Project Milestone 1

Members:
- Derek Williams (Project Leader)
- Calvin Wieland
- Andrew Neal
- Philip Rickey

Project:
- Fitness Club and Gym Management System

Part 1, Exhaustive Functional & Non-Functional Requirements:

Functional Requirements:
- **Users should be able to register as a member of a gym**
  - Because gyms have memberships that are needed in order to use their facility. It is in the functional category because we need to write a function to handle the validation and creation of memberships.
- **Users should be able to manage membership plans**
  - Members should be able to freely create or update their gym membership. It is in the functional category because there should be a function that allows users to update their membership plan.
- **Users should be able to automate payments and track payment history**
  - Because it makes the gym experience easier for the user and they can view what money they've spent on the gym. It's functional because there can be a function that automates payments and can retrieve payment history.
- **Users should be able to view and schedule fitness classes**
  - Users who want to participate in fitness classes should be able to easily do so on the management system. It's functional because a function can handle the retrieving of classes and schedule them for the user.
- **Users should be able to view trainer profiles and their expertise**
  - Users should know who they're getting training from to determine if the trainer is the right fit. It's a function because a function can retrieve trainer information for user viewing.
- **Users should be able to log and view fitness progress**
  - Being able to log and view fitness progress increases the user experience and allows them to more easily achieve their goals. A function will be able to allow users to view their fitness data by database retrieval and let users log their progress by database updating.

- **System should sync data from smart watches or fitness bands**
    - Much of a user's fitness data is collected by watches and fitness bands, so the management system should be able to track this data. A function will be able to collect this data and save it in the system.
- **Admins should be able to manage and schedule fitness classes**
    - A manager needs to have the ability to create classes so that users can sign up for them. It's functional because a function will be able to allow admins to update the gym database, thus creating classes.
- **Admins should be able to manage trainers and their profiles**
    - An admin will need to add trainers and information to their profiles for users to view. It's functional because a function will be able to update the database for creating and updating trainer profiles.
- **Admins should be able to manage member payments**
    - If a member doesn't know how the system works, the admins should be able to create payments for the members. Similarly to users being able to manage payments, a function will be able to access the database to manage member payments.

Non-Functional Requirements:

- **System should be able to handle several hundred users without significant performance impact**
    - It is important that our system can handle a large influx of users without a decrease in performance. This is a non-functional requirement since it describes how well our system should handle multiple users and not how it should handle them.
- **All sensitive user information should be encrypted**
    - Encrypted information is vital to ensuring the security of our users data and maintaining a user, company trust relationship. This is a non-functional requirement since it explains how data should be handled and not what our system will do with it.
- **System should be responsive and fast**
    - A responsive and fast system is important to maintain user satisfaction and productivity. This is a non-functional requirement since it describes how our system should perform on all actions rather than a specific action that it should perform.
- **System should be easy to navigate and user friendly**
    - A simple user interface is important for user experience and satisfaction and should be easy to learn and intuitive. This falls under a non-functional requirement since it describes how our system should appear, now what it does.

- **Multi factor authentication should be implemented to ensure security of user profiles**
    - Multi factor authentication is important to ensure that a user's account is secure and only accessible by them and prevent important information stored, such as credit cards on file, from being compromised due to common cyber attacks. This classifies as a non-functional requirement since it handles how our login function will work rather than what it will do.
- **System should be scalable**
    - Scalability is important to ensure that when our system becomes larger and includes more features and users, that it can continue to operate with optimal performance. This falls under the label of a non-functional requirement as it entails how our system should perform and be designed rather than what it does.
- **System should be reliable under standard conditions**
    - A reliable system ensures that the user can rely on the service working and have the peace of mind that it will operate without any errors or frequent crashes. This is a non-functional requirement since it defines the dependability of the system rather than a particular function that it will perform.
- **System should work on different devices owned by the users such as phones and laptops**
    - Ensuring that the web system works on different devices such as iPhones, Androids, Laptops, and Tablets ensures that the user can access the system from anywhere and ensure satisfaction with the product. This falls under the non-functional requirements field since it describes the systems accessibility rather than a particular function it should perform
- **System should be maintainable**
    - A maintainable system ensures that developers can quickly patch issues that may arise and implement new features easily. This falls under the label of a non-functional requirement since it focuses on how we should design our system and documentation.
- **System should be available at all times (except for routine service)**
    - The system being available at all times except for routine service ensures that users can always access and use the system. This is a non-functional requirement since it does not necessarily say what the system should do rather how it should operate.

Part 2, Tools & Technologies Selection

Editor:

- **Visual Studio Code (VSCode)**

- We chose to use VSCode as our primary editor as it is one of the most widely used editors for web development. It is simple to use, cross platform, and offers an extensive library of add ons which can simplify and improve our development process. It is also well integrated with git and GitHub which is necessary since our team has chosen it as our version control system.

Languages & Frameworks:

- **React.js**
  - We chose React.js because it is the most straightforward one to use. It is also easy to work in as you can write everything using just TypeScript or JavaScript. Installing libraries (such as libraries that enable POST/GET requests to a database or state management) is also very straightforward due to using a package manager such as npm. It also has a ton of documentation and is popular so any problem encountered is easily searchable along with the solutions.

- **TypeScript**
  - We chose to use TypeScript as it adds static typing to JavaScript and works very well with the React.js library. TypeScript is also a good choice as it provides better code scalability compared to JavaScript which is a key component that our application must be. TypeScript is also compiled into JavaScript before it is deployed and run so we can easily catch errors before they are run. This is extremely valuable as it helps ensure that our application code is reliable and secure.

- **PostgreSQL**
  - We chose to use PostgreSQL because it's reliable, flexible, and has a lot of documentation for it. It's also open-source, and has support for all OS since we are all using different OS. It handles both simple and complex queries really well. PostgreSQL supports structured data like traditional databases but also works with semi-structured data using features like JSONB. This makes it versatile for many real-world scenarios. Plus, it's known for being stable, secure, and scalable. By using it, we get practical experience since a large number of companies use psql as their database of choice. Overall, it's the perfect mix of simplicity and power for learning and building real applications.

- **Next.js**
  - We selected Next.js as our web framework due to its powerful hybrid static and server rendering capabilities, which align perfectly with our goal of creating a fast and responsive fitness management platform. With built-in support for page routing, API endpoints, and performance optimizations such as automatic code

splitting and lazy loading, Next.js allows us to deliver a highly scalable and efficient application. Its seamless integration with React.js ensures that our front-end development remains smooth and modular, while features like static generation improve SEO and overall load times. Next.js also supports TypeScript natively, making it an ideal match for our chosen tech stack and enabling maintainability as the project scales.

Version Control Systems:

- **GitHub**
  - Our team decided to use GitHub / got as our primary VCS as it is free, easy to learn and use, and is well implemented with our editor, VSCode. We chose GitHub as the git provider as its web and desktop interfaces are easy to understand and use as well as offers the ability to add collaborators which is a very valuable feature that allows teamwork and collaboration.

Deployment & Hosting Technologies:

- **Vercel**
  - We chose Vercel as our deployment and hosting provider due to its ease of use, performance optimization, and native support for Next.js projects. Vercel simplifies continuous deployment by automatically building and deploying changes from our GitHub repository, ensuring rapid development and feedback cycles. With features like automatic HTTPS, global CDN, and serverless functions, it guarantees that our application remains secure, fast, and globally accessible, enhancing user satisfaction across all device types. Vercel's built-in CI/CD capabilities allow for seamless scalability, making it a reliable and developer-friendly solution for hosting our fitness club and gym management system.