Akvo Charts

A React component library for ECharts, enabling easy integration and customization of interactive, high-performance charts in React applications.

Introduction

Objective

Many projects involve repeating tasks by copying and pasting chart and map components from other projects. This inefficient practice leads to inconsistencies and increased maintenance efforts.

Our solution is to create a dedicated React component library for both ECharts and Leaflet. This will eliminate redundancy, optimize integration, and ensure a consistent approach to data visualization and mapping in all projects.

Platforms like Power Bi require complex setups with cron-jobs for Excel or detailed SQL integrations. In contrast, ECharts and Leaflet offer smooth connections with back-end APIs. Our library will make data filtering, parameter handling, and styling customization effortless. This ensures that charts and maps match the host site's design and functionality perfectly.

Examples like the Chart Component, Maps Component, and Maps Utilities are commonly copied and pasted across projects. While this method is simple, it lacks long-term maintainability. Our objective is to create a tool that can generate dashboards with minimal configuration or parameters, reducing redundancy and improving maintainability in all projects.

Benefits

- **Consistency:** Ensures uniform charting and mapping components across different projects.
- **Efficiency:** Reduces time spent on repetitive tasks, allowing developers to focus on core functionalities.
- **Seamless Integration:** Simplifies the connection with back-end APIs for dynamic data handling.
- **Customization:** Enables easy styling adjustments to match the host site's aesthetics.
- **Maintenance:** Eases the maintenance and updating process by centralizing chart and map components in one library.

Project Showcase

The ECharts and Leaflet libraries have been successfully implemented in various projects, demonstrating their versatility and effectiveness. These projects include:

- Wash in School:
  - https://si-wins.akvotest.org/
  - https://siwins.akvo.org/
- Income Drive Calculator (IDH): https://idc.akvo.org/
- 2SCALE: https://2scale.tc.akvo.org/
- Wash SDG Portal:
  - https://wai-ethiopia.akvotest.org/
  - https://districtwash-ug.org/
  - https://districtwash-np.org/
  - https://districtwash-bd.org/
- MIS Portal: https://ug-wai-mis.tc.akvo.org/
- UNEP Actions: https://unep.tc.akvo.org/
- IDH Farmfit: https://idh-prod.tc.akvo.org/login
- Kabarole WASH: https://si-hcf-wash.tc.akvo.org/
- Covid Uganda: https://covid-ug5w.tc.akvo.org/
- and Many more

Goals

1. **Develop Reusable Components:** To standardize and simplify the integration process, create a library of reusable React components for ECharts and Leaflet.
2. **Enhance Data Handling:** Make sure the components can handle complex data filtering and parameterization to meet various project requirements.
3. **Improve Customization Options:** Offer extensive styling options so components can easily be customized to match the visual identity of any host site.
4. **Optimize API Integration:** Enable real-time data visualization and mapping by assisting smooth integration with back-end APIs.
5. **Support and Maintenance:** Establish a strong framework for ongoing support and maintenance to keep the library up-to-date with the latest features and security patches.

Low-Level Design

Architecture Overview

We're building a React component library to make charting and mapping easier and more consistent across TC projects. The library will be divided into several key parts:

1. **Component Layer:** Where all the chart and maps components live.
2. **Service Layer:** Handles all the data fetching and API calls and also stores JWT.
3. **Utility Layer:** Contains helper functions to make data transform easier and unified.
4. **Styling Layer:** Ensures everything looks good and stays consistent.

1. Component Layer

This is the heart of Akvo charts library, containing reusable React components that are easy to configure and use.

- **Chart Components:**
  - **BaseChart:** A foundational component for all charts, setting up ECharts with basic settings.
  - **LineChart, BarChart, PieChart:** Specific chart types built on top of BaseChart, each with their unique configurations.
  - Etc...
- **Map Components:**
  - **BaseMap:** The foundational map component, setting up Leaflet with basic settings.
  - **MarkerMap, HeatMap:** Specific map types built on top of BaseMap, adding more features.

Each component will be designed to accept props for data, options, and styling, to making them flexible and reusable.

2. Service Layer

The service layer takes care of all the heavy lifting related to data. This includes fetching data from APIs and processing it into a format Akvo-Charts components can use.

- **API Service:** Manages API calls to fetch the data needed for Akvo-Charts and maps.

- **Data Processing Service:** Transforms raw data into something Akvo-Charts components can work with.
- **Configuration Service:** Manages default settings and configurations for Akvo-Charts components.

3. Utility Layer

This layer is packed with helpful functions that make our components and services work smoothly together.

- **Data Utilities:** Functions for filtering, sorting, and manipulating data.
- **Config Utilities:** Functions to merge user settings with default configurations.
- **Style Utilities:** Functions to ensure consistent styling across components.

4. Styling Layer

The styling layer ensures our components look good and consistent across different projects.

- **Base Styles:** Common styles for all components.
- **Theming:** Allows for custom themes to match the host site's design.
- **Responsive Design:** Ensures components look great on all screen sizes.

Build and Deployment Scripts

The Akvo-Charts project utilizes a set of scripts defined in the package.json file to handle various tasks related to building, testing, and deploying the library.

```
"scripts": {
  "build": "microbundle-crl --no-compress --format modern,cjs --css-modules 'ae-[local]'",
  "start": "microbundle-crl watch --no-compress --format modern,cjs --css-module 'ae-[local]'",
  "prepare": "run-s build",
  "test": "run-s test:unit test:lint test:build ",
  "test:build": "run-s build",
  "test:lint": "eslint --config .eslintrc.json ./src/ --ext .js,.jsx",
  "test:unit": "cross-env CI=1 react-scripts test --env=jsdom",
  "test:watch": "react-scripts test --env=jsdom",
  "predeploy": "cd example && yarn install && yarn run build",
  "deploy": "gh-pages -d example/build"
}
```

Here’s a breakdown of what each script does:

1. **`build`:** This script uses microbundle-crl to bundle the library. It generates two formats (modern and cjs) without compression and applies CSS modules with a naming convention of `ac-[local]`.
2. **`start`:** This script is for development purposes. It watches for changes and rebuilds the library on the fly, using the same settings as the build script.
3. **`prepare`:** This script runs the build script as a part of the package preparation before publishing or deploying.
4. **`test`:** This script runs a series of sub-scripts to execute unit tests, linting, and build tests.
   - **`test:build`:** This script triggers the build process as part of the testing sequence.
   - **`test:lint`:** This script uses ESLint to lint the source files according to the configurations specified in .eslintrc.json.
   - **`test:unit`:** This script runs unit tests in a CI environment using react-scripts with jsdom as the test environment.
   - **`test:watch`:** This script runs unit tests in watch mode for ongoing development.
5. **`predeploy`:** This script prepares the example project for deployment by navigating into the example directory, installing dependencies, and building the example project.
6. **`deploy`:** This script deploys the built example project to GitHub Pages.