# 

**Enterprise Data Architecture Part 4: End-to-End Solution Integration**

*Luwei Wang (lw3511), Zachary Zeng (nz900), Vincent Xu (yx2021)*

*New York University CSCI-GA.2433 - Fall 2024*

*Github link:*[*https://github.com/zaczeng816/ds-ga-2433-proj*](https://github.com/zaczeng816/ds-ga-2433-proj)

1. **Abstract**

This project explores an end-to-end solution integrating enterprise data architecture to manage and process patient information efficiently. Using AWS RDS as the central relational database and leveraging SQLAlchemy as the ORM tool, the project demonstrates the seamless interaction between backend machine learning models and a user-friendly frontend. The workflow includes user authentication, dynamic insurance prediction, and real-time data modification. Key features like RMSE and MSE scores are incorporated to validate the predictive models, ensuring the system's robustness. The solution serves as a comprehensive case study for combining cloud infrastructure with machine learning and data engineering to address business challenges in the healthcare domain.

1. **Introduction**

Managing healthcare data requires robust systems to ensure accuracy, scalability, and security. This project addresses the need for an integrated system that allows healthcare providers to manage patient information, predict insurance premiums, and maintain real-time updates seamlessly. Leveraging cloud infrastructure (AWS RDS) and ORM frameworks (SQLAlchemy), the solution demonstrates a reliable and efficient approach to enterprise data management. The application incorporates modern machine learning techniques to deliver actionable insights while ensuring data integrity across the system.

1. **Business Use Cases**

* **Insurance Premium Prediction**: Provide patients with personalized insurance premium predictions based on their demographic and health data. The prediction model integrates seamlessly with the backend system, delivering results in real-time.
* **Patient Data Management**: Enable healthcare providers to manage patient information dynamically. The system allows modifications to key attributes while ensuring that critical identifiers remain immutable.
* **User Authentication and Test Case Validation**: Implement secure login functionality and provide pre-configured test cases (e.g., user CID001) for system testing and demonstration.
* **Real-Time Data Update:** Sync modifications in the frontend directly with the AWS backend, ensuring data consistency and system reliability.
* **System Performance Metrics:** Incorporate error metrics like RMSE and MSE to evaluate the predictive model's accuracy, providing insights for model improvement.

1. **System Architecture**

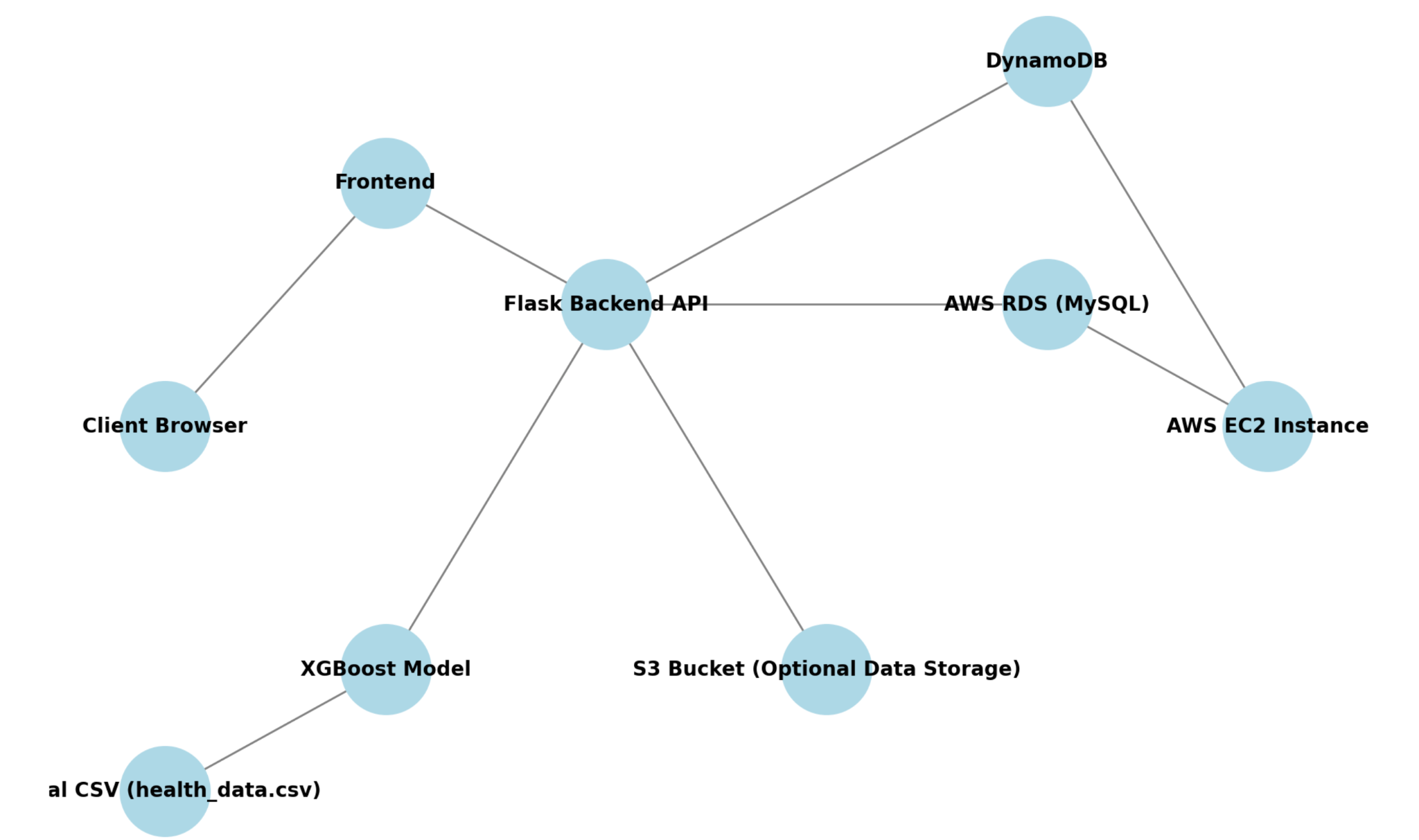


Figure1: The client interacts with the frontend, which communicates with the Flask backend API hosted on an AWS EC2 instance. The backend API handles requests to the AWS RDS for structured data, DynamoDB for unstructured data, and XGBoost for machine learning predictions. Optional data storage is available via an S3 bucket

### **Implementation Details**

#### 1. Database Setup

* Database Engine: AWS RDS MySQL instance was used for structured data storage. AWS DynamoDB table was used for unstructured data storage.
* Tables:
  + CustomerData: Stores patient demographic and account information.
  + UserLogin: Handles user credentials for authentication.
  + Health\_Data: Stores the health check records for each customer with the CID as identifier. Data used in the prediction model.
* Initialization:
  + add-file.py initializes the database tables and populates them with randomly generated data for testing purposes (1,000 users, one fixed test case CID001).

#### 2. Backend API

* Framework: Flask for lightweight and modular REST API development.
* Endpoints:
  + /get\_profile: Fetches user information by CID from the CustomerData table.
  + /update\_profile: Allows updating user details, synchronizing changes directly to the database.
  + /login: Authenticates user credentials from the UserLogin table.
  + /train: Trains an insurance premium prediction model using XGBoost.
  + /predict: Provides real-time insurance premium predictions.
* ORM: SQLAlchemy was used to define models (UserLogin and CustomerData) and handle CRUD operations.
* Error Handling: Exceptions are caught, and meaningful error messages are returned in the API response.

#### 3. Machine Learning Integration

* Data Source: Health\_data unstructured data from AWS DynamoDB contains health metrics for model training.
* Features:
  + Vital signs such as BMI, blood pressure, and pulse rate.
  + Derived feature has\_hypertension based on the primary diagnosis.
* Model:
  + Algorithm: XGBoost regression for predicting monthly\_premium.
  + Metrics: MSE, RMSE, and R² for evaluation.
* Workflow:
  + Split the dataset into training (80%) and test (20%).
  + Train the model and save metrics during /train.
  + Real-time predictions via /predict.

#### 4. Frontend

* Framework: React.js with HTML and JavaScript for user interaction.
* Features:
  + Login interface.
  + Profile management: Update user details and view personal information.
  + Insurance prediction: Users enter health metrics to predict premiums.
* Integration: Frontend communicates with Flask APIs for dynamic data retrieval and updates.

#### 5. DynamoDB Integration

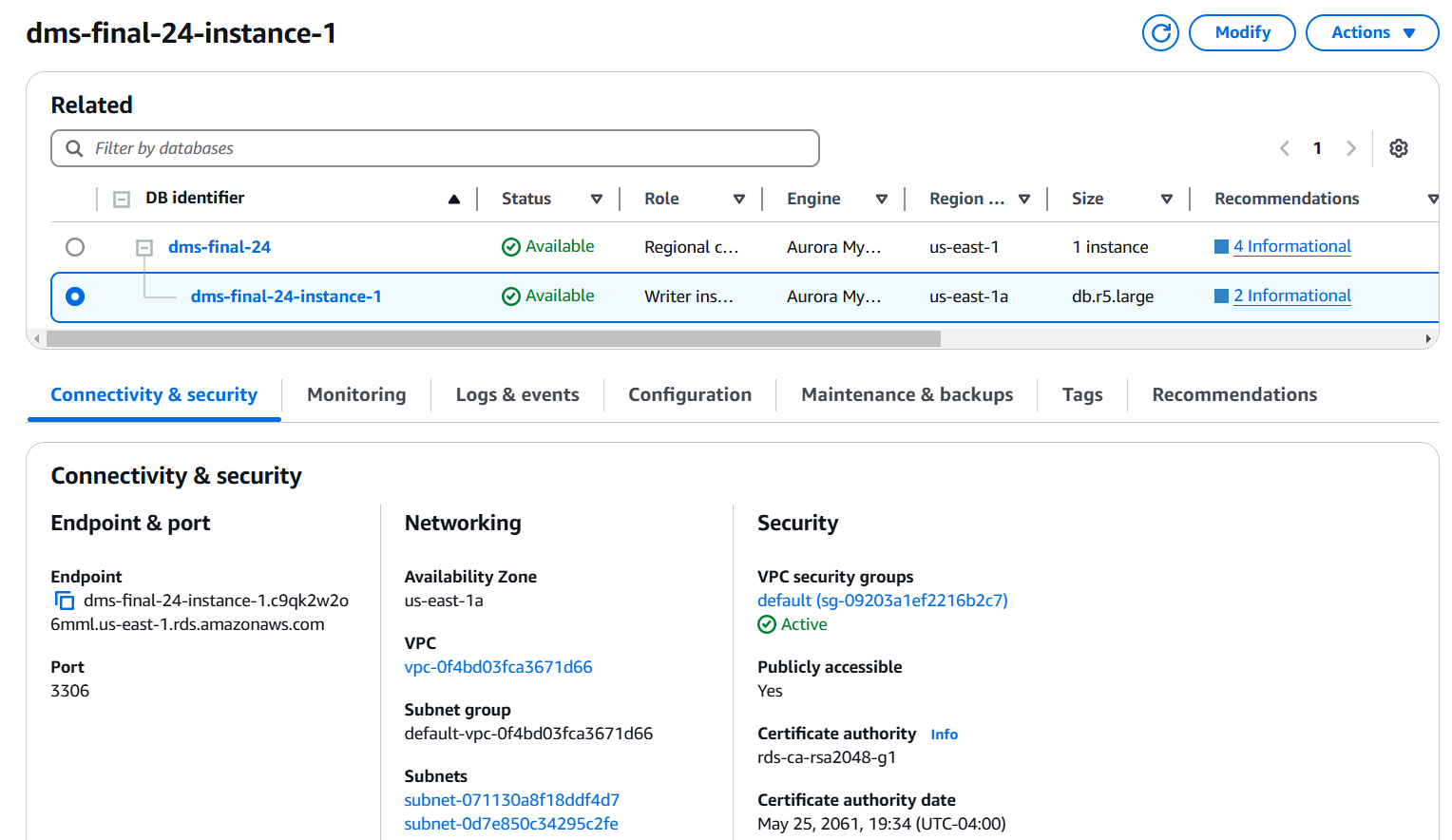
* Purpose: Store unstructured data like health records, lab reports, and medical visits.
* Table Design:
  + Table: MedicalRecords with attributes documentId, customerId, type, and date.
  + Secondary Indexes: CustomerTypeIndex and TypeDateIndex for faster querying.
* Records:
  + Health Checkup, Medical Visit, and Lab Report samples were added.
  + Data includes structured and nested information like vitals, diagnoses, and lab test results.

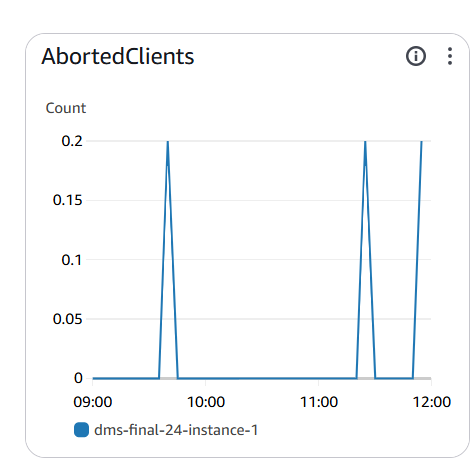
#### 6. Deployment

* Environment:
  + Hosted backend on an AWS EC2 instance for stable API access.
  + AWS RDS and DynamonDB as the backend database for persistent storage.
  + DynamoDB for document-based medical records.
* API Testing:
  + Tools: Postman for manual testing.
  + Automated: Test cases for all endpoints, covering valid and invalid inputs.

### **Demo**

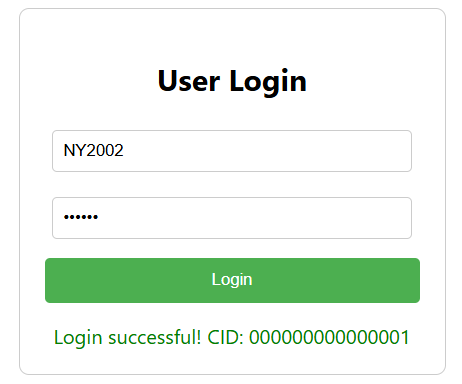
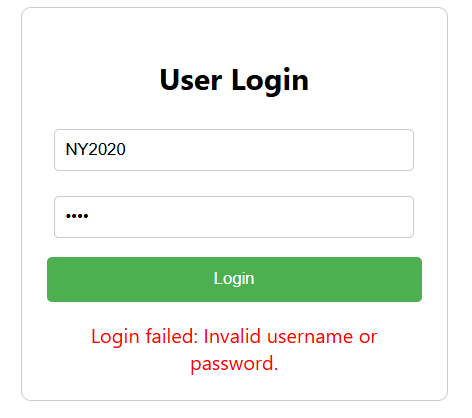
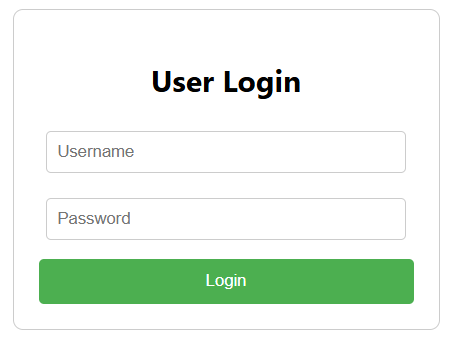
RDS datatable:



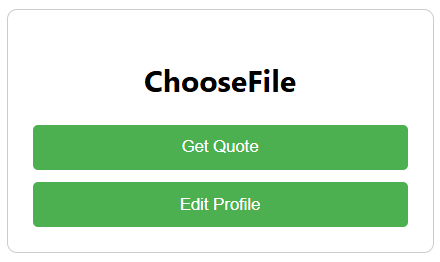


Frontend Workflow:

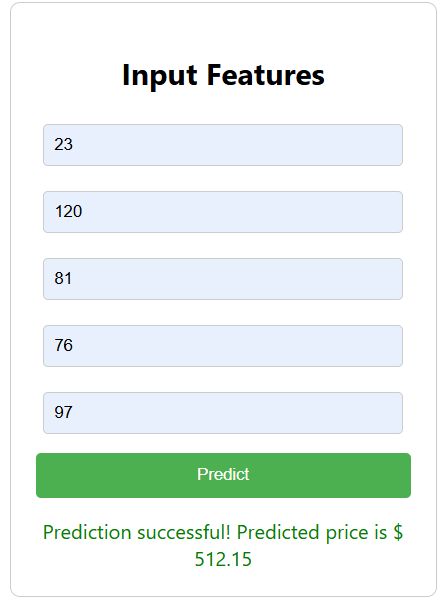
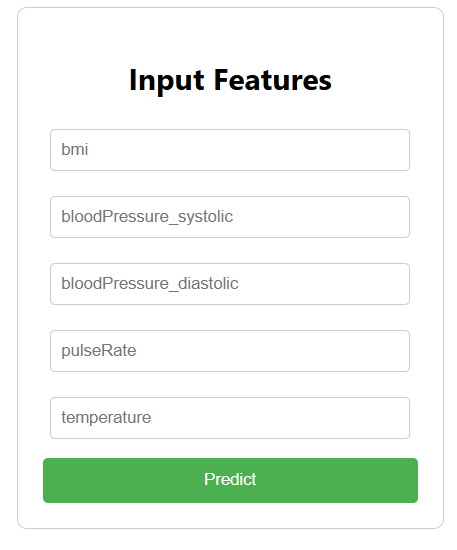
1. Login Page



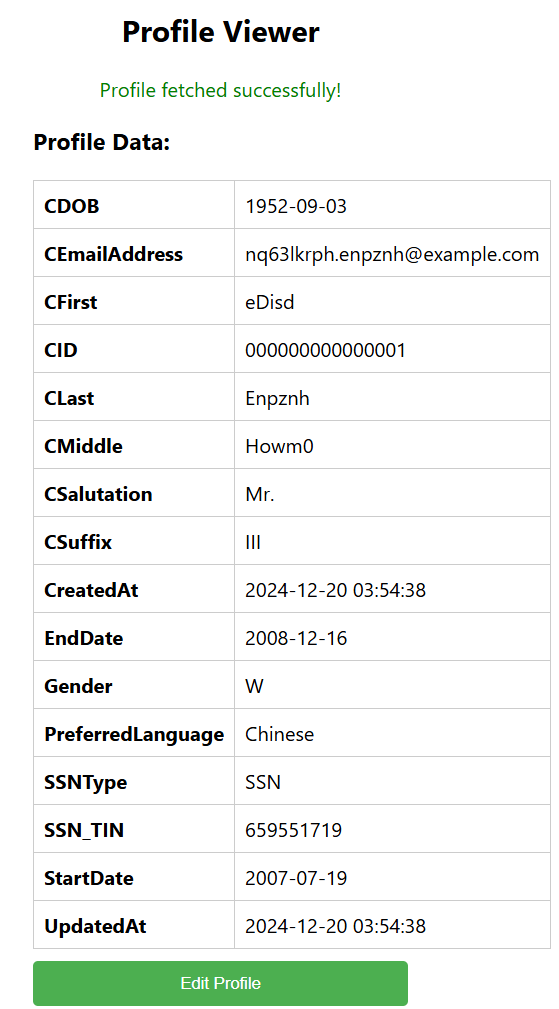
1. File Upload



1. Feature Input



1. ML Prediction Output



1. Profile Edit

