Plan de documentație pentru Aplicația Stației Meteo

1. Introducere și prezentare generală

1.1 Context și motivație

- Prezentarea problemei pe care o rezolvă aplicația
- Importanța monitorizării datelor meteorologice
- Beneficiile unei stații meteo conectate

1.2 Obiective și scopul proiectului

- Colectarea și vizualizarea datelor meteorologice în timp real
- Analiza și vizualizarea avansată a datelor istorice
- Oferirea de interfețe intuitive pentru monitorizare

1.3 Arhitectura generală

- Arhitectura client-server
- Componentele principale ale sistemului
- Diagrama de ansamblu a sistemului

2. Arhitectura hardware

2.1 Platformă și specificații

- Raspberry Pi 4 ca unitate centrală de procesare
- Specificații tehnice (procesor, memorie, conectivitate)
- Avantajele utilizării Raspberry Pi în acest context

2.2 Senzori și echipamente

- Lista completă a senzorilor utilizați
- Specificații tehnice pentru fiecare senzor (model, acuratețe, interval de măsurare)
- Diagrama de conectare a senzorilor la Raspberry Pi

2.3 Alimentare și autonomie

- Sistem de alimentare (surse, baterii)
- Consum energetic
- Strategii pentru autonomie extinsă

3. Arhitectura software

3.1 Prezentare generală

- Structura arhitecturală (backend, frontend, baza de date)
- Tipul arhitecturii (RESTful)
- Diagrama componentelor software

3.2 Backend (Python/Flask)

- Structura aplicației Flask
- API-uri și endpoint-uri
- Procesarea și stocarea datelor
- Managementul autentificării și autorizării

3.3 Frontend (React)

- Arhitectura aplicației React
- Componentele principale
- State management
- Routing și navigare
- Responsive design

3.4 Baza de date

- Structura bazei de date SQLite
- Schema bazei de date (tabele, relații)
- Strategie de stocare și arhivare a datelor

3.5 Comunicare și protocoale

- Protocoale de comunicare (HTTP/HTTPS, WebSockets)
- Format de schimb de date (JSON)
- Securizarea comunicației

4. Funcționalitățile sistemului

4.1 Colectare date

- Senzori și parametri monitorizați
- Frecvența de eșantionare
- Prelucrarea și validarea datelor primare
- Simularea senzorilor pentru testare

4.2 Dashboard

- Vizualizare în timp real a parametrilor
- Reprezentări grafice interactive
- Statistici rapide și alerte

4.3 Vizualizări avansate

- Hărți interactive cu suprapuneri meteo
- Vizualizări 3D pentru parametri meteo
- Integrare cu servicii meteo externe (Windy.com, RainViewer)

4.4 Istoric și statistici

- Vizualizarea și filtrarea datelor istorice
- Generarea de rapoarte statistice
- Exportul datelor în diverse formate

4.5 Sistem de autentificare

- Nivele de acces (utilizator/administrator)
- Managementul utilizatorilor
- Funcționalități specifice administratorului
- Securitatea autentificării

4.6 Asistent meteo

- Interfața conversațională
- Capacitățile de răspuns ale asistentului
- Tehnologiile de inteligență artificială utilizate

5. Implementare și tehnologii

5.1 Backend

- Python 3.x
- Flask și extensii (Flask-JWT-Extended, Flask-Login, Flask-CORS)
- SQLAlchemy ORM
- Biblioteci pentru procesarea datelor (NumPy, Pandas)

5.2 Frontend

- React 18
- Chakra UI pentru interfața utilizator
- React Query pentru gestionarea stării și cererilor
- Biblioteci de vizualizare (react-chartjs-2, leaflet)
- React Router pentru navigare

5.3 Baza de date

- SQLite pentru stocarea locală
- Structura tabelelor
- Indexare și optimizare

5.4 DevOps și deployment

- Automatizare build (scripts)
- Strategii de deployment
- Monitorizare și logging

6. Caracteristici de securitate

6.1 Autentificare și autorizare

- JSON Web Tokens (JWT)
- Hashing-ul parolelor
- Managementul sesiunilor
- Controlul accesului bazat pe roluri

6.2 Securitatea API-urilor

- Validarea input-urilor
- Rate limiting
- Protecția împotriva atacurilor comune (CSRF, XSS)

6.3 Securitatea datelor

- Criptarea datelor sensibile
- Backup și recuperare
- Strategii de retenție a datelor

7. Fluxuri de date și procese

7.1 Fluxul de date pentru colectarea informațiilor

- De la senzori la baza de date
- Procesarea și validarea datelor
- Diagrama fluxului de date

7.2 Fluxul interacțiunii utilizator

- User journeys pentru diferite cazuri de utilizare
- Diagrame de secvență pentru procesele principale

7.3 Procesarea în timp real

- Strategii pentru actualizarea în timp real
- Polling vs WebSockets
- Optimizări pentru performanță

8. Modele de date

8.1 Schema bazei de date

- Structura detaliată a tabelelor
- Relațiile dintre tabele
- Indecși și optimizări

8.2 Modele pentru date meteorologice

- Structura datelor pentru diferiți parametri meteo
- Serializare/deserializare
- Validare și normalizare

8.3 Modele pentru utilizatori și autentificare

- Structura modelului User
- Stocarea parolelor și a tokenurilor
- Gestionarea rolurilor și permisiunilor

9. Interfețe API

9.1 API-uri interne

- Documentarea completă a endpoint-urilor
- Parametri, cereri și răspunsuri
- Coduri de status și gestionarea erorilor

9.2 Integrări externe

- API-uri și servicii externe utilizate
- Metode de autentificare și autorizare
- Rate limiting și cache

10. Testare și calitatea codului

10.1 Strategia de testare

- Testare unitară

- Testare integrată
- Testare end-to-end

10.2 Instrumente și metodologii

- Biblioteci și framework-uri de testare
- CI/CD
- Code coverage

10.3 Asigurarea calității

- Code reviews
- Linting și formatare
- Documentație și comentarii

11. Performanță și optimizări

11.1 Frontend

- Lazy loading și code splitting
- Memoizare și optimizarea re-renderurilor
- Bundle size optimization

11.2 Backend

- Caching
- Optimizarea bazei de date
- Procesare asincronă

11.3 Monitorizare și profiling

- Instrumente de monitorizare
- Metrici de performanță

- Strategii de optimizare

12. Scalabilitate și extensibilitate

12.1 Arhitectură modulară

- Structurarea codului pentru extensibilitate
- Separarea responsabilităților
- Plugin și extensii

12.2 Scalabilitate

- Gestionarea volumelor mari de date
- Strategii pentru scaleup și scaleout
- Optimizări pentru performanță în condiții de încărcare

12.3 Dezvoltare viitoare

- Roadmap pentru funcționalități viitoare
- API-uri pentru integrări de terță parte
- Extinderea capabilităților hardware

13. Ghid de utilizare

13.1 Instalare și configurare

- Cerințe de sistem
- Pași de instalare (hardware și software)
- Configurare inițială

13.2 Utilizare pentru utilizatori standard

- Navigarea în interfață
- Vizualizarea și interpretarea datelor
- Utilizarea funcționalităților de bază

13.3 Administrare

- Gestionarea utilizatorilor
- Configurarea sistemului
- Monitorizarea performanței
- Backup și restaurare

14. Concluzii și contribuții

14.1 Recapitularea realizărilor

- Rezumatul funcționalităților implementate
- Îndeplinirea obiectivelor inițiale

14.2 Provocări și soluții

- Dificultăți întâmpinate în dezvoltare
- Abordări inovative

14.3 Direcții viitoare

- Îmbunătățiri potențiale
- Noi funcționalități planificate
- Oportunități de cercetare

15. Anexe

15.1 Codul sursă și documentație tehnică

- Repository GitHub
- Documentație API
- Specificații detaliate

15.2 Diagrame și scheme

- Diagrame arhitecturale
- Scheme electrice
- Modele UML

15.3 Licențe și atribuiri

- Licențe software
- Atribuiri pentru biblioteci și resurse terțe
- Contribuții și mulțumiri

—----------

3. Arhitectura Software

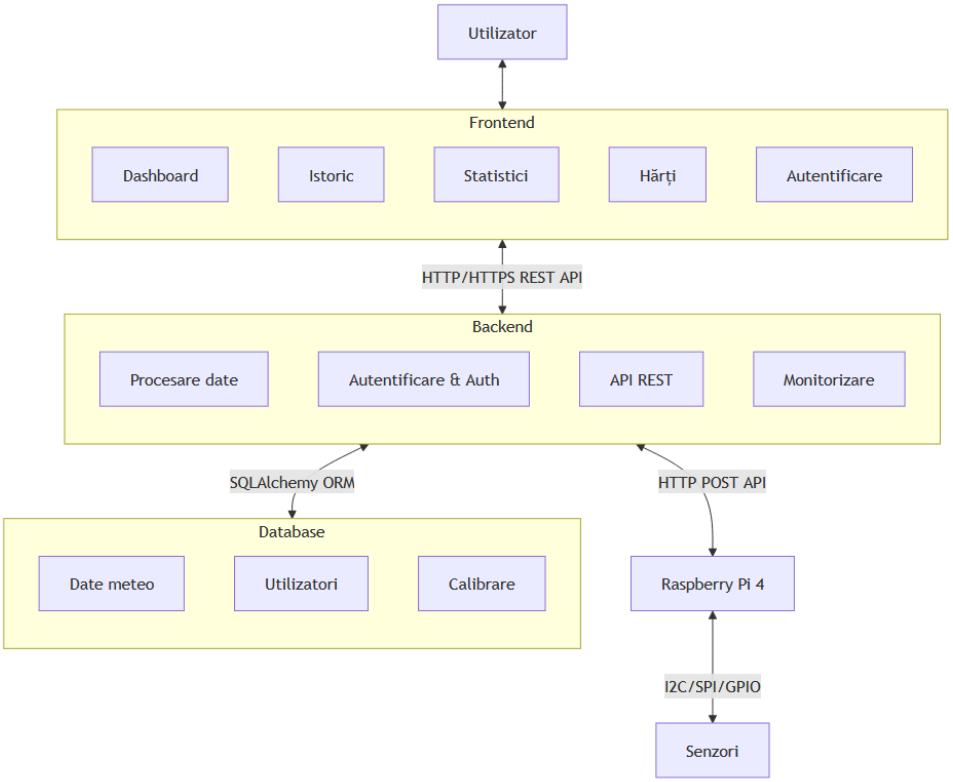
3.1 Prezentare generală

Aplicația Stației Meteo implementează o arhitectură modernă client-server, bazată pe principiile separării responsabilităților și pe o abordare orientată către servicii. Această arhitectură permite dezvoltarea independentă a componentelor, facilitând scalabilitatea, mentenabilitatea și testabilitatea întregului sistem.

Componentele principale ale arhitecturii:

1. Frontend (Client): Aplicație web dezvoltată în React, responsabilă cu interacțiunea cu utilizatorul și prezentarea datelor într-un mod intuitiv și atractiv.
2. Backend (Server): API RESTful dezvoltat în Flask (Python), care gestionează logica de business, procesarea datelor și comunicarea cu baza de date.
3. Baza de date: SQLite pentru stocarea persistentă a datelor meteorologice și a informațiilor utilizatorilor.
4. Modul de achiziție date: Script Python care rulează pe Raspberry Pi 4 pentru colectarea datelor de la senzori și transmiterea acestora către backend.

Arhitectura generala a aplicatiei:



Tipul arhitecturii: Aplicația urmează principiile unei arhitecturi RESTful, cu separarea clară între frontend și backend. Această abordare permite:

- Dezvoltarea independentă a componentelor
- Scalarea separată a fiecărei componente
- Integrarea facilă cu alte sisteme
- Testarea independentă a fiecărui nivel

Fluxul general al datelor:

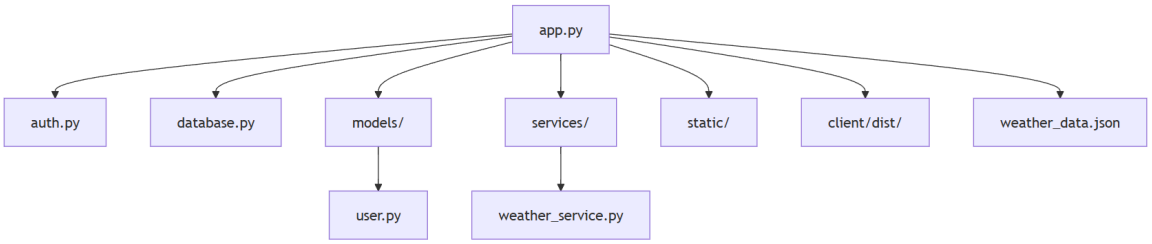
1. Senzorii colectează date meteorologice

2. Modulul de achiziție date le trimite către backend prin API-ul REST
3. Backend-ul validează, procesează și stochează datele în baza de date
4. Frontend-ul solicită date de la backend prin API
5. Utilizatorul interacționează cu frontend-ul pentru a vizualiza și analiza datele

3.2 Backend (Python/Flask)

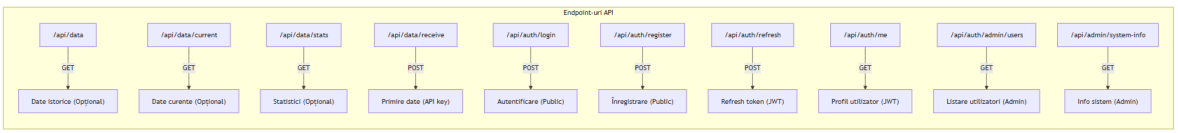
Backend-ul aplicației este dezvoltat utilizând Flask, un framework web lightweight și flexibil pentru Python, care facilitează crearea de API-uri RESTful și servirea conținutului static.

Structura aplicației Flask



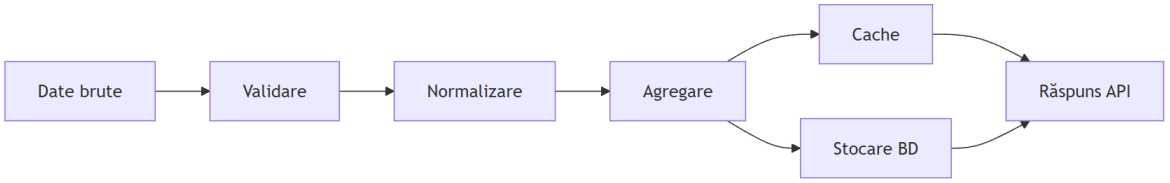
API-uri și endpoint-uri

Backend-ul expune un set complet de API-uri RESTful pentru interacțiunea cu clientul:



Procesarea și stocarea datelor

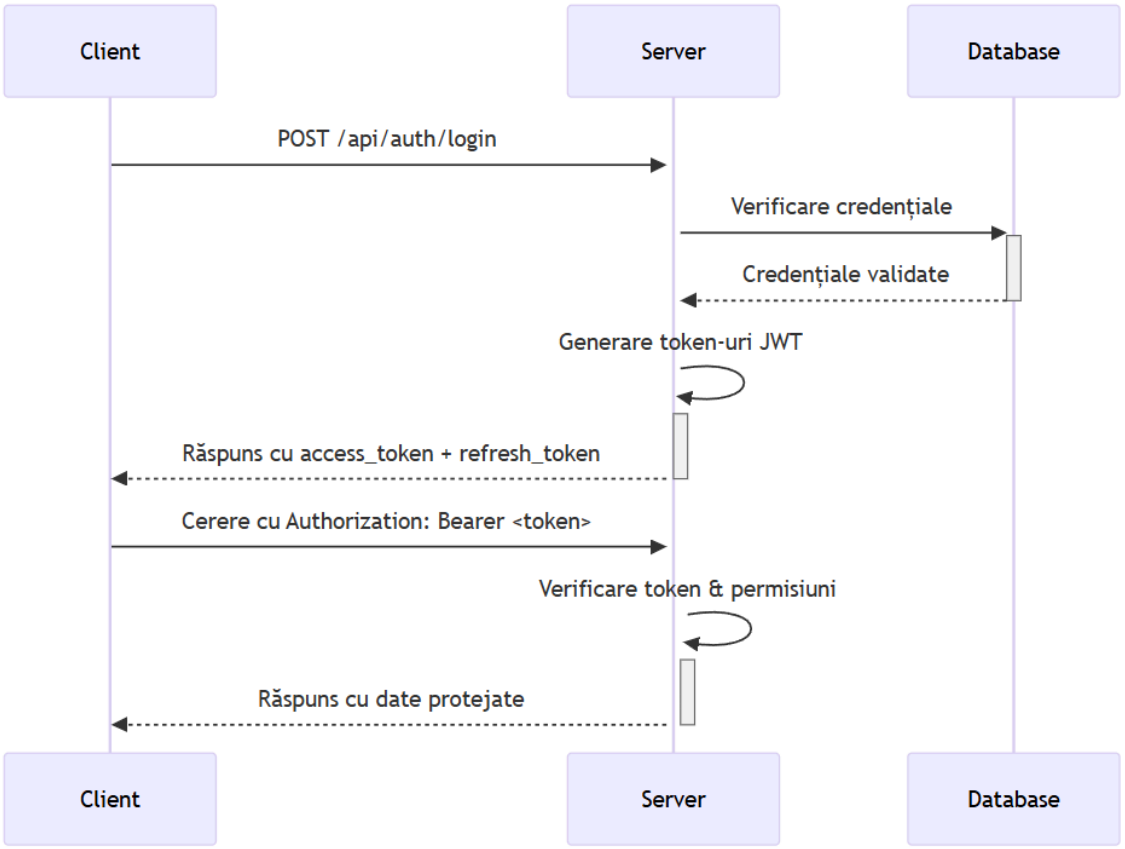
Backend-ul implementează un model sofisticat de procesare a datelor:



1. Validare: Verificarea integrității și validitatea datelor primite de la senzori
2. Normalizare: Standardizarea unităților de măsură și formatelor
3. Agregare: Calculul valorilor medii, minime și maxime pentru diferite intervale de timp
4. Persistență: Stocarea datelor în baza de date SQLite
5. Caching: Implementarea unui mecanism de cache pentru a optimiza performanța la cereri frecvente

Managementul autentificării și autorizării

Sistemul de autentificare utilizează o combinație de JWT (JSON Web Tokens) și Flask-Login pentru a asigura securitatea și flexibilitatea:



- JSON Web Tokens (JWT): Utilizate pentru autentificarea API-urilor, cu token-uri de acces (short-lived) și de reîmprospătare (long-lived)
- Hash-uri de parole: Parolele sunt stocate folosind algoritmi de hashing securizați (Werkzeug)
- Control de acces bazat pe roluri: Implementare a două roluri principale - utilizator standard și administrator
- Middleware de autorizare: Verificarea permisiunilor pentru rutele protejate

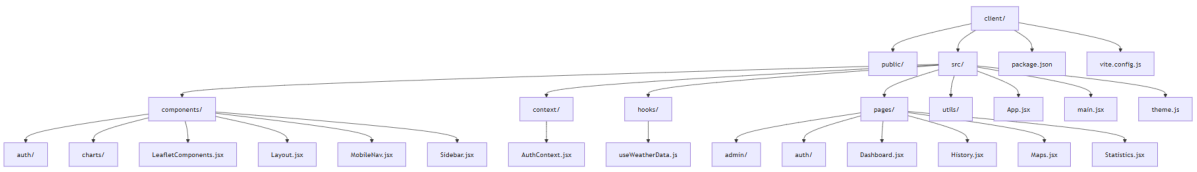
```python
# Exemplu de middleware pentru verificarea rolului de administrator
@jwt_required()
def admin_required():
    current_user_id = get_jwt_identity()
    current_user = User.query.get(current_user_id)

    if not current_user or not current_user.is_admin():
        return jsonify({'error': 'Acces interzis'}), 403
```

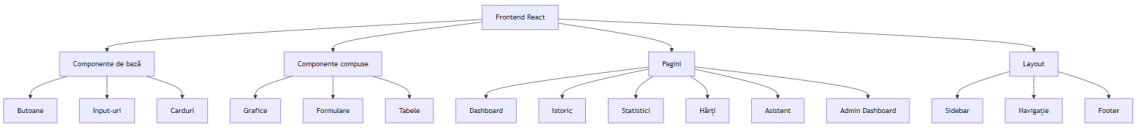
3.3 Frontend (React)

Frontend-ul este dezvoltat utilizând React, o bibliotecă JavaScript pentru construirea interfețelor utilizator interactive, cu o arhitectură modulară și componentizată, care facilitează dezvoltarea și mentenabilitatea.

Arhitectura aplicației React



Componentele principale



Frontend-ul este structurat în jurul unor componente reutilizabile, organizate pe mai multe niveluri:

1. Componente de bază: Butoane, input-uri, carduri, adaptate temei aplicației
2. Componente compuse: Grafice, formulare, tabele, panouri de control
3. Pagini: Dashboard, Istoric, Statistici, Hărți, Asistent
4. Layout: Structura de bază a aplicației, inclusiv bara laterală și navigarea

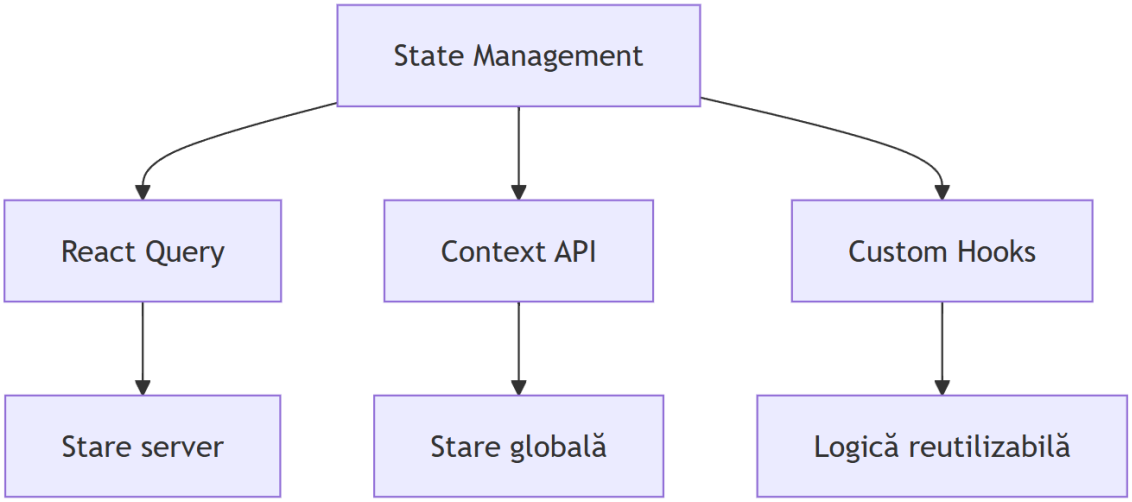
Exemple de componente cheie:

- Dashboard: Afișează datele curente și grafice pentru parametrii principali
- Charts: Componente reutilizabile pentru diferite tipuri de grafice (linie, bară, gauge)

- Maps: Vizualizări interactive pe hartă cu diferite straturi (heatmap, 3D, satelit)
- AdminDashboard: Panou de administrare cu statistici de sistem și gestionarea utilizatorilor

State management

Aplicația utilizează o combinație de tehnici moderne pentru gestionarea stării:



React Query: Pentru gestionarea stării server-side (date de la API)

```
// Exemplu de utilizare React Query
function useCurrentWeather() {
  return useQuery({
    queryKey: ['currentWeather'],
    queryFn: weatherService.getCurrentData,
    refetchInterval: 5000, // Actualizare la fiecare 5 secunde
  });
}
```

Context API: Pentru starea globală a aplicației (autentificare, preferințe)

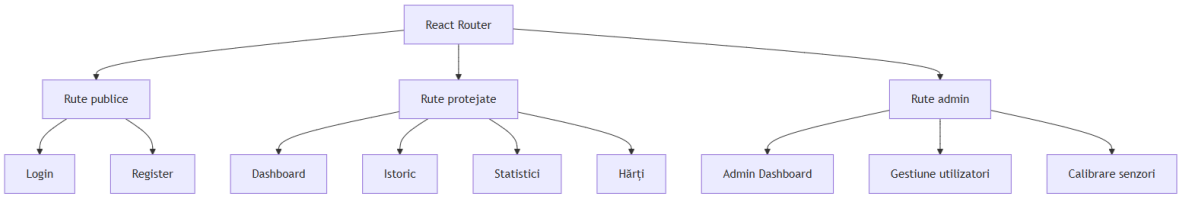
```jsx
// Exemplu din AuthContext
const AuthContext = createContext(null);

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  // ...
  return (
    <AuthContext.Provider value={{ user, login, logout, isAdmin }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Hooks personalizați: Pentru logica reutilizabilă și encapsularea funcționalităților complexe

```jsx
// Hook personalizat pentru urmărirea istoricului unui parametru
function useParameterHistory(parameter, period = '24h') {
  // ...implementare
  return { data, loading, error };
}
```

Routing și navigare



- Navigarea în aplicație este gestionată prin React Router, care asigură:
    - Rutare pentru diferitele pagini ale aplicației
    - Protejarea rutelor care necesită autentificare
    - Restricționarea accesului la rutele administrative

```
// Exemplu de configurare a rutelor
<Routes>
  {/* Rute publice */}
  <Route path="/login" element={<LoginPage />} />
  <Route path="/register" element={<RegisterPage />} />

  {/* Rute protejate */}
  <Route path="/" element={<ProtectedRoute><Layout /></ProtectedRoute>}>
    <Route index element={<Dashboard />} />
    <Route path="history" element={<History />} />
    {/* ... alte rute ... */}
  </Route>

  {/* Rute pentru administrare */}
  <Route path="/admin" element={<AdminRoute><Layout /></AdminRoute>}>
    <Route index element={<AdminDashboard />} />
    {/* ... alte rute administrative ... */}
  </Route>
</Routes>
```

Responsive design

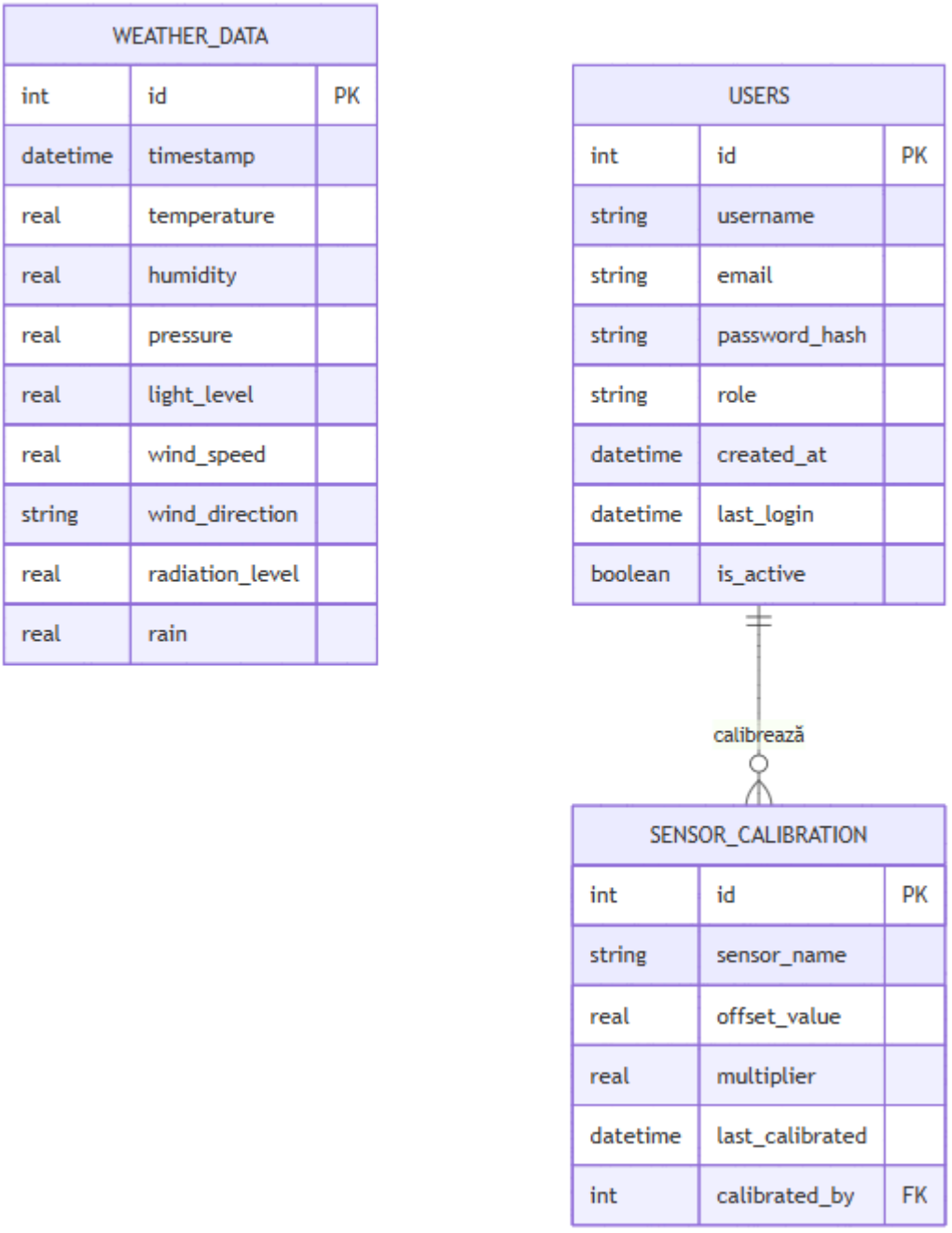
Aplicația implementează un design responsive care se adaptează la diferite dimensiuni de ecran:

- Utilizarea sistemului de grid și breakpoint-uri din Chakra UI
- Navigare diferită pentru dispozitive mobile vs. desktop
- Reorganizarea componentelor în funcție de dimensiunea ecranului

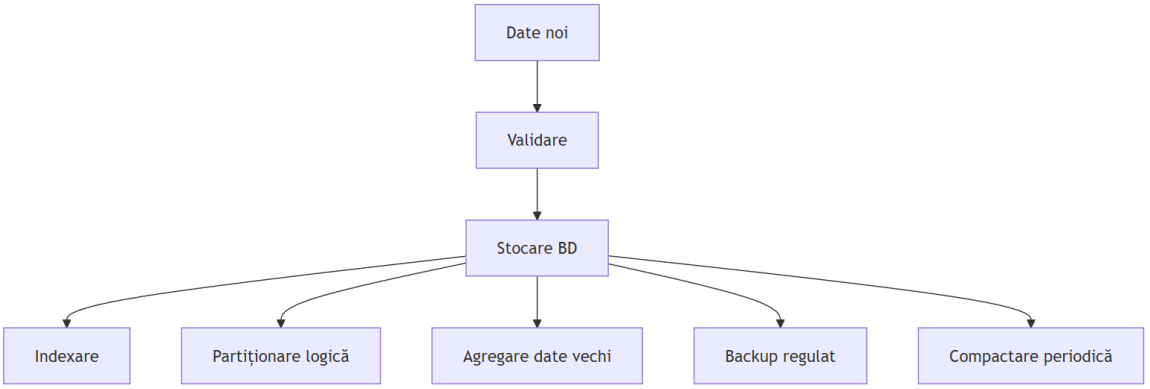
3.4 Baza de date

Aplicația utilizează SQLite ca sistem de management al bazei de date, oferind un echilibru excelent între simplitate, performanță și portabilitate.

Structura bazei de date SQLite



Strategie de stocare și arhivare a datelor

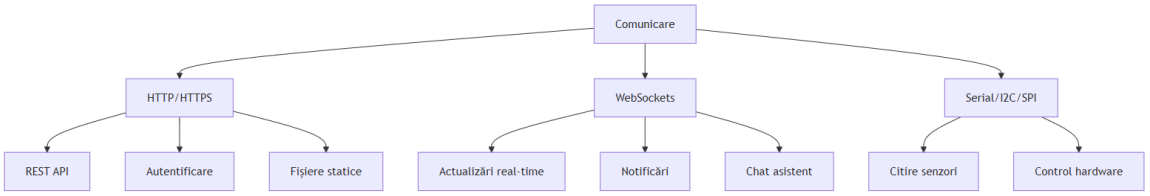


Pentru a asigura performanța și a gestiona eficient volumul de date, aplicația implementează:

1. Indexarea coloanelor frecvent utilizate în căutări (timestamp, parametri)
2. Partajarea logică a datelor istorice pe intervale de timp
3. Agregarea automată a datelor vechi (rezoluție redusă pentru date mai vechi de 30 zile)
4. Backup-uri regulate ale bazei de date
5. Compactare periodică pentru optimizarea spațiului de stocare

3.5 Comunicare și protocoale

Protocoale de comunicare



Aplicația utilizează următoarele protocoale de comunicare

1. HTTP/HTTPS: Pentru comunicarea dintre frontend și backend
    - Cereri REST pentru date
    - Autentificare și autorizare
    - Transferul fișierelor statice

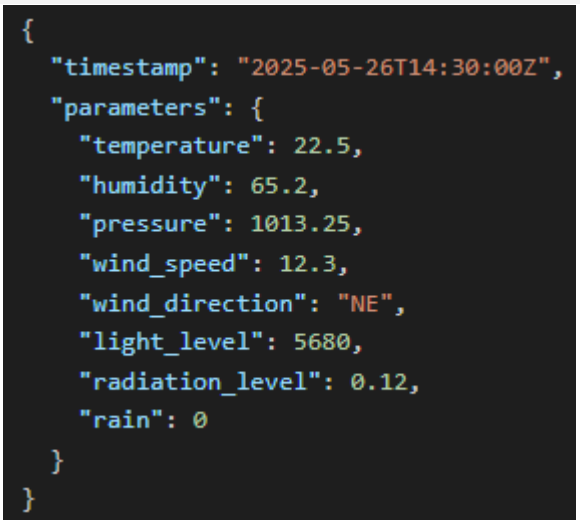
2. WebSockets (planificat pentru actualizări viitoare):
    - Actualizări în timp real pentru dashboard
    - Notificări pentru condiții meteo extreme
    - Chat cu asistentul meteo
3. Serial/I2C/SPI: Pentru comunicarea cu senzorii (la nivelul Raspberry Pi)
    - Citirea datelor de la senzori analogici și digitali
    - Controlul modulelor hardware

Format de schimb de date

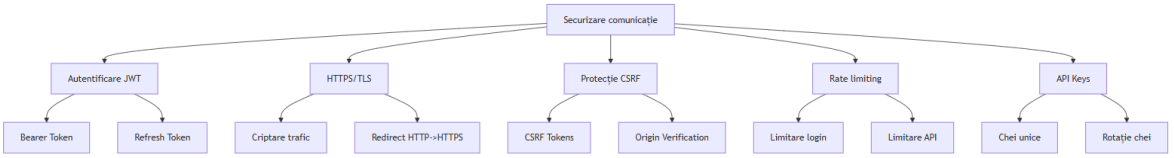
Aplicația utilizează JSON (JavaScript Object Notation) ca format standard pentru schimbul de date, datorită:

- Simplității și lizibilității
- Suportului nativ în JavaScript și Python
- Flexibilității în reprezentarea structurilor de date complexe
- Dimensiunii reduse comparativ cu alte formate (ex. XML)

Exemplu de date meteo în format JSON:



Securizarea comunicației



Pentru a asigura securitatea comunicației dintre componentele sistemului, sunt implementate următoarele măsuri:

1. Autentificare cu JWT: Toate cererile către API-urile protejate includ un token JWT în header-ul Authorization
2. HTTPS (în producție): Criptarea comunicației între client și server
    - Certificat SSL/TLS pentru domeniul aplicației
    - Redirecționare automată HTTP → HTTPS
3. Validare CSRF: Protecție împotriva atacurilor Cross-Site Request Forgery
    - Token-uri CSRF pentru operațiunile sensibile
    - Verificarea originii cererilor
4. Rate limiting: Limitarea numărului de cereri pentru a preveni atacurile de tip brute force și DoS

```
# Exemplu de implementare rate limiting
@limiter.limit("5 per minute")
@app.route("/api/auth/login", methods=["POST"])
def login():
    # ... implementare login
```

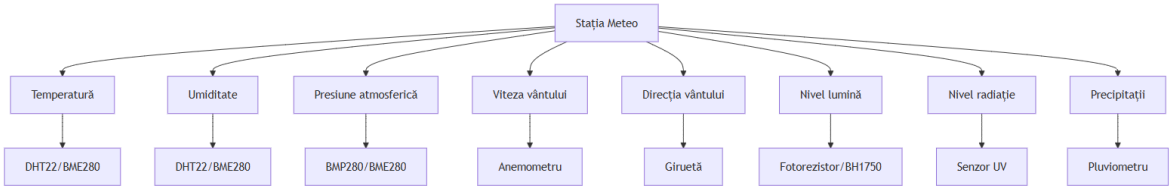
5. API Keys: Pentru comunicarea între modulul de achiziție date și backend
    - Chei unice pentru fiecare sursă de date
    - Rotație periodică a cheilor pentru securitate sporită

4. Funcționalitățile sistemului

4.1 Colectare date

Sistemul de colectare a datelor reprezintă fundamentul întregii aplicații, asigurând fluxul continuu de informații meteorologice pentru procesare și vizualizare.

Senzori și parametri monitorizați



Sistemul utilizează o varietate de senzori conectați la Raspberry Pi 4 pentru a măsura parametrii meteorologici:

- Temperatură și umiditate: Senzori de tip DHT22 sau BME280, cu precizie de ±0.5°C pentru temperatură și ±2% pentru umiditate
- Presiune atmosferică: Senzori BMP280 sau BME280, cu precizie de ±1 hPa
- Vânt: Anemometru pentru viteză și giruetă pentru direcție
- Lumină: Fotorezistori sau module specializate BH1750
- Radiație UV: Senzori UV pentru monitorizarea radiației solare
- Precipitații: Pluviometru basculant pentru măsurarea cantității de precipitații

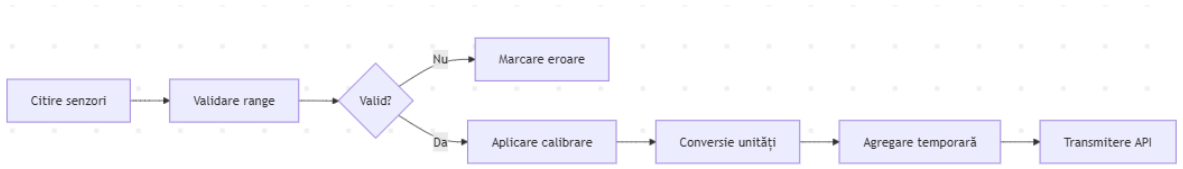
Frecvența de eșantionare

Datele sunt colectate cu frecvențe diferite, optimizate pentru fiecare tip de senzor:

- Parametri cu variație rapidă (temperatură, umiditate, presiune): la fiecare 60 secunde
- Parametri cu variație lentă (radiație, precipitații): la fiecare 5 minute
- Evenimente meteorologice (direcția vântului): la schimbare sau la fiecare 30 secunde

Acest sistem de eșantionare echilibrează nevoia de date actualizate frecvent cu optimizarea resurselor hardware și de stocare.

Prelucrarea și validarea datelor primare



Datele colectate trec printr-un proces de validare și prelucrare:

1. Validarea range-ului: Verificarea dacă valorile se încadrează în intervalele fizic posibile
2. Filtrarea outlierilor: Eliminarea valorilor aberante prin algoritmi statistici
3. Aplicarea calibrării: Utilizarea offset-urilor și multiplicatorilor specifici fiecărui senzor
4. Conversia unităților: Standardizarea unităților de măsură (°C, hPa, mm, etc.)
5. Agregarea temporară: Calculul valorilor medii pentru intervalul de raportare

Exemplu de cod pentru validarea și calibrarea datelor:

```python
def process_sensor_data(sensor_id, raw_value):
    # Obținere calibrare pentru senzor
    calibration = get_sensor_calibration(sensor_id)

    # Verifică dacă valoarea este în intervalul valid
    valid_range = SENSOR_VALID_RANGES.get(sensor_id, (-float('inf'), float('inf')))
    if not (valid_range[0] <= raw_value <= valid_range[1]):
        log_error(f"Valoare invalidă pentru senzorul {sensor_id}: {raw_value}")
        return None

    # Aplică calibrarea
    calibrated_value = raw_value * calibration.multiplier + calibration.offset

    # Conversie unități dacă e necesar
    final_value = convert_units(calibrated_value, sensor_id)

    return final_value
```

Simularea senzorilor pentru testare

Pentru a facilita dezvoltarea și testarea aplicației fără necesitatea unei instalări hardware complete, am implementat un modul de simulare a senzorilor care:

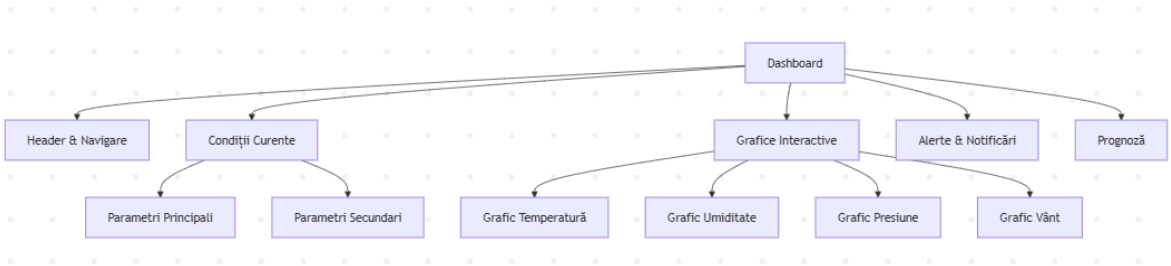
- Generează date aleatorii în intervale realiste pentru fiecare parametru
- Simulează variații diurne și sezoniere pentru parametri precum temperatura
- Introduce ocazional “anomalii” controlate pentru a testa robustețea sistemului
- Poate fi configurat să simuleze diferite condiții meteorologice (furtună, cer senin, etc.)

Acest simulator rulează ca un script Python separat și transmite datele către backend folosind același API ca și sistemul real.

4.2 Dashboard

Dashboard-ul reprezintă interfața principală a aplicației, oferind o prezentare centralizată și intuitivă a tuturor parametrilor meteorologici relevanți.

Componente și layout



Dashboard-ul este structurat în zone funcționale distincte:

1. Secțiunea de condiții curente: Afișează valorile actuale pentru toți parametrii monitorizați, cu indicatori vizuali pentru valorile ieșite din normal
2. Grafice interactive: Prezintă evoluția parametrilor în ultimele 24 de ore, cu posibilitatea de zoom și pan
3. Alerte și notificări: Evidențiază condițiile extreme sau schimbările semnificative
4. Prognoză simplă: Oferă o predicție de bază bazată pe tendințele recente și date istorice

Vizualizare în timp real

Dashboard-ul se actualizează automat la fiecare 5 secunde pentru a reflecta cele mai recente date. Implementarea acestei funcționalități folosește React Query pentru polling eficient:

```jsx
function CurrentWeatherDisplay() {
  const { data, isLoading, error } = useQuery({
    queryKey: ['currentWeather'],
    queryFn: weatherService.getCurrentData,
    refetchInterval: 5000,  // Actualizare la fiecare 5 secunde
    staleTime: 2000,        // Consideră datele valide pentru 2 secunde
  });

  if (isLoading) return <LoadingSkeleton />;
  if (error) return <ErrorDisplay message={error.message} />;

  return (
    <Grid templateColumns="repeat(4, 1fr)" gap={6}>
      <WeatherCard
        title="Temperatură"
        value={data.temperature}
        unit="°C"
        icon={<TempIcon />}
        trend={calculateTrend(data.temperature_history)}
      />
      <WeatherCard
        title="Umiditate"
        value={data.humidity}
        unit="%"
        icon={<HumidityIcon />}
      />
      {/* Alte carduri pentru parametri */}
    </Grid>
  );
}
```

Reprezentări grafice interactive

Graficele din dashboard utilizează biblioteca Chart.js adaptată pentru React (react-chartjs-2), oferind:

- Vizualizări line, bar și gauge pentru diferite tipuri de date
- Interactivitate: hover pentru detalii, zoom pentru intervale specifice
- Teme adaptate pentru modul întunecat/luminos
- Responsivitate pe diferite dispozitive

Exemplu de componentă pentru graficul de temperatură:

```
function TemperatureChart({ data, period = '24h' }) {
  const chartData = {
    labels: data.map(item => formatTime(item.timestamp)),
    datasets: [
      {
        label: 'Temperatură (°C)',
        data: data.map(item => item.temperature),
        borderColor: theme.colors.red[500],
        backgroundColor: `rgba(${hexToRgb(theme.colors.red[500])}, 0.1)`,
        fill: true,
        tension: 0.4,
      },
    ],
  };

  const options = {
    responsive: true,
    plugins: {
      legend: { position: 'top' },
      tooltip: { mode: 'index', intersect: false },
    },
    scales: {
      y: {
        suggestedMin: Math.min(...data.map(item => item.temperature)) - 2,
        suggestedMax: Math.max(...data.map(item => item.temperature)) + 2,
      },
    },
  };

  return <Line data={chartData} options={options} />;
}
```

Statistici rapide și alerte

Dashboard-ul prezintă statistici calculate în timp real:

- Valorile minime și maxime pentru ziua curentă
- Comparații cu valorile medii pentru perioada anului
- Tendințe pentru următoarele ore

Sistemul de alerte evidențiază condițiile care necesită atenție:

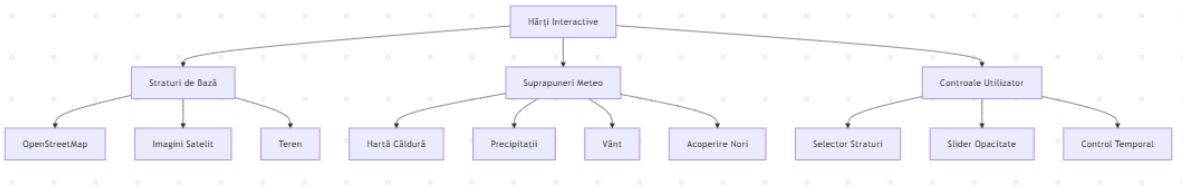
- Temperaturi extreme (caniculă sau îngheț)

- Schimbări rapide de presiune (posibil front atmosferic)
- Viteze mari ale vântului
- Precipitații abundente

4.3 Vizualizări avansate

Pe lângă dashboard-ul standard, aplicația oferă vizualizări avansate care permit o înțelegere mai profundă a datelor meteorologice prin reprezentări interactive și contextualizate.

Hărți interactive cu suprapuneri meteo

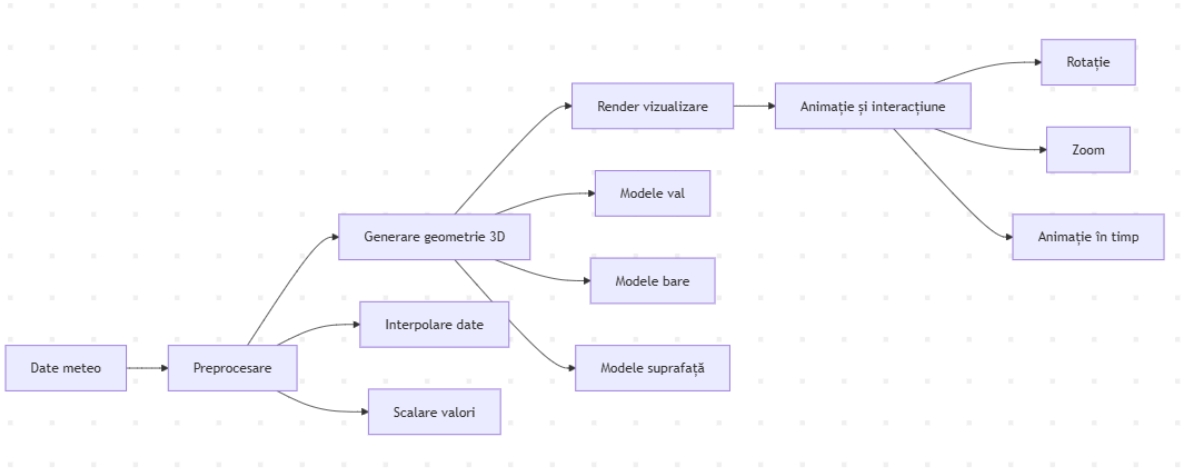


Componenta de hărți utilizează biblioteca Leaflet împreună cu diverse plugin-uri pentru a oferi:

- Straturi de bază multiple: OpenStreetMap, imagini satelit, teren
- Suprapuneri meteorologice: Temperatură (heatmap), precipitații, vânt, acoperire cu nori
- Integrare cu servicii externe: Windy.com, RainViewer pentru date globale
- Controale interactive: Selectare straturi, ajustare opacitate, timeline

Datele locale de la stația meteo sunt reprezentate pe hartă prin markeri și zone de influență, iar datele globale sunt obținute prin API-uri externe.

Vizualizări 3D pentru parametri meteo



Vizualizările 3D oferă o perspectivă unică asupra datelor meteorologice, permițând identificarea de tipare și corelații care ar fi dificil de observat în reprezentări 2D tradiționale.

Componenta de vizualizare 3D include:

- Reprezentări de tip “val” pentru parametri precum temperatura și presiunea
- Grafice 3D cu bare pentru precipitații și radiație
- Suprafețe 3D pentru reprezentarea distribuției spațiale a parametrilor
- Animații care arată evoluția parametrilor în timp

Implementarea utilizează CSS 3D Transforms și animații, fără a necesita biblioteci 3D complexe, asigurând astfel compatibilitatea și performanța pe diverse dispozitive.

Integrare cu servicii meteo externe

Aplicația integrează date din multiple surse externe pentru a completa informațiile locale:

- Windy.com API: Modele atmosferice globale, inclusiv predicții
- RainViewer: Date radar pentru precipitații în timp real
- OpenWeatherMap: Date și prognoze pentru locații din proximitate

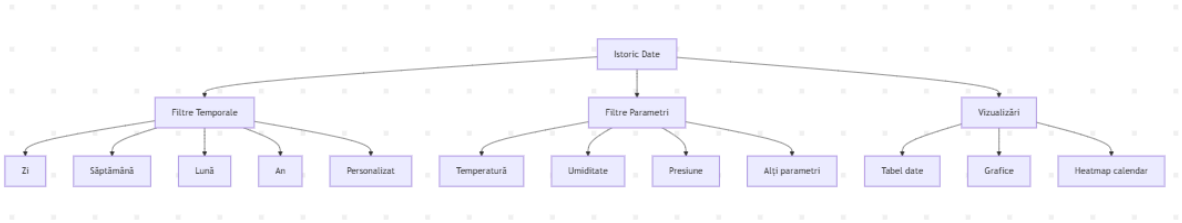
Aceste integrări permit:

- Compararea datelor locale cu cele din modele globale
- Vizualizarea tendințelor meteorologice la scară mai largă
- Plasarea condițiilor locale în context regional

4.4 Istoric și statistici

Componenta de istoric și statistici permite analiza detaliată a datelor colectate de-a lungul timpului, oferind insight-uri valoroase despre tiparele și tendințele meteorologice.

Vizualizarea și filtrarea datelor istorice



Interfața de istoric oferă utilizatorului control complet asupra vizualizării datelor:

- Filtre temporale flexibile: Selectare zi, săptămână, lună, an sau interval personalizat
- Selecție de parametri: Posibilitatea de a alege unul sau mai mulți parametri pentru vizualizare
- Opțiuni de vizualizare: Tabel de date, grafice liniare, grafice de distribuție, heatmap calendar

Datele sunt încărcate dinamic pe măsură ce utilizatorul ajustează filtrele, folosind paginare și lazy loading pentru a gestiona eficient seturi mari de date.

Generarea de rapoarte statistice

Sistemul de statistici oferă informații valoroase derivate din datele brute:

- Statistici de bază: Valori minime, maxime, medii, mediane, deviații standard
- Analiza tendințelor: Identificarea tendințelor pe termen lung și a anomaliilor
- Comparații: Confruntarea datelor curente cu perioade similare din trecut
- Recorduri: Evidențierea valorilor record pentru fiecare parametru

Aceste statistici sunt generate automat și pot fi accesate prin interfața aplicației sau exportate în diverse formate.

Exportul datelor în diverse formate

Aplicația oferă funcționalități de export al datelor pentru utilizare în alte aplicații sau analize:

- CSV: Format tabelar compatibil cu Excel, Google Sheets, etc.
- JSON: Format structurat pentru integrare cu alte aplicații
- PDF: Rapoarte formatate pentru imprimare și distribuire
- PNG/JPG: Export grafice ca imagini

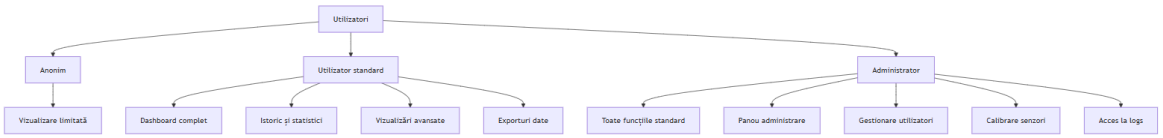
Exporturile pot fi configurate în detaliu, permițând utilizatorului să selecteze:

- Intervalul temporal exact
- Parametrii incluși
- Rezoluția temporală (date brute sau agregate)
- Formatul de dată și separatorii (pentru CSV)

4.5 Sistem de autentificare

Sistemul de autentificare asigură accesul securizat la aplicație și permite diferențierea funcționalităților disponibile în funcție de rolul utilizatorului.

Nivele de acces



Aplicația implementează trei nivele de acces:

1. Anonim: Acces limitat la date de bază, fără posibilitatea de a vedea istoricul detaliat sau a face modificări
2. Utilizator standard: Acces complet la vizualizări, istoric și funcționalități de export
3. Administrator: Toate drepturile utilizatorului standard plus acces la funcționalități administrative

Managementul utilizatorilor

Administratorii pot gestiona utilizatorii sistemului prin:

- Crearea de conturi noi: Înregistrarea de utilizatori cu diverse roluri
- Editarea conturilor: Modificarea datelor utilizatorilor, resetarea parolelor
- Dezactivarea conturilor: Suspendarea temporară a accesului
- Vizualizarea sesiunilor active: Monitorizarea și închiderea forțată a sesiunilor

Interfața de administrare permite filtrarea și căutarea utilizatorilor, precum și vizualizarea activității recente a acestora.

Funcționalități specifice administratorului

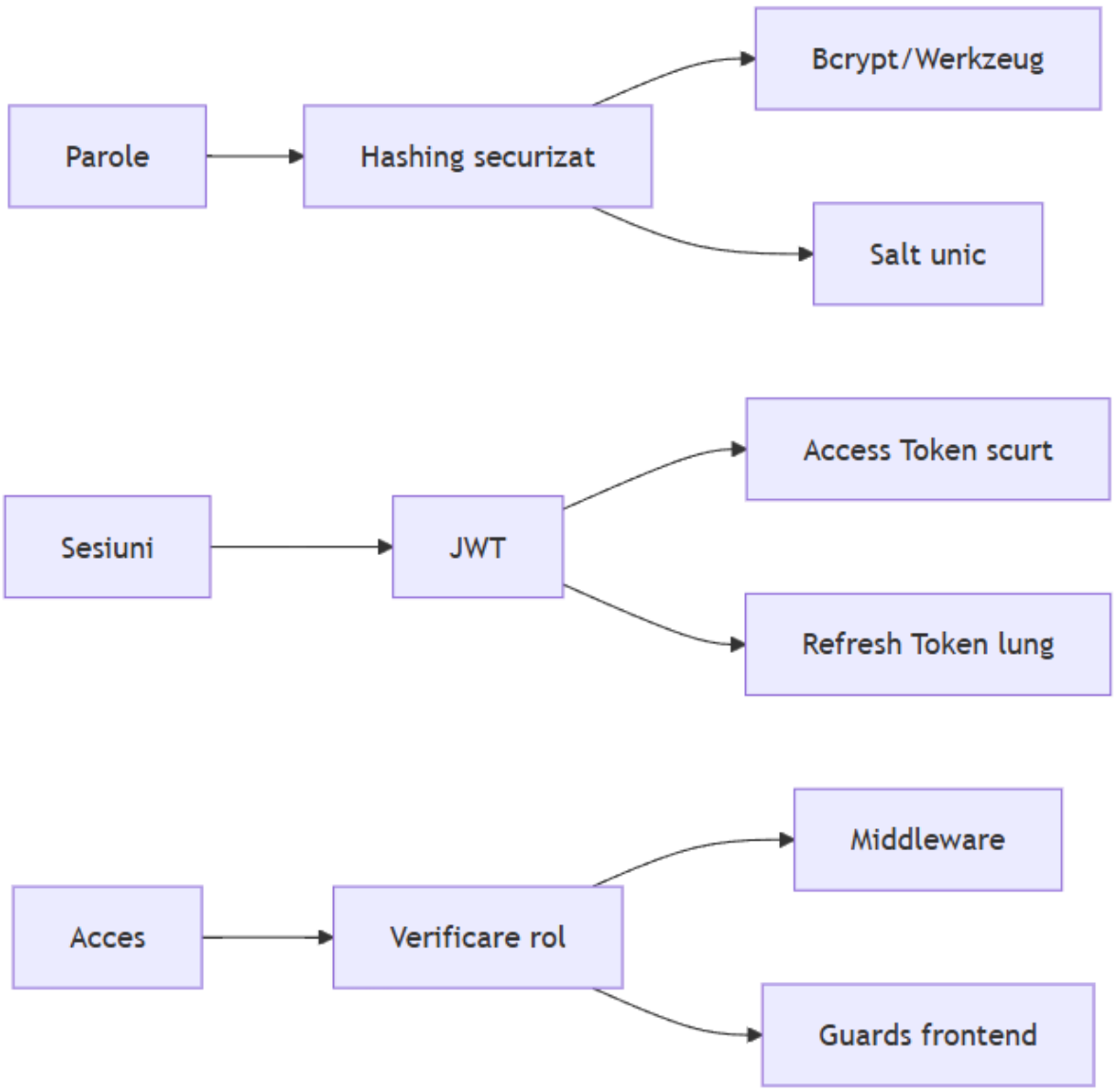
Panoul de administrare oferă acces la funcționalități avansate:

- Monitorizarea sistemului: Utilizare CPU, memorie, spațiu pe disc

- Gestionarea bazei de date: Backup, optimizare, curățare date vechi
- Calibrarea senzorilor: Ajustarea offset-urilor și multiplicatorilor pentru precizie
- Jurnale de sistem: Vizualizarea log-urilor pentru diagnosticare
- Configurare avansată: Ajustarea parametrilor de funcționare a aplicației

Aceste funcționalități sunt accesibile doar utilizatorilor cu rol de administrator, asigurând controlul și securitatea sistemului.

Securitatea autentificării



4.6 Asistent meteo

Asistentul meteo reprezintă o inovație în domeniul stațiilor meteo personale, oferind o interfață conversațională pentru interacțiunea cu datele meteorologice. Acesta permite utilizatorilor să adreseze întrebări în limbaj natural și să primească răspunsuri relevante.
Interfața asistentului este proiectată pentru a fi accesibilă și intuitivă:

- Chat textual: Permite formularea de întrebări prin text
- Interfață vocală (opțional): Recunoaștere și sinteză vocală pentru interacțiune hands-free
- Sugestii de întrebări: Propuneri contextualizate de întrebări pe care utilizatorul le poate adresa

Capacitățile de răspuns ale asistentului

Asistentul meteo poate răspunde la o gamă largă de întrebări și comenzi:

- Informații curente: “Care este temperatura acum?”, “Plouă în acest moment?”
- Istoric și statistici: “Care a fost temperatura maximă ieri?”, “Cum a evoluat presiunea în ultima săptămână?”
- Comparații: “Este mai cald acum decât ieri la aceeași oră?”, “Compară umiditatea de azi cu media lunii”
- Predicții: “Va ploua mâine?”, “Care este tendința presiunii atmosferice?”
- Explicații: “De ce scade presiunea atmosferică?”, “Ce înseamnă un indice UV de 8?”

Răspunsurile includ nu doar text, ci și elemente vizuale relevante (grafice, indicatori) pentru a ilustra informațiile prezentate.

Tehnologiile de inteligență artificială utilizate

Asistentul integrează multiple tehnologii de inteligență artificială:

- Procesare de limbaj natural (NLP): Pentru înțelegerea întrebărilor în limbaj natural
- Învățare automată: Pentru personalizarea răspunsurilor în funcție de preferințele utilizatorului
- Analiza seriilor temporale: Pentru identificarea tendințelor și modelelor în datele istorice
- Sisteme bazate pe reguli: Pentru interogarea precisă a bazei de date și generarea răspunsurilor

5. Implementare și tehnologii

5.1 Backend

Backend-ul aplicației este implementat folosind tehnologii moderne și robuste pentru a asigura performanța și scalabilitatea:

- Python 3.x: Limbajul principal de programare, ales pentru claritatea codului și ecosistemul bogat de biblioteci
- Flask: Framework web lightweight, oferind flexibilitate și extensibilitate
- Extensii Flask:
    - Flask-JWT-Extended: Pentru autentificare bazată pe token
    - Flask-Login: Pentru gestionarea sesiunilor utilizatorilor
    - Flask-CORS: Pentru gestionarea cererilor cross-origin
- SQLAlchemy ORM: Pentru abstractizarea bazei de date și manipularea obiectelor
- Biblioteci pentru procesarea datelor:
    - NumPy: Pentru operații numerice eficiente
    - Pandas: Pentru analiză și manipulare de date

Exemplu de structură a aplicației Flask:

```python
from flask import Flask, jsonify
from flask_jwt_extended import JWTManager
from flask_login import LoginManager
from database import db, User

app = Flask(__name__)
app.config['JWT_SECRET_KEY'] = os.environ.get('JWT_SECRET_KEY', 'secret-key-de-test-trebuie-schimbata-in-productie')
app.config['JWT_ACCESS_TOKEN_EXPIRES'] = timedelta(hours=1)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///weather_station.db'

# Inițializare extensii
jwt = JWTManager(app)
login_manager = LoginManager(app)
db.init_app(app)

# Înregistrare blueprint-uri
from auth import auth_bp
from api import api_bp

app.register_blueprint(auth_bp, url_prefix='/auth')
app.register_blueprint(api_bp, url_prefix='/api')
```

5.2 Frontend

Frontend-ul este construit pe baza unui stack modern, asigurând o experiență de utilizare fluidă și interactivă:

- React 18: Bibliotecă JavaScript pentru crearea interfețelor utilizator
- Chakra UI: Framework de componente pentru un design consistent și accesibil
- React Query: Pentru gestionarea stării și a cererilor către API
- Biblioteci de vizualizare:
    - react-chartjs-2: Pentru grafice interactive
    - leaflet: Pentru hărți interactive
- React Router: Pentru navigarea în aplicație

Exemplu de structură a aplicației React:

```jsx
import React from 'react';
import { ChakraProvider, theme } from '@chakra-ui/react';
import { QueryClient, QueryClientProvider } from 'react-query';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import ProtectedRoute from './components/ProtectedRoute';
import DashboardPage from './pages/DashboardPage';
import LoginPage from './pages/auth/LoginPage';
import RegisterPage from './pages/auth/RegisterPage';

const queryClient = new QueryClient();

function App() {
  return (
    <ChakraProvider theme={theme}>
      <QueryClientProvider client={queryClient}>
        <AuthProvider>
          <Router>
            <Routes>
              <Route path="/login" element={<LoginPage />} />
              <Route path="/register" element={<RegisterPage />} />
              <Route
                path="/*"
                element={
                  <ProtectedRoute>
                    <DashboardPage />
                  </ProtectedRoute>
                }
              />
            </Routes>
          </Router>
        </AuthProvider>
      </QueryClientProvider>
    </ChakraProvider>
  );
}
```

5.3 Baza de date

Aplicația utilizează SQLite pentru stocarea datelor, o soluție optimă pentru implementări locale:

- SQLite: Motor de bază de date relațională, lightweight și fără server
- Structura tabelelor:
    - users: Informații despre utilizatori și autentificare

    - weather_data: Date meteorologice primare
    - weather_aggregates: Date agregate pentru diferite intervale de timp
- Indexare și optimizare:
    - Indecși pentru coloanele utilizate frecvent în filtrare (timestamp, parametri)
    - Optimizare pentru interogări frecvente

5.4 DevOps și deployment

Procesele de dezvoltare și deployment sunt automatizate pentru a asigura consistența și fiabilitatea:

- Automatizare build:
    - Script-uri pentru build-ul aplicației frontend
    - Generarea de asset-uri statice
- Strategii de deployment:
    - Deployment local pe Raspberry Pi
    - Configurație pentru deployment în cloud (opțional)
- Monitorizare și logging:
    - Logging la nivel de aplicație
    - Monitorizarea performanței și a resurselor hardware

6. Caracteristici de securitate

6.1 Autentificare și autorizare

Sistemul implementează măsuri robuste de securitate pentru autentificare:

- JSON Web Tokens (JWT):
    - Tokens cu durată limitată
    - Refresh tokens pentru experiență fluidă
- Hashing-ul parolelor:
    - Utilizarea bibliotecii Werkzeug pentru hashing securizat

    - Salt unic pentru fiecare parolă
- Managementul sesiunilor:
    - Sesiuni cu durată limitată
    - Invalidare la delogare
- Controlul accesului bazat pe roluri:
    - Diferențiere între utilizatori standard și administratori
    - Restricționarea accesului la funcționalități administrative

6.2 Securitatea API-urilor

API-urile sunt securizate împotriva mai multor tipuri de vulnerabilități:

- Validarea input-urilor:
    - Verificarea tipurilor și formatelor de date
    - Sanitizarea input-urilor pentru prevenirea injecțiilor
- Rate limiting:
    - Limitarea numărului de cereri per IP/utilizator
    - Prevenirea atacurilor de tip brute force
- Protecția împotriva atacurilor comune:
    - Măsuri CSRF (Cross-Site Request Forgery)
    - Prevenirea XSS (Cross-Site Scripting)
    - Headerele de securitate adecvate

6.3 Securitatea datelor

Datele stocate și transmise sunt protejate prin:

- Criptarea datelor sensibile:
    - Criptarea credențialelor de acces
    - Transmiterea securizată prin HTTPS
- Backup și recuperare:
    - Backup-uri periodice automate
    - Proceduri de recuperare în caz de corupere
- Strategii de retenție a datelor:
    - Politici clare pentru stocarea datelor
    - Anonimizarea datelor vechi

7. Fluxuri de date și procese

7.1 Fluxul de date pentru colectarea informațiilor

Diagrama de mai jos ilustrează fluxul de date de la senzori până la baza de date:

Principalele etape ale procesului:

- Colectarea datelor brute de la senzori
- Validarea și filtrarea outlier-ilor
- Normalizarea și conversia unităților
- Transmiterea către backend prin API
- Procesarea și stocarea în baza de date

7.2 Fluxul interacțiunii utilizator

Interacțiunea utilizator urmează mai multe fluxuri principale:

- Vizualizare dashboard:
    - Autentificare → Accesare dashboard → Vizualizare date curente → Interacțiune cu grafice
- Analiza datelor istorice:
    - Autentificare → Accesare istoric → Setare filtre → Vizualizare → Export (opțional)
- Administrare sistem (doar administratori):
    - Autentificare → Accesare panou admin → Gestionare utilizatori/configurare/monitorizare

7.3 Procesarea în timp real

Aplicația implementează strategii eficiente pentru actualizarea în timp real:

- Polling:

    - Actualizări regulate la intervale configurabile (implicit 5 secunde)
    - Optimizări pentru reducerea traficului (fetch doar date noi)
- WebSockets (planificat):
    - Pentru comunicare bidirecțională în timp real
    - Reducerea latențelor și a overhead-ului
- Optimizări pentru performanță:
    - Caching la nivel de client și server
    - Compresie pentru reducerea volumului de date transferate

8. Modele de date

8.1 Schema bazei de date

Structura principalelor tabele din baza de date:

- users:
    - id (PK)
    - username
    - email
    - password_hash
    - role
    - created_at
    - last_login
- weather_data:
    - id (PK)
    - timestamp
    - temperature
    - humidity
    - pressure
    - wind_speed
    - wind_direction
    - light
    - uv
    - rain

    - source_id (FK la sources)
- weather_aggregates:
    - id (PK)
    - period_start
    - period_end
    - period_type (hourly, daily, monthly)
    - temperature_min, temperature_max, temperature_avg
    - humidity_min, humidity_max, humidity_avg
    - [alte agregate pentru fiecare parametru]

8.2 Modele pentru date meteorologice

Structura datelor pentru parametrii meteo:

- Structura de bază:
    - Timestamp
    - Valoare
    - Unitate de măsură
    - Metadate (sursă, precisie)
- Validare și normalizare:
    - Verificarea încadrării în range-uri valide
    - Convertirea la unități standard
    - Eliminarea outlier-ilor

8.3 Modele pentru utilizatori și autentificare

Structura modelului de utilizator:

- User:
    - Atribute de bază (id, username, email)
    - Atribute de securitate (password_hash, role)
    - Metode pentru autentificare și verificare permisiuni
- Token-uri JWT:
    - Payload-ul include id utilizator și rol
    - Expirare configurabilă
    - Refresh token cu durată mai lungă de viață

9. Interfețe API

9.1 API-uri interne

Principalele endpoint-uri ale API-ului intern:

- Autentificare:
  - POST /auth/login: Autentificare utilizator
  - POST /auth/register: Înregistrare utilizator nou
  - POST /auth/refresh: Reîmprospătare token
  - POST /auth/logout: Delogare utilizator
- Date meteo:
  - GET /api/weather/current: Date meteo curente
  - GET /api/weather/history: Istoric date meteo
  - POST /api/weather/data: Trimitere date noi (protejat)
  - GET /api/weather/stats: Statistici agregate
- Administrare:
  - GET /api/admin/users: Listare utilizatori (doar admin)
  - POST /api/admin/users: Creare utilizator (doar admin)
  - PUT /api/admin/calibration: Actualizare calibrare senzori

9.2 Integrări externe

Aplicația interacționează cu mai multe API-uri externe:

- Windy.com API:
  - Obținere de prognoze și date meteo globale
  - Integrare pentru vizualizări pe hartă
- RainViewer:
  - Date radar pentru precipitații
- OpenWeatherMap:
  - Date meteo comparative și prognoze
- Metode de autentificare:
  - API key-uri securizate
  - Rate limiting pentru respectarea limitelor de utilizare

10. Testare și calitatea codului

10.1 Strategia de testare

Aplicația este testată la multiple niveluri:

- Testare unitară:
    - Testarea componentelor individuale
    - Izolarea dependențelor prin mocking
- Testare integrată:
    - Verificarea interacțiunii între componente
    - Testarea fluxurilor de date
- Testare end-to-end:
    - Simularea interacțiunii utilizatorilor
    - Verificarea funcționalității complete

10.2 Instrumente și metodologii

Procesul de testare utilizează instrumente specializate:

- Biblioteci și framework-uri:
    - Pytest pentru backend
    - Jest și React Testing Library pentru frontend
- CI/CD:
    - Teste automate la fiecare commit
    - Verificări de calitate a codului
- Code coverage:
    - Monitorizarea acoperirii cu teste
    - Rapoarte automate de coverage

10.3 Asigurarea calității

Calitatea codului este asigurată prin:

- Code reviews:
    - Verificarea codului înainte de integrare
    - Aderența la standarde și convenții
- Linting și formatare:
    - ESLint pentru JavaScript/React
    - Pylint pentru Python
- Documentație și comentarii:
    - Docstrings pentru funcții și clase
    - JSDoc pentru componente React
    - README și documentație de API

11. Performanță și optimizări

11.1 Frontend

Optimizările frontend se concentrează pe:

- Lazy loading și code splitting:
    - Încărcarea componentelor doar când sunt necesare
    - Împărțirea bundle-ului pentru încărcare mai rapidă
- Memoizare și optimizarea re-renderurilor:
    - Utilizarea React.memo și useMemo
    - Optimizarea dependențelor pentru effects
- Bundle size optimization:
    - Minimizarea și compresarea asset-urilor
    - Tree shaking pentru eliminarea codului neutilizat

11.2 Backend

Optimizările backend includ:

- Caching:
    - Cache pentru rezultatele interogărilor frecvente
    - Strategie de invalidare a cache-ului

- Optimizarea bazei de date:
    - Indexarea corespunzătoare
    - Optimizarea query-urilor complexe
- Procesare asincronă:
    - Delegarea sarcinilor intensive în background
    - Utilizarea cozilor pentru procesarea datelor

11.3 Monitorizare și profiling

Performanța sistemului este monitorizată continuu:

- Instrumente de monitorizare:
    - Logging și metrics pentru backend
    - Performance monitoring pentru frontend
- Metrici de performanță:
    - Timp de răspuns API
    - Utilizare resurse (CPU, memorie, I/O)
    - Performanța query-urilor
- Strategii de optimizare:
    - Identificarea și rezolvarea bottleneck-urilor
    - Benchmark-uri pentru funcționalitățile critice

12. Scalabilitate și extensibilitate

12.1 Arhitectură modulară

Aplicația este proiectată modular pentru flexibilitate:

- Structurarea codului:
    - Separarea clară a responsabilităților
    - Interacțiune prin interfețe bine definite
- Separarea responsabilităților:
    - Componente cu funcționalități clare și specifice
    - Limitarea dependențelor între module

- Plugin și extensii:
    - Sistem de plugin-uri pentru funcționalități opționale
    - Hooks pentru extinderea comportamentului existent

12.2 Scalabilitate

Strategiile de scalare includ:

- Gestionarea volumelor mari de date:
    - Agregarea și arhivarea datelor vechi
    - Paginare și lazy loading pentru rezultate
- Strategii pentru scaleup și scaleout:
    - Optimizarea pentru hardware mai puternic
    - Pregătire pentru distribuire pe mai multe instanțe
- Optimizări pentru performanță:
    - Profilarea și eliminarea blocajelor
    - Caching strategic pentru reducerea încărcării

12.3 Dezvoltare viitoare

Planul de dezvoltare include:

- Roadmap pentru funcționalități viitoare:
    - Integrare cu sisteme de automatizare smart home
    - Modele predictive bazate pe ML
    - Aplicație mobilă nativă
- API-uri pentru integrări de terță parte:
    - Documentație completă OpenAPI
    - Exemple de integrare și SDK-uri
- Extinderea capabilităților hardware:
    - Suport pentru senzori adiționali
    - Integrare cu alte platforme IoT

13. Ghid de utilizare

13.1 Instalare și configurare

Procesul de instalare include:

- Cerințe de sistem:
    - Raspberry Pi 4 (minim 2GB RAM)
    - Sistemul de operare Raspberry Pi OS
    - Python 3.8+ și Node.js 14+
- Pași de instalare:
    - Clonarea repository-ului
    - Instalarea dependințelor (pip install -r requirements.txt)
    - Configurarea backend-ului și frontend-ului
- Configurare inițială:
    - Setarea variabilelor de mediu
    - Crearea contului de administrator
    - Configurarea senzorilor

13.2 Utilizare pentru utilizatori standard

Ghidul pentru utilizatori standard include:

- Navigarea în interfață:
    - Dashboard principal
    - Secțiunea de istoric și statistici
    - Vizualizări avansate
- Vizualizarea și interpretarea datelor:
    - Înțelegerea graficelor și indicatorilor
    - Filtrarea datelor istorice
    - Exportul rapoartelor
- Utilizarea funcționalităților de bază:
    - Configurarea preferințelor personale
    - Setarea alertelor
    - Interacțiunea cu asistentul meteo

13.3 Administrare

Funcționalitățile administrative includ:

- Gestionarea utilizatorilor:
    - Crearea și modificarea conturilor
    - Setarea rolurilor și permisiunilor
- Configurarea sistemului:
    - Calibrarea senzorilor
    - Ajustarea parametrilor de funcționare
- Monitorizarea performanței:
    - Verificarea jurnalelor de sistem
    - Diagnosticarea problemelor
- Backup și restaurare:
    - Crearea copiilor de siguranță
    - Proceduri de restaurare

14. Concluzii și contribuții

14.1 Recapitularea realizărilor

Proiectul a implementat cu succes:

- Sistem complet de monitorizare meteo
- Interfață modernă și intuitivă
- Funcționalități avansate de analiză și vizualizare
- Arhitectură scalabilă și extensibilă

14.2 Provocări și soluții

Principalele provocări întâlnite:

- Optimizarea performanței pe hardware limitat (Raspberry Pi)
- Gestionarea eficientă a volumelor mari de date
- Asigurarea preciziei și fiabilității măsurătorilor
- Implementarea vizualizărilor avansate

14.3 Direcții viitoare

Planuri de dezvoltare:

- Implementarea de algoritmi predictivi
- Extinderea integrărilor cu alte sisteme
- Dezvoltarea aplicației mobile companion
- Îmbunătățirea funcționalităților de asistent virtual

15. Anexe

15.1 Codul sursă și documentație tehnică

Accesul la resursele proiectului:

- Repository GitHub: github.com/user/weather-station
- Documentație API: Disponibilă în format OpenAPI
- Specificații detaliate: Incluse în repository

15.2 Diagrame și scheme

Documentația include diagrame pentru:

- Arhitectura sistemului
- Schema bazei de date
- Diagrame de secvență pentru procesele principale
- Scheme electrice pentru conexiunile hardware

15.3 Licențe și atribuiri

Informații despre licențe:

- Licență MIT pentru codul sursă
- Atribuiri pentru bibliotecile open-source utilizate
- Mulțumiri contribuitorilor și comunității