Sentiment Analysis Application

This is a sentiment analysis application that allows users to analyze the sentiment of a given text. A product is rated by the customer, the sentiment analysis system assigns a sentiment to the rating. User can give feedback about the sentiment, and admin(s) can all feedbacks.

The frontend employs the Metamask extension via web3 to display the balance on the ETH wallet.

Components

- **Frontend** is developed in react. React components use the restful API of the *backend* to fetch their data and authentication. Framework is React and language is Typescript.
- **Backend**: serves restful APIs for authentication, storing ratings and feedbacks and showing their list. Framework is Express and language is Typescript. Renaming of 'backend' is to be considered, since there is a third component as follows:
- **Analyze-engine** is a module implemented in Python that uses the open-source `transformers` library of Huggin-face to categorize the sentiment of a given sentence. This module provides restful API to receive texts and return sentiments.

Run

Currently three modules need to be set up separately:

1. **Analyze-engine**: Make sure the python dependencies are installed. `cd ./analyze-engine`, then `python3 app.py`. Restful API will be provided over localhost:8001
2. **Backend**: `cd ./backend`, then `npm run dev`. The backend will provide restful API over the localhost:8000 (details further below)
3. **Frontent**: `cd ./frontend`, then `npm run build`, `npm run preview`: to enable `http://localhost:4173/`

Data model

- `human`: representing users who sign in.
  - `uuid`: unique string user ID
  - `name` is the username
  - `hashed_password`,
  - `role`: A string with either `admin` or `user`
- `text`: This stores the ratings that users are able to add. Columns:
  - `id`,
  - `text`,
  - `userId`: the `uuid` of the user that issued the rating.
  - `sentiment`: Enum that has four values: `Undefined`, `Good`, `Bad`, `Neutral`. Default is `Undefined`, until the analyze-engine evaluation is received.
- `Feedback`: Users' feedback about the sentiment of their rating. -`id` integer primary key
  - `content`: The text of the feedback,
  - `textId`: Foreign key to the rating that this feedback concerns. `Feedback` and `text` have a

one-to-one relationship.

Restful APIs

Backend APIs:

- To signup: `POST` to `api/auth/signup` body: `{ name: string, password: string }`. To become an admin, sys-admins have to engage manually.
- To login: `POST` to `api/auth/signin` with body: `{ name: string, password: string }`. Response is: `{token : string, isAdmin: string}`. Token is the jwt token to be saved locally in the browser. `isAdmin` is either `true` or `false`.
- Putting a new feedback for a sentiment: `POST` to `api/feedback/new`, with the jwt token in header. Send the `{textId: number, feedback:string}` in the body. Possible previous feedbacks will be overwritten.
- Listing feedback: `POST api/feedback/all` with the jwtToken. Response is a list of `Feedback` objects such as following: `Feedback { id: number;content: string;originalText: string;sentiment: string;}`
- To add a text: `POST` to `api/text/analyze` with the jwtToken in the header, and the text in the body.
- To list texts of a user: `POST` to `api/text/all` with the `jwtToken` in the header. Response is an array of prototype Text: `Text {id: number,text: string,sentiment: string}`

Sentiment Analysis API:

- Have the sentiment for a text calculated: `Post` to `localhost:8001/analyze`, with `{text:string}`. Response is: `{text: string,sentiment: string,confidence: number}`

Testing

- The backend has been fitted with the `jest` library and one sample test, that checks for the `healthy` signal, to demonstrate the ability of smoketesting. The `app` object in `backend.setup.ts` is exported now to be testable.

Next steps:

Features that could not arrive soon enough:

- Expand testing to cover enough of the code, and log better.
- Use https, and hash the passwords on frontend before sending over the network.
- Dockerization: 3 docker containers for the 3 modules. Don't forget logging configurations.
- Benchmarking Docker Compose, Kubernetes and rivals, and making a decision there for the orchestration of dockers.
- Benchmarking and selection of a cloud service, then using Terraform or a similar tool to configure for the IaC.
- Benchmark the deployment tools based on the need, and make a decision (Github Actions, Gitlab, Jenkins)
- Add products and implement functionality to use the Metamask for purchasing them!