Project documentation

Project name: PawTime

Summary (singular purpose): A platform where Finnish dog owners can search for nearby dog parks, reserve timeslots, and share details about their dogs for solo or social visits.

Detailed description: Users can browse nearby dog parks and reserve time slots for themselves, with their reservation visible to others. The reservation will indicate whether they prefer to meet other dogs during their visit or not. Additionally, users can provide details about their dog, including breed, size, and personality, and specify the types of dogs they'd like to meet to ensure safe meeting. The platform uses map API to share a map view and Geoapify Places API to see public dog parks in Finland.

Main learnings: I wanted to learn React and challenge myself with combining many APIs and use a database.

GitHub repository URL: https://github.com/MiisaSuorsa/paw-time

Public URL: https://pawtime.netlify.app/

Contact details: +358 404163901(messages only), or miisa.suorsa@student.lut.fi

Instructions

To run the app locally, download the repository and follow these instructions.

First you need to set up your own MongoDB connection (use MongoDB Atlas (Cloud-Based)):

1. Create a free account at MongoDB Atlas.
2. Set up a new cluster.
3. Whitelist your IP in Atlas (or allow access from any IP for development purposes).
4. Get your connection string from Atlas, which will look something like this: mongodb+srv://<username>:<password>@cluster0.mongodb.net/<databaseName>?retryWrites=true&w=majority
5. Create a .env File in the root directory of the project.
6. Add your MONGO_URI=[your connection string]

Next you can modify the code and run the application:

1. Change urls to work on local running. (files can be found from paths: server/server.js, src/hooks/usePostReservations.js, src/hooks/useReservations.js)
2. Run *npm install* to install required dependencies.
3. To run the application, run *npm start* , this will run the server and client simultaneously.

Detailed descriptions

Front end:

What does the front end do?

- Displays a map with dog park locations using markers that users can click to view details.
- Display the map based on the user's location.
- Redirects users to a reservation page where they can view and book time slots for a selected park.
- Fetches and displays available and already booked reservations, including details such as whether others can join.
- Provides a form for users to submit new reservations
- Ensures smooth navigation between pages (home page with the map and the reservation page).

What do users see?

- A map interface that allows them to locate nearby dog parks.
- A reservation form where they can:
    - View existing reservations.
    - Submit their own reservation with preferences.
- Confirmation messages or error handling feedback when interacting with the system (e.g. submitting a reservation or fetching data).
- Details about each reservation, such as time slots and whether the reserving user is open to others joining.
- Clear navigation and visual elements to guide them through reserving park time.

How does this front-end support the app's purpose?

- Making it easy for users to find dog parks through a visually intuitive map interface.

- Enabling users to quickly and visually identify parks through the map interface.
- Simplifying the reservation process with an intuitive flow: select park → view reservations → book a time slot.
- Encouraging community interaction by showing whether existing reservations are open for others to join.
- Providing a user-friendly interface to enhance engagement and usability.

What technologies will be used to implement the front end?

- React: The main framework for building the user interface to be dynamic and a component-based.
- React Router: For handling navigation between pages (e.g., home and reservations).
- Map API: Leaflet for rendering the map and park markers.
- Geoapify: Places API for getting the data of dog parks.
- JavaScript: Core programming language.
- Fetch API: For making HTTP requests to the backend.
- Netlify: To host the front end publicly.

How do these choices support the purpose of the app?

- React: Offers reusable components, making it easy to manage and update features like maps, forms, and reservation lists.
- React Router: Ensures smooth navigation between pages, improving user experience.
- Map APIs: Provide intuitive geographical visuals.
- Geoapify: provide a ready made information about the dog parks.
- Fetch API: Handles communication with the backend for retrieving and submitting reservation data.
- Netlify: Simplifies deployment, making the app available to users quickly and reliably.

Are there other choices that could do the same?

Yes, alternatives exist for these technologies:

- Vue.js or Angular instead of React: Both are robust frameworks for building dynamic front ends.
- OpenLayers instead of Leaflet: Another option for interactive maps.
- Render or other hosting platform for hosting the whole application.

Why not use these other choices?

The chosen technologies are preferred because:

- React has a larger community and ecosystem, making it easier to find resources and libraries. I also wanted to challenge myself and learn React.
- Leaflet is well-documented and widely used, providing reliability and ease of integration with React.
- I chose to use Netlify in the beginning, and found only later that it wouldn't support to hosting my backend. I still decided to use Netlify for the frontend since I already had deployed it there.

Are there any wireframes or diagrams guiding these choices?

No

Will any design or CSS frameworks be used?

Very little, but yes, a basic CSS is used. The react app creates a bit of a CSS along with it, and I used bootstrap with the navigation bar.

Back end:

What is the primary role of the back end in this project?

- Managing data storage and retrieval in MongoDB (e.g. storing reservations and fetching the data).
- Providing a secure and reliable API for the front end to interact with the database.

How does the back end support the app's purpose?

- Acting as the bridge between the front end and the database, ensuring accurate and up-to-date reservation information is available to users.
- Enforcing data integrity through schema validation (e.g., ensuring only valid reservations are saved).

What technologies will be used to implement the back end?

- Node.js: The runtime environment for building the server.
- Express.js: A lightweight framework to define routes and handle HTTP requests.
- MongoDB Atlas: The database used to store and manage reservation data.

- Mongoose: A library to define and interact with MongoDB schemas.
- Cors: To manage cross-origin requests, allowing communication between the frontend and backend.

How do these choices support the purpose of the app?

- Node.js + Express: Provide a fast, scalable, and flexible server that can handle asynchronous requests efficiently.
- MongoDB: It allows dynamic and scalable data storage, which is ideal for storing reservation details.
- Mongoose: Simplifies MongoDB interactions with built-in validation, making it easier to maintain data consistency.
- Cors: Ensures secure cross-origin communication between your Netlify-hosted frontend and Render-hosted backend.

Are there other choices that could do the same?

Back-end Frameworks:

- Django (Python) or Ruby on Rails: These frameworks offer robust tools for building backends but might be overkill for a simple reservation system.
- Spring Boot (Java): A solid choice for enterprise-level applications, but more complex to set up and maintain.

Databases:

- PostgreSQL: A relational database with powerful features, suitable if the app required complex data relationships.
- Firebase: A serverless backend with real-time database capabilities, good for simpler apps.

Why not use these other choices?

- Node.js + Express aligns well with React for full-stack JavaScript development, reducing context-switching between languages.
- MongoDB is ideal for flexible, JSON-like data structures (e.g., reservations), avoiding the complexity of rigid schemas in relational databases.
- Alternatives like Django or Spring Boot have steeper learning curves and require more configuration.

What platform will be used?

The backend is deployed on Render.

Where will the project be hosted?

- Frontend: Hosted on Netlify.
- Backend: Hosted on Render, allowing it to serve API endpoints for the frontend.

In what language will the back end be written?

The backend is written in JavaScript, leveraging Node.js and Express.js for server-side programming.

How will the back end handle security and data privacy?

- Environment Variables:
  - Sensitive data like the MongoDB URI and API keys are stored securely in `.env` files and on Render's environment configuration.
- Cross-Origin Resource Sharing (CORS):
  - The backend restricts which domains can access it
- Validation:
  - Mongoose schemas validate reservation data before saving it to MongoDB to ensure integrity.
- HTTPS:
  - Render automatically enables HTTPS for secure data transmission between the frontend and backend.

Database:

What type of data will the app be storing?

Reservation data:

- Park ID or location details.
- Reservation date and time.
- User preferences (e.g., whether others can join the reservation).
- Other information about the reservation.

How does the choice of database support the app’s purpose?

- Handling flexible and scalable data storage: MongoDB's schema-less design allows dynamic reservation records that can adapt to evolving requirements.
- Enabling quick querying: MongoDB supports fast filtering and sorting for large datasets, such as finding reservations by park ID and date.

What database management system will be used?

- MongoDB: A NoSQL document-based database system.
- Managed via Mongoose, an ODM (Object Data Modeling) library in Node.js.

How do these choices support the purpose of the app?

- MongoDB:
    - Works seamlessly with Node.js and Mongoose.
    - Handles large, unstructured, or semi-structured datasets, which aligns with flexible reservation data.
    - Scales easily as the app grows in user base or adds new features.
- Mongoose:
    - Simplifies database interactions by providing schema definitions, validation, and hooks (e.g., pre-save checks).
    - Protects data integrity by enforcing rules on the structure of stored documents.

Are there other choices that could do the same?

- PostgreSQL or MySQL: Could store reservations in structured tables with columns for park IDs, dates, etc.
- Firebase Realtime Database or Firestore: Could handle dynamic data storage and real-time updates.

Why not use these other choices?

- PostgreSQL or MySQL: These require rigid schemas, which can slow down development when requirements evolve. MongoDB's flexibility makes it easier to adapt to changing data structures.
- Firebase:
    - Works best for small-scale apps or apps requiring real-time updates.
    - Limited querying capabilities compared to MongoDB.

MongoDB strikes the right balance between flexibility, scalability, and simplicity.

How will the database handle security and data privacy?

- Environment Variables: MongoDB URI (with username and password) is stored securely in `.env` files and Render environment settings.
- Access Control: MongoDB Atlas can restrict access based on IP addresses.
- Data Validation:Mongoose ensures only valid and structured data gets stored in the database.

API schema:

What type of data will the API be handling?

- Reservations: Fields like park ID, date, time, user preferences, and other details.

How does the choice of API schema support the app's purpose?

- Defining clear endpoints (`/api/reservations`) for managing reservations.
- Allowing users to:
    - Fetch reservations on the page.
    - Post new reservations based on user input.

What technologies will be used to implement the API schema?

- Node.js: To run the server.
- Express.js: To define routes and handle requests.
- Mongoose: To validate and interact with MongoDB.

How do these choices support the purpose of the app?

- Node.js and Express: Enable lightweight, fast, and asynchronous handling of requests, suitable for a reservation system.
- Mongoose: Ensures data integrity and provides easy querying

Are there other choices that could do the same?

- Django REST Framework: For building APIs in Python.

Why not use these other choices?

- Django REST Framework: Adds unnecessary complexity by using another programming language (Python).

Third-party APIs:

What third-party APIs will the app be using?

- Map APIs (Leaflet): To display the map and markers.
- Geoapify (Placees API): To get the location and other informal data of dog parks.
- Hosting APIs:
    - Render API: For deploying and managing the backend server.
    - Netlify API: For deploying and hosting the frontend.

How do these APIs support the app's purpose?

- Map API with Geoapify: Helps users find and book reservations visually, enhancing the user experience.
- Hosting APIs: Provide seamless deployment and scaling options for both frontend and backend.

What technologies will be used to implement these APIs?

Map API with Geoapify: Used in the React frontend to render interactive maps.

How do these choices support the purpose of the app?

Map API with Geoapify: Ensure a smooth and reliable user experience for visualizing dog park locations.

Are there other choices that could do the same?

OpenStreetMap: A free alternative to Leaflet with similar functionality.

Why not use these other choices?

No specific reason to not use OpenStreetMap. Leaflet just had good documentation about using with React, so I chose that.

Architecture:

What is the overall architecture of the app? How are the separate components organized to work together?

The app follows a 3-tier architecture:

Frontend (React + Netlify):

- Hosted on Netlify.
- Displays the user interface (map, reservation form, etc.).
- Fetches data from the backend using API endpoints.

Backend (Node.js + Express + Render):

- Hosted on Render.
- Acts as an intermediary, providing data from the database to the frontend.
- Handles user inputs like creating or fetching reservations.

Database (MongoDB):

- Stores reservation data.
- Provides query capabilities for fetching and saving data.

How does this architecture support the app's purpose?

- Separation of concerns: Each tier is responsible for a distinct role (UI, logic, and data storage).
- Scalability: The backend and database can scale independently of the frontend.
- Flexibility: Future features (e.g., user authentication, park management) can be integrated without disrupting the core architecture.

What technologies will be used to implement this architecture?

- Frontend: React, hosted on Netlify.
- Backend: Node.js, Express, hosted on Render.
- Database: MongoDB Atlas, managed using Mongoose.
- APIs: RESTful APIs to connect the backend with the frontend and database.

How do these choices support the purpose of the app?

- React: Allows for a dynamic and user-friendly frontend with reusable components.
- Node.js/Express: Provides a lightweight and fast backend, ideal for handling API requests.

- MongoDB: Supports flexible and scalable data storage for reservations.
- Netlify and Render: Simplify hosting, deployment, and scaling.

Are there other choices that could do the same?

Yes, alternatives include:

- Frontend:
  - Vue.js or Angular instead of React.
- Backend:
  - Python/Django, Ruby on Rails, or PHP/Laravel.
- Database:
  - PostgreSQL, MySQL, or Firebase.

Why not use these other choices?

- React: Well-suited for this project due to its component-based structure and widespread support.
- Node.js/Express: Lightweight and integrates seamlessly with MongoDB.
- MongoDB: Provides the flexibility that a reservation system requires.

Definition of success (e.g. “The project will be successful when a user can…”)

The project will be successful when:

1. A user can view dog parks on the map and make a reservation at a park.
2. Users can fetch and view existing reservations for a park, including details about whether others are allowed to join.
3. The system reliably stores and retrieves reservation data from the database, ensuring an error-free user experience.

Self-reflection

If you could send advice back to yourself at the beginning of the course, what would you tell your less-experienced self? In other words, what do you wish you had known when you started your project?

Don't overthink, it's a big project to do by yourself and without instructions to follow. But you are capable of doing the design decisions and implementing them as you wished. Prioritize focus time for the project and take small parts to improve in one set instead of thinking the whole and getting overwhelmed.

Evaluate your technical growth through this project. What practical steps did you take to address gaps in your technical knowledge or skill? What new skills or knowledge did you acquire? In what areas do you feel you need more development?

I had done web applications course a one year before this, but clearly I had forgotten a lot of full stack development. Understanding the whole frontend or whole backend was easy, but understanding them together was a big challenge that I believe I succeeded in. To understand the communication between frontend and backend I did read different sources, reflected on my web application course coding tasks, and asked clarifying questions from ChatGPT.

I'm also very impressed on the technical skill of hosting the app online. I had previously done it once by using GitHub pages, but Netlify and Render were actually really useful. Even though I learned that Netlify as a hosting service wasn't capable of hosting my backend, so that I needed to host the backend separately on Render.

The biggest/most important technical skill that I have learned through this project is React. I had some knowledge about it but didn't quite understand or know how to code before this project. Now, I feel like I've finally understood the idea of it, its components and hooks, and why my one friend, who is a frontend developer, really likes it and encouraged me to learn. Maybe next I'll try it with Typescrip as she also recommended.

For the future, I think I still need more practical experience and learn by doing with full stack. Even when, I have learned a lot on this course, but this is still something that I can't fully learn on one course and need more time for it. I also have had a really hard Autumn personally and scheduling this self-organized project was challenging for me. I'm proud that I've managed to get a working solution even when its details aren't developed as far as I planned.

Describe a technical risk you took in your project. What was the outcome, and what technical or personal insights did you gain from this experience, regardless of its success or failure?

My technical risk was to take a new library to learn (React) into the project, even when I knew that I'll have difficulties on the schedule due to personal difficulties in this Autumn. I feel like the project idea itself wasn't also the simplest one, since I had to use few API and handle data fetching and posting. One technical aspect that I seriously did research on was websockets that I then decided to leave out of the scope when developing the first version of the app. That would be an ideal addition to the app in the future if I decode to further develop it, but based on my research I decided that it will be left out. I understood that for that use I should have chosen a different hosting service and for that I also should pay for that service, which I wasn't willing to do now.

Discuss a significant technical decision you made in your project. How did you approach this decision, and how would you assess the effectiveness of your decision-making process? What would you do differently next time?

As I mentioned previously one technical decision was made to leave out websockets even when it would have been a perfect addition to it. To make this decision I did research on the websockets itself, how they work and how could I apply them to my project. I also did research on the hosting services and asked questions from ChatGPT to clarify the things that I thought to understand from other sources.

When starting the project I didn't do much research on the hosting services, since it was required for the project to have a public url and I implemented it before I had more ideated my project itself. This came into a problem later when I started implementing the backend and learned that I can't run my server on Netlify. I then decided to just run the frontend on Netlify and backend on Render, instead of hosting the whole app in Render (which I believe would have been possible). So next time, I would ideate my project a bit more and list it's requirements regarding the hosting service, to be able to choose more wisely the right service.

How did you incorporate empathy into the design and development of your project, particularly in terms of user experience and team collaboration? How did this consideration affect the project?

I did have conversations about my idea with my few friends who study in the same field with me. From them I was able to get a developer perspective of my idea and solution, and confirmation on the decisions I had made. I also had few friends and family members who have zero technical skills of coding, to show my solution and idea and get feedback from them.

I don't feel like these affected to my project in a significant way. I mostly just got confirmation that my idea and implementation was good. Of course there are many things to improve, but in simplicity it has been good.

Reflect on your project as a whole. What technical aspects were most successful, and which

ones would you improve upon in future projects? How could course content or structure better support these technical endeavors?

As a whole I think my project is good. It's not too simple, and have required a lot of skills to implement. My most successful technical aspects are including react, leaflet, Places API, and MongoDB. From these I had previously really used only MongoDB. To further develop the project, these would still need to be improved also. In MongoDB I should also save the user data, that I now didn't include in the project. I also should find a connecting information of PlacesAPI that could be saved into MongoDB reservation data, to be able to identify the reservation locations and to show only relevant ones for the user.

I don't think that the course content could better support these improvements. Those are just things that I didn't have time to implement further.

What aspects of the course or teaching methods do you think could be improved? That is, how can I make this experience better?

I believe the content of the course is good, but to improve students' participation there are two things. Firstly, I think the biggest issue is the Moodle page itself. There could be tabs to divide the information, for example lectures, assignments, grading, and project details. The assignments were unclear since you have to click the lecture day to even see that there is an assignment and to be able to see instructions on the assignment you need to click to add submission first. The deadlines for the one assignment also seem unclear on the calendar as submission closes and submission deadline have different dates there. Also, the deadline for midnight should be at 23:59, since it will show it on the next date when set to 12 AM. (Sorry if it sounds too judgemental, but I've had many courses where these kind of things really decrease the motivation of students and create misunderstandings of assignments and deadlines)

Second thing to improve the teaching methods is the coalition with students. Maybe if there would be more information and instructions on how to do it and something that motivates more to participate in it. For example, the peer review could be done among the students in the coalition, so then there could be review about the process itself and not only the end result. I think this could actually be a very good idea since I think this course have been more about learning along the course of full stack, instead on proving the learning at the end of the course.

If you were to assign yourself a grade, what would it be? Please use the Fibonacci Scale. That is to say, give your score as one of the following numbers: 0, 1, 2, 3, 5, 8, or 13. 13 is the highest score possible, 8 is just over half of highest, 1 corresponds to almost, but better than, nothing.

I would assign myself with a grade of a 5 or 8. I think I did a good job and managed to develop a working full stack web application that is hosted online and has a public url. Only minus I would give for my project that it doesn't completely full fill my idea with all the details, like specifying what park has what reservation is not automatically filled based on the user action but instead the

user can fill any text data to the park detail. The reservation page could also be formatted with css to have a personalized look. Maybe if I wouldn’t have had my personal obstacles that affected my available time, I would have been able to make it more fancy and user friendly.