Technical Specification

NASA Global Temperature Analysis Platform

1. System Architecture

1.1 Overview

The system follows a modern JAMstack architecture with static generation, serverless API routes, and client-side hydration. Data processing occurs offline in Python, outputs are stored as static JSON, and the Next.js frontend consumes this data for interactive visualization.

1.2 Architecture Diagram (Components)

Data Layer:

- NASA GISTEMP source data (CSV files)
- Python ETL pipeline (pandas, numpy, scipy)
- Processed JSON outputs stored in /public/data

Application Layer:

- Next.js 14 with App Router
- React Server Components for initial render
- Client components for interactive visualizations
- API routes for data endpoints

Presentation Layer:

- React components with TypeScript
- Tailwind CSS for styling
- Plotly.js for charts
- Leaflet for maps

Deployment Layer:

- Vercel Edge Network (CDN)
- GitHub for version control and CI/CD trigger
- Automatic deployments on push to main

2. Project Structure

Directory Layout:

```
nasa-temperature-analysis/ ├── app/                        # Next.js App
Router │   ├── layout.tsx              # Root layout │   ├── page.tsx
# Homepage │   ├── global-trends/ │   │   └── page.tsx │   ├── regional-
analysis/ │   │   └── page.tsx │   ├── geographic-view/ │   │   └── page.tsx
│   ├── about/ │   │   └── page.tsx │   └── api/ │       ├── data/ │       │
└── route.ts        # Data API endpoint │       └── stats/ │           └──
route.ts        # Stats API endpoint ├── components/                # React
components │   ├── charts/ │   │   ├── TimelineChart.tsx │   │   ├──
RegionalChart.tsx │   │   ├── HeatMap.tsx │   │   └── WarmingStripes.tsx │
├── ui/ │   │   ├── Header.tsx │   │   ├── Footer.tsx │   │   ├──
StatsPanel.tsx │   │   └── LoadingState.tsx │   └── layout/ │       └──
Navigation.tsx ├── lib/                        # Utility functions │   ├──
data-loader.ts │   ├── stats.ts │   └── formatters.ts ├── types/
# TypeScript definitions │   └── temperature.ts ├── public/
# Static assets │   ├── data/                   # Processed JSON data │   │
├── global-annual.json │   │   ├── regional.json │   │   ├── geographic.json
│   │   └── metadata.json │   └── images/ ├── data-pipeline/            #
Python ETL scripts │   ├── download_data.py │   ├── process_data.py │   ├──
calculate_stats.py │   ├── generate_outputs.py │   └── requirements.txt ├──
scripts/ │   └── update-data.sh        # Automation script ├── .github/ │
└── workflows/ │       └── deploy.yml        # CI/CD config ├──
package.json ├── tsconfig.json ├── tailwind.config.js ├── next.config.js └──
README.md
```

3. Technology Stack (Detailed)

3.1 Frontend Dependencies

```
{   "dependencies": {     "next": "^14.2.0",     "react": "^18.3.0",
"react-dom": "^18.3.0",     "typescript": "^5.4.0",     "plotly.js":
"^2.30.0",     "react-plotly.js": "^2.6.0",     "leaflet": "^1.9.4",
"react-leaflet": "^4.2.1",     "date-fns": "^3.3.0",     "clsx": "^2.1.0",
"tailwind-merge": "^2.2.0"   },   "devDependencies": {     "@types/node":
"^20.11.0",     "@types/react": "^18.2.0",     "@types/leaflet": "^1.9.8",
"tailwindcss": "^3.4.0",     "postcss": "^8.4.0",     "autoprefixer":
"^10.4.0",     "eslint": "^8.57.0",     "eslint-config-next": "^14.2.0"   } }
```

3.2 Python Dependencies

```
# requirements.txt pandas==2.2.0 numpy==1.26.3 scipy==1.12.0 requests==2.31.0
python-dateutil==2.8.2
```

4. Data Pipeline Implementation

4.1 Download Script (download_data.py)

```
import requests import os from pathlib import Path  # URLs for NASA GISTEMP
data URLS = {     'global':
'https://data.giss.nasa.gov/gistemp/tabledata_v4/GLB.Ts+dSST.csv',
'hemispheric':
'https://data.giss.nasa.gov/gistemp/tabledata_v4/NH.Ts+dSST.csv',
'zonal': 'https://data.giss.nasa.gov/gistemp/tabledata_v4/ZonAnn.Ts+dSST.csv'
}  def download_data():     """Download NASA GISTEMP datasets"""     data_dir
= Path('data/raw')     data_dir.mkdir(parents=True, exist_ok=True)
for name, url in URLS.items():         print(f"Downloading {name} data...")
response = requests.get(url)         response.raise_for_status()
filepath = data_dir / f"{name}.csv"         with open(filepath, 'wb') as f:
f.write(response.content)         print(f"Saved to {filepath}")  if __name__
== '__main__':     download_data()
```

4.2 Data Processing Script (process_data.py)

```
import pandas as pd import numpy as np from scipy import stats from pathlib
import Path import json  def load_and_clean_data(filepath):     """Load and
clean NASA data"""     # NASA format: Year, Jan, Feb, ..., Dec, J-D, D-N,
DJF, MAM, JJA, SON     df = pd.read_csv(filepath, skiprows=1)         #
Handle missing values (NASA uses *** for missing)     df = df.replace('***',
np.nan)          # Convert to numeric     numeric_cols = df.columns[1:]
df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors='coerce')
return df  def calculate_trends(df, column='J-D'):     """Calculate linear
trend and statistics"""     years = df['Year'].values     temps =
df[column].values          # Remove NaN values     mask = ~np.isnan(temps)
```

```
years_clean = years[mask]     temps_clean = temps[mask]          # Linear
regression     slope, intercept, r_value, p_value, std_err =
stats.linregress(         years_clean, temps_clean     )          # Calculate
trend line     trend_line = slope * years + intercept          return {
'slope': slope,         'intercept': intercept,         'r_squared': r_value
** 2,         'p_value': p_value,         'warming_rate_per_decade': slope *
10,         'trend_line': trend_line.tolist()     }  def
generate_output_data():     """Generate processed JSON files for frontend"""
# Load data     global_df = load_and_clean_data('data/raw/global.csv')
# Calculate overall trend     trend_stats = calculate_trends(global_df)
# Prepare annual data     annual_data = {         'years':
global_df['Year'].tolist(),         'temperatures': global_df['J-
D'].tolist(),         'trend': trend_stats['trend_line'],
'statistics': {             'warming_rate':
round(trend_stats['warming_rate_per_decade'], 3),             'r_squared':
round(trend_stats['r_squared'], 4),             'current_anomaly':
round(global_df['J-D'].iloc[-1], 2)         }     }          # Find warmest
years     warmest = global_df.nlargest(10, 'J-D')[['Year', 'J-
D']].to_dict('records')     annual_data['warmest_years'] = warmest          #
Calculate decadal averages     global_df['decade'] = (global_df['Year'] //
10) * 10     decadal = global_df.groupby('decade')['J-
D'].mean().reset_index()     annual_data['decadal_averages'] =
decadal.to_dict('records')          # Save to public directory     output_dir
= Path('../public/data')     output_dir.mkdir(parents=True, exist_ok=True)
with open(output_dir / 'global-annual.json', 'w') as f:
json.dump(annual_data, f, indent=2)          print("Processed data saved
successfully!")  if __name__ == '__main__':     generate_output_data()
```

5. Frontend Implementation

5.1 Data Types (types/temperature.ts)

```
export interface TemperatureData {   years: number[];   temperatures:
number[];   trend: number[];   statistics: {     warming_rate: number;
r_squared: number;     current_anomaly: number;   };   warmest_years: Array<{
Year: number;     'J-D': number;   }>;   decadal_averages:
Array<{     decade: number;     'J-D': number;   }>; }  export interface
ChartConfig {   title: string;   xAxisLabel: string;   yAxisLabel: string;
showLegend: boolean;   height: number; }
```

5.2 Timeline Chart Component

```
'use client';  import React from 'react'; import dynamic from 'next/dynamic';
import { TemperatureData } from '@/types/temperature';  const Plot =
dynamic(() => import('react-plotly.js'), { ssr: false });  interface
TimelineChartProps {   data: TemperatureData; }  export default function
TimelineChart({ data }: TimelineChartProps) {   const traces =
[     {       x: data.years,       y: data.temperatures,       type:
'scatter',       mode: 'lines',       name: 'Annual Mean',       line:
{ color: '#EF4444', width: 2 },       hovertemplate: '<b>%{x}</b><br>Anomaly:
%{y:.2f}°C<extra></extra>'     },     {       x: data.years,       y:
data.trend,       type: 'scatter',       mode: 'lines',       name: 'Trend
Line',       line: { color: '#1F4788', width: 3, dash: 'dash' },
hoverinfo: 'skip'     }   ];    const layout = {     title: 'Global
Temperature Anomaly (1880-Present)',     xaxis: {       title: 'Year',
gridcolor: '#E5E7EB'     },     yaxis: {       title: 'Temperature Anomaly
(°C)',       gridcolor: '#E5E7EB',       zeroline: true,       zerolinecolor:
'#9CA3AF'     },     hovermode: 'closest',     plot_bgcolor: '#F9FAFB',
paper_bgcolor: 'white',     font: { family: 'Inter, sans-serif' },
margin: { t: 50, r: 50, b: 50, l: 60 }   };    const config =
{     responsive: true,     displayModeBar: true,     displaylogo: false,
modeBarButtonsToRemove: ['lasso2d', 'select2d']   };    return (     <div
className="w-full">       <Plot         data={traces}         layout={layout}
config={config}         style={{ width: '100%', height: '500px' }}       />
</div>   ); }
```

5.3 Stats Panel Component

```
import { TemperatureData } from '@/types/temperature';  interface
StatsPanelProps {   data: TemperatureData; }  export default function
StatsPanel({ data }: StatsPanelProps) {   return (     <div className="grid
grid-cols-1 md:grid-cols-3 gap-6">       <StatCard         title="Current
Anomaly"         value={`${data.statistics.current_anomaly > 0 ? '+' : ''}$
{data.statistics.current_anomaly}°C`}         description="Compared to 1951-
1980 baseline"       />       <StatCard         title="Warming Rate"
value={`+${data.statistics.warming_rate}°C`}         description="Per decade
(recent trend)"       />       <StatCard         title="Hottest Year"
value={data.warmest_years[0].Year.toString()}         description={`$
{data.warmest_years[0]['J-D']}°C anomaly`}       />     </div>   ); }
```

```
function StatCard({ title, value, description }: {   title: string;   value:
string;   description: string; }) {   return (     <div className="bg-white
rounded-lg shadow-md p-6 border border-gray-200">       <h3 className="text-
sm font-medium text-gray-500 uppercase tracking-wide">         {title}
</h3>       <p className="mt-2 text-3xl font-bold text-gray-900">{value}</p>
<p className="mt-1 text-sm text-gray-600">{description}</p>     </div>   ); }
```

6. API Implementation

6.1 Data Endpoint (app/api/data/route.ts)

```
import { NextResponse } from 'next/server'; import { promises as fs } from
'fs'; import path from 'path';  export async function GET(request: Request) {
try {     const { searchParams } = new URL(request.url);     const dataset =
searchParams.get('dataset') || 'global-annual';          // Read data file
const filePath = path.join(process.cwd(), 'public', 'data', `$
{dataset}.json`);     const fileContents = await fs.readFile(filePath,
'utf8');     const data = JSON.parse(fileContents);          return
NextResponse.json(data);   } catch (error) {     return NextResponse.json(
{ error: 'Failed to load data' },       { status: 500 }     );   } }  export
const dynamic = 'force-static'; export const revalidate = 86400; //
Revalidate once per day
```

7. Configuration Files

7.1 Next.js Config (next.config.js)

```
/** @type {import('next').NextConfig} */ const nextConfig =
{   reactStrictMode: true,   images: {     domains: [],   },   webpack:
(config) => {     // Handle plotly.js     config.resolve.alias =
{       ...config.resolve.alias,       'plotly.js':
'plotly.js/dist/plotly.js',     };     return config;   }, };  module.exports
= nextConfig;
```

7.2 Tailwind Config (tailwind.config.js)

```
/** @type {import('tailwindcss').Config} */ module.exports = {   content: [
'./app/**/*.{js,ts,jsx,tsx,mdx}',     './components/**/*.
{js,ts,jsx,tsx,mdx}',   ],   theme: {     extend: {       colors:
{         primary: '#1F4788',         secondary: '#2E75B6',       },
fontFamily: {         sans: ['Inter', 'system-ui', 'sans-serif'],       },
},   },   plugins: [], };
```

7.3 TypeScript Config (tsconfig.json)

```
{   "compilerOptions": {     "target": "ES2020",     "lib": ["dom",
"dom.iterable", "esnext"],     "allowJs": true,     "skipLibCheck": true,
"strict": true,     "forceConsistentCasingInFileNames": true,     "noEmit":
true,     "esModuleInterop": true,     "module": "esnext",
"moduleResolution": "bundler",     "resolveJsonModule": true,
"isolatedModules": true,     "jsx": "preserve",     "incremental": true,
"plugins": [{ "name": "next" }],     "paths": {       "@/*":
["./*"]     }   },   "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx",
".next/types/**/*.ts"],   "exclude": ["node_modules"] }
```

8. Deployment & CI/CD

8.1 Vercel Configuration

Setup Steps:

1. Connect GitHub repository to Vercel
2. Configure build settings: Framework Preset = Next.js
3. Set build command: npm run build
4. Set output directory: .next
5. Configure custom domain in Vercel dashboard
6. Enable automatic deployments on push to main branch

8.2 GitHub Actions Workflow

```
name: Deploy to Vercel  on:   push:     branches: [main]   pull_request:
branches: [main]  jobs:   deploy:     runs-on: ubuntu-latest          steps:
- uses: actions/checkout@v3              - name: Setup Node.js         uses:
actions/setup-node@v3         with:           node-version: '18'
cache: 'npm'              - name: Install dependencies         run: npm ci
- name: Run linter         run: npm run lint              - name: Build
project         run: npm run build              - name: Deploy to Vercel
uses: amondnet/vercel-action@v25         with:           vercel-token: $
{{ secrets.VERCEL_TOKEN }}           vercel-org-id: ${{ secrets.VERCEL_ORG_ID
}}           vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
vercel-args: '--prod'
```

9. Git Setup & Best Practices

9.1 Initial Setup

```
# Initialize repository git init git add . git commit -m "Initial commit:
Project structure and setup"  # Create GitHub repository (via GitHub CLI or
web interface) gh repo create nasa-temperature-analysis --public --source=.
--remote=origin  # Push to GitHub git branch -M main git push -u origin main
```

9.2 Branch Strategy

- **main:** Production-ready code, automatically deploys to Vercel
- **develop:** Integration branch for features
- **feature/*:** Individual feature branches (e.g., feature/timeline-chart)
- **fix/*:** Bug fix branches

9.3 Commit Convention

```
# Format: <type>(<scope>): <subject>  feat(charts): Add timeline
visualization component fix(api): Resolve data loading error for regional
endpoint docs(readme): Update installation instructions style(ui): Improve
responsive layout for mobile refactor(data): Optimize JSON structure for
performance test(stats): Add unit tests for trend calculations chore(deps):
Update Next.js to v14.2.0
```

9.4 .gitignore

```
# Dependencies node_modules/ .pnp .pnp.js  # Testing coverage/  #
Next.js .next/ out/ build/ dist/  # Production *.log npm-debug.log* yarn-
debug.log* yarn-error.log*  # Environment .env .env.local .env.production  #
IDE .vscode/ .idea/ *.swp *.swo *~  # OS .DS_Store Thumbs.db  # Python
__pycache__/ *.py[cod] .venv/ venv/  # Data (raw downloads) data/raw/*.csv
```

10. Performance Optimization

10.1 Data Optimization Strategies

- Pre-aggregate data during Python processing (annual, decadal)
- Use gzip compression for JSON files
- Implement lazy loading for large datasets
- Cache API responses with appropriate TTL

10.2 Frontend Optimization

- Use dynamic imports for heavy visualization libraries
- Implement React.memo for expensive chart components
- Use Next.js Image component for optimized image delivery
- Enable static generation where possible
- Minimize bundle size with tree shaking

11. Testing Strategy

11.1 Data Validation Tests

```
# test_data_processing.py import pytest import pandas as pd from
data_pipeline.process_data import calculate_trends, load_and_clean_data  def
test_data_loading():     df = load_and_clean_data('data/raw/global.csv')
assert not df.empty     assert 'Year' in df.columns     assert
df['Year'].min() >= 1880  def test_trend_calculation():     # Create mock
data     df = pd.DataFrame({         'Year': range(1880, 2024),         'J-
D': [i * 0.01 for i in range(144)]  # Linear increase     })         trends
= calculate_trends(df)     assert trends['slope'] > 0  # Should show warming
assert 0 <= trends['r_squared'] <= 1     assert trends['p_value'] < 0.05  #
Statistically significant
```

11.2 Manual Testing Checklist

- Verify all charts render correctly
- Test hover interactions and tooltips
- Confirm responsive behavior on mobile (320px-375px)
- Test on tablet (768px-1024px)
- Validate data export functionality
- Check API endpoints return correct data
- Verify navigation between pages
- Test loading states and error handling

12. Documentation Requirements

12.1 README.md Structure

- Project title and description
- Live demo link
- Features list with screenshots
- Technology stack
- Installation instructions
- Usage examples
- Data source attribution
- License information

13. Maintenance & Updates

13.1 Monthly Data Update Script

```
#!/bin/bash # scripts/update-data.sh  echo "Updating NASA temperature
data..."  # Navigate to data pipeline cd data-pipeline  # Activate Python
environment (if using venv) source venv/bin/activate  # Run data update
pipeline python download_data.py python process_data.py python
generate_outputs.py  echo "Data updated successfully!" echo "Commit and push
changes to trigger deployment."  # Optional: Auto-commit # git add
../public/data/ # git commit -m "data: Update temperature data ($(date +%Y-
%m))" # git push origin main
```

14. Quick Start Commands

```
# Clone repository git clone https://github.com/YOUR_USERNAME/nasa-
temperature-analysis.git cd nasa-temperature-analysis  # Install dependencies
npm install  # Process data (first time setup) cd data-pipeline pip install -
r requirements.txt python download_data.py python process_data.py cd ..  #
Run development server npm run dev # Open http://localhost:3000  # Build for
production npm run build npm start  # Deploy to Vercel (after connecting
repo) vercel --prod
```

End of Technical Specification