
Bufflist

Team

Nick Cervasio

Gio Evans

Spencer Furgerson

Trevor Liss

Alex Scarola

Tools

Agile

Purpose: Project methodology to prioritize units of work into smaller manageable pieces to deliver functioning software after each period

- On weeks with lighter sprints, we found ourselves straying from the agile framework due to a lack of acceptable user-stories to work on.
- We were able to either pivot on or modify potential features as the project developed - taking advantage of the agile mentality.

Jira

Purpose: Organization tool to manage and track software projects.

- Jira was great for keeping our project organized, allowing us to track and assign user stories easily.
- Keeping the board organized/developing stories became a bit of an unnecessary hassle. As the project evolved the board occasionally became out of sync.

GitHub

Purpose: Distributed version control tool where development teams can store and merge projects.

- Best platform for managing a group project with multiple people developing.
- On weeks where we had major feature updates, merge conflicts were often difficult and time consuming to solve.
- Our team could have benefitted from more extensive use of the branching feature for bigger pushes.

PostgreSQL

Purpose: Popular open-source relational database, characterized by many features that aid in software development.

- Very simple interface and functionality.
- Class exercises made it easy to integrate into our project design.
- Was initially hard to figure out how to host the database, but lab 6 showed us how to easily fix that problem using Docker.

Docker - LocalHost

Purpose: Popular open-source platform to containerize applications for standard performance across operating systems.

- Inconsistency with how often container should be restarted when editing certain file types.
- Docker allowed each of us to seamlessly host the database on our respective systems.
- Necessary to install additional dependencies that were not taught during class in order to maintain functionality of a container (i.e- cookies).

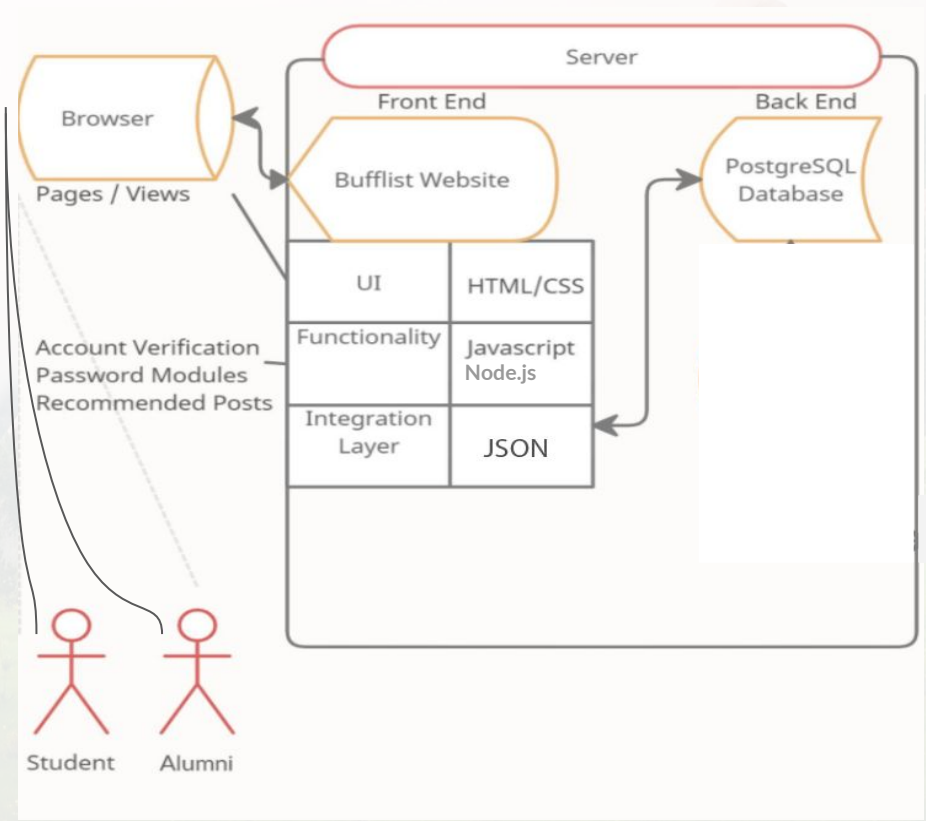
Node.js

Purpose: Powerful runtime environment to bring simple and scalable event-driven programming to websites and back-end API services.

- Simplifies the inclusion of Javascript without hard-coding client side scripts for all functionality.
- Generally a steep learning curve with debugging, writing functions and syntax. HTML to ejs conversion was complicated at times.

Architecture

Architecture Diagram



Explanation

- PostgreSQL stored the Users, Listings, and Pictures data tables
- JSON’s express.js used to generate queries based on user input that are tasked to the database and return the relevant data
- Pages built using EJS, CSS and Javascript connected with the database to display the stored Listings, Pictures and user information

- Docker containers on each of our local systems allowed us to see a visual of the front end and back end aspects and develop the website’s functionality

Challenges

Including complex or wishful features in the original plan:

- We chose to scale down some of these features or eliminate them entirely.
- It would have made more sense to start small with the core features and reachable extras.
- Thus, we could have finished those features and scaled up as time permitted.

Starting in one framework, ending in another:

- We started our development with static HTML and CSS, then switched to Node.js around mid project. This was due to uncertainty around whether or not we would use Node later on.

- This was overcame through the whole team contributing to the new end product and recycling usable code.

- We would have likely sought extra help to begin developing in Node.js from the beginning.

Constant changes to the database structure

- Team members that developed the backend database adjusted the structure each time we realized a new requirement.

- This might have been mitigated earlier on by developing a high level prototype to gain an accurate depiction of what would be necessary in the database.

Developing the frontend and backend separately

- The frontend and backend structures were developed separately and not merged until about halfway through the project.

- This was because we started our backend in PostgreSQL and were not sure if that would be what we stuck with the entire project.

- We likely would have brought the frontend and backend together earlier to ease the integration of the functionality.


DEMO!