Architecture Document

Application

The project will employ a microservices architecture with event-driven and serverless components to ensure scalability, flexibility, and efficient resource utilization. This approach allows for independent development, deployment, and scaling of different features within the application.

Microservices Architecture

It outlines the use of microservices in building scalable and modular systems for CVD detection. Each microservice is responsible for a specific function such as data collection, feature selection, or model inference, enabling independent development, deployment, and scaling. This modularity enhances system flexibility and allows for easier updates or integration of new features without disrupting the entire application.

Event-Driven Architecture

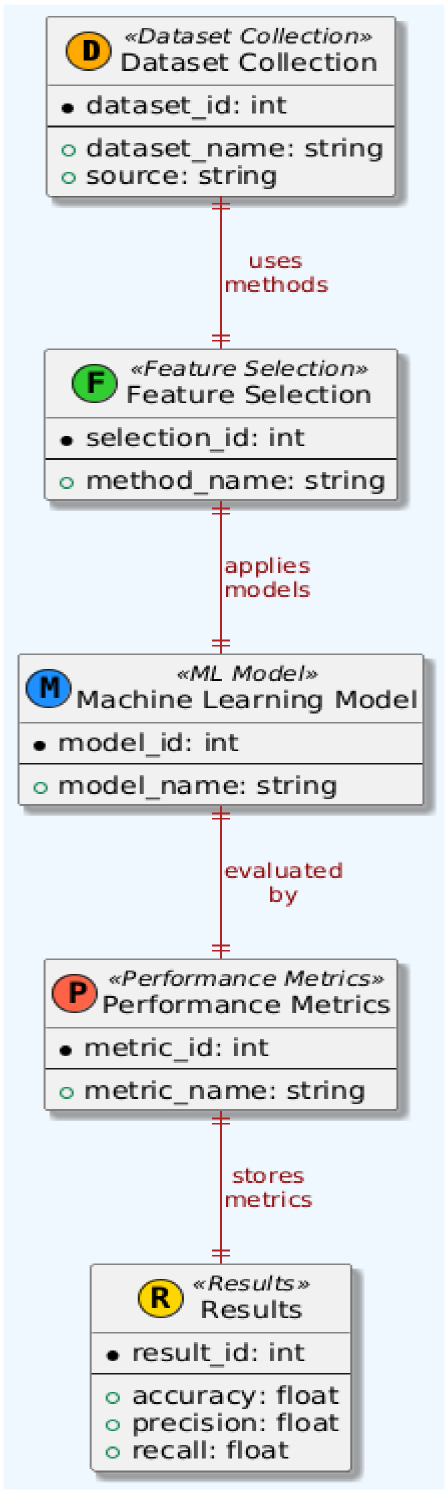
Event-driven architecture (EDA) is leveraged for real-time processing of patient data and triggering of machine learning models. In the context of CVD detection, events such as new patient data inputs or sensor readings can trigger the execution of specific microservices. This allows the system to process and respond to incoming data streams efficiently, supporting timely detection and decision-making.

Serverless Computing

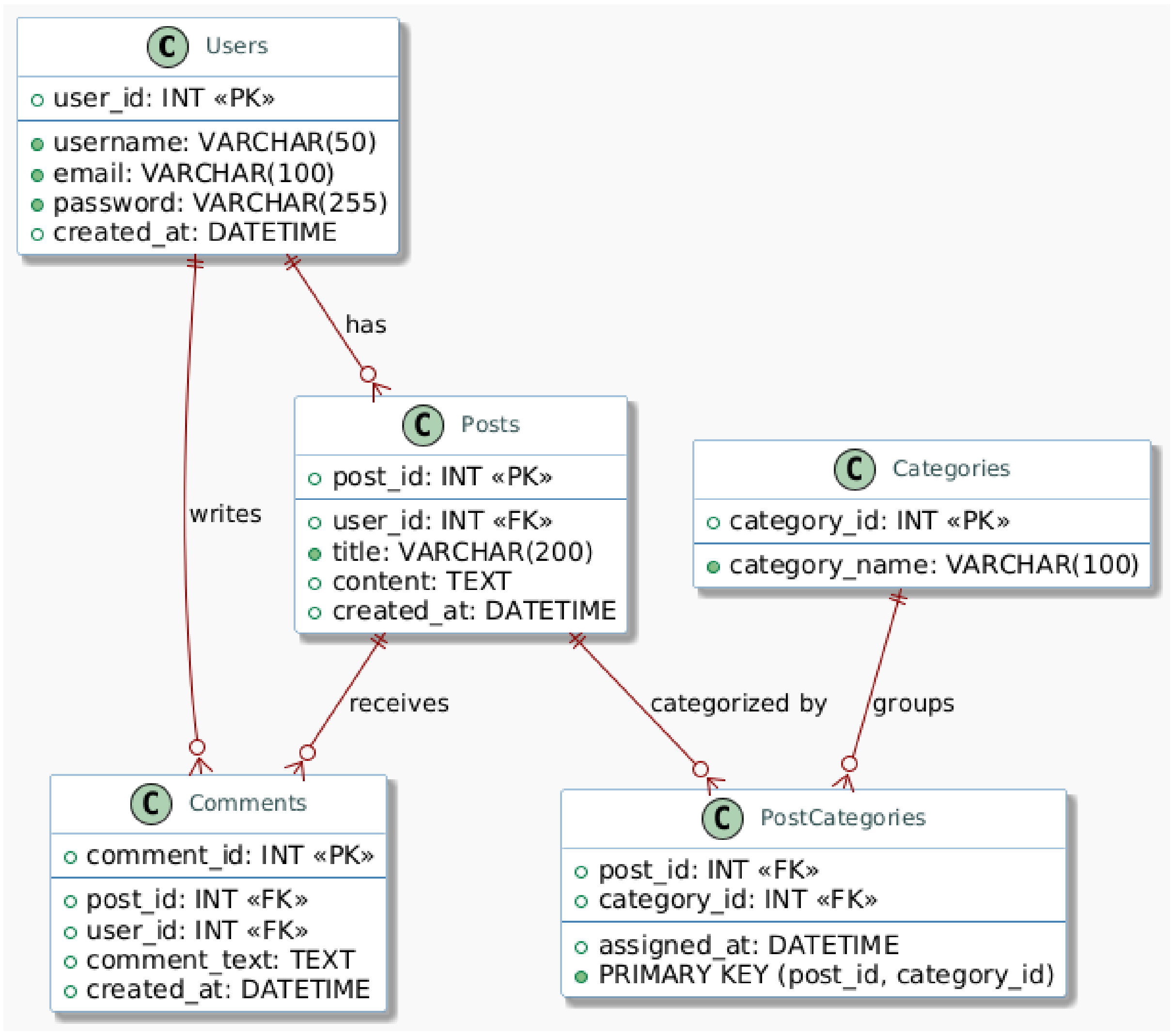
Serverless computing is employed to handle on-demand execution of machine learning models and data processing tasks without the need for managing underlying infrastructure. This approach is particularly useful in handling varying loads, such as spikes in data processing requests from wearable devices or health monitoring systems. Serverless platforms like AWS Lambda allow the system to automatically scale in response to these demands, providing a cost-effective and highly scalable solution for CVD detection.

Database

1.ER Diagram


D
«Dataset Collection»
Dataset Collection
● dataset_id: int
○ dataset_name: string
○ source: string
uses methods
F
«Feature Selection»
Feature Selection
● selection_id: int
○ method_name: string
applies models
M
«ML Model»
Machine Learning Model
● model_id: int
○ model_name: string
evaluated by
P
«Performance Metrics»
Performance Metrics
● metric_id: int
○ metric_name: string
stores metrics
R
«Results»
Results
● result_id: int
○ accuracy: float
○ precision: float
○ recall: float

2.Schema Design



Frequency of Data Exchanges

This section specifies how often data will be exchanged between parties. It includes:

- Daily: Data is exchanged every day, at a specified time, or at regular intervals throughout the day.
- Weekly: Data is exchanged once a week, on a specified day and time.
- Monthly: Data is exchanged once a month, on a specified date and time.
- On-Demand: Data is exchanged as and when required, based on a request from one of the parties.
- Real-Time: Data is exchanged continuously or in near real-time as updates occur.
- Batch Processing: Data is collected over a period (daily, weekly) and exchanged in batches at the end of the period.

Data Sets

This section describes the types of data that will be exchanged, including:

- Data Categories: Define the categories of data to be exchanged, such as customer information, transaction data, financial records, etc.
- Data Format: Specify the format in which the data will be exchanged, such as CSV, XML, JSON, or a proprietary format.
- Data Fields: List the specific fields or attributes that will be included in each dataset, like name, address, account number, transaction amount, etc.
- Data Volume: Estimate the size or volume of data to be exchanged, such as the number of records or the size in megabytes/gigabytes.
- Data Sensitivity: Indicate whether the data is sensitive or confidential, which may require additional security measures.

3. Mode of Exchanges (API, File, Queue, etc.)

This section defines the technical method through which data will be exchanged:

- **API (Application Programming Interface)**: Data is exchanged through a set of defined APIs. The contract should specify:
  - API endpoints
  - Request/response formats (e.g., REST, SOAP)
  - Authentication and authorization mechanisms (e.g., OAuth, API keys)
  - Error handling and retry mechanisms
- **File Transfer**: Data is exchanged through file transfers, which may include:
  - File transfer protocols (e.g., FTP, SFTP, FTPS)
  - Encryption methods (e.g., PGP, SSL/TLS)
  - File naming conventions
  - Schedule and location for file drop-off/pick-up
  - Handling of large files (e.g., chunking, compression)
- **Message Queue**: Data is exchanged through message queuing systems like:
  - Queue technology (e.g., RabbitMQ, Kafka, Azure Service Bus)
  - Message formats

    - Queue configuration (e.g., FIFO, Pub/Sub)
    - Retry and failure handling
    - Security considerations for message queuing
- **Email**: Data is exchanged via email attachments:
    - Email encryption and security
    - File attachment limits
    - Secure email transfer methods (e.g., S/MIME)
- **Direct Database Connection**: Data is exchanged via direct access to the database:
    - Database type (e.g., SQL, NoSQL)
    - Access credentials and roles
    - Query limitations and performance considerations
    - Security measures (e.g., VPN, SSL)
- **Cloud-Based Storage**: Data is exchanged via cloud storage services:
    - Cloud provider and storage type (e.g., AWS S3, Azure Blob)
    - Access methods (e.g., signed URLs, direct API access)
    - Security and encryption
    - Backup and recovery protocols