ski - Shadman Rakib, Kevin Cao, Ivan Mijacika
SoftDev pd2
P4 - Recipe Extractor

Description:

Our project is an API that takes in a url to a recipe, extracts all information relating to the recipe, and returns the data in a simple, structured format.

Components

Backend: Node.js /w Typescript

- API endpoint for extraction
- Extraction
    - Schema.org JSON-LD data takes preference. If there is no Schema, move to the next method.
    - Look for microdata tags and extract using Regex. If there is no microdata, move to the next method.
    - Manually extract data
        - Assumption: directions and ingredients come in chunks/links.
        - Detecting DOM nodes that are ingredients. Ingredients may have a known ingredient name or measurement units. Create clusters of them. Nodes between two ingredient nodes are assumed to be ingredients too. This helps with edge cases for uncommon ingredients.
            - Check if any node has the text “Ingredients”. Sometimes ingredient sections are denoted with a section heading. This simplifies the process.
        - Detecting DOM nodes that are directions. Directions may start with step number, ex “1)”, “1.”, and “Step 1”. Directions may also start with verbs, like “cook”.
            - Check if any node has the text “Directions”, “Instructions”, “Method”. Sometimes instruction sections are denoted with a section heading. This simplifies the process.
- Libraries
    - Express - build API
    - Microdata - to handle parsing microdata tags
    - Cheerio - virtual DOM with jquery-like syntax to be used for extraction algorithms
    - Compromise.cool - natural language processing library to detect verbs, etc

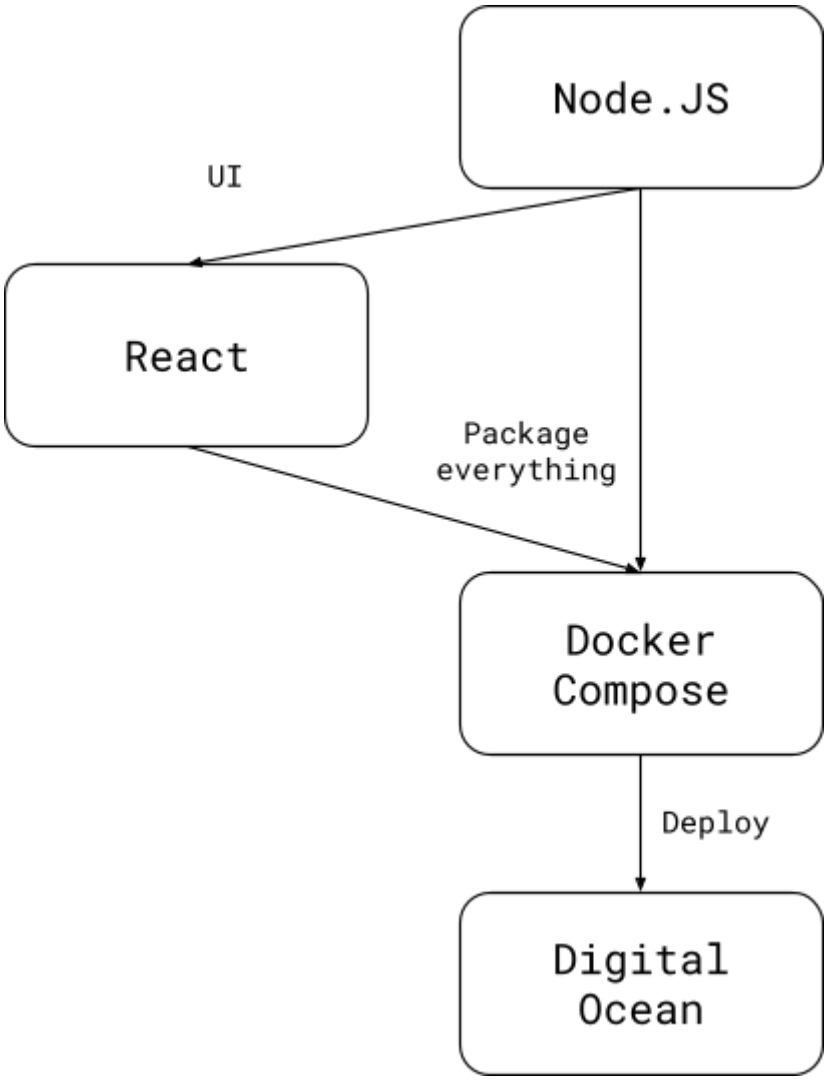
Frontend: React

- Interact with backend API and display extracted recipe
- Single page with two main states
    - asking for link to url that needs extraction
    - displaying extracted recipe

- Styling: SASS Modules or ChakraUI
- Components
    - Hero
    - IngredientList
        - Ingredient
    - DirectionList
        - Direction

DevOps: Docker
- Docker compose
- Containers make deployment easier

Component Map



Data

```
interface Recipe {
        // ? means optional
        title?: string;
        directions: string[];
        ingredients: string[];
        image?: string[]; //url
        url: string;
}
```

Repo Structure

```
appname/
   __init__.js
   package.json
   Dockerfile
   public/ [static/]
        css/
        js/
   views/ [templates/]
design.pdf
devlog.txt
flag.jpg
README.md
```

The file structure above is similar to a Flask/Apache server setup, with the main difference being that it uses docker. Additionally, instead of manually installing dependencies, we use docker to automatically create images and containers to deploy our web-app in one command.

The only requirements to run this is to have both docker Once you have that, a simple cloning of the repo is required and the command ‘docker-compose up’ will have the web application deployed.

Task Division

Shadman:	Parsing recipe data
Kevin:	Web server and deployment using docker and nginx
Ivan:	Frontend