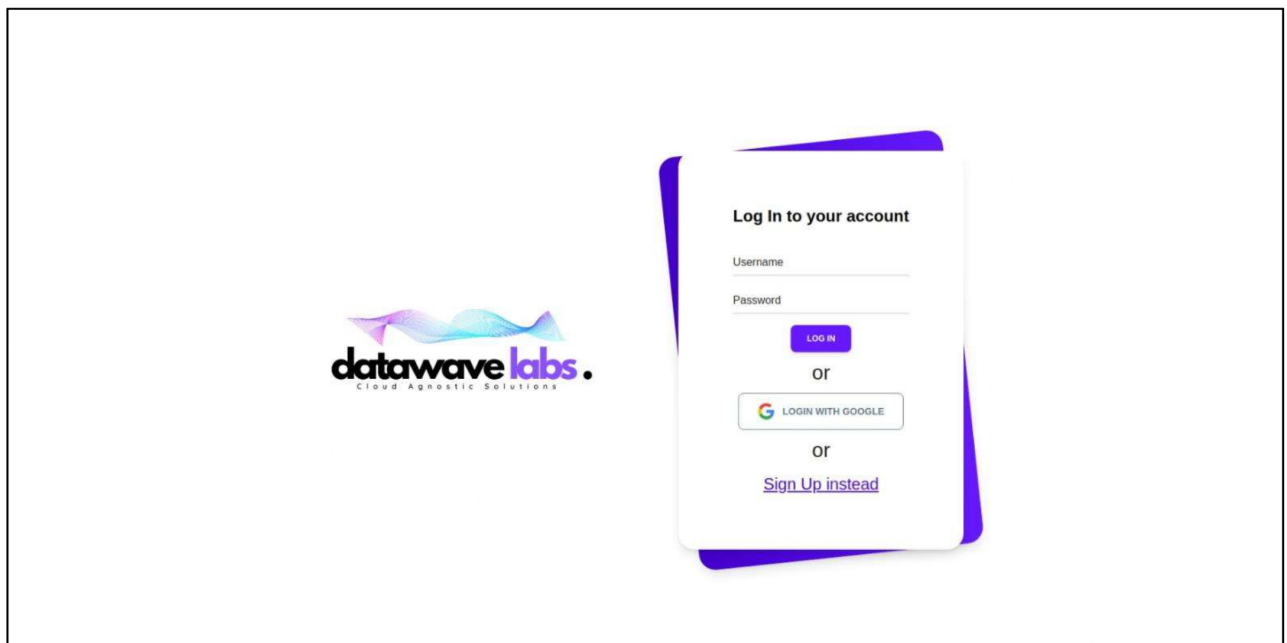


Fig 2.1: Login Page

2.1 User Sign-up and Login

The user sign-up and login process are a critical aspect of ensuring that an application is secure and accessible only to authorized users. At Datawave Labs, I focused on creating a secure and efficient authentication system that balanced usability with robust security measures. The authentication flow included email/password-based authentication as well as session management, each designed to provide a smooth and secure user experience.

2.1.1 Email/Password Authentication

Email and password authentication is one of the most common methods used to secure access to web applications. At Datawave Labs, I implemented a streamlined sign-up process that required users to provide a unique email address and a strong password. This approach was chosen for its simplicity and wide user acceptance, ensuring that new users could easily board themselves into the system.

The server architecture for this feature was designed using SQL Alchemy, where the user schema was carefully defined to ensure data integrity and security. Below is a simplified representation of the user schema that outlines how the essential fields were structured:

Column	Type
ID	Integer
Username	Character varying (30)
Email	Character varying (255)
Password	Character varying (255)
Google ID	Character varying (255)

Fig 2.2: User Schema

In this schema, each user is identified by a unique ID and has a unique username and email address. The password field stores the user's encrypted password, ensuring that even if the database were compromised, the actual passwords would not be exposed. An optional field for Google ID was included to allow for Google OAuth integration, providing flexibility in how users could log in.

To secure the passwords, I utilized Json Web Tokens, ensuring that passwords are not stored in plain text. During sign-up, the password provided by the user is hashed before being stored in the database. When a user attempts to log in, the password they enter is hashed again, and this hash is compared with the stored hash to verify the user's identity.

The email verification process was also integrated into the sign-up flow to enhance security. After providing their email and password, users receive a verification email containing a link to confirm their identity. Only after this verification step is the account fully activated, reducing the risk of fake or malicious accounts being created.

2.1.2 Session Management

Session management is a critical component in ensuring a secure and seamless user experience after a user has logged in. At Datawave Labs, I implemented a session management system that utilized a combination of cookies and JSON Web Tokens (JWT). This system allowed users to stay authenticated during their interactions with the application without the need to repeatedly enter their credentials.

JWTs were selected for their secure method of storing and transmitting user information between the client and server. Upon a successful login, a JWT was generated, encapsulating the user's unique identifier along with necessary claims, such as their role within the application. This token was then cryptographically signed using a secret key and sent to the client, where it was stored in a secure, HTTP-only cookie, safeguarding it from unauthorized access by client-side scripts.

With each subsequent request, the server could validate the JWT, ensuring that the user remained authenticated throughout their session. The token also included an expiration time, minimizing the risk of misuse if the token were ever compromised.

To strengthen security further, I implemented protections against common web vulnerabilities such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). XSS protection was achieved by sanitizing user inputs and ensuring that JWT cookies were HTTP-only, thereby inaccessible to malicious scripts. For CSRF protection, I implemented anti-CSRF tokens in forms and AJAX requests, ensuring that all requests were genuinely initiated by the authenticated user.

Session management also encompassed features such as session expiration and logout functionalities. When JWT expired, users were automatically logged out and required to log in again to continue using the application. This mechanism ensured that even in the event of token theft, the window of opportunity for misuse was extremely limited.

The token stored in the cookies was a string that looked like this:

served as a key component, granting the application temporary access to the user's Google profile information.

The login object, for instance, would look something like this:



```
1 {  
2   "access_token": "your_access_token_here",  
3   "token_type": "Bearer",  
4   "expires_in": 3599,  
5   "authuser": "0",  
6   "prompt": "none"  
7 }
```

Fig 2.4 Authentication Token Object

The next step involved verifying the access_token. To achieve this, I created a function that made a request to Google's OAuth2.0 API, which returned user information associated with the token. This function fetched the user's details such as their ID, email, and name.

Here's a simplified version of how the token verification and information retrieval was handled:



```
1 axios.get("https://www.googleapis.com/oauth2/v1/userinfo", {  
2   headers: {  
3     Authorization: `Bearer ${accessToken}`,  
4   },  
5 });
```

Fig 2.5: Google User Information

Upon receiving the user's information, the system checked whether the user already existed in the database by matching the email address. If the user was new, a new account would be created using a google ID, username and password. If the user was returning, the system allowed them to log in using the verified Google account.

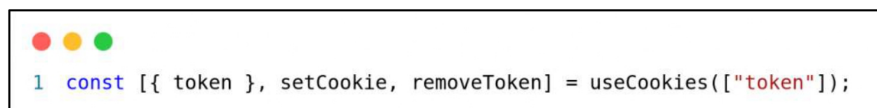
This implementation not only simplified the sign-up and login process for users but also ensured that sensitive information, such as passwords, was securely managed and protected from potential breaches. By leveraging Google OAuth2, I was able to enhance the authentication flow at Datawave Labs, providing a secure, efficient, and user-friendly way for users to access the platform.

2.2.2 Token Handling

Token handling is a critical aspect of maintaining secure and efficient user authentication in modern web applications. At Datawave Labs, I utilized the useCookies hook from the react-cookie library to manage JSON Web Tokens (JWT) effectively. This approach ensured that user sessions were both secure and easily managed, while also allowing for smooth integration with the server system.

Managing Tokens with react-cookie

In the client, token management was primarily handled using the useCookies hook. This hook provided a straightforward way to store, retrieve, and delete cookies, which were used to store the JWT. Here's how it was implemented:



```
1 const [{ token }, setCookie, removeToken] = useCookies(["token"]);
```

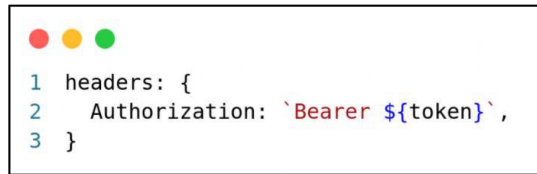
Fig 2.6: Using cookies with useCookies hook

In this implementation:

- **token:** This holds the current JWT if it exists.
- **setToken:** This function is used to store the JWT in a cookie upon successful login or token refresh.
- **removeToken:** This function is used to remove the JWT from the cookie, typically when the user logs out.

Bearer Authentication Token

For every request that required user authentication, the stored JWT was sent as a Bearer token in the authorization header. This ensured that only authenticated users could access certain endpoints or resources.



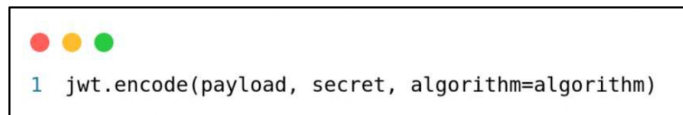
```
1 headers: {  
2   Authorization: `Bearer ${token}`,  
3 }
```

Fig 2.7: Sending headers from client

This Bearer token method is a widely accepted practice in modern web applications, as it provides a secure way to transmit the token, allowing the server to validate the user's identity and permissions for each request.

Token Handling on the Server

On the server, I used PyJWT to encode the JWT. The token was generated using the `jwt.encode` function, which securely encoded the user's information into a token that could be transmitted to the client.



```
1 jwt.encode(payload, secret, algorithm=algorithm)
```

Fig 2.8: Creation of Authentication token

- **payload:** This included the user's unique identifier and any other relevant claims, such as their role within the application.
- **secret:** A secret key used to sign the token, ensuring its integrity.
- **algorithm:** The algorithm used to encode the token, typically HS256.

This encoded token was then sent to the client, where it was stored in the cookie using `setToken`.

Logout and Token Removal

When the user chose to log out, the `removeToken` function was called to delete the JWT from the cookie, effectively ending the user's session.



```
1 removeCookie("token", { path: "/" });
```

Fig 2.9: Removing Cookies for logout

This simple yet effective approach ensured that once the user logged out, their token was removed, preventing any further unauthorized access.

By implementing these token handling practices, I ensured that the authentication system at Datawave Labs was robust, secure, and user-friendly, allowing for seamless user experiences across the application.



User Account Management

Chapter 3

3.1 Username Management

3.2 Password Management

3. User Account Management

User Account Management is crucial for maintaining secure and user-friendly access to the platform. Currently, at Datawave Labs, this section focuses on basic functionalities like username and password changes, enabling users to update their credentials safely.

Future enhancements will expand these features, adding capabilities like multi-factor authentication and profile customization, to further improve security and user experience. The following subsections will outline the existing username and password management processes, providing a foundation for future updates.

3.1 Username Management

The username management system ensures that each user has a unique and valid identifier. The username is a critical component of user profiles, serving as a distinct label for each individual. The system is designed to enforce rules that maintain the uniqueness and validity of usernames, providing a seamless user experience and preventing conflicts within the platform.

3.1 Unique Username Validation

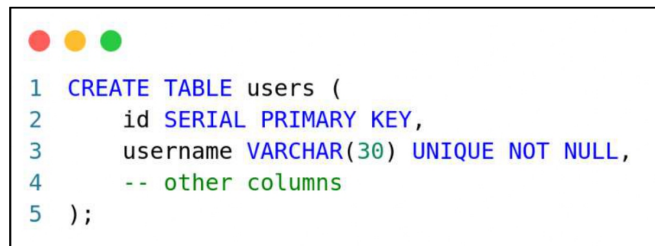
Unique username validation is a key feature of the user management system. It ensures that each username is distinct across the platform by enforcing the following rules:

- **Uniqueness Check:** When a user attempts to create or change their username, the system checks against existing usernames in the database to confirm that the new username is not already in use. This prevents duplicate usernames and ensures each user can be uniquely identified.
- **Validation Logic:** The validation process includes checking for the existence of the username in the user database. If the username is already taken, the system prompts the user to choose a different one. This process helps in maintaining the integrity and individuality of each user's profile.

By implementing these validation checks, the system guarantees that usernames remain unique and that no two users share the same username, enhancing user management and identification within the platform.

3.1.2 Change Username flow

In the current user management system, users can change their usernames through a simple and intuitive interface. The username serves as a unique identifier within the system, ensuring that no two users can share the same username. This uniqueness is enforced through sql schema validation processes that check for duplicate entries before committing any changes.

A code editor window with a white background and a black border. At the top left, there are three colored circles: red, yellow, and green. The code is as follows:

```
1 CREATE TABLE users (  
2     id SERIAL PRIMARY KEY,  
3     username VARCHAR(30) UNIQUE NOT NULL,  
4     -- other columns  
5 );
```

Fig 3.1: Database User Schema

When a user opts to change their username, the system prompts them to enter the new desired username. The interface provides immediate feedback if the username is already in use or if it does not meet the system's requirements. This ensures that users can quickly adjust without having to repeatedly submit the form.

To enhance user experience, the system also includes proper error handling and validation alerts. These alerts notify the user of any issues during the username change process, such as trying to select a username that is already taken. This approach minimizes the risk of user frustration and contributes to a smoother, more efficient account management process.

3.2 Password Management

Password management is a fundamental aspect of user account security and integrity. This system is designed to handle user passwords with measures to protect sensitive information and ensure that only authorized users can access their accounts.

3.2.1 Secure Password Reset

The password reset feature is crucial for maintaining account security and ensuring that users can regain access to their accounts if necessary. This process involves several key validations to ensure that password changes are handled securely and correctly.

Validation Points

- **Current Password Verification:** Before a new password can be set, the system requires that the current password be correctly entered. This step verifies the user's identity and ensures that the password change request is legitimate.
- **New Password Uniqueness:** The new password must be different from the previous one. This prevents users from reusing their old passwords and helps in maintaining a secure password history.
- **Minimum Length Requirement:** The new password must be at least 8 characters long. This length requirement helps to ensure that passwords are sufficiently complex and less susceptible to unauthorized access.

Error Handling

To enhance the user experience, the system includes proper error validation mechanisms. Alerts are displayed to inform users of any issues with their password reset request, such as incorrect current password entries, failure to meet the length requirement, or attempts to reuse the old password. These alerts guide users in correcting their input and completing the password reset process successfully.

The following screenshot shows the password change interface where users can input their current password, set a new password, and receive immediate feedback on their input.

Token Update

After a successful password reset, as well as after a username change, the system updates the authentication token stored in cookies. This ensures that the user's session reflects the most current authentication credentials, enhancing security and preventing potential access issues.



Chapter 4

Cloud Account Integration

2.1 AWS Integration

2.2 Azure Integration

2.3 GCP Integration

4. Cloud Account Integration

The Cloud Integrations feature of the app is designed to support a wide range of cloud services, enhancing the application's capability to manage and utilize various cloud-based resources. The app supports integration with several powerful cloud services and clusters, including:

- **VS Code IDE:** Provides a cloud-based development environment for coding and debugging.
- **JupyterHub:** Facilitates interactive computing and data analysis through Jupyter notebooks.
- **MLflow:** Manages the machine learning lifecycle, including experimentation, reproducibility, and deployment.
- **Airflow:** Orchestrates complex workflows and data pipelines with a robust scheduling system.
- **Superset:** Delivers advanced data visualization and business intelligence capabilities.

To leverage these cloud services, users must first add a cloud account within the app. This process involves linking their cloud accounts to the application, which enables seamless integration and management of the supported services. The cloud account setup allows users to connect and configure their chosen services, ensuring that the app can interact with them effectively and provide a unified experience for managing cloud-based resources.

4.1 AWS Integration

Integrating AWS with the app involves a straightforward process that ensures users can securely connect their AWS accounts and manage them effectively. This integration allows the app to interact with AWS services and resources, leveraging AWS's powerful cloud infrastructure.

4.1.1 Connecting AWS Account

To connect to an AWS account, users must provide their AWS credentials, which include the access key and secret key. These credentials are crucial for authenticating the connection and ensuring secure access to AWS services.

Steps for AWS Integration:

- **Add Cloud Account Form:** Users fill out a form in the app to connect their AWS account. The form consists of three main sections:
 - **Choose Platform:** Select from AWS, Azure, or GCP.
 - **Environment Mode:** Specify the environment, such as Prod, Dev, UAT, Sandbox, or Other.
 - **Credentials:** For AWS, users input their Access Key and Secret Key.
- **Server Validation:** In the server, the provided AWS credentials are validated using the boto3 library. The validation process involves creating an STS (Security Token Service) client and invoking the `get_caller_identity()` method to verify the credentials:

```
1 import boto3
2
3 def verify_aws_account(access_key, secret_key):
4     try:
5         boto3.client('sts',
6                       aws_access_key_id=access_key,
7                       aws_secret_access_key=secret_key
8                     ).get_caller_identity()
9         return True
10    except:
11        return False
```

Fig 4.1: Verifying AWS account

If the credentials are valid, the response confirms the identity, allowing the integration to proceed.

- **Database Entry:** Once validated, the AWS account details are added to the database.

This structured approach ensures that AWS accounts are securely integrated into the app, enabling users to manage and utilize AWS resources effectively.

4.2 Azure Integration

Integrating Azure with the app involves verifying Azure credentials to ensure a secure and valid connection. The process allows the app to interact with Azure services and resources, providing users with robust cloud management capabilities.

Connecting Azure Account

To integrate an Azure account, users need to provide specific credentials, including the Subscription ID, Tenant ID, Client ID, and Client Secret. These credentials are used to authenticate and establish a secure connection to Azure services.

Steps for Azure Integration:

- **Add Cloud Account Form:** Users submit their Azure credentials through the app's cloud account form. The form includes:
 - **Choose Platform:** Select Azure from the list of platforms.
 - **Environment Mode:** Specify the environment, such as Prod, Dev, UAT, Sandbox, or Other.
 - **Credentials:** Enter the Tenant ID, Client ID, and Client Secret.
- **Form Submission:** A POST request is sent to the /cloud endpoint with the provided credentials.
- **Server Validation:** The server validates the Azure credentials using the `azure.identity` and `azure.mgmt.resource` libraries. The `verify_azure_account` function performs the validation by attempting to list Azure subscriptions. If the credentials are valid, the function successfully lists subscriptions; otherwise, an error is raised:

```
1 from azure.core.exceptions import HttpResponseError
2 from azure.identity import ClientSecretCredential
3 from azure.mgmt.resource import SubscriptionClient
4
5 def verify_azure_account(tenant_id, client_id, client_secret):
6     try:
7         credential = ClientSecretCredential(
8             tenant_id=tenant_id, client_id=client_id,
9             client_secret=client_secret
10        )
11        subscription_client = SubscriptionClient(credential)
12        for subscription in subscription_client.subscriptions.list():
13            pass
14        return True
15    except HttpResponseError:
16        return False
```

Fig 4.2: Verifying Azure account

- **Database Entry:** Upon successful validation, the Azure account details are recorded in the database. The database schema for storing cloud account information is the same as for AWS.

This comprehensive approach ensures that Azure accounts are securely and correctly integrated into the app, enabling users to manage and utilize Azure resources effectively.

4.3 GCP Integration

Integrating Google Cloud Platform (GCP) with the app allows users to manage and interact with GCP services effectively. The integration process involves validating GCP credentials to ensure a secure and authorized connection.

Connecting GCP Account

To integrate a GCP account, users must provide a JSON file containing their service account credentials. This file includes various pieces of information required to authenticate and connect to GCP services.

Steps for GCP Integration:

- Add Cloud Account Form: Users submit their GCP credentials through the app's cloud account form. The form includes:
 - Choose Platform: Select GCP from the list of platforms.
 - Environment Mode: Specify the environment, such as Prod, Dev, UAT, Sandbox, or Other.
 - Credentials: Upload the JSON file containing the service account details.
- Form Submission: A POST request is sent to the /cloud endpoint with the provided JSON data.
- Server Validation: The server validates the GCP credentials using the google.cloud and google.oauth2 libraries. The verify_gcp_account function attempts to create a storage.Client instance with the provided credentials:

```
1 from google.cloud import storage
2 from google.oauth2 import service_account
3
4 def verify_gcp_account(json_data):
5     try:
6         credentials = service_account.Credentials.from_service_account_info(json_data)
7         storage.Client(credentials=credentials)
8         return True
9     except:
10        return False
```

Fig 4.3: Verifying GCP account

If the credentials are valid, the function successfully creates a client instance; otherwise, an exception is raised.

- Database Entry: Upon successful validation, the GCP account details are stored in the database. The schema for storing cloud account information remains consistent with other platforms.

This process ensures that GCP accounts are securely and effectively integrated into the app, allowing users to manage and utilize GCP resources seamlessly.



Chapter 5

Cluster Management

- 5.1 Cluster Creation Workflow
- 5.2 Authentication & Authorization
- 5.3 Real-Time Status Updates

5. Cluster Management

Cluster management in the application is a critical feature designed to provide users with the ability to manage and interact with various computational environments seamlessly. The creation and deletion of clusters were managed by another dedicated team, who handled these processes using Terraform. These clusters were then packaged into Docker images, which were essential for my part of the project.

My task focused on automating the workflow to run these Docker images as soon as a POST request was received at the /clusters endpoint. This process had to be triggered only after proper authentication and authorization checks were successfully completed to ensure the security and integrity of the system. Once the Docker container booted up, the system would send an Event Source response, which is crucial for managing live status updates of the cluster. This real-time feedback allowed users to monitor the state of their clusters as they were being created or deleted, ensuring transparency and immediate awareness of any issues or progress in the cluster management process.

This automated approach streamlined the interaction with clusters, allowing for efficient deployment and management without the need for manual intervention, aligning with the project's goals of enhancing user experience and operational efficiency.

5.1 Cluster Creation Workflow

The cluster creation workflow is a pivotal aspect of our cloud management system. It automates the deployment of infrastructure within a cloud environment, ensuring that clusters are provisioned with minimal manual intervention. While the current implementation is tailored to AWS, the system is designed with future scalability in mind, allowing for seamless adaptation to other cloud providers like Azure and GCP.

Client Workflow

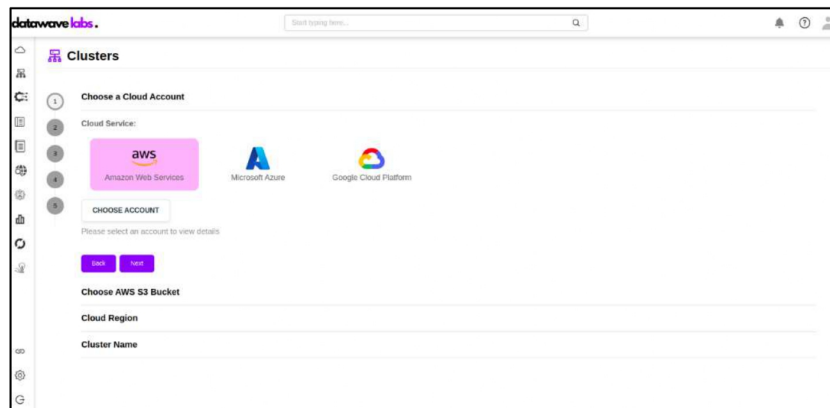


Fig 5.1: Add Cluster Page

The process begins in the client, where the user is prompted to enter several key details required for cluster creation. These inputs include selecting a cloud account, specifying an S3 bucket, choosing a region, and defining a cluster name. While these fields are essential for initiating the process, it's important to note that certain configurations are currently hardcoded in Terraform scripts. These configurations, such as VPC CIDR blocks and Kubernetes version, will eventually be made editable through the client, offering users greater flexibility in customizing their clusters.

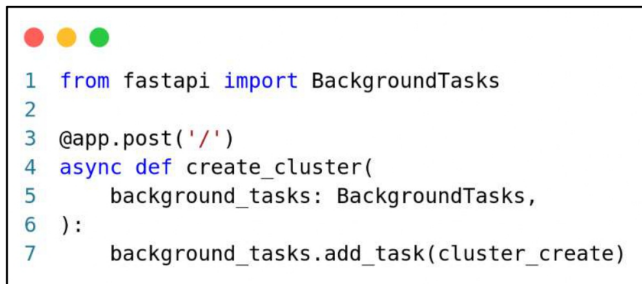
For now, the client gathers the necessary details from the user and compiles them into a structured request. This data is then sent via a POST request to the /cluster endpoint of the server API. The request body typically includes the following parameters:

The parameters that are sent from the client are crucial for the server to initiate the cluster creation process. The account ID identifies the user's cloud account, while provider specifies the cloud platform (currently limited to AWS). The cluster_name_prefix and cluster_name_suffix help in generating a unique name for the cluster, ensuring no conflicts with existing resources. region dictates where the cluster will be deployed, and bucket_name provides the S3 bucket necessary for storing Terraform state files.

Server Process

Upon receiving the request, the server begins the cluster creation process. Given that this is a time-consuming task, it is essential to handle it asynchronously. To achieve this, we use the Background Tasks

feature from FastAPI, which allows the cluster creation to run in the background while the API immediately responds to the user. This ensures that the user doesn't experience delays or timeouts while the cluster is being provisioned.

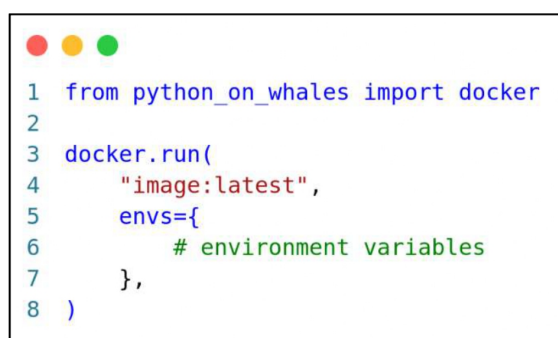


```
1 from fastapi import BackgroundTasks
2
3 @app.post('/')
4 async def create_cluster(
5     background_tasks: BackgroundTasks,
6 ):
7     background_tasks.add_task(cluster_create)
```

Fig 5.2: Initializing a Background Process

The `cluster_create` function is responsible for orchestrating the cluster creation process. At the heart of this function is the Python SDK `python_on_whales`, which is used to interact with Docker containers. This SDK provides a convenient and powerful interface for managing Docker operations within Python applications.

To initiate the cluster creation, we use the `docker.run` command from `python_on_whales`. This command spins up a Docker container that runs the necessary Terraform scripts to provision the cluster. The Docker container is started with several environment variables that are passed from the client, ensuring that the configuration is specific to the user's requirements.



```
1 from python_on_whales import docker
2
3 docker.run(
4     "image:latest",
5     envs={
6         # environment variables
7     },
8 )
```

Fig 5.3: Using python sdk to Run Docker Container

In this command, the Docker container is launched using the docker image, and it is named dynamically based on the cluster's prefix and suffix. The environment variables passed into the container include the

AWS access key, secret key, region, and S3 bucket name. These credentials and configurations are crucial for the Terraform scripts within the container to successfully apply the infrastructure changes.

Notably, several parameters in the environment variables are currently hardcoded. These include the VPC CIDR block, public and private CIDR blocks, and the Kubernetes version. In future iterations, these parameters will be customizable through the client, offering users greater control over their cluster configurations.

Future Enhancements

While the current implementation is limited to AWS, the architecture is designed with future expansion in mind. The same workflow can be easily adapted to support other cloud providers like Azure and GCP. The modular design of the system allows for the introduction of additional cloud platforms with minimal changes to the existing codebase.

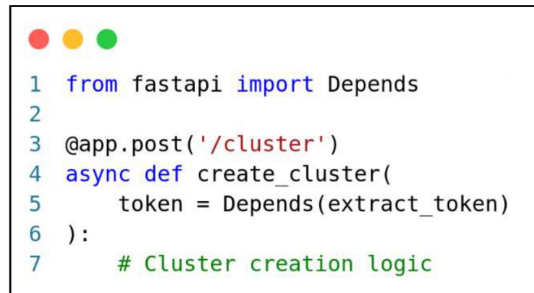
5.2 Authentication & Authorization

The security of cluster management is paramount, and our system employs robust mechanisms to ensure that only authorized users can initiate cluster creation. This involves both authentication and authorization processes.

Authentication

Authentication is handled using Bearer tokens. When a client makes a request to create a cluster, a Bearer token is included in the request header. This token is essential for verifying the identity of the user making the request. If the token is missing or invalid, the server responds with a 401 Unauthorized status code, indicating that the request cannot be processed due to authentication failure.

In the server, the authentication process is facilitated by the `extract_token` function, which is invoked in every request that requires authentication. Here is a simplified example of how this is implemented:

A code editor window with a white background and a black border. At the top left, there are three colored circles: red, yellow, and green. The code is written in a monospaced font with syntax highlighting: line numbers 1-7 are on the left; 'from fastapi import Depends' is on line 1; '@app.post('/cluster')' is on line 3; 'async def create_cluster(' is on line 4; 'token = Depends(extract_token)' is on line 5; '):' is on line 6; and '# Cluster creation logic' is on line 7.

```
1 from fastapi import Depends
2
3 @app.post('/cluster')
4 async def create_cluster(
5     token = Depends(extract_token)
6 ):
7     # Cluster creation logic
```

Fig 5.4: Using Tokens from Auth Headers

The `extract_token` function parses and validates the Bearer token from the request. If the token is valid, it decodes it to retrieve the user's information, which is then used to authorize the request. If the token is invalid or absent, the server will not process the request, ensuring that unauthorized users cannot initiate cluster creation.

Authorization

In our system, we have planned to implement Role-Based Access Control (RBAC) to manage permissions for cluster creation. Under RBAC, different roles are assigned specific permissions, and only users with the appropriate role (such as an admin) will be authorized to create clusters.

Currently, this RBAC system is in the planning stages and has not yet been implemented. Once integrated, it will ensure that only users with the admin role have the capability to create clusters, while other users will be restricted from accessing this functionality.

In summary, the authentication and authorization mechanisms in place ensure that only validated and authorized users can initiate cluster creation. The Bearer token mechanism provides a secure way to verify user identity, while the planned implementation of RBAC will further enhance security by enforcing role-based permissions.

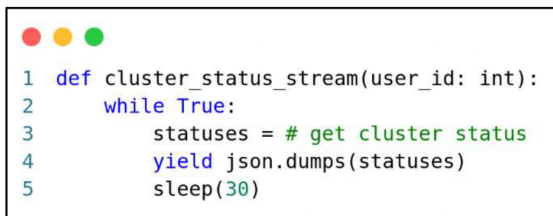
5.3 Real-Time Status Updates

To keep users informed about the progress of cluster creation, real-time status updates are implemented using server-sent events (SSE). This setup ensures that the client receives continuous updates on cluster statuses as they change.

Server Implementation

In the server, the EventSourceResponse from the sse_starlette.sse library is utilized to provide real-time updates. When a GET request is made to the /cluster/status endpoint, the server streams updates to the client.

The core of this implementation is the cluster_status_stream function:




```
1 def cluster_status_stream(user_id: int):
2     while True:
3         statuses = # get cluster status
4         yield json.dumps(statuses)
5         sleep(30)
```

Fig 5.5: Implementation of SSE

Client Implementation

On the client side, the EventSourcePolyfill library handles the SSE connection and manages authentication headers. This ensures that the Bearer token is included with each request to secure the data:




```
1 import { EventSourcePolyfill } from "event-source-polyfill";
2
3 const eventSource = new EventSourcePolyfill("/cluster/status", {
4     headers: {
5         Authorization: `Bearer ${token}`,
6     },
7 });
8
```

Fig 5.6: Client Side handling SSE with EventSourcePolyfill

The client-side code listens for incoming messages and updates the UI:


- **Handle Messages:** When a message is received, the data is parsed and used to update the status of clusters in the application state:



```
1 eventSource.onmessage = (event) => {
2     const res = JSON.parse(event.data);
3     // Updating the state
4 };
5
```

Fig 5.7: Extracting messages from event source

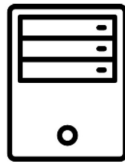
- **Handle Errors:** Logs errors if the event source fails:



```
1 eventSource.onerror = (err) => {
2     console.log("EventSource failed:", err);
3 };
```

Fig 5.8: Handling Errors

This system provides real-time updates on cluster creation status, ensuring that users receive timely information and can monitor the progress of their clusters effectively.



Chapter 6

Server Development

6.1 Database Development with Psycopg2

6.2 API Testing with Postman

6. Server Development

This section focuses on the server-side development of the application, detailing the server infrastructure and processes. It covers the setup and management of the database with psycopg2 and API testing to ensure that the server-side components function correctly and efficiently. This comprehensive approach ensures a robust and reliable server system, crucial for the overall performance and stability of the application.

6.1 Database Development with Psycopg2

Initially, for managing the database, I utilized the psycopg2 library in Python. This approach involved directly connecting to the PostgreSQL database and executing SQL commands to manage the schema and data. For instance, I used psycopg2 to establish a connection using a database URL and then executed SQL queries to create and manage tables. An example of this process included:



```
1 conn = psycopg2.connect(db_url)
2 cur = conn.cursor()
3 cur.execute("""
4     CREATE TABLE IF NOT EXISTS ..... );
5 """)
```

Fig 6.1: Creating Database Connection with Psycopg2

In this setup, database connections and operations were managed manually, which included handling schema migrations and data queries directly through SQL statements.

However, as the project evolved, I was advised to transition using SQLAlchemy ORM for database management as direct SQL queries can be cumbersome and error-prone for complex relationships and migrations. SQLAlchemy provides a higher-level abstraction, allowing for more flexible and manageable database interactions through its Object-Relational Mapping (ORM) capabilities. This change was aimed at streamlining database operations and improving code maintainability by leveraging SQLAlchemy's features for handling schema migrations, data manipulation, and queries in a more Pythonic way.

6.2 API Testing with Postman

During my internship, I explored more about Postman to facilitate API testing. This tool allowed me to efficiently test various endpoints and manage requests through organized collections and folders.

Postman's intuitive interface and powerful features, such as environment variables and automated tests, streamlined the process of verifying API functionality and performance. The ability to save and share API requests in Postman collections greatly enhanced my workflow, making it easier to collaborate with team members and ensure that the APIs met the required specifications.

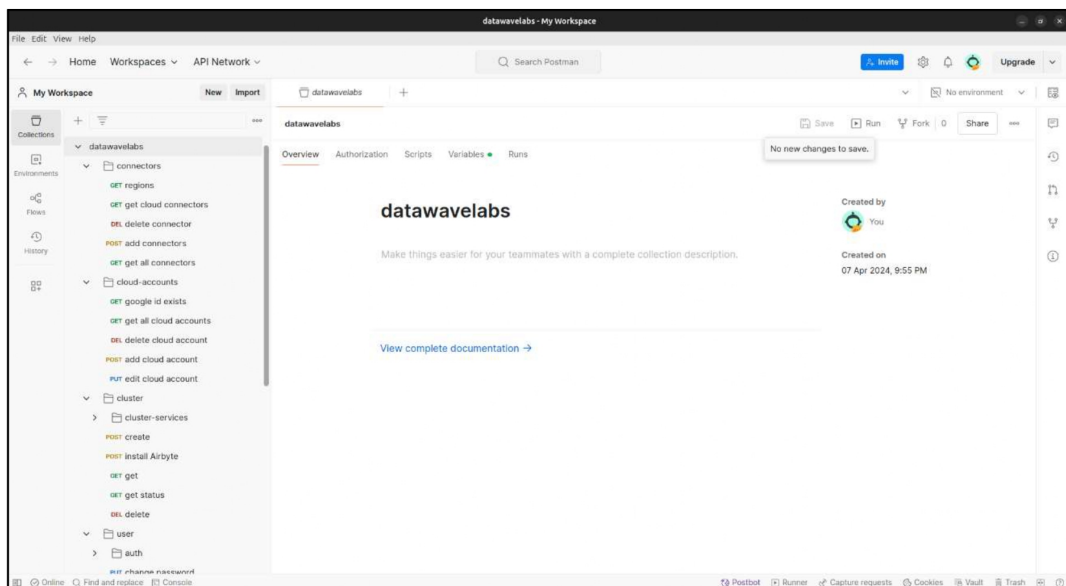


Fig 6.2: Postman API Collection



Notification System

Chapter 7

7.1 Overview of Notification Mechanism

7.2 Integration with Services

7. Notification System

Effective communication and timely alerts are crucial in managing cloud infrastructure and clusters. The notification system implemented serves as a backbone to ensure that all relevant stakeholders are promptly informed of critical events, status changes, and system updates. By integrating Redis Pub/Sub for real-time message delivery and developing robust notification routes, the system enables seamless information flow. This section delves into the various components of the notification system, focusing on how they contribute to maintaining operational efficiency and responsiveness across the platform.

7.1 Overview of Notification Mechanism

The notification system is designed to provide real-time updates and alerts to users based on various events within the application. It plays a crucial role in keeping users informed about the status of different processes, particularly in a complex environment where multiple services interact with each other. To achieve this, the system leverages Redis Pub/Sub, a powerful messaging framework that facilitates real-time communication. Redis Pub/Sub is used to broadcast messages to various subscribers instantly. This ensures that notifications are delivered promptly, without the need for constant polling by the client. When an event occurs, such as a change in the status of a cluster or an update in user account management, a message is published to a specific Redis channel. Subscribers listening to this channel immediately receive the message and can act accordingly, such as displaying an alert to the user or updating the status in the user interface. This approach not only enhances the responsiveness of the application but also improves user experience by providing timely and relevant information without unnecessary delays.

7.2 Integration with Services

In modern web applications, real-time notifications are essential for enhancing user engagement and delivering timely updates. To achieve this, Redis serves as an efficient message broker, enabling seamless real-time communication between different parts of the system. In this section, we will delve into how Redis is integrated into a notification system using Server-Sent Events (SSE). This setup ensures that notifications are published, transmitted, and received in real time, allowing users to stay informed about relevant events as they occur.

Redis Client Initialization:

The first step in integrating Redis with the notification system is to establish a connection between the application and the Redis server. This is accomplished by initializing a Redis client. In a Python-based system, this can be done using the `aioredis` library, which supports asynchronous communication. Asynchronous interactions are crucial in high-performance applications because they prevent the system from being blocked while waiting for responses from Redis.

Once the Redis client is initialized, it can be used across the application to interact with the Redis server asynchronously. This setup ensures non-blocking communication, which is vital for maintaining the responsiveness of the application, especially when handling a large volume of notifications.

Publishing Notifications:

Publishing notifications to users involves sending messages to a specific Redis channel. The Redis publish-subscribe (pub/sub) model is ideal for this scenario. In this model, a publisher sends messages to a channel, and any subscribers to that channel receive those messages in real-time. This approach allows for decoupled communication between the sender and the receiver, making the system more scalable and easier to maintain.

Listening for Notifications:

On the client side, the system needs to listen for incoming notifications. This is done by subscribing to the Redis channel dedicated to notifications. The subscriber continuously waits for messages on this channel and processes them as they arrive.

The listener function operates in an infinite loop, continuously waiting for new messages. This ensures that notifications are received and processed as soon as they are published, providing users with timely updates.



Chapter 8

Conclusion

8.1 Summary of Work Done

8.2 Lessons Learned

8.3 Future Work and Recommendations

8. Conclusion

The conclusion of this report encapsulates the journey undertaken throughout this internship, reflecting on the key accomplishments, challenges faced, and lessons learned. This section provides a comprehensive summary of the work done, highlighting the skills and knowledge gained. It also discusses the obstacles encountered and the strategies employed to overcome them. Finally, it offers recommendations for future work, emphasizing areas for improvement and potential enhancements to the project.

8.1 Summary of Work Done

During my internship at Datawave Labs, I was involved in the development and enhancement of various components of a full-stack application. My primary responsibilities included integrating cloud services, managing user authentication and authorization, developing server functionalities, and implementing a real-time notification system. I began by setting up the foundation for user account management, focusing on implementing robust sign-up and login mechanisms, as well as integrating Google Sign-In using OAuth2. This was followed by the development of a secure token-based authentication system, ensuring that only authorized users could access the application's core features. In the realm of cloud integration, I worked on connecting the application to AWS, Azure, and GCP services, allowing users to manage their cloud resources directly from the platform. This also involved creating an automated workflow for cluster management, where Docker images were executed in response to API calls, and real-time status updates were provided to the users. On the server, I initially utilized Psycpg2 for database management, later transitioning to SQLAlchemy ORM for more efficient handling of database operations. I also extensively used Postman for API testing, which streamlined the process of debugging and validating the server services. Additionally, I contributed to the development of a notification system using Redis Pub/Sub, enabling real-time communication within the application. This system was integral in sending alerts and updates to users about the status of their cloud resources and other critical events. Overall, the work done during this internship not only contributed to the development of a robust and scalable application but also provided me with invaluable experience in full-stack development, cloud integration, and server management.

8.2 Lesson Learned

Throughout my internship at Datawave Labs, I gained significant insights into both technical and professional aspects of software development. One of the most valuable lessons was the importance of balancing between writing clean, maintainable code and delivering features within tight deadlines. This experience highlighted the need for effective time management and the ability to prioritize tasks based on their impact and urgency.

I also learned the value of collaboration and communication in a development team. Regular standup meetings and ongoing discussions with my colleagues and mentors taught me how crucial it is to keep everyone aligned on project goals and challenges. This collaborative environment also provided me with exposure to different perspectives and problem-solving approaches, enhancing my ability to think critically and adapt to new situations. Working extensively with cloud technologies and integrating them into the application expanded my understanding of cloud infrastructure and its role in modern software solutions. I deepened my knowledge of AWS, Azure, and GCP, learning how to implement scalable and secure cloud-based systems. Another key lesson was the importance of thorough testing and validation. Using tools like Postman, I realized how crucial it is to test APIs rigorously to ensure they perform as expected under various conditions. This not only improved the reliability of the application but also helped in identifying and resolving issues early in the development process.

Lastly, the internship underscored the significance of continuous learning. Whether it was transitioning from Psycopg2 to SQLAlchemy ORM or exploring new libraries and frameworks, the experience reinforced the idea that staying updated with emerging technologies is essential for long-term growth in the tech industry.

8.3 Challenges Faced

During my internship at Datawave Labs, I encountered several challenges that tested my adaptability and problem-solving skills. One of the primary challenges was time management. Balancing multiple tasks, meeting project deadlines, and simultaneously managing my academic responsibilities required careful planning and prioritization. This experience taught me the importance of setting realistic goals and efficiently allocating time to ensure that all aspects of the project were adequately addressed. Another significant challenge was the need to quickly learn and adapt to new technologies and tools. For example, transitioning from using Psycopg2 to SQLAlchemy ORM for database management required me to grasp

the new framework's concepts rapidly while still delivering on my assigned tasks. Similarly, working with advanced cloud integration techniques, such as setting up clusters and managing authentication, pushed me to learn and implement these technologies swiftly, often without prior experience. Working with technologies and frameworks that I wasn't initially confident with, such as Docker for containerization and FastAPI for building server services, also posed a challenge. These technologies were critical to the success of the project and mastering them on the go required persistent effort and a willingness to step out of my comfort zone. Collaboration and communication also presented challenges, particularly in a remote work environment. Coordinating with team members across different time zones, ensuring clear communication of ideas, and staying aligned with the project's goals required me to develop stronger communication skills and become more proactive in seeking and providing feedback.

Lastly, ensuring the reliability and security of the systems I developed was an ongoing challenge. Implementing robust authentication and authorization mechanisms, managing cloud services securely, and ensuring the scalability of the application demanded meticulous attention to detail and a deep understanding of best practices in software development. These challenges not only enhanced my technical expertise but also reinforced the importance of diligence and thoroughness in every aspect of development.



References

9. References

1. FastAPI Documentation. Retrieved from <https://fastapi.tiangolo.com/>.
2. psycpg2 Documentation. Retrieved from <https://www.psycpg.org/docs/>.
3. SQLAlchemy Documentation. Retrieved from <https://docs.sqlalchemy.org/>.
4. Docker SDK for Python Documentation. Retrieved from <https://docker-py.readthedocs.io/en/stable/>.
5. Terraform Documentation. Retrieved from <https://www.terraform.io/docs/>.
6. Google OAuth 2.0 Documentation. Retrieved from <https://developers.google.com/identity/protocols/oauth2>.
7. Azure SDK for Python Documentation. Retrieved from <https://learn.microsoft.com/en-us/azure/developer/python/sdk/>.
8. Google Cloud SDK Documentation. Retrieved from <https://cloud.google.com/sdk/docs>.
9. SSE Starlette Documentation. Retrieved from <https://sse-starlette.readthedocs.io/en/latest/>.
10. Event Source Polyfill Documentation. Retrieved from <https://github.com/Yaffle/EventSource>.
11. Docker Documentation. Retrieved from <https://docs.docker.com/>.
12. Kubernetes Documentation. Retrieved from <https://kubernetes.io/docs/home/>.
13. Redis Pub/Sub Documentation. Retrieved from <https://redis.io/docs/manual/pubsub/>.
14. Postman Documentation. Retrieved from <https://learning.postman.com/docs/getting-started/introduction/>.
15. Python Dotenv Documentation. Retrieved from <https://saurabhdaware.github.io/psycpg2-python-dotenv/>.

These references provide the foundational knowledge and tools used throughout the project, contributing significantly to the successful completion of the work.