Project information

Created on
March 01, 2026

Name	Last update
backend	5 minutes ago
frontend	53 minutes ago
.gitignore	1 week ago
README.md	4 minutes ago
docker-compose.yml	53 minutes ago
load_test.sh	5 minutes ago
nginx.conf	1 month ago

README.md

HomeAssistant Dashboard

A distributed HomeAssistant middleware application with a modern GUI for monitoring metrics and receiving notifications. Built as a DSys Challenge Project with a full-stack implementation.

🎯 Overview

HomeAssistant Dashboard is a middleware application designed to:

- Display various system metrics and HomeAssistant data in a centralized dashboard
- Provide user authentication and secure data access
- Send notifications when metrics fall outside acceptable ranges
- Offer a responsive web interface for real-time monitoring
- Support distributed deployment with load balancing

🛠️ Tech Stack

Backend

- **Framework**: FastAPI (Python 3.12+)
- **Database**: PostgreSQL
- **Authentication**: JWT (JSON Web Tokens)
- **ORM**: SQLAlchemy
- **Server**: Uvicorn

Frontend

- **Framework**: React 19 with TypeScript
- **Build Tool**: Vite
- **Routing**: React Router
- **Styling**: CSS

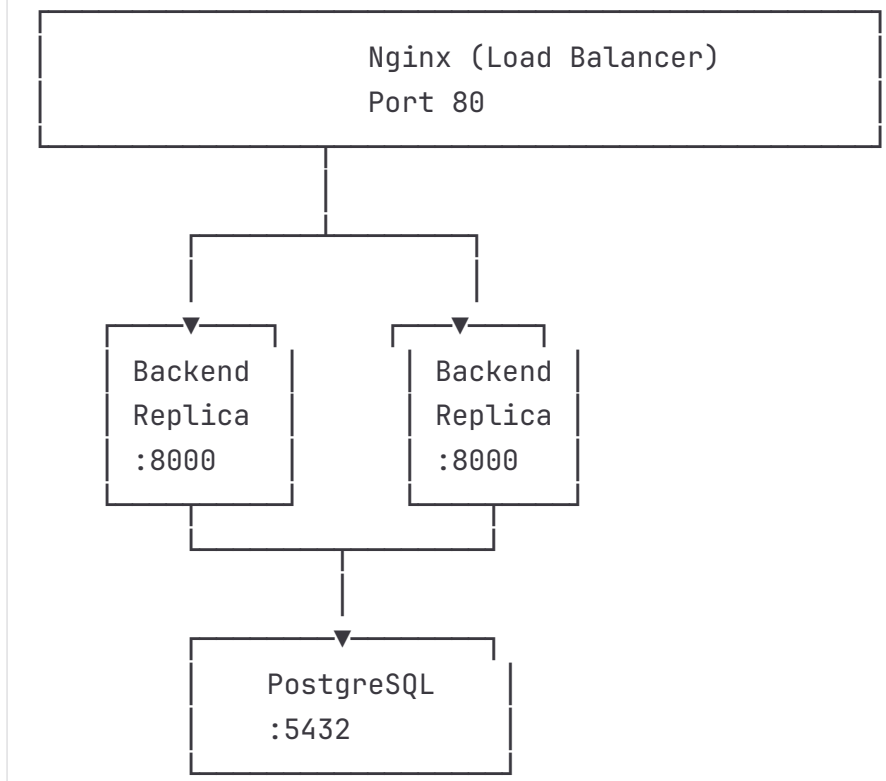
Infrastructure

- **Containerization**: Docker & Docker Compose

- **Load Balancing**: Nginx (Alpine)
- **Database**: PostgreSQL 18

🏗️ Architecture

The application follows a distributed microservices architecture:



Frontend served by Backend on port 80 (via Nginx)

Key Features:

- **Load Balancing**: Nginx distributes traffic across 2 backend replicas
- **Database**: PostgreSQL with health checks
- **Stateless Backend**: Multiple instances can be scaled horizontally
- **Container ID Injection**: Each response includes the container ID for debugging

📦 Prerequisites

- Docker & Docker Compose (or local development):
  - Python 3.12+
  - Node.js 18+
  - PostgreSQL 18
- Git

🚀 Setup & Installation

Using Docker Compose (Recommended)

1. **Clone the repository**

```
git clone <repository-url>
cd homeassistant-dashboard
```

2. **Create environment file**

```
cp .env.example .env
```

Configure the `.env` file with:

```
DB_USER=homeassistant
DB_PASS=your_secure_password
DB_NAME=homeassistant_db
DB_PORT=5432
JWT_SECRET_KEY=your_jwt_secret_key_here
```

3. **Build and start services**

```
docker-compose up -d
```

4. **Verify services are running**

```
docker-compose ps
```

Local Development Setup

Backend Setup

```
cd backend
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -e .
```

Frontend Setup

```
cd frontend
npm install
```

🏃 Running the Project

With Docker Compose

```
# Start all services
docker-compose up -d

# View logs
docker-compose logs -f

# Stop services
docker-compose down

# Stop and remove volumes
docker-compose down -v
```

Local Development

Backend

```
cd backend
source venv/bin/activate
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

Backend API will be available at: `http://localhost:8000`

Frontend

```
cd frontend
npm run dev
```

Frontend will be available at: `http://localhost:5173` (default Vite port)

📁 Project Structure

```
homeassistant-dashboard/
├── backend/
│   ├── app/
│   │   ├── __init__.py          # App initialization
│   │   ├── main.py              # FastAPI application & endpoints
```

```
│   │   ├── models.py            # SQLAlchemy models (User, etc.)
│   │   ├── database.py          # Database configuration
│   │   ├── security.py          # JWT & password handling
│   │   └── ...
│   ├── Dockerfile               # Backend container definition
│   └── pyproject.toml           # Python dependencies
│
├── frontend/
│   ├── src/
│   │   ├── main.tsx             # React entry point
│   │   ├── App.tsx              # Main App component
│   │   ├── App.css              # Styles
│   │   ├── config.ts            # Frontend configuration
│   │   ├── components/
│   │   │   ├── Dashboard.tsx     # Main dashboard view
│   │   │   ├── Login.tsx         # Login page
│   │   │   ├── Register.tsx      # Registration page
│   │   │   └── Monitor.tsx       # Metrics monitoring
│   │   ├── api/
│   │   │   └── client.ts        # API client utilities
│   │   └── assets/
│   ├── Dockerfile               # Frontend container definition
│   ├── package.json             # Node dependencies
│   ├── vite.config.ts           # Vite configuration
│   └── tsconfig.json            # TypeScript configuration
│
├── docker-compose.yml           # Docker Compose orchestration
├── nginx.conf                   # Nginx configuration
├── .env                         # Environment variables (create locally)
├── .env.example                 # Environment template
└── README.md                    # This file
```

🔌 API Endpoints

Authentication

- `POST /login` - User login (returns access & refresh tokens)
- `POST /register` - User registration
- `POST /refresh` - Refresh access token

User Management

- `GET /users/me` - Get current user info (requires auth)
- `GET /users/{user_id}` - Get specific user details

Metrics & Monitoring

- `GET /metrics` - Get current metrics (requires auth)
- `GET /metrics/history` - Get historical metrics
- `POST /notifications` - Create alert notification
- `GET /notifications` - Get user notifications

Note: Authentication uses Bearer tokens in the `Authorization` header:

```
Authorization: Bearer <access_token>
```

🎨 Frontend Components

Dashboard (`Dashboard.tsx`)

- Main view displaying real-time metrics
- Shows container IDs for distributed debugging
- Responsive layout for various screen sizes

Login (`Login.tsx`)

- User authentication form

- JWT token storage
- Redirect to dashboard on success

Register (`Register.tsx`)

- New user registration
- Password validation
- Account creation flow

Monitor (`Monitor.tsx`)

- Metrics visualization
- Alert status display
- Historical data charts

⚙️ Environment Variables

Create a `.env` file in the project root:

```
# Database Configuration
DB_USER=homeassistant
DB_PASS=secure_password_123
DB_NAME=homeassistant_db
DB_PORT=5432

# JWT Configuration
JWT_SECRET_KEY=your_super_secret_jwt_key_change_this_in_production

# Backend Configuration (optional)
BACKEND_URL=http://localhost:8000

# Frontend Configuration (optional)
VITE_API_URL=http://localhost:8000
```

⚠️ **Security Note**: Never commit `.env` to version control. Use `.env.example` as a template.

🔧 Development

Backend Development

Running Tests

```
cd backend
pytest
```

Code Formatting

```
black app/
```

Linting

```
flake8 app/
```

Frontend Development

Running Dev Server

```
cd frontend
npm run dev
```

Building for Production

```
npm run build
```

Linting

```
npm run lint
```

Type Checking

```
npm run type-check  # if configured
```

Load Testing

Run the provided load test script:

```
./load_test.sh
```

This simulates multiple concurrent requests to test the distributed load balancing.

🔐 Security Features

- **JWT Authentication**: Secure token-based authentication
- **Password Hashing**: Argon2 password hashing via pwdlib
- **HTTPS Ready**: Configure with SSL certificates in production
- **CORS**: Configurable cross-origin requests
- **Database**: Secure PostgreSQL with parameterized queries

📊 Monitoring & Debugging

Container ID Injection

Each API response includes the container ID in the response body, allowing you to see which backend instance handled your request.

Nginx Load Balancing

Nginx automatically routes requests across backend replicas with:

- Connection timeout: 2s
- Failed upstream detection with fallback
- Health checks via database

PostgreSQL Health Checks

The database service includes health checks to ensure stability:

```
Health Check: pg_isready -U $DB_USER -d $DB_NAME
Interval: 5s
Timeout: 5s
Retries: 5
```

🚀 Deployment

Production Considerations

1. **Environment Variables**: Use secure secret management (AWS Secrets Manager, HashiCorp Vault, etc.)
2. **HTTPS**: Configure SSL certificates with Nginx
3. **Database Backups**: Set up regular PostgreSQL backups
4. **Scaling**: Increase backend replicas in `docker-compose.yml`
5. **Monitoring**: Integrate with monitoring tools (Prometheus, Grafana, etc.)
6. **Logging**: Configure centralized logging (ELK Stack, Loki, etc.)