BruinEats

by B-Plate Enjoyers

Amanda Xu
Shannon Wang
Rose Wang
Julia Gu
James Wang

BruinEats

- A web app designed for UCLA students to rate and review the food trucks offered to students with meal plans
- View all food trucks in one place
- Tailored to make reviews as helpful to students as possible
  - Wait time
  - Meal Period (Lunch, Dinner, Late Night)
  - Sorting, filtering and liking reviews

Team

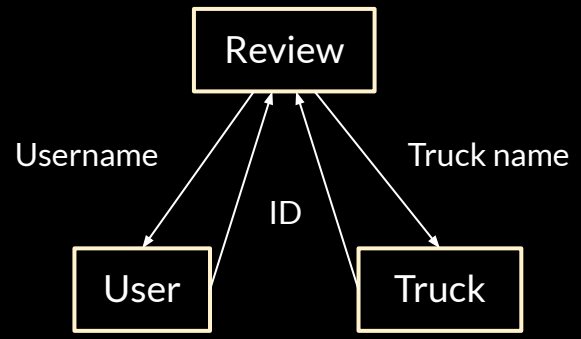
Amanda Frontend	**Shannon** Backend	**Rose** Frontend	**Julia** Backend	**James** Backend

Basic Features

1. App can display dynamic data to the user.
    a. Truck page changes based on logged in status
        i. View food trucks and their reviews
    b. Dashboard page where user can view reviews and choose their favorite truck
2. App can upload data from the client to the back-end.
    a. Saving reviews, users, trucks into database
3. User can meaningfully search through server data.
    a. Search for trucks from a dropdown menu
    b. Search for reviews using filters and sort methods

Basic Features Design Decisions

Frontend	Backend

• Global state to remember if user is logged in
  ○ Determines whether or not post review section on Truck pages is viewable
  ○ You can access dashboard page logged in, else user can only go to login page
• Sorting, filtering reviews always available; regardless of login
	• Division into three linked models
	  ○ 
	  ○ Creating a review will add its ID to the user who posted it and the truck it reviews

Schema

Trucks

_id
name
blurb
reviews
ratingAvg (updated on review addition)
waitTimeAvg

User

_id
name
    first
    last
username
email
password
reputation
favorite
reviews
likes

Review

_id
username
truckname
meal
waitTime
rating
review
likes
date (automatically generated)

Security Component

1. Create an account
   a. User must choose a unique username
   b. User must type an email that includes the @ character
   c. User must choose a password of ≥ 8 characters and is “strong” password based on the zxcvbn algorithm
2. User must login with that account: if invalid username/password, it is notified on the page

Motivation:

1. In order to post review, users must be logged in

Security Feature Design Decisions

Frontend	Backend
• Displaying messages on the side to represent invalid usernames/passwords/email • Toggling on and off password during creating account and logging in	• Querying our database to check if user login details are valid • Enforcing a unique username in the user schema for account creation

Additional Features

1. Post reviews
    a. Wait time and meal period, blurbs
2. Visit user dashboard page
    a. View your own reviews: sort and filter
    b. Selecting a favorite restaurant, know your own reputation
        i. Build reputation
            1. Posting: +10
            2. Liking a post (not your own): +1
3. Locations page
    a. To view all trucks and summary of their info on one page
    b. Visit truck pages from locations page

Additional Features Design Decisions

Frontend	Backend
• Sorting is a radio button (single choice) • Filtering is checkboxes (can select multiple) • Liking posts is a toggle button (only editable assuming user is not liking their own post) • Reputation scores are displayed on dashboard for user to check as they post and like posts • Posting reviews: a form in which all data is passed over to back end	• Query database for truck documents while filtering for only documents with the desired fields • Toggle a user's like to a review given the array of reviews a user has liked • Follow a review when a user likes a review to ensure review is not posted by the user • Update user reputation within functions that add reviews and toggle likes

Future Improvements

1. Post anonymously (mostly used for guest users who haven't logged in)
2. View profiles of other users
    a. So an individual can gauge for themselves how trustworthy a review is
3. More technical form of user authentication
    a. Sending an email with a code or some form of duo authentication

Largest Challenges

Frontend	Backend	Meshing the Two Together

• Storing a logged in state accessible on all pages
  ○ Use a context hook that essentially globalizes the state so all pages access login state
	• Writing a schema for our database
	  ○ Contingent on how backend and frontend would interact
	  ○ Had to be rewritten multiple times as we changed what data we needed
	  ■ MealReview
	  ■ ratingAvg
	  ■ image
		• Forcing a re-render of a truck page when a new review is submitted to show the review under the posts column

Libraries Used

Frontend (Credits for components usage)

- **Tailwind, DaisyUI:** NavBar, Buttons look consistencies, Star Ratings
- **React-password-strength-bar:** Viewing strength of password in account creation

Hooks:

- **useContext:** storing a login and userLoggedIn state accessible in all pages
- **useState:** inputs for forms and storing states in general
- **useNavigate:** navigating between pages
- **useEffect:** ensure that we don't work with undefined data before fetch request is resolved

Libraries Used Cont.

Backend

- **cors:** allow our frontend and backend to communicate on different ports
- **dotenv:** safely store our mongoose connection URL
- **express, mongoose:** enable us to interact with our database/frontend
- **nodemon:** library used for development only, which would automatically rerun our backend server so that we didn't have to run "npm start" each time we updated our backend

DEMO