A report on

# Placement Bot: A Conversational AI for Placement Information

Group 13 - Mini Project

By,


| ID | Name |
|---|---|
| 2024H1030053H | Krish Shah |
| 2024H1030047H | Sourav Solanki |
| 2024H1030050H | Vedanshee Dave |
| 2020B4A71567H | T.Sai Sathwik |

**Guided By: Prof. Subrakanta Panda,**
**CSIS Department, BITS Pilani Hyderabad Campus**

# Introduction

**Problem Statement:** The placement process can often be cumbersome for students, as they need to constantly search for and access real-time information regarding eligibility criteria, schedules, and interview processes. The lack of a centralized system can lead to confusion and delays, making it difficult for students to stay updated on placement-related information in a timely manner.

**Objective**:This project aims to create a **Placement Bot** powered by AWS technologies. The goal is to help students easily retrieve placement-related information, including **eligibility criteria**, **company visit schedules**, and **interview rounds**. The bot uses **AWS Lex** for natural language understanding and multi-language support, **AWS Lambda** for processing backend logic, and **DynamoDB** for storing real-time data. Moreover, **AWS Cognito** ensures secure access by authenticating users with valid university email addresses.

**Technologies Used**:

- **AWS Lex** for building the conversational bot interface.
- **AWS Lambda** for dynamic responses and backend logic.
- **Amazon DynamoDB** for persistent storage of placement data.
- **AWS Cognito** for secure user authentication.
- **AWS S3 and CloudFront** for hosting the bot frontend.
- **AWS CloudWatch** for monitoring and logging.

By using serverless services such as **AWS Lambda** and **DynamoDB**, the bot can dynamically scale based on the traffic load, ensuring smooth performance during high-demand periods. The scope also includes robust monitoring and logging using **AWS CloudWatch**, providing valuable insights into the bot's interactions and helping to optimize its performance over time. The bot can be accessed via text or voice and can handle multiple languages, starting with **English** and **Spanish**, to cater to a wider audience.

Architecture

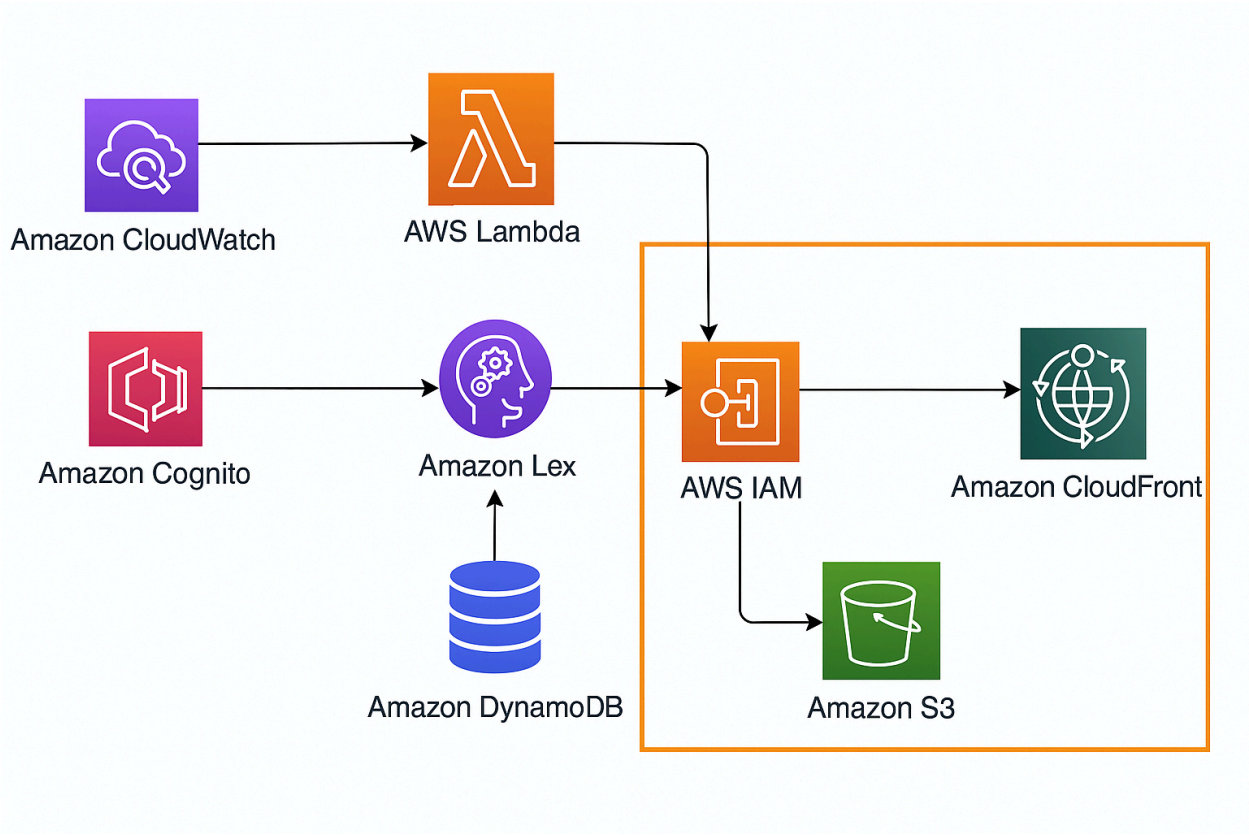


**Figure: 0 Architecture**

The architecture of the Placement Bot utilizes various AWS services that work together to provide a robust, scalable, and secure platform for delivering placement-related information to users. The system leverages a combination of AWS Lex, Lambda, DynamoDB, Cognito, CloudWatch, IAM, S3, and CloudFront to implement both the frontend and backend services, enabling voice and text-based interactions.

The components interact with each other in the following way:

1. Amazon Lex (Chatbot Engine):

Amazon Lex serves as the primary conversational interface for the Placement Bot. It is responsible for processing user inputs, recognizing intents, and triggering responses. Lex handles both text and voice interactions, which are crucial for understanding user queries related to placement details such as

company names, CGPA requirements, rounds, etc.
Functionality:

- Lex uses intents to understand user input (e.g., queries about placement companies, eligibility criteria).
- It processes slot values to gather necessary details from the user.
- Integrates with AWS Lambda to provide dynamic, context-aware responses to users.

2. AWS Lambda (Backend Logic):

AWS Lambda acts as the backend processor for the Placement Bot. When Lex receives a user's input and determines the appropriate intent, it invokes a corresponding Lambda function to process the data, fetch information, or interact with other AWS services.
Functionality:

- Executes custom code to process user input, fetch relevant placement data from DynamoDB, and generate responses.
- Lambda interacts with DynamoDB to store and retrieve data such as company names, CGPA, and rounds.
- It also handles integration with third-party APIs if required, to pull real-time data like placement schedules.

3. Amazon DynamoDB (Database Storage):

DynamoDB is used to store all the placement-related information, including company details, eligibility criteria, rounds, and visit dates. This NoSQL database service ensures low-latency access to data and scalability.
Functionality:

- Stores data related to placement companies, which Lex and Lambda access to provide information to the users.

- Provides a scalable solution to manage growing amounts of placement-related data without compromising performance.

4. Amazon Cognito (User Authentication):

Amazon Cognito is used to manage user authentication. It ensures that only authorized users can access the Placement Bot by verifying user credentials. This is particularly important for maintaining secure access control and ensuring that placement data is provided only to students of the correct institution.

Functionality:

- Handles user registration and login, supporting multi-factor authentication.
- Restricts access based on user email domain (e.g., students with @hyderabad.bits-pilani.ac.in emails).

5. AWS Identity and Access Management (IAM):

IAM provides secure access management across AWS services. It allows for the creation of roles and policies to control permissions for AWS resources. The Placement Bot uses IAM to ensure that services like Lex, Lambda, and DynamoDB are able to interact securely and are granted only the necessary permissions to perform their tasks.

Functionality:

- Defines roles for the bot to allow or restrict access to resources like DynamoDB, Lambda, and Lex.
- Ensures that only authorized entities have access to sensitive data and functionality.

6. Amazon S3 (Frontend Hosting):

Amazon S3 is used to host the static frontend files of the Placement Bot, including HTML, CSS, and JavaScript. The frontend is responsible for presenting the user interface (UI) where students can interact with the bot.
Functionality:

- Stores static web files and makes them publicly available for users to access.
- Supports the integration of S3 with CloudFront for fast content delivery across different geographical regions.

7. Amazon CloudFront (Content Delivery Network - CDN):

CloudFront is a Content Delivery Network (CDN) service that caches static content from the S3 bucket and serves it to users globally with low latency. It improves the performance of the Placement Bot frontend, ensuring that users can access the bot's interface quickly regardless of their location.
Functionality:

- Distributes the static content stored in S3 to users globally, reducing load times.
- Uses edge locations to improve the speed and reliability of content delivery.

8. Amazon CloudWatch (Monitoring and Logging):

Amazon CloudWatch is used for monitoring and logging the various services involved in the Placement Bot’s operation. It collects metrics from Lex, Lambda, and other AWS services, allowing developers to track system performance, identify potential issues, and optimize service usage.
Functionality:

- Monitors Lambda invocations, error rates, and performance metrics.
- Tracks Lex interaction logs to identify common errors or failed interactions.
- Provides visibility into how users interact with the bot, helping to improve its accuracy and efficiency.

Work Flow:

1. User Interaction:
    - The user interacts with the Placement Bot through the frontend, which is hosted on Amazon S3 and distributed via CloudFront for optimal performance. The user can ask placement-related queries via text or voice.

2. Lex Processing:
    - Amazon Lex receives the user's query, processes it to understand the intent (e.g., query about a specific company's eligibility or placement rounds), and triggers the corresponding AWS Lambda function.

3. Lambda Execution:
    - The Lambda function processes the request, fetching data from DynamoDB if needed (e.g., details about a company or its rounds) and constructs a dynamic response. If the request is more complex, Lambda can also query external APIs for additional information.

4. Response:
    - Once the Lambda function processes the request, it sends the appropriate response back to Lex, which then delivers it to the user via the frontend.

5. Authentication & Access Control:

- Amazon Cognito manages user authentication, ensuring that only authorized users (students with a specific email domain) can access the Placement Bot. This helps restrict access and ensures secure usage of the bot.

6. Monitoring & Optimization:
    - Amazon CloudWatch tracks and monitors all interactions between the frontend, Lex, Lambda, and DynamoDB. It helps in identifying any issues or bottlenecks in the system and optimizing performance. CloudWatch also logs interactions, providing insights into bot usage patterns and user behavior.

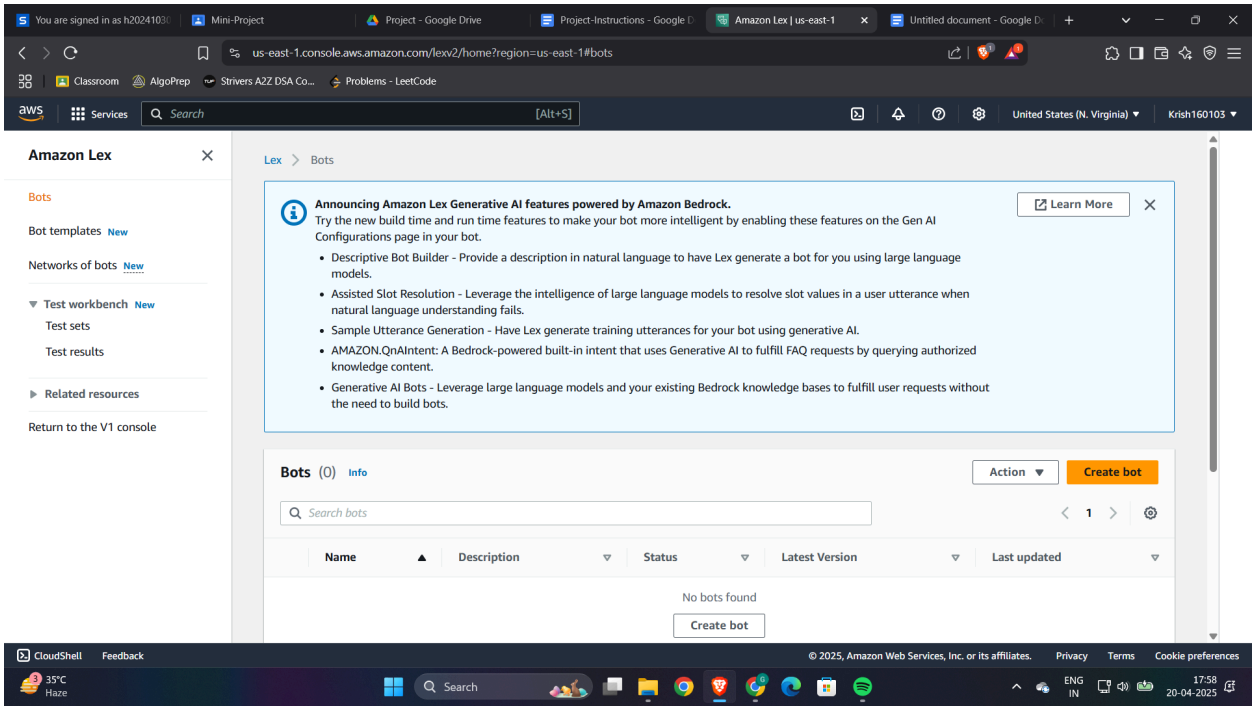


Figure: 1 Amazon Lex Console - Introduction Screen

Figure 1 illustrates the Amazon Lex Console where we begin the process of creating a new bot. The screen introduces the AI-powered capabilities of Lex, particularly the multi-language support and intent classification features. These features are essential for building a multi-lingual Placement Bot that can understand and respond to users in different languages, enhancing the user experience. The console offers easy navigation for creating and managing Lex bots, enabling the integration of machine learning models to assist in natural language understanding and intent recognition.

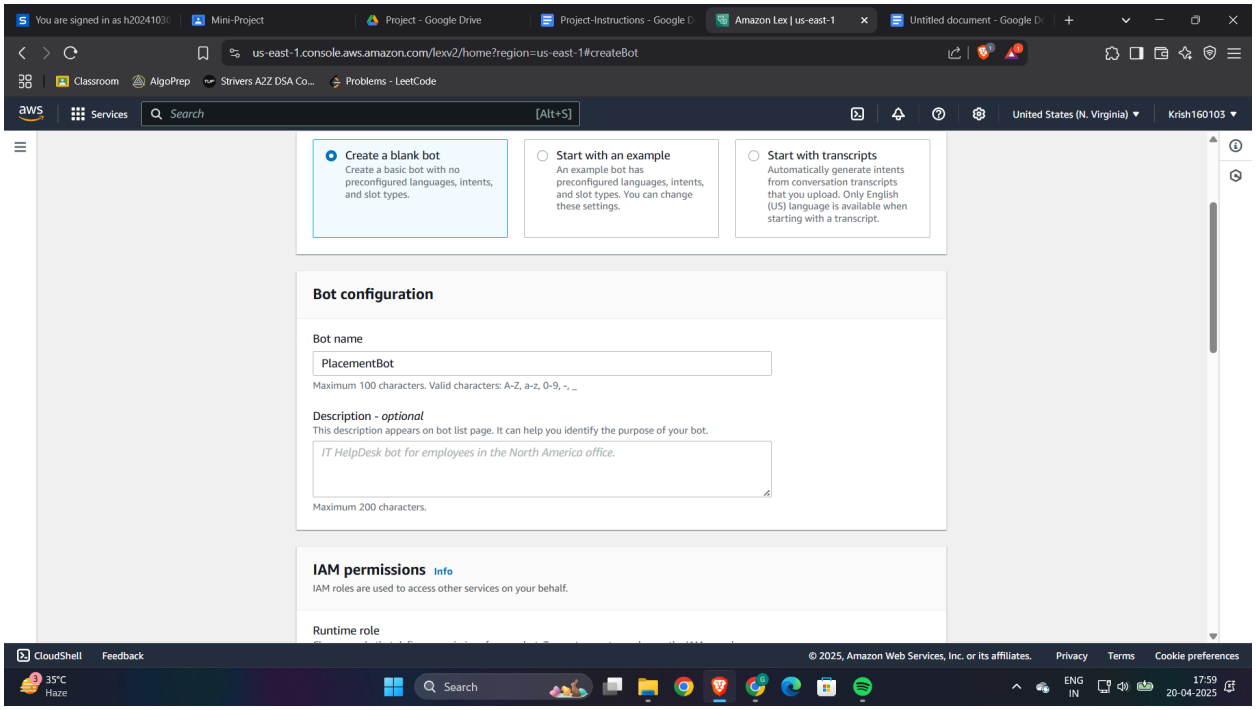


Figure: 2 Bot Configuration - Placement Bot Creation

Figure 2 displays the bot configuration screen where the PlacementBot is created. Here, the bot is named “PlacementBot” and the description is set to provide placement assistance to students. During this phase, important attributes such as the bot’s name and IAM permissions are configured to define the bot's role and access rights. Proper configuration of these settings is crucial to ensure that the bot can access other AWS services, such as Lambda and DynamoDB, for dynamic responses and data storage.

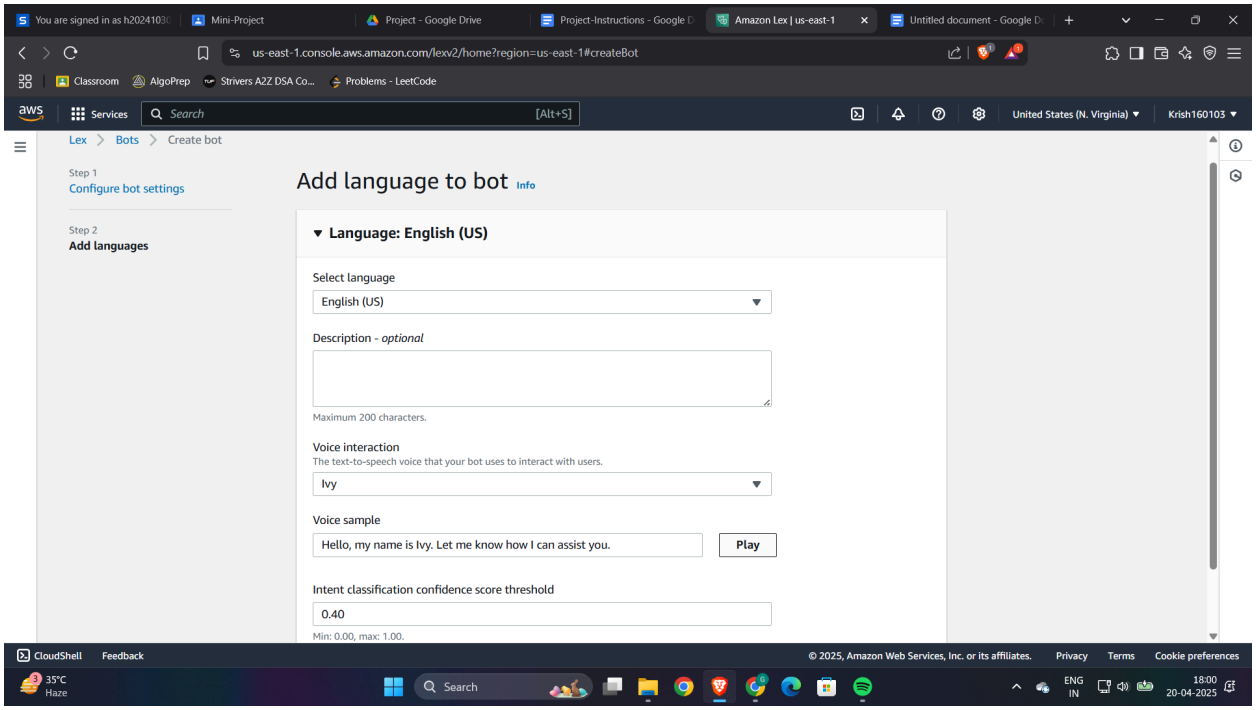


Figure: 3 Adding Language to Bot

In Figure 3, the bot's language configuration is being set up for English (US). AWS Lex allows the creation of bots that support multiple languages, enabling users to interact with the bot in different languages. For the PlacementBot, English (US) is chosen as the first language. The welcome message for the bot is also set, which greets users in English: "Hello, how may I assist you? Let me know how I can help." This process is essential for building the language model of the bot, which Lex uses to process user inputs in the specified language.

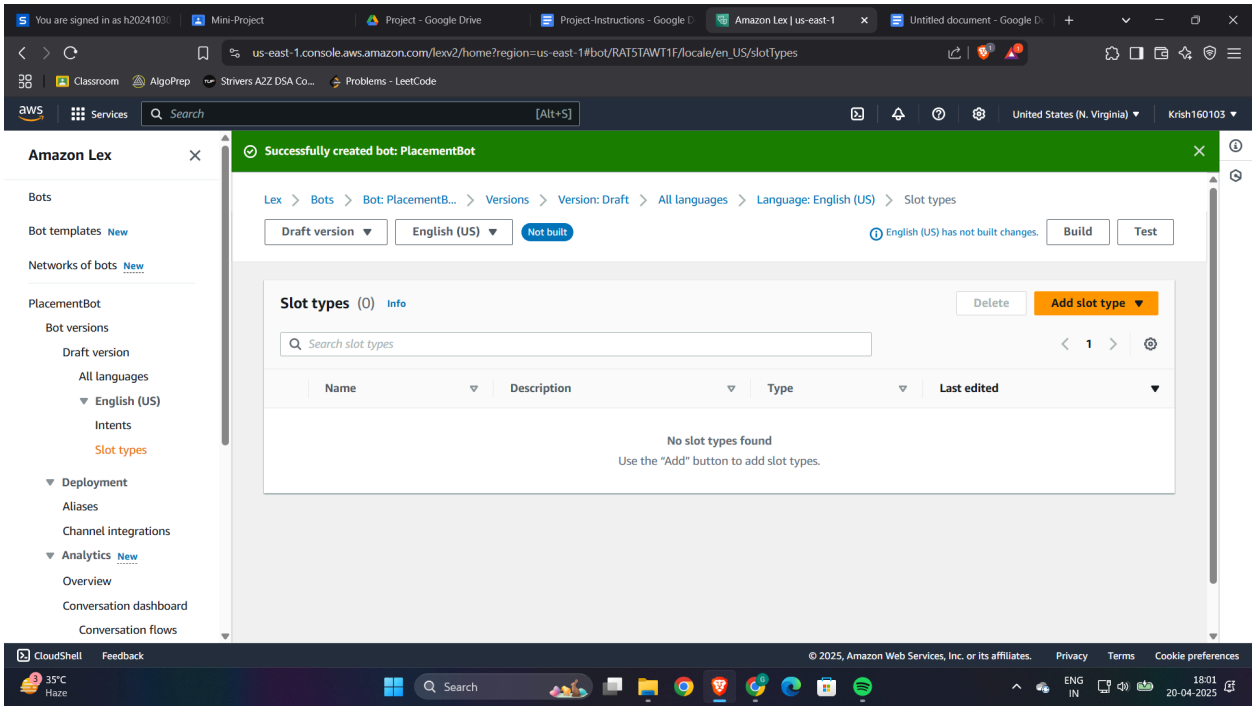


Figure: 4 Configuring Slot Types

This image shows the configuration page where **Slot Types** are defined for the bot. It is used to customize the slots that the bot will collect from users, such as company names for eligibility queries.

It shows the Slot Types Configuration screen within the Amazon Lex Console. **Slot types** are used to define the parameters or entities that the bot will collect from the user during conversations. In this case, the **company names** (Google, Amazon, etc.) are added as **custom slot values**. These values will be used to help the bot understand and extract the required information (like company eligibility) from the user queries. By defining the slot types here, we ensure that the bot can handle specific and structured data inputs efficiently.

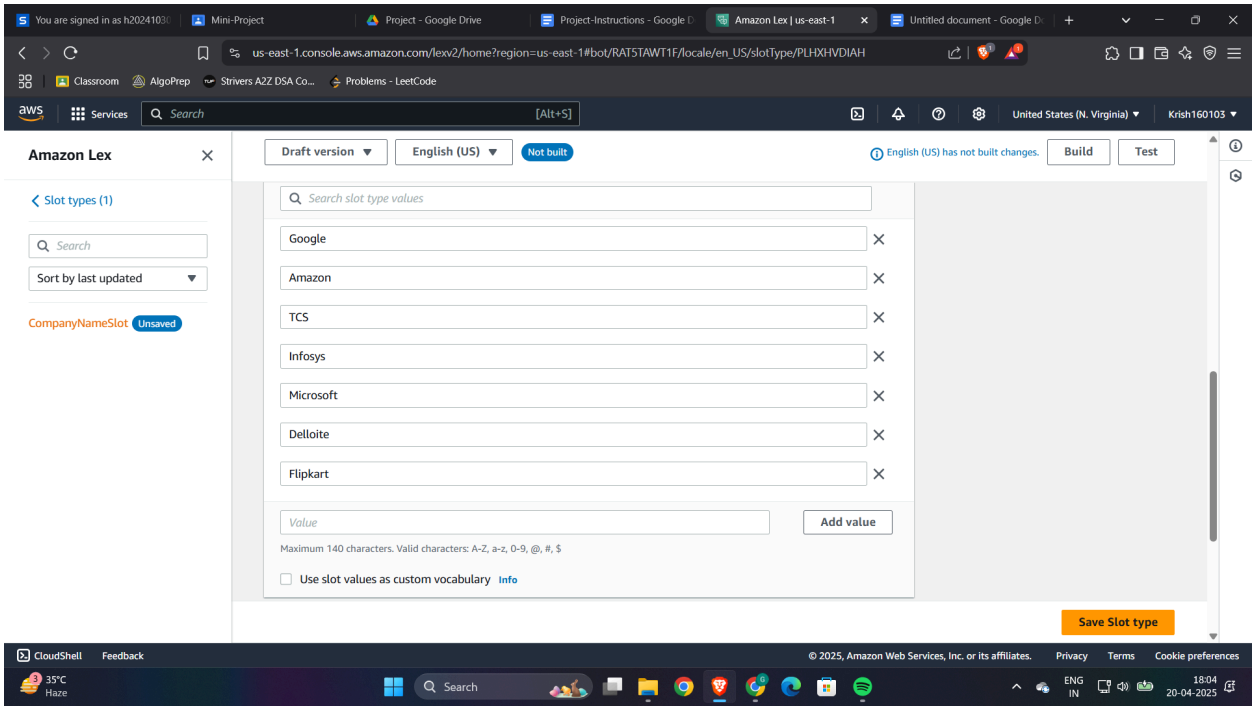


Figure: 5 Defining Slot values

This screenshot illustrates how specific company names (such as Google, Amazon, Microsoft, etc.) are added as values in the company slot type.

In Figure 5, slot values for the company name slot type are being defined. By adding companies like Google, Amazon, Microsoft, and others, the bot can recognize and process these as valid inputs during user interactions. When a student asks about eligibility or schedule for a particular company, the bot matches the input to one of these predefined values. This is part of the bot's entity recognition, where it processes structured data based on the user's input and provides dynamic, data-driven responses by interacting with services like Lambda and DynamoDB.

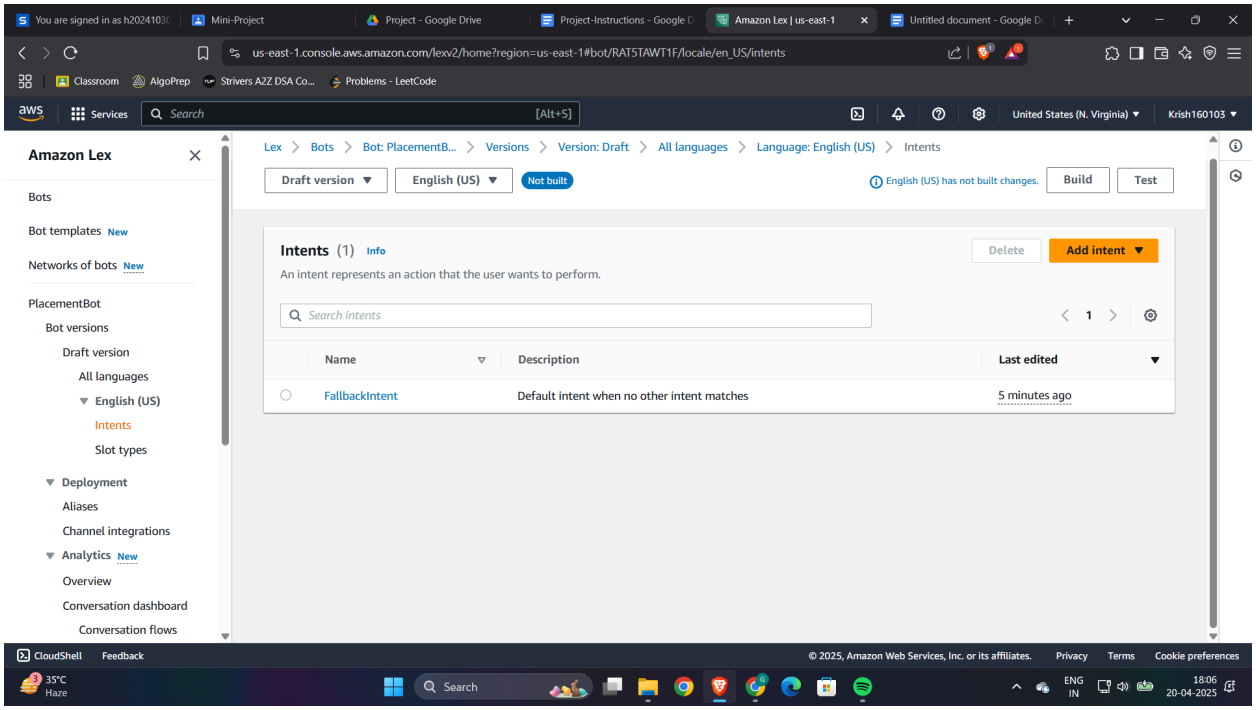


Figure: 6 Creating a New Latent

This image displays the creation of a new intent in Amazon Lex, where the bot is set to respond to specific queries related to company eligibility.

Intents in Amazon Lex define the actions that a bot should perform in response to user input. In this case, a new intent for "CompanyEligibilityIntent" is being created. This intent will allow the bot to answer questions like "What is the eligibility for Google?" or "What CGPA is required for Amazon?" By configuring this intent, the bot can recognize these types of queries and respond with the relevant data stored in a DynamoDB table. Intents are the core functionality of a chatbot, as they allow the bot to understand user input and trigger appropriate responses.

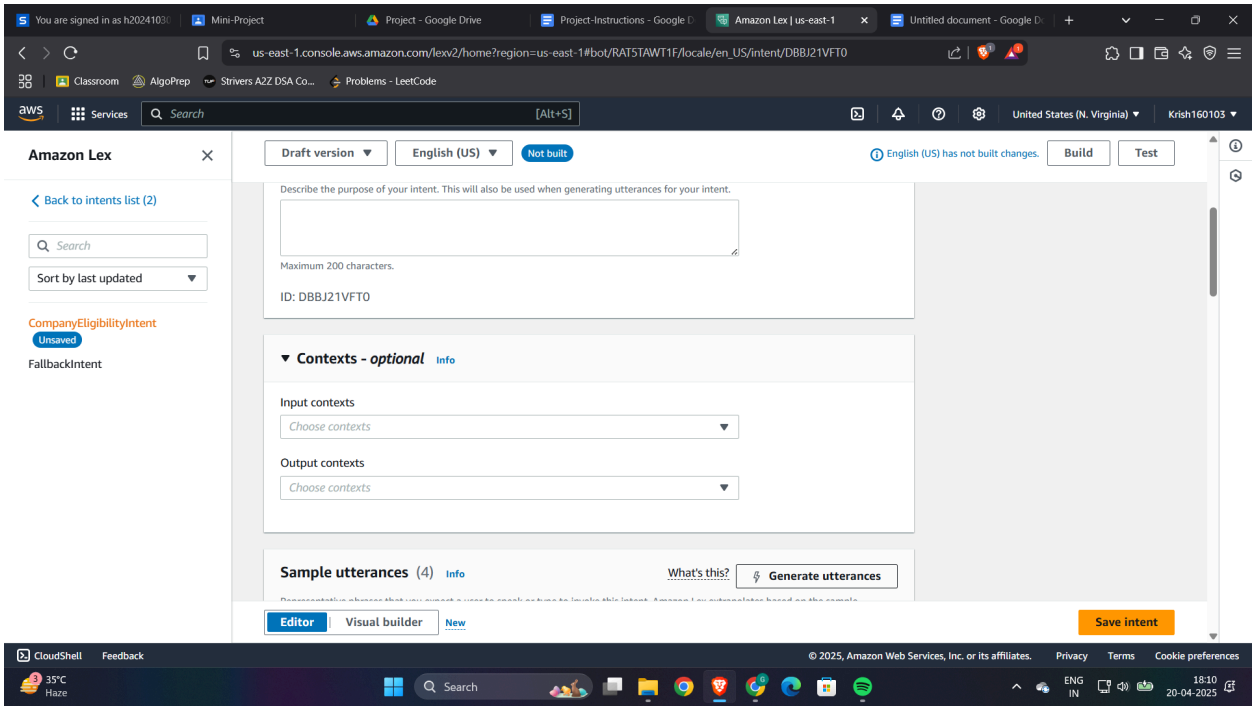


Figure: 7 Intent Configuration

This image shows the configuration of the CompanyEligibilityIntent in the Lex Console, where the input and output contents are set for the intent.

Figure 7 displays the configuration of the CompanyEligibilityIntent in Amazon Lex. For each intent, we define how Lex processes input contents (user queries) and output contents (bot responses). In this example, the CompanyEligibilityIntent is set up to process user queries related to company eligibility for placement, such as “What is the eligibility for Google?” or “What is the CGPA required for Amazon?” The Sample Utterances section contains sample phrases that users might say, which helps Lex to recognize user inputs and trigger the appropriate intent.

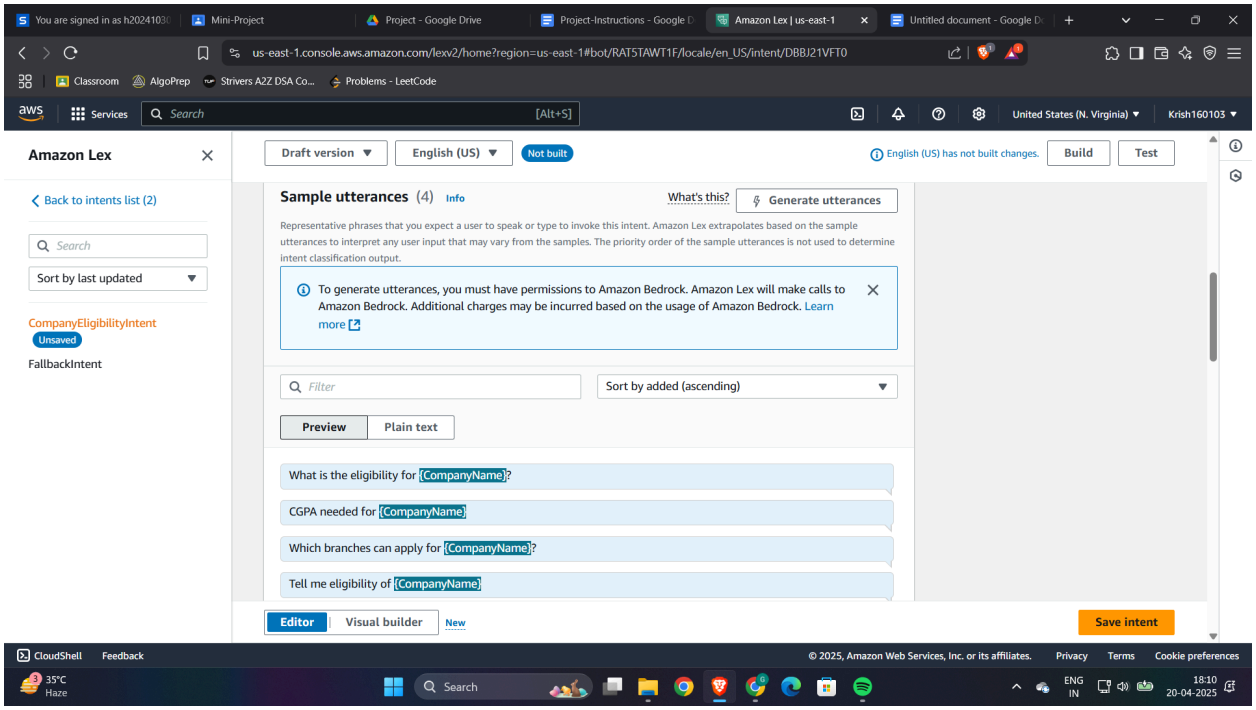


Figure: 8 Adding Sample Utterances for the CompanyEligibilityIntent

This screenshot shows how sample utterances for the CompanyEligibilityIntent are added. The user can inquire about eligibility, CGPA requirements, and branch eligibility for specific companies.

It showcases the sample utterances added to the CompanyEligibilityIntent in the Amazon Lex Console. Sample utterances are predefined phrases that Lex uses to understand user input. For example, the user may ask, "What is the eligibility for Google?" or "Which branches can apply for Google?" By providing multiple sample utterances, the bot can handle a wide variety of similar queries. This is an important aspect of intent recognition, ensuring that the bot understands the user's question and responds accordingly.

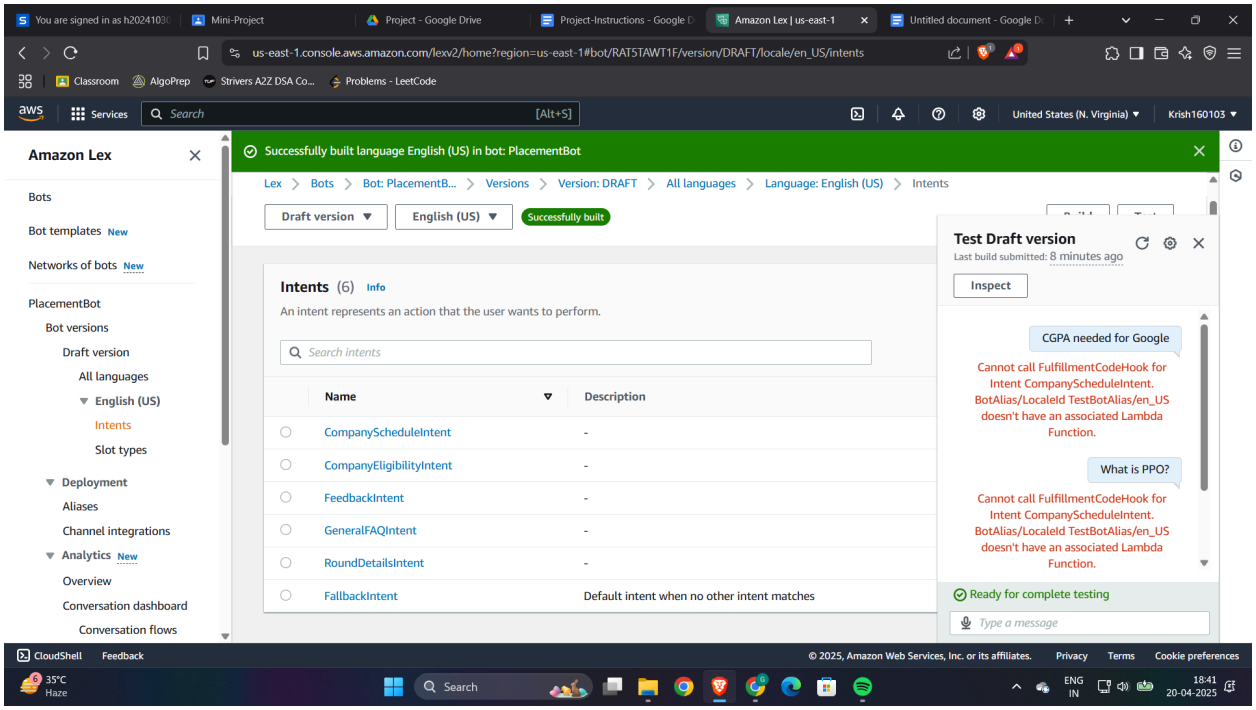


Figure: 9 Successfully Created Intents for Placements

Figure 9 indicates that the PlacementBot has successfully been created, with multiple intents like CompanyEligibilityIntent now ready to handle user queries. After defining intents and utterances, we proceed to test the bot to ensure it responds accurately to various user inputs. The testing console allows us to simulate conversations with the bot, verifying that the intents trigger the correct responses. This process is essential for validating the bot's functionality before deploying it in a real-world scenario.

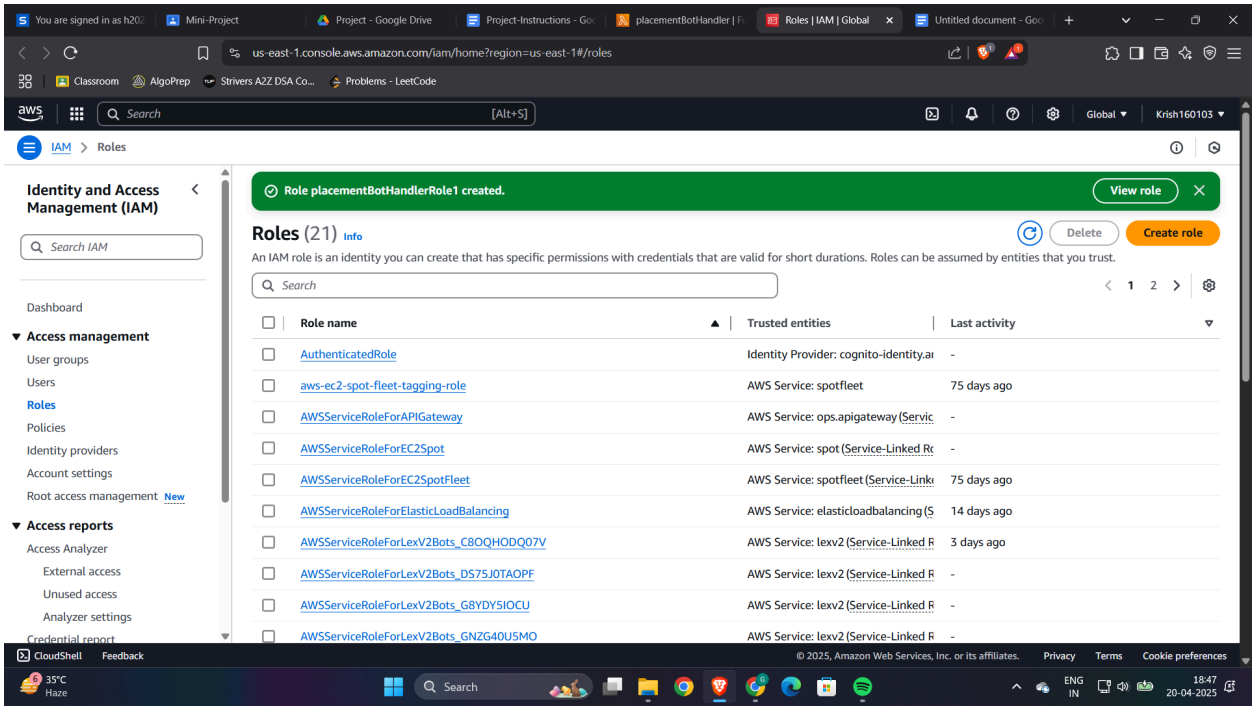


Figure: 10 IAM Roles for Lambda Functions

Figure 10 displays the IAM roles configured for the Lambda functions associated with the PlacementBot. IAM roles define the permissions for AWS services, such as Lambda, to interact with other AWS resources (e.g., DynamoDB, CloudWatch, etc.). In this case, we ensure that the Lambda functions have the appropriate permissions to query the placement data from DynamoDB and log interactions to CloudWatch. Proper IAM role configuration is critical for security and functionality, ensuring that only authorized resources can access sensitive data.

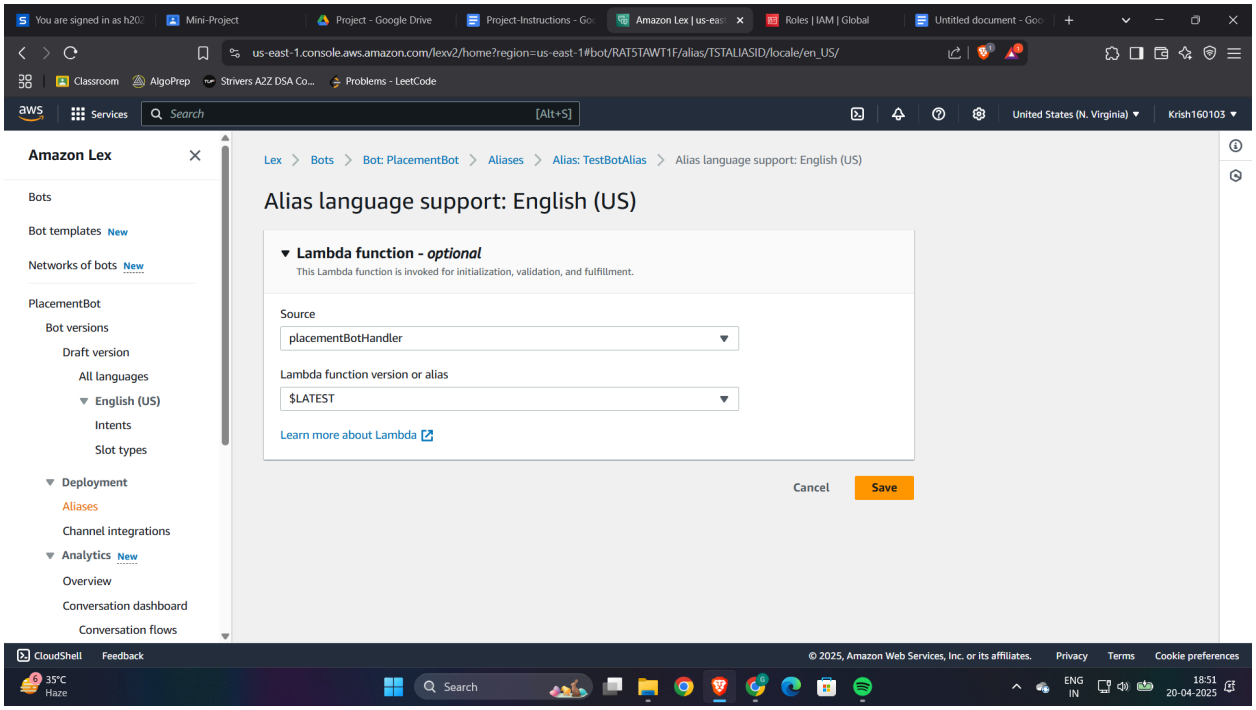


Figure: 11 Setting Lambda Function for Lex Intent

Figure 11 illustrates the configuration of Lambda functions for the PlacementBot. In this step, we link the Lambda function to the Lex intents. AWS Lex invokes Lambda functions to process the bot's logic, such as querying DynamoDB for placement data or handling user-specific logic. The Lambda function provides dynamic responses based on the intents triggered by user inputs. This integration is key to enabling the real-time data retrieval and dynamic responses that make the bot interactive and useful for students.

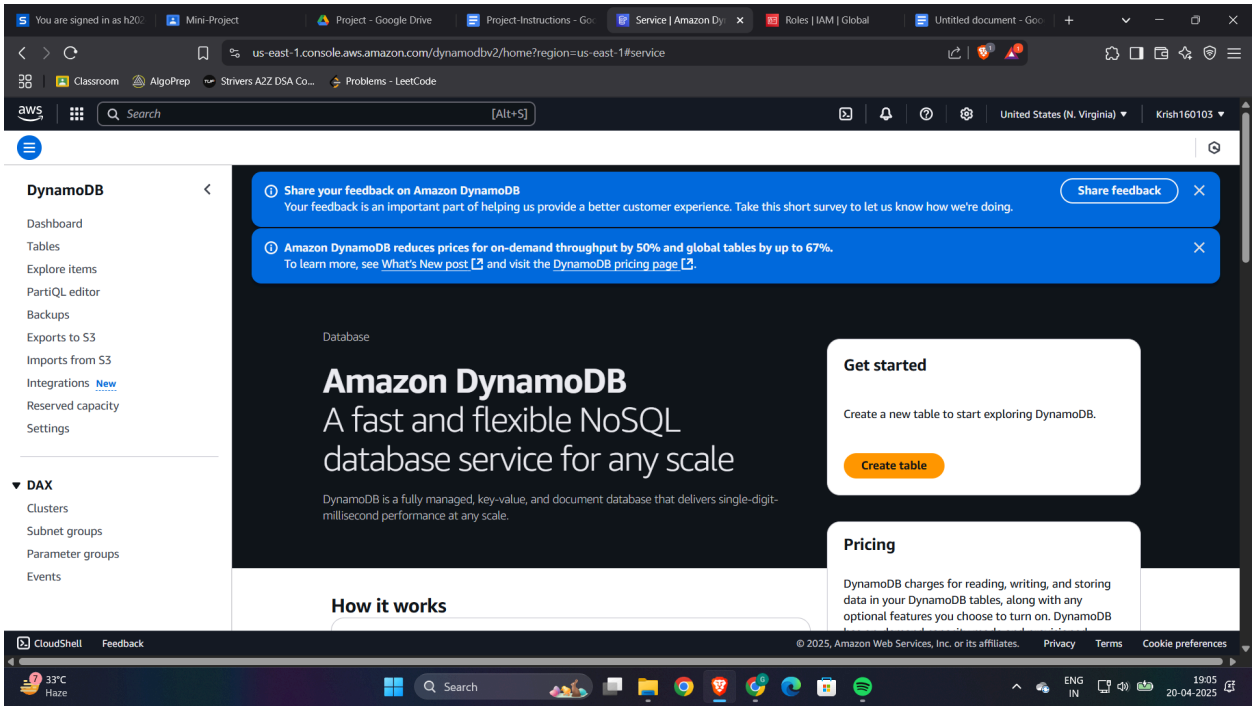


Figure: 12 Amazon DynamoDB Overview Page

Figure 12 shows the **Amazon DynamoDB** console, which is used to manage the placement-related data stored for the **PlacementBot**. DynamoDB is a **NoSQL database** that provides fast and scalable data storage for applications that require low-latency responses, like the **PlacementBot**. In this case, DynamoDB is used to store and retrieve placement data such as company eligibility, interview schedules, and selection rounds. This data is queried by Lambda functions when the user requests specific placement information.

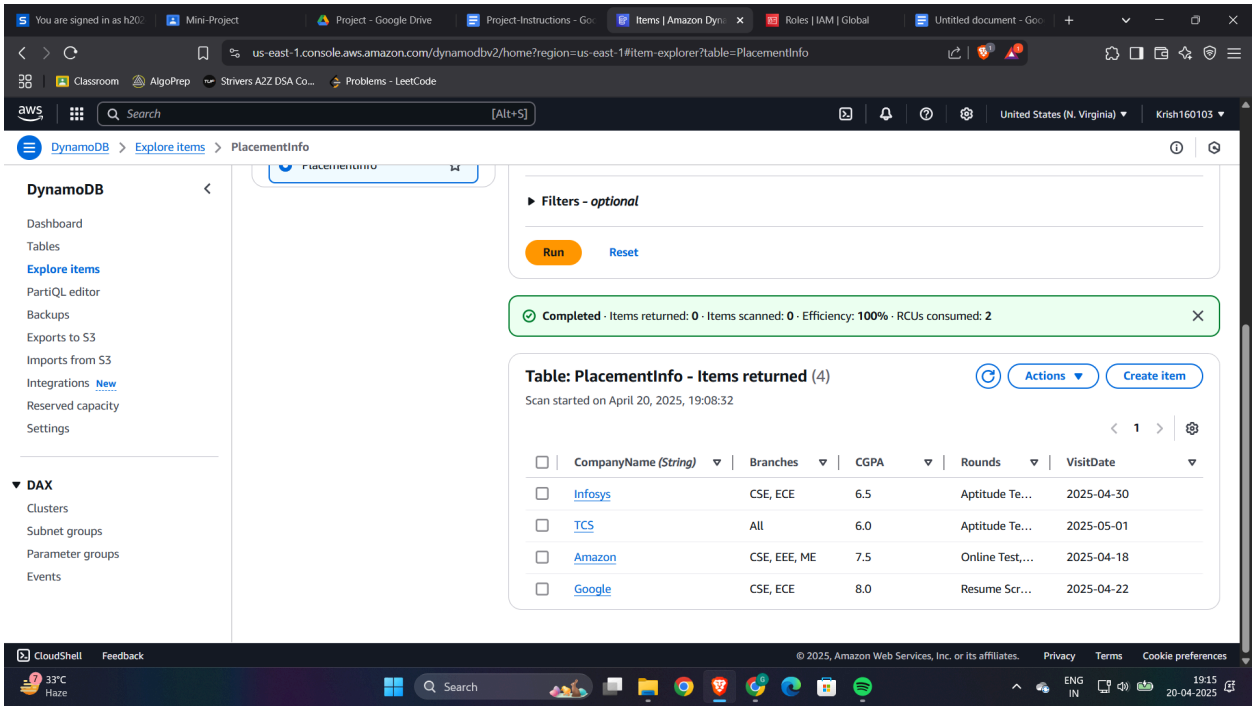


Figure: 13 DynamoDB Table - Placement Info Data

Figure 13 illustrates the **PlacementInfo** table in **DynamoDB**, which contains crucial placement data used by the **PlacementBot**. The table stores information such as company names (Google, Amazon, etc.), branches eligible for placement, CGPA requirements, and scheduled visit dates. When a user interacts with the bot, the Lambda function queries this table to fetch the relevant data and provide an accurate response. This data-driven approach ensures that the bot provides students with **real-time** and **personalized information** about the companies visiting the campus for placements.

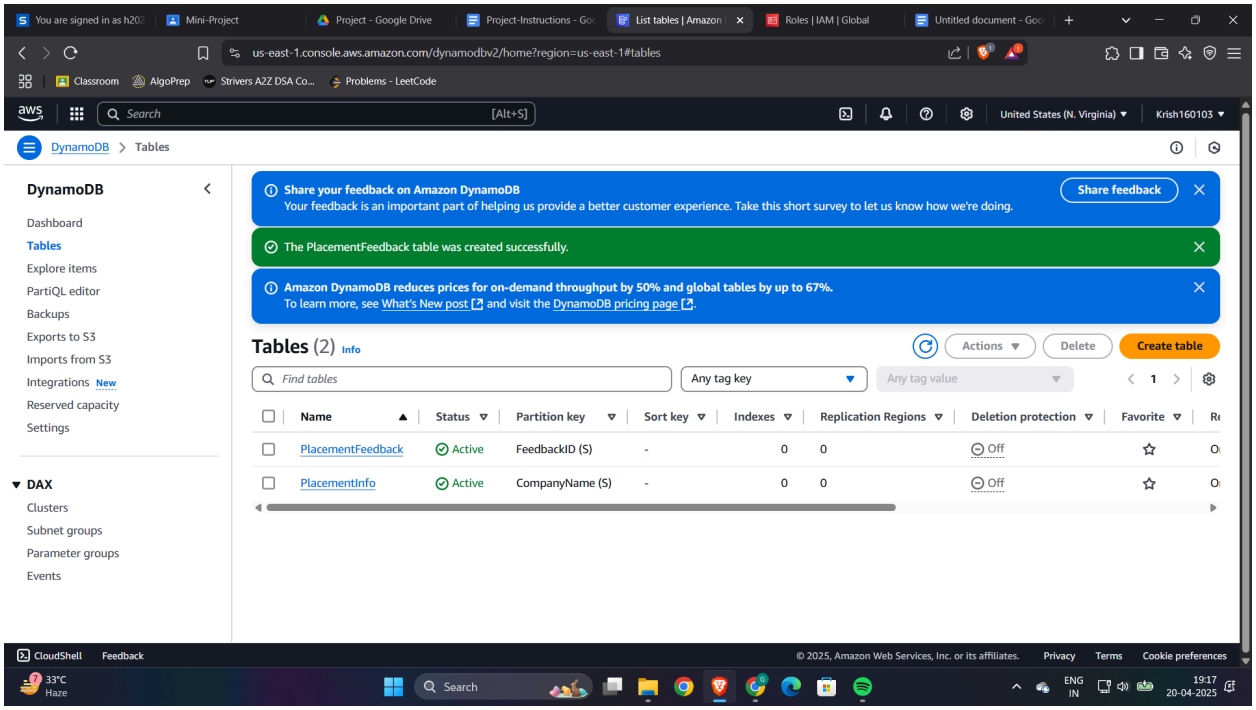


Figure: 14 DynamoDB Table Creation for Placement Information

In this figure, we see the creation of a DynamoDB table titled **PlacementInfo** to store critical data related to various companies participating in the placement process. The table contains entries such as company names, CGPA requirements, eligible branches, rounds, and visit dates. Using DynamoDB for storing this information provides a fast, scalable NoSQL database that ensures low-latency access to placement data for the bot. This integration enables the bot to retrieve relevant placement information dynamically based on user queries, ensuring real-time responses. With DynamoDB, the bot can store and access large amounts of data efficiently, ensuring high availability and fault tolerance.

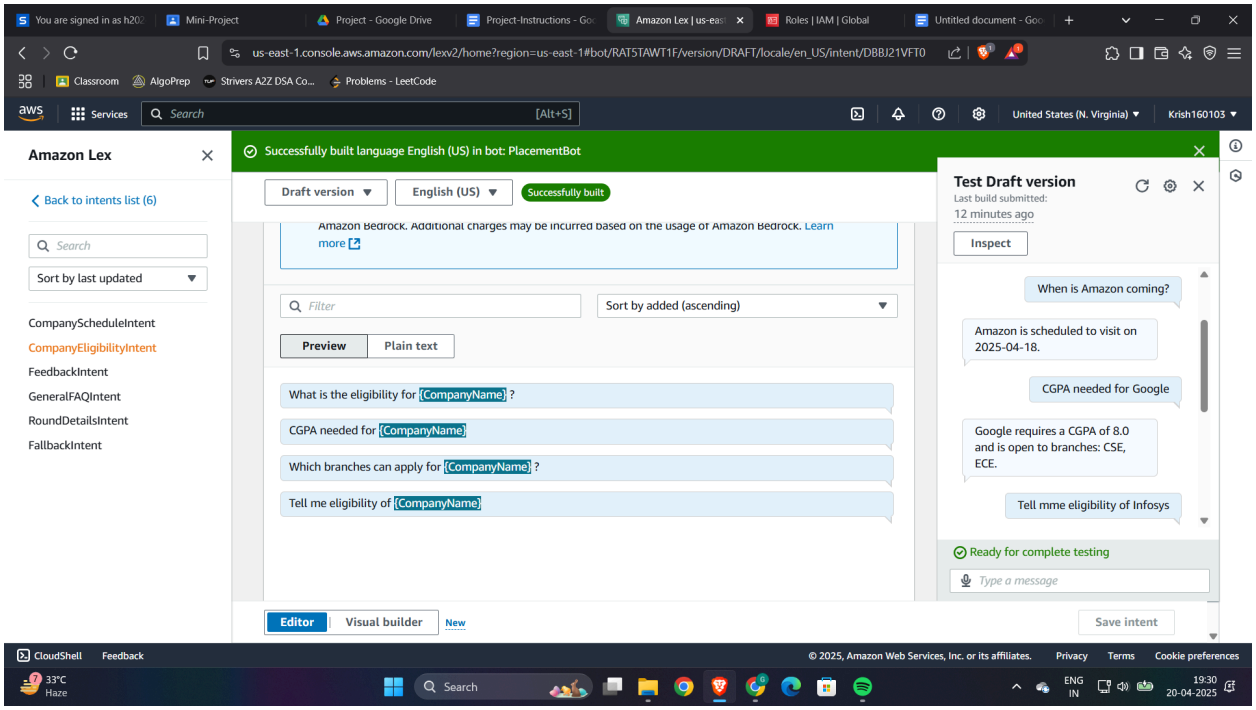


Figure: 15

This screenshot shows the view of data stored in the DynamoDB PlacementInfo table, where company-specific placement details are stored for easy retrieval.

Once the DynamoDB table is created, this figure shows how the data can be viewed within the AWS console. Each row in the PlacementInfo table corresponds to a different company, with columns such as "CompanyName," "CGPA," "Branches," and "Rounds" holding relevant placement information. The Placement Bot accesses this table to provide real-time information about companies, such as eligibility criteria, visit dates, and rounds. This simple and efficient data storage structure allows the bot to fetch and deliver personalized, up-to-date placement details with minimal latency, improving the user experience.

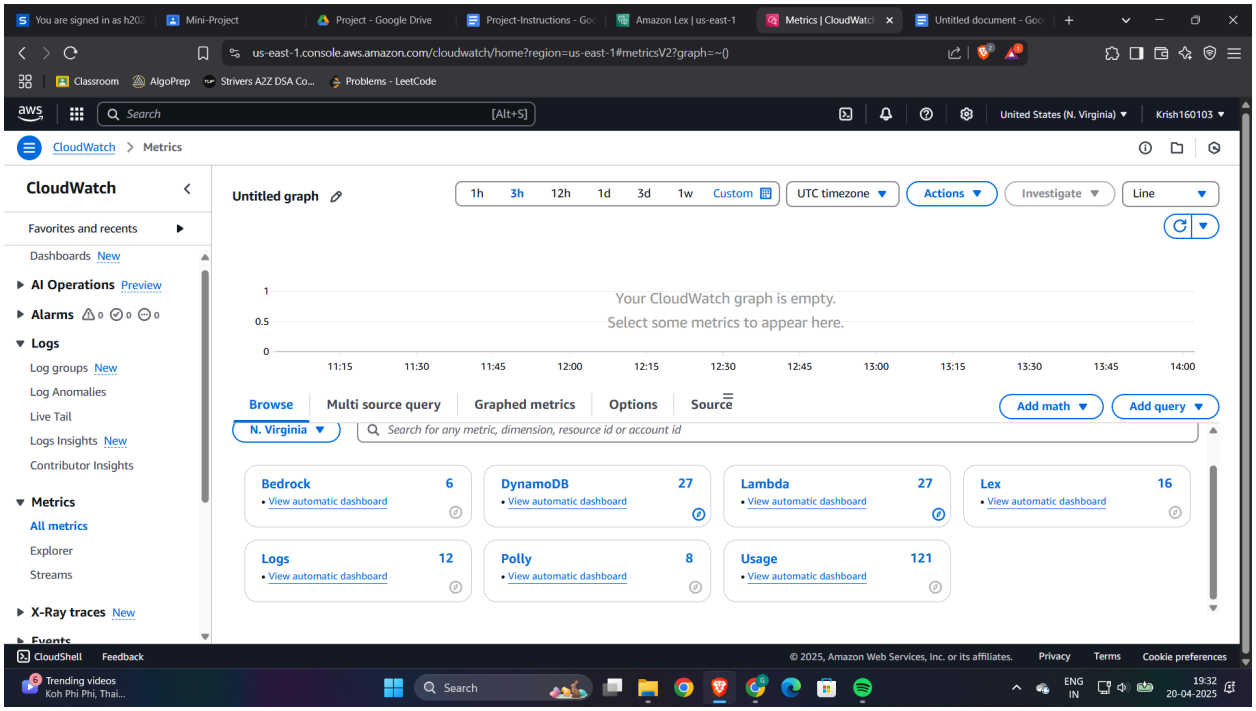


Figure: 16

The screenshot shows the CloudWatch metrics dashboard, which is used to monitor and visualize the performance of various AWS services like Lambda, DynamoDB, and Lex.

Amazon CloudWatch plays a crucial role in monitoring and optimizing the Placement Bot. In this image, we can see CloudWatch being used to monitor the performance of various AWS services, including DynamoDB, Lambda, and Lex. CloudWatch tracks key metrics such as usage, errors, and invocation counts. For instance, it monitors how often Lambda functions are invoked, the number of requests made to Lex, and the overall system health. By visualizing these metrics, developers can identify performance bottlenecks, troubleshoot issues, and optimize the bot's response time, ensuring that the bot remains responsive and efficient during usage.

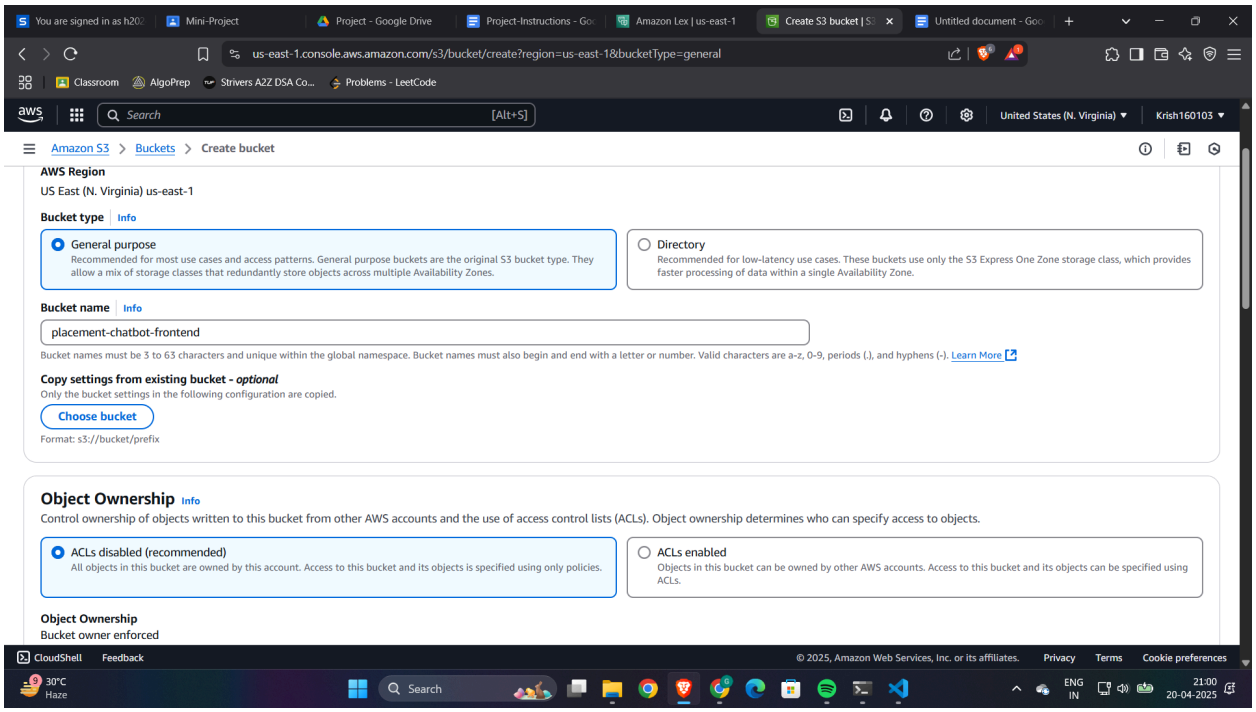


Figure: 17

This screenshot shows the creation of an S3 bucket for hosting the frontend files of the Placement Bot, including HTML, CSS, and JavaScript files.

To make the Placement Bot accessible to users, the frontend files (HTML, CSS, and JavaScript) are stored in an Amazon S3 bucket. This figure shows the process of creating a new S3 bucket titled placement-chatbot-frontend. The bucket will be used to store the static web assets that users can access through a browser. Amazon S3 is chosen for this task because of its durability, scalability, and integration with other AWS services like CloudFront, which can be used for CDN (Content Delivery Network) distribution to ensure fast access to the frontend globally. The use of S3 makes the frontend easy to deploy, manage, and access securely.

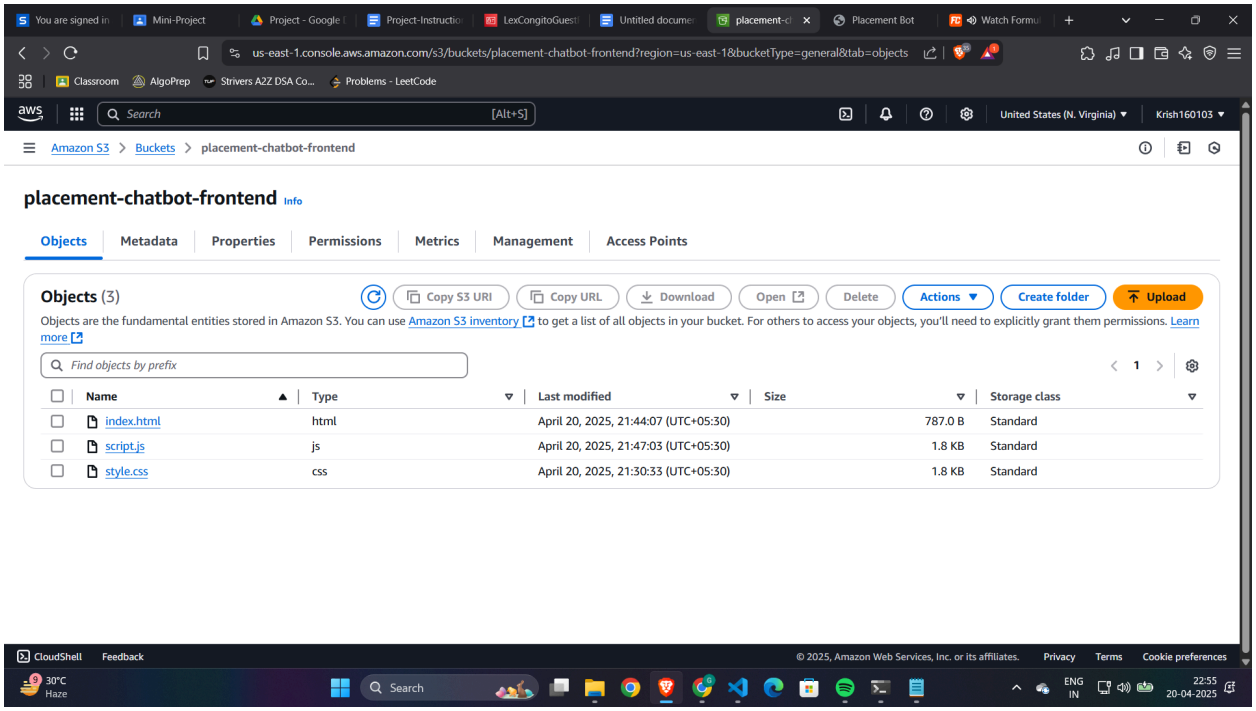


Figure: 18

In Figure 18, it shows the process of uploading the frontend files (HTML, CSS) to the S3 bucket, which will be used to serve the bot's user interface.

We can see the files being uploaded to the previously created S3 bucket. These files include the bot's HTML structure, CSS for styling, and JavaScript for bot interaction. Once uploaded, these files will be publicly accessible, allowing users to interact with the bot via a web interface. This process ensures that the bot's frontend is efficiently hosted and easily maintainable. S3 provides high availability, security, and scalability, making it an ideal choice for hosting static files. With CloudFront integrated, the content can be delivered with low latency to users worldwide.

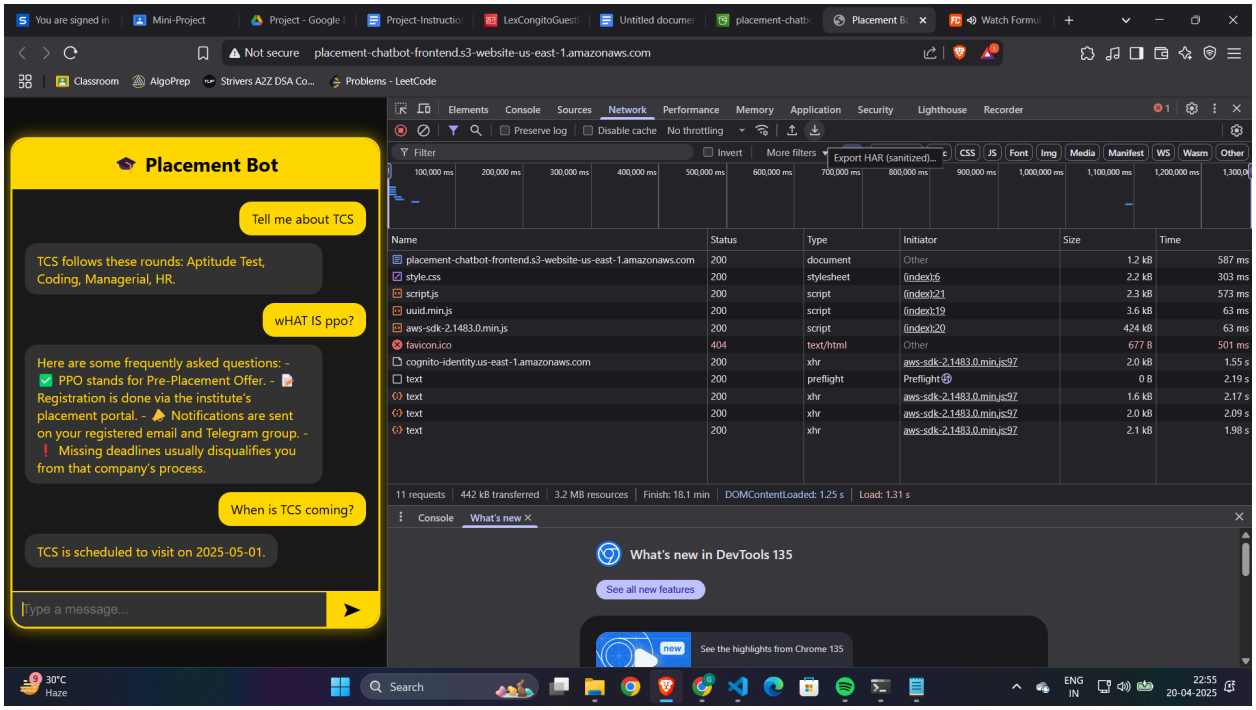


Figure: 19

This screenshot shows the Placement Bot's frontend in action, where the user interacts with the bot to ask placement-related questions.

In this figure, we can see the frontend of the Placement Bot in action. The user interacts with the bot by typing queries such as "Tell me about TCS" or "What is the eligibility for Google?" The chatbot responds in real-time with information about the placement process for various companies, providing answers such as "TCS follows these rounds: Aptitude Test, Coding, Managerial, HR" and "Google requires a CGPA of 8.0 and is open to branches: CSE, ECE." The interaction is facilitated by the bot's integration with AWS Lex, which processes the user input, and AWS Lambda, which handles the backend logic to retrieve the relevant data.

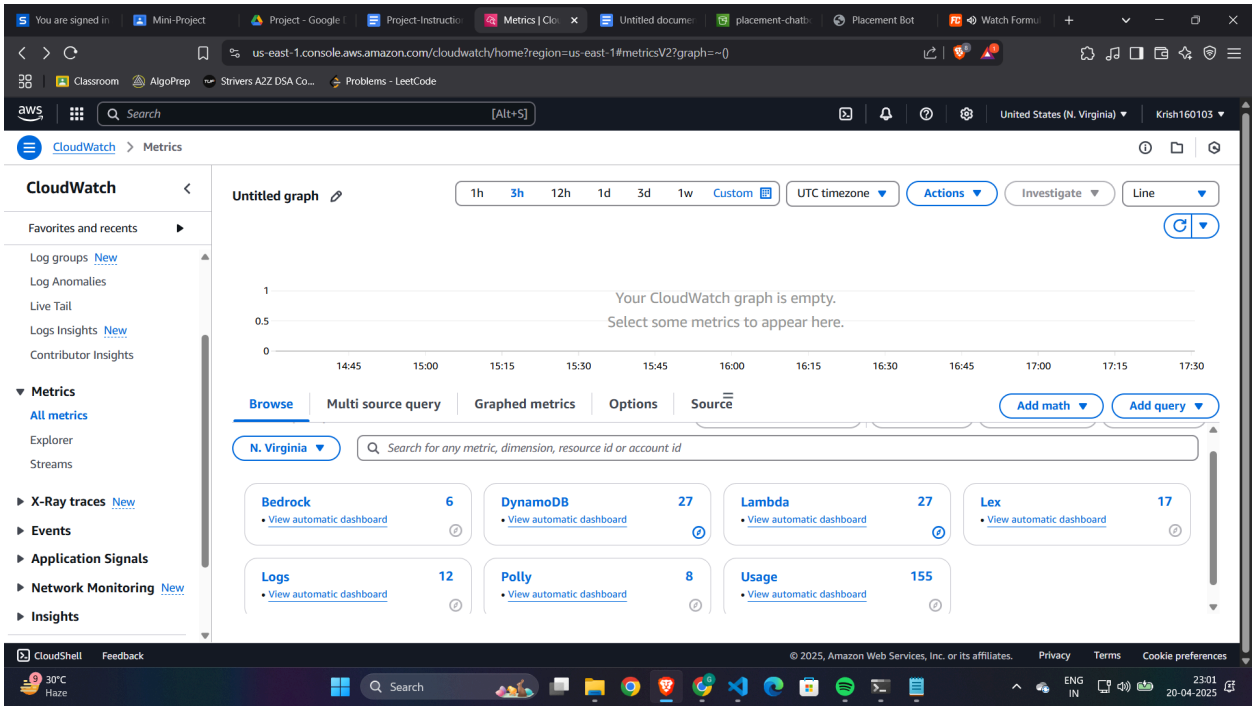


Figure: 20 CloudWatch Metrices

This screenshot shows the CloudWatch dashboard with metrics for the Placement Bot's performance, including the number of invocations for Lex, Lambda, and other AWS services.

CloudWatch is an important monitoring tool used for tracking the performance of various AWS resources. This figure shows the metrics for the Placement Bot, including the number of invocations for AWS Lex, Lambda, and DynamoDB, along with their usage and error rates. Monitoring these metrics allows developers to ensure that the bot is functioning optimally. If any service experiences issues, CloudWatch alerts can help developers take immediate action to resolve the problem, thereby maintaining a seamless user experience.

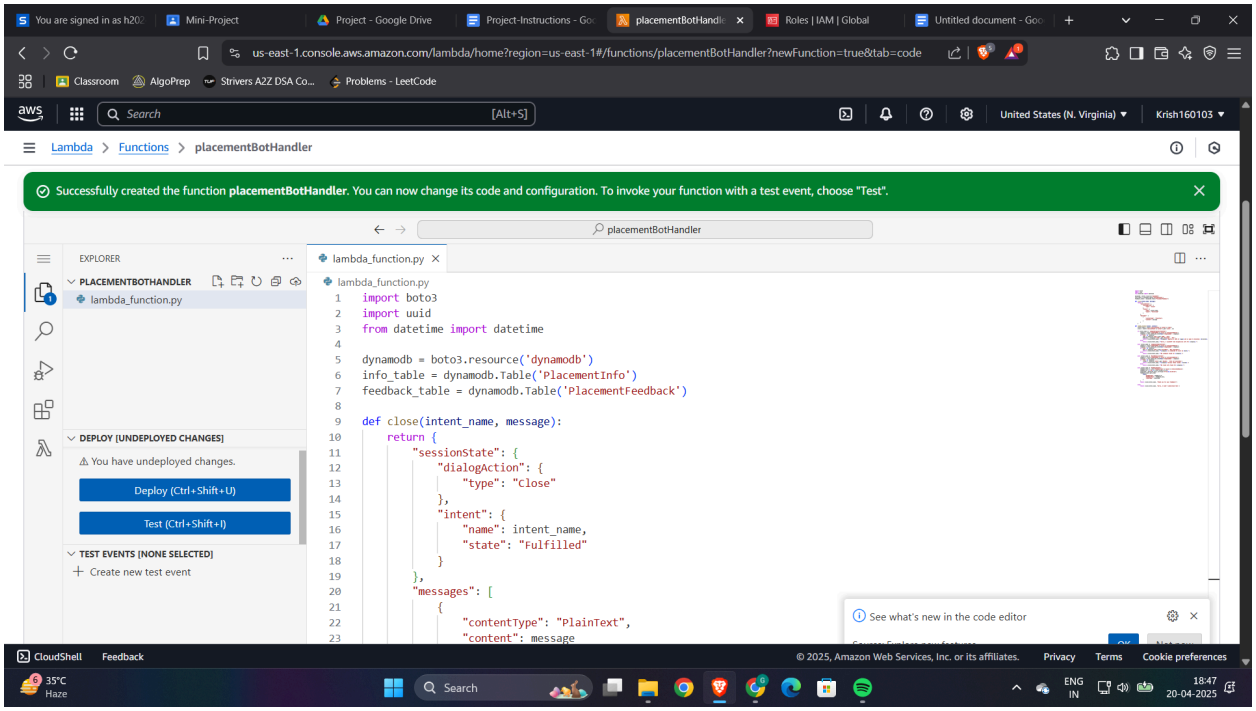


Figure: 21

The screenshot illustrates the Lambda function code that processes user inputs and sends appropriate responses, playing a key role in the bot's backend logic.

In this screenshot, we can see the Lambda function that powers the backend of the Placement Bot. AWS Lambda is used to execute the logic needed for processing user requests and generating dynamic responses. For example, when a user queries the bot about a company's eligibility criteria, Lambda processes the request, queries the data from DynamoDB, and returns the appropriate response to the user. The Lambda function is integrated with Amazon Lex to handle different intents, and its ability to execute custom code makes it ideal for providing personalized responses based on real-time data.

## Feedback Mechanism:

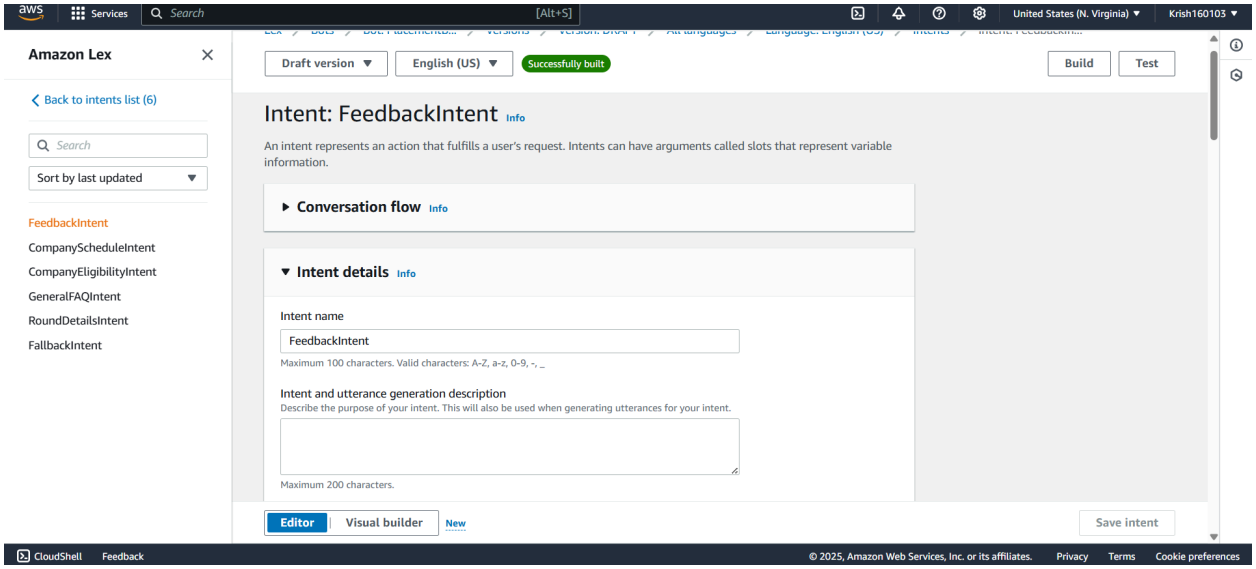


Figure: 22 Feedback Mechanism in Amazon Lex

This image shows the configuration of the "FeedbackIntent" in Amazon Lex, enabling the bot to handle feedback from users after interactions.

In addition to providing placement-related information, the Placement Bot also incorporates a feedback mechanism. As shown in this figure, a "FeedbackIntent" is configured within Amazon Lex. This intent is designed to capture user feedback, allowing users to share their thoughts on the bot's performance. The feedback collected is stored in DynamoDB, providing valuable insights into how well the bot is performing. This data can be used for improving the bot’s responses and making it more effective in future interactions.

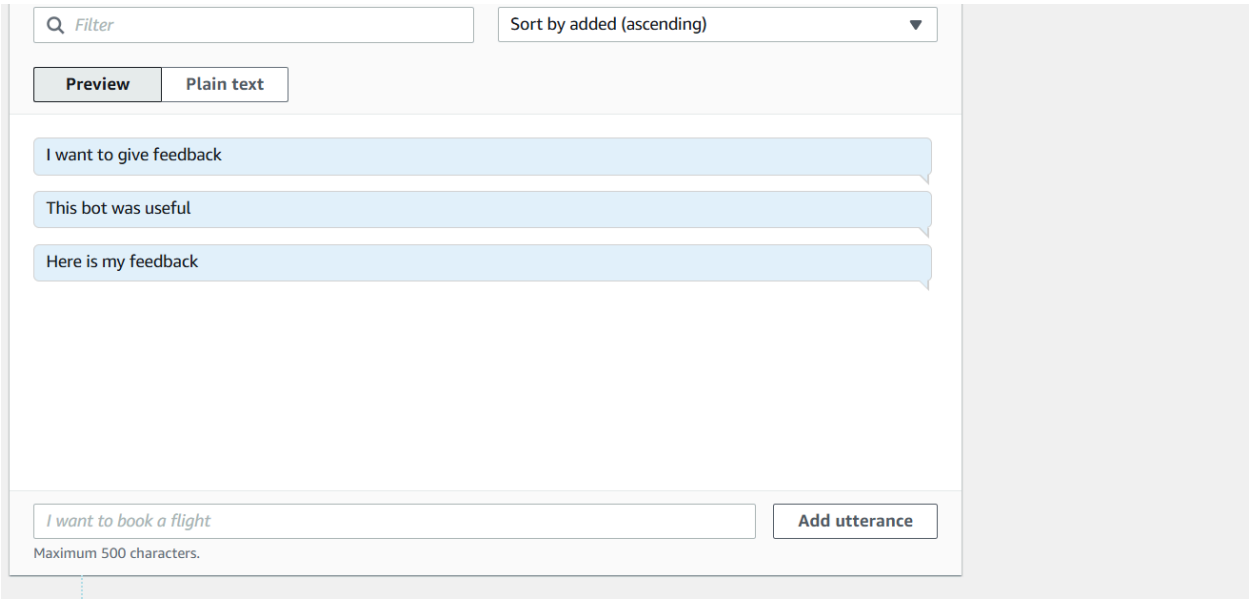


Figure: 23 Sample Utterances for Feedback Intent

Here, sample utterances are defined for the "FeedbackIntent" in Amazon Lex, allowing the bot to recognize different ways users may provide feedback.

In this figure, we see the configuration of sample utterances for the "FeedbackIntent." These utterances help Amazon Lex understand the different ways users may express feedback. By defining multiple variations, such as "I want to give feedback," "This bot was useful," and "Here is my feedback," the bot becomes more flexible in handling user inputs. This ensures that the bot can respond to feedback in a variety of ways, improving its accuracy and responsiveness during conversations.

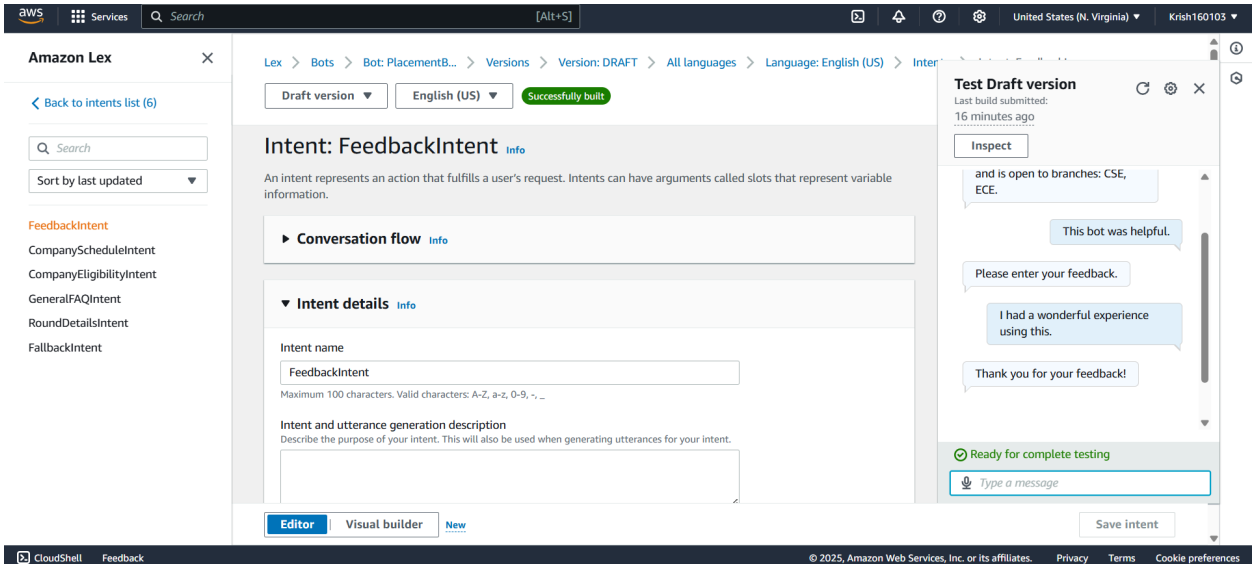


Figure: 24 Creating Feedback Intent in Amazon Lex

The above image shows the configuration of the "FeedbackIntent" in Amazon Lex. This intent represents the feedback mechanism where the user is prompted to provide their feedback. The design of the "FeedbackIntent" allows the bot to capture feedback from the user through a simple conversational interaction. We have set up a specific prompt to ask for feedback after the primary task is completed. This ensures that users can express their thoughts on the bot's performance. Once the user inputs their feedback, it is captured and stored for further analysis, helping improve the bot's responses and functionalities.

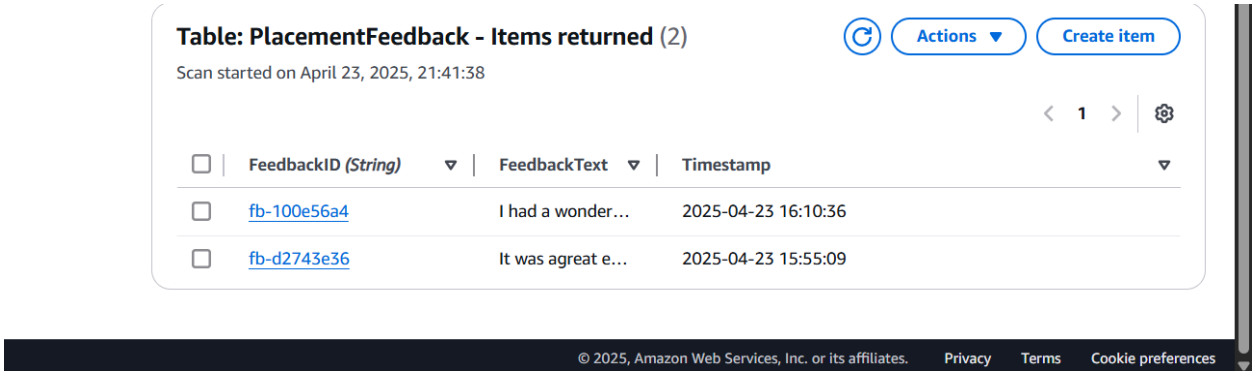


Figure: 25 Storing Feedback in DynamoDB Table

After the feedback is captured, the data is stored in an Amazon DynamoDB table as shown in the image. The table is called PlacementFeedback, and it contains columns for the FeedbackID, FeedbackText, and Timestamp. This feedback mechanism enables the system to collect valuable data from users, which can then be analyzed for enhancing user experience. The feedback is essential for identifying areas of improvement and tailoring responses according to user expectations.

## Admission PIN control:

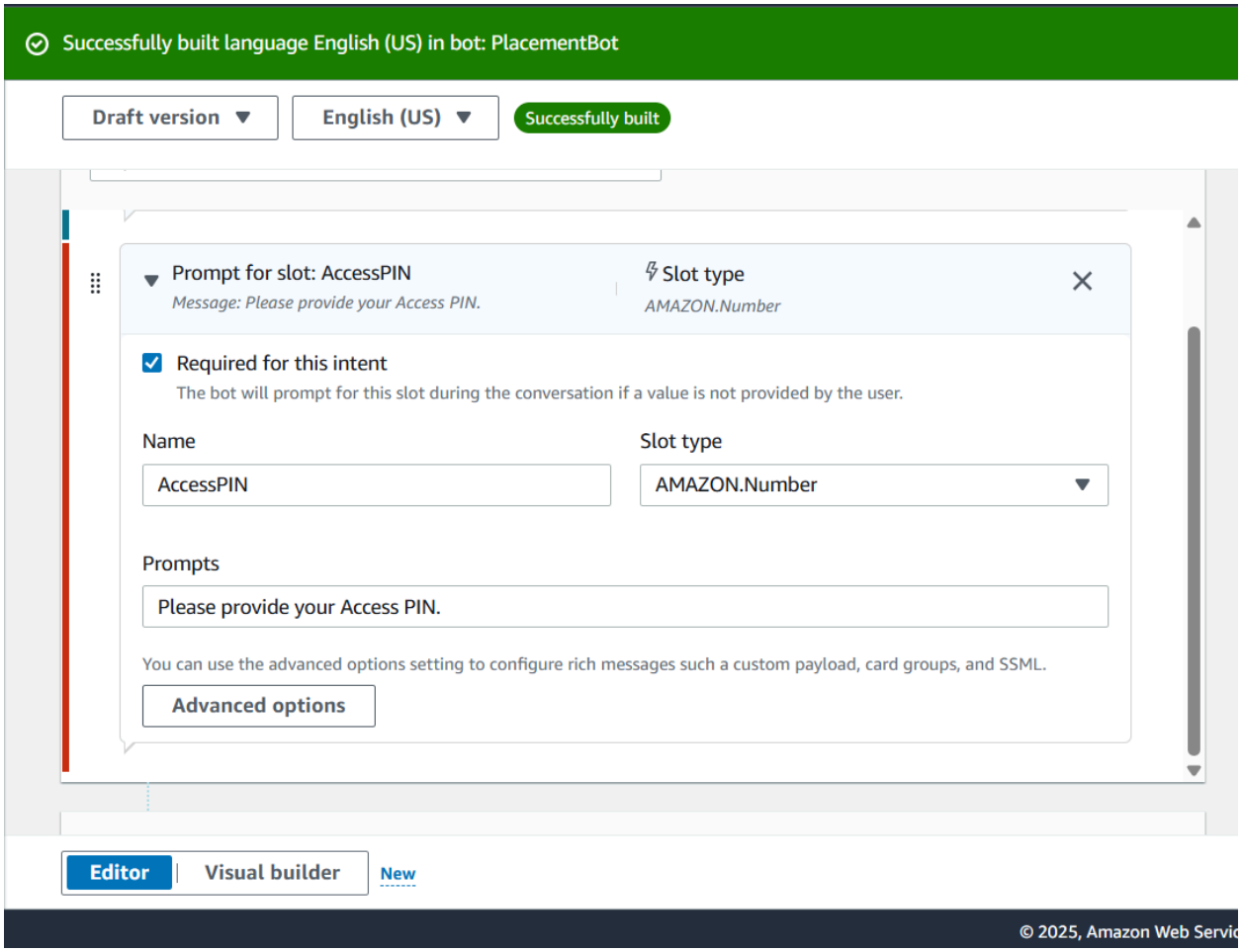


Figure: 26

The image depicts the configuration of the "AccessPIN" slot in Amazon Lex. The bot prompts the user to enter a PIN before proceeding with certain intents. This Admission PIN control is essential for securing access to more sensitive or privileged actions, such as accessing placement-related data. By defining this slot and setting the AMAZON.Number slot type, the bot is able to accept numeric input and verify if the user has provided the correct PIN. This added layer of security ensures that only authorized users can interact with specific parts of the bot.

```
if intent_name == 'CompanyEligibilityIntent':
    pin_slot = slots.get('AccessPIN')
    if not pin_slot or 'value' not in pin_slot:
        return {
            "sessionState": {
                "dialogAction": {"type": "ElicitSlot", "slotToElicit": "AccessPIN"},
                "intent": {
                    "name": intent_name,
                    "slots": slots,
                    "state": "InProgress"
                }
            },
            "messages": [{"contentType": "PlainText", "content": "Please provide your Access PIN."}]
        }
```

Figure: 27 Lambda Function for PIN Validation

The above image shows a snippet of the AWS Lambda function responsible for validating the PIN entered by the user. This function is called when the user inputs the PIN. The Lambda function checks whether the PIN is valid and proceeds accordingly. If the PIN is invalid, the bot prompts the user to enter the correct PIN again. This function adds logic to handle the PIN validation and ensures the security of the bot's interactions.

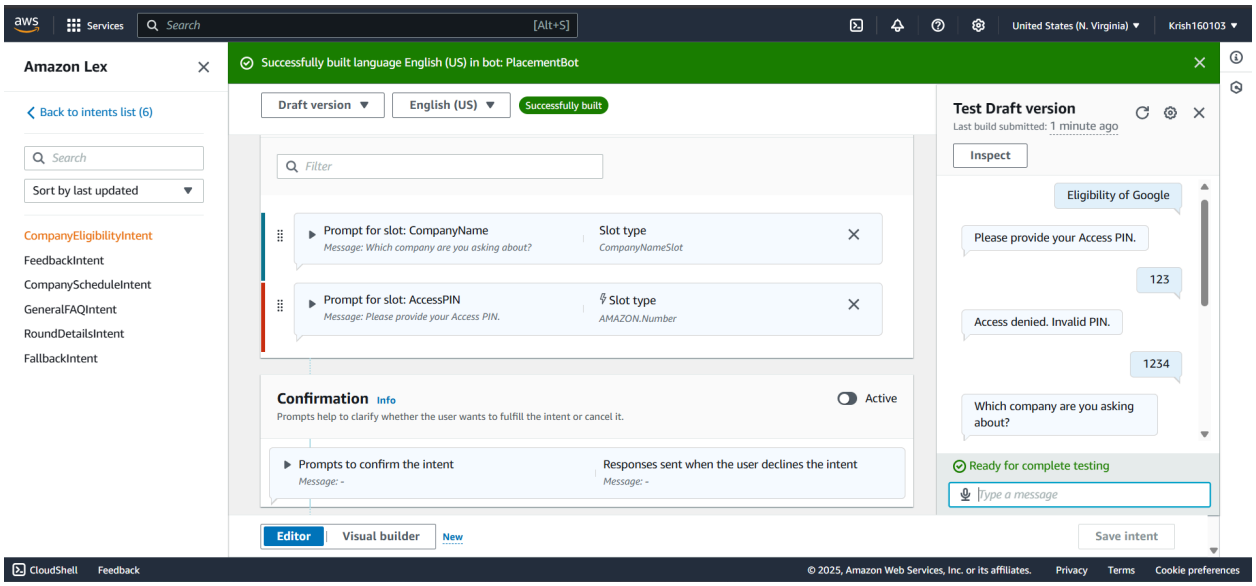


Figure: 28 Configuring Prompts for Company Eligibility Intent in Lex

In the image, the configuration of the "CompanyEligibilityIntent" is shown, where the bot asks users for the CompanyName and AccessPIN. The prompt for CompanyName is designed to ask users about the company they are interested in learning more about, while the AccessPIN prompt is displayed to ensure security. After receiving the user's input, the bot processes the information and fetches the eligibility details for the specified company. The flow ensures that users can get personalized and secure access to the information.

## Dynamic scalability:

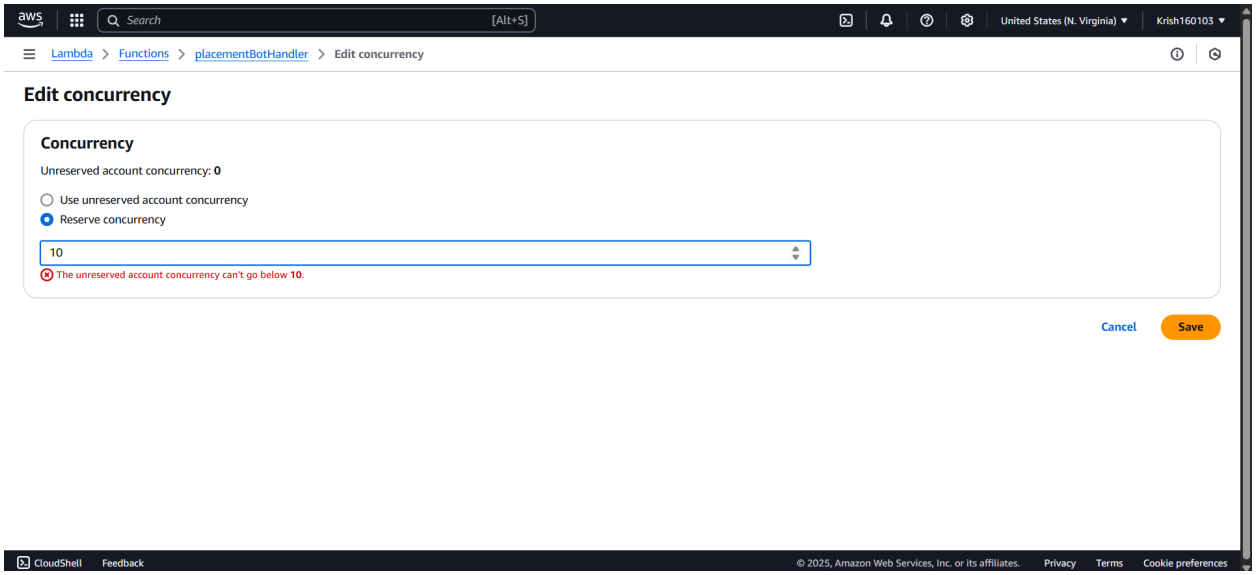


Figure: 29 *Enabling Dynamic Scalability in AWS Lambda*

This image highlights the configuration of dynamic scalability settings for the Lambda function. AWS Lambda's scalability feature ensures that the chatbot can handle varying traffic loads efficiently. In the case of sudden traffic spikes (e.g., during peak hours when many users interact with the bot), AWS Lambda dynamically adjusts the number of running instances to maintain performance without compromising response time. The flexibility of this configuration helps in optimizing costs while ensuring a smooth user experience.

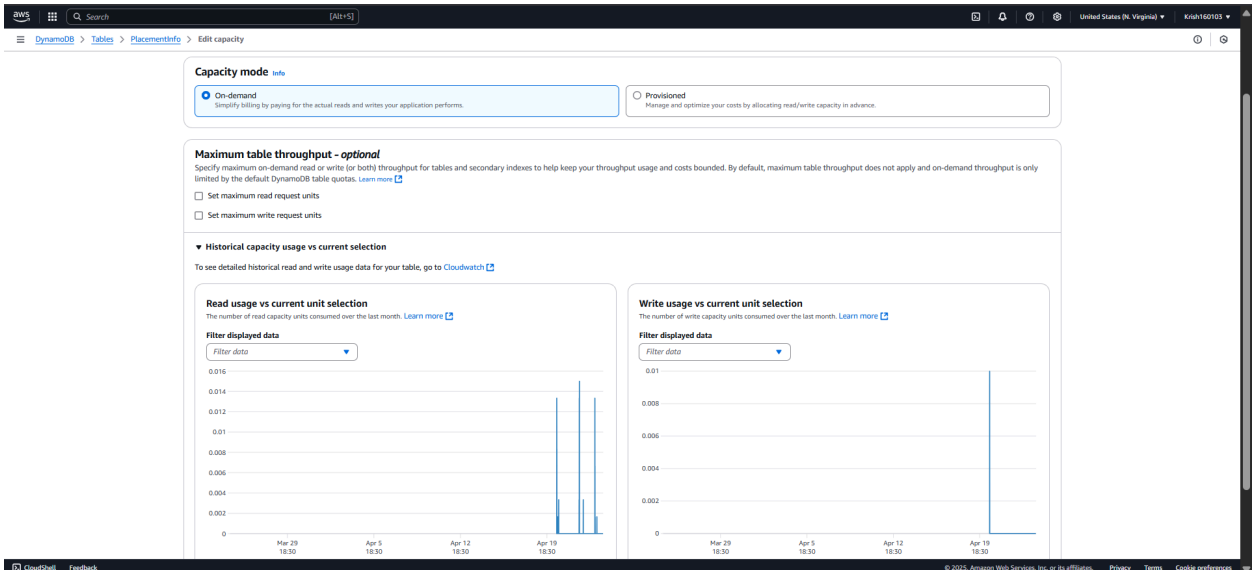


Figure: 30 Monitoring DynamoDB Read/Write Capacity

This image displays the configuration of DynamoDB's read and write capacity. Monitoring these metrics is crucial to understanding the performance and efficiency of the database. During periods of high usage, the bot interacts with the database to fetch company-specific information. The ability to monitor capacity helps optimize the system's resource allocation and ensures that data queries do not result in delays or errors.

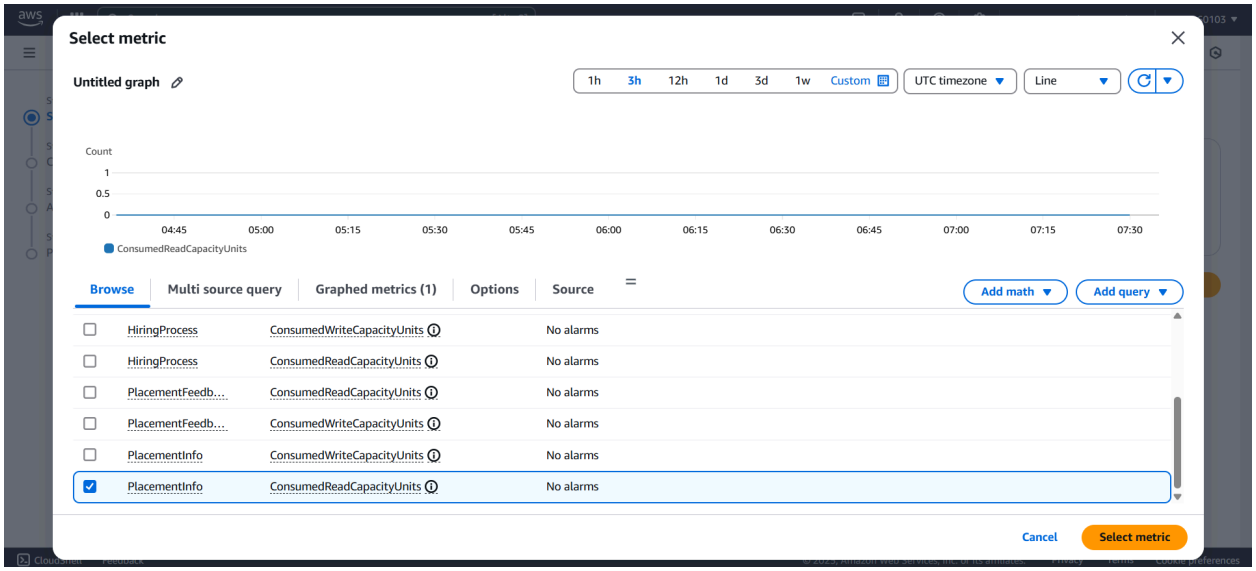


Figure: 31 Setting Up CloudWatch Alarms

CloudWatch alarms play a vital role in tracking the bot's health. The above image illustrates the configuration of an alarm that is set up to monitor errors generated by the bot. By creating an alarm, we can get real-time notifications when an issue arises, such as failed requests or unhandled errors. This proactive monitoring ensures that any bottlenecks or problems in the bot's performance can be addressed swiftly, improving overall reliability.

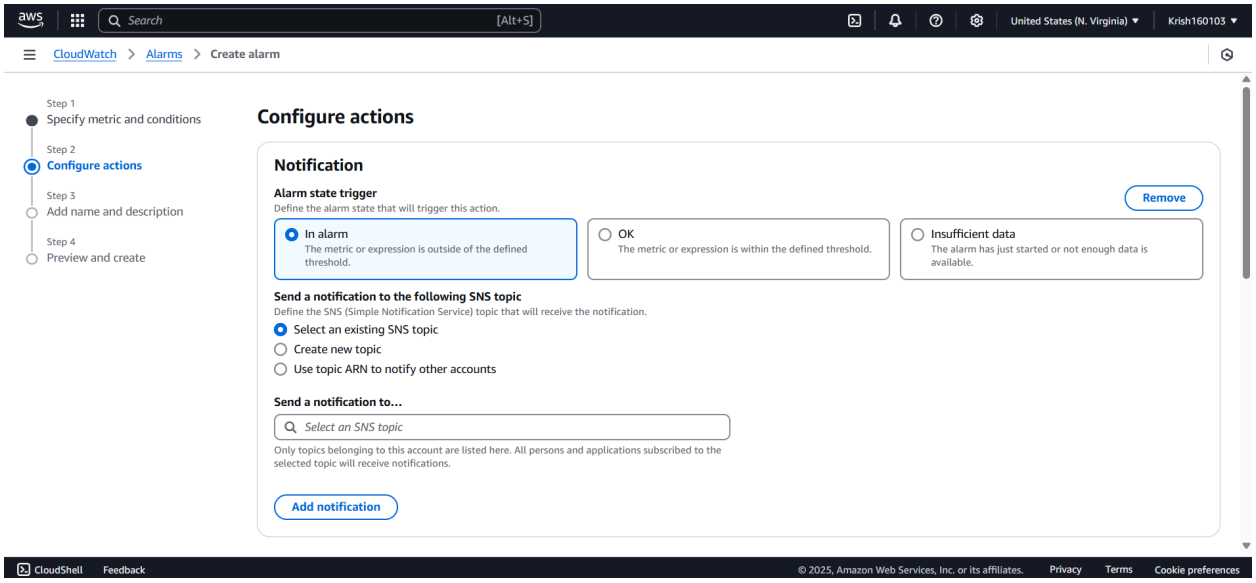


Figure: 32 Viewing CloudWatch Dashboard for monitoring Bot Metrices

This image shows the CloudWatch dashboard where key metrics for different services (like Lex, Lambda, DynamoDB, etc.) are monitored. By having real-time visibility into the performance metrics, the bot administrators can track usage patterns, error rates, and latency issues. These insights help in troubleshooting problems and optimizing performance, ensuring the bot operates efficiently during peak loads.

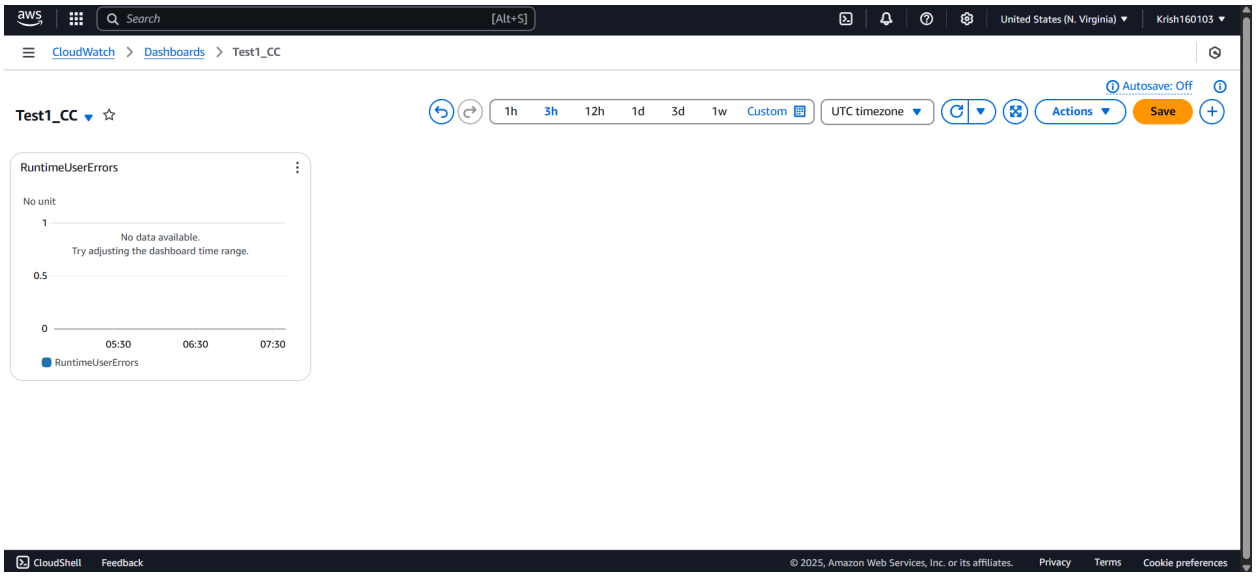


Figure: 33 CloudWatch Dashboard - Test Metrics

The image shows an empty CloudWatch dashboard. CloudWatch is integrated into the system to monitor various aspects of the Placement Bot's performance, such as API call metrics, Lambda function performance, and error rates. CloudWatch allows us to track operational issues, view logs, and set up alarms for specific events, ensuring the system's smooth functioning. Although this screenshot currently displays no data, it serves as a monitoring tool to visualize system behavior in real-time. Metrics related to the bot's backend components, including AWS Lambda, Amazon Lex, DynamoDB, and more, will be populated here to enable better monitoring, debugging, and optimization.

## Authentication:

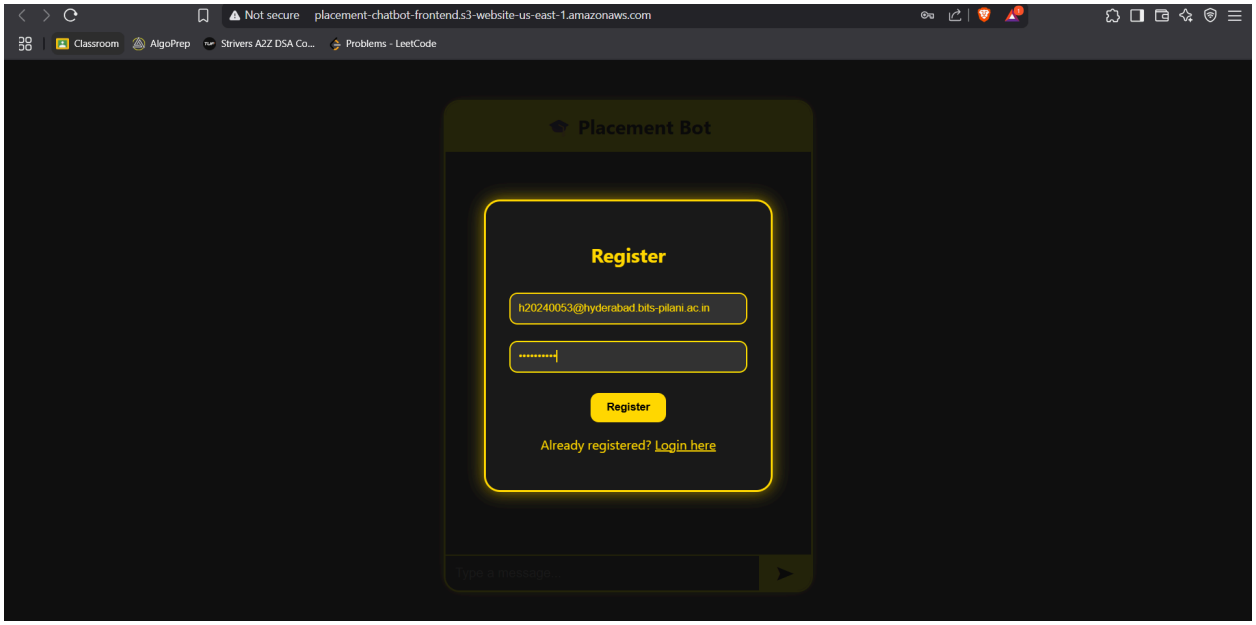


Figure: 34 Adding Security layer through authentication

This image showcases the registration interface of the Placement Bot, where users can securely sign up using their institutional email address (xxxxxxxx@hyderabad.bits-pilani.ac.in). The implementation ensures that only valid email addresses from the institute can be used to register. Upon successful registration, the user's credentials are securely stored and validated, allowing them to access the bot's features and information. This security layer helps in restricting unauthorized access to the bot, which is crucial for protecting sensitive data and maintaining integrity.

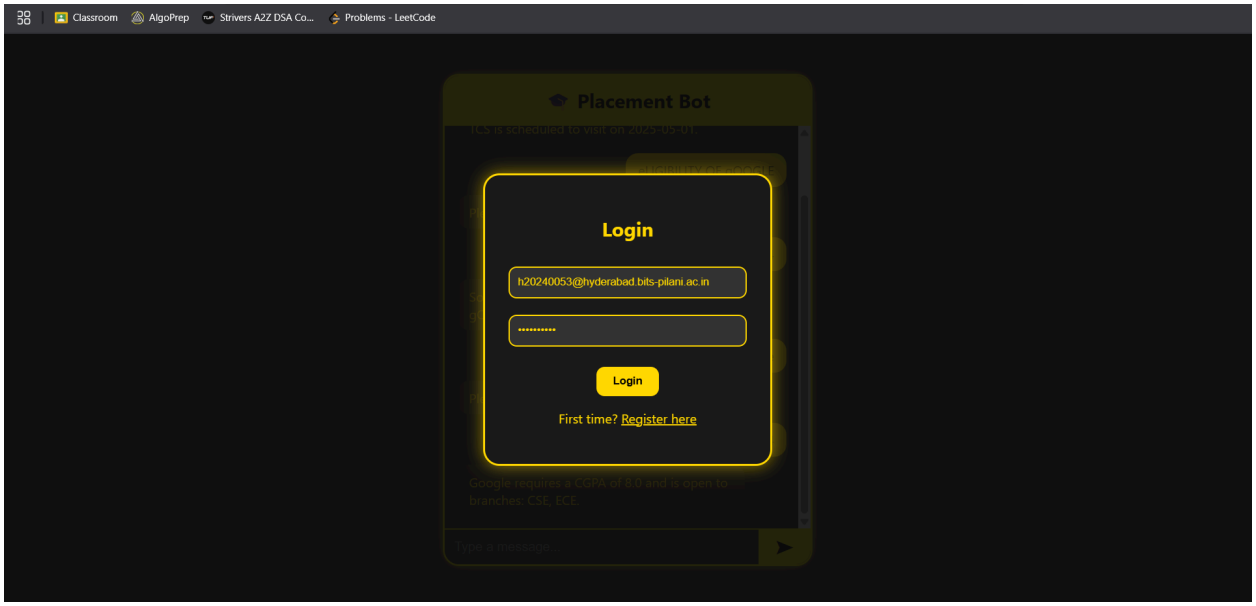


Figure: 35 Login Screen

The screenshot displays the login page for the Placement Bot, where users input their registered email and password to gain access. The login page of the Placement Bot is shown, where users can input their registered email and password to authenticate themselves. This page is designed to handle secure logins, ensuring that only authorized users with valid credentials can access the chatbot. The integration with Amazon Cognito helps manage the user sessions and ensures that only verified users have access to the bot. The login process is critical to prevent unauthorized access and enhance the overall security of the bot.

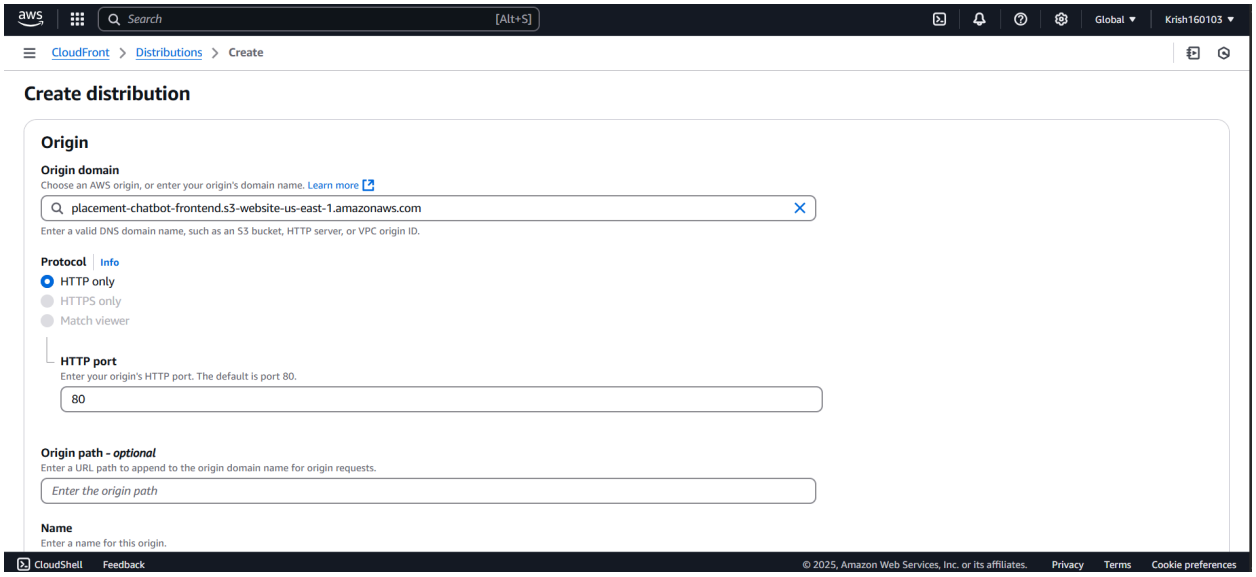


Figure: 36 CloudFront Distribution Step

This figure highlights the configuration of the Amazon CloudFront distribution for the Placement Bot's frontend application. CloudFront serves as the Content Delivery Network (CDN) to ensure faster and more reliable delivery of the web assets (HTML, CSS, JavaScript, and other files) to end-users. By using CloudFront, the bot's frontend is distributed across multiple edge locations, reducing latency and improving performance for users accessing the bot from various geographical locations. The CloudFront distribution setup ensures that the frontend is accessible and performs optimally.

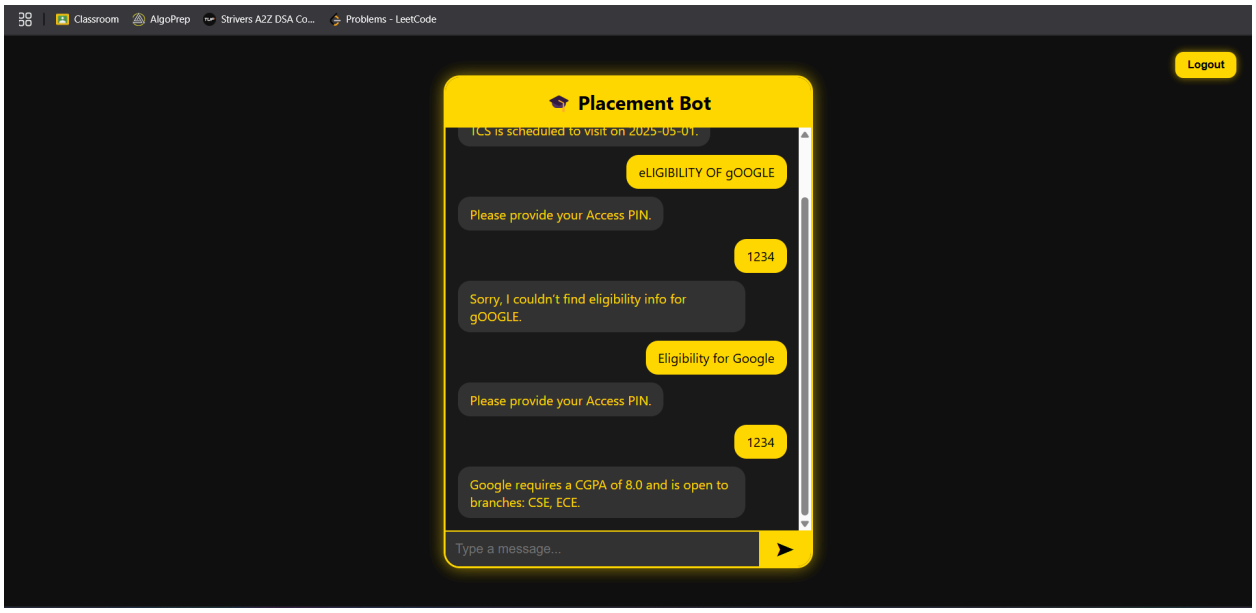


Figure: 37 User Interaction with the Bot

This screenshot demonstrates how the Placement Bot interacts with users during a conversation, asking for a user's access PIN.

In Figure 37, the bot is actively engaging with the user, requesting an access PIN to authenticate and proceed with the conversation. This demonstrates how the bot handles more specific user inputs like a PIN and uses session-based interactions. This is part of the added security measures, where users must provide a valid PIN before the bot gives detailed information. This secure step ensures that sensitive information is only provided to authorized individuals, making the system more robust in terms of both functionality and security.

## Multilingual support:

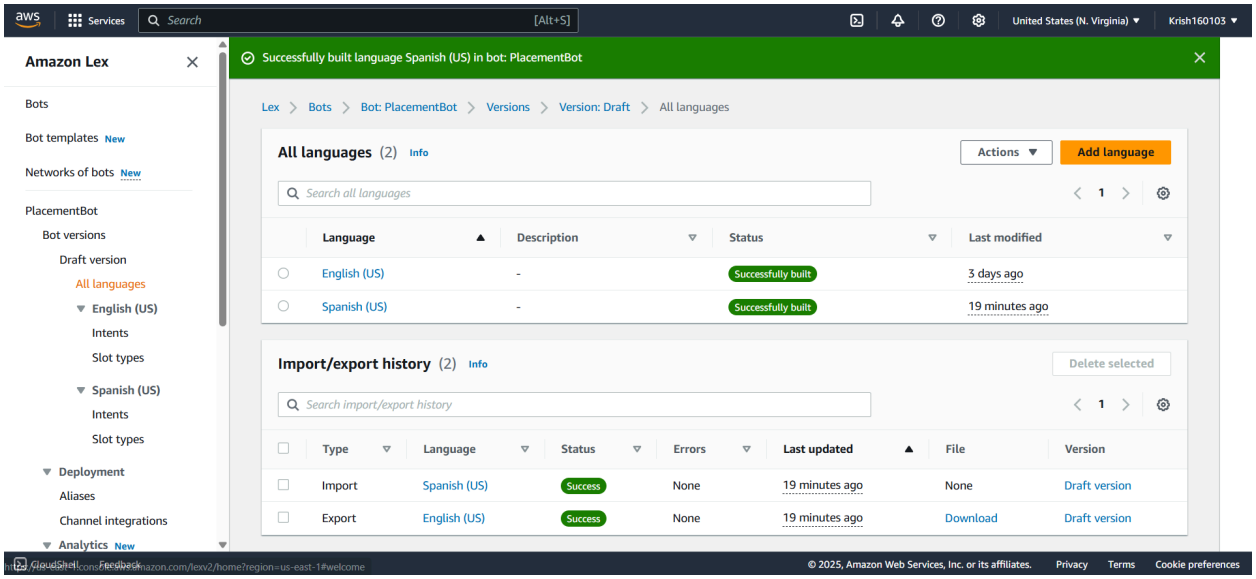


Figure: 38 Multilingual Support in the Bot

The screenshot displays the Amazon Lex interface where multilingual support for the Placement Bot is configured, showing both English and Spanish language options.

This figure depicts the multilingual setup in Amazon Lex for the Placement Bot. The bot has been configured to support multiple languages, starting with English and Spanish. This feature allows the bot to cater to a broader audience, providing seamless support in both languages. The addition of a second language helps in enhancing user accessibility and makes the bot more inclusive. The setup involves adding intents, sample phrases, and responses in both languages, ensuring that users can interact with the bot in their preferred language.

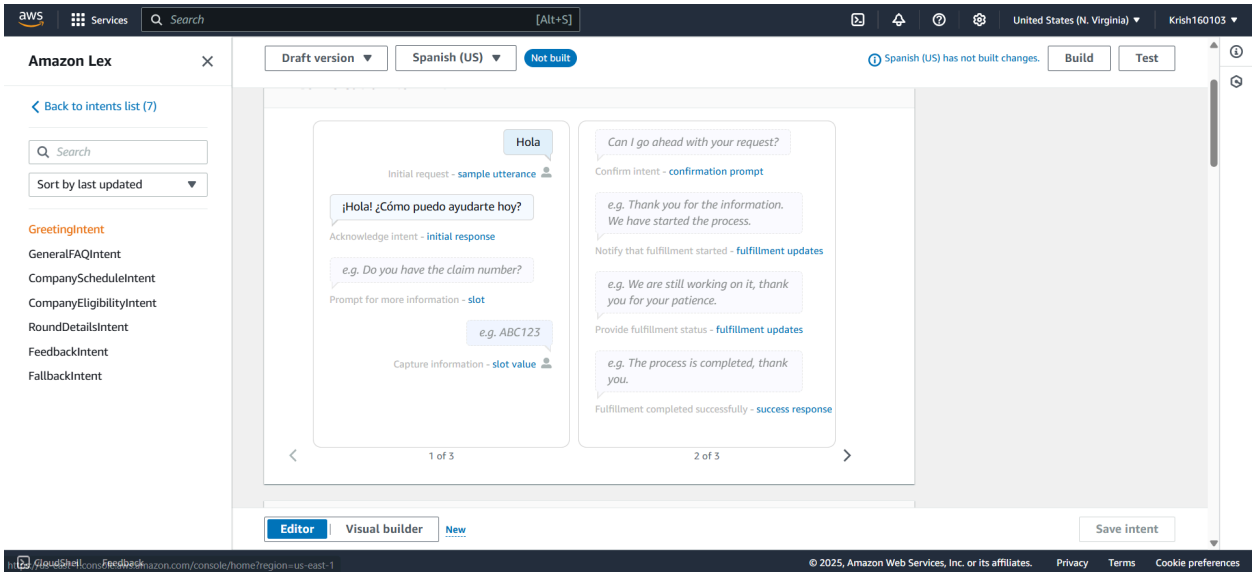


Figure: 39 testing the Spanish Language Intent

The screenshot shows the bot's testing phase in Amazon Lex, where it responds to Spanish queries like "*¿Cómo puedo ayudarte hoy?*" ("How can I assist you today?").

This image illustrates the testing phase of the Placement Bot in the Spanish language. The bot can understand and respond to user queries in Spanish, as shown in the conversation, which demonstrates that the multilingual support is functioning as expected. This feature not only broadens the bot's user base but also provides a seamless experience for Spanish-speaking users, allowing them to interact naturally without facing language barriers. This setup enhances user experience and expands the reach of the bot to a more diverse audience.

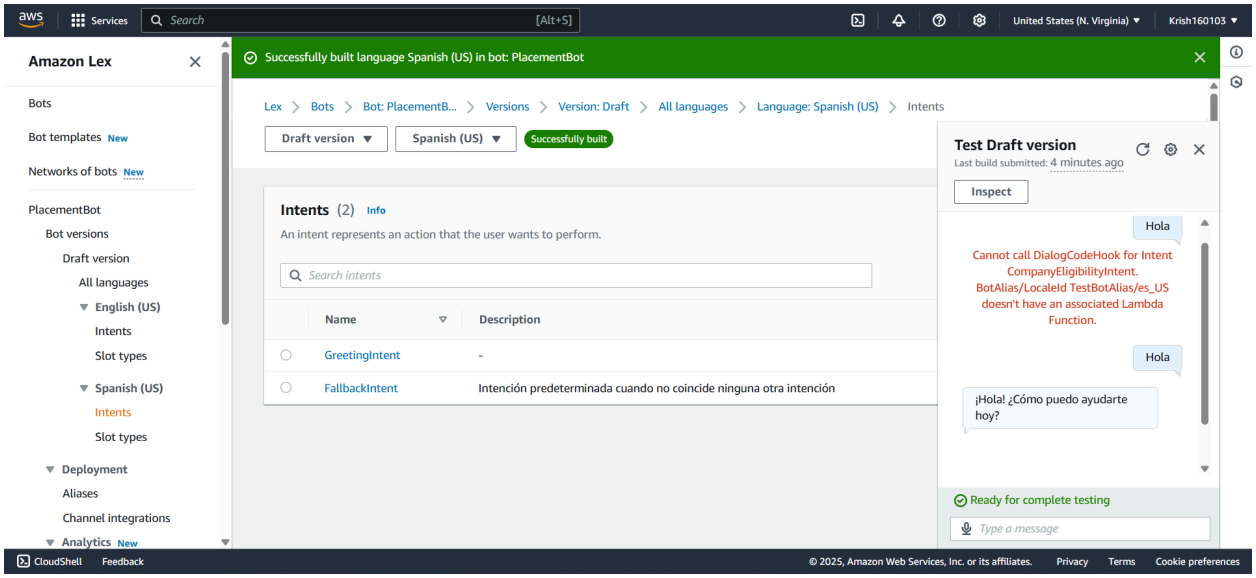


Figure: 40 Spanish Language Intent Configuration

The screenshot shows the successful creation of intents in the Spanish version of the Placement Bot in Amazon Lex.

The configuration of intents in the Spanish language version of the Placement Bot. The intents in Spanish are fully configured and tested to ensure they work similarly to their English counterparts. As shown, the bot understands various intents like "CompanyEligibilityIntent" and "FeedbackIntent" in Spanish. By enabling these intents, the bot can provide accurate and relevant responses to users in Spanish, offering a complete multilingual experience. This step ensures that the bot is well-equipped to handle queries in both English and Spanish, providing a global reach.

# Challenges faced while building:

1. **Complex Intent Mapping and Slot Filling**:
   Ensuring accurate intent recognition and proper slot filling was challenging due to variations in user input. Synonyms, phrasing differences, and ambiguous queries made it difficult to map inputs to the correct intents and extract necessary parameters.

2. **Multilingual Support Complexity**:
   Extending the bot to support both English and Spanish involved translating intents, utterances, and responses. Ensuring consistency and accuracy in responses across languages, while maintaining the bot's functionality, required extensive testing and fine-tuning.

3. **Integrating Different Lambda Functions to Intents**:
   Managing the integration of multiple Lambda functions with different intents was complex. Each intent needed a specific Lambda function for dynamic responses, and ensuring that the right function was triggered for the appropriate intent was crucial for accurate user interactions.

4. **Frontend Integration Issues (UUIDv4 Issue)**:
   Integrating the frontend with the backend services, particularly generating unique session IDs using UUIDv4, caused session management issues. Ensuring that each user had a unique session ID without conflicts was essential for maintaining session integrity.

# Conclusion:

In conclusion, the Placement Bot built using AWS technologies provides a robust and scalable solution for addressing the challenges students face in accessing real-time placement information. By leveraging AWS Lex for natural language processing, AWS Lambda for dynamic backend logic, and DynamoDB for storing real-time placement data, the bot offers a highly efficient, user-friendly, and interactive platform for students.

The multilingual support enhances the accessibility of the bot to a diverse audience, allowing students to interact with it in both English and Spanish. The security features, such as Amazon Cognito for user authentication and access control, ensure that only authorized users can access placement information, protecting sensitive data.

Despite its strengths, the Placement Bot faces challenges related to handling user query ambiguity, maintaining up-to-date placement data, and managing traffic spikes during high-demand periods. Further improvements can be made in terms of expanding the bot's multilingual capabilities, enhancing its contextual understanding, and ensuring seamless integration with third-party APIs.

Overall, the Placement Bot significantly streamlines the placement process for students, making it easier for them to obtain the necessary information. With continuous monitoring, optimization, and updates, the bot can provide a more personalized and efficient experience, ultimately improving the placement process for students and institutions alike.