**# Weather Analytics Dashboard**

A real-time weather data analytics and visualization web application built with Flask, Socket.IO, and Chart.js.

**## 📋 Overview**

This project provides a comprehensive weather monitoring dashboard that collects, stores, and displays weather data for multiple Vietnamese cities in real-time. The application features automatic data collection, real-time updates via WebSocket, interactive charts, and data export capabilities.

**## ✨ Features**

- **\*\*Real-time Weather Data\*\***: Live weather updates for multiple cities

- **\*\*Interactive Dashboard\*\***: Modern, responsive web interface

- **\*\*Data Visualization\*\***: Temperature trends with Chart.js

- **\*\*Automatic Data Collection\*\***: Background service collecting data every 10 minutes

- **\*\*WebSocket Integration\*\***: Real-time updates without page refresh

- **\*\*Data Export\*\***: CSV export functionality for reports

- **\*\*Database Storage\*\***: SQLite database for historical data

- **\*\*Multi-city Support\*\***: Monitors Ho Chi Minh City, Hanoi, Da Nang, Can Tho, and Hue

**## 🏗️ Architecture**

```

├── main.py                 # Flask application entry point

├── requirements.txt        # Python dependencies

├── src/

│   ├── models/            # Database models

│   │   ├── user.py        # User model and database setup

│   │   └── weather.py     # Weather data model

│   ├── routes/            # API routes

│   │   ├── user.py        # User management routes

│   │   └── weather.py     # Weather API endpoints

│   └── services/          # Business logic

│       ├── weather\_service.py    # Weather data collection

│       └── realtime\_service.py   # WebSocket real-time service

├── static/                # Frontend assets

│   ├── index.html         # Main dashboard page

│   └── app.js            # Frontend JavaScript

└── database/             # SQLite database files

```

**## 🚀 Getting Started**

**### Prerequisites**

- Python 3.7 or higher

- pip (Python package installer)

**### Installation**

1. **\*\*Clone the repository\*\***

   ```bash

   git clone <repository-url>

   cd "Webapp for Realtime Weather Data Analytics and Display"

   ```

2. **\*\*Install dependencies\*\***

   ```bash

   pip install -r requirements.txt

   ```

3. **\*\*Run the application\*\***

   ```bash

   python main.py

   ```

4. **\*\*Access the dashboard\*\***

   Open your browser and navigate to: `http://localhost:5002`

**## 🔧 Configuration**

**### Weather API**

The application currently uses mock data for demonstration. To use real weather data:

1. Sign up for a free API key at [OpenWeatherMap](https://openweathermap.org/api)

2. Replace `demo\_key` in `src/services/weather\_service.py` with your actual API key:

   ```python

   self.api\_key = "your\_actual\_api\_key\_here"

   ```

**### Database**

The application uses SQLite by default. The database file is automatically created at `database/app.db` on first run.

**### Cities Configuration**

To modify the monitored cities, edit the `CITIES` list in `src/routes/weather.py`:

```python

CITIES = ["Ho Chi Minh City", "Hanoi", "Da Nang", "Can Tho", "Hue"]

```

**## 📊 API Endpoints**

**### Weather Endpoints**

- `GET /api/weather/current?city=<city\_name>` - Get current weather for a city

- `GET /api/weather/latest?limit=<number>` - Get latest weather records

- `GET /api/weather/city/<city\_name>?limit=<number>` - Get weather data for specific city

- `GET /api/weather/dashboard` - Get dashboard data for all cities

- `POST /api/weather/collect` - Manually trigger data collection

**### User Endpoints**

- `GET /api/users` - Get all users

- `POST /api/users` - Create a new user

**## 🔄 Real-time Features**

The application uses Socket.IO for real-time communication:

**### WebSocket Events**

**\*\*Client → Server:\*\***

- `connect` - Client connection

- `disconnect` - Client disconnection

- `request\_data` - Request specific data type

**\*\*Server → Client:\*\***

- `dashboard\_update` - Dashboard data updates

- `latest\_data\_update` - Latest weather data

- `new\_weather\_data` - New weather record notification

- `status\_update` - System status messages

**## 🎨 Frontend Features**

**### Dashboard Components**

1. **\*\*Statistics Cards\*\***: Total records, cities count, last update time, average temperature

2. **\*\*Current Weather Grid\*\***: Live weather for all monitored cities

3. **\*\*Temperature Chart\*\***: Interactive line chart showing temperature trends

4. **\*\*Data Table\*\***: Detailed weather records with sorting and pagination

5. **\*\*Control Buttons\*\***: Manual data collection, refresh, and export functions

**### Responsive Design**

- Mobile-friendly responsive layout

- Modern glassmorphism design

- Smooth animations and transitions

- Real-time status notifications

**## 🔒 Security Features**

- CORS enabled for cross-origin requests

- SQL injection protection via SQLAlchemy ORM

- Input validation and error handling

- Secure WebSocket connections

**## 📈 Performance**

- Background data collection to avoid blocking main thread

- Efficient database queries with proper indexing

- WebSocket for real-time updates without polling

- Client-side caching for better user experience

**## 🧪 Testing**

To test the application:

1. **\*\*Start the server\*\***

   ```bash

   python main.py

   ```

2. **\*\*Test API endpoints\*\***

   ```bash

   # Test dashboard data

   curl http://localhost:5002/api/weather/dashboard

   # Test data collection

   curl -X POST http://localhost:5002/api/weather/collect

   # Test latest data

   curl http://localhost:5002/api/weather/latest?limit=5

   ```

3. **\*\*Test WebSocket connection\*\***

   Open the dashboard in your browser and check the browser console for WebSocket connection messages.

**## 🐛 Troubleshooting**

**### Common Issues**

1. **\*\*Port already in use\*\***

   - Change the port in `main.py`: `socketio.run(app, host='0.0.0.0', port=5003)`

2. **\*\*Database errors\*\***

   - Delete the `database/app.db` file to reset the database

   - Check file permissions in the database directory

3. **\*\*WebSocket connection issues\*\***

   - Ensure no firewall is blocking the connection

   - Check browser console for error messages

4. **\*\*Unicode encoding errors\*\***

   - Set environment variable: `set PYTHONIOENCODING=utf-8`

**## 📝 Development**

**### Adding New Cities**

1. Add city name to `CITIES` list in `src/routes/weather.py`

2. Update mock data in `weather\_service.py` if using demo mode

3. Restart the application

**### Extending the API**

1. Add new routes in `src/routes/`

2. Register blueprints in `main.py`

3. Update frontend JavaScript if needed

**### Database Schema Changes**

1. Modify models in `src/models/`

2. Delete existing database file for development

3. Restart application to recreate tables

**## 🤝 Contributing**

1. Fork the repository

2. Create a feature branch: `git checkout -b feature-name`

3. Commit changes: `git commit -am 'Add feature'`

4. Push to branch: `git push origin feature-name`

5. Submit a pull request

**## 📄 License**

This project is licensed under the MIT License - see the [LICENSE](LICENSE) file for details.

**## 👨‍💻 Author**

**\*\*Tran The Hao\*\***

**## 🙏 Acknowledgments**

- OpenWeatherMap for weather data API

- Flask community for the excellent web framework

- Chart.js for beautiful data visualization

- Socket.IO for real-time communication

**## 📞 Support**

For support and questions:

- Create an issue in the repository

- Check the troubleshooting section above

- Review the API documentation

---

**\*\*Note\*\***: This application is designed for educational and demonstration purposes. For production use, consider implementing additional security measures, error handling, and performance optimizations.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A graph with colorful lines and numbers

AI-generated content may be incorrect.

A close up of a number

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.