Journey Builder Documentation

- Kavya Chandrika Vempalli
- kavyachandrikavempalli@gmail.com

Journey Builder is an interactive application built with React, designed for a coding challenge aimed at streamlining the setup and visualization of field mappings throughout a Directed Acyclic Graph (DAG) of forms. At its essence, the application analyzes and displays a set of form nodes, which may have dependencies among them, and enables the user to designate prefill mappings from upstream sources to form fields downstream. This is particularly beneficial in form automation processes where values from previous forms are utilized in later forms. The blueprint JSON, which outlines the graph structure, contains comprehensive details about the schema of each form, as well as their relationships and dependencies. By converting this data into a consistent format, Journey Builder facilitates intuitive interaction and mapping through a streamlined, adaptable user interface

Directory Structure:

journey-builder/

```
├── public/
├── src/
│   ├── components/
│   │   ├── FormList.tsx
│   │   └── PrefillModal.tsx
│   ├── mock/
│   │   └── blueprint.json
│   ├── types/
│   │   └── formTypes.ts
│   ├── utils/
│   │   ├── parseBlueprints.ts
│   │   └── getDependencies.ts
│   ├── App.tsx
└── └── App.css
```

Executing the application locally is simple and attainable for developers familiar with fundamental frontend tools. First, users need to have Node.js and npm set up. By cloning the repository from GitHub and executing the npm install command, the required dependencies are installed. Launching the application with npm start initiates a development server and displays the app in a browser window at localhost:3000. The sample data utilized for generating graphs is located in a local blueprint.json file found in the mock directory. This document imitates the response from a real backend API and includes the nodes, edges, and forms necessary to build the form graph. While running, this file is analyzed and converted into functional data structures via a utility function, guaranteeing that the frontend can accurately and responsively display forms along with their related fields

The architecture of the application is built for extensibility, allowing for straightforward integration of additional data sources. Developers have the ability to enhance functionality by adjusting or improving

the dependency resolution logic present in getDependencies.ts. For instance, enhancing support for action-level or global property-based data sources can be achieved by adding extra structures to the dependency map and integrating them into the modal view. The PrefillModal.tsx component is built to dynamically display available fields and can be modified to incorporate these new sources without interrupting the current process. Developers can also modify formTypes.ts to add more types and maintain the integrity of TypeScript type safety. This modular structure facilitates future expansion and modification, suitable for large-scale enterprise integrations or enhanced prefill mapping situations

There are multiple patterns in Journey Builder's implementation that should be highlighted. The project employs a clear separation of concerns by separating data transformation logic into the utils folder and the UI rendering logic into components. The parseBlueprints.ts utility processes raw DAG data, converting it into an array of Form objects that the React app displays. The App.tsx component acts as the main control center, overseeing global state with useState and reacting to user inputs by conditionally rendering modals and form panels. The user interface utilizes uniform styling methods through CSS variables for theming, promoting design consistency and maintainability. Field-level actions, like activating the prefill modal and removing mappings, are well-defined and managed with prompt visual responses. These methods enhance code clarity while also promoting scalability and maintenance.

To sum up, Journey Builder is a thoughtfully designed, flexible, and easy-to-use application that showcases contemporary frontend development techniques. By clearly separating data parsing, UI composition, and state management, it provides a strong base for developing more sophisticated form automation tools. Developers aiming to enhance its functionality can achieve this with ease, due to its modular structure and well-defined interfaces. The current codebase accommodates everything, from extending the prefill logic and integrating with external APIs to improving the visual representation of the form DAG. The documentation, component structure, and architectural transparency render this a perfect model of a scalable frontend solution for DAG-based form workflows. Upcoming enhancements might involve unit testing, local or remote storage for prefill mappings, and enhanced DAG visualization utilizing graph libraries like D3.js or React Flow.