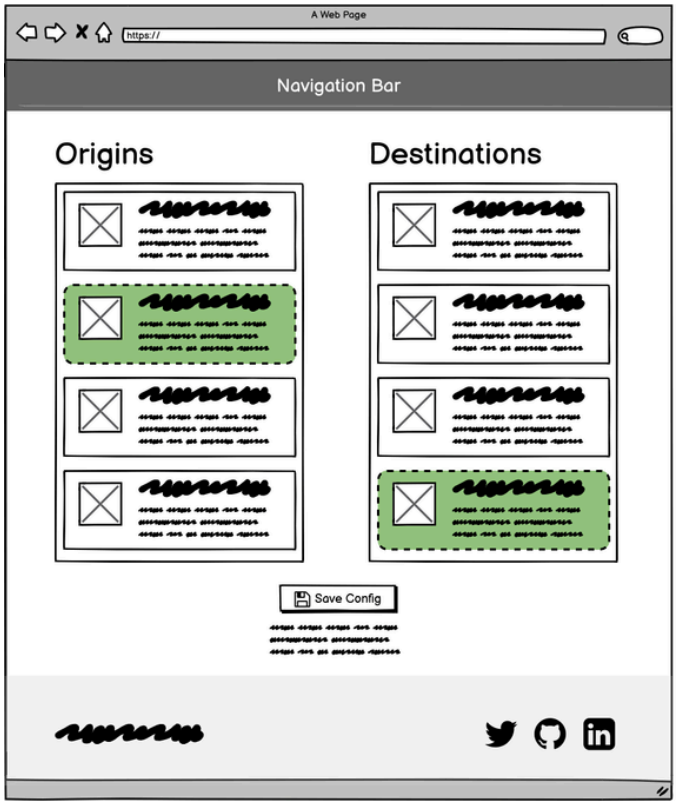
ZenRows Technical Assessment (Fullstack Engineer)

ZenRows is a web scraping API and service designed to help users extract data from websites efficiently and at scale.

For this exercise, let's imagine you're joining our team to work on a new full-stack feature. Your goal is to deliver a maintainable MVP that demonstrates your problem-solving skills across both frontend and backend. We're looking for thoughtful decisions, clean code, and a practical approach - don't worry about perfection, just focus on delivering a working, well-organized solution.

Feature Context:

We want to better understand our customers' needs for configured scraping jobs. To do this, we're giving them a way to select configuration pairs - matching a supported Origin (a website we can scrape, like Zillow or Redfin) with a supported Destination (where the scraped data will be sent, such as AWS S3 or PostgreSQL). Both lists of Origins and Destinations are pre-configured, and you'll find sample data in the provided test-data.json file.



Requirements:

Your app should display both lists to the customer, allowing them to select one Origin and one Destination to form a configuration pair. The UI should be natural and intuitive: users can select or unselect items with a click, but only one item from each list can be selected at a time. When both an Origin and a Destination are selected, the Save button becomes active. Clicking Save should store the selected pair and show a message like "You saved {origin} and {destination}." The saved pair must persist even after a page reload or navigation.

For **extra points**, extend the feature to let customers save multiple different configuration pairs, not just one.

Tech Recommendations:

We suggest you use a dockerized Laravel app for the backend (You can use for example a basic Laravel implementation) and Vue.js for the frontend (to stick to our current tech stack of the website), but you're welcome to explain your reasoning if you choose differently. Please apply the provided design style (Figma link for a high-fidelity design) and seed your database with the test-data.json file. Also, you can use any external resource you need (Google, StackOverflow,

ChatGPT, GitHub Copilot, etc.). Whatever you think will help you to solve the problem, use it!

Focus on delivering a clear, maintainable MVP - avoid over-engineering, and keep your code organized, at the next steps we are going to extend the feature with more requirements, e.g. support of more origins, destinations, creating custom origins, destinations. If you run out of time, just leave comments explaining what you'd do next. Include a Dockerfile so your app can be run easily, and briefly explain your technical decisions and how to run the application in comments or a short README.

Enjoy the exercise, and good luck!