

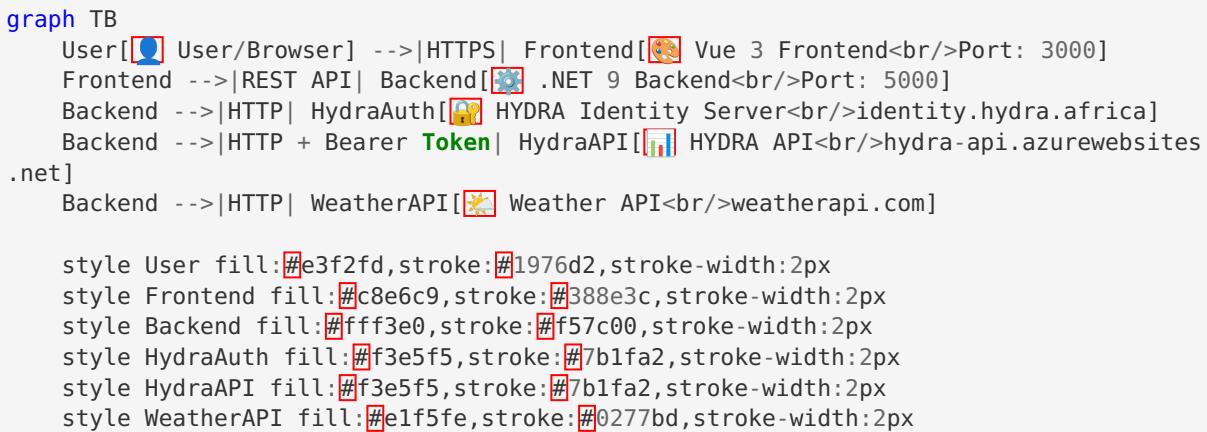
System Architecture

This document provides a comprehensive overview of the HYDRA Energy Intelligence Dashboard architecture, including system components, data flow, and deployment architecture.

Table of Contents

- [High-Level Architecture](#)
- [Component Architecture](#)
- [Data Flow Architecture](#)
- [Frontend Architecture](#)
- [Backend Architecture](#)
- [Authentication Flow](#)
- [API Architecture](#)
- [Deployment Architecture](#)
- [Technology Stack](#)

High-Level Architecture



Architecture Style: Layered Architecture with Client-Server Pattern

Key Characteristics:

- **Separation of Concerns:** Clear separation between presentation, business logic, and data access
- **Stateless Communication:** RESTful API with no server-side session state
- **Third-Party Integration:** Aggregates data from multiple external APIs
- **Modern Tech Stack:** .NET 9 backend with Vue 3 frontend

Component Architecture

```

graph TB
    subgraph "Frontend Layer - Vue 3"
        UI[UI Components]
        Store[State Management<br/>Pinia Stores]
        Router[Vue Router]
        APIClient[API Client<br/>Axios]

        UI --> Store
        UI --> Router
        Store --> APIClient
    end

    subgraph "Backend Layer - .NET 9"
        Controllers[Controllers]
        Services[Services]
        Models[Models/DTOs]
        HTTPClients[HTTP Clients<br/>Polly Policies]

        Controllers --> Services
        Services --> Models
        Services --> HTTPClients
    end

    subgraph "External APIs"
        HAPI[HYDRA API]
        WAPI[Weather API]
    end

    APIClient -->|JSON| Controllers
    HTTPClients -->|HTTP| HAPI
    HTTPClients -->|HTTP| WAPI

    style UI fill:#4caf50,color:#fff,stroke:#2e7d32,stroke-width:2px
    style Store fill:#4caf50,color:#fff,stroke:#2e7d32,stroke-width:2px
    style Controllers fill:#ff9800,color:#fff,stroke:#e5100,stroke-width:2px
    style Services fill:#ff9800,color:#fff,stroke:#e5100,stroke-width:2px
    style HAPI fill:#9c27b0,color:#fff,stroke:#6a1b9a,stroke-width:2px
    style WAPI fill:#03a9f4,color:#fff,stroke:#01579b,stroke-width:2px

```

Data Flow Architecture

```

sequenceDiagram
    actor User
    participant Frontend as Vue 3 Frontend
    participant Store as Pinia Store
    participant API as .NET Backend
    participant HYDRA as HYDRA API
    participant Weather as Weather API

    User->>Frontend: Select Site & Date Range
    Frontend->>Store: Update UI State
    Store->>API: POST /api/energy/consumption

    API->>API: Check Authentication

    alt Token Missing or Expired
        API->>HYDRA: POST /connect/token
        HYDRA-->>API: Bearer Token
        API->>API: Store Token
    end

    API->>HYDRA: POST /Sensor/exportAggregatedNumbers
    Note over API,HYDRA: With Bearer Token
    HYDRA-->>API: Energy Data (JSON)

    API->>API: Transform Data<br/>(Calculate Consumption)

    par Fetch Weather Data
        API->>Weather: GET /history.json
        Weather-->>API: Weather Data
    end

    API->>API: Aggregate Response
    API-->>Store: Combined Data

    Store->>Store: Update State

    par Trigger Analytics
        Store->>API: POST /api/analytics/moving-average
        API-->>Store: Moving Average Data

        Store->>API: POST /api/analytics/anomalies
        API-->>Store: Anomaly Data

        Store->>API: POST /api/forecast/predict
        API-->>Store: Forecast Data

        Store->>API: POST /api/insights/generate
        API-->>Store: Insights
    end

    Store->>Frontend: Update Components
    Frontend->>User: Display Dashboard

```

Data Flow Steps:

1. **User Input:** Site selection and date range
2. **State Update:** Pinia store manages application state
3. **API Request:** Frontend makes HTTP request to backend

4. **Authentication:** Backend authenticates with HYDRA (if needed)
 5. **Data Fetching:** Backend fetches energy and weather data
 6. **Data Processing:** Backend performs analytics calculations
 7. **Response Aggregation:** Backend combines all data
 8. **State Update:** Frontend stores update with new data
 9. **UI Render:** Components reactively re-render
 10. **Display:** User sees updated dashboard
-

Frontend Architecture

```

graph TB
    subgraph "Vue 3 Application"
        subgraph "Views"
            Dashboard[Dashboard View]
        end

        subgraph "Components"
            DatePicker[Date Range Picker]
            SiteSelector[Site Selector]
            EnergyChart[Energy Chart<br/>Chart.js]
            SummaryCards[Summary Cards]
            ForecastDisplay[Forecast Display]
            WeatherCorr[Weather Correlation]
            InsightsPanel[Insights Panel]
            LoadingSpinner[Loading Spinner]
            ErrorAlert[Error Alert]
        end

        subgraph "State Management - Pinia"
            EnergyStore[Energy Store]
            WeatherStore[Weather Store]
            AnalyticsStore[Analytics Store]
            ForecastStore[Forecast Store]
            InsightsStore[Insights Store]
            UIStore[UI Store]
        end

        subgraph "Services"
            APIService[API Service<br/>Axios]
        end

        subgraph "Composables"
            DateFormatter[Date Formatter]
            NumberFormatter[Number Formatter]
        end

        Dashboard --> DatePicker
        Dashboard --> SiteSelector
        Dashboard --> EnergyChart
        Dashboard --> SummaryCards
        Dashboard --> ForecastDisplay
        Dashboard --> WeatherCorr
        Dashboard --> InsightsPanel
        Dashboard --> LoadingSpinner
        Dashboard --> ErrorAlert

        DatePicker --> UIStore
        SiteSelector --> UIStore
        EnergyChart --> EnergyStore
        EnergyChart --> AnalyticsStore
        EnergyChart --> WeatherStore
        SummaryCards --> EnergyStore
        SummaryCards --> AnalyticsStore
        ForecastDisplay --> ForecastStore
        WeatherCorr --> WeatherStore
        WeatherCorr --> EnergyStore
        InsightsPanel --> InsightsStore
        LoadingSpinner --> UIStore
        ErrorAlert --> UIStore

        EnergyStore --> APIService
        WeatherStore --> APIService
    
```

```
AnalyticsStore --> APIService
ForecastStore --> APIService
InsightsStore --> APIService

DatePicker --> DateFormatter
EnergyChart --> NumberFormatter
end

APIService -->|HTTP| BackendAPI[Backend API]

style Dashboard fill:#4caf50,color:#fff
style EnergyChart fill:#2196f3,color:#fff
style EnergyStore fill:#ff9800,color:#fff
style APIService fill:#f44336,color:#fff
```

Frontend Key Features:

1. **Reactive Components:** Vue 3 Composition API for reactive data binding
 2. **Centralized State:** Pinia stores for predictable state management
 3. **Type Safety:** TypeScript for compile-time type checking
 4. **Reusable Composables:** Shared logic across components
 5. **Error Handling:** Centralized error management in UI store
 6. **Loading States:** Global and component-level loading indicators
-

Backend Architecture

```

graph TD
    subgraph "API Layer"
        AC[Auth Controller]
        EC[Energy Controller]
        ANC[Analytics Controller]
        FC[Forecast Controller]
        WC[Weather Controller]
        IC[Insights Controller]
        HC[Health Controller]
    end

    subgraph "Service Layer"
        HAS[HYDRA API Service]
        WS[Weather Service]
        ANS[Analytics Service]
        FS[Forecast Service]
        IS[Insights Service]
    end

    subgraph "Model Layer"
        Models[DTOs & Models]
        Config[Configuration]
    end

    subgraph "Infrastructure"
        HTTPClient[HTTP Client Factory<br/>with Polly Policies]
        Logger[Logging Service]
        CORS[CORS Middleware]
        ErrorHandler[Error Handler]
    end

    AC --> HAS
    EC --> HAS
    EC --> WS
    ANC --> ANS
    FC --> FS
    WC --> WS
    IC --> IS

    HAS --> HTTPClient
    WS --> HTTPClient

    ANS --> Models
    FS --> Models
    IS --> Models

    AC --> Logger
    EC --> Logger
    ANC --> Logger
    FC --> Logger
    WC --> Logger
    IC --> Logger

    HTTPClient --> Logger

    style AC fill:#ff6f00,color:#fff
    style EC fill:#ff6f00,color:#fff
    style HAS fill:#1976d2,color:#fff
    style WS fill:#1976d2,color:#fff
    style ANS fill:#1976d2,color:#fff
    style HTTPClient fill:#388e3c,color:#fff
    style Logger fill:#d32f2f,color:#fff

```

Backend Layers:

1. Controller Layer

- **Responsibility:** Handle HTTP requests, validate input, return responses
- **Pattern:** RESTful API design
- **Error Handling:** Global exception middleware

2. Service Layer

- **Responsibility:** Business logic, data processing, external API integration
- **Pattern:** Dependency injection
- **Key Services:**
- **Hydra ApiService:** Authentication and energy data fetching
- **WeatherService:** Weather data integration
- **AnalyticsService:** Moving average and anomaly detection
- **ForecastService:** Energy consumption forecasting
- **InsightsService:** Natural language insight generation

3. Infrastructure Layer

- **HTTP Client Factory:** Resilient HTTP calls with retry and circuit breaker
 - **Logging:** Structured logging for debugging and monitoring
 - **CORS:** Cross-origin resource sharing configuration
 - **Configuration:** Environment-based settings
-

Authentication Flow

```

sequenceDiagram
    participant Client as Frontend
    participant API as Backend API
    participant Auth as HYDRA Identity<br/>Server
    participant Data as HYDRA Data<br/>API

    Note over API: First Request Received

    API->>API: Check Token Cache

    alt Token Not Cached or Expired
        API->>Auth: POST /connect/token
        Note over API,Auth: Content-Type: application/x-www-form-urlencoded<br/>Body:
        client_id, client_secret, grant_type, etc.

        Auth->>Auth: Validate Credentials

        alt Valid Credentials
            Auth-->>API: 200 OK<br/>{access_token, expires_in, token_type}
            API->>API: Cache Token<br/>(expires_in seconds)
            Note over API: Token Valid for ~30 days
        else Invalid Credentials
            Auth-->>API: 401 Unauthorized
            API-->>Client: 500 Internal Server Error<br/>{error: "Authentication
failed"}
        end
    end

    Note over API: Token Available

    API->>Data: POST /Sensor/exportAggregatedNumbers
    Note over API,Data: Authorization: Bearer {token}

    alt Valid Token
        Data->>Data: Authorize Request
        Data->>Data: Fetch Energy Data
        Data-->>API: 200 OK<br/>{energy data array}
        API-->>Client: 200 OK<br/>{processed data}
    else Invalid/Expired Token
        Data-->>API: 401 Unauthorized
        API->>API: Clear Token Cache
        API->>Auth: POST /connect/token<br/>(Retry Authentication)
        Note over API: Automatic retry mechanism
    end

```

Authentication Details:

Credentials (from case study):

```
URL: https://identity.hydra.africa/connect/token
Method: POST
Content-Type: application/x-www-form-urlencoded

Body:
- client_id: ro.client
- client_secret: secret
- grant_type: password
- scope: api1
- username: ll-wc-04@hydra.africa
- password: CpBzdnYM7Qb6b4q
```

Token Management:

- Tokens cached in memory for 30 days (expires_in value)
 - Automatic re-authentication on 401 responses
 - Circuit breaker prevents repeated failed auth attempts
-

API Architecture

```

graph LR
    subgraph "Public Endpoints"
        E1[GET /health]
        E2[POST /api/auth/login]
    end

    subgraph "Energy Endpoints"
        E3[GET /api/energy/sites]
        E4[POST /api/energy/consumption]
    end

    subgraph "Analytics Endpoints"
        E5[POST /api/analytics/moving-average]
        E6[POST /api/analytics/anomalies]
        E7[POST /api/analytics/statistics]
    end

    subgraph "Forecast Endpoints"
        E8[POST /api/forecast/predict]
    end

    subgraph "Weather Endpoints"
        E9[GET /api/weather/current]
        E10[GET /api/weather/historical]
    end

    subgraph "Insights Endpoints"
        E11[POST /api/insights/generate]
    end

    style E1 fill:#4caf50,color:#fff
    style E2 fill:#2196f3,color:#fff
    style E3 fill:#ff9800,color:#fff
    style E4 fill:#ff9800,color:#fff
    style E5 fill:#9c27b0,color:#fff
    style E6 fill:#9c27b0,color:#fff
    style E7 fill:#9c27b0,color:#fff
    style E8 fill:#f44336,color:#fff
    style E9 fill:#00bcd4,color:#fff
    style E10 fill:#00bcd4,color:#fff
    style E11 fill:#e91e63,color:#fff

```

API Endpoint Reference:

Endpoint	Method	Description	Request	Response
/health	GET	Health check	-	Status
/api/auth/login	POST	Test HYDRA auth	-	Token info
/api/energy/sites	GET	Get available sites	-	Site list
/api/energy/consumption	POST	Get energy data	siteld, startDate, endDate	Energy data array
/api/analytics/moving-average	POST	Calculate MA	energyData, windowSize	MA values
/api/analytics/anomalies	POST	Detect anomalies	energyData	Anomaly list
/api/analytics/statistics	POST	Calculate stats	energyData	Statistics
/api/forecast/predict	POST	Predict 3 days	energyData	Forecast array
/api/weather/current	GET	Current weather	location	Weather data
/api/weather/historical	GET	Historical weather	location, date	Weather data
/api/insights/generate	POST	Generate insights	energyData, anomalies, weather	Insights array

Deployment Architecture

Docker Deployment

```

graph TB
    subgraph "Docker Host"
        subgraph "Docker Network: hydra-network"
            subgraph "Frontend Container"
                Nginx[Nginx Web Server<br/>Port: 80]
                VueBuild[Vue 3 Build<br/>Static Files]
                Nginx --> VueBuild
            end

            subgraph "Backend Container"
                DotNet[.NET 9 Runtime<br/>Port: 5000]
                API[Backend API<br/>Application]
                DotNet --> API
            end

            Nginx -.->|API Proxy| DotNet
        end
    end

    subgraph "External Services"
        HYDRA[HYDRA API]
        WEATHER[Weather API]
    end

    User[User Browser] -->|Port 3000| Nginx
    API -->|HTTPS| HYDRA
    API -->|HTTPS| WEATHER

    style Nginx fill:#4CAF50,color:#fff
    style DotNet fill:#FF9800,color:#fff
    style HYDRA fill:#9C27B0,color:#fff
    style WEATHER fill:#2196F3,color:#fff

```

Docker Compose Configuration:

```
services:
  backend:
    build: ./Backend
    ports: ["5000:5000"]
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - HYDRA_AUTH_URL=https://identity.hydra.africa/connect/token
      - HYDRA_API_URL=https://hydra-api.azurewebsites.net
      - WEATHER_API_KEY=${WEATHER_API_KEY}
    networks: [hydra-network]
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
    interval: 30s

  frontend:
    build: ./Frontend
    ports: ["3000:80"]
    environment:
      - VITE_API_BASE_URL=http://localhost:5000/api
  depends_on:
    backend: {condition: service_healthy}
  networks: [hydra-network]
```

Cloud Deployment (Azure Example)

```

graph TD
    subgraph "Azure Cloud"
        subgraph "Azure App Service"
            FrontendApp[Frontend<br/>App Service<br/>Node.js/Static]
            BackendApp[Backend<br/>App Service<br/>.NET 9]
        end

        subgraph "Azure Services"
            AppInsights[Application<br/>Insights]
            KeyVault[Key Vault<br/>Secrets]
        end

        FrontendApp -->|API Calls| BackendApp
        BackendApp --> AppInsights
        BackendApp --> KeyVault
    end

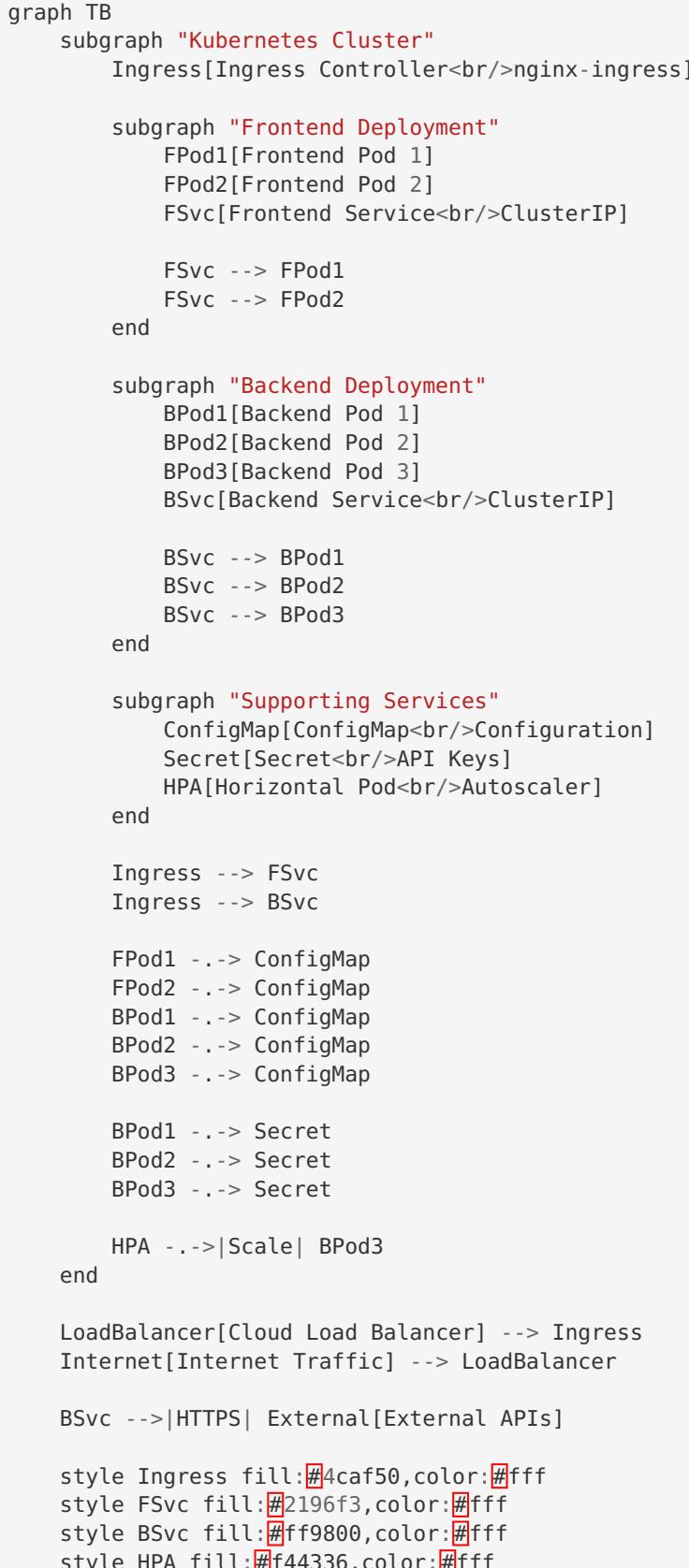
    subgraph "External"
        CDN[Azure CDN<br/>Static Assets]
        HYDRA[HYDRA API]
        WEATHER[Weather API]
    end

    Internet[Internet Users] --> CDN
    CDN --> FrontendApp
    BackendApp --> HYDRA
    BackendApp --> WEATHER

    style FrontendApp fill:#4CAF50,color:#fff
    style BackendApp fill:#FF9800,color:#fff
    style AppInsights fill:#2196F3,color:#fff
    style KeyVault fill:#F44336,color:#fff

```

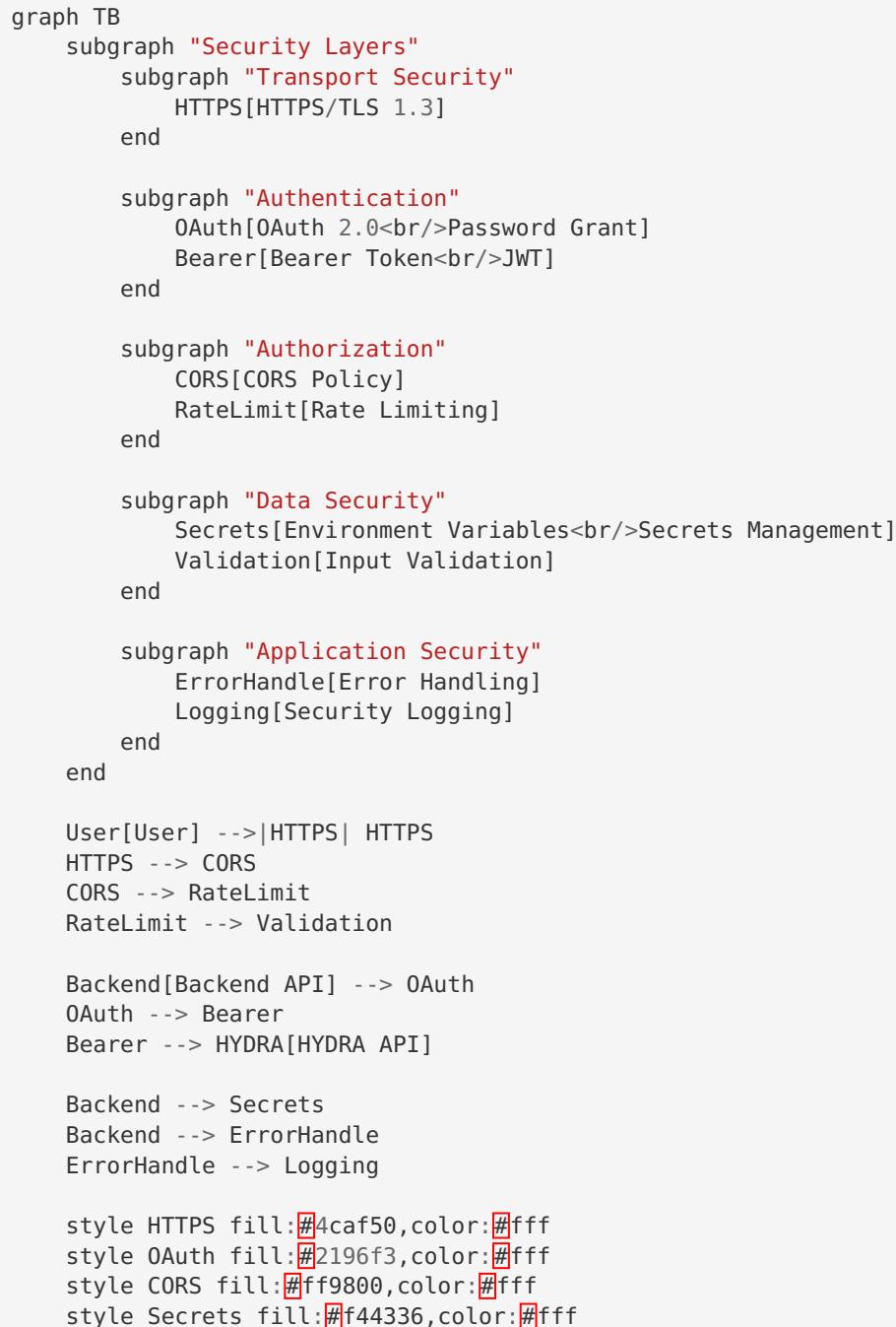
Kubernetes Deployment



Technology Stack

```
mindmap
root((HYDRA<br/>Dashboard))
  Frontend
    Vue 3
      Composition API
      TypeScript
      Vite
    State Management
      Pinia
  UI
    Tailwind CSS
    Chart.js
  HTTP Client
    Axios
  Routing
    Vue Router
  Backend
    .NET 9
    ASP.NET Core
    C# 13
  HTTP Client
    Polly
    Resilience Policies
  API Documentation
    Swagger/OpenAPI
  Logging
    Serilog
    Console
  DevOps
    Containerization
      Docker
      Docker Compose
    Web Server
      Nginx
    CI/CD
      GitHub Actions
      Azure DevOps
  Cloud
    Azure
    AWS
    Google Cloud
  External APIs
    HYDRA API
      Energy Data
      Device Info
    Weather API
      Historical Data
      Current Weather
  Identity Server
    OAuth 2.0
    Bearer Tokens
```

Security Architecture



Security Measures:

1. Transport Security:

- HTTPS for all communications
- TLS 1.2+ encryption

2. Authentication:

- OAuth 2.0 with HYDRA Identity Server
- Bearer token authentication
- Token caching and automatic refresh

3. Authorization:

- CORS policy (configurable origins)
- Rate limiting (future enhancement)

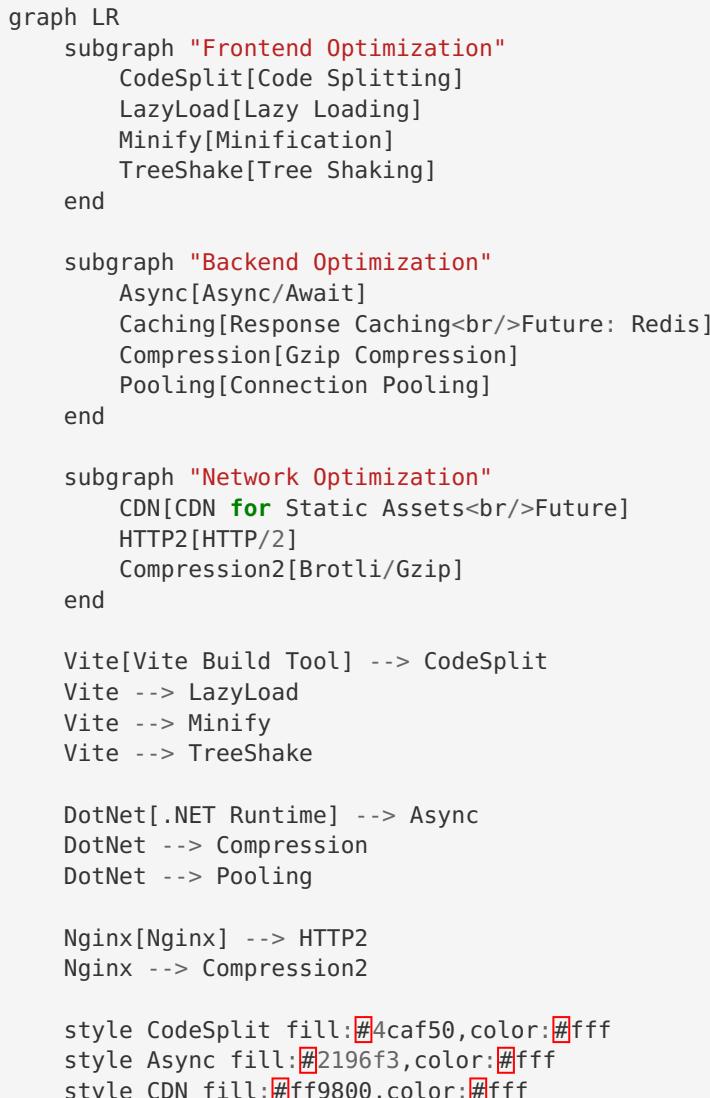
4. Data Security:

- Environment variables for secrets
- No hardcoded credentials
- Input validation on all endpoints

5. Application Security:

- Global error handling
- Security logging
- No sensitive data in logs

Performance Optimization



Monitoring & Observability

```

graph TD
    subgraph "Application"
        Frontend[Frontend<br/>Vue 3]
        Backend[Backend<br/>.NET 9]
    end

    subgraph "Logging"
        Console[Console Logging]
        File[File Logging<br/>Future]
    end

    subgraph "Monitoring - Future"
        AppInsights[Application Insights<br/>Azure]
        Prometheus[Prometheus<br/>Metrics]
        Grafana[Grafana<br/>Dashboards]
    end

    subgraph "Health Checks"
        Health[/health Endpoint]
        Ready[/health/ready<br/>Future]
    end

    Frontend --> Console
    Backend --> Console
    Backend --> Health

    Backend -.->|Future| AppInsights
    Backend -.->|Future| Prometheus
    Prometheus -.->|Future| Grafana

    style Frontend fill:#4CAF50,color:#fff
    style Backend fill:#FF9800,color:#fff
    style Health fill:#2196F3,color:#fff

```

Error Handling Flow

```

graph TB
    Request[Incoming Request] --> Validation{Input<br/>Validation}
    Validation -->|Invalid| BadRequest[400 Bad Request]
    Validation -->|Valid| Auth{Authentication}
    Auth -->|Failed| AuthFailed[500 Auth Error]
    Auth -->|Success| Processing[Process Request]
    Processing --> ExternalAPI{External<br/>API Call}
    ExternalAPI -->|Timeout| Retry{Retry<br/>Policy}
    ExternalAPI -->|Error| Retry
    ExternalAPI -->|Success| Transform[Transform Data]
    Retry -->|Max Retries| ServiceError[503 Service Unavailable]
    Retry -->|Retry Success| Transform
    Transform --> Success[200 OK + Data]
    BadRequest --> Logger[Log Error]
    AuthFailed --> Logger
    ServiceError --> Logger
    Logger --> Client[Return to Client]
    Success --> Client
    style Success fill:#4CAF50,color:#fff
    style BadRequest fill:#FF9800,color:#000
    style AuthFailed fill:#F44336,color:#fff
    style ServiceError fill:#F44336,color:#fff
  
```

Scalability Considerations

For future scaling to 500+ sites with real-time updates (see Technical Question 5), the architecture would evolve to:



Conclusion

This architecture provides:

- ✓ Modularity:** Clear separation of concerns
- ✓ Scalability:** Can be extended to support high load
- ✓ Maintainability:** Clean code structure and documentation
- ✓ Security:** Multiple layers of security controls
- ✓ Performance:** Optimized for fast loading and responsiveness

 **Observability:** Comprehensive logging and health checks

 **Flexibility:** Easy to add new features and integrations

The current implementation is optimized for the case study requirements while being designed with future scalability in mind.