Real-Time Grid Decarbonization Dashboard - Technical Breakdown

Project Overview

A 4-6 hour Streamlit-based dashboard that visualizes real-time renewable energy penetration across U.S. power grids using the GridStatus library and EIA API. This dashboard serves as a “compass” for grid decarbonization efforts, providing actionable insights for energy analysts, grid operators, and sustainability professionals.

1. Technical Architecture Components

Backend Data Processing

Primary Data Sources

- **GridStatus Library**: Unified API for ISO data (CAISO, ERCOT, PJM, NYISO, MISO, ISONE, SPP)
- **EIA API v2**: Hourly balancing authority data via `/v2/electricity/rto/region-sub-ba-data/`
- **Real-time feeds**: 5-15 minute intervals from ISOs vs. hourly from EIA

Data Pipeline Architecture

```python
# Core data fetching structure
import streamlit as st
from gridstatus import CAISO, ERCOT, PJM, NYISO, MISO
import requests
import pandas as pd
from datetime import datetime, timedelta

@st.cache_data(ttl=300)  # 5-minute cache
def fetch_iso_data(iso_name, start_time, end_time):
    """Fetch real-time generation data from specific ISO"""
    iso_map = {
        'CAISO': CAISO(),
        'ERCOT': ERCOT(),
        'PJM': PJM(),
        'NYISO': NYISO(),
        'MISO': MISO()
    }

    iso_client = iso_map[iso_name]
    return iso_client.get_fuel_mix(start=start_time, end=end_time)

@st.cache_data(ttl=3600)  # 1-hour cache for EIA data
def fetch_eia_demand_data(region, start_date, end_date):
    """Fetch demand data from EIA API"""
    api_key = st.secrets["EIA_API_KEY"]
    url = f"https://api.eia.gov/v2/electricity/rto/region-sub-ba-data/data"

    params = {
        'api_key': api_key,
        'frequency': 'hourly',
        'data[0]': 'value',
        'facets[parent][]': region,
        'start': start_date,
        'end': end_date,
        'sort[0][column]': 'period',
        'sort[0][direction]': 'desc',
        'length': 5000
    }

    response = requests.get(url, params=params)
    return pd.DataFrame(response.json()['response']['data'])
```

Caching Strategy

- **Real-time data**: 5-minute TTL using `@st.cache_data`
- **Historical data**: 1-hour TTL for EIA API calls
- **Computed metrics**: Session-level caching for derived calculations
- **Error handling**: Graceful fallback to cached data on API failures

Real-time Update Mechanism

```
# Auto-refresh implementation
def setup_auto_refresh():
    """Configure automatic dashboard refresh"""
    refresh_interval = st.sidebar.selectbox(
        "Refresh Interval",
        options=[60, 300, 900],  # 1min, 5min, 15min
        index=1,
        format_func=lambda x: f"{x//60} minutes"
    )

    # JavaScript-based auto-refresh
    st.markdown(f"""
    <script>
    setTimeout(function(){{
        window.location.reload();
    }}, {refresh_interval * 1000});
    </script>
    """, unsafe_allow_html=True)
```

API Integration Specifications

GridStatus Library Setup

```
# Environment configuration (.env file)
EIA_API_KEY=your_eia_api_key_here
ERCOT_USERNAME=your_ercot_username  # if using ERCOT API directly
ERCOT_PASSWORD=your_ercot_password

# Installation and imports
# pip install gridstatus plotly streamlit pandas
import os
from dotenv import load_dotenv
load_dotenv()
```

EIA API Integration

```
# Direct EIA API calls for additional data
class EIAClient:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://api.eia.gov/v2"

    def get_renewable_generation(self, region, start_date, end_date):
        """Fetch renewable generation data by fuel type"""
        endpoint = f"{self.base_url}/electricity/rto/fuel-type-data/data"

        params = {
            'api_key': self.api_key,
            'frequency': 'hourly',
            'data[0]': 'value',
            'facets[parent][]': region,
            'facets[fueltype][]': ['SUN', 'WND', 'WAT', 'WAS'],  # Solar, Wind, Hydro,
Biomass
            'start': start_date,
            'end': end_date,
            'sort[0][column]': 'period',
            'sort[0][direction]': 'desc'
        }

        response = requests.get(endpoint, params=params)
        if response.status_code == 200:
            return pd.DataFrame(response.json()['response']['data'])
        else:
            st.error(f"EIA API Error: {response.status_code}")
            return pd.DataFrame()
```

2. UI/UX Design Specifications

Layout Framework

- **Grid System**: 3-column responsive layout using Streamlit's `st.columns()`
- **Color Scheme**: Green-blue gradient palette reflecting renewable energy themes
- **Typography**: Clean, modern fonts with clear hierarchy (titles, metrics, labels)

```
# Color palette configuration
COLORS = {
    'renewable': '#2E8B57',      # Sea Green
    'fossil': '#CD5C5C',         # Indian Red
    'nuclear': '#4682B4',        # Steel Blue
    'storage': '#9370DB',        # Medium Purple
    'background': '#F8F9FA',     # Light Gray
    'text': '#212529',           # Dark Gray
    'accent': '#28A745'          # Success Green
}

# Custom CSS styling
def load_custom_css():
    st.markdown(f"""
    <style>
    .main-header {{
        background: linear-gradient(90deg, {COLORS['renewable']} 0%,
{COLORS['accent']} 100%);
        color: white;
        padding: 1rem;
        border-radius: 0.5rem;
        margin-bottom: 2rem;
    }}

    .metric-card {{
        background: white;
        border: 1px solid #e9ecef;
        border-radius: 0.5rem;
        padding: 1.5rem;
        box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    }}

    .renewable-text {{ color: {COLORS['renewable']}; font-weight: bold; }}
    .fossil-text {{ color: {COLORS['fossil']}; font-weight: bold; }}
    </style>
    """, unsafe_allow_html=True)
```

Responsive Design Principles

- **Desktop**: Full 3-column layout with side panels
- **Tablet**: 2-column layout with collapsible sidebar
- **Mobile**: Single-column stacked layout with swipeable charts

```
# Responsive layout implementation
def create_responsive_layout():
    # Detect screen size using JavaScript
    screen_width = st.sidebar.number_input("Screen Width", value=1200, key="screen_widt
h")

    if screen_width >= 1200:
        # Desktop layout
        col1, col2, col3 = st.columns([1, 2, 1])
        return col1, col2, col3, "desktop"
    elif screen_width >= 768:
        # Tablet layout
        col1, col2 = st.columns([1, 2])
        return col1, col2, None, "tablet"
    else:
        # Mobile layout
        return st.container(), None, None, "mobile"
```

Typography and Visual Hierarchy

```
# Typography system
def apply_typography():
    st.markdown("""
    <style>
    .big-font { font-size: 2.5rem; font-weight: 700; line-height: 1.2; }
    .medium-font { font-size: 1.5rem; font-weight: 600; line-height: 1.3; }
    .small-font { font-size: 1rem; font-weight: 400; line-height: 1.4; }
    .caption-font { font-size: 0.875rem; font-weight: 400; color: #6c757d; }
    </style>
    """, unsafe_allow_html=True)
```

3. Pre-built Component Libraries

Streamlit Components

```
# Core Streamlit components for energy dashboards
import streamlit as st
import streamlit_echarts as st_echarts  # Advanced charting
import streamlit_agrid as st_agrid      # Data tables
import streamlit_plotly_events as plotly_events  # Interactive charts

# Key component imports
essential_components = [
    'st.metric',           # KPI display cards
    'st.plotly_chart',     # Interactive visualizations
    'st.selectbox',        # Region/ISO selection
    'st.date_input',       # Time range selection
    'st.multiselect',      # Fuel type filtering
    'st.progress',         # Renewable penetration progress bars
    'st.columns',          # Responsive grid layout
    'st.expander',         # Collapsible detail sections
    'st.sidebar',          # Control panel
    'st.container'         # Layout containers
]
```

Plotly Chart Library

```
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Chart templates for energy visualization
def create_renewable_penetration_chart(data):
    """Stacked area chart showing renewable vs non-renewable generation"""
    fig = px.area(
        data,
        x='timestamp',
        y='value',
        color='fuel_type',
        color_discrete_map={
            'Solar': '#FFD700',
            'Wind': '#87CEEB',
            'Hydro': '#4682B4',
            'Natural Gas': '#CD5C5C',
            'Coal': '#2F4F4F',
            'Nuclear': '#9370DB'
        },
        title="Real-Time Generation Mix"
    )

    fig.update_layout(
        hovermode='x unified',
        legend=dict(orientation="h", yanchor="bottom", y=1.02),
        height=400
    )

    return fig

def create_penetration_gauge(renewable_percentage):
    """Gauge chart for current renewable penetration"""
    fig = go.Figure(go.Indicator(
        mode = "gauge+number+delta",
        value = renewable_percentage,
        domain = {'x': [0, 1], 'y': [0, 1]},
        title = {'text': "Renewable Penetration %"},
        delta = {'reference': 50},  # Target threshold
        gauge = {
            'axis': {'range': [None, 100]},
            'bar': {'color': COLORS['renewable']},
            'steps': [
                {'range': [0, 25], 'color': "lightgray"},
                {'range': [25, 50], 'color': "yellow"},
                {'range': [50, 75], 'color': "orange"},
                {'range': [75, 100], 'color': "green"}
            ],
            'threshold': {
                'line': {'color': "red", 'width': 4},
                'thickness': 0.75,
                'value': 90
            }
        }
    ))

    return fig
```

Mapping Libraries

```
import plotly.express as px
import folium
from streamlit_folium import st_folium

def create_iso_map(iso_data):
    """Interactive map showing renewable penetration by region"""

    # ISO region coordinates
    iso_coords = {
        'CAISO': {'lat': 36.7783, 'lon': -119.4179, 'name': 'California ISO'},
        'ERCOT': {'lat': 31.0000, 'lon': -100.0000, 'name': 'Texas (ERCOT)'},
        'PJM': {'lat': 40.0000, 'lon': -79.0000, 'name': 'PJM Interconnection'},
        'NYISO': {'lat': 43.0000, 'lon': -75.0000, 'name': 'New York ISO'},
        'MISO': {'lat': 44.0000, 'lon': -89.0000, 'name': 'Midcontinent ISO'}
    }

    # Create Plotly map
    fig = px.scatter_mapbox(
        data_frame=iso_data,
        lat='latitude',
        lon='longitude',
        size='total_capacity',
        color='renewable_percentage',
        hover_name='iso_name',
        hover_data=['renewable_mw', 'total_mw'],
        color_continuous_scale='RdYlGn',
        size_max=50,
        zoom=3,
        mapbox_style='open-street-map'
    )

    fig.update_layout(height=500)
    return fig
```

4. Data Visualization Requirements

Essential Chart Types

Primary KPI Visualizations

```
# Key Performance Indicators for renewable penetration
renewable_kpis = {
    'penetration_rate': {
        'formula': '(Renewable Generation ÷ Total Demand) × 100',
        'visualization': 'gauge',
        'target': '>50%',
        'update_frequency': '5min'
    },
    'capacity_factor': {
        'formula': '(Actual Output ÷ Nameplate Capacity) × 100',
        'visualization': 'line_chart',
        'target': '>35% for wind, >25% for solar',
        'update_frequency': 'hourly'
    },
    'curtailment_rate': {
        'formula': '(Curtailed Generation ÷ Available Generation) × 100',
        'visualization': 'bar_chart',
        'target': '<5%',
        'update_frequency': 'hourly'
    },
    'carbon_intensity': {
        'formula': 'CO2 emissions ÷ Total Generation (lbs/MWh)',
        'visualization': 'line_chart',
        'target': '<800 lbs/MWh',
        'update_frequency': 'hourly'
    }
}

def create_kpi_dashboard(data):
    """Create comprehensive KPI dashboard"""

    # Calculate current metrics
    current_renewable_pct = calculate_renewable_percentage(data)
    current_capacity_factor = calculate_capacity_factor(data)
    current_curtailment = calculate_curtailment_rate(data)

    # Display KPI cards
    col1, col2, col3, col4 = st.columns(4)

    with col1:
        st.metric(
            label="Renewable Penetration",
            value=f"{current_renewable_pct:.1f}%",
            delta=f"{current_renewable_pct - 45:.1f}%"  # vs. 45% baseline
        )

    with col2:
        st.metric(
            label="Capacity Factor",
            value=f"{current_capacity_factor:.1f}%",
            delta=f"{current_capacity_factor - 32:.1f}%"  # vs. 32% baseline
        )

    with col3:
        st.metric(
            label="Curtailment Rate",
            value=f"{current_curtailment:.1f}%",
            delta=f"{current_curtailment - 3:.1f}%",  # vs. 3% baseline
            delta_color="inverse"  # Lower is better
        )

    with col4:
```

```
        carbon_intensity = calculate_carbon_intensity(data)
        st.metric(
            label="Carbon Intensity",
            value=f"{carbon_intensity:.0f} lbs/MWh",
            delta=f"{carbon_intensity - 850:.0f}",  # vs. 850 baseline
            delta_color="inverse"
        )
```

Time Series Visualizations

```python
def create_generation_timeline(data):
    """Multi-series time-series chart for generation sources"""

    fig = make_subplots(
        rows=2, cols=1,
        subplot_titles=('Generation by Source', 'Renewable Penetration %'),
        specs=[[{"secondary_y": False}], [{"secondary_y": True}]],
        vertical_spacing=0.1
    )

    # Stacked area chart for generation
    renewable_sources = ['Solar', 'Wind', 'Hydro', 'Geothermal', 'Biomass']
    fossil_sources = ['Natural Gas', 'Coal', 'Oil']

    for source in renewable_sources:
        fig.add_trace(
            go.Scatter(
                x=data['timestamp'],
                y=data[source],
                mode='lines',
                stackgroup='renewable',
                name=source,
                line=dict(color=RENEWABLE_COLORS[source])
            ),
            row=1, col=1
        )

    for source in fossil_sources:
        fig.add_trace(
            go.Scatter(
                x=data['timestamp'],
                y=data[source],
                mode='lines',
                stackgroup='fossil',
                name=source,
                line=dict(color=FOSSIL_COLORS[source])
            ),
            row=1, col=1
        )

    # Penetration percentage line
    fig.add_trace(
        go.Scatter(
            x=data['timestamp'],
            y=data['renewable_percentage'],
            mode='lines+markers',
            name='Renewable %',
            line=dict(color=COLORS['accent'], width=3)
        ),
        row=2, col=1
    )

    fig.update_layout(height=600, hovermode='x unified')
    return fig
```

Chart Specifications by Metric Type

Renewable Penetration Tracking

- **Primary**: Stacked area chart (renewable vs. fossil generation)

- **Secondary**: Gauge chart (current penetration percentage)
- **Tertiary**: Sparklines (24-hour trend indicators)

Geographic Distribution

- **Primary**: Choropleth map (penetration by ISO region)
- **Secondary**: Bubble map (capacity by location)
- **Tertiary**: Heat map (time vs. region penetration matrix)

Operational Metrics

- **Primary**: Line charts (capacity factors over time)
- **Secondary**: Bar charts (curtailment by source)
- **Tertiary**: Scatter plots (efficiency correlations)

5. User Interaction Design

Filter Controls

```
def create_control_panel():
    """Comprehensive filter and control system"""

    st.sidebar.title("🔋 Grid Controls")

    # ISO Region Selection
    selected_isos = st.sidebar.multiselect(
        "Select ISO Regions",
        options=['CAISO', 'ERCOT', 'PJM', 'NYISO', 'MISO', 'ISONE', 'SPP'],
        default=['CAISO', 'ERCOT', 'PJM'],
        help="Choose which grid regions to display"
    )

    # Time Range Selection
    time_range = st.sidebar.radio(
        "Time Range",
        options=['Last 24 Hours', 'Last 7 Days', 'Last 30 Days', 'Custom'],
        index=0
    )

    if time_range == 'Custom':
        start_date = st.sidebar.date_input("Start Date")
        end_date = st.sidebar.date_input("End Date")
    else:
        days_back = {'Last 24 Hours': 1, 'Last 7 Days': 7, 'Last 30 Days': 30}
[time_range]
        end_date = datetime.now()
        start_date = end_date - timedelta(days=days_back)

    # Fuel Type Filtering
    st.sidebar.subheader("⚡ Generation Sources")

    show_renewables = st.sidebar.checkbox("Renewable Sources", value=True)
    if show_renewables:
        renewable_types = st.sidebar.multiselect(
            "Renewable Types",
            options=['Solar', 'Wind', 'Hydro', 'Geothermal', 'Biomass'],
            default=['Solar', 'Wind', 'Hydro']
        )

    show_conventional = st.sidebar.checkbox("Conventional Sources", value=True)
    if show_conventional:
        conventional_types = st.sidebar.multiselect(
            "Conventional Types",
            options=['Natural Gas', 'Coal', 'Nuclear', 'Oil'],
            default=['Natural Gas', 'Nuclear']
        )

    # View Mode Selection
    view_mode = st.sidebar.selectbox(
        "Dashboard View",
        options=['Real-Time', 'Historical Analysis', 'Forecasting', 'Comparison'],
        index=0
    )

    # Advanced Options
    with st.sidebar.expander("🔧 Advanced Options"):
        show_curtailment = st.checkbox("Show Curtailment Data", value=False)
        normalize_data = st.checkbox("Normalize by Capacity", value=False)
        show_carbon_intensity = st.checkbox("Include Carbon Metrics", value=True)

    return {
```

```
        'isos': selected_isos,
        'start_date': start_date,
        'end_date': end_date,
        'renewable_types': renewable_types if show_renewables else [],
        'conventional_types': conventional_types if show_conventional else [],
        'view_mode': view_mode,
        'show_curtailment': show_curtailment,
        'normalize_data': normalize_data,
        'show_carbon_intensity': show_carbon_intensity
    }
```

Drill-down Capabilities

```
def implement_drill_down_functionality():
    """Interactive drill-down from summary to detailed views"""

    # Main chart with click events
    selected_points = plotly_events(
        main_chart,
        click_event=True,
        hover_event=False,
        select_event=True,
        key="main_chart"
    )

    if selected_points:
        selected_iso = selected_points[0]['customdata'][0]
        selected_time = selected_points[0]['x']

        st.subheader(f"📊 Detailed View: {selected_iso} at {selected_time}")

        # Create detailed breakdown
        col1, col2 = st.columns(2)

        with col1:
            # Fuel mix pie chart for selected time
            fuel_mix_data = get_fuel_mix_for_time(selected_iso, selected_time)
            pie_chart = px.pie(
                fuel_mix_data,
                values='generation_mw',
                names='fuel_type',
                title=f"Generation Mix - {selected_iso}"
            )
            st.plotly_chart(pie_chart, use_container_width=True)

        with col2:
            # Hourly trend for selected ISO
            hourly_data = get_hourly_trend(selected_iso, selected_time)
            line_chart = px.line(
                hourly_data,
                x='hour',
                y='renewable_percentage',
                title=f"24-Hour Renewable Trend - {selected_iso}"
            )
            st.plotly_chart(line_chart, use_container_width=True)

        # Detailed data table
        with st.expander("📋 Raw Data"):
            detailed_data = get_detailed_data(selected_iso, selected_time)
            st.dataframe(detailed_data, use_container_width=True)
```

Time Controls

```
def create_time_navigation():
    """Time-based navigation controls"""

    col1, col2, col3, col4, col5 = st.columns([1,1,1,1,1])

    with col1:
        if st.button("⏮️ -24h"):
            st.session_state.time_offset -= 24

    with col2:
        if st.button("⏪ -1h"):
            st.session_state.time_offset -= 1

    with col3:
        if st.button("⏸️ Now"):
            st.session_state.time_offset = 0

    with col4:
        if st.button("⏩ +1h"):
            st.session_state.time_offset += 1

    with col5:
        if st.button("⏭️ +24h"):
            st.session_state.time_offset += 24

    # Time slider for historical navigation
    if st.session_state.get('view_mode') == 'Historical Analysis':
        time_slider = st.slider(
            "Historical Time Navigation",
            min_value=0,
            max_value=len(historical_timestamps)-1,
            value=len(historical_timestamps)-1,
            format="timestamp"
        )

        selected_timestamp = historical_timestamps[time_slider]
        return selected_timestamp

    return datetime.now() + timedelta(hours=st.session_state.get('time_offset', 0))
```

6. Information Display Strategy

Hierarchy and Priority System

Priority Level 1: Critical Real-Time Metrics

```
def display_critical_metrics():
    """Top-priority information always visible"""

    # Header with key system status
    st.markdown("""
    <div class="main-header">
        <h1>🌿 Real-Time Grid Decarbonization Dashboard</h1>
        <div style="display: flex; justify-content: space-between; align-items: cen-
ter;">
            <div>Current Grid Status: <span class="renewable-text">HIGH RENEWABLE</
span></div>
            <div>Last Update: {}</div>
        </div>
    </div>
    """.format(datetime.now().strftime("%H:%M:%S")), unsafe_allow_html=True)

    # Critical KPI row - always visible
    kpi_col1, kpi_col2, kpi_col3, kpi_col4 = st.columns(4)

    with kpi_col1:
        current_penetration = get_current_renewable_penetration()
        st.metric(
            "🌱 Renewable Penetration",
            f"{current_penetration:.1f}%",
            delta=f"{current_penetration - 45:.1f}%"
        )

    with kpi_col2:
        grid_demand = get_current_total_demand()
        st.metric(
            "⚡ Grid Demand",
            f"{grid_demand:,.0f} MW",
            delta=f"{grid_demand - get_yesterday_demand():,.0f} MW"
        )

    with kpi_col3:
        carbon_avoided = calculate_carbon_avoided()
        st.metric(
            "🌍 CO₂ Avoided Today",
            f"{carbon_avoided:,.0f} tons",
            delta=f"{carbon_avoided - get_yesterday_carbon_avoided():,.0f} tons"
        )

    with kpi_col4:
        renewable_capacity = get_renewable_capacity_online()
        st.metric(
            "🔋 Renewable Online",
            f"{renewable_capacity:,.0f} MW",
            delta=f"{renewable_capacity - get_yesterday_capacity():,.0f} MW"
        )
```

Priority Level 2: Regional Analysis

```
def display_regional_analysis():
    """Secondary priority: regional breakdown and trends"""

    st.subheader("🗺️ Regional Grid Analysis")

    # Tab-based organization for regional data
    tab1, tab2, tab3 = st.tabs(["📊 Live Generation", "🏭 Capacity Status", "📈
Trends"])

    with tab1:
        # Real-time generation by region
        col1, col2 = st.columns([2, 1])

        with col1:
            regional_map = create_iso_penetration_map()
            st.plotly_chart(regional_map, use_container_width=True)

        with col2:
            # Top renewable regions
            top_regions = get_top_renewable_regions()
            st.subheader("🏆 Top Renewable Regions")

            for i, region in enumerate(top_regions[:5]):
                st.markdown(f"""
                **{i+1}. {region['iso']}**
                {region['percentage']:.1f}% renewable
                {region['renewable_mw']:,.0f} MW clean energy
                """)

    with tab2:
        # Capacity utilization analysis
        capacity_chart = create_capacity_utilization_chart()
        st.plotly_chart(capacity_chart, use_container_width=True)

    with tab3:
        # Trend analysis
        trend_chart = create_regional_trend_analysis()
        st.plotly_chart(trend_chart, use_container_width=True)
```

Priority Level 3: Detailed Analytics

```
def display_detailed_analytics():
    """Third priority: detailed analysis and forecasting"""

    with st.expander("🔍 Advanced Analytics", expanded=False):

        analysis_type = st.selectbox(
            "Analysis Type",
            ["Curtailment Analysis", "Price Correlation", "Weather Impact", "Forecast-
ing"]
        )

        if analysis_type == "Curtailment Analysis":
            curtailment_data = analyze_renewable_curtailment()
            curtailment_chart = create_curtailment_analysis_chart(curtailment_data)
            st.plotly_chart(curtailment_chart, use_container_width=True)

            st.markdown(f"""
            **Curtailment Summary:**
            - Total curtailed today: {curtailment_data['total_curtailed']:,.0f} MWh
            - Primary cause: {curtailment_data['primary_cause']}
            - Estimated value lost: ${curtailment_data['value_lost']:,.0f}
            """)

        elif analysis_type == "Price Correlation":
            price_renewable_correlation = analyze_price_renewable_correlation()
            correlation_chart = cre-
ate_price_correlation_chart(price_renewable_correlation)
            st.plotly_chart(correlation_chart, use_container_width=True)

        elif analysis_type == "Forecasting":
            forecast_data = generate_renewable_forecast()
            forecast_chart = create_forecast_chart(forecast_data)
            st.plotly_chart(forecast_chart, use_container_width=True)
```

Real-time vs Historical Data Strategy

```python
def manage_data_display_strategy():
    """Smart data display based on user context and data freshness"""

    # Data freshness indicators
    data_age = get_data_age_minutes()

    if data_age <= 5:
        freshness_indicator = "🟢 LIVE"
        freshness_color = "green"
    elif data_age <= 15:
        freshness_indicator = "🟡 RECENT"
        freshness_color = "orange"
    else:
        freshness_indicator = "🔴 DELAYED"
        freshness_color = "red"

    st.markdown(f"""
    <div style="text-align: right; color: {freshness_color}; font-weight: bold;">
        Data Status: {freshness_indicator} ({data_age} min ago)
    </div>
    """, unsafe_allow_html=True)

    # Adaptive display based on data availability
    if data_age <= 15:
        # Show real-time focused view
        display_realtime_dashboard()
    else:
        # Fall back to historical analysis
        st.warning("⚠️ Real-time data delayed. Showing latest available historical
data.")
        display_historical_dashboard()

    # Always provide option to view historical trends
    if st.checkbox("📊 Show Historical Context"):
        historical_comparison_chart = create_historical_comparison()
        st.plotly_chart(historical_comparison_chart, use_container_width=True)
```

7. Implementation Roadmap

Phase 1: Core Infrastructure (1.5 hours)

```
# Hour 1-1.5: Basic setup and data connections

# 1. Project setup (15 min)
"""
mkdir grid_decarbonization_dashboard
cd grid_decarbonization_dashboard
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install streamlit plotly pandas gridstatus python-dotenv requests
"""

# 2. Environment configuration (15 min)
"""
Create .env file:
EIA_API_KEY=your_api_key_here
"""

# 3. Basic Streamlit app structure (30 min)
"""
app.py basic structure:
- Import statements
- Page configuration
- Sidebar setup
- Main content area
- Basic data fetching function
"""

# 4. Data connection testing (30 min)
"""
Test connections to:
- GridStatus library for at least one ISO
- EIA API for basic data retrieval
- Error handling for API failures
"""
```

Phase 2: Data Processing & KPI Calculation (1 hour)

```
# Hour 1.5-2.5: Core data processing

# 1. Data fetching and caching (30 min)
def setup_data_pipeline():
    # Implement caching decorators
    # Create data fetching functions for each source
    # Handle API rate limits and errors
    pass

# 2. KPI calculation functions (30 min)
def implement_kpi_calculations():
    # Renewable penetration calculation
    # Capacity factor calculations
    # Carbon intensity calculations
    # Curtailment analysis
    pass
```

Phase 3: Visualization Development (1.5 hours)

```
# Hour 2.5-4: Chart creation and layout

# 1. Primary visualizations (45 min)
"""
- Renewable penetration gauge
- Generation mix stacked area chart
- Regional map visualization
- Time series trends
"""

# 2. Secondary charts (30 min)
"""
- Capacity factor line charts
- Curtailment analysis
- Price correlation plots
"""

# 3. Layout and responsive design (15 min)
"""
- Column layouts
- Tab organization
- Mobile responsiveness
"""
```

Phase 4: Interactivity & Polish (1 hour)

```
# Hour 4-5: User interactions and final polish

# 1. Control panel implementation (30 min)
"""
- Filter controls
- Time range selection
- ISO region selection
- Fuel type filtering
"""

# 2. Drill-down functionality (20 min)
"""
- Chart click events
- Detailed view displays
- Data export capabilities
"""

# 3. Final polish and testing (10 min)
"""
- CSS styling
- Error handling
- Performance optimization
- Documentation
"""
```

Phase 5: Enhancement & Documentation (1 hour)

```
# Hour 5-6: Advanced features and documentation

# 1. Advanced analytics (30 min)
"""
- Forecasting capabilities
- Correlation analysis
- Anomaly detection
"""

# 2. Performance optimization (15 min)
"""
- Caching optimization
- Data compression
- Loading indicators
"""

# 3. Documentation and deployment prep (15 min)
"""
- README.md creation
- Configuration documentation
- Deployment instructions
"""
```

Deployment Checklist

```
deployment_steps = [
    "✅ Environment variables configured",
    "✅ API keys secured and tested",
    "✅ Dependencies listed in requirements.txt",
    "✅ Error handling for all API calls",
    "✅ Responsive design tested",
    "✅ Performance optimized with caching",
    "✅ Documentation complete",
    "✅ Ready for Streamlit Cloud deployment"
]
```

8. Best Practices

Energy Dashboard Design Patterns

Grid Operator UI Conventions

```
# Industry-standard design patterns for grid operations

grid_operator_conventions = {
    'color_coding': {
        'normal_operation': '#28a745',      # Green
        'warning_levels': '#ffc107',        # Yellow/Orange
        'critical_alerts': '#dc3545',       # Red
        'offline_status': '#6c757d'         # Gray
    },

    'status_indicators': {
        'real_time': 'Pulsing dot animation',
        'delayed': 'Static colored indicator',
        'offline': 'Grayed out with strikethrough',
        'maintenance': 'Orange with wrench icon'
    },

    'alert_hierarchy': {
        'emergency': 'Full screen alert with sound',
        'high_priority': 'Top banner notification',
        'medium_priority': 'Side panel notification',
        'low_priority': 'Status bar indicator'
    }
}

def implement_grid_operator_patterns():
    """Apply industry-standard grid operator UI patterns"""

    # Status indicators for each ISO
    for iso in iso_list:
        status = get_iso_status(iso)

        if status == 'online':
            st.markdown(f"🟢 **{iso}** - ONLINE")
        elif status == 'delayed':
            st.markdown(f"🟡 **{iso}** - DELAYED")
        elif status == 'offline':
            st.markdown(f"🔴 **{iso}** - OFFLINE")

    # Critical alerts system
    critical_alerts = get_critical_alerts()
    if critical_alerts:
        for alert in critical_alerts:
            st.error(f"🚨 CRITICAL: {alert['message']}")

    # Warning system
    warnings = get_system_warnings()
    if warnings:
        for warning in warnings:
            st.warning(f"⚠️ WARNING: {warning['message']}")
```

Performance Optimization Patterns

```python
def implement_performance_best_practices():
    """Key performance optimizations for energy dashboards"""

    # 1. Intelligent caching strategy
    @st.cache_data(ttl=300, max_entries=100)
    def cached_iso_data(iso, start_time, end_time):
        # Cache expensive API calls
        return fetch_iso_data(iso, start_time, end_time)

    # 2. Progressive data loading
    def progressive_data_loading():
        # Load critical data first
        with st.spinner("Loading critical metrics..."):
            critical_data = load_critical_kpis()
            display_critical_metrics(critical_data)

        # Load secondary data asynchronously
        with st.spinner("Loading regional analysis..."):
            regional_data = load_regional_data()
            display_regional_analysis(regional_data)

    # 3. Data compression for large datasets
    def optimize_data_transfer():
        # Downsample time series for display
        if len(time_series_data) > 1000:
            time_series_data = time_series_data.resample('15T').mean()

        # Round numerical values to reduce precision
        numerical_columns = time_series_data.select_dtypes(include=[np.number])
        time_series_data[numerical_columns.columns] = numerical_columns.round(2)

        return time_series_data

    # 4. Efficient chart rendering
    def optimize_chart_performance():
        # Limit data points in charts
        max_points = 500
        if len(chart_data) > max_points:
            chart_data = chart_data.iloc[::len(chart_data)//max_points]

        # Use appropriate chart types for data size
        if len(chart_data) > 100:
            return px.line(chart_data)  # Line charts for large datasets
        else:
            return px.scatter(chart_data)  # Scatter for smaller datasets
```

Data Quality and Reliability

```python
def implement_data_quality_patterns():
    """Ensure data reliability and quality in energy dashboards"""

    # 1. Data validation pipeline
    def validate_energy_data(data):
        validation_results = {
            'completeness': check_data_completeness(data),
            'consistency': check_data_consistency(data),
            'timeliness': check_data_timeliness(data),
            'accuracy': check_data_accuracy(data)
        }

        return validation_results

    # 2. Anomaly detection
    def detect_data_anomalies(data):
        # Statistical outlier detection
        Q1 = data.quantile(0.25)
        Q3 = data.quantile(0.75)
        IQR = Q3 - Q1

        outliers = data[(data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))]

        if len(outliers) > 0:
            st.warning(f"⚠️ {len(outliers)} potential data anomalies detected")

            with st.expander("View Anomalies"):
                st.dataframe(outliers)

        return outliers

    # 3. Data source reliability indicators
    def display_data_source_health():
        st.sidebar.subheader("📊 Data Source Health")

        data_sources = ['GridStatus', 'EIA API', 'CAISO', 'ERCOT', 'PJM']

        for source in data_sources:
            health_score = get_source_health_score(source)

            if health_score >= 95:
                st.sidebar.success(f"✅ {source}: {health_score}%")
            elif health_score >= 80:
                st.sidebar.warning(f"⚠️ {source}: {health_score}%")
            else:
                st.sidebar.error(f"❌ {source}: {health_score}%")
```

Accessibility and Usability

```
def implement_accessibility_best_practices():
    """Ensure dashboard accessibility for all users"""

    # 1. Color accessibility
    def ensure_color_accessibility():
        # Use colorblind-friendly palettes
        colorblind_safe_palette = {
            'primary': '#1f77b4',    # Blue
            'secondary': '#ff7f0e',  # Orange
            'success': '#2ca02c',    # Green
            'warning': '#d62728',    # Red
            'info': '#9467bd'        # Purple
        }

        # Provide non-color indicators
        st.markdown("""
        <style>
        .status-online::before { content: "✓ "; }
        .status-warning::before { content: "⚠ "; }
        .status-error::before { content: "✗ "; }
        </style>
        """, unsafe_allow_html=True)

    # 2. Keyboard navigation support
    def add_keyboard_navigation():
        # Add tab indices and focus indicators
        st.markdown("""
        <style>
        .stButton > button:focus {
            border: 3px solid #4CAF50;
            outline: none;
        }

        .stSelectbox > div > div > div:focus {
            border: 2px solid #4CAF50;
        }
        </style>
        """, unsafe_allow_html=True)

    # 3. Screen reader support
    def add_screen_reader_support():
        # Add ARIA labels and descriptions
        st.markdown("""
        <div aria-label="Renewable energy penetration dashboard" role="main">
            <h1 aria-level="1">Grid Decarbonization Dashboard</h1>
        </div>
        """, unsafe_allow_html=True)

    # 4. Alternative text for visualizations
    def add_chart_descriptions():
        # Provide text descriptions of charts
        st.markdown("""
        **Chart Description**: This stacked area chart shows the proportion
        of renewable vs. conventional energy generation over time. The green
        area represents renewable sources (solar, wind, hydro) while the
        red area shows fossil fuel generation.
        """)
```

Security and Configuration Management

```python
def implement_security_best_practices():
    """Security best practices for energy dashboard deployment"""

    # 1. API key management
    def secure_api_key_handling():
        # Use Streamlit secrets management
        try:
            api_key = st.secrets["EIA_API_KEY"]
        except KeyError:
            st.error("❌ EIA API key not configured. Please add to Streamlit
secrets.")
            st.stop()

        # Never log or display API keys
        logger.info("API connection established")  # Don't log the key

        return api_key

    # 2. Input validation
    def validate_user_inputs():
        # Validate date ranges
        if start_date > end_date:
            st.error("❌ Start date must be before end date")
            return False

        # Validate ISO selections
        valid_isos = ['CAISO', 'ERCOT', 'PJM', 'NYISO', 'MISO', 'ISONE', 'SPP']
        if not all(iso in valid_isos for iso in selected_isos):
            st.error("❌ Invalid ISO selection")
            return False

        return True

    # 3. Error handling without exposing internals
    def safe_error_handling():
        try:
            data = fetch_sensitive_data()
        except APIException as e:
            # Log detailed error internally
            logger.error(f"API Error: {e}")

            # Show user-friendly message
            st.error("⚠️ Unable to fetch data. Please try again later.")

            # Provide fallback
            return get_cached_data()
```

Summary

This comprehensive technical breakdown provides a complete implementation guide for the Real-Time Grid Decarbonization Dashboard. The 4-6 hour development timeline is realistic given the modular approach and extensive use of pre-built components. Key success factors include:

1. **Robust data pipeline** using GridStatus and EIA APIs with proper caching
2. **Intuitive UI/UX** following energy industry conventions and accessibility standards
3. **Interactive visualizations** with drill-down capabilities and real-time updates

4. **Performance optimization** through intelligent caching and progressive loading
5. **Production-ready practices** including security, error handling, and documentation

The dashboard will serve as an effective tool for tracking grid decarbonization progress, enabling stakeholders to make data-driven decisions about renewable energy integration and grid operations.

Sources: GridStatus Documentation, EIA API Documentation, Energy Dashboard UX Research, Streamlit Component Libraries, Professional Energy Dashboard Analysis