



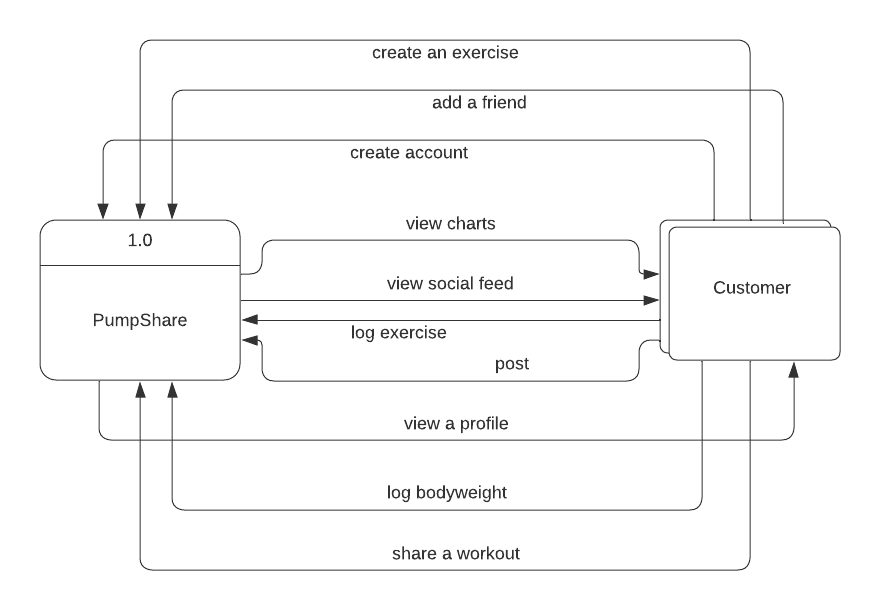
Group 1

Elliot Thapaliya | Frederick Schuck | Jake Blackburn | Luke Simone | Seth Witherspoon

# Project Overview

PumpShare aims to tackle the challenges individuals face in maintaining motivation and tracking progress in their fitness endeavors. With numerous workout platforms available, users often encounter fragmented experiences lacking in simplicity and engagement. PumpShare seeks to bridge this gap by offering a seamless and immersive fitness solution that caters to the diverse needs of its user base. By providing detailed workout logging, social interaction features, progress tracking tools, and collaboration opportunities, PumpShare aims to create a comprehensive platform that not only helps users stay accountable and motivated but also fosters a sense of community within the fitness ecosystem.

Stakeholders in the PumpShare ecosystem include individual users seeking convenient and personalized fitness experiences, fitness leaders looking to engage with their followers and create specialized content, and health and wellness brands seeking to connect with a dedicated fitness audience. Each stakeholder stands to benefit from PumpShare's innovative approach to fitness, whether through enhanced motivation and progress tracking for users, expanded reach and engagement for fitness leaders, or strategic partnership opportunities for health brands. PumpShare's features address the challenges faced by these stakeholders, offering a holistic solution that redefines the fitness experience.

**Context Diagram**

# Architectural Overview

When brainstorming the architectural design for PumpShare, we considered various options. Flask, a Python web framework, was initially considered for the backend. However, after careful consideration, the MERN stack, featuring MongoDB, Express.js, React.js, and Node.js, came up as the optimal choice. This decision was influenced by the popularity of Node.js and React.js frameworks. Node.js offers superior performance and scalability, while React.js's component-based architecture enhances frontend usability and development efficiency. Additionally, using a NoSQL database like MongoDB was preferred for PumpShare's workout logger over PostgreSQL due to its schema flexibility, allowing for easier storage of diverse workout data structures. MongoDB's scalability and ability to handle unstructured data align well with the dynamic nature of fitness data, providing a more adaptable solution for PumpShare's evolving needs.

# Subsystem Architecture

# 

# The PumpShare application is structured into three primary layers: the frontend (client), the backend (server), and the database layer.

# The *frontend layer* (client) is built using React.js and is responsible for rendering the user interface and managing user interactions. It encompasses various components defining the UI's structure and behavior, including those for user authentication, workout logging, social features, and progress tracking. These components interact with the backend via API calls to fetch and display data to users.

# In the *backend layer* (server), data management, business logic, and database communication are handled. This layer follows the Model-View-Controller (MVC) pattern. The Model Layer defines data schemas and interacts directly with MongoDB for CRUD operations and data integrity enforcement. The Controller Layer receives frontend requests, processes them, and orchestrates interactions between the frontend and the database layer. This promotes modularity and helps facilitate communication between the frontend and backend.

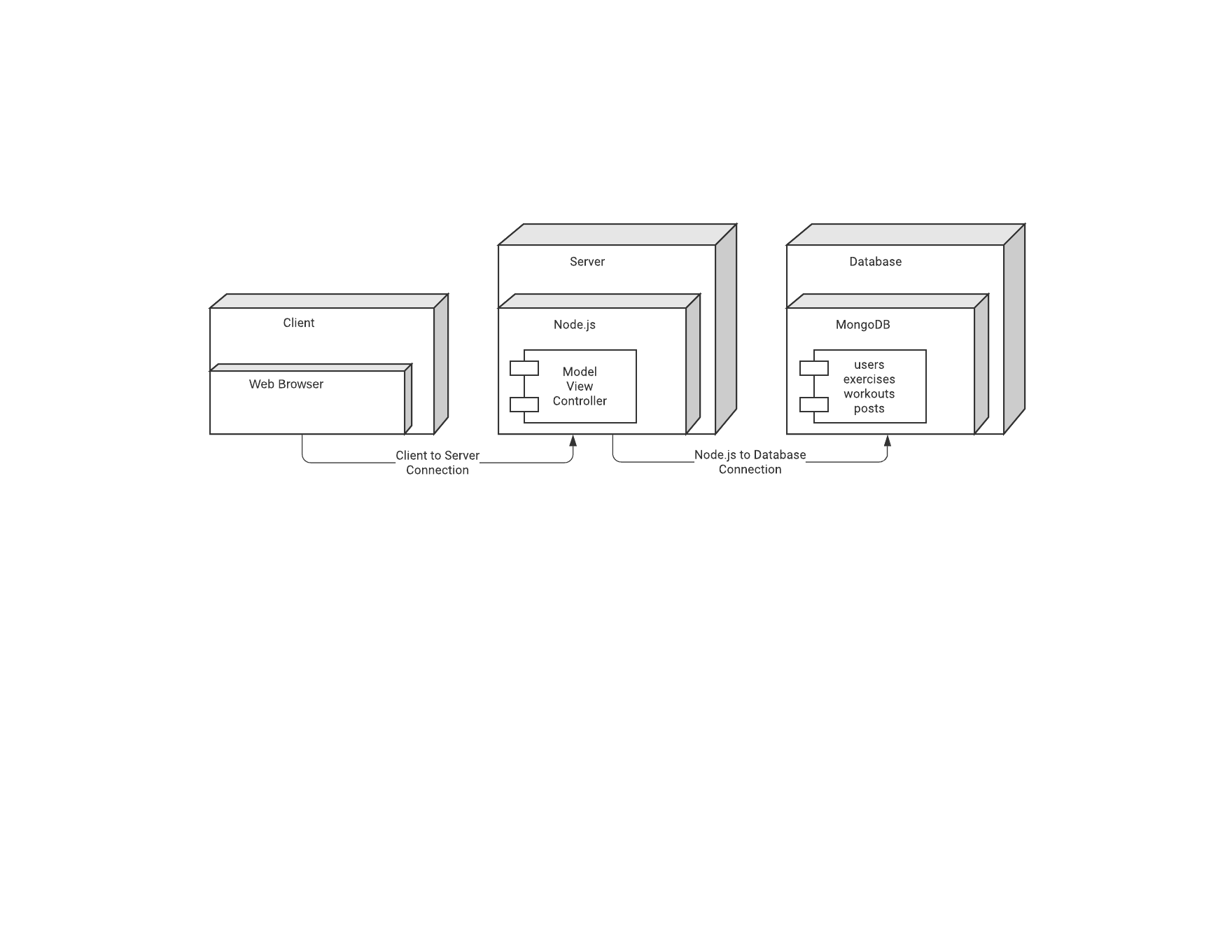
# The *database layer* stores and manages application data in document format. MongoDB's flexibility in handling unstructured data aligns well with PumpShare's dynamic data requirements. The backend's Model Layer interacts directly with MongoDB, performing CRUD operations and managing data storage. This architectural design ensures a clear separation of responsibilities, promotes scalability, and addresses the specific challenges of PumpShare's fitness-oriented application.

# 

# Deployment Architecture

The client represents the web browser where users interact with the frontend of the application. Communication between the client and the server occurs over the HTTP or HTTPS protocols. The server consists of the Node.js runtime environment, which hosts the backend logic of the application, including Model, View, and Controller components following the MVC (Model-View-Controller) pattern. When a user interacts with the frontend, HTTP or HTTPS requests are sent to the Node.js server, which processes these requests and sends back appropriate responses.

MongoDB serves as the backend database, storing collections such as users, exercises, workouts, and posts. Communication between the server and the MongoDB database occurs using MongoDB's native protocol, which is implemented over TCP/IP, the underlying network protocol used for communication between computers on the internet. When the Node.js server needs to perform database operations, it sends requests to the MongoDB database using this protocol. This architecture enables seamless communication between the different components of the system, facilitating the flow of data and interactions within the application.



# Data Model

In this case, MongoDB serves as the backend database for storing application data, leveraging its flexible document-based data model. User profiles, containing attributes like username, email, and timestamps, are stored in a collection named "users," with each document representing a profile. Workout routines, comprising details such as routine name, exercises, and duration, are stored in collections named "workouts" and "exercises," with each document representing a specific routine or exercise. Social network posts, including content, author details, and timestamps, are stored in a collection named "posts," each document representing a single post. Additionally, other application data like user interactions and settings find their place in relevant MongoDB collections, organized based on their structure and usage.

MongoDB's document-based approach allows for efficient storage of diverse structured data within the same database, facilitating scalability, flexibility, and performance. With each collection capable of accommodating documents with varying schema, MongoDB enables seamless management of different types of data, ensuring optimal functionality for PumpShare's fitness application.

# 

# Global Control Flow

PumpShare operates on an event-driven model, allowing users to interact flexibly with the application's features based on their preferences. Users can engage with various functionalities in any order, triggering corresponding actions within the system dynamically.

While PumpShare includes time-dependent features like workout reminders, it doesn't operate as a real-time application with strict time constraints. Instead, time-dependent actions are handled asynchronously, triggered by user interactions or scheduled events.

Concurrency in PumpShare is managed through multiple threads, enhancing system performance by handling concurrent user requests and database operations efficiently. Threads, utilized by components such as server-side controllers and database access modules, are synchronized using asynchronous programming paradigms in Node.js to ensure thread safety and prevent race conditions in shared resources.

# Detailed System Design

For our model, we used the MERN stack, featuring MongoDB, Express.js, React.js, and Node.js. We used Express and Mongodb for the backend and React for controlling all the front-end stuff. The backend component, powered by Node.js and Express.js, forms the backbone of our application, handling server-side logic and routing. MongoDB, a NoSQL database, serves as our persistent data store. We chose a no SQL database for the flexibility it provides with data. React.js builds our frontend with a dynamic user interface allowing us to do very unique work. All of these form to make a MERN stack which allows us to provide a full-stack development environment. We chose this because it provides a comprehensive JavaScript environment which is a scalable and consistent experience that encompasses the entire app development process. It is also great for coders of varying skill levels. It has allowed us to use a single language and a consistent toolset to construct both the front and back ends of the application which reduces development time and complexity.

#### 

# Static view

#### 

#### 

#### 

#### 

Our user table holds the information about to user you see in the table above. A user then can create an exercise with a name, duration, reps, and calories burned and then it will generate an id. After some exercises are made they can be added to a routine that takes them as an array and also adds up the duration and calories burned while also making an id for the routine as well. Finally, a user can make a post like most typical social media apps that will have their username and title with some content and an image if they want.

# Dynamic view

# 

# 

When a user comes to the site they are asked to sign up or log in. If they don’t log in correctly or don't have an account they’ll have to sign up. One in they are led to a dashboard with all the information they need to do anything of the things we’ve talked about before. From the page, they can make activities, view routines, make a post, and view your profile. They can also make, update, and delete workouts to meet their specifications.