**Online Bookstore Management System**

**Abstract**

The primary objective of this project is to create an innovative and user-friendly Online Bookstore Management System. The system allows users to efficiently search for and purchase books based on various criteria such as title, author, and subject. Through a secure online platform, customers can conveniently order books, make payments using credit cards or PayPal, and have their selected books delivered to their doorstep. The system aims to provide a seamless and time-saving alternative to traditional bookstore shopping.

**Introduction**

In the ever-evolving landscape of software development, this project leverages contemporary architectural design and concepts to deliver a robust and adaptable Online Bookstore Management System. As the stewards of application development change, so does the philosophy and implementation specifics. The design of this system prioritizes flexibility, enabling it to embrace new ideas rather than relying on outdated approaches.

Features of the Online Bookstore Management System (Django - Admin Side):

Dashboard: The admin dashboard serves as the central hub for accessing all core functions of the system.

Manage Books: Administrators can efficiently administer book-related information, including adding, modifying, and removing books from the system.

Manage Categories: This feature allows administrators to update and delete information related to book categories, enhancing the system's organizational structure.

Secure Login and Logout: The system incorporates a secure login and logout system by default, ensuring the protection of sensitive user information.

Manage Orders: Administrators can review and manage customer orders on a case-by-case basis, having the authority to approve or reject orders. A comprehensive list of client orders is maintained for reference.

Manage Users: The administrator has the capability to add, update, and block user accounts, ensuring effective user management within the system.

Features of the Online Bookstore Management System (Django - Frontend Side):

Login Page: Customers can securely log in by entering their credentials on the dedicated login page.

Register Page: New customers can create their login credentials on the registration page, facilitating a streamlined onboarding process.

Homepage: The default landing page displays an array of books available for purchase. Customers can search for books by entering keywords in the search box.

View Book Page: Customers can explore detailed information about a selected book, add it to their cart, and view unique product details.

Cart List: The page displays the selected products in the customer's cart, enabling a seamless checkout process.

My Orders Page: Customers can view a comprehensive list of their past orders, promoting transparency and order tracking.

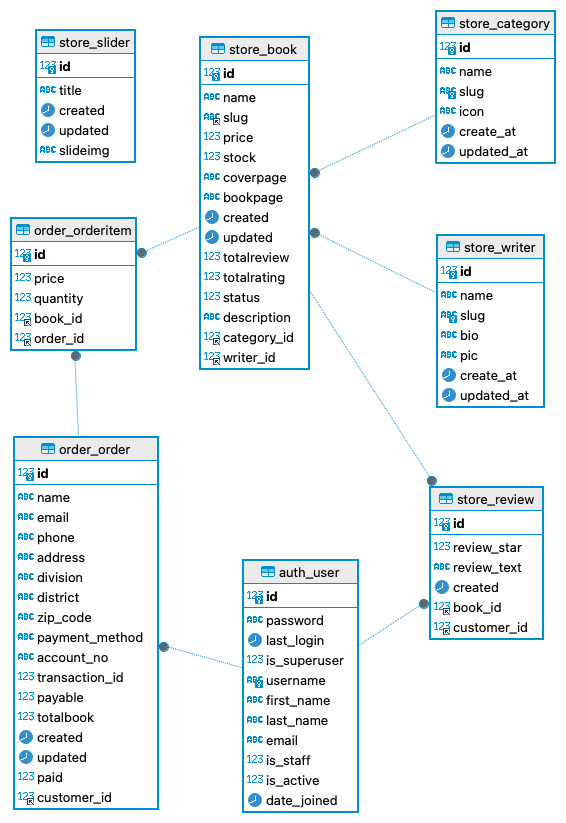
PayPal and Credit Card Payments: The system supports secure payment methods, including PayPal and credit card transactions, enhancing user convenience.

**Database Description**

The system is designed around a central database that stores crucial information. The administrator's details, such as name, email, and login credentials, are securely stored. Customer information includes name, address, and phone number.

For order tracking, the system maintains data on purchased books, payment details (type, amount, date, time, transaction ID), order status, and overall cost.

Each book is associated with category information, author details, and specifics like title, description, writer's name, rating, and reviews, contributing to a comprehensive and organized database structure.



An Entity-Relationship (ER) diagram is a visual representation of the relationships among entities within a database. It helps to illustrate the logical structure of a database by showcasing how different entities are related to each other. In the context of the Online Bookstore Management System, let's delve deeper into the ER diagram based on the provided tables.

*Entities:*

Admin Entity:

Attributes: id, password, last\_login, is\_superuser, username, first\_name, last\_name, email, is\_staff, is\_active, date\_joined.

Category Entity:

Attributes: id, name, slug, icon, create\_at, updated\_at.

Writer Entity:

Attributes: id, name, slug, bio, pic, create\_at, updated\_at.

Book Entity:

Attributes: id, name, slug, price, stock, coverpage, bookpage, created, updated, totalreview, totalrating, status, description, category\_id, writer\_id.

Order Entity:

Attributes: id, customer\_id, name, email, phone, address, division, district, zip\_code, payment\_method, account\_no, transaction\_id, payable, totalbook, created, updated, paid.

*Relationships:*

Admin-Order Relationship:

An admin can be associated with multiple orders, but each order is managed by one admin. This relationship is illustrated by the admin\_id foreign key in the Order table.

Cardinality: One-to-Many (1:N), as one admin can manage multiple orders.

Modality: Mandatory, as each order must be associated with an admin.

Category-Book Relationship:

A category can have multiple books, but each book belongs to one category. This relationship is depicted by the category\_id foreign key in the Book table.

Cardinality: One-to-Many (1:N), as one category can have multiple books.

Modality: Mandatory, as each book must belong to a category.

Writer-Book Relationship:

A writer can have multiple books, but each book is written by one writer. This relationship is represented by the writer\_id foreign key in the Book table.

Cardinality: One-to-Many (1:N), as one writer can write multiple books.

Modality: Mandatory, as each book must have a writer

Customer-Order Relationship:

A customer can place multiple orders, but each order is associated with one customer. This relationship is indicated by the customer\_id foreign key in the Order table.

Cardinality: One-to-Many (1:N), as one customer can place multiple orders.

Modality: Mandatory, as each order must be associated with a customer.

**Deployment**

Deploying a Django application on AWS involves several steps, including setting up infrastructure, configuring services, and managing domain registration. Below is a detailed procedure for deploying a Django application on AWS using services like EC2, RDS, CodeCommit, Route 53, ACM (AWS Certificate Manager), and API Gateway.

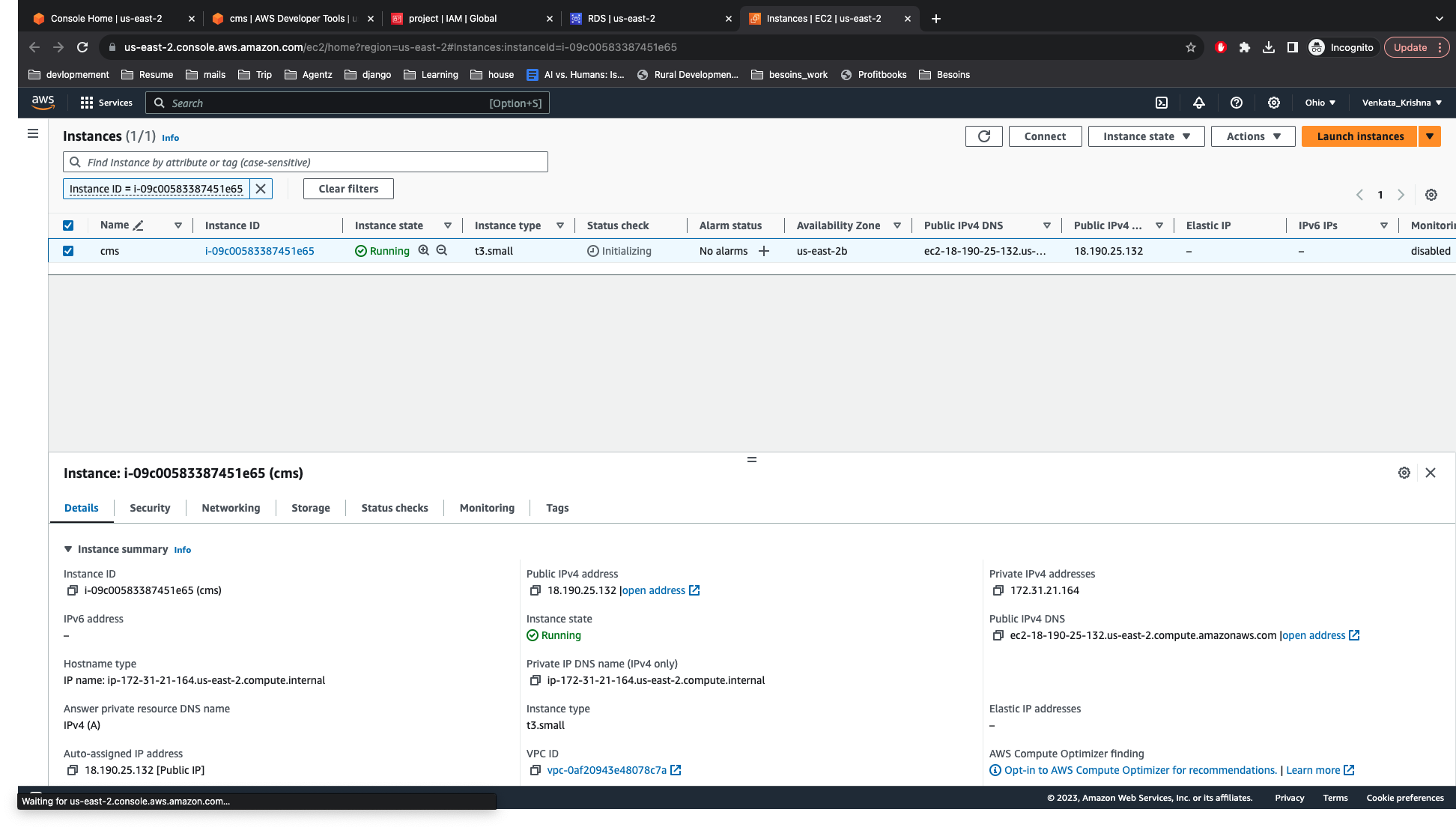
*Set Up EC2 Instance and RDS*

AWS EC2 and RDS Creation Provision EC2 instances (servers) for running your web application by selecting suitable Machine Image, Server Type, Storage, Security Group and Key pair. An RDS instance is created for the database. Choose the right database engine, server type, and settings that suit your application requirements.

Create EC2 Instance:

Launch an EC2 instance with the desired specifications.

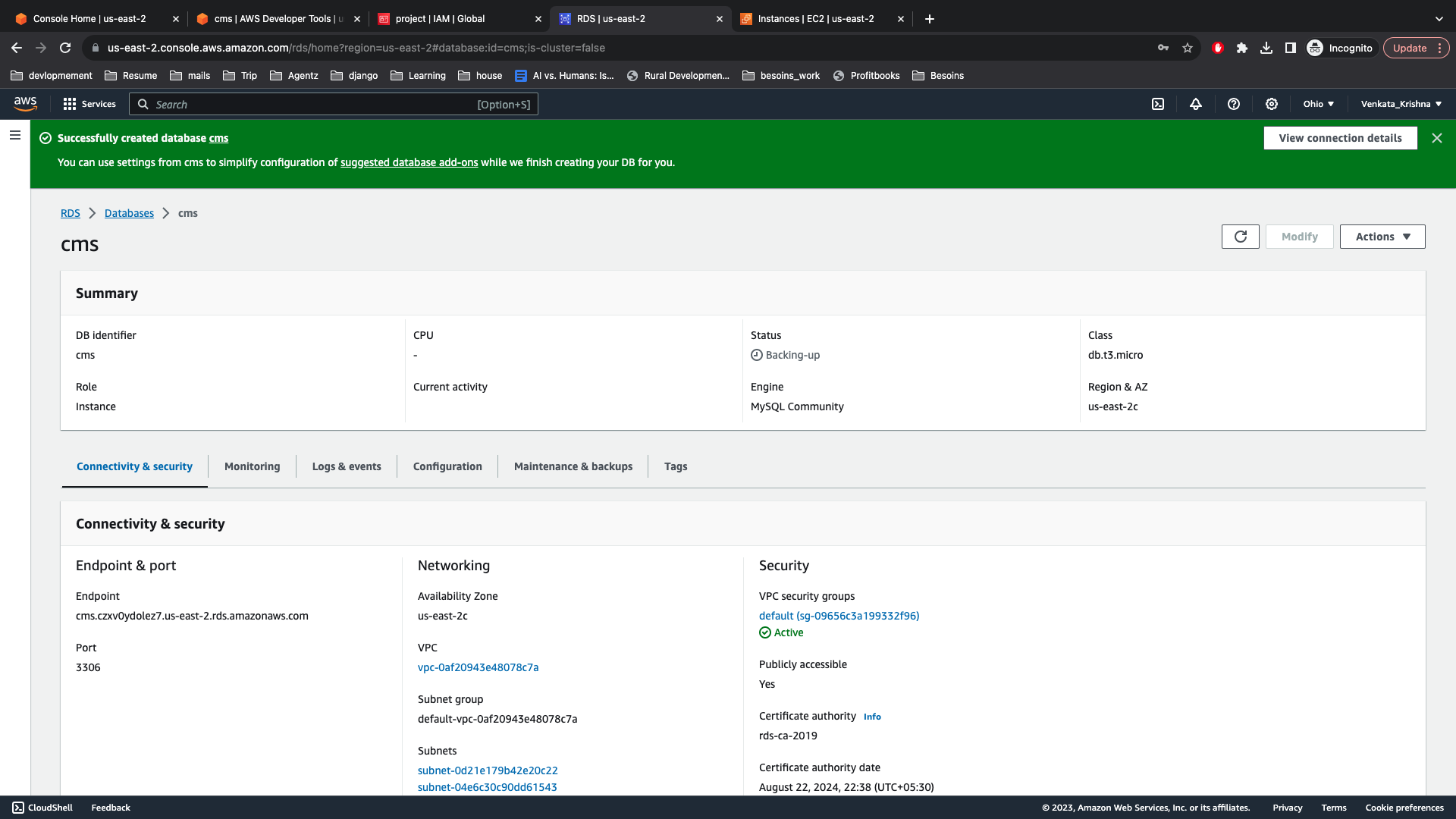
Configure security groups to allow necessary inbound traffic, especially for SSH (port 22) and HTTP/HTTPS (ports 80 and 443).



Create RDS Instance:

Set up an RDS instance with the required database engine (e.g., PostgreSQL, MySQL).

Configure security groups to allow database connections from the EC2 instance.



*CodeCommit Repository and Code Deployment*

CodeCommit Repository Creation Create a repository in AWS CodeCommit and push all your Django application code into this repository. This includes the application logic, UI, and all other related code.

Create CodeCommit Repository:

Set up a CodeCommit repository to host your Django application code.

Push your Django project code to the CodeCommit repository.

Clone Code to EC2 Instance:

SSH into the EC2 instance.

Clone the Django project repository using git clone.

Install Dependencies:

Install necessary dependencies and packages on the EC2 instance using pip.

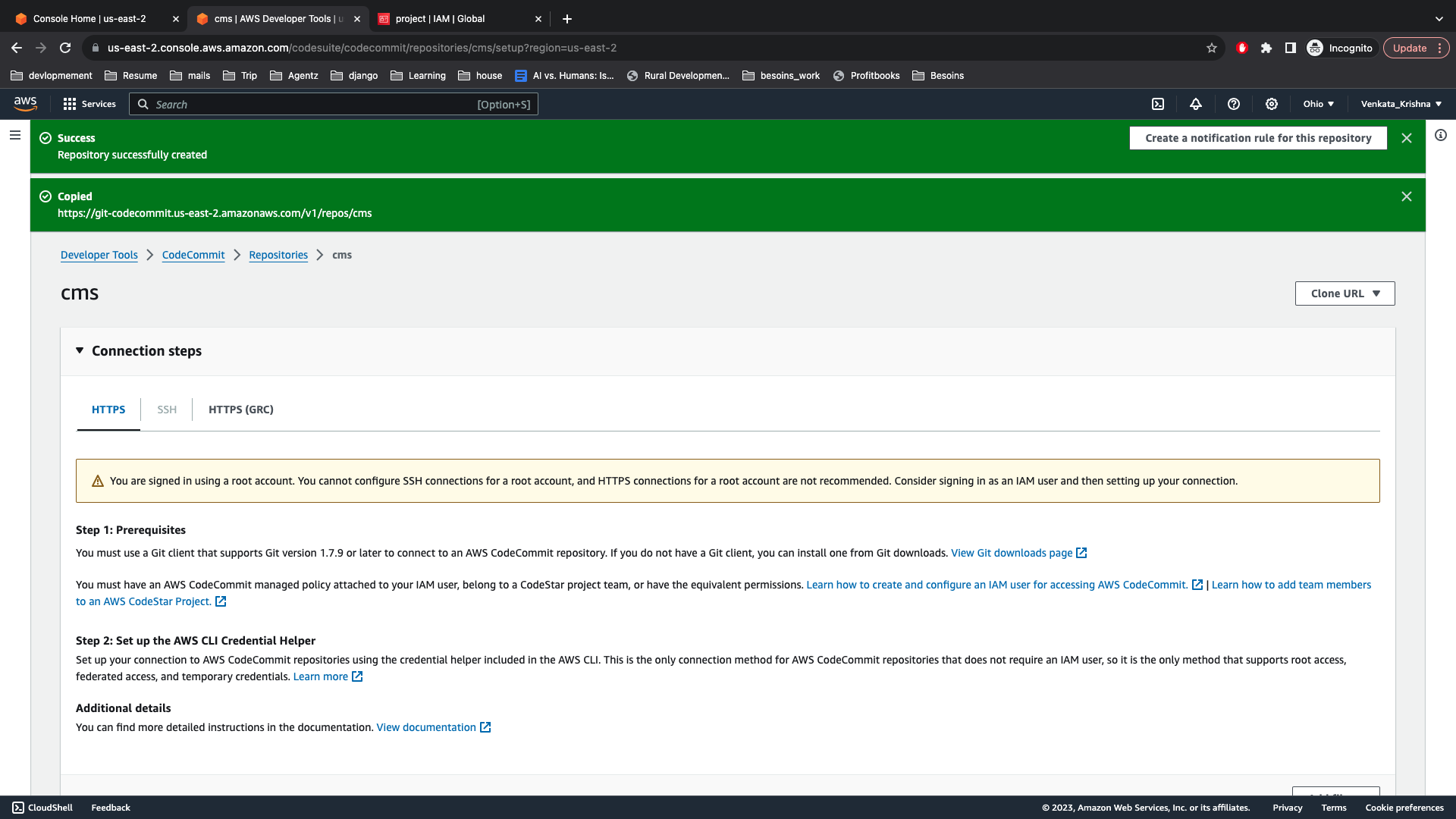
Configure Django Settings:

Update Django settings to use the RDS database.

Configure static files and media settings.

Run Django Application:

Start the Django development server or use a production-ready server like Gunicorn.



*Domain Registration and SSL Certificate*

Register A Domain In Route53 Choose a suitable domain name and register it using AWS Route53.

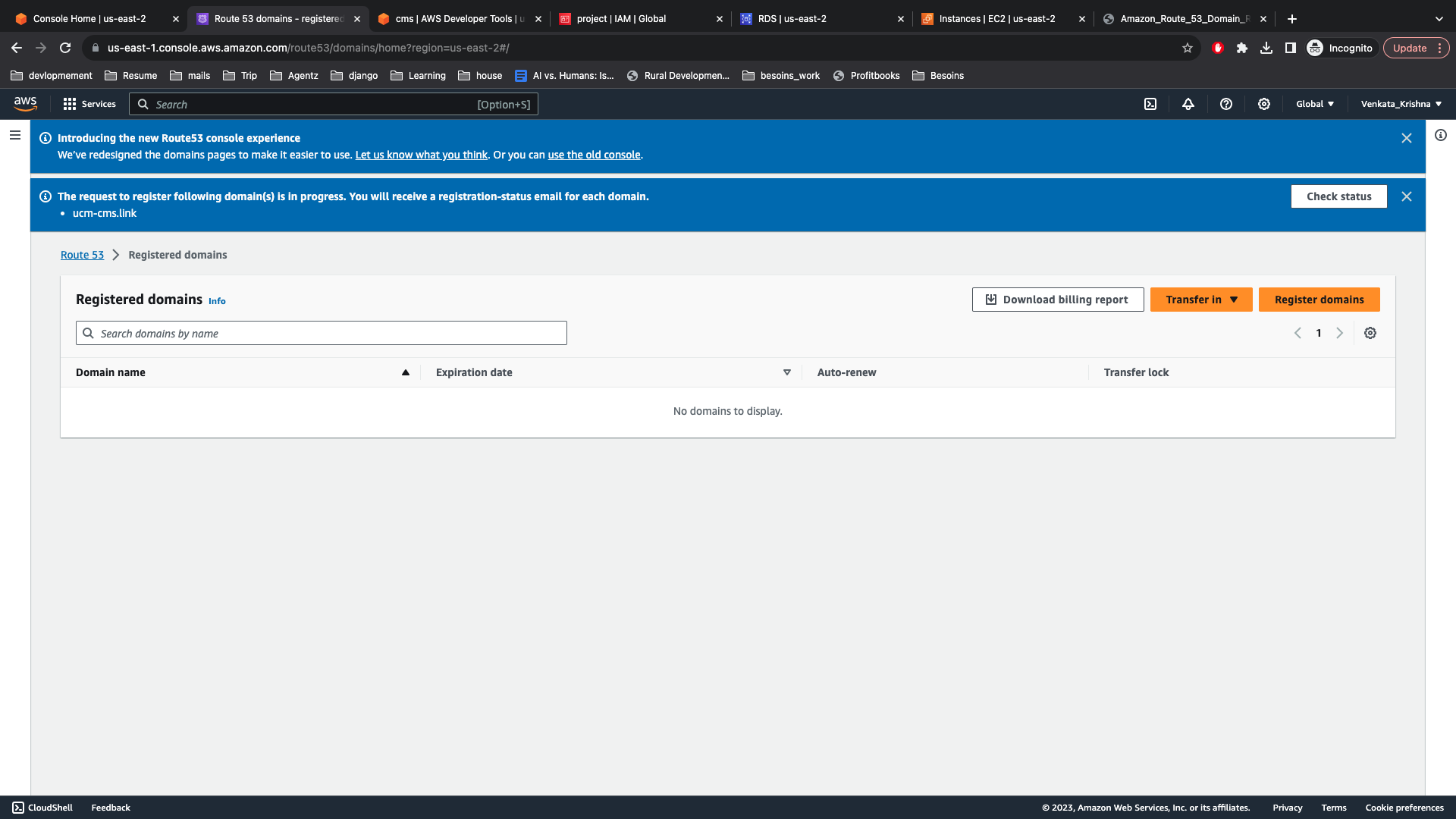
Register a Domain in Route 53:

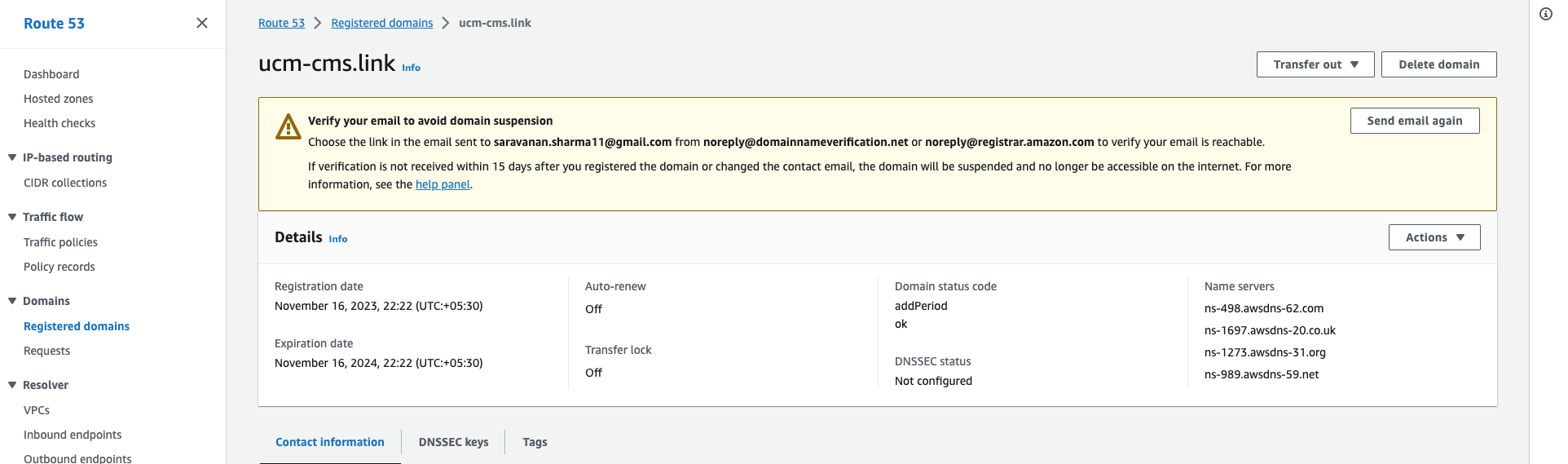
In AWS Route 53, register a new domain or transfer an existing one.

Request ACM Certificate:

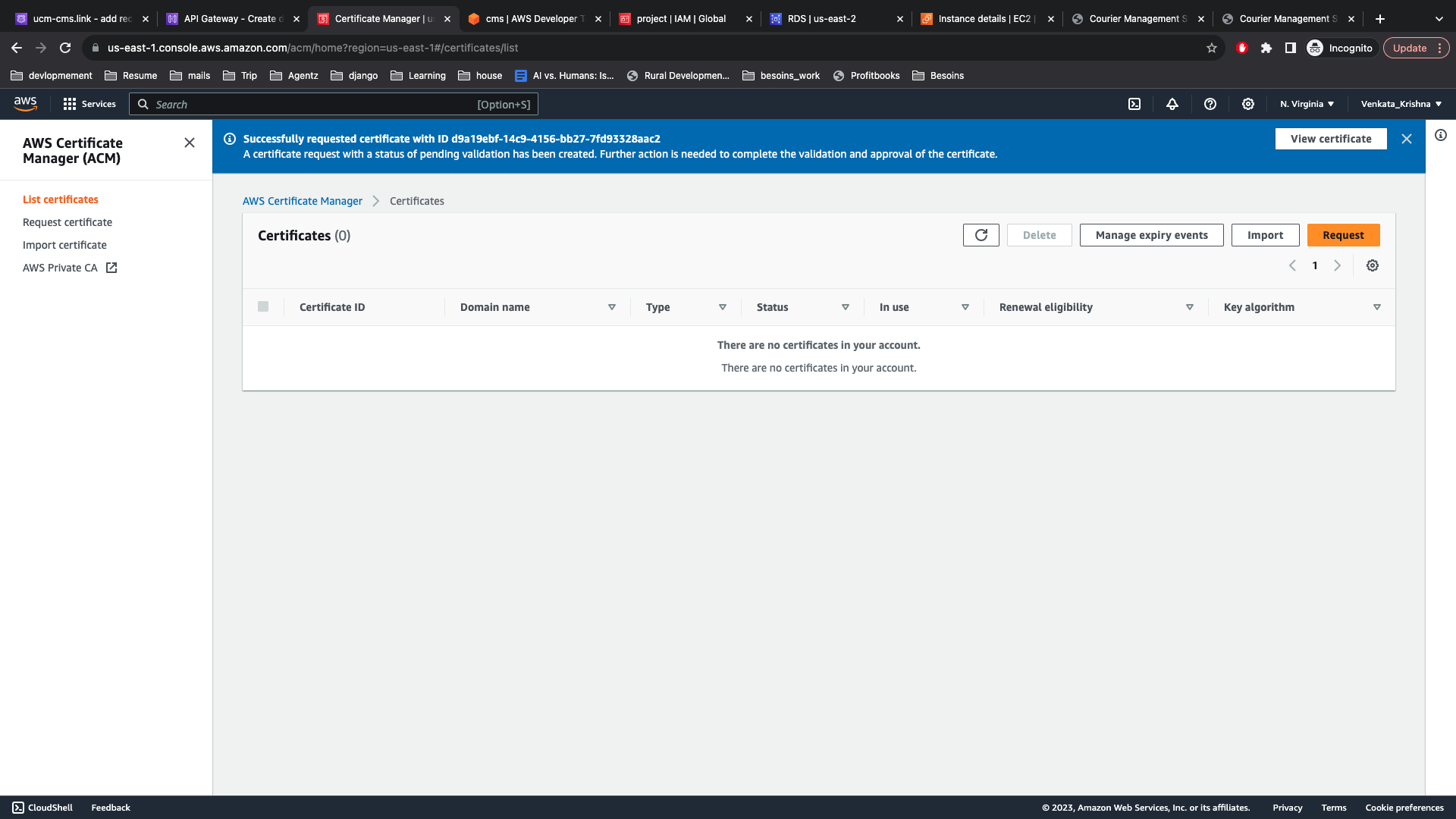
In ACM, request an SSL/TLS certificate for your domain.

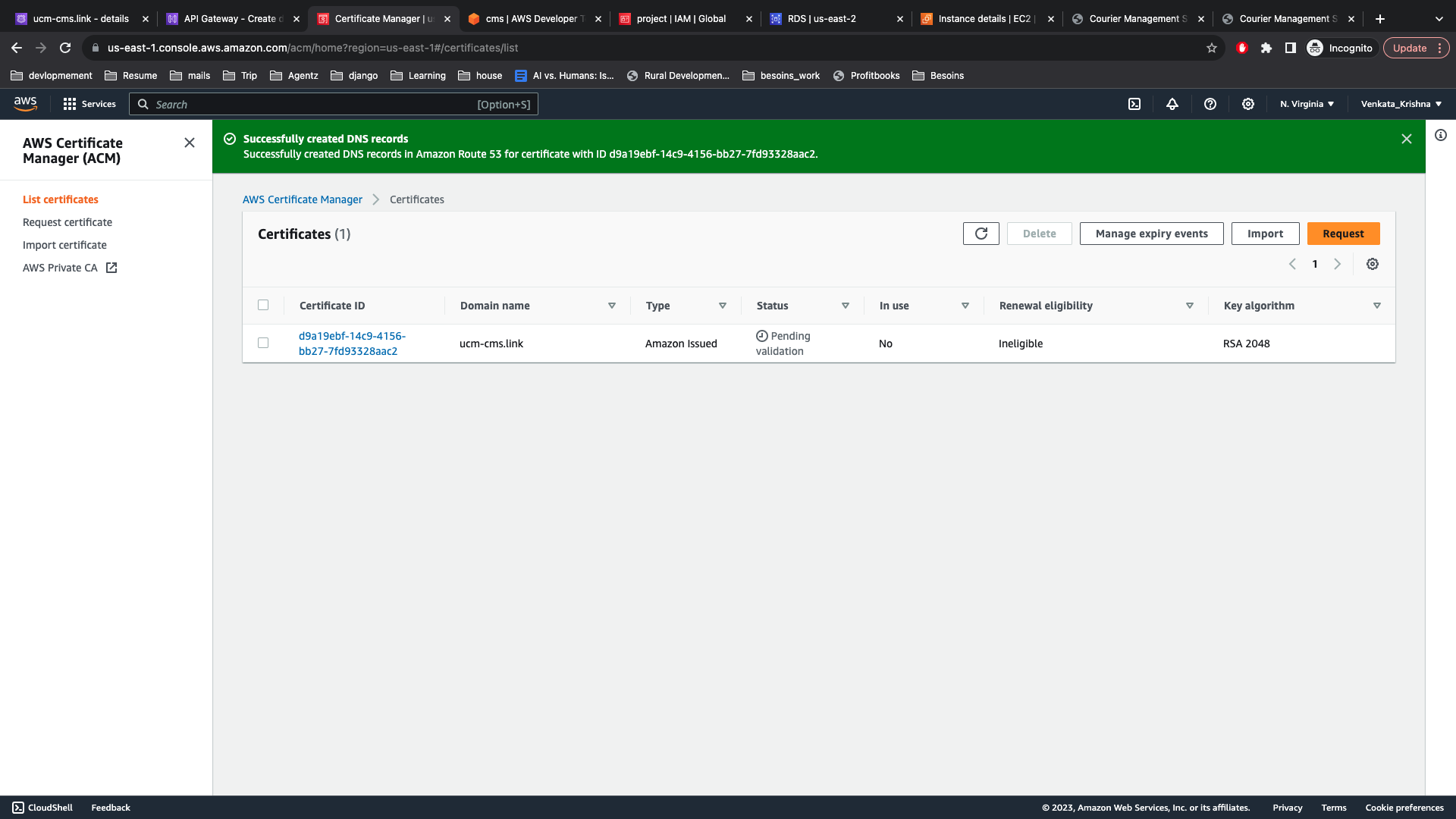
Follow the validation steps to prove domain ownership.





Request for an ACM Certificate Request an ACM (AWS Certificate Manager) certificate for your domain. This will help in HTTPS connections to your application and better security. Validate the certificate with DNS validation or Email validation.





*Configure API Gateway*

Create An API Gateway Create an API Gateway in AWS and configure it to be a HTTP/RESTful API. Enable CORS and other necessary settings as per your application requirements.

Create API Gateway:

Set up a new API in API Gateway.

Create a new resource and method, such as HTTP proxy integration for Lambda.

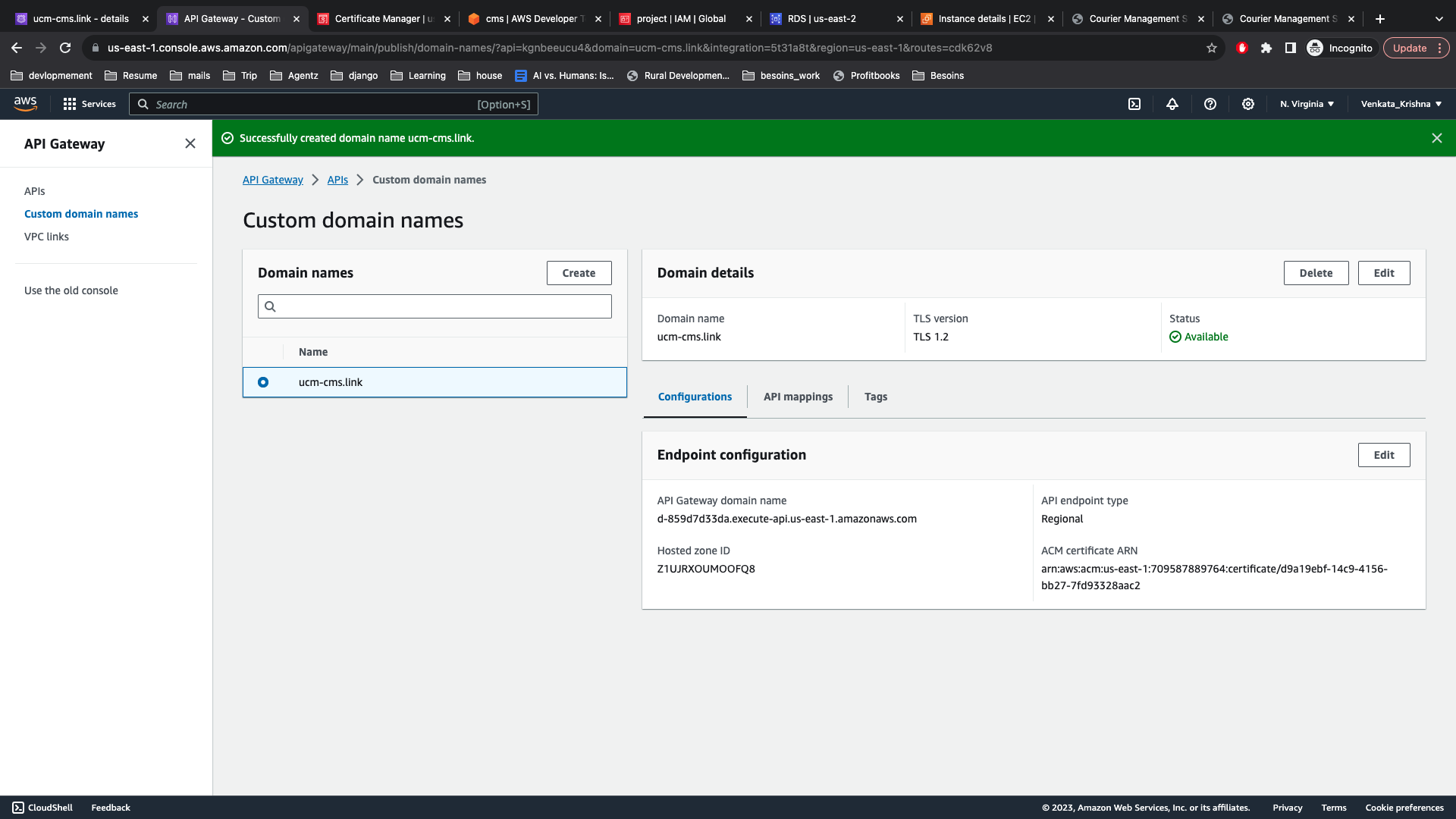
Custom Domain Name:

Configure a custom domain name in API Gateway.

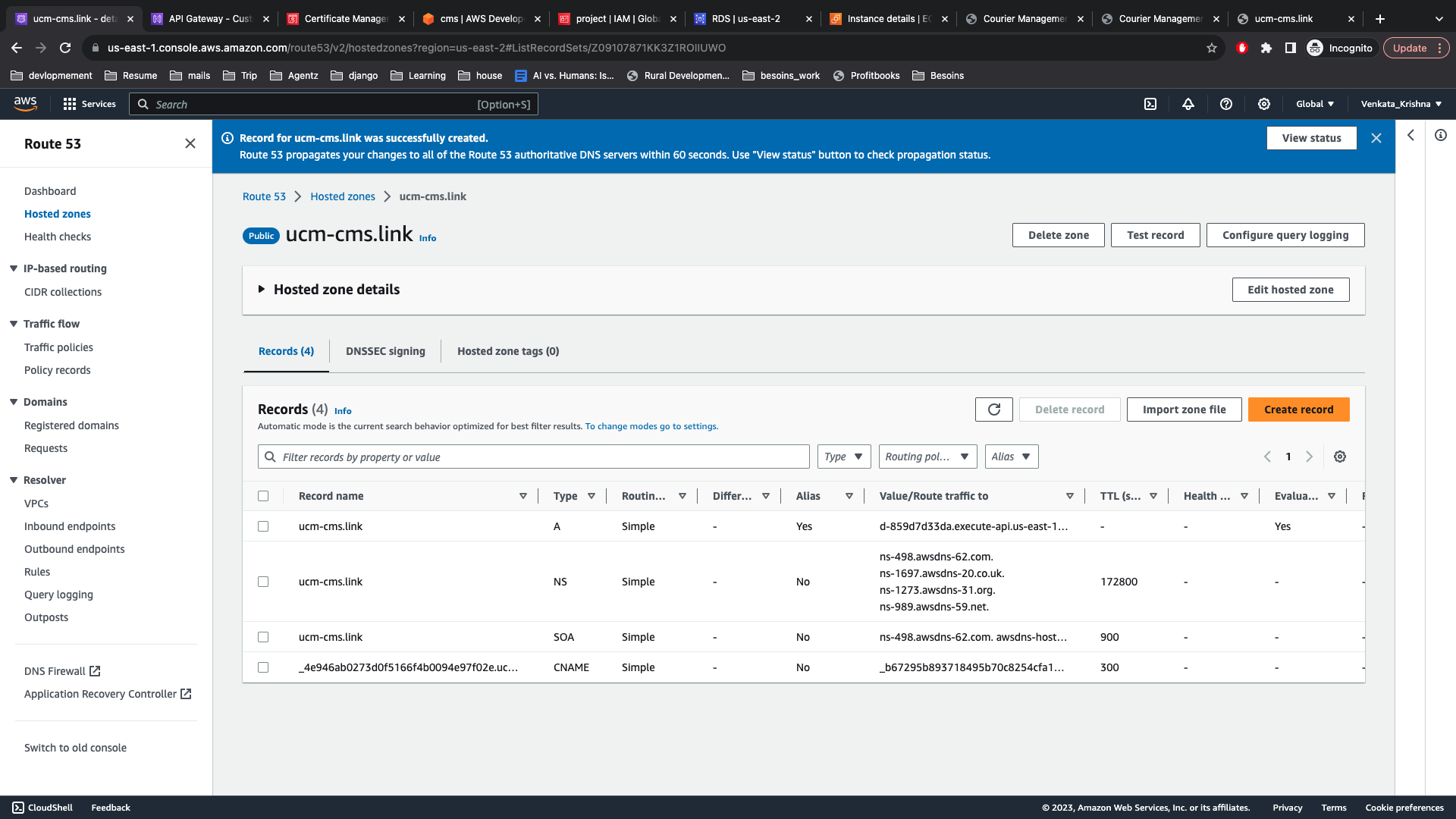
Select the ACM certificate for SSL termination.

Map API Gateway To EC2 Map this API Gateway to your EC2 public IP by creating a new Integration in API Gateway. Set the integration type to HTTP Proxy and provide your EC2 public IP.

Custom Domain Mapping For API In API Gateway, create a custom domain mapping for your API. Use the domain name registered in Route53 and the ACM certificate for SSL.



Create Record in Route 53 Finally, create a new record in Route53 for your domain and map it to the API Gateway's alias. This ensures that when someone types your domain name in the browser, it will be redirected to your application hosted on the EC2 instance.



ADD APPLICATION SCREENSHOT FROM YOUR SYSTEM

**Conclusion**

In conclusion, the development and deployment of the Online Bookstore Management System represent a significant achievement in leveraging modern software development practices to provide a streamlined and efficient solution for book enthusiasts. The system's architecture, built on Django and deployed on AWS, showcases a robust and scalable design, ensuring a seamless user experience.

The key features of the Online Bookstore Management System include a comprehensive admin dashboard, allowing administrators to manage books, categories, orders, and user accounts efficiently. The secure login and logout system, along with order management capabilities, enhances the system's security and user interaction. On the frontend, customers benefit from a user-friendly interface with features such as login, registration, book browsing, cart management, order tracking, and diverse payment options.

The database structure, including entities like Admin, Category, Writer, Book, and Order, facilitates organized data management. The relationships among these entities, as visualized in the Entity-Relationship (ER) diagram, contribute to a normalized and efficient relational database.The deployment process on AWS involves creating EC2 and RDS instances, setting up CodeCommit for version control, registering a domain with Route 53, obtaining an SSL certificate through ACM, and configuring API Gateway for secure and scalable access.