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:

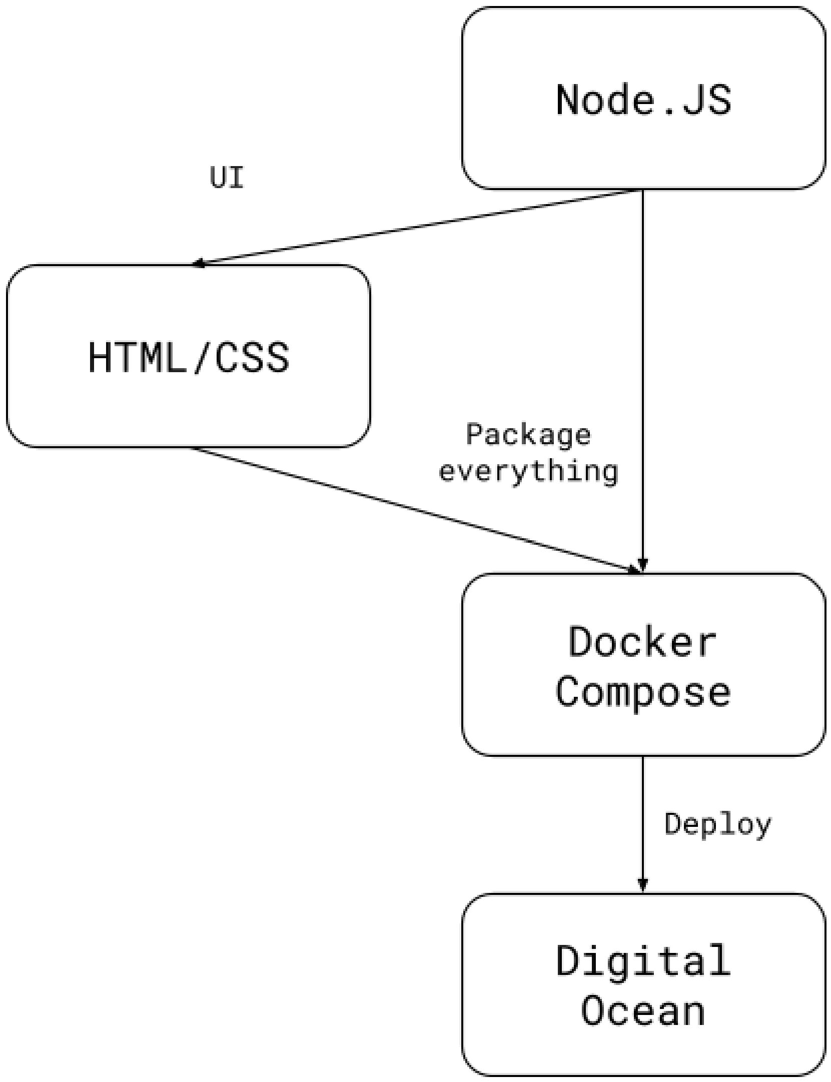
Backend: Node.js /w Typescript
- API endpoint for extraction
- Extraction
    - Schema.org JSON-LD data takes preference. If there is no Schema, move to the next method.
    - Look for microdata tags and extract using Regex. If there is no microdata, move to the next method.
    - Manually extract data
        - 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: HTML and CSS
- 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

- Components
    - Hero
    - IngredientList
        - Ingredient
    - DirectionList
        - Direction

DevOps: Docker
- Docker build
- 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/
  src/
        data/
        models/
        test/
        utils/ [functions]
        app.ts
  views/ [templates/]
        index.html
  public/ [static/]
        css/
  package.json [requirements.txt]
  tsconfig.json
  Dockerfile
design.pdf
devlog.txt
flag.jpg
README.md
```

Dockerfile

```
FROM node:16-slim
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
EXPOSE 8080
CMD ["npm", "run", "start"]
```

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

There are multiple ways to run this application.

- ➢ Pull image from Docker Hub (Requires Docker)
  - Open up your terminal
    - “sudo docker pull kcao20/recipe-extractor”
    - “sudo docker run -d -p 80:8080 kcao20/recipe-extractor”
- ➢ Create image and container using source code (Requires Docker)
  - Open up your terminal
    - “git clone https://github.com/shadmanrakib/recipe-extractor”
    - “cd app”
    - “sudo docker build -t kcao20/recipe-extractor .”
    - “sudo docker run -d -p 80:8080 kcao20/recipe-extractor”
- ➢ Using node.js (Requires Node.js)
  - Open up your terminal
    - “git clone https://github.com/shadmanrakib/recipe-extractor”
    - “cd app”
    - “npm i”
    - “npm run start”
  - Go to localhost:8080

Ted Topics

- Docker (https://piazza.com/class/kv0wqn7faux3ye?cid=339)
  - Simple deployment of applications
- Node.js (https://piazza.com/class/kv0wqn7faux3ye?cid=332)
  - Libraries/Modules
- Typescript (https://piazza.com/class/kv0wqn7faux3ye?cid=386)
  - Easier than js

Task Division

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

SHIP DATE: 6/10