System Design Document

Project Name: EpiReady

Introduction

This document outlines the system design for the project EpiReady in Sprint 1. The system can now register and login users as well as report shipments and alerts.

System Overview

The main features implemented in Sprint 1 consists of:

- User Authentication: Users can login/register to manage their shipments/alerts.
- Shipment Management: Users can add new shipments, and view the statuses of their shipments, as well as delete and edit shipments.
- Alert Management: Users can view and discard alerts related to their shipments

CRC Cards

Class Name: ShowShipments (/epiready-frontend/src/ShowShipments.jsx)	
Subclasses: AddShipment, ShipmentCard Parent Class: N/A	
Responsibilities: - Get shipments from the backend that user created - Shows shipment cards that hold the brief overview of the shipment - Displays an add button that lets users add shipment.	Collaborators: - Shipment

Class Name: ShowLogs (/epiready-frontend/src/components/Logs/ShowLogs.jsx)	
Subclasses: AlertLog Parent Class: N/A	
Responsibilities: - Contains the Alert logs of dummy data.	

Class Name: ShipmentCard (/epiready-frontend/src/components/shipment/ShipmentCard.jsx)

Subclasses:
Parent Class: ShowShipments

Responsibilities:
- Shows overview of data provided by the backend in a presentable format.

Class Name: AlertLogs (/epiready-frontend/src/components/Logs/ActionLogs.jsx)

Subclasses: NA
Parent Class: ShowLogs

Responsibilities:
- Shows Alert messages of dummy data and lets users delete them.

Class Name: Alert (/epiready-backend/models/alert.py)

Subclasses: N/A
Parent Class: N/A

Responsibilities:
- Get alerts from backend that are triggered based on weather conditions
- Return them to the frontend and update in real time
- Send users real time alerts to SMS and email

Collaborators:
- ActionLog
- Shipment
- /epiready-backend/config/database.py

Class Name: ActionLog (/epiready-backend/models/alert.py)

Subclasses: N/A
Parent Class: N/A

Responsibilities:
- Represents an action taken in response to an alert, mapping to the action_logs table.
- Stores details about the action: which alert it belongs to, type, status, details and time of creation and completion

Collaborators:
- Alert
- /epiready-backend/config/database.py

- Automatically updates time of completion when status set to completed	

Class Name: Shipment (/epiready-backend/models/shipment.py)	
Subclasses: N/A Parent Class: N/A	
Responsibilities: - Represents shipments, maps to shipments table - Stores details about shipment including user id, product type, origin destination, etc. - Automatically updates time of arrival when status is set to completed	Collaborators: - User - Alert - TemperatureData - /epiready-backend/config/database.py

Class Name: TemperatureData (/epiready-backend/models/temperature.py)	
Subclasses: N/A Parent Class: N/A	
Responsibilities: - Represents reading of temperature, maps to temperature_data table - Stores sensor id, temperature, time, location and associated shipment id.	Collaborators: - Shipment - /epiready-backend/config/database.py

Class Name: User (/epiready-backend/models/user.py)	
Subclasses: N/A Parent Class: N/A	
Responsibilities: - Represents reading of temperature, maps to temperature_data table - Stores sensor id, temperature, time, location and associated shipment id.	Collaborators: - Shipment - /epiready-backend/config/database.py

Class Name: WeatherData (/epiready-backend/models/weather.py)

Subclasses: N/A Parent Class: N/A	
Responsibilities: - Represents weather data, maps to weather table - Stores location, temperature, humidity, aqi and time when the data was collected.	Collaborators: - /epiready-backend/config/database.py

System Interaction with the Environment

Operating System:
The app requires a development environment that supports Node.js, Flask, and React.

Technology Stack:
Frontend: React + Vite
Backend: Flask
Database: PostgreSQL

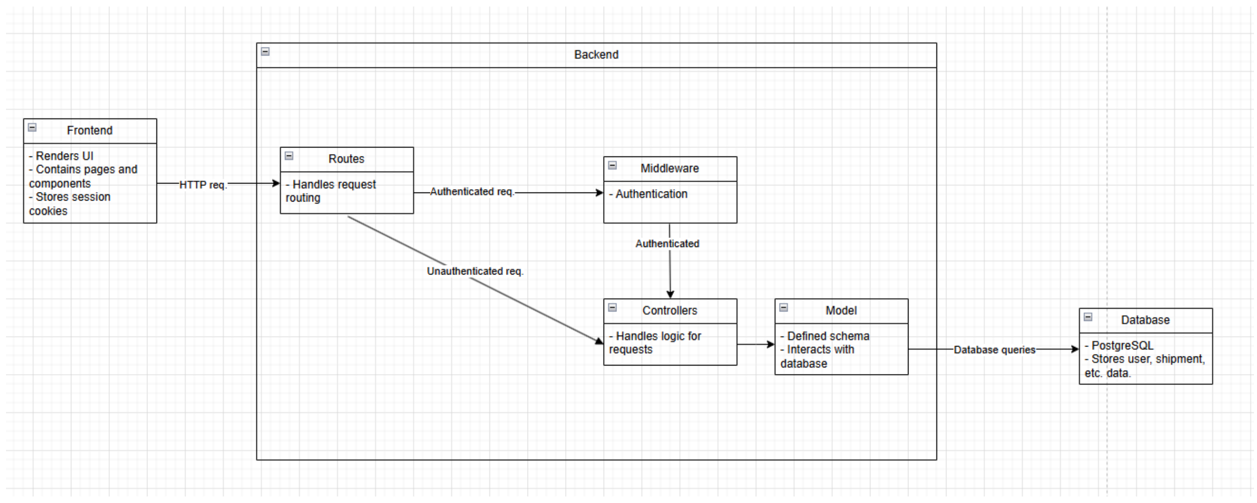
System Architecture

Frontend:
- Handles the UI and renders pages for the user to interact with.
- Sends HTTP requests to the backend through fetch to retrieve information on shipments, alerts, etc.
- Provides certain interactions based on whether the user is signed in or not.
    - Only logged-in users can manage or view their shipments and receive real-time alerts

Backend:
- Authenticates users.
- Stores and retrieves information by interacting with the database.
- Serves API responses.

Database:
- Stores user data, shipment data, alerts, etc.
- Provides information to the backend.

Architecture Diagram:



System Decomposition

Frontend:

- Pages:
  - Home (Landing) Page: Displays dashboard, logo, and recent alerts for all users.
  - Login Page: Allows users to log in, uses session cookies
  - Signup Page: Allows new users to register.
  - Monitor/Shipments/Track Pages: For logged-in users to view, create, and track shipments, monitor conditions, and see shipment logs.
  - Alerts Page: Shows real-time alerts and action logs.
  - Credits Page: Displays attributions and credits.
  - Navbar Component: Handles navigation and highlights the current page.
  - ActionLog/Shipment Components: Modular components for displaying logs and shipment cards.

Backend:

- Controllers:
  - Auth Controller: Handles user registration, login, and session cookie generation.
  - Shipment Controller: Handles CRUD operations for shipments.
  - User Controller: Handles user profile and management.
  - Alert Controller: Handles alert creation and retrieval.
- Middleware:
  - Authentication Middleware: Verifies bearer token on protected routes (e.g., shipment creation, alert access).
- Models:
  - User Model: Defines schema for users and manages user data.
  - Shipment Model: Defines schema for shipments and manages shipment data.
  - Alert Model: Defines schema for alerts and manages alert data.
  - Temperature/Weather Models: For storing sensor and weather data.

Database:

- The database is "epiready".
- Collections within the database:
  - Users: Stores hashed passwords, emails, and user profile data.
  - Shipments: Stores shipment details, status, and related metadata.
  - Alerts: Stores alert messages, types, timestamps, and related shipment/user references.
  - Logs: Stores action logs for auditing and monitoring.
  - Weather/Temperature: Stores weather and sensor readings for shipments.

Error Handling

Invalid user input during register:

- Password or email missing:
  - Backend (auth.py): Returns 400 - Bad Request with message "Please type both email and password as they are both required."
  - Frontend (signup.jsx, AuthenticationForm.jsx): Displays error message from backend.
- Email already registered:
  - Backend: Returns 400 - Bad Request with message "This email address is already registered as a user."
  - Frontend: Displays error message from backend.

Shipment creation errors:

- Missing required fields:
  - Backend (shipment.py): Returns 400 - Bad Request with message "Missing fields: ...".
  - Frontend (AddShipment.jsx): Displays "You must fill all fields in order to submit the shipment".
- Unauthorized shipment access:
  - Backend: Returns 401 - Unauthorized if invalid session.
  - Frontend: Displays "Please log in to access the shipments".

Database Errors:

- Database commit/connection failure:
  - Backend: Returns 500 - Internal Server Error with error message.
  - Frontend: Displays "Something unexpected happened. Please try again later" (from AddShipment.jsx, signup.jsx, login.jsx).

Uncaught Errors:

- Unexpected errors:
  - Backend: Catches exceptions in controllers, returns 500 - Internal Server Error with error message.