TEONITE

Zadanie testowe - frontend

Cel zadania

Celem zadania jest przedstawienie danych, wystawionych przez API, którego obraz jest dostępny w załączeniu.

Sposób realizacji

Statystyki pobrane z API powinny być przedstawione w formie tabeli, użytkownik ma możliwość wybierania statystyk danych autorów.

Wymagania funkcjonalne

- W liście rozwijanej mamy do wyboru danego autora (autorzy pobierani z adresu <api_url>/authors/)
- Lista rozwijana powinna dać możliwość wybrania wielu autorów jednocześnie oraz opcję "All", po wybraniu której pobierane są statystyki słów dla całego bloga (dostępne po adresem <api_url>/stats/)
- Przedstawienie statystyk dla autora w postaci tabeli:
    - W kolumnie "Words" pojawiają się słowa najczęściej używane przez autora, słowa posortowane według liczby użycia
    - W kolumnie "Count" wyświetlana jest liczba użycia danego słowa - w przypadku wybrania wielu autorów, liczby są sumowane
    - Statystyki dla autora pobierane są z <api_url>/stats/<author>/

Wymagania niefunkcjonalne

- kod powinien być napisany w React w wersji co najmniej 16.3 + ES6 (Babel)
- kod wystawiony jako publiczne repozytorium na https://GitLab.com
- Zależności aplikacji umieszczone są w pliku package.json (wykorzystanie npm)
- Za pomocą Redux powinny być trzymane wewnętrzne stany aplikacji - połączenie z Reactem za pomocą react-redux
- Zapis danych pobranych z API powinien odbywać się poprzez redux-saga, a następnie dane te są przekazywane będą do reduxa
- Aplikacja ma być wystawiona pod adresem localhost:3000/stats/, routing stworzony w oparciu o react-router v4
- Cały projekt ma być uruchamiany za pomocą jednej komendy docker-compose

Wymagania dodatkowe (nieobowiązkowe)

- automatyczny build obrazu docker za pomocą GitLab CI
- po zbudowaniu automatyczne wrzucanie obrazu Docker na DockerHub (https://hub.docker.com/)
- wykorzystanie webpack jako bundlera

Przykład tabeli

AUTHORS:

	Word	Count
1	TEONITE	500
2	jest	444
3	super	333
4	tra	234
5	la	123
6	lala	98

7	heja	65
8	ho	40
9	pa	37
10	papa	18

Zawartość pliku docker-compose.yml do uruchomienia API

```
version: "2"
services:
  scraper:
    image: teonite/test-scraper
  depends_on:
    - db
  entrypoint: dockerize -wait tcp://db:5432 scrapy crawl blog

  api:
    image: teonite/test-api
    ports:
      - "8080:8080"
    depends_on:
      - db
      - scraper
    entrypoint: dockerize -wait tcp://db:5432 python3 manage.py
runserver 0.0.0.0:8080

  db:
    image: postgres:9.6
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=scrape
```

Kryteria oceny

- spełnienie wymagań funkcjonalnych jest obowiązkowe
- kod aplikacji będzie oceniany pod kątem jakości (im mniej błędów tym lepiej) i czytelności
- design strony nie będzie oceniany

Pomoc w realizacji zadania

Poradniki/dokumentacja:

- https://reactjs.org/tutorial/tutorial.html
- https://redux.js.org/basics
- https://reacttraining.com/react-router/web/example/basic
- https://github.com/redux-saga/redux-saga
- https://docs.docker.com/engine/getstarted/
- https://docs.docker.com/compose/

- https://scotch.io/tutorials/setup-a-react-environment-using-webpack-and-babel

Prezentacja Łukasza Piłatowskiego o dockerze:

https://www.youtube.com/watch?v=3JhmsyCHejQ

Dodatkowo każdy z Was otrzymuje koło ratunkowe, czyli może jednokrotnie poprosić o naszą pomoc w sytuacji, kiedy będziecie z jakiegoś powodu zblokowani. Pomocy możemy Wam udzielić zdalnie lub też w siedzibie firmy. Na spotkanie należy umówić się drogą mailową (mkoziel@teonite.com).

Powodzenia! 🙂