**Stocker – A Cloud based stock trading application using Flask and AWS**

**Project Description:**

Stocker is a next-generation stock trading platform designed to cater to the needs of modern digital finance. It allows users to execute real-time stock transactions, manage investment portfolios, and access insightful market data in a secure and responsive setup. The backend is developed using Flask, ensuring high flexibility and performance. Hosting on AWS EC2 provides dynamic scalability, while Amazon DynamoDB offers a serverless, fast, and reliable data storage solution. The frontend is crafted with modern CSS, delivering a clean, interactive, and user-friendly experience that ensures smooth navigation and efficient trading.

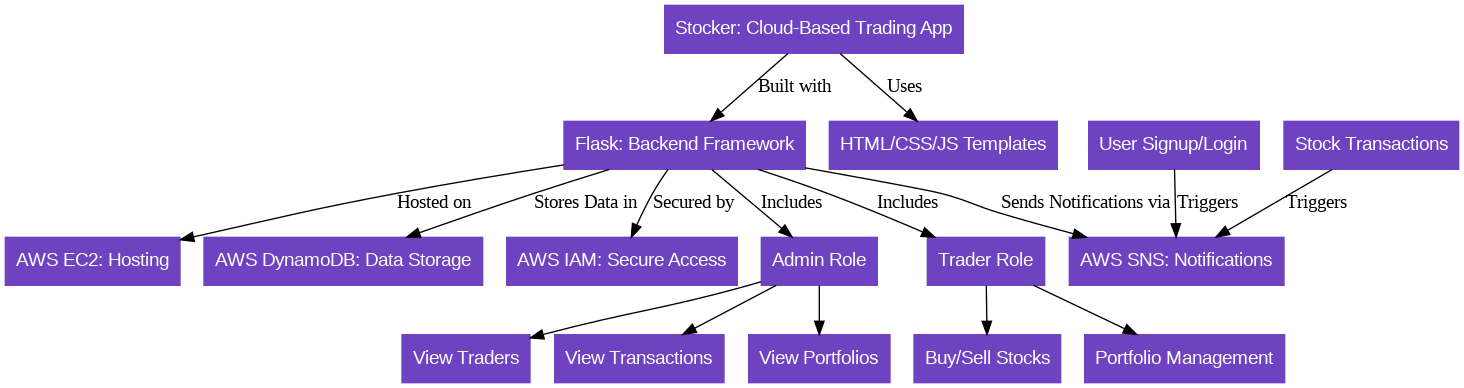
**Scenario 1: Secure User Authentication**

Stocker prioritizes the security of its users by implementing robust authentication mechanisms to safeguard trading accounts. The platform uses encrypted password storage, secure session management, and role-based access controls to ensure that user data remains protected at all times. By focusing on advanced security practices, Stocker provides users with confidence and peace of mind when managing their financial assets online.

**Scenario 2: Real-Time Trade Execution**  
In the fast-moving world of stock trading, timing is everything. Stocker is engineered to deliver real-time trade execution, allowing users to capitalize on market opportunities the moment they arise. The platform's backend is optimized for low-latency performance, ensuring that buy and sell orders are processed almost instantly. By leveraging cloud infrastructure and efficient database operations, Stocker minimizes delays and ensures that users can respond swiftly to market fluctuations, enhancing their trading strategies and maximizing potential profits.  
**Scenario 3: Portfolio Tracking**  
Effective portfolio management is key to successful investing, and Stocker provides an intuitive portfolio tracking to make this process effortless. Users can easily monitor their holdings and analyze within a user-friendly dashboard. The system, enables investors to make informed decisions based on current market conditions. With Stocker's portfolio tracking feature, users gain valuable insights into their investment health without the complexity typically associated with financial analysis.

**Architecture**

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement.

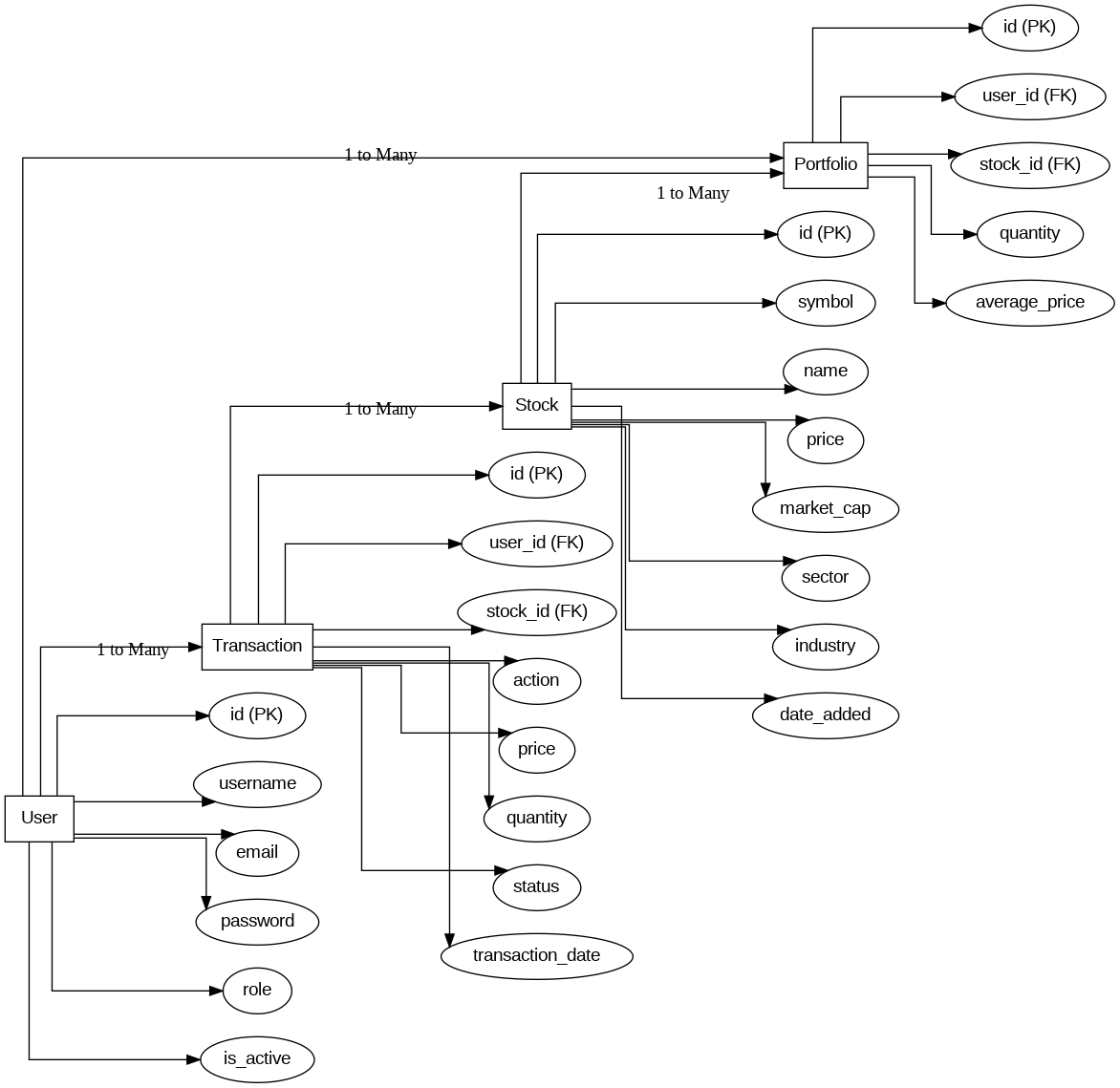


**Pre-requisites**

* AWS Account Setup:  
  <https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
* AWS IAM (Identity and Access Management):  
  <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
* AWS EC2 (Elastic Compute Cloud):  
  <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
* AWS DynamoDB:   
  [https://docs.aws.amazon.com/amazondynamodb/Introduction.html](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* Amazon SNS:   
  <https://docs.aws.amazon.com/sns/latest/dg/welcome.htm>l
* Git Documentation:   
  <https://git-scm.com/doc>
* VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)  
  <https://code.visualstudio.com/download>

**Entity Relationship (ER) Diagram**

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization, data integrity, and simplified query design.



**Project Workflow**

**Milestone 1. Backend Development and Application Setup**

* Develop the Backend Using Flask.
* Integrate AWS Services Using boto3.

**Milestone 2. AWS Account Setup and Login**

* Set up an AWS account if not already done.
* Log in to the AWS Management Console

**Milestone 3. DynamoDB Database Creation and Setup**

* Create a DynamoDB Table.
* Configure Attributes for User Data and Book Requests.

**Milestone 4. SNS Notification Setup**

* Create SNS topics for book request notifications.
* Subscribe users and library staff to SNS email notifications.

**Milestone 5. IAM Role Setup**

* Create IAM Role
* Attach  Policies

**Milestone 6. EC2 Instance Setup**

* Launch an EC2 instance to host the Flask application.
* Configure security groups for HTTP, and SSH access.

**Milestone 7. Deployment on EC2**

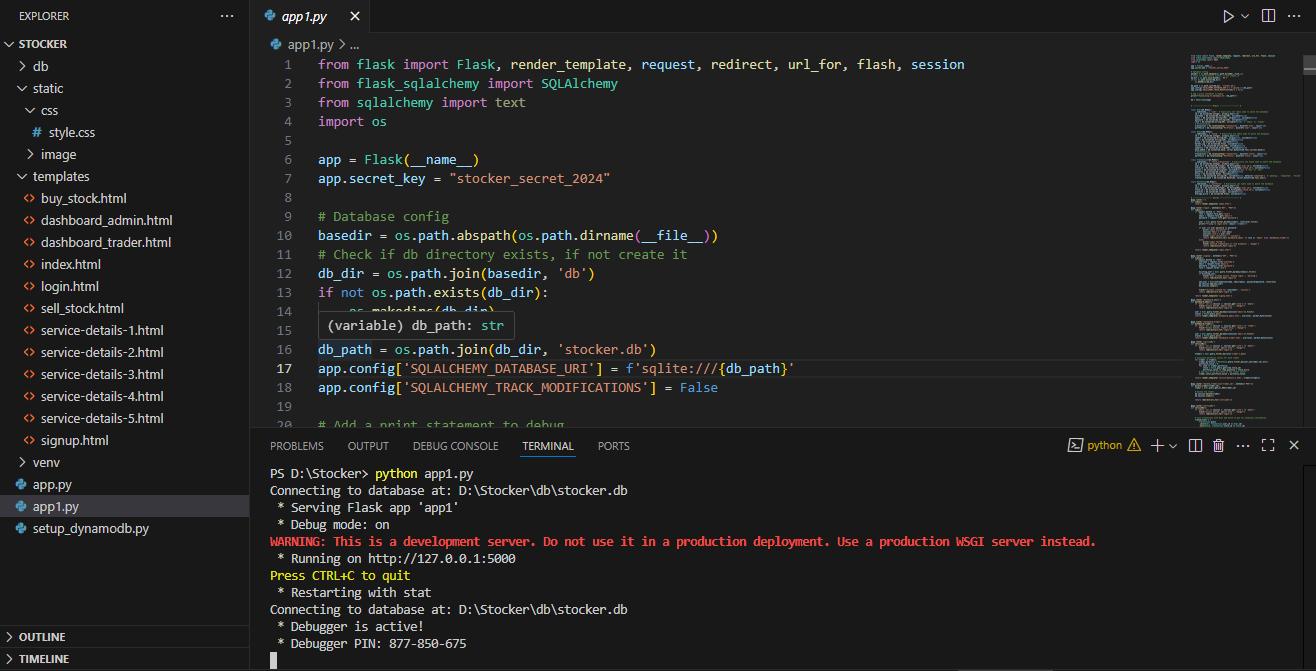
* Upload Flask Files
* Run the Flask App

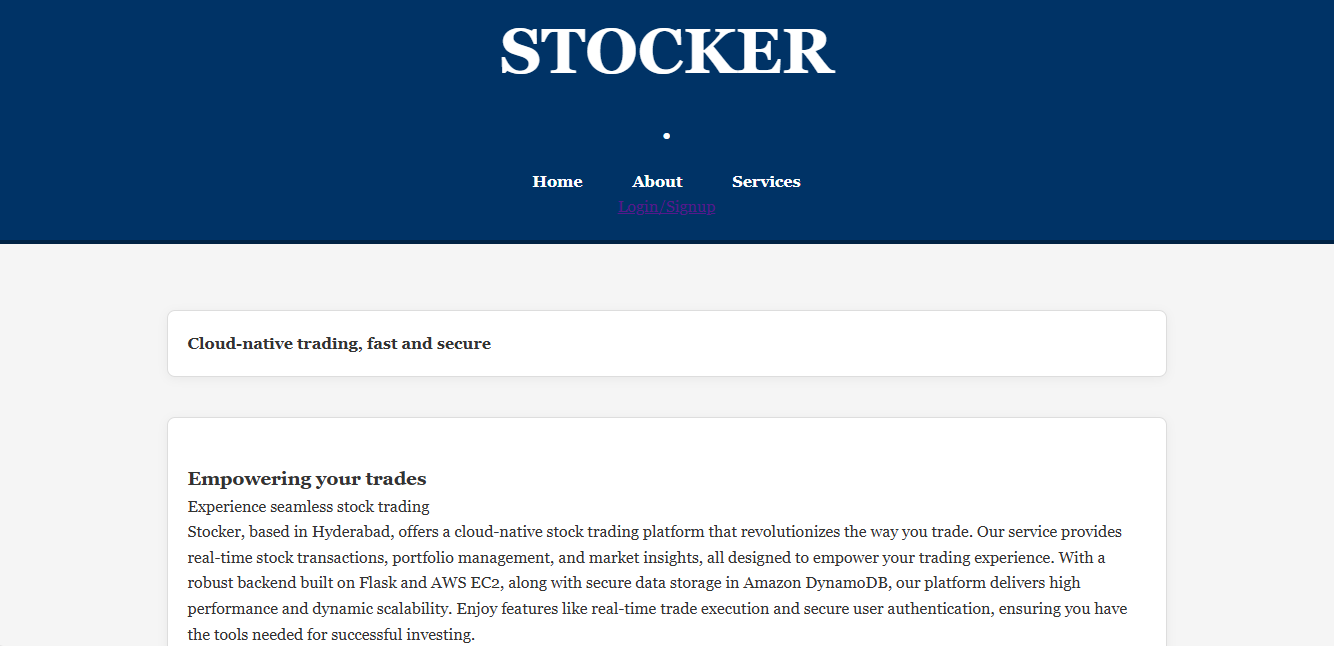
**Milestone 8. Testing and Deployment**

* Conduct functional testing to verify user signup, login, buy/sell stocks and notifications.

**Milestone 1 : Web Application Development and Setup**

Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access.





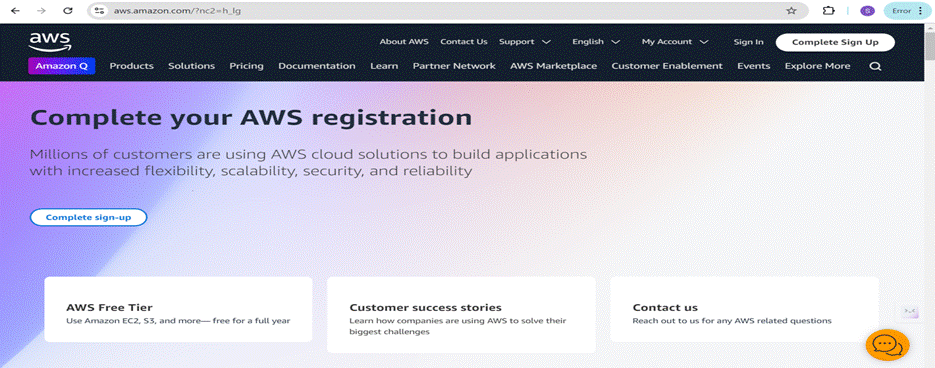
**Milestone 2 : AWS Account Setup**

Once your website is locally deployed and fully functional, we need to create a AWS ACCOUNT

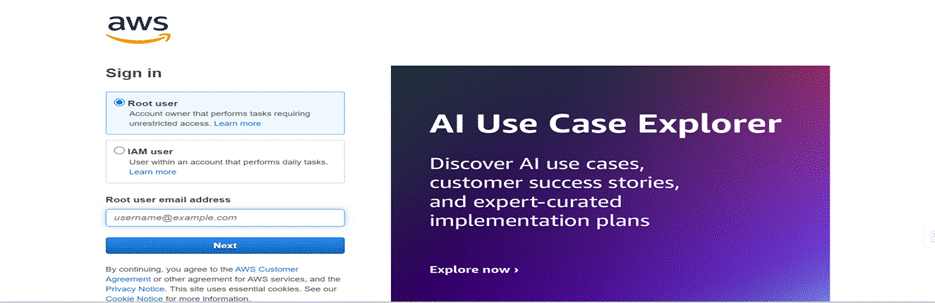
**AWS Account Setup and Login**

**This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.**

* Go to the AWS website (<https://aws.amazon.com/>).
* Click on the "Create an AWS Account" button.
* Follow the prompts to enter your email address and choose a password.
* Provide the required account information, including your name, address, and phone number.
* Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
* Complete the identity verification process.
* Choose a support plan (the basic plan is free and sufficient for starting).
* Once verified, you can sign in to your new AWS accounts.



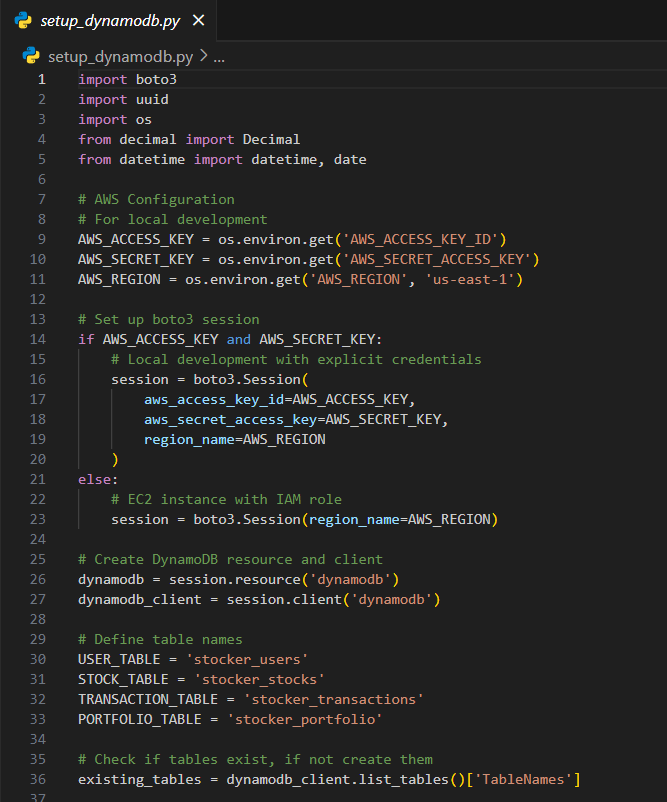
* Log in to the AWS Management Console
* After setting up your account, log in to the[AWS Management Console](https://aws.amazon.com/console/)



**Milestone 3 : DynamoDB Database Creation and Setup**

Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

* Create DynamnoDB tables and Insert sample data using setupdynamodb.py





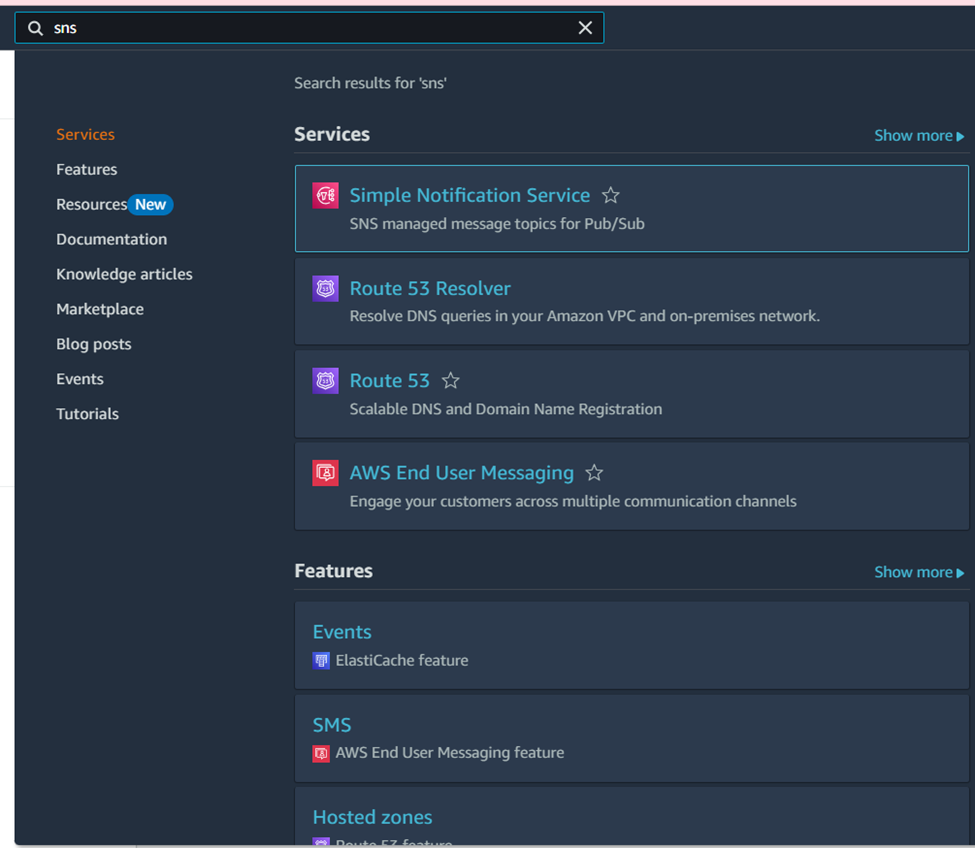


**Milestone 4 : SNS Notification Setup**

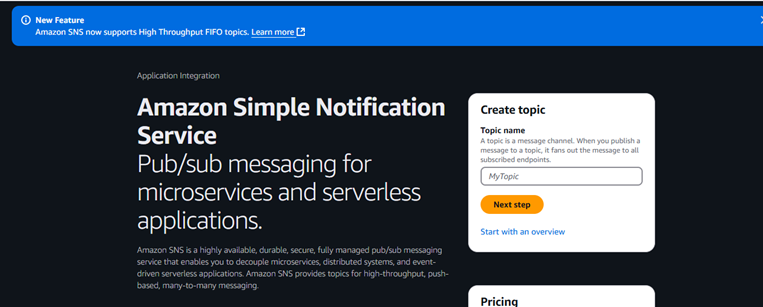
Amazon SNS is a fully managed messaging service that enables real-time notifications through channels like SMS, email, or app endpoints. You create topics, configure subscriptions, and integrate SNS into your app to send notifications based on specific events.

**SNS topics for email notifications**

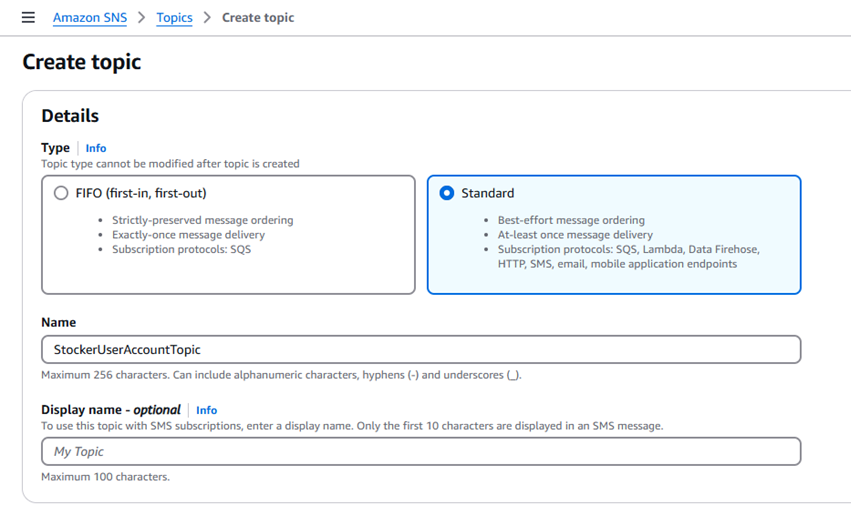
* In the AWS Console, search for SNS and navigate to the SNS Dashboard.

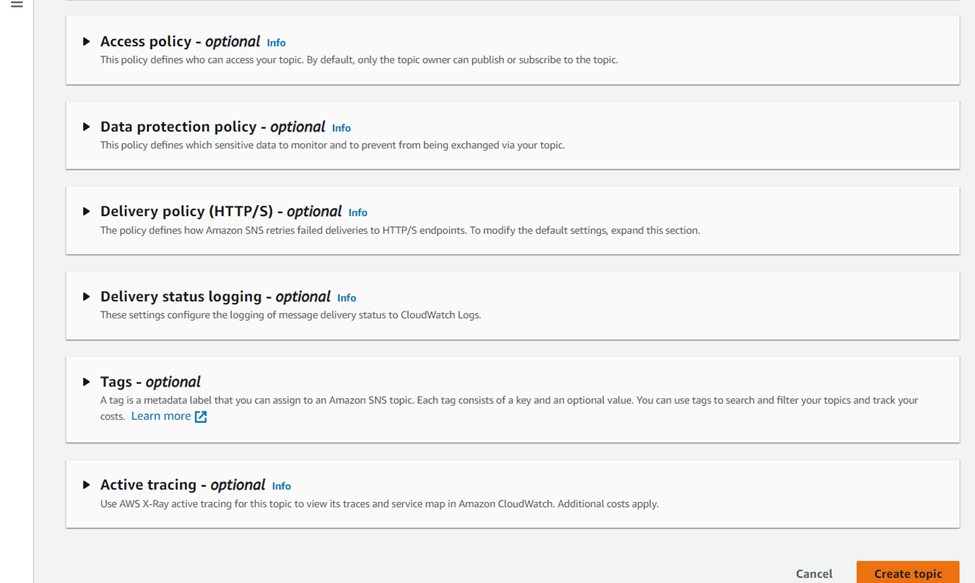


* click next step

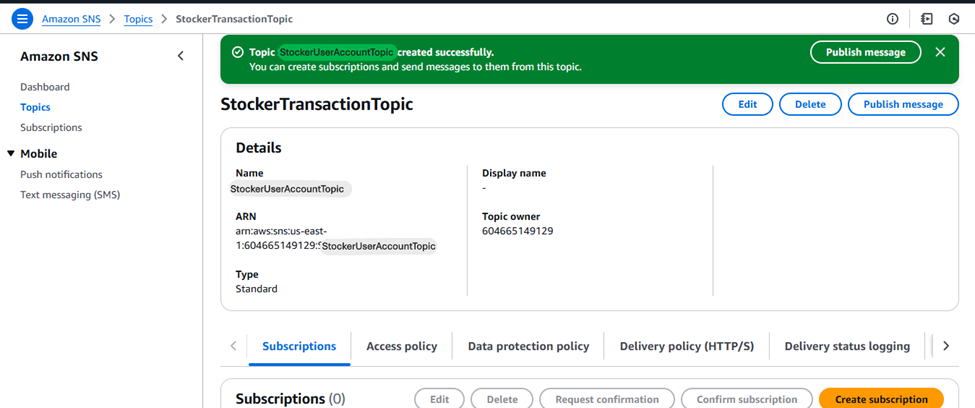


in Create Topic  Name  type ‘StockerUserAccountTopic’ and Choose Standard type for general notification use cases and Click on Create Topic.



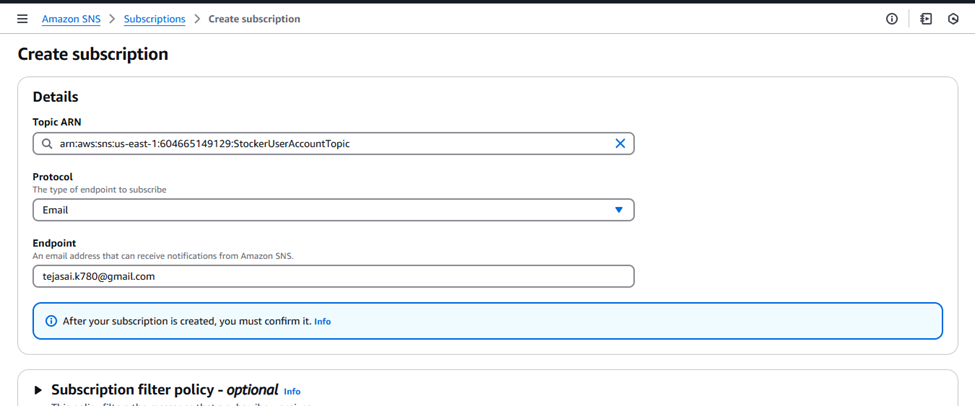


* Note down the Topic ARN.

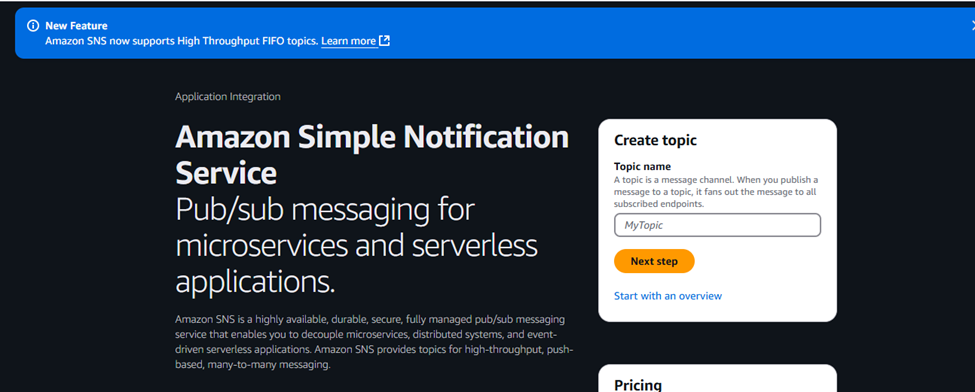


* Click create subscription

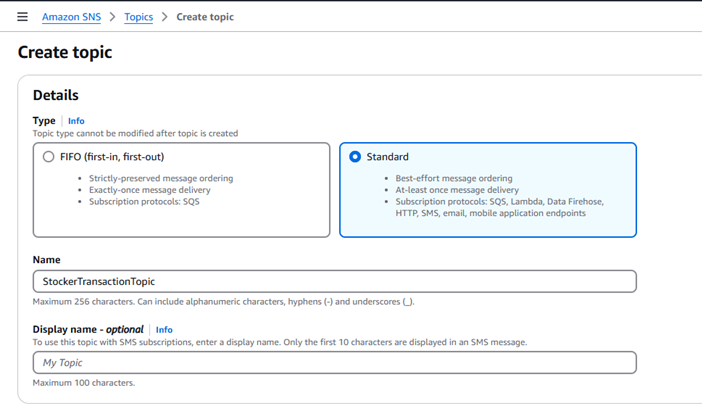
* Select the created “StockerUserAccountTopic” topic ARN and select protocol as Email The type in your Email
* Scroll down and create

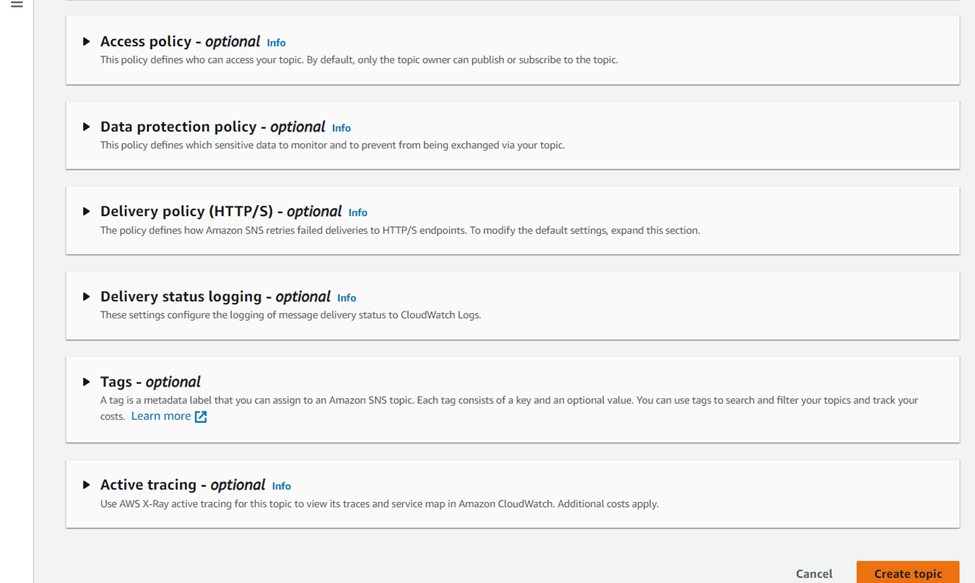


* Repeat the same to create StockerTransactionTopic

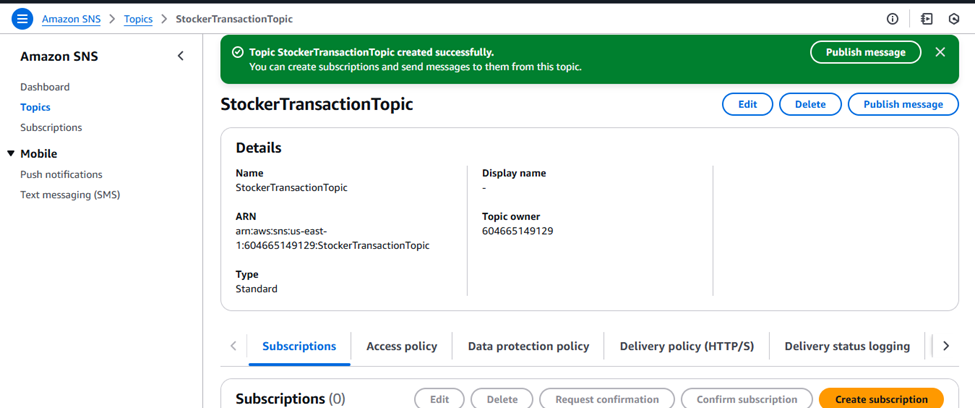


* In Create Topic  Name  type “StockerTransactionTopic” and Choose Standard type for general notification use cases and Click on Create Topic.

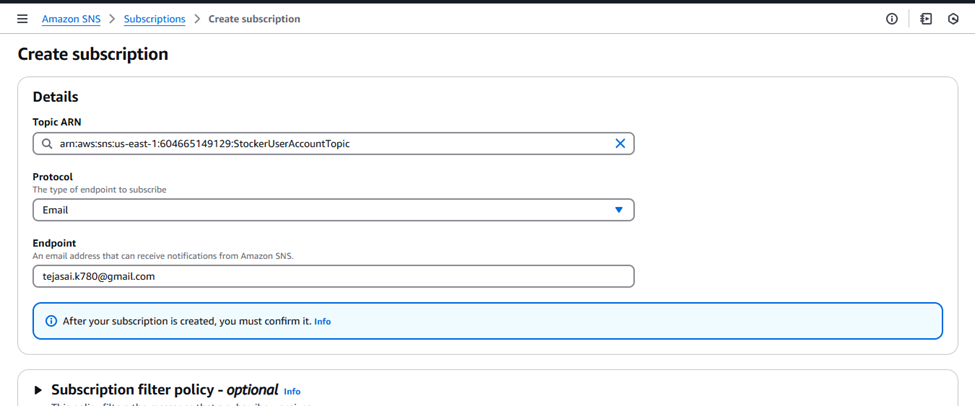




* Note down the Topic ARN.



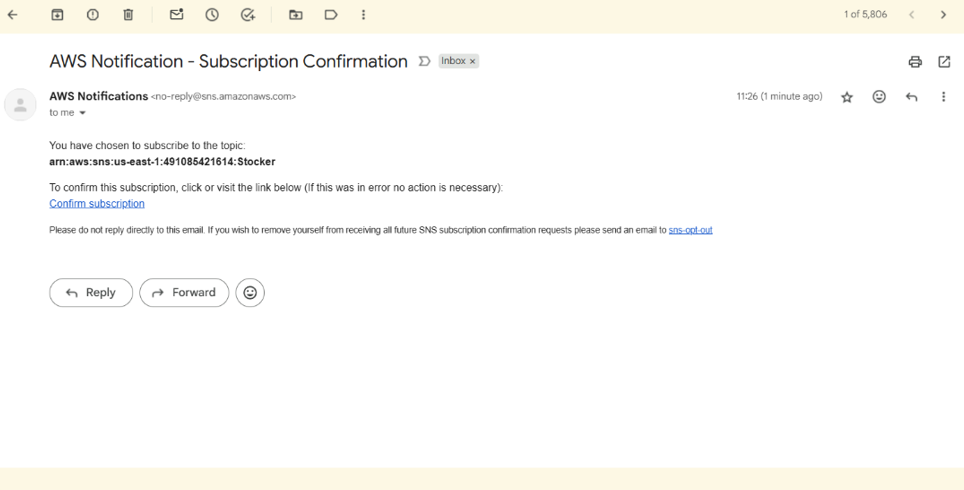
* Click create subscription
* Select the created “StockerTransactionTopic” topic ARN and select protocol as Email The type in your Email
* Scroll down and creat

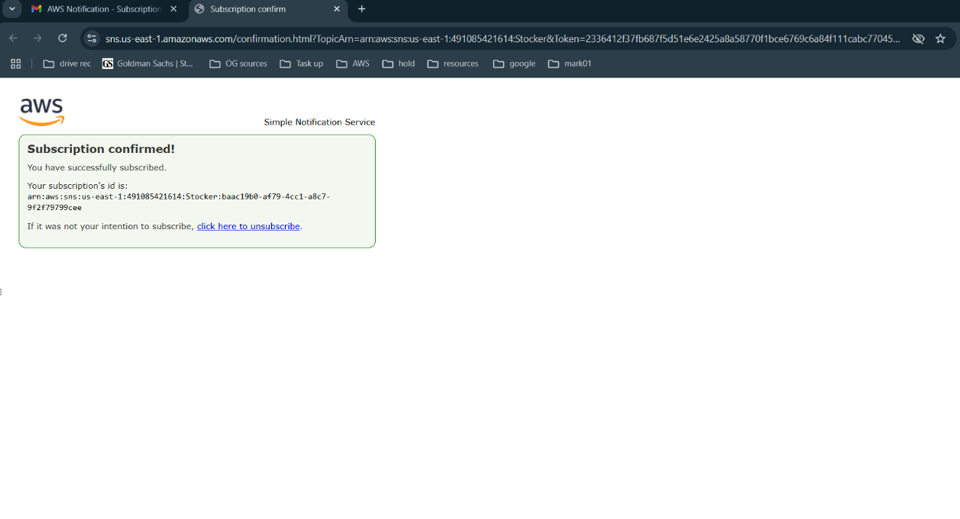


**Subscribe users and admin**

After subscription request for the confirm subscription will be sent to your mail

* Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.



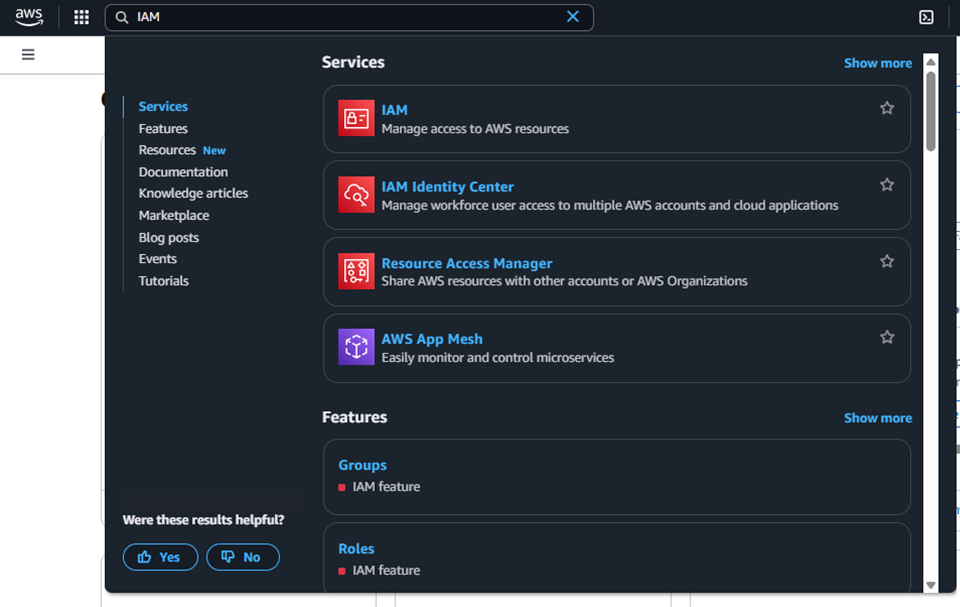


**Milestone 5 : IAM Role Setup**

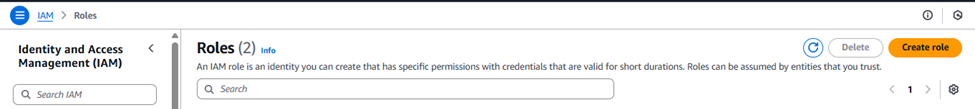
IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

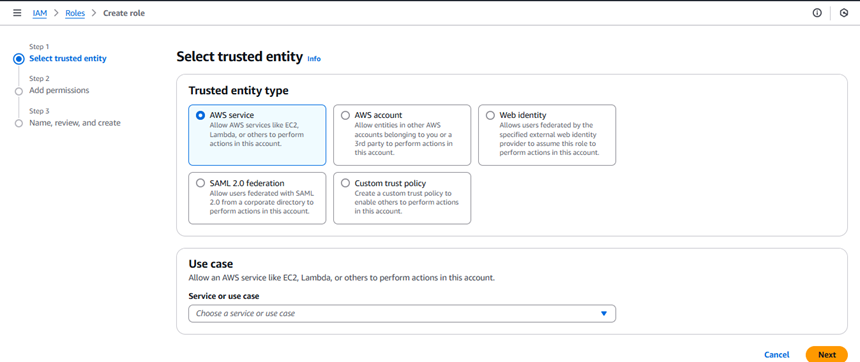
**Create IAM Role**

* In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.

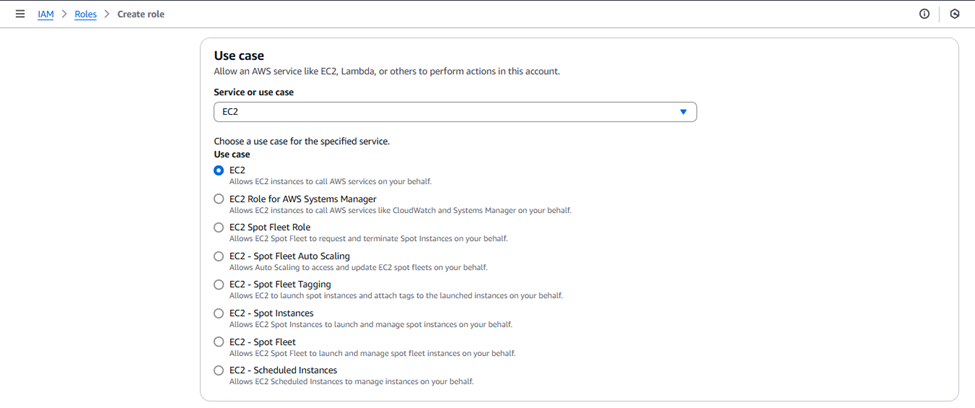


* Click create role

In Select trusted entity select AWS service



* In Use case select EC2 - Allow EC2 instances to call on your behalf



* Scroll down and Click next

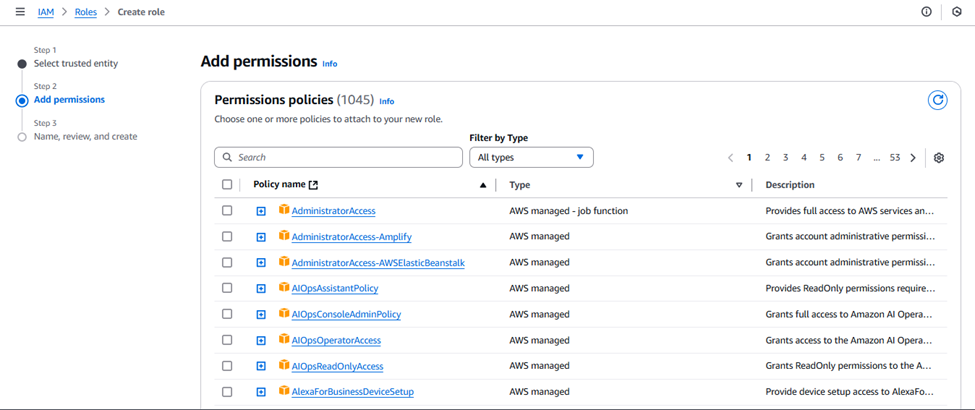
**Attach Policies**

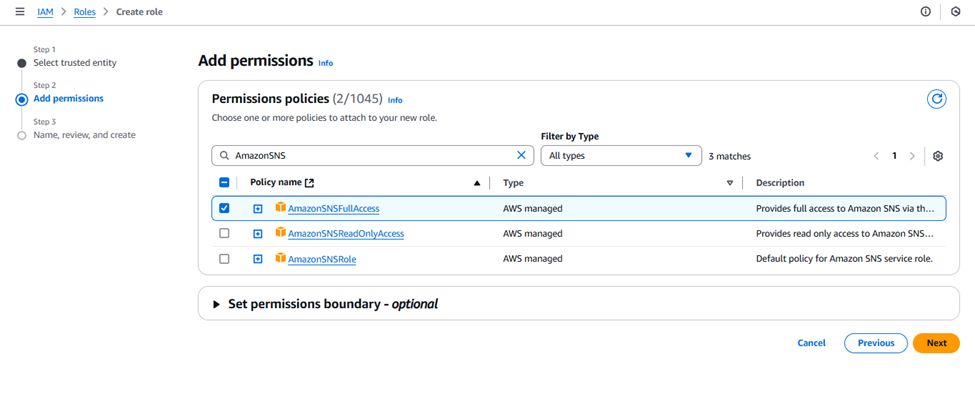
**Attach the following policies to the role:**

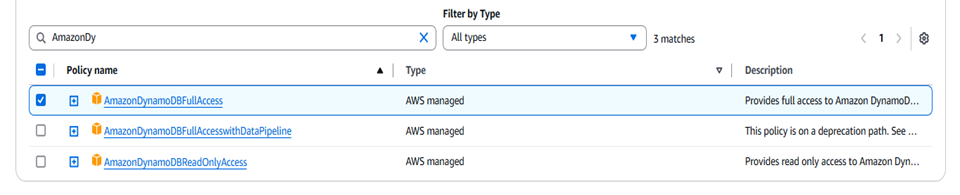
AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.

AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.

* In Add Permissions search and select AmazonSNSFullAccess and AmazonDynamoBDFullAccess

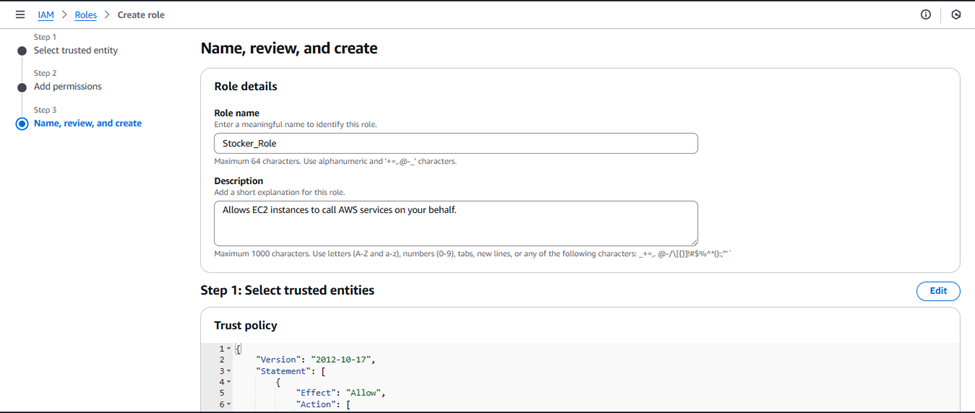


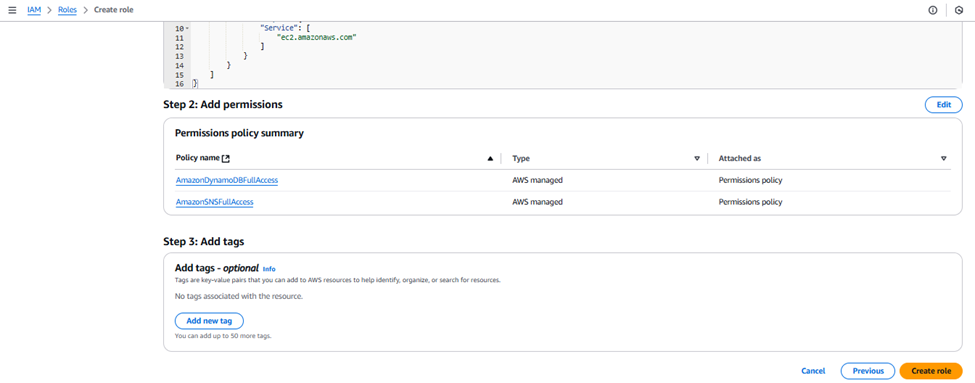




* Click next

* In Name, Review and Create type Role name as “Stocker\_Role”



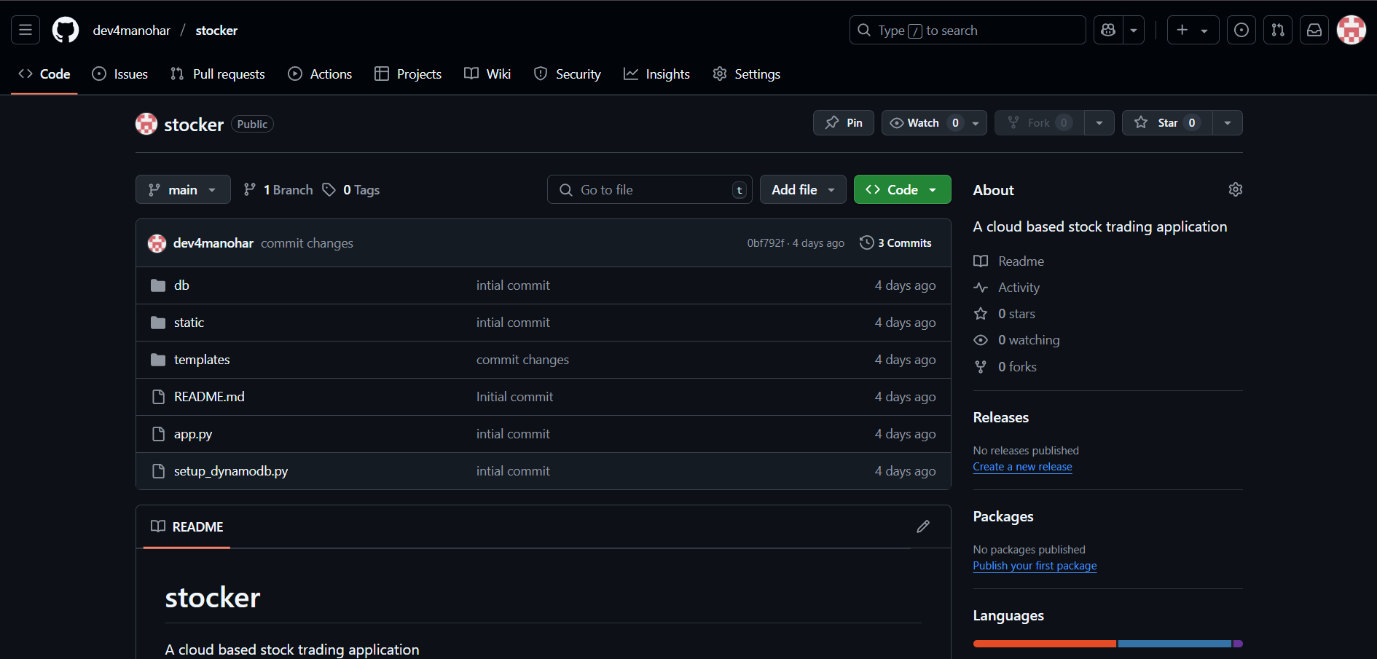


**Milestone 6 : EC2 Instance setup**

To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an Elastic IP for consistent public access, and configure your application or services to be publicly accessible.

**Load your Project Files to GitHub**

* Load your Flask app and Html files into GitHub repository.



**Launch an EC2 instance to host the Flask**

* In the AWS Console, navigate to EC2 and launch a new instance.
* Click on Launch instance to launch EC2 instance
* Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).
* Create and download the key pair for Server access.

**Configure security groups for HTTP, and SSH access.**

* Click Edit
* To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

* Now connect the EC2 with the files
* EC2 Linex2 Terminal will open in a new tab

* EC2 Linex2 Terminal will open in a new tab

**Milestone 7 : Deployment on EC2**

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

**Install Software on the EC2 Instance**

Install Python3, Flask, and Git:

On Amazon Linux 2:

* run sudo yum update -y
* run sudo yum install -y python3 python3-pip git

**Clone Your Flask Project from GitHub**

**Clone your project repository from GitHub into the EC2 instance using Git.**

Run: git clone : git@github.com:dev4manohar/stocker.git

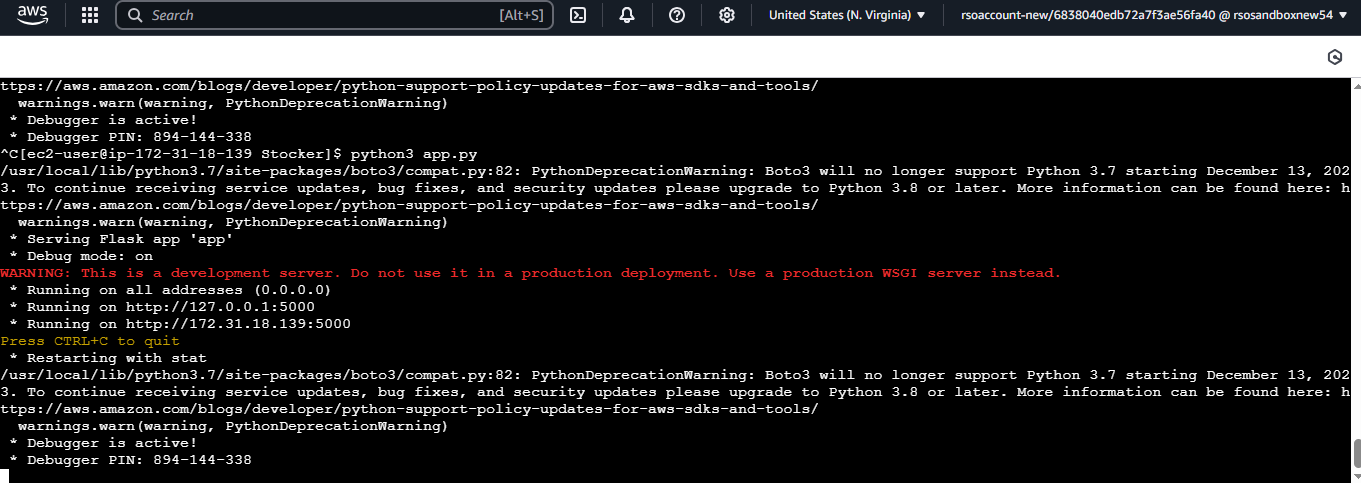
* This will download your project to the EC2 instance.

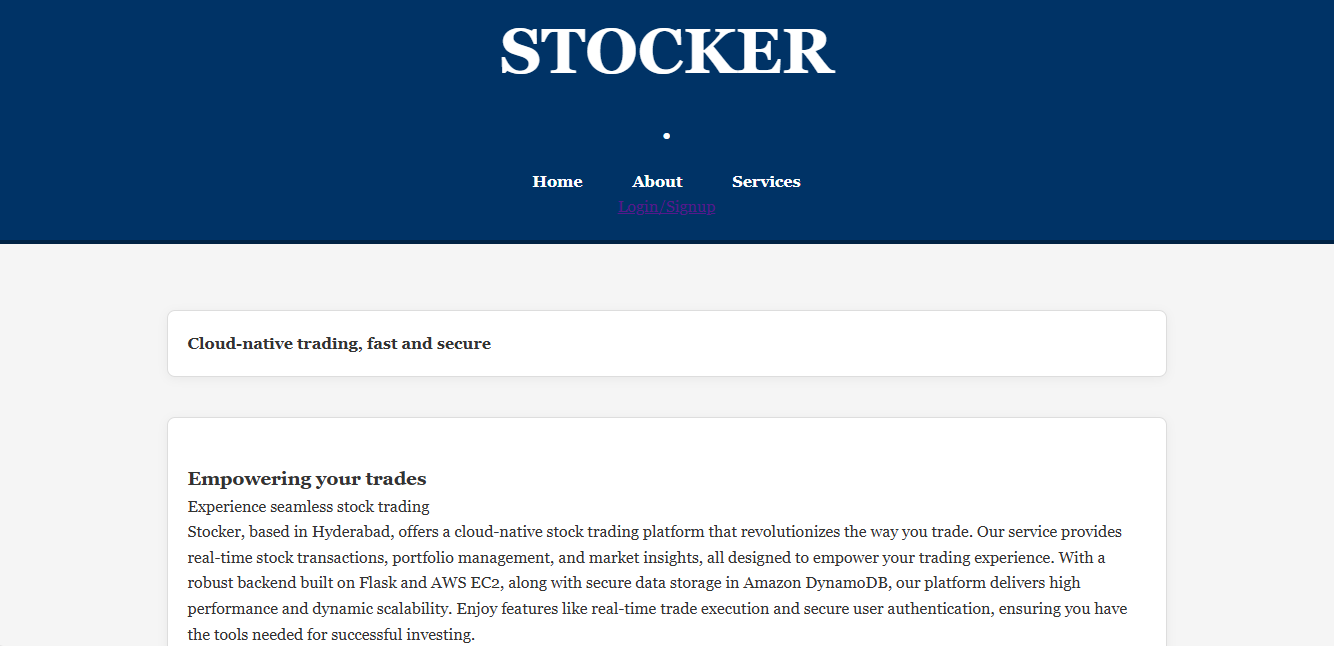
**To navigate to the project directory, run the following command:**

* run cd Stocker
* run cd stocker - AWS
* Now run pip3 install -r requirements.txt

**Now run the DynamoDb setup file by running the command**

* run python3 setup\_dynamodb.py
* Go to DynamoDB tables section and verify weather the tables have been created
* Return to EC2 Instance terminal
* Run the flask Application python3 app.py





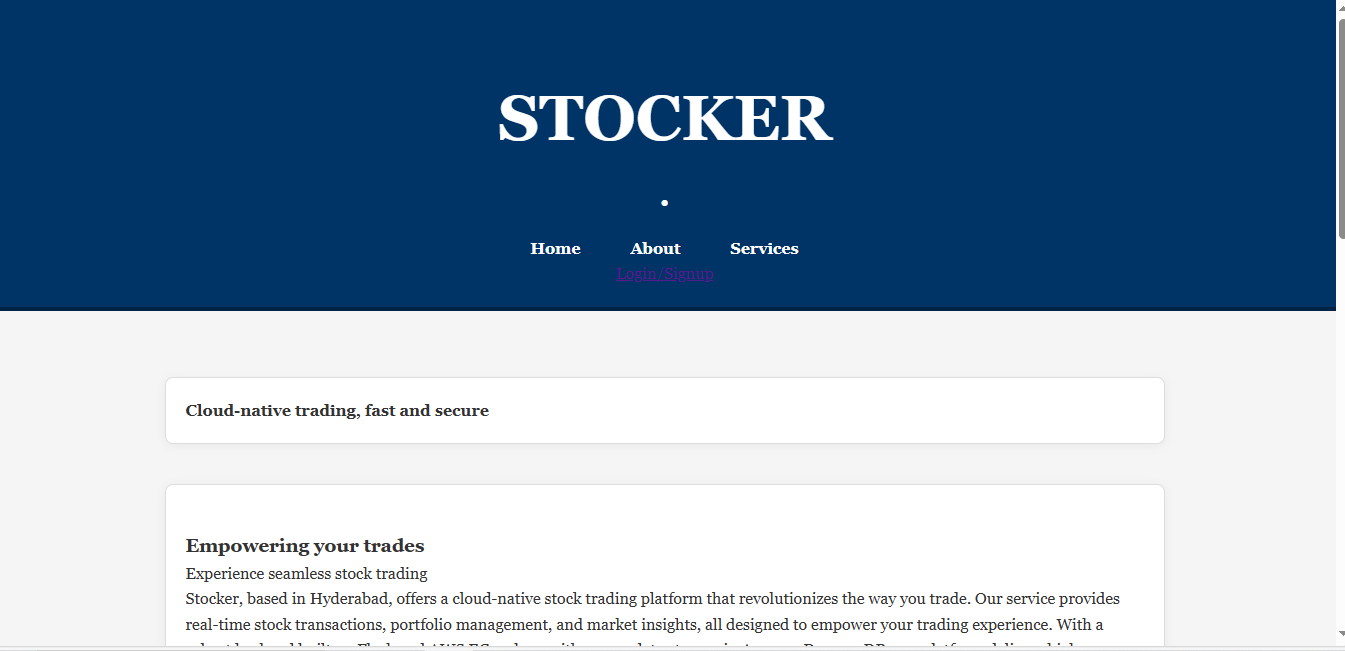
public link : http://172.31.18.139:5000/

**Milestone 8 : Testing and Deployment**

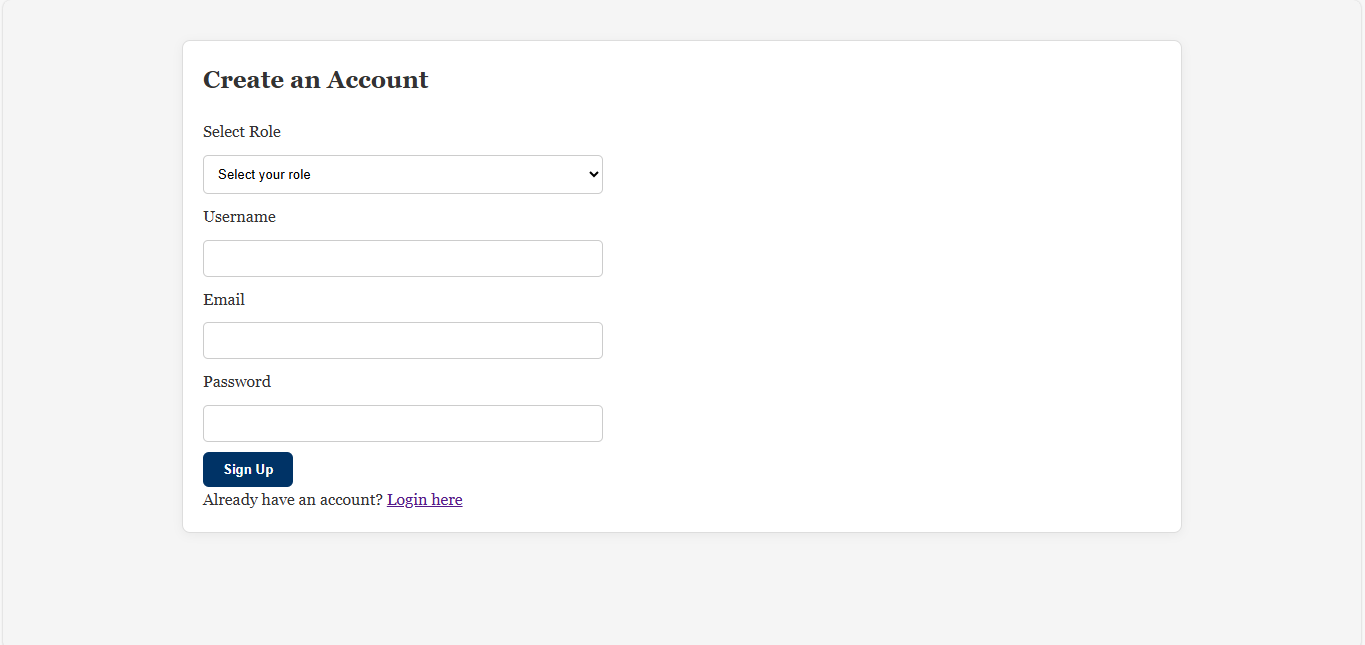
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

**Functional Testing to verify the Project**

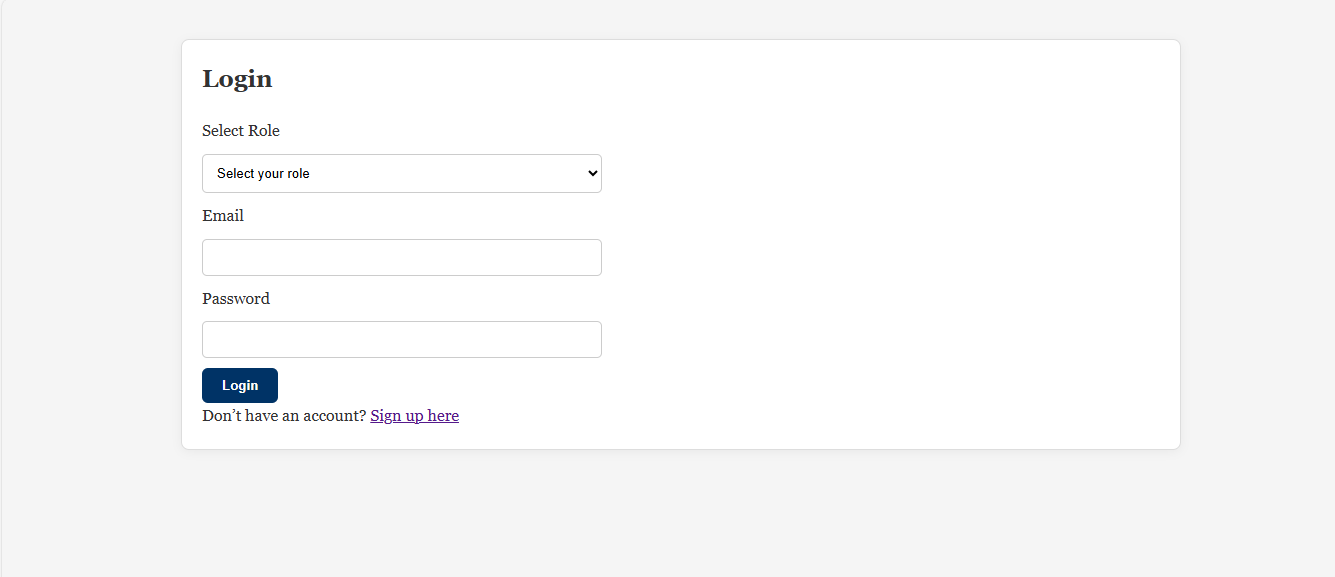
Home page:-



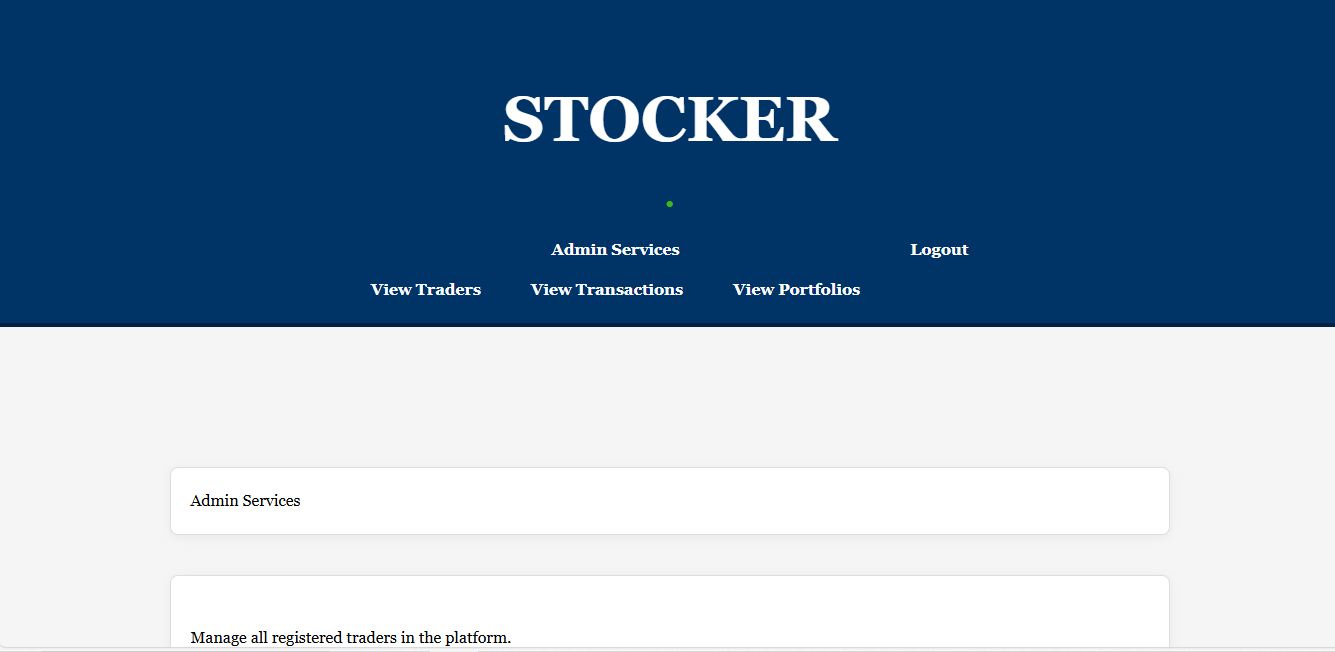
Signup page:-



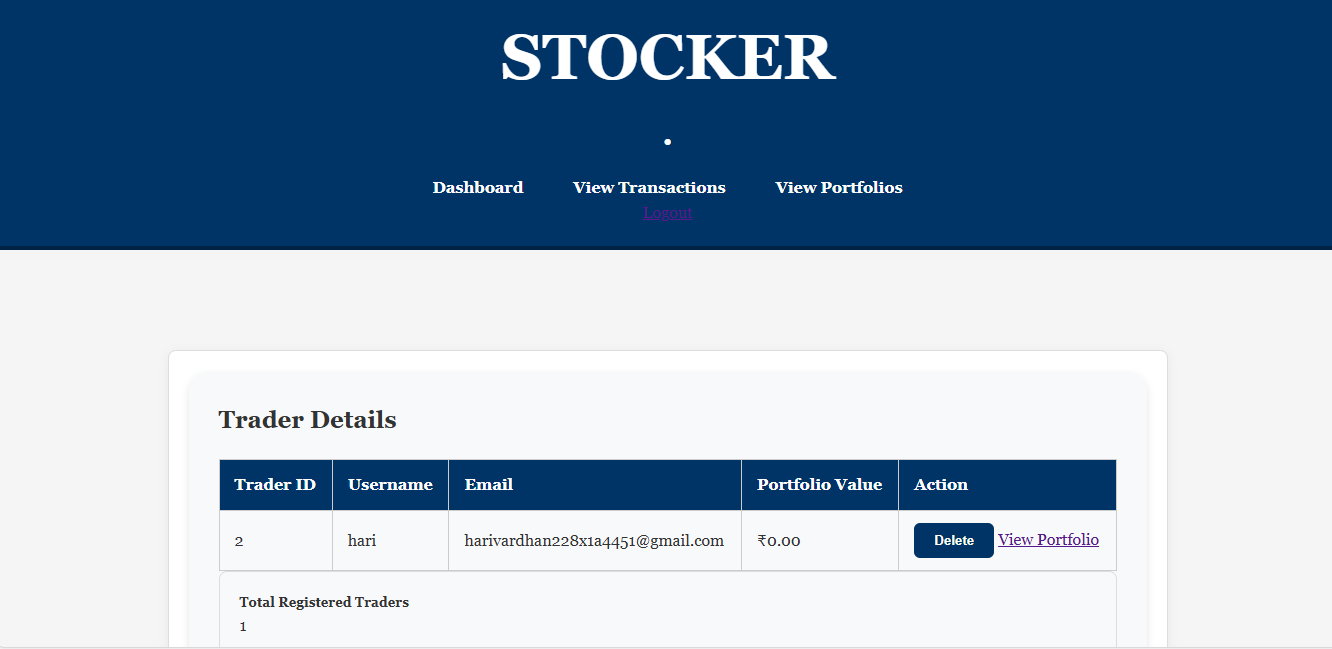
Login page:-



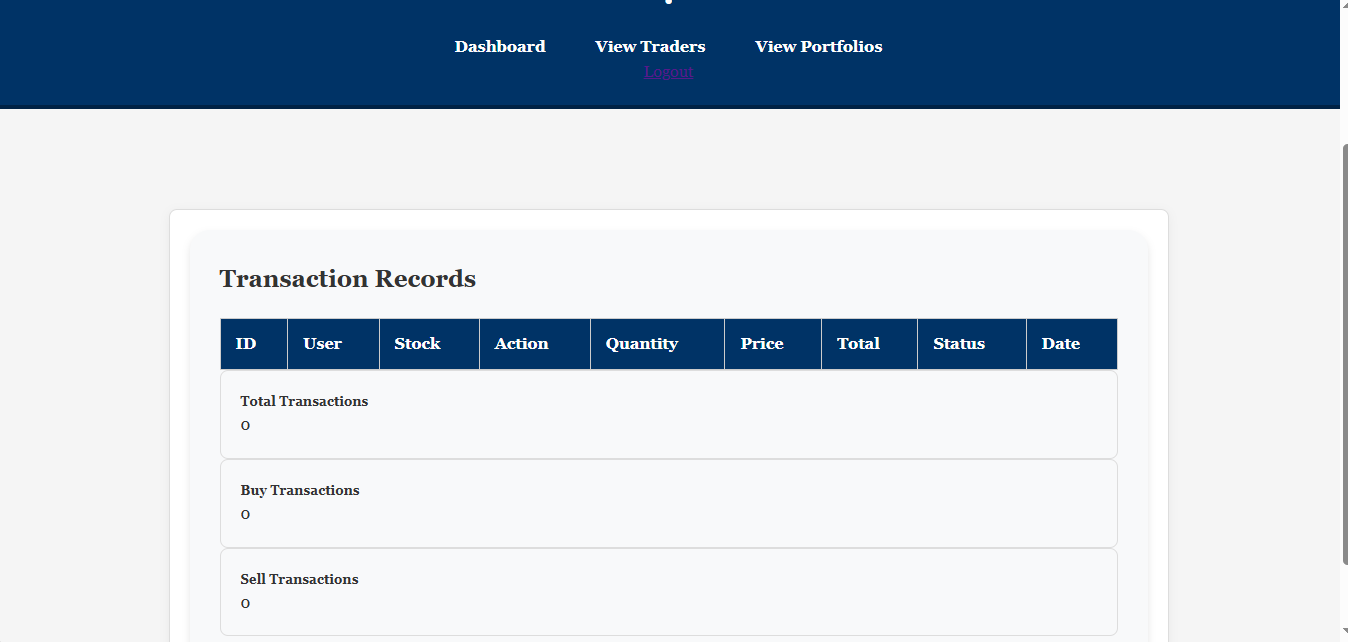
 Admin Dashboard:-



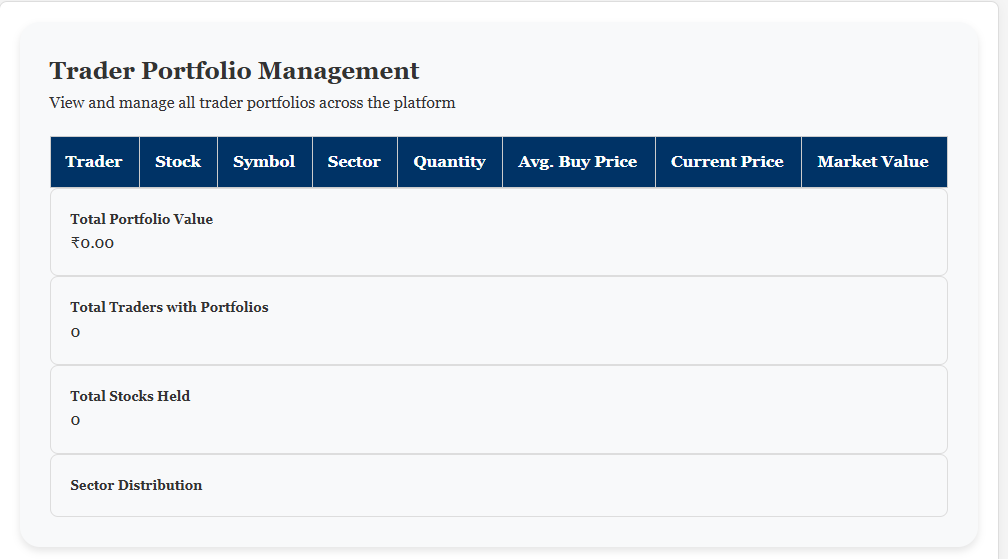
Trader details:-



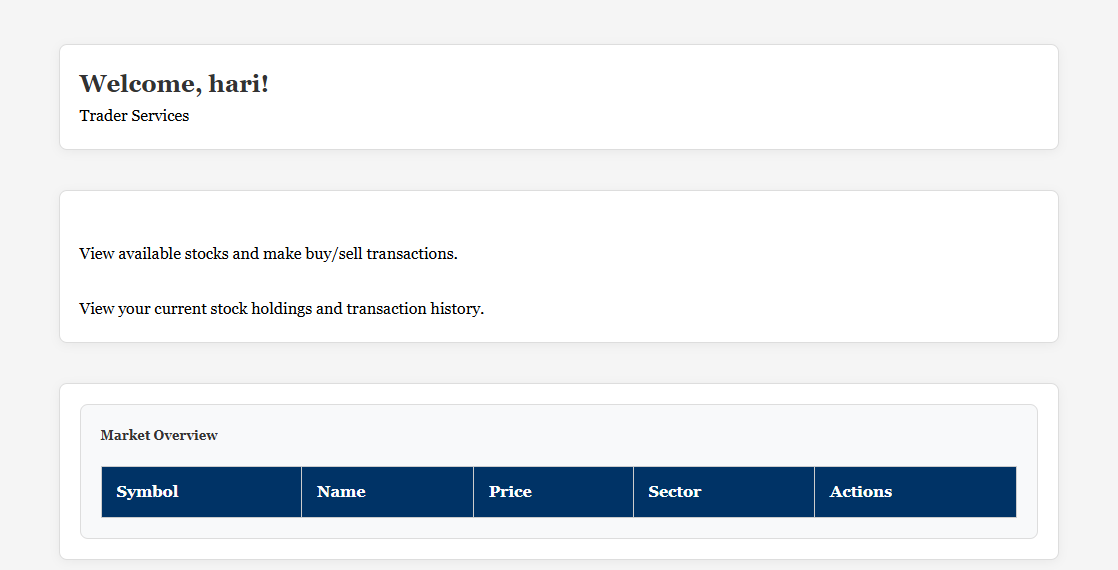
Transaction Records:-



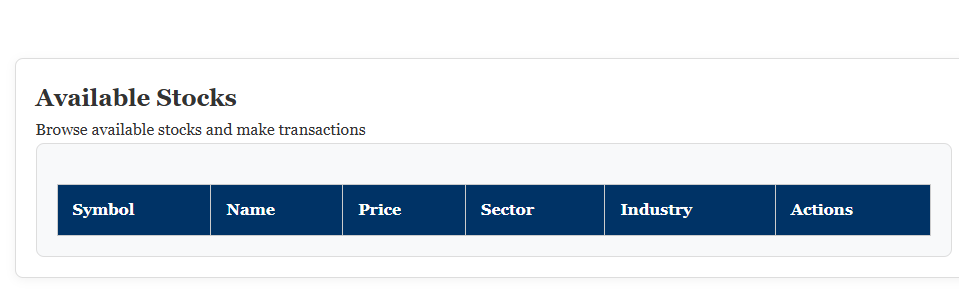
Trader Portfolio Management:-



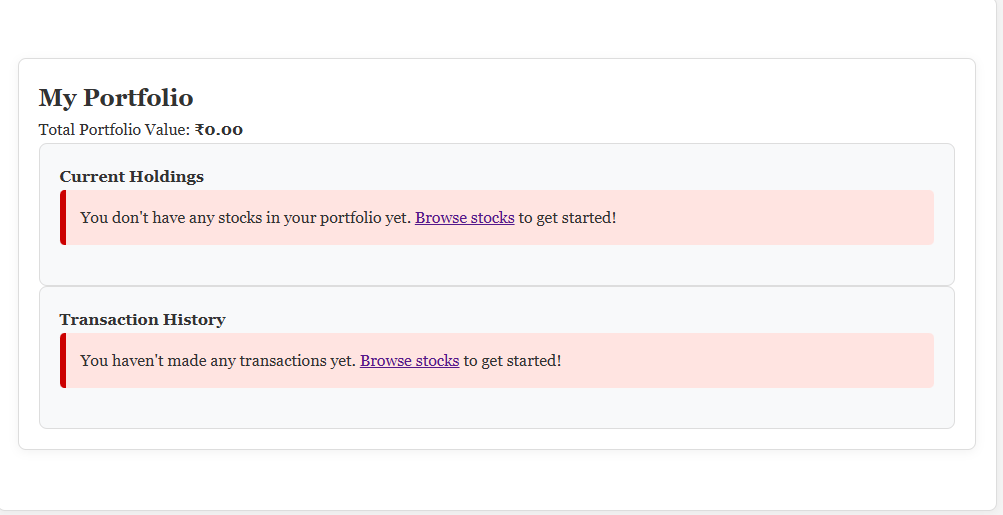
Trader Dashboard:-



Available Stocks:-



My Portfolio:-



**Conclusion**

In this project, we successfully implemented a comprehensive stock trading application using Flask framework, deployed on Amazon EC2. The application leverages AWS's robust ecosystem, particularly DynamoDB for efficient and scalable database operations, providing reliable storage for user accounts, stock portfolios, and transaction histories. This serverless database approach ensures high availability and automatic scaling based on application demands.

Integration with Amazon SNS has enhanced the user experience by implementing real-time notifications for critical events such as user authentication (login/signup) and stock transactions (buy/sell). This notification system keeps users informed about account activities and stock movements, adding an important security and convenience layer to the application. The event-driven architecture enables seamless communication between different components of the system.

The deployment on EC2 instances provides the flexibility and computational power necessary for handling varying user loads while maintaining responsive performance. This cloud-based approach eliminates the need for physical infrastructure management and allows for easy scaling as the application grows. Overall, this project demonstrates the effective use of modern cloud technologies to build a robust, scalable, and feature-rich stock trading platform that can be further expanded with additional functionalities in the future.