**CSC867 Team Sushi**

In order to complete the project we decided to split the team into two seperate groups. Three would work on the backend and three would work on the frontend. We used Kafka to create a restaurant which would then put it into the “conveyer” by the producer. Our consumer would then listen to anything added on to that “conveyer” and then would create that restaurant in our database. For websocket, we created live updates for the restaurants so that if a business creates a new restaurant, then it will be updated live. We would have added it for the reviews, but we decided to focus on finishing the minimal requirements since other members had other projects to do. For redis, we cached user information so we wouldn’t have to hit the database too much. For docker, it was configured successfully and is hosted in our remote server. Both frontend and backend are together in docker. We divided each services into their own separate containers. Following the microservice architecture, we multiple services which handled the restaurant, review, login, register, user, and auth.

One of the difficulties was configuring Docker. Our configuration would work when running in PM2, but it failed in Docker. This was usually due to different Node version in Docker or the docker version in “docker-compose.yml” was set wrong.

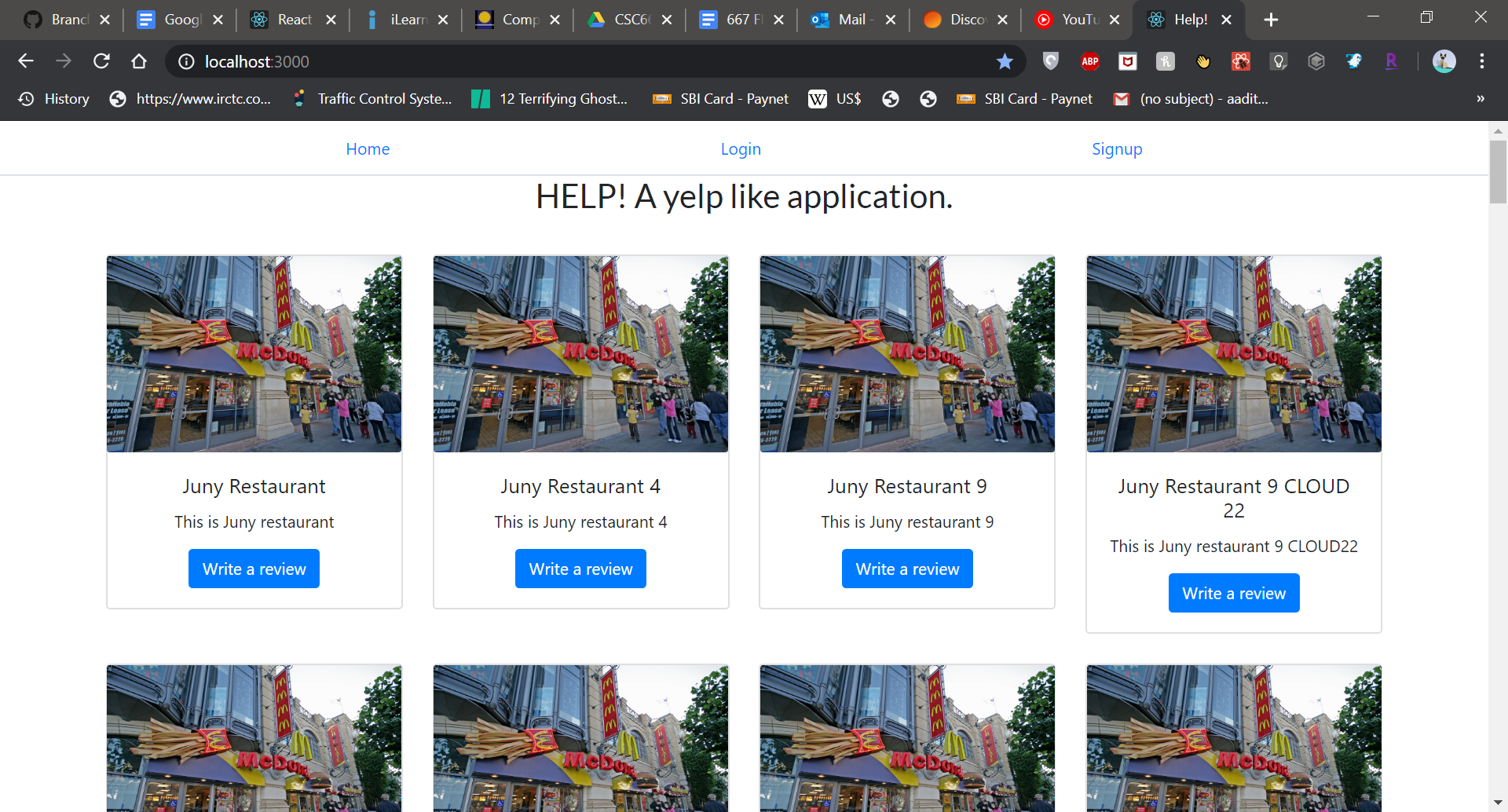
We used Postman to test the backend since we didn’t want to rely on the Frontend to test our backend. The frontend used mock data rather than needing to rely on data from the backend during the beginning of the project. This allowed us both to work in parallel without relying on one another.

There is also documentation written for the backend to assist in understanding how the API works. If you enter the “backend” folder in the GitHub, you can scroll down to see the readme which will contains the documentation.

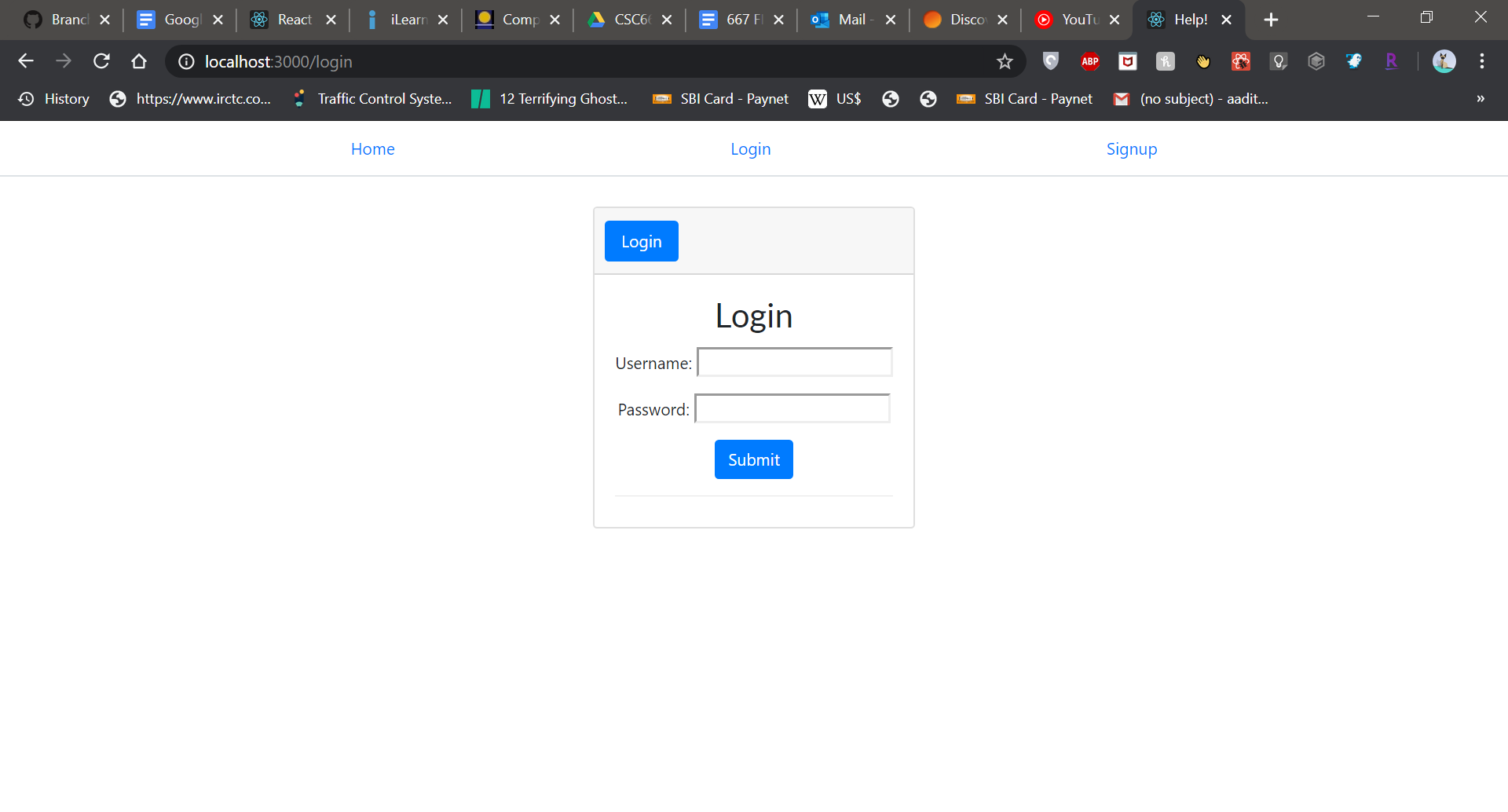
The problems faced by the frontend team was to organize how the flow of the application would be. There were some issues that we were facing with implementing the kafka and websocket code on the front end, but with the help of the backend team, we were able to solve it. Redux implementation was straightforward, used for keeping the username and personal data persistent, to make the website more personal. Transforming from dummy data to the actual data provided from the backend turned out to be a bit of a hassle, but we did solve it eventually. The login and signup was using the same service for login/registration making it a bit difficult to combine everything with the backend.

**Screenshots:**

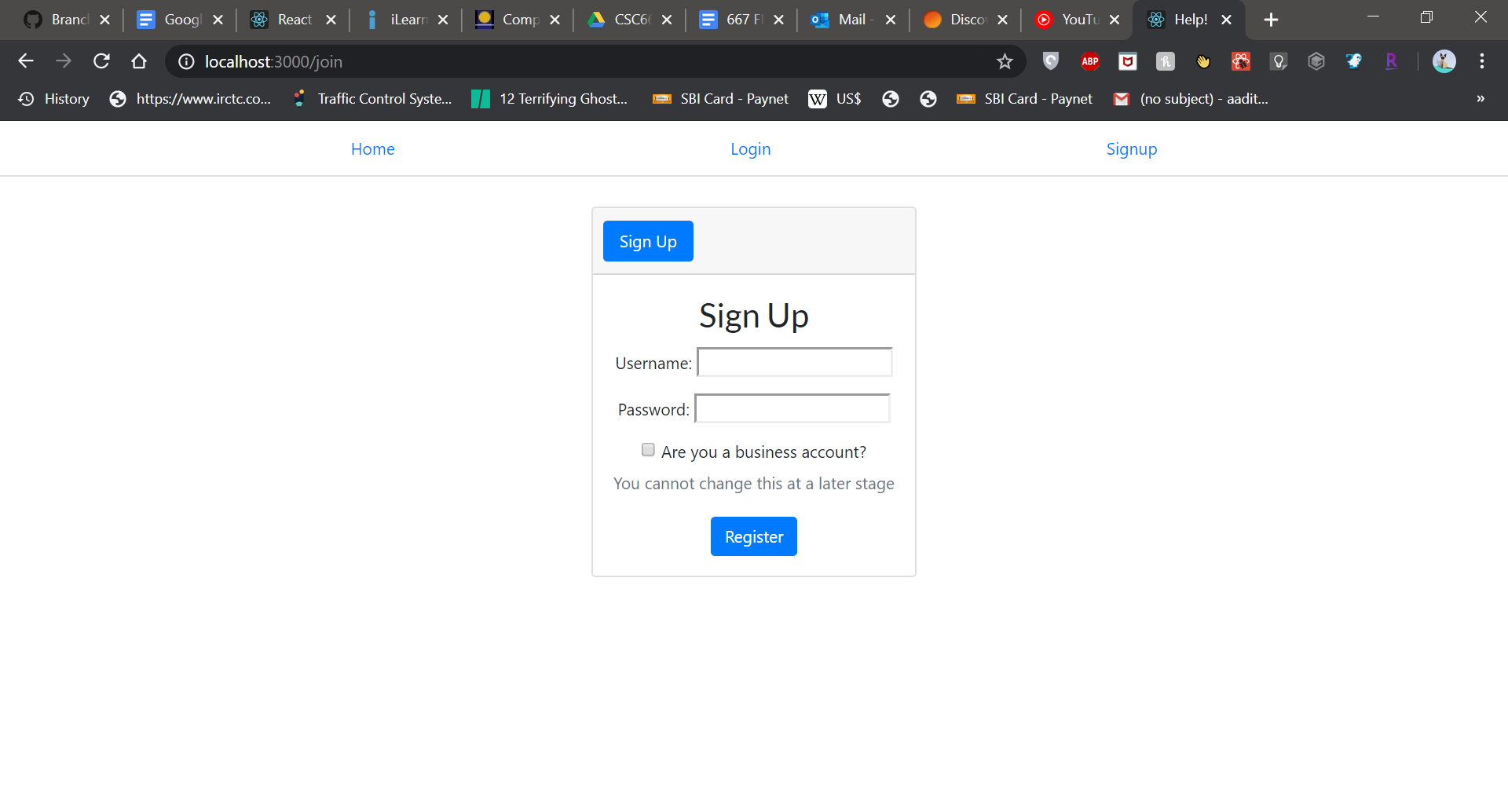
Home Page:



Login Page:

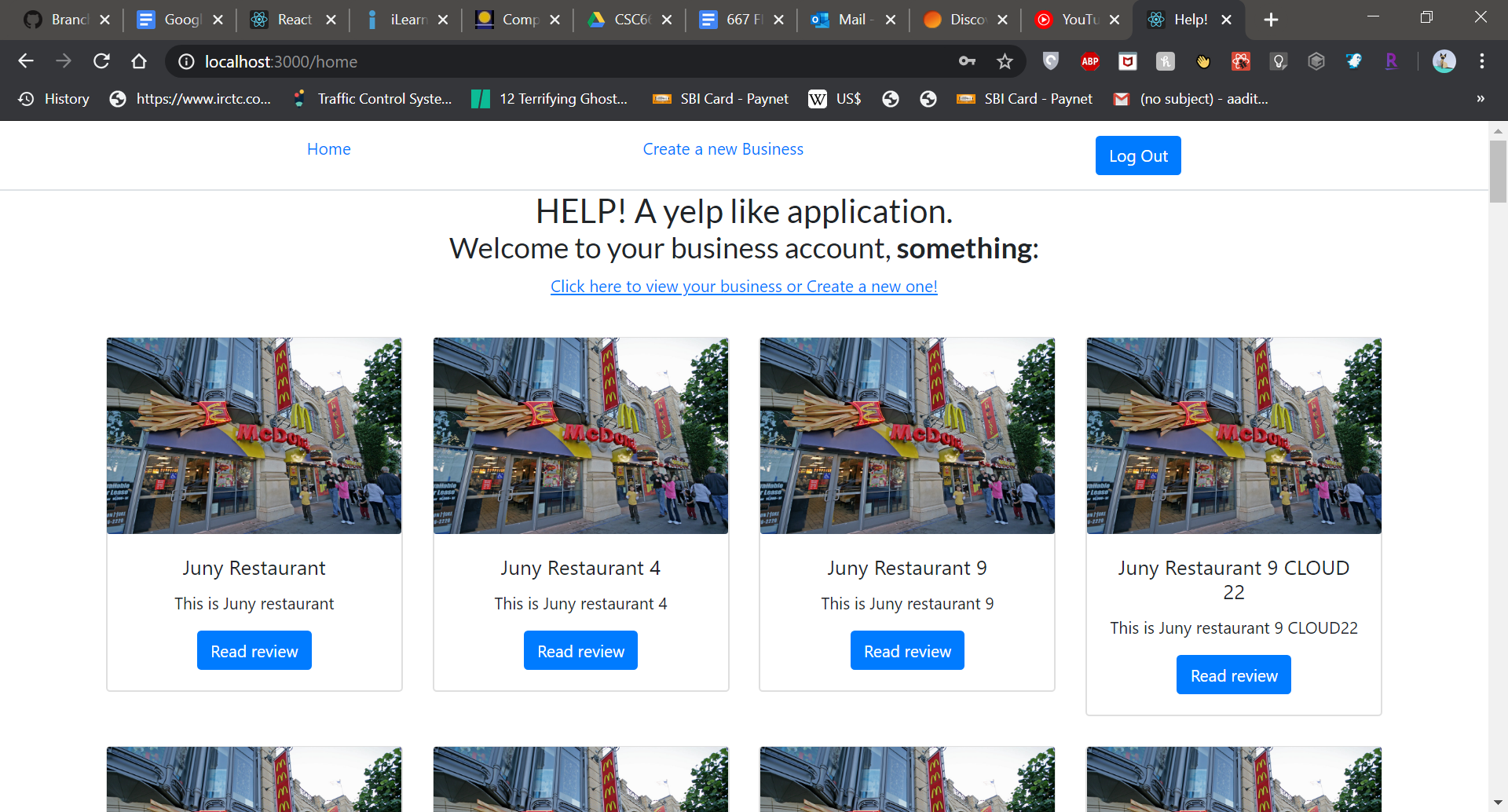


Signup page:



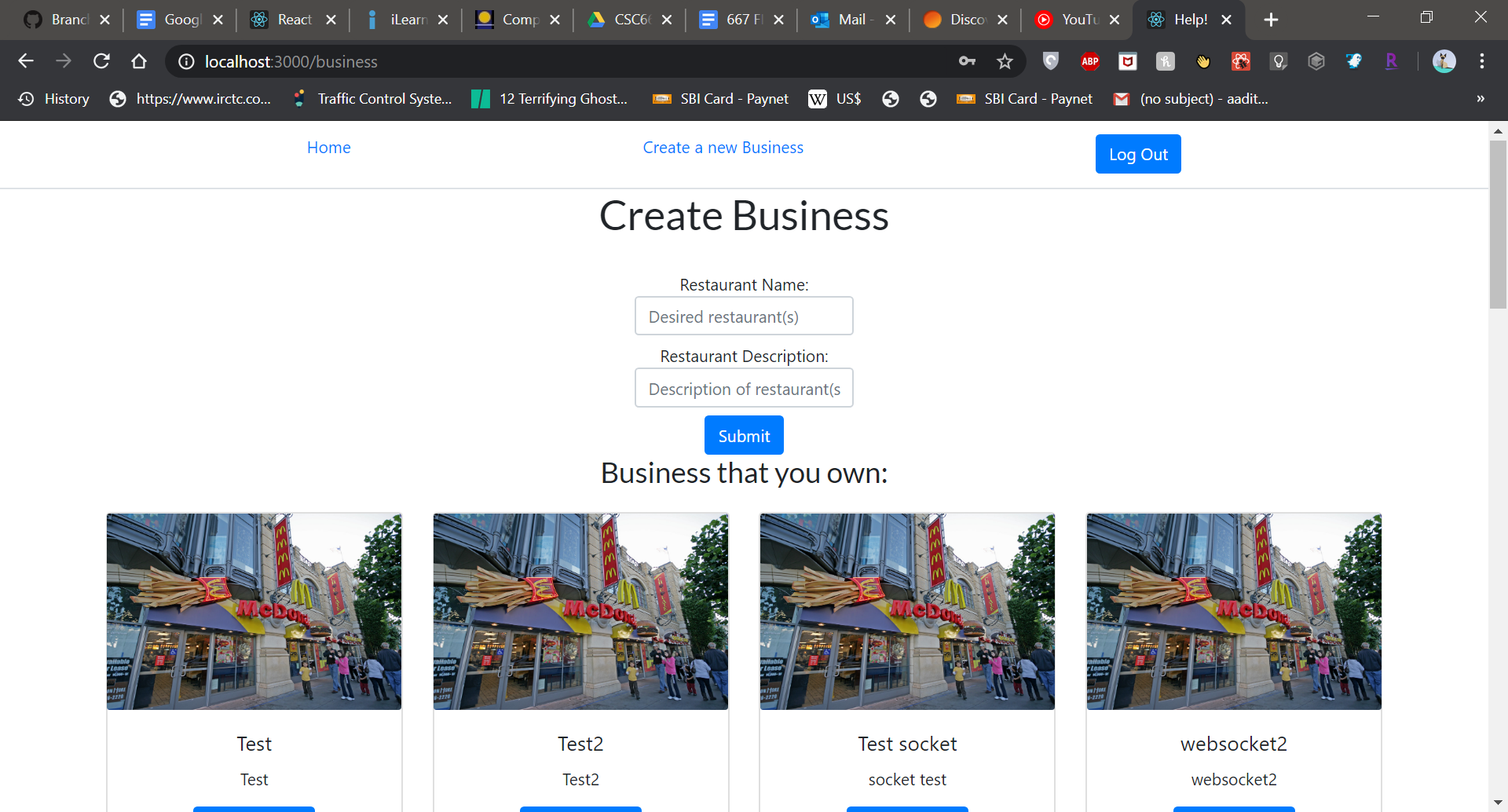
Signup has the option to be a business account, which cannot be changed at a later stage.

Business account home page:



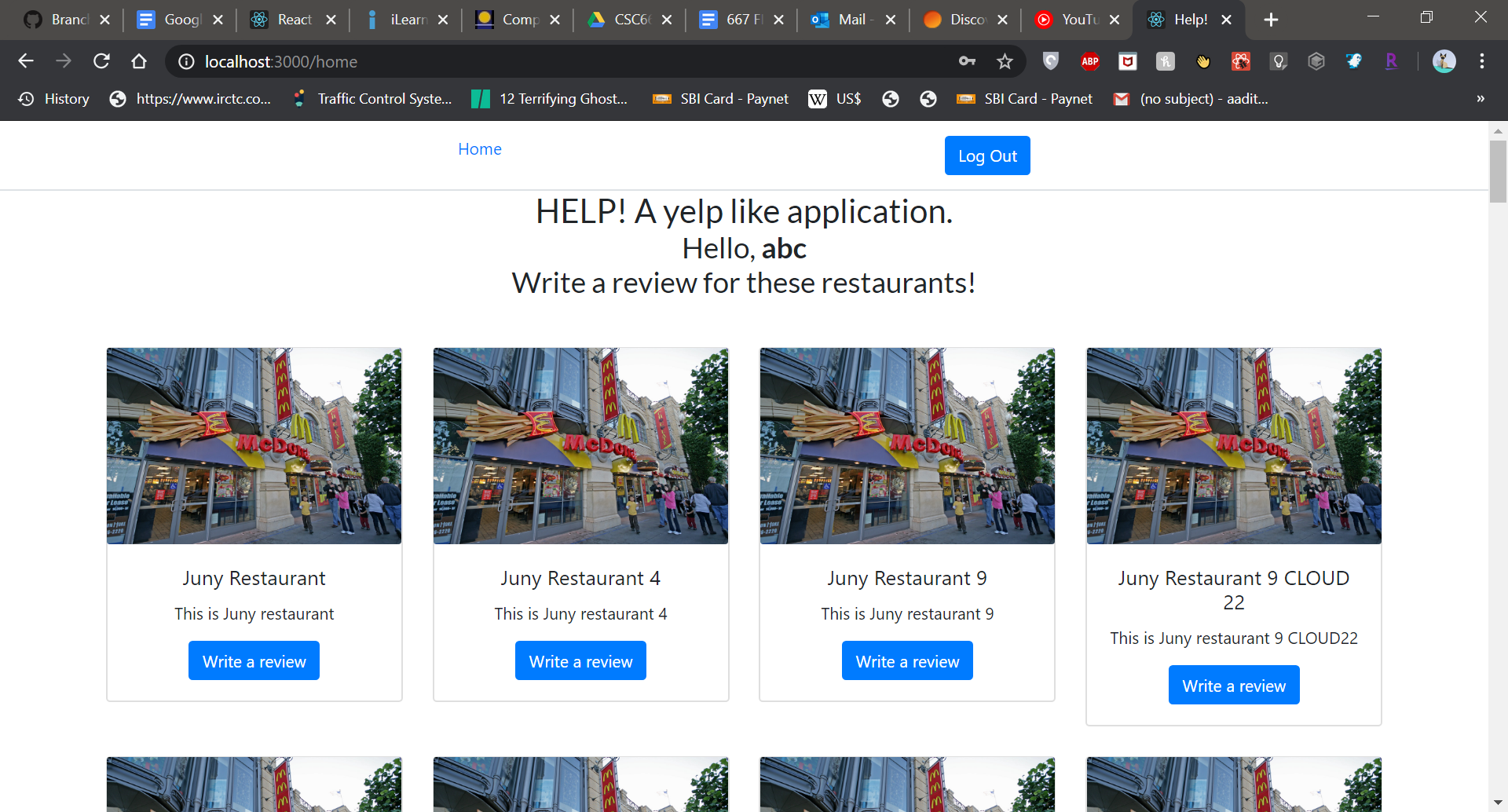
Business user has the option to create a new business but can only read the reviews for businesses.

Create Business Page:

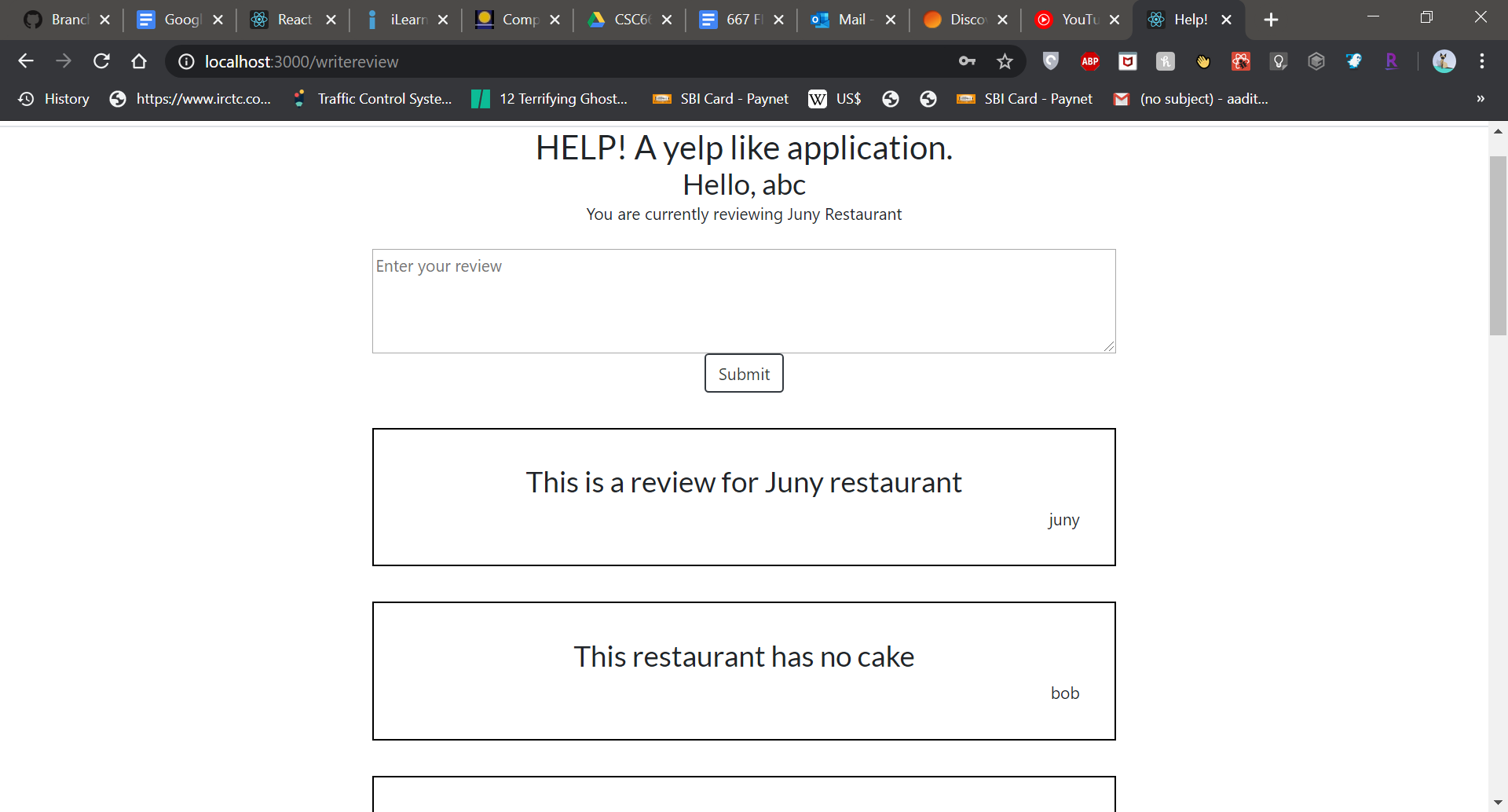


The create business page shows the restaurant owned by a business user, with the option to add a new business. The new business is updated on the homepage using web socket and kafka.

Normal User account homepage:

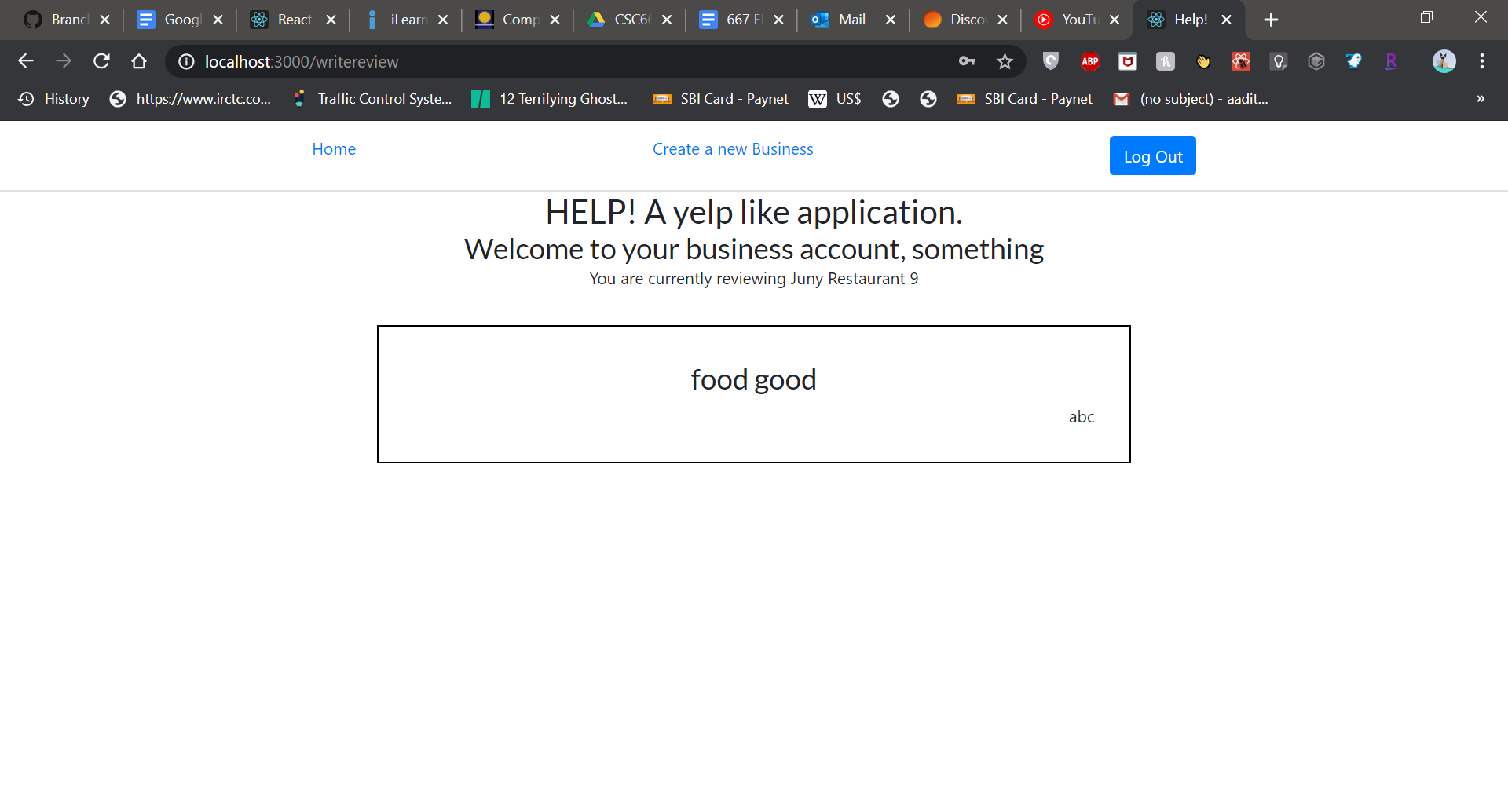


Write a review (User account):



How reviews would look to a user, with a user account having the option to post a restaurant for a restaurant.

Read a review (Business account):



The business account would only have the option to read reviews, and not write their own review.