Submitted by : Aishwarya Gupta
Email : apgupta@scu.edu

# Career Readiness Inventory Architecture

## Project Requirements Specification

**Title**: Real-Time Data Processing and Visualization System for Educational Data
**Objective**: Develop a scalable and secure cloud-based solution to receive, process, store, and visualize form-based educational data including multiple-choice questions (MCQs) dynamically on a React frontend.

**Functional Requirements**

1. **Data Reception:** The system must be capable of securely receiving data from an external webhook and should handle data payloads ranging from 1 to 10,000 requests per month, with a potential maximum of 1,000 requests in a single day.
2. **Data Storage**: The database should automatically scale in response to varying loads to ensure cost-effectiveness and efficiency.
3. **Dynamic Data Visualization:** The React frontend must dynamically update visualizations to reflect changes in the data stored in the database.
4. **Security Compliance:** The entire architecture must comply with the HECVAT security protocol to ensure the protection of student data.

**Non-Functional Requirements**

1. **Scalability:** The system must scale automatically to handle increases in data input and querying loads without manual intervention.
2. **Security:** Use AWS-managed services to ensure robust security measures are in place and secure student data including data encryption (both in transit and at rest).
3. **Reliability:** The system should maintain high availability and guarantee data integrity.

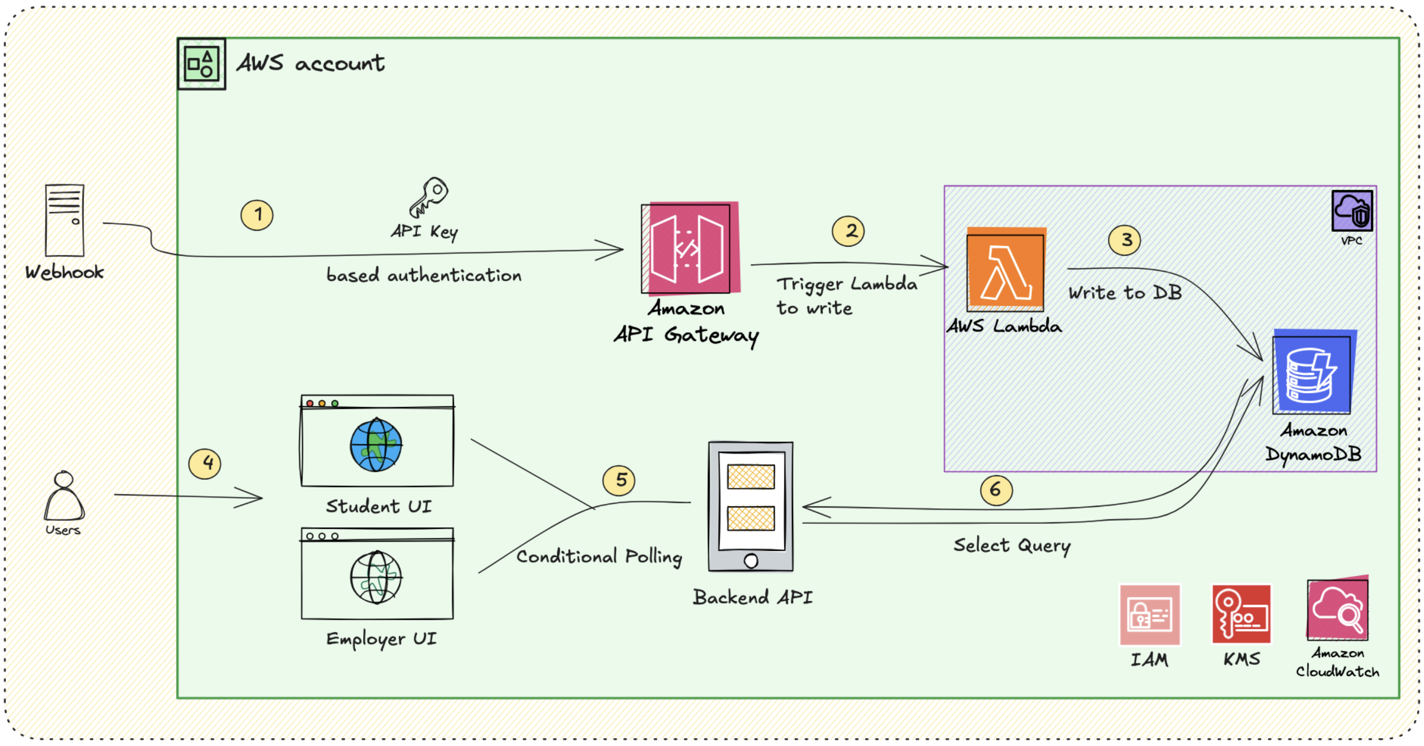
## Suggested Architecture

**Components:**

1. **API Gateway**: Handles incoming requests securely. API Gateway acts as the entry point for incoming webhook data, configured with API keys for authentication.
2. **AWS Lambda**: Processes the webhook data. Lambda functions are configured within a VPC to ensure secure access to internal AWS resources and can perform operations such as data transformation and logging before storing data in DynamoDB.
3. **DynamoDB**: Stores form and MCQ data securely. Accessed through VPC endpoints to ensure that all data traffic between DynamoDB and the Lambda functions within the VPC remains on the AWS network, enhancing security.
4. **VPC Endpoints**: Allows private connections between the VPC and AWS services like DynamoDB, ensuring that data does not travel over the public internet.
5. **React Frontend**: Displays data visualizations and updates dynamically.

Submitted by : Aishwarya Gupta
Email : apgupta@scu.edu

6. **AWS Key Management Service (KMS):** Manages encryption keys used for encrypting data in transit and at rest, ensuring compliance with security protocols for handling sensitive student data.
7. **AWS CloudWatch:** Monitors the operation of the application, tracking logs, metrics, and events to provide visibility into application performance, operational health, and security auditing.
8. **AWS IAM:** Manages access control with granular permissions to AWS resources, ensuring that only authorized and authenticated entities can access sensitive data or perform specific actions.



Career Readiness Inventory Architecture v1

**Data Flow :**

1. **Webhook to API Gateway**: External data sources send data to the API Gateway endpoint. API Gateway securely processes these requests and passes the data to the configured Lambda functions.
2. **API Gateway to Lambda**: Lambda functions, which reside within the VPC, are triggered by API Gateway. These functions are responsible for processing the data, which may include parsing, validating, and possibly enriching the data before storage.
3. **Lambda to DynamoDB**: Processed data is stored in DynamoDB tables. The interaction between Lambda and DynamoDB is facilitated by VPC endpoints, ensuring secure and private data flow.
4. **User Views React UI:** The user visits the page and views the data visualisation reports on the page
5. **Conditional Polling on React UI:** The frontend initially fetches data. After this, the front end begins polling the API at a set interval, but only when user interaction is detected or

when a button is triggered on UI (refresh). This leads to near-real-time data visualisation, which is highly cost-effective.

6. **Backend API and DynamoDB:** The backend API fetches the updated data from DynamoDB and returns it to the React UI. **Note:** This can be implemented as a Lambda or as an EC2 depending on the Polling constraints. I'm assuming Lambda Implementation.

## Time Estimation with Parallel Execution and MVP Focus

Following are the stories required to build the above architecture -

**Story 0:** Comprehensive Security with AWS IAM
- **Estimated Time**: 1 day

**Story 1:** Configure API Gateway to receive webhook data, including setting up API keys and rate-limiting.
- **Estimated Time**: 1 day

**Story 2:** Develop and deploy AWS Lambda functions to process incoming data and write to DynamoDB as well for backend API to Read data from DynamoDB.
- **Estimated Time:** 2-3 days

**Story 3**: Configure DynamoDB tables to store data securely.
- **Estimated Time:** 2 days

**Story 4:** Configure VPC endpoints for secure AWS service communication.
- **Estimated Time:** 1 day

**Story 5:** Implement basic polling in the existing React UI to fetch updates from the backend.
- **Estimated Time:** 1 day

**Story 6:** Implement data encryption both in transit and at rest using AWS KMS.
- **Estimated Time:** 1 day

**Story 7:** Set up basic monitoring and logging using AWS CloudWatch.
- **Estimated Time:** 1 day

**Story 8**: Thoroughly Testing all the functionalities ( functional testing, load testing)
- **Estimated Time:** 1-2 days

**Total Estimated Time:** 14 days (+ 2 days of buffer)

## Monthly Cost Estimation

The following are the costs per Service -

**API Gateway:** $3.50 per million API calls.

**Lambda:** 1 million free requests per month and 400,000 GB-seconds of compute time.

Submitted by : Aishwarya Gupta
Email : apgupta@scu.edu

**DynamoDB:** 25 GB of storage, 25 provisioned Write Capacity Units (WCU), and 25 Read Capacity Units (RCU) free per month. (exceeding limits - $1.25 per million WCUs, and reads are $0.25 per million RCUs)

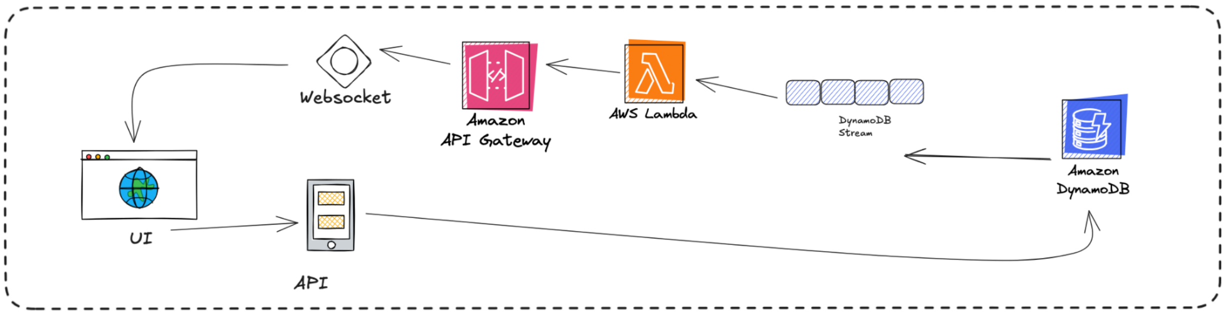
**Scenario Estimates:**

- **100 Responses/Month**: Likely within the free tier of all services.
- **1,000 Responses/Month**:
  - **API Gateway**: $0.0035
  - **Lambda**: Still within the free tier.
  - **DynamoDB**: Potentially still within the free tier, depending on item size.
- **5,000 Responses/Month**:
  - **API Gateway**: 5,000 requests x $3.50 / 1,000,000 = $0.0175
  - **Lambda**: Assuming each request consumes 0.1 GB-second, total GB-seconds = 500. Within the free tier.
  - **DynamoDB**: If each item write consumes less than 1 WCU, the operation remains within the free tier.

## Future Improvements

Implementing WebSockets instead of Conditional Polling
Pros:
1. WebSockets allow for real-time data transmission between the client and server. This means React UI can receive updates instantly as soon as data changes on the server.
2. With WebSockets, the server can push updates only when there are changes, reducing the need for frequent unnecessary requests and data transfer.

Cons:
**Message Delivery Charges**: AWS charges for every message sent and received through the WebSocket connection. The cost is typically about $1.00 per million messages.
**Connection Time Charges**: This is based on the total connection time, charged per hour. For example, the cost might be around $0.25 per million connection minutes.



Brief Overview of Websocket for Dynamic Frontend Updation