Capstone Project Plan: Active Highway Projects Viewer

Abstract

The **Active Highway Projects Viewer** is an open-source, public-facing interactive dashboard that enables users to explore current and awarded highway construction projects in Kentucky—**independently of the Esri software stack**. Built with modern JavaScript frameworks and leveraging open data APIs and spatial visualizations, the application empowers the public, contractors, and government stakeholders with intuitive access to authoritative project data.

At its core, this tool visualizes the **Approved Six-Year Highway Plan (6YP)**—a legislatively approved framework updated every two years by the Kentucky General Assembly. The 6YP outlines major highway improvement projects across the state and is developed through collaborative input from KYTC, Area Development Districts (ADDs), Metropolitan Planning Organizations (MPOs), and Highway District Offices. The Viewer displays currently active and awarded projects derived from this plan, with data reflecting the availability of both state and federal highway funds.

Designed for accessibility and responsiveness, the dashboard supports interactive filtering, dynamic charting, map-based exploration, and integration with state-maintained APIs, making it a valuable tool for planning, transparency, and public engagement in infrastructure development.

Problem Statement

The Kentucky Transportation Cabinet (KYTC) offers GIS-based access to construction project data using Esri out-of-the-box tools. However, this approach limits public access due to licensing, performance issues, and limited interoperability. Users looking to explore highway projects often find the existing interfaces unintuitive and inflexible.

This project aims to create an open-source, device-responsive, non-ESRI dashboard that presents current transportation project data in a visually informative and user-friendly manner. It also aims to develop modular functions for integration into similar dashboards.

Goals and Objectives

- Create an interactive dashboard using open web technologies to explore active Kentucky Transportation Cabinet (KYTC) highway projects.
- Replace the reliance on proprietary mapping tools with Leaflet.js and open APIs.
- Integrate public API endpoints, such as the KYTC Spatial API, to retrieve route-specific data.
- Use SQLite to store and query project data for portability and offline access.
- Implement filters for district, county, and project type.
- Ensure the user interface is modern and responsive, optimized for both mobile and desktop use

---

Features

- Integrate a third-party API into your project (MANDATORY).
    - Integration with the **KYTC Spatial API** to return route-specific project metadata.
- Choose at least three items from the FEATURES LIST
    - Select two from the first set of features
        - Analyze data that is stored in arrays, objects, sets, or maps, and display information about it in your app.
        - Visualize data in a user-friendly way. (e.g., graph, chart, etc.)
          This can include using libraries like ChartJS
    - and one from the second set
        - Implement modern interactive UI features (e.g., table/data sorting, autocomplete, drag-and-drop, calendar-date-picker, etc.)
    - Additional - Develop a project using a JavaScript framework. I choose to implement Svelte.

Core Features

- Map integration using **Leaflet.js** to display the locations of awarded and current projects.
- Filters by **Project Type**, **District**, and **County** with searchable dropdowns.
- Interactive summary cards (Active Projects, Total Projects, Total Length filtered by map extent).
- Use of **fetch()** to get project data from a public source and store it in SQLite or JSON files.
- Data visualization via **Plotly.js** for bar and chart summaries.
- Responsive UI utilizing Flexbox/Grid and media queries.

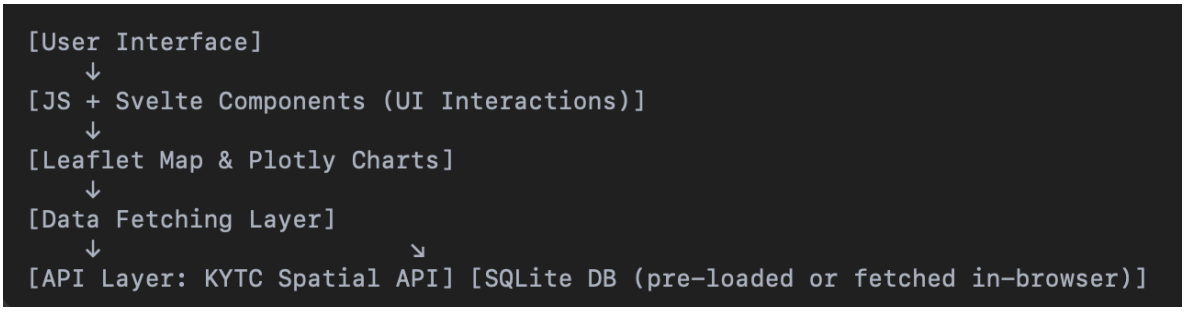
Stretch Goals

- Incorporate **Svelte** for component-based interactivity.
- Add data table with sorting, filtering, and autocomplete features (Tabulator.js or similar).
- Allow users to bookmark or download project reports.
- Add persistent state or caching of API results in browser storage.
- Add user feedback modal and email export.

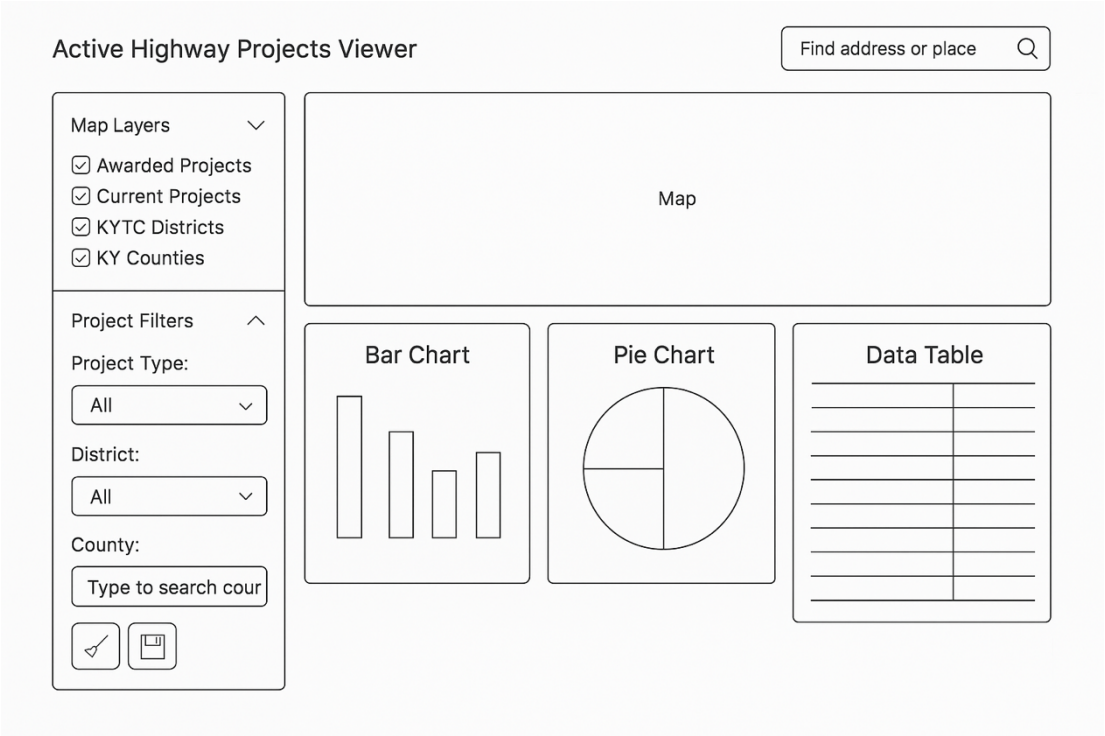
Technologies

- **Frontend**: HTML5, CSS3 (with Bootstrap 5), JavaScript ES6+, Svelte (optional)
- **Mapping**: Leaflet.js, Esri Leaflet plugin (for compatibility), GeoJSON
- **Visualization**: Plotly.js
- **Database**: SQLite (accessed via JavaScript via WASM or pre-processed Python)
- **Data/API**: Fetch + KYTC Spatial API, KYTC Open Data CSVs/GeoJSONs
- **Other Tools**: ChatGPT, Grammarly, Git, GitHub, VS Code, GitHub Copilot, Python (for pre-processing data into SQLite), PanZoom, Tabulator.js.

Architecture



Mockup (Wireframe of UI)



Timeline & Milestones

Week 1

- Finalize core UI layout, filters, map, and initial Leaflet basemap
- Begin populating map with awarded project lines
- Define schema and load project data to SQLite

Week 2

- Finish SQLite data pipeline using fetch → SQLite
- Add KYTC API integration: route info from click
- Begin chart integration with Plotly

Week 3

- Complete chart integration (bar, pie, length by district)
- Add autocomplete and table sorting (Tabulator or native)

Week 4

- Polish filter interactions (county, district, project type)
- Begin responsive layout polishing
- GitHub commits every major milestone

Week 5

- Initial README, document fetch and SQLite process
- Start testing across devices (Chrome, Safari, mobile)
- Add AI code usage comments if applicable

Week 6

- Add Svelte components (optional stretch goal)
- Implement Save/Bookmark user interaction logic
- Begin documentation of SQLite access and route APIs

Week 7

- Conduct usability testing
- Confirm final feature set and remove non-functional parts
- Polish CSS (spacing, icons, typography, colors)

Week 8

- Final testing: map filters, charts, database queries
- Update README with final instructions

Week 9

- Final push: polish, bug fixes, commit history review
- Submit project

Risks and Mitigation Strategies

Risk	Mitigation
SQLite integration in-browser may be limited	Pre-generate SQLite from raw data using Python and ship with app
API limits or downtime from KYTC Spatial API	Cache minimal data in localStorage or fallback to JSON
Svelte integration may introduce build complexity	Encapsulate it as an optional feature, not core requirement
Timeline slippage	Weekly progress reviews and modular component development
Data alignment or structure changes	Write flexible, schema-aware fetch parser logic

---

Testing and Evaluation Plan

- **Manual Testing** for all UI components on Chrome, Safari, Firefox.
- **Device Responsiveness** tested with dev tools and physical devices.
- **User Feedback** from peers/mentors collected earlier.
- **Performance Metrics**: map load speed, filter responsiveness, data fetch timing.
- **Code Quality**: encapsulated modules, comments for AI-aided code.
- **Functional Checklists** to track working filters, visualizations, API usage.