Blue Ocean Engineering Journal

Demo: https://gallant-torvalds-547222.netlify.app/

Links:

- Ticketing System: Trello Board
- GitHub Repository: Repository Link
- Project Organizer: Google Doc
- Business Requirements: Google Doc

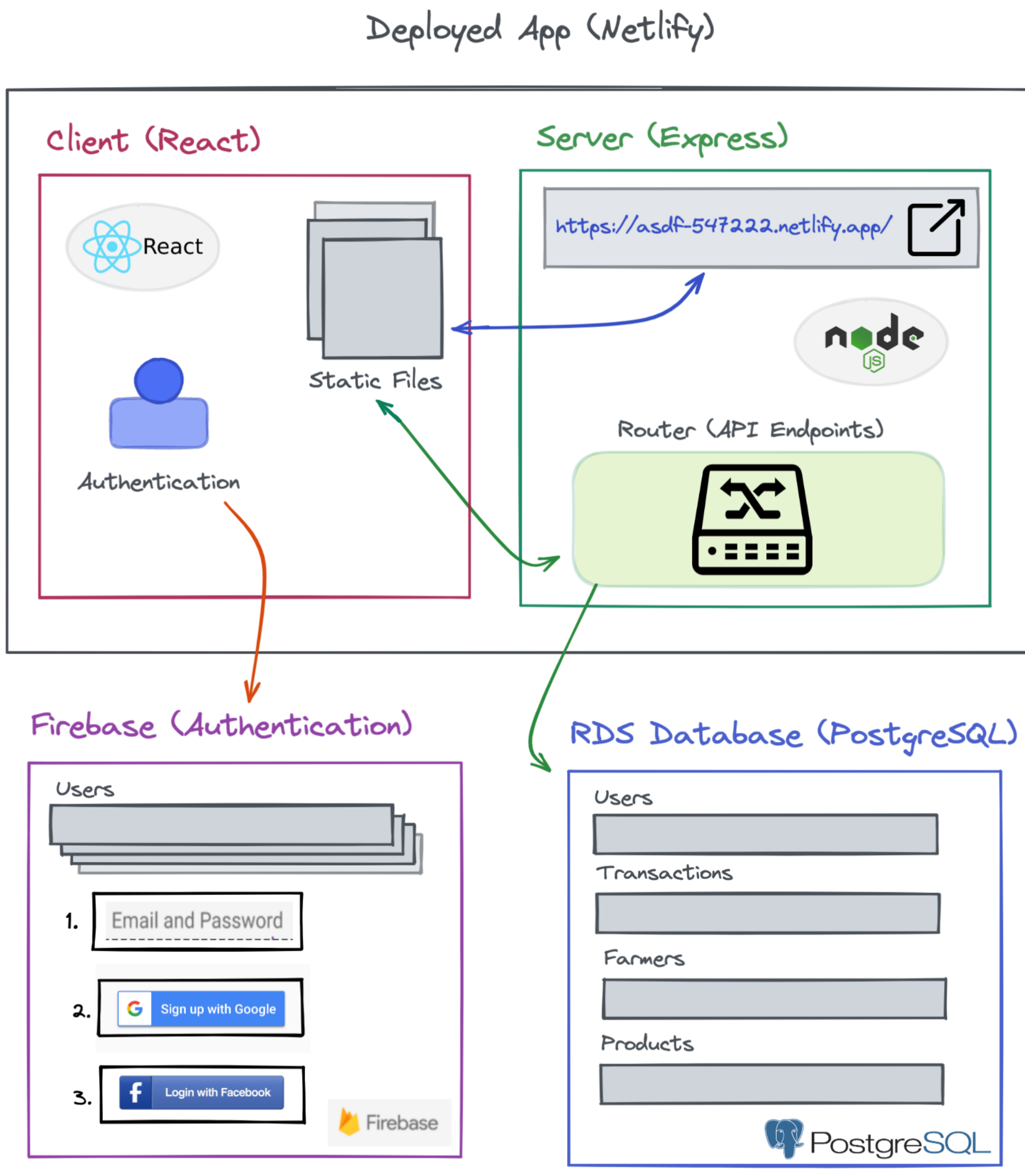
- Google Drive Folder: Drive Folder - OR SEE /resources folder in **MAIN**
- Discord Group: **Discord Link**

Contributors (Team 1)

- Spencer Lepine
  - https://www.linkedin.com/in/spencer-lepine/
  - https://github.com/spencerlepine
  - Jeff Liu
  - Nicholas Ma
  - Cassandra Barragan
  - Andrew Hang
  - Bradley Caliva
  - Lawrence Sun
  - Walter Tang

Project Roles

Jeff Lui - Project Manager
Andrew Hang - Architecture Owner
Spencer Lepine - UI Owner



Day 1

Project Meeting to decide roles. Landed Role of UI Owner:
UI Owner (Elected by team)
You will develop the initial wireframes that will help generate and ultimately accompany the user stories.

Created business requirements document after client meeting. Discussed project with team. Thought about databases. Thought about realistic features within project timeframe (1 week). I created wireframes based on business requirements. Planned out the pages/route sitemap for the site. Visit the wireframe on Figma here.

Day 2:

Updated the wireframe with new user stories. Worked on polishing the business requirements based on Client expectations after another meeting. Worked on fleshing out the code base and getting the git workflow established BEFORE writing code.

Day 3:

Set up Continuous integration PR #1

Set up GitHub Actions to run test's on pull request, verify code styles with linter, and upload test coverage output to coveralls for a README badge.

Used husky, lint-staged, and ESLint to enforce code styles during pre-commit hooks that disable you from pushing messy code. Added Prettier to auto-format code (like single or double quotes)

See https://github.com/spencerlepine/continuous-integration-boilerplate for my continuous integration set up.

Day 4:

Set up firebase authentication
Style login /signup page with Material UI
Add jest testing

Day 5:

Created AuthContext to store signup/login methods connected to firebase, and store a currentUser object in state (with keys like displayName, email, and uid)

Created a Login/Signup Page that connects to Firebase and stores a user record with email/credentials.

Day 6:

Worked with Cassandra building out the Sign Up/Login Page, and the User Account Details page that renders when you are logged in.

Worked on adding some styles with Material UI components.

Worked on setting up the Postgres database. Deployed to AWS RDS and had postgres connection string to use. Group already made schema and dummy data, and they loaded into the SQL database.

Worked on adding customer records with address and data in the PostgreSQL database ALONG with Firebase.

Worked on setting up the Sequelize ORM to have a query in the backend.

Started building the Express server API endpoints. Designed a SCHEMA file with route constants.

Took out dummy data and wrote API queries in the React client.

Day 7:

Worked on more API controller functions and database queries.

Worked on bug fixes for storing the User Type (customer, farmer, or nutritionist)

Worked on style updates and final touches to Account Pages.

Deployed the project to Netflify. Netlify runs node express server on localhost port 8001, and serves static files from the React client that makes requests to localhost:8001.

Challenges

Situation: Need to store user records in Firebase AND PostgreSQL database. Firebase third party login/signup does not indicate signup, when do we create our user record? Not a binary state (logged in/out), logic was too entangled.

Task: Managing multiple user account types for one site, Conditionally render pages for admins.

Action: Show account type selection dropdown BEFORE login form. Stored the user_type in the database if missing. Work on storing user_type in React state/context. Having issues waiting for the database to fetch user account type.
Result: Stored the User type in local storage in the browser. Only let the user log in in callback if the account type matches.

Situation: Building the API endpoints and connecting everything to the database.
Task:. Needed to write queries to the database, to connect client and express server and then the database. Collaborating with team members on endpoints. Some data was missing/broken. Team was designing the data management ourselves, without clear direction.
Action: Switched dummy data from client to API, to connect client/server, and then connect server and database with queries. Slowly build each endpoint one at a time.

Continuous Integration

- GitWorkflow → Code Reviews + Feature branches
- GithubActions → Automated Testing + Coverage Upload
- Pre-commits → Enforce code styles

Deployment

- Netlify → Automatic Builds + Working main branch
  → Third Party + Static Files

Accounts / Authentication

- Firebase → Handle User Records
- Postgres database → Store User info
- Customer Types → Customer | Farmer | Nutritionist

Sign up / Login Page + Account Page