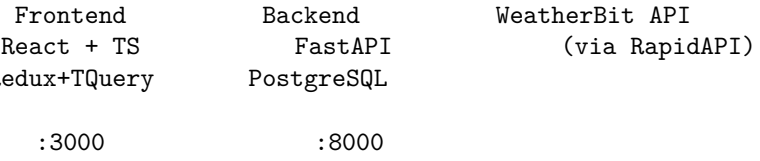
Weather Monitor - Full-Stack Application

A production-grade weather monitoring application built with FastAPI + PostgreSQL (backend) and React 18 + TypeScript (frontend).

System Architecture Overview



Key architectural decisions: - **Clean Architecture** with clear separation: API layer → Service layer → Repository layer → Database - **Repository Pattern** abstracts database access behind interfaces - **Service Layer** encapsulates all business logic - **DTOs (Pydantic schemas)** define strict API contracts between layers - **Dependency Injection** via FastAPI's `Depends()` for testability - **Redux Toolkit** for client state (auth, UI) + **TanStack Query** with **Axios** for all server-state / API calls

---

Backend Folder Structure

```
backend/
  alembic/                  # Database migrations
     versions/
        001_initial_schema.py
     env.py
     script.py.mako
  app/
     api/
        v1/
           endpoints/
              auth.py         # Signup, login, refresh, /me
              weather.py      # Current weather, forecasts
              watchlist.py    # CRUD watchlist
              preferences.py  # User preferences
           router.py
     core/
        config.py          # Pydantic Settings
        dependencies.py    # Auth dependency injection
        exceptions.py      # Domain exceptions
        security.py        # JWT + bcrypt utilities
```

```
    db/
        base.py            # SQLAlchemy DeclarativeBase
        session.py         # Async session factory
    middleware/
        error_handler.py   # Global exception handler
        logging.py         # Request logging
    models/
        user.py
        location.py
        watchlist.py
        preferences.py
    repositories/
        user_repository.py
        location_repository.py
        watchlist_repository.py
        preferences_repository.py
    schemas/               # Pydantic DTOs
        auth.py
        weather.py
        watchlist.py
        preferences.py
    services/
        auth_service.py
        weather_client.py  # HTTP client with retry/timeout
        weather_service.py # Data transformation
        watchlist_service.py
        preferences_service.py
    tests/
        conftest.py        # Fixtures: test DB, client, auth
        unit/
            test_auth_service.py
            test_weather_service.py
            test_watchlist_service.py
        integration/
            test_auth_api.py
            test_watchlist_api.py
            test_preferences_api.py
    main.py
alembic.ini
requirements.txt
pytest.ini
Dockerfile
.env.example
```

Frontend Folder Structure

```
frontend/
  src/
      app/
          store.ts           # Redux store configuration
          hooks.ts           # Typed useDispatch/useSelector
      components/
          ui/                # shadcn/ui components
              button.tsx
              input.tsx
              card.tsx
              skeleton.tsx
              badge.tsx
          layout/
              Header.tsx
              AppLayout.tsx
          auth/
              LoginForm.tsx
              SignupForm.tsx
          weather/
              CurrentWeather.tsx
              DailyForecast.tsx
              HourlyForecast.tsx
              WeatherAlerts.tsx
              SearchBar.tsx
          watchlist/
              WatchlistPanel.tsx
          common/
               ErrorBoundary.tsx
               LoadingSpinner.tsx
               EmptyState.tsx
      features/
          api/
              apiSlice.ts    # (deprecated - see lib/axios.ts)
          auth/
              authSlice.ts   # Auth Redux state
              authApi.ts     # Auth TanStack Query mutations
          weather/
              weatherSlice.ts
              weatherApi.ts
          watchlist/
              watchlistApi.ts
          preferences/
               preferencesApi.ts
      hooks/
```

```
        useGeolocation.ts
        useDebounce.ts
    lib/
        axios.ts           # Axios instance + auth interceptor
        queryClient.ts     # TanStack React Query client
        utils.ts
    pages/
        HomePage.tsx
        SearchPage.tsx
        WatchlistPage.tsx
        LoginPage.tsx
        SignupPage.tsx
    types/
        api.ts
    styles/
        globals.css
    __tests__/
        authSlice.test.ts
        weatherSlice.test.ts
        utils.test.ts
        ErrorBoundary.test.tsx
    App.tsx
    main.tsx
  package.json
  tsconfig.json
  vite.config.ts
  tailwind.config.js
  jest.config.ts
  Dockerfile
  nginx.conf
```

Database Schema

```sql
-- Users table
CREATE TABLE users (
    id          VARCHAR(36) PRIMARY KEY,
    username    VARCHAR(50) UNIQUE NOT NULL,
    hashed_password TEXT NOT NULL,
    role        VARCHAR(20) NOT NULL DEFAULT 'USER',
    is_active   BOOLEAN NOT NULL DEFAULT true,
    created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at  TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX idx_users_username ON users(username);
```

```sql
-- Locations table
CREATE TABLE locations (
    id           VARCHAR(36) PRIMARY KEY,
    city_name    VARCHAR(100) NOT NULL,
    country_code VARCHAR(10),
    latitude     FLOAT,
    longitude    FLOAT,
    created_at   TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX idx_locations_city ON locations(city_name);

-- Watchlist items
CREATE TABLE watchlist_items (
    id          VARCHAR(36) PRIMARY KEY,
    user_id     VARCHAR(36) NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    location_id VARCHAR(36) NOT NULL REFERENCES locations(id) ON DELETE CASCADE,
    added_at    TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CONSTRAINT uq_user_location UNIQUE (user_id, location_id)
);
CREATE INDEX idx_watchlist_user ON watchlist_items(user_id);

-- User preferences
CREATE TABLE user_preferences (
    id              VARCHAR(36) PRIMARY KEY,
    user_id         VARCHAR(36) UNIQUE NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    default_city    VARCHAR(100),
    default_country VARCHAR(10),
    default_lat     VARCHAR(20),
    default_lon     VARCHAR(20),
    units           VARCHAR(10) NOT NULL DEFAULT 'metric',
    updated_at      TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX idx_prefs_user ON user_preferences(user_id);
```

API Endpoints

Method	Endpoint	Auth	Description
POST	/api/v1/auth/signup	No	Create a new account
POST	/api/v1/auth/login	No	Login, get JWT tokens
POST	/api/v1/auth/refresh	No	Refresh access token
GET	/api/v1/auth/me	Yes	Get current user info

Method	Endpoint	Auth	Description
POST	/api/v1/auth/promote-admin	Yes	Promote current user to admin
GET	/api/v1/weather/current	Yes	Get current weather
GET	/api/v1/weather/forecast	Yes	Get daily/hourly/alerts
GET	/api/v1/watchlist	Yes	List saved locations
POST	/api/v1/watchlist	Yes	Add location to watchlist
DELETE	/api/v1/watchlist/{id}	Yes	Remove from watchlist
GET	/api/v1/preferences	Yes	Get user preferences
PUT	/api/v1/preferences	Yes	Update user preferences
GET	/health	No	Health check

Full interactive docs available at `http://localhost:8000/docs` (Swagger UI).

Environment Setup

1. Clone and configure environment

```
cp .env.example .env
# Edit .env with your values:
# - Set RAPIDAPI_KEY (get from https://rapidapi.com/weatherbit/api/weather)
# - Set JWT_SECRET_KEY (generate with: openssl rand -hex 32)
```

2. RapidAPI Key Configuration

1. Go to https://rapidapi.com/weatherbit/api/weather
2. Subscribe to a plan (free tier available)
3. Copy your API key
4. Set `RAPIDAPI_KEY` in your `.env`

Local Development

Backend Setup

```
cd backend

# Create virtual environment
python -m venv venv
source venv/bin/activate  # Linux/macOS
# or: venv\Scripts\activate  # Windows

# Install dependencies
pip install -r requirements.txt
```

```
# Copy env file
cp .env.example .env

# Start PostgreSQL (via Docker)
docker run -d --name weather-pg \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=postgres \
  -e POSTGRES_DB=weather_db \
  -p 5432:5432 postgres:16-alpine

# Start server (tables are created automatically on startup)
uvicorn app.main:app --reload --port 8000

# Optional: run Alembic migrations for incremental schema changes
# alembic upgrade head
```

Frontend Setup

```
cd frontend

# Install dependencies
npm install

# Start dev server
npm run dev
```

Running Tests

```
# Backend tests
cd backend
pytest                              # All tests
pytest app/tests/unit/              # Unit tests only
pytest app/tests/integration/       # Integration tests only
pytest --cov=app --cov-report=html  # With coverage report

# Frontend tests
cd frontend
npm test                            # All tests
npm test -- --coverage              # With coverage
```

Docker (One-Command Startup)

```
# From project root
cp .env.example .env
```

```
# Edit .env with your RAPIDAPI_KEY

docker compose up --build
```

This starts: - **PostgreSQL** on port 5432 - **Backend (FastAPI)** on port 8000 (auto-creates all tables on startup) - **Frontend (React via serve)** on port 3000

Access the app at `http://localhost:3000` API docs at `http://localhost:8000/docs`

Testing Strategy

Backend

Type	What	How
Unit	Password hashing, JWT creation	Direct function calls
Unit	Weather service data transform	Mocked WeatherBit client
Unit	Watchlist business logic	Mocked repositories
Integration	Auth API (signup/login/refresh)	httpx AsyncClient + SQLite
Integration	Watchlist API (CRUD)	httpx AsyncClient + SQLite
Integration	Preferences API	httpx AsyncClient + SQLite

Tests use SQLite in-memory for speed. External API calls are mocked.

Frontend

Type	What	Framework
Slice	authSlice reducers	Jest
Slice	weatherSlice reducers	Jest
Utility	formatTemp, getWeatherBgClass	Jest
Component	ErrorBoundary rendering	React Testing Library

Admin Registration

Admin users see a **data source indicator** on weather cards showing whether data comes from the live WeatherBit API or the mock fallback.

How to become an admin

1. **Sign up** as a regular user (via the app or API):

```
curl -X POST http://localhost:8000/api/v1/auth/signup \
  -H "Content-Type: application/json" \
  -d '{"username": "myadmin", "password": "securepass123"}'
```

2. **Log in** to get your access token:

```
curl -X POST http://localhost:8000/api/v1/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username": "myadmin", "password": "securepass123"}'
```

3. **Promote to admin** using the admin secret key:

```
curl -X POST http://localhost:8000/api/v1/auth/promote-admin \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <YOUR_ACCESS_TOKEN>" \
  -d '{"secret_key": "admin-secret-change-me-in-production"}'
```

4. **Log out and log back in** on the frontend so the UI picks up your new ADMIN role.

Configuration

Set `ADMIN_SECRET_KEY` in `backend/.env` to a secure value:

```
ADMIN_SECRET_KEY=your-strong-secret-key-here
```

Once promoted, the admin badge appears next to your username in the header, and weather cards display a **LIVE API** (green) or **MOCK DATA** (amber) indicator.

Mock Data Fallback

When the WeatherBit API is unavailable (403 Not Subscribed, 429 Rate Limit, 502 Gateway Error, or network timeout), the backend automatically falls back to **dynamically generated mock data**.

How it works: - Mock data produces realistic temperature, humidity, wind, and forecast values - Temperature varies by **latitude** (equator is hotter, poles are colder) and **time of day** (cooler at night, warmer at midday) - Data refreshes every **5 minutes** to simulate real weather changes - 25+ cities are pre-seeded with accurate coordinates; unknown cities get generated values - The `data_source` field in API responses indicates `"live"` or `"mock"` - Admin users see a visual badge on the frontend; regular users see weather normally

No configuration needed — fallback is fully automatic. When the real API becomes available again, live data resumes seamlessly.

Key Design Decisions & Trade-offs

1. **SQLAlchemy 2.0 async** over SQLModel: Better type support and community maturity for production
2. **UUID strings** for primary keys: Portable across databases, no sequence contention
3. **JWT in localStorage**: Standard SPA approach; for higher security, consider httpOnly cookies
4. **SQLite for tests**: Fast execution without Docker dependency; trade-off is slight dialect differences
5. **Single WeatherBit client instance**: Tenacity retry with exponential backoff handles transient failures
6. **TanStack Query + Axios** for server-state: Separates concerns — Redux handles client state (auth, UI), TanStack Query handles server-state (caching, refetching, mutations)
7. **Feature-based folder structure**: Scales better than type-based (all auth files together vs. all reducers together)
8. **shadcn/ui pattern**: Copy-paste UI components for full customization control vs. npm dependency
9. **Graceful degradation**: Alerts endpoint returns empty array on failure instead of blocking the entire response
10. **Multi-stage Docker builds**: Minimal production images (no build tools in runtime)
11. **Auto table creation via lifespan**: `Base.metadata.create_all` runs on startup so no manual migration step is needed for fresh environments; Alembic remains available for incremental schema changes
12. **Mock data fallback**: When the external API fails (403/429/502/timeout), the backend seamlessly returns dynamically generated mock data using latitude-based temperatures and time-of-day cycles; responses include `data_source` field so the frontend can indicate the source
13. **Admin via secret key promotion**: Instead of a separate admin signup flow, any authenticated user can promote themselves using a server-side secret key — simple and secure for single-admin setups