Vinayak Rao
CS426
Professor Richards
November 12th, 2025

Project Title: Home Health Monitoring System (HHMS)

Purpose:

- Problem: In the United States, the growing elderly population is driving increased need for long term patient care. A large sector of long term care is home care, where a patient remains at their residence and is supported by a family member or caretaker. Home care is beneficial because it reduces the load of health institutions like nursing homes while letting patients remain in a familiar environment; however, home care patients, especially those with chronic conditions, require continuous monitoring of their vital signs. Since they live at home, this cannot be done by medical professionals.

  TLDR: The number of patients in long term home care in the US is growing, and there is a need for a vital sign monitoring system.

- Solution: Introducing the Home Health Monitoring System, colloquially referred to as HHMS. HHMS continuously collects health data from patients' devices, analyzes that data for anomalies, and alerts medical professionals when something is wrong.

- Inspiration: During HackUMass 2025, my team worked on healthNET, an application that enables hospitals to train machine learning models on multiple datasets without directly sharing data. That project made me realize that there are a lot of applications in healthcare that require a scalable web application. Also, during my last internship, I built an anomaly detection system and am passionate about such projects.

- Suitability: This project is very suitable for this course. First, it is a solution that inherently requires scalability. In order to ensure the system can accommodate thousands if not millions of patients, it is necessary to design a scalable architecture. Second, this project requires a database, backend analysis, and communication between multiple services, which will test core course concepts and provide me with the right amount of challenge.

Service Boundaries:

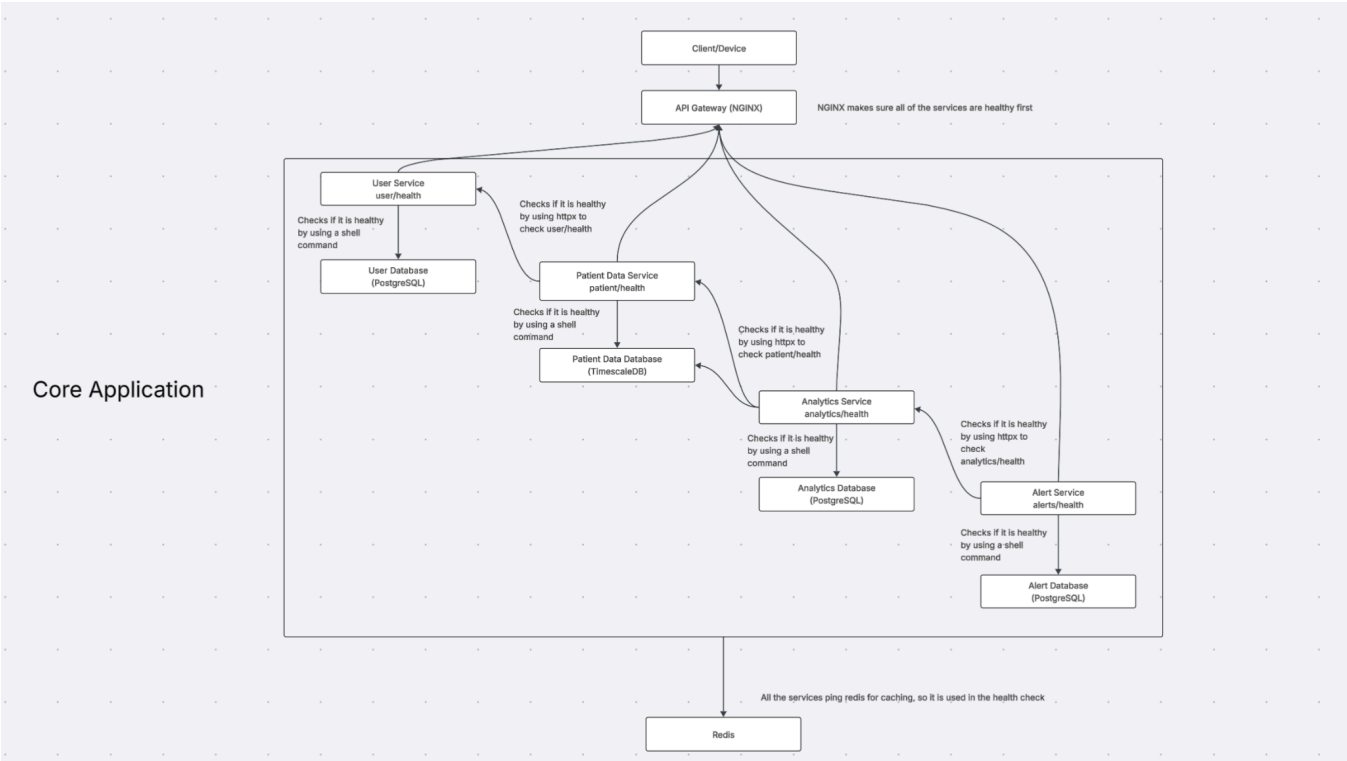
1. User Service:
    a. Purpose: Handles the core identity and relationship information regarding the users of the system, which are the patients and doctors. This service will handle registering users and obtaining information about them.
    b. This exists as a logical, separate service in order to couple the users of the system into a unified model, especially since patients and doctors would be associated with each other.
    c. Example Routes:
        i. POST /users/register
        ii. POST /users/{user_id}
2. Patient Data Service:
    a. Purpose: Handles the ingestion of all vital signs and time-series health data from patient devices.
    b. This exists as a logical, separate service because the volume of patient data is much larger than the volume of user data, and the patient data service is very write heavy. Thus, it will require very different scaling than the user service. Additionally, the patient data collected may change and should be separate from the users themselves.
    c. Example Routes:
        i. POST /patient/{patient_id}
        ii. GET /patient/{patient_id}/latest
3. Analytics Service:
    a. Purpose: Handles the analytics of patient data to detect anomalies
    b. This exists as a logical, separate service because it is computationally intensive and will require a very different approach to scalability than data ingestion.
    c. Example Routes:
        i. POST /analytics/run
4. Alerting Service:
    a. Purpose: Handles alerting via APIs like Twilo
    b. This exists as a logical, separate service because it is logically separate from data ingestion and analysis and should be able to scale separately.
    c. Example Routes:
        i. GET /alerts/{patient_id}
        ii. POST /alerts/send/{patiend_id}
5. Redis:
    a. Purpose: Serves as an event bus between data ingestion and analytics. Also serves as a cache for frequent queries.
6. NGINX:

a. Purpose: Serves as an API gateway and load balancer for scalability

Data Flow:

1. User Service:
    a. The User Service depends on redis and its postgreSQL server to be set up. Thus, its health checkpoint will check database connectivity via psycopg2 and asyncpg.
2. Alerting Service:
    a. The Alerting Service depends on redis and its postgreSQL server to be set up. Thus, its health checkpoint will check database connectivity via psycopg2 and asyncpg.
    b. The Alerting service intuitively requires the Analytics Service, since alerts are created based on analytics. Thus, the health check for the alerting service will use httpx to check the analytics service.
3. Analytics Service:
    a. The Analytics Service depends on redis and its postgreSQL server to be set up. Thus, its health checkpoint will check database connectivity via psycopg2 and asyncpg.
    b. The Analytics Service intuitively requires the patient data service, which handles all of the patient data. Thus, the health check for the analytics service will use httpx to check the patient data service.
4. Patient Data Service:
    a. The Patient Data Service depends on redis and its TimescaleDB server to be set up. Thus, its health checkpoint will check database connectivity via psycopg2 and asyncpg.
    b. The patient data service intuitively depends on the user service because all the patient data must be associated with an existing patient. Thus, the health check for the patient data service will use httpx to check the patient data service.

Diagram:


Core Application
Client/Device
API Gateway (NGINX)
NGINX makes sure all of the services are healthy first
User Service
user/health
Checks if it is healthy
by using a shell
command
User Database
(PostgreSQL)
Checks if it is healthy
by using httpx to
check user/health
Patient Data Service
patient/health
Checks if it is healthy
by using a shell
command
Patient Data Database
(TimescaleDB)
Checks if it is healthy
by using httpx to
check patient/health
Analytics Service
analytics/health
Checks if it is healthy
by using a shell
command
Analytics Database
(PostgreSQL)
Checks if it is healthy
by using httpx to
check
analytics/health
Alert Service
alerts/health
Checks if it is healthy
by using a shell
command
Alert Database
(PostgreSQL)
All the services ping redis for caching, so it is used in the health check
Redis

Technology Stack:

1. FastAPI:
    a. FastAPI will serve as the core web framework due to my familiarity with FastAPI and simplicity.
2. PostgreSQL:
    a. PostgreSQL will serve as the database for most of the microservices. PostgreSQL is open-source and the most popular open-source relational database. The data for my project is highly structured, so a relational database is suitable.
3. TimescaleDB:
    a. TimescaleDB will serve as the database for patient health data. It is an extension of PostgreSQL that is specialized for high-performance time-series data management, which makes it suitable for health data that is a time series and is ingested frequently.
4. Redis:
    a. Redis will serve as the message broker and cache. I have familiarity with Redis through the course and it is an extremely popular service for caching.
5. NGINX:
    a. NGINX will serve as the API gateway and load balancer. My application will require a lot of scaling, and NGINX does a great job of scaling docker containers.
6. Docker and Docker Compose:
    a. Docker and Docker Compose will be used for containerization and orchestration.
7. Twilio
    a. Twilio will be used as the alerting API. Twilio is robust and also offers free credits. Twilio also has both SMS and email alerting, which is suitable for my application.