Form builder developer's manual (Frontend)

Contents

1. Introduction

Form Builder Services is a dynamic and versatile platform designed to bridge the gap between Creating Forms and Viewing Form Submissions with a modern approach using Vite and React. With a robust infrastructure and cutting-edge technology stack, our platform empowers users to easily create customized forms and view submissions in real-time. The service includes proper validation features to ensure data accuracy and security. Additionally, users can take advantage of interface for filling forms, making it convenient for organizations to gather information efficiently. This service also provides insights into unused forms, helping organizations optimize their form-building process.

2. Getting Started

- Clone the **form_builder_frontend** repo from this link (https://github.com/raghav0011/form_builder_frontend) using GIT clone command.
- After that, install all, the package required to run the project using NPM Install command & run the project using NPM RUN DEV.
- The project will automatically run in the defined port in milliseconds as it is using VITE + REACT.

3. Project Structure

3.1 Directory Structure

The project follows a well-organized directory structure to maintain code readability and scalability. Here is an overview of the primary directories and their purposes:

- **`src/`:** This is where most of the project's source code resides.
  - **`components/`:** Contains reusable React components.
  - **`SharedComponents/`:** Contains all MUI reusable components, which is used across the project.
  - **`hooks/`:** Contains all the API payload passing and calling.
  - **`apis/`:** Contains all the API calling by getting payload through hooks.
  - **`styles/`:** Stores CSS or styling-related files.
  - **`utils/`:** Contains utility functions used throughout the project.
  - **`App.js`:** The main application component.
  - **`constants:`:** Contains all the constant data.
- **`public/`:** Contains static assets like images, fonts, and the index.html file.
- **`.env/`:** Contains the API consumed by the project.
- **`package.json/`:** Contains all the required packages for the project to run.

3.2 Key Files and Folders

Here are some of the most important files and folders in our project:

- **src/App.js:** This is the root component of our application, where we render other components. It is the starting point for our React tree.

- **src/components/:** This folder contains various React components. Each component should have its own directory, including its JavaScript file, CSS file, and any additional assets it requires.
- **src/utils/:** Utility functions and helper files should be stored here. For example, you might find functions for data manipulation, API requests, or date formatting in this directory.
- **src/hooks:** This is also one of the main folder which contains all the actions related to API and project add and view functionality.

4. Architecture

4.1 Overview of the Project Architecture

Our project follows a component-based architecture using React. Here is a high-level overview of how it works:

- **Components:** We divide our UI into reusable components. Components can be further categorized into "presentational" (UI components) and "container" components (handling logic and state management).
- **State Management:** Because our project is small in size task project, we have chosen not to use Redux for state management. Instead, we have used prop drilling because the project is not particularly large on its own; rather, it is more akin to a support functionality project for form building.
- **Styling:** We use CSS modules to scope CSS at the component level and also many inline css, ensuring that styles do not leak between components.

5. Coding Standards

5.1 Best Practices

To ensure code quality and best practices, we recommend the following:

- **Modularization:** Keep components and functions modular to encourage reusability and maintainability.
- **Comments:** Document complex logic and functions with comments to explain their purpose and usage.

- **Error Handling:** Implement proper error handling and validation for API requests and user inputs.

5.3 Naming Conventions

Follow meaningful and consistent naming conventions for variables, functions, and components. Use descriptive names to improve code readability.

- We have used **Pascalcase** for component and file naming convention.
- For function naming, we have used **Camelcase** naming convention.
- Variables name must be relevant to the data type it stores.

6. Dependencies

6.1 List of External Dependencies

Here is a list of the major external dependencies used in the project:

- **Vite with React:** Link to React Documentation
- **Material-UI:** Link to Material-UI Documentation
- **Axios:** Link to Axios Documentation

7. Development Workflow

7.1 Branching Strategy

- `develop`: The develop branch contains the production-ready code.
- Other branches are created according to the type of problem developer is going to fix.
- We use **Pascalcase** for this naming approach.

7.2 Commit Messages

We use descriptive commit messages following the conventional commit format. Commit messages should be relevant to the type of commit the developer is going to make. Example: Add user authentication feature.

8. Deployment

8.1 Deployment Environments

We have multiple deployment environments:

- **Development:** Used for developing new features.
- **Production:** The live environment used by our users.

8.2 Deployment Procedure

Deployment to these environments is automated using continuous integration/continuous deployment (CI/CD) pipelines in Github pages.

9. Contact Information

If you have questions or encounter issues, feel free to reach out to the following team members:

Raghav GC