Project Report

Section 1: README

This project introduces a comprehensive Gym Management System designed to optimize the administration and user experience within fitness facilities. The system caters to both users and management, offering functionalities such as Membership Management, Class Scheduling, Attendance Tracking, Equipment maintenance, Invoicing, and Feedback. The technology stack includes Angular for the frontend, Node.js for the backend, and SQL for database management. The user flow encompasses a range of activities from user registration to feedback submission. The data model incorporates entities like User, Trainer, Management, Workout Class, and more, facilitating structured and efficient data management. This Gym Management System aims to enhance operational efficiency, user satisfaction, and overall gym experience.

Section 1.1: Creating and running the application

Prerequisites

- Node.js version 10.24.1 installed.
- npm version 6.14.12 installed.
- Angular CLI 12.2.9 installed.
- MySQL version 8.2.0

Steps for installation for macOS

Using the Terminal, run the following commands:

- brew install nvm
- nvm install 10.24.1
- nvm use 10.24.1 -> Now using node v10.24.1 (npm v6.14.12)
- rm -rf node_modules - To remove the existing node modules
- npm install
- If the server doesn't run using this node version, please install version 12.14.1.
- npm start

Steps for installation for windows

Using the Terminal, run the following commands:

- Please go to the nodeJS link https://nodejs.org/en/download
- Download the latest LTS version - 20.10.0
- Run the .exe file to install Node
- Node and npm will be installed
- Go to the root folder of the project
- Run npm i - to install all the packages
- If the npm i command fail due to any issue, please run npm i –force
- npm start

TO RUN THE WEB APPLICATION

```
## Install node modules
npm i

# ProjectGym
This project was generated with [Angular CLI](https://github.com/angular/angular-cli) version 12.2.9.

## Development server
Run `ng serve` for a dev server. Navigate to `http://localhost:4200/`. The app will automatically reload
if you change any of the source files.

## Code scaffolding
Run `ng generate component component-name` to generate a new component. You can also use `ng
generate directive|pipe|service|class|guard|interface|enum|module`.

## Build
Run `ng build` to build the project. The build artifacts will be stored in the `dist/` directory.

## Running unit tests
Run `ng test` to execute the unit tests via [Karma](https://karma-runner.github.io).

## Running end-to-end tests
Run `ng e2e` to execute the end-to-end tests via a platform of your choice. To use this command, you
need to first add a package that implements end-to-end testing capabilities.

## Further help
To get more help on the Angular CLI use `ng help` or go check out the [Angular CLI Overview and
Command Reference](https://angular.io/cli) page.
```

TO RUN THE NODE.JS BACKEND SERVER

```
## cd backend
The app.js server resides inside the backend folder. So please make sure to go into the folder before
starting the server.

##.env
Please edit the .env file to update the database credentials required to connect to the
‘gymmanagement’ DB.

# .env
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=
DB_DATABASE=gymManagement
PORT=3000


## node server.js
Run `node server.js` to start the backend server which connects with the mySQL database. The
server will run on localhost 3000 port.
```

<u>Section 2: Technical Specifications</u>

The system was built using the following technologies:

1) **Frontend Framework: Angular Framework (version 12.2.0) with Bootstrap**

Angular, a powerful and widely-used frontend framework, was employed for building the user interface of the Gym Management System. The incorporation of Bootstrap, a popular CSS framework, enhanced the system's UI with pre-designed components, responsive layouts, and styling. Angular's component-based architecture, coupled with Bootstrap's features, allowed us to create a dynamic, responsive, and aesthetically pleasing frontend application. The modular structure of Angular facilitated the development of maintainable and scalable frontend code, ensuring a seamless and visually appealing user experience.

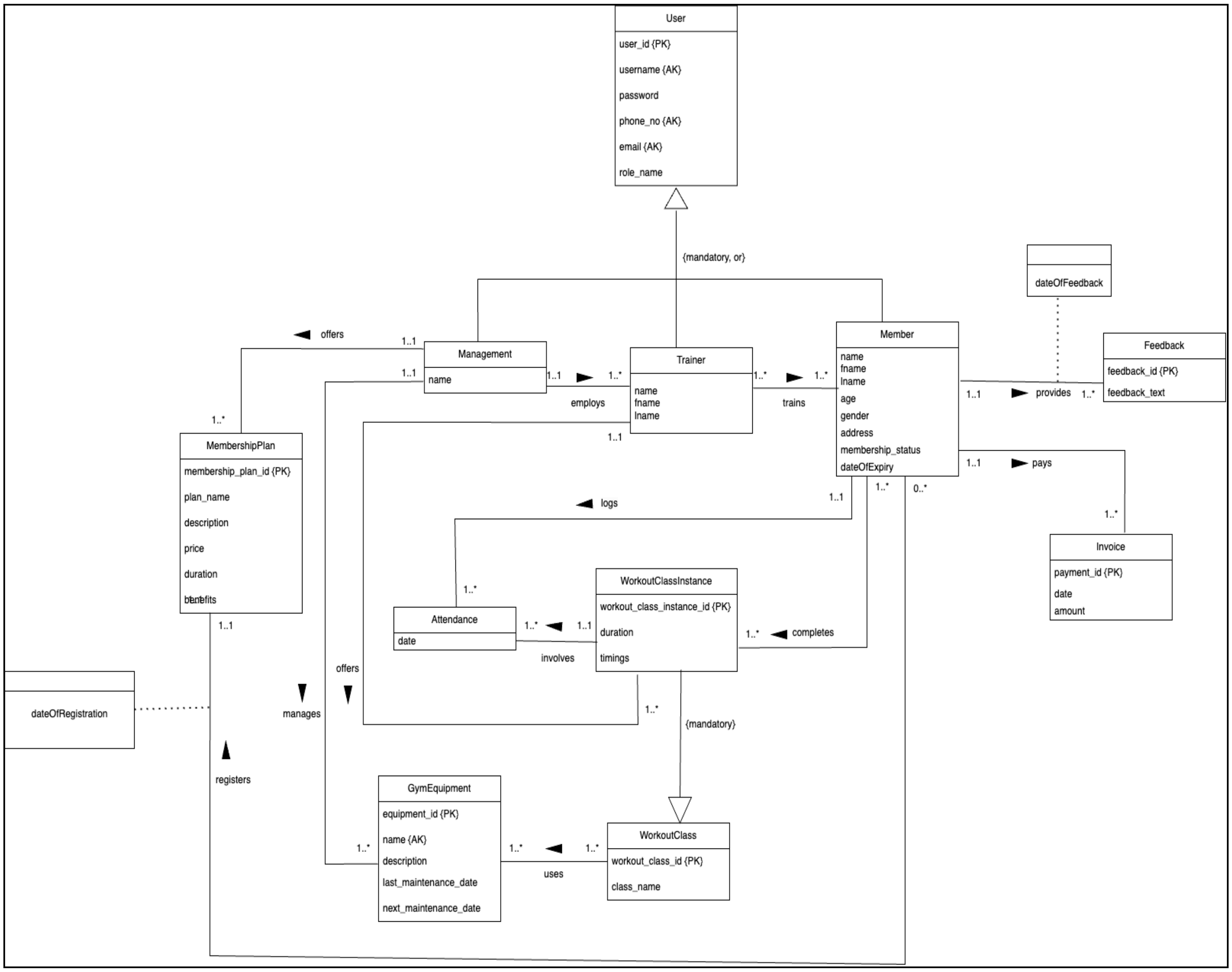
2) Backend Framework: Node.js (version 12.24.1)

Node.js, a popular and efficient backend runtime environment, served as the foundation for the Gym Management System's backend. Leveraging the event-driven, non-blocking I/O model of Node.js, we achieved high performance and responsiveness in handling concurrent user requests.
Express.js, a minimalist and flexible Node.js web application framework, was used to streamline the development of RESTful APIs, enabling smooth communication between the frontend and backend components.

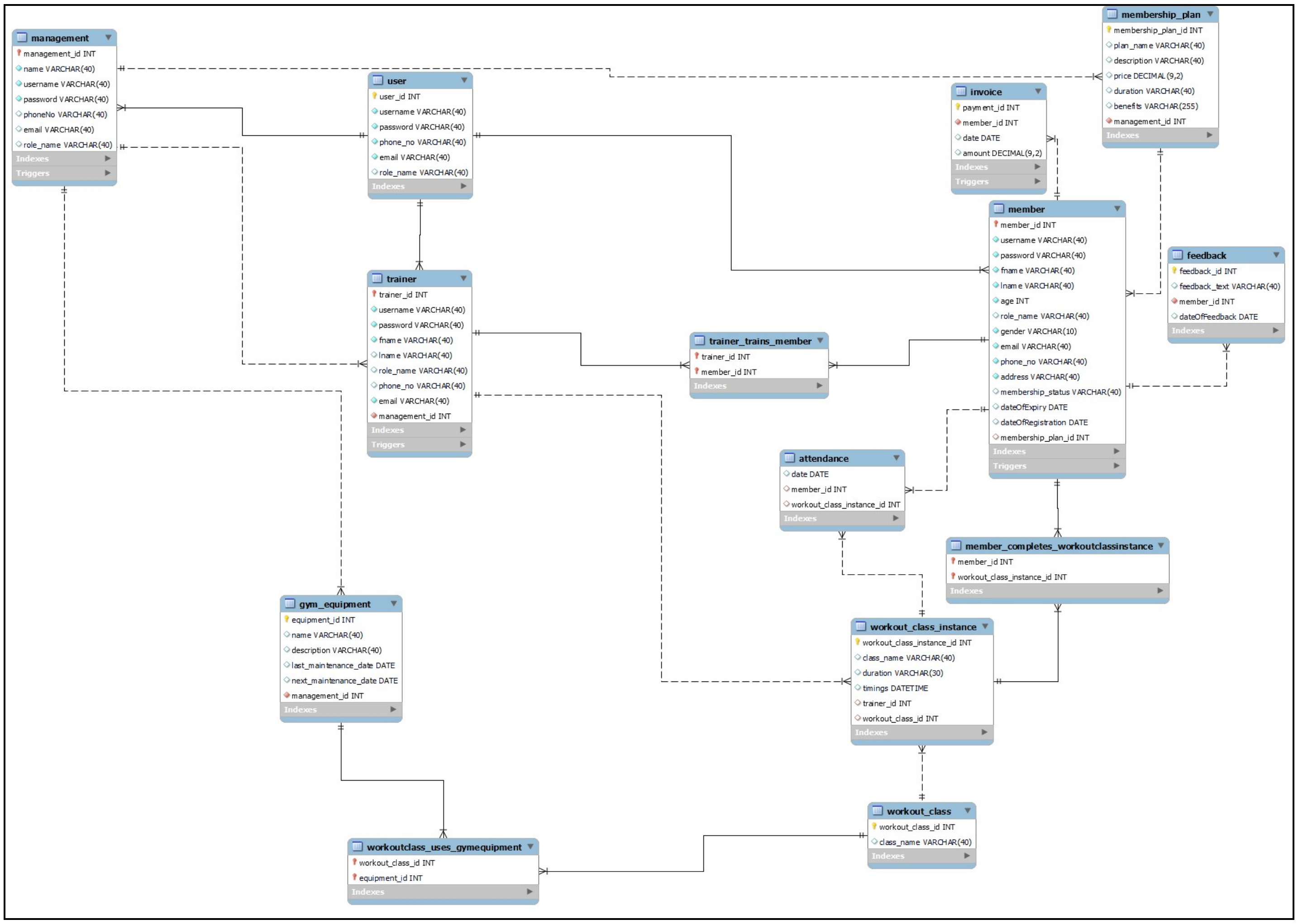
3) Database Management System: SQL (Structured Query Language) (version 8.2.0):

SQL, a standard language for relational database management systems, was utilized to design and manage the database schema for the Gym Management System. We opted for a relational database to ensure data integrity, consistency, and the ability to handle complex queries. By leveraging Angular for the frontend, Node.js for the backend, and SQL for the database, we created a robust, scalable, and performant Gym Management System. This technology stack not only ensures a seamless user experience but also provides a solid foundation for future enhancements and optimizations.

<u>Section 3: UML Diagram</u>



Section 4: Logical Design



<u>**Users**</u>:
Attributes:
(user_id{PK}, username{AK}, email{AK}, password, phone_no {AK}, role_name)

- The Gym Management System utilizes the concept of inheritance through Generalization and Specialization. The primary entity, "User," serves as the generalized entity with shared attributes like user_id, username, password, phone_number, and email. The specialized entities, namely "Member," "Trainer," and "Management," inherit these common attributes while introducing entity-specific details. This design approach ensures a structured and efficient representation of user roles, facilitating organized storage and retrieval of data in the relational database.

<u>**Management:**</u>
Attributes:
(management_id{PK}, username{AK}, name, email{AK}, password, phone_no{AK}, role_name)

Relationships:
(a) Employs Trainers (one-to-many)

- Database Linkage: Implemented through a foreign key in the Trainer table referencing the Management table. This enables a one-to-many relationship, allowing a single management entity to employ multiple trainers.
- Trainer Assignment: The logical design supports efficient assignment and tracking of trainers by the management. It ensures seamless coordination between the management and trainers, facilitating effective oversight of staff allocation.

(b) Manages GymEquipment (one-to-many)

- Data Organization: Employing a one-to-many relationship, this design enables the management to oversee a diverse range of gym equipment. Each equipment record in the Gym Equipment table is linked to a specific management entity, ensuring structured data organization.
- Maintenance Tracking: The logical model supports effective tracking of gym equipment maintenance schedules. It provides a streamlined approach for management to monitor and organize equipment-related tasks.

(c) Offers MembershipPlan (one-to-many)

- Diverse Membership Plans: The logical design enables a one-to-many relationship, allowing a single management entity to offer a variety of membership plans. Each plan in the Membership Plan table is associated with a specific management entity, offering flexibility in plan creation.
- User-Friendly Configuration: It facilitates easy configuration and management of membership plans, including varied pricing, durations, and benefits. The design ensures that the management can efficiently tailor membership offerings to meet the diverse needs of users.

<u>Trainer:</u>

Attributes:
(trainer_id{PK}, username{AK}, email{AK}, name, fname, lname,password, phone_no{AK}, role_name)

Relationships:

(a) Trains Members (many-to-many)

- Junction Table Implementation: The logical design involves a many-to-many relationship between the Trainer and Member entities, implemented through a junction table. This enables a flexible mapping of multiple trainers to multiple members and vice versa, supporting diverse training relationships.

(b) Offers WorkoutClassInstance (one-to-many)

- The logical design represents a one-to-many relationship where a trainer owns or offers multiple workout classes. Each entry in the Trainer table corresponds to several entries in the WorkoutClassInstance table. This design ensures a clear association between trainers and the classes they offer.

<u>Member:</u>

Attributes:

(user_id{PK}, username{AK}, email{AK}, name, fname, lname, age, gender, address, membership_status, dateOfExpiry, password, phone_no{AK}, role_name)

Relationship:

(a) Provides Feedback

- Feedback Storage: The logical design establishes a one-to-many relationship between Member and Feedback, allowing members to provide multiple feedback entries. This supports effective feedback management, enabling improvements based on user suggestions.
- User-Centric Feedback: Members can easily submit feedback through the system, fostering a user-centric approach. The design facilitates the storage and retrieval of feedback data, promoting continuous interaction between members and the management.
  (i) Relationship Attribute: dateOfFeedback

(b) Pays Invoice

- Transaction Tracking: The one-to-many relationship between Member and Invoice enables effective tracking of payment transactions.

(c) Attends WorkoutClassInstance

- Workout Completion Record: The logical design establishes a one-to-many relationship, allowing members to complete multiple workout class instances. This supports the recording and tracking of members' participation in various workout sessions.

(d) Registers for MembershipPlan

- Membership Plan Association: The one-to-many relationship allows members to register for multiple membership plans. This flexibility accommodates varying preferences and fitness needs among members.
- Membership Plan Retrieval: The logical design supports the retrieval of membership plans registered by a specific member. This feature enhances member engagement by providing easy access to information about their chosen plans and associated benefits.
  (i) Relationship Attribute: dateOfRegistration

(e) Logs Attendance

- Logical Design - Attendance Logging: The logical design illustrates a one-to-many relationship where a member logs multiple attendances. Each entry in the Member table corresponds to several entries in the Attendance table.

Invoice:
Attributes:
(payment_id{PK}, date, amount)

Feedback:
Attributes:
(feedback_id{PK}, feedback_text)

WorkoutClass:

Attributes:
(workout_class_id{PK}, class_name)

Relationships:
(a) Uses GymEquipment
- Equipment Utilization: The logical design establishes a many-to-many relationship between WorkoutClass and GymEquipment, allowing a workout class to use multiple pieces of gym equipment. This supports diverse and comprehensive workout routines that involve various equipment.
- Workout Session Planning: Through SQL queries, the design facilitates the retrieval of gym equipment used in a specific workout class. Trainers and management can efficiently plan and organize workout sessions by ensuring the availability of the required equipment.

WorkoutClassInstance:
Attributes:
(workout_class_instance_id{PK}, duration, timings)
Relationships:
(a) Involves Attendance
- Logical Design - Involvement with Attendance: The logical design establishes a one-to-many relationship between the WorkoutClassInstance and Attendance tables. This relationship signifies that each instance of a workout class involves multiple attendance records.

Attendance:
Attributes:
(attendance_id{PK}, date)

GymEquipment:
Attributes:
(equipment_id{PK}, name{AK}, description, last_maintenance_date, next_maintenance_date)

MembershipPlan:
Attributes:
(membership_plan_id{PK}, plan_name, price, duration, benefits)

Section 5: User Flow

Section 5.1: Workflow
The user flow for the Gym Management System involves a series of actions and interactions between the users, trainers and the system. The following is a high-level description of the user flow based on the provided functionalities:

Member Workflow:

(a) Member Registration:
- Users can register by providing necessary details such as name, email, phone, and address.

- During registration, users may choose a membership plan.

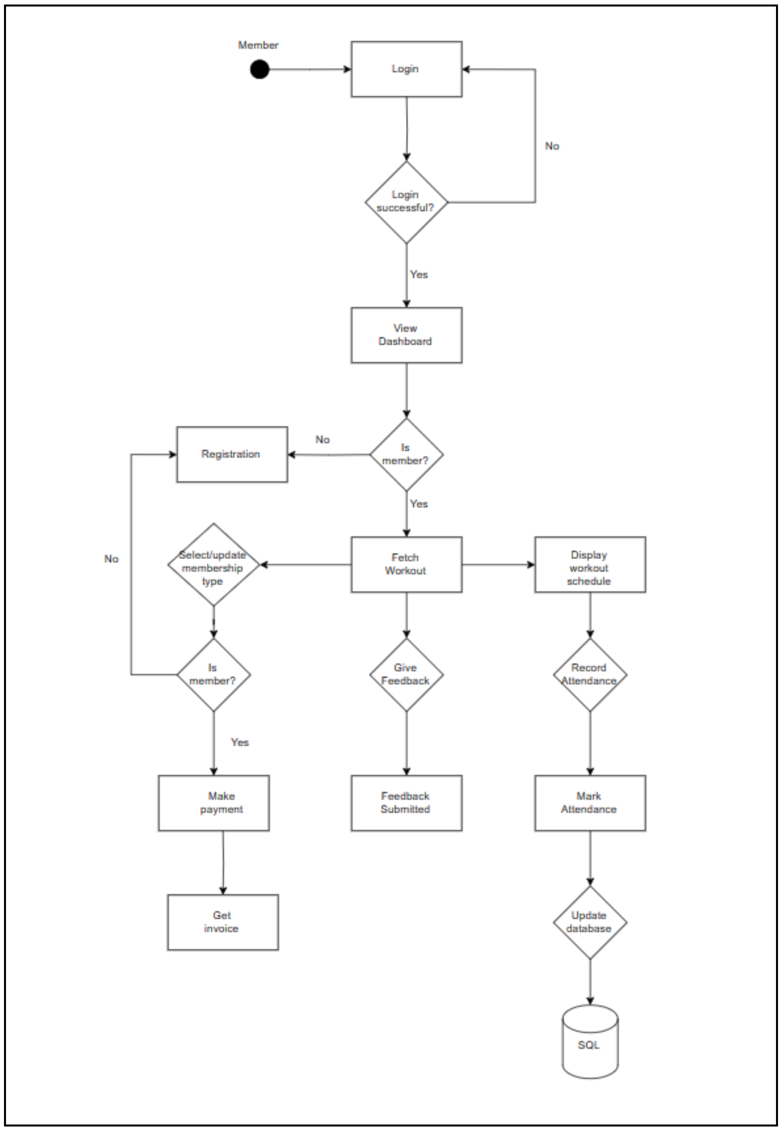
(b) Member Login:
- Registered users can log in by entering their credentials.
- The system validates the user's credentials.
- If valid, the system displays the user's dashboard; otherwise, an error message is shown.

(c) Membership Management:
- Users can browse and select various membership plans offered by the gym.
- Users can sign up for a selected membership plan.

(d) Class Scheduling:
- Users can view fitness class schedules.
- Users can attend fitness classes.

(e) Payment and Invoicing:
- Users can make payments for memberships.
- Users receive invoices for their transactions.

(f) Feedback:
- Members can submit feedback.



Trainer Workflow:

(a) Trainer Login:

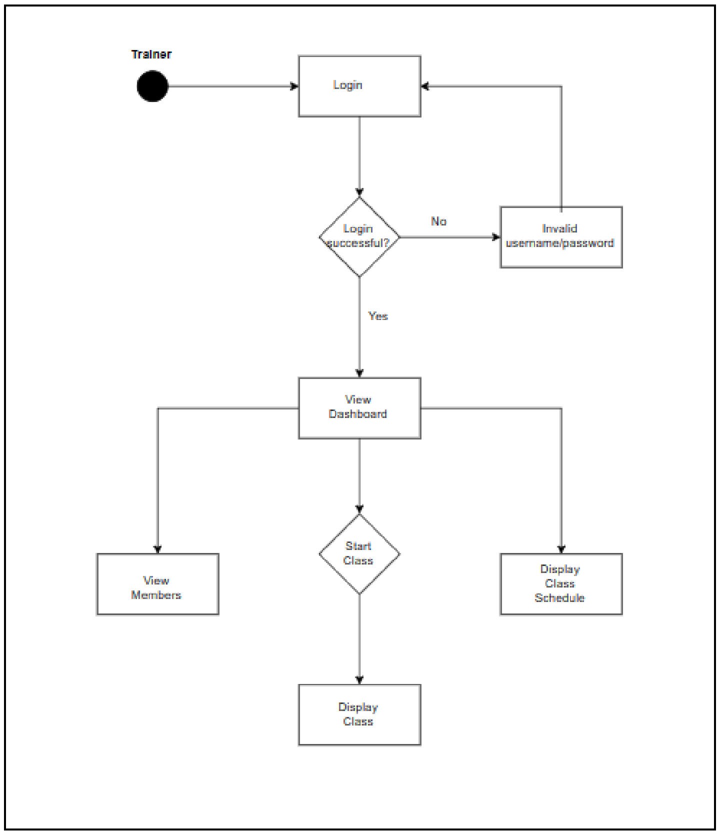
- Trainers can log in by entering their credentials, which is created by the management.
- The system validates the trainer's credentials.
- If valid, the system displays the trainer's dashboard; otherwise, an error message is shown.

(b) View Class Schedule:

- Trainers can view their assigned fitness class schedules, and the number of students in each class.

(c) Train Users:

- Trainers can view the list of users assigned to them.
- Monitor user progress and provide guidance.



Management Workflow:

(a) Membership Plan Management:
- Management can create, modify, and offer various membership plans.
- Management aims to attract and retain members through diverse plans.

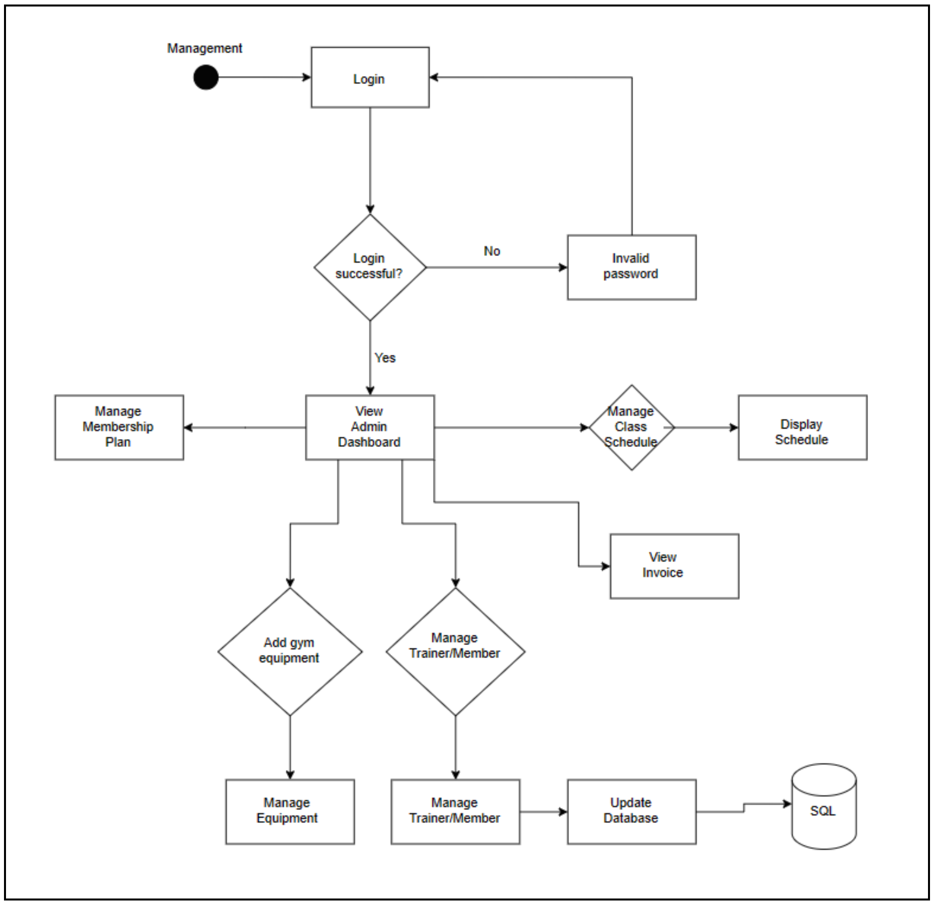
(b) Trainer Management:
- Administrative staff can manage trainers.
- Trainers can be assigned to classes, and their schedules can be tracked.

(c) Equipment and Maintenance:
- Record of gym equipment is maintained.
- Schedule maintenance activities for gym equipment.

(d) View feedback:
- Address feedback submitted by members.

(e) Manage class schedule:
- Management can create and manage class schedules by assigning them to a particular trainer and also adding equipments.

(f) Manage users:
- Management can create and manage users by registering them with the membership plans available.



This user flow ensures that both members and administrators can efficiently interact with the Gym Management System. Members can manage their memberships, give feedback and participate in fitness classes, while administrators have tools to manage memberships, classes, trainers, and overall gym operations. The trainer workflow ensures that trainers have access to the tools they need to effectively manage their classes.
The system aims to enhance the user experience, streamline operations, and maintain an organized and safe gym environment.

Section 5.2: Triggers, Procedures, Functions and Events used

(a) Procedures:

(i) Retrieves members who have taken the 'Basic' membership plan:

```
DELIMITER //
CREATE PROCEDURE GetMembersWithBasicPlan()
BEGIN
   SELECT *
   FROM member
   WHERE membership_plan_id = (
      SELECT membership_plan_id
      FROM membership_plan
      WHERE plan_name = 'Basic'
      LIMIT 1
   );
```

```
END //
DELIMITER ;
```

(ii) Adding a member by the Admin

```
DELIMITER //
CREATE PROCEDURE AddMember(
    IN p_username VARCHAR(40),
    IN p_password VARCHAR(40),
    IN p_fname VARCHAR(40),
    IN p_lname VARCHAR(40),
    IN p_age INT,
    IN p_gender VARCHAR(10),
    IN p_email VARCHAR(40),
    IN p_phone_no VARCHAR(40),
    IN p_address VARCHAR(40),
    IN p_membership_status VARCHAR(40),
    IN p_dateOfRegistration DATE,
    IN p_membership_plan_id INT
)
BEGIN
    DECLARE v_user_id INT;
    -- Insert into user table
    INSERT INTO user (username, password, phone_no, email, role_name)
    VALUES (p_username, p_password, p_phone_no, p_email, 'Member');
    -- Get the generated user_id
    SET v_user_id = LAST_INSERT_ID();
    -- Insert into member table
    INSERT INTO member (
        member_id, username, password, fname, lname, age, role_name,
        gender, email, phone_no, address, membership_status,
        dateOfExpiry, dateOfRegistration, membership_plan_id
    )
    VALUES (
        v_user_id, p_username, p_password, p_fname, p_lname, p_age,
        'Member', p_gender, p_email, p_phone_no, p_address,
        p_membership_status, NULL, p_dateOfRegistration, p_membership_plan_id
    );
END //
DELIMITER ;
```

(iii) Generate Invoice Procedure:

```
DELIMITER //

CREATE PROCEDURE GenerateInvoice(
    IN p_member_id INT,
    IN p_date DATE,
    IN p_amount DECIMAL(9,2)
)
BEGIN
    -- Insert into invoice table
    INSERT INTO invoice (member_id, date, amount)
    VALUES (p_member_id, p_date, p_amount);
END //
DELIMITER ;
```

(iv) Add trainer procedure

```
DELIMITER //
CREATE PROCEDURE AddTrainer(
    IN p_username VARCHAR(40),
    IN p_password VARCHAR(40),
    IN p_fname VARCHAR(40),
    IN p_lname VARCHAR(40),
    IN p_phone_no VARCHAR(40),
    IN p_email VARCHAR(40),
    IN p_management_id INT
)
BEGIN
    DECLARE v_user_id INT;
    -- Insert into user table
    INSERT INTO user (username, password, phone_no, email, role_name)
    VALUES (p_username, p_password, p_phone_no, p_email, 'Trainer');

    -- Get the generated user_id
    SET v_user_id = LAST_INSERT_ID();

    -- Insert into trainer table
    INSERT INTO trainer (
        trainer_id, username, password, fname, lname, role_name,
        phone_no, email, management_id
    )
    VALUES (
        v_user_id, p_username, p_password, p_fname, p_lname,
        'Trainer', p_phone_no, p_email, p_management_id
    );
END //
DELIMITER ;
```

(v) Add membership plan procedure

```
DELIMITER //
CREATE PROCEDURE AddMembershipPlan(
    IN p_plan_name VARCHAR(40),
    IN p_price DECIMAL(9,2),
    IN p_duration VARCHAR(40),
    IN p_benefits VARCHAR(40),
    IN p_management_id INT
)
BEGIN
    -- Insert into membership_plan table
    INSERT INTO membership_plan (plan_name, price, duration, benefits, management_id)
    VALUES (p_plan_name, p_price, p_duration, p_benefits, p_management_id);
END //
DELIMITER ;
```

(vi) Procedure to retrieve information about members, their trainers, the workout classes they attend, and the associated gym equipment:

```
DELIMITER //
CREATE PROCEDURE GetMemberTrainingDetails(IN memberId INT)
BEGIN
    SELECT
        m.member_id,
        m.username AS member_username,
        m.fname AS member_first_name,
        m.lname AS member_last_name,
        t.trainer_id,
        t.username AS trainer_username,
        wc.class_name AS workout_class,
        we.name AS gym_equipment
    FROM
        member m
    JOIN
        trainer_trains_member tt ON m.member_id = tt.member_id
    JOIN
        trainer t ON tt.trainer_id = t.trainer_id
    LEFT JOIN
        workout_class_instance wci ON tt.trainer_id = wci.trainer_id
    LEFT JOIN
        workout_class wc ON wci.workout_class_id = wc.workout_class_id
    LEFT JOIN
        workoutClass_uses_gymEquipment wcuge ON wc.workout_class_id = wcuge.workout_class_id
    LEFT JOIN
        gym_equipment we ON wcuge.equipment_id = we.equipment_id
    WHERE
        m.member_id = memberId;
END //
DELIMITER ;
```

CALL GetMemberTrainingDetails(1);

(b) TRIGGERS:

(i)Trigger to Update Membership Status in Member Table:

```
DELIMITER //
CREATE TRIGGER UpdateMembershipStatus
BEFORE INSERT ON member
FOR EACH ROW
BEGIN
   IF NEW.dateOfExpiry IS NOT NULL AND NEW.dateOfExpiry <= CURDATE() THEN
      SET NEW.membership_status = 'Expired';
   END IF;
END //
DELIMITER ;
```

(ii)Trigger to Update Membership Status on Invoice Generation:

```
DELIMITER //
CREATE TRIGGER UpdateMembershipStatusOnInvoice
AFTER INSERT ON invoice
FOR EACH ROW
BEGIN
   -- Update membership status to 'Active' when a new invoice is generated
   UPDATE member
   SET membership_status = 'Active'
   WHERE member_id = NEW.member_id;
END //
DELIMITER ;
```

(iii)Trigger to set default membership status if no value is inserted

```
DELIMITER //
CREATE TRIGGER SetDefaultMembershipStatus
BEFORE INSERT ON member
FOR EACH ROW
BEGIN
   IF NEW.membership_status IS NULL OR NEW.membership_status = "" THEN
      SET NEW.membership_status = 'InActive'; -- Set default membership status
   END IF;
END //
DELIMITER ;
```

(iv) Trigger to disallow changing username in child tables:

1)Member table

```
DELIMITER //
CREATE TRIGGER member_before_update
BEFORE UPDATE ON member
FOR EACH ROW
BEGIN
```

```
    IF NEW.username != OLD.username THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot modify username in the member table';
    END IF;
END;
//
DELIMITER ;
```

2)Management table

```
DELIMITER //
CREATE TRIGGER management_before_update
BEFORE UPDATE ON management
FOR EACH ROW
BEGIN
    IF NEW.username != OLD.username THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot modify username in the management table';
    END IF;
END;
//
DELIMITER ;
```

3)Trainer table

```
DELIMITER //
CREATE TRIGGER trainer_before_update
BEFORE UPDATE ON trainer
FOR EACH ROW
BEGIN
    IF NEW.username != OLD.username THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot modify username in the trainer table';
    END IF;
END;
//
DELIMITER ;
```

(c) Functions:

(i)Function to Check Membership Expiry Status:

```
DELIMITER //
CREATE FUNCTION IsMembershipExpired(memberId INT)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE expiryDate DATE;
    SELECT dateOfExpiry INTO expiryDate
    FROM member
    WHERE member_id = memberId;
    RETURN expiryDate < CURDATE();
END //
DELIMITER ;
```

(ii)Function to Check Equipment Maintenance Status:

This function checks if gym equipment needs maintenance based on the last maintenance date.

```
DELIMITER //
CREATE FUNCTION NeedsMaintenance(equipmentId INT)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE lastMaintenanceDate DATE;
    SELECT last_maintenance_date INTO lastMaintenanceDate
    FROM gym_equipment
    WHERE equipment_id = equipmentId;
    RETURN lastMaintenanceDate IS NULL OR lastMaintenanceDate < CURDATE() - INTERVAL 30
DAY;
END //
DELIMITER ;
```

(iii) Function to Get Member's Membership Plan Details:

```
DELIMITER //
CREATE FUNCTION GetMemberMembershipPlan(memberId INT)
RETURNS VARCHAR(255)
READS SQL DATA
BEGIN
    DECLARE planDetails VARCHAR(255);
    SELECT CONCAT('Plan: ', mp.plan_name, ', Price: $', mp.price, ', Duration: ', mp.duration)
    INTO planDetails
    FROM member m
    LEFT JOIN membership_plan mp ON m.membership_plan_id = mp.membership_plan_id
    WHERE m.member_id = memberId;
    RETURN IFNULL(planDetails, 'No membership plan');
END //
DELIMITER ;
```

(iv)Function to Get Member's Last Workout Class Instance Details:

```
DELIMITER //
CREATE FUNCTION GetMemberLastWorkoutClassInstance(memberId INT)
RETURNS VARCHAR(255)
READS SQL DATA
BEGIN
    DECLARE workoutClassDetails VARCHAR(255);
    SELECT CONCAT('Class: ', wci.class_name, ', Duration: ', wci.duration, ', Timings: ', wci.timings)
    INTO workoutClassDetails
    FROM member_completes_workoutClassInstance mcw
    INNER JOIN workout_class_instance wci ON mcw.workout_class_instance_id =
wci.workout_class_instance_id
    WHERE mcw.member_id = memberId
    ORDER BY wci.timings DESC
    LIMIT 1;
    RETURN IFNULL(workoutClassDetails, 'No workout class history');
END //
DELIMITER ;
```

(d) EVENTS:

(i) Event to Schedule Maintenance for Gym Equipment:

```
DELIMITER //
CREATE EVENT ScheduleEquipmentMaintenanceEvent
ON SCHEDULE EVERY 7 DAY
DO
BEGIN
    UPDATE gym_equipment
    SET last_maintenance_date = CURDATE(),
        next_maintenance_date = CURDATE() + INTERVAL 30 DAY
    WHERE next_maintenance_date IS NOT NULL AND next_maintenance_date <= CURDATE();
END //
DELIMITER ;
```

Section 6: Lessons Learnt

Section 6.1: Technical expertise gained

Developing the Gym Management System using a modern technology stack involves gaining technical expertise in several areas. Here are the key technical skills and knowledge that can be gained from this project:

(a) Front-end Development with Angular and Bootstrap:

- Component-Based Architecture: Understanding and implementing a frontend application with Angular's component-based architecture.
- Data Binding: Leveraging two-way data binding and other data-binding techniques to ensure seamless communication between the frontend and backend.
- UI Component Libraries: Exploring and utilizing UI component libraries provided by Angular for consistent and responsive user interfaces.

(b) Backend Development with Node.js and Express.js:

- RESTful API Design: Designing and implementing RESTful APIs using Express.js for communication between the frontend and backend.
- Middleware Usage: Understanding and applying middleware for tasks such as request processing, error handling, and authentication.

- Event-Driven and Non-Blocking I/O: Grasping the event-driven, non-blocking I/O model of Node.js for building highly scalable and efficient backend services.

(c) Database Management with SQL:
- Relational Database Concepts: Understanding and implementing concepts such as tables, relationships, and normalization for a relational database.
- Query Optimization: Learning to write efficient SQL queries and optimizing database performance.
- Database Schema Design: Designing a normalized and well-structured database schema based on the requirements of the Gym Management System.

(d) Database Systems (MySQL Workbench):
- Installation and Configuration: Setting up and configuring a relational database management system.

(e) Full-Stack Application Development:
- Integration of Frontend and Backend: Integrating frontend and backend components to create a cohesive full-stack application.
- User Authentication: Implementing secure user authentication mechanisms to protect sensitive data.

(f) Project Management:
- Agile Methodologies: Gaining experience in an Agile development environment, including sprint planning, user story creation, and iterative development.

(g) Documentation:
- Technical Documentation: Creating documentation for the project, including API documentation, database schema documentation, and README files.

(h) Problem Solving and Troubleshooting:
- Debugging Skills: Developing proficiency in debugging code and troubleshooting issues during development.

By working on the Gym Management System, one can acquire a comprehensive set of technical skills applicable to web development, databases, and project management. These skills are valuable for building robust and scalable software solutions in various domains.

Section 6.2: Insights

The Gym Management System provides valuable insights into user behavior, fitness trends, and facility usage. By analyzing data related to attendance, workout classes, and feedback, administrators can make informed decisions to enhance member experience and optimize operational efficiency.

(i) Time Management Insights:
Time management insights are gained through tracking attendance, class instances, and workout classes. The system allows for efficient scheduling of classes, trainers, and equipment usage. Analyzing these data points aids in identifying peak hours, optimizing class schedules, and ensuring the availability of resources when needed.

(ii) Data Domain Insights:

The data domain of fitness and gym management encompasses a diverse set of information, including membership plans, workout details, equipment maintenance, and financial transactions. Insights into this data domain enable administrators to tailor services, create targeted membership plans, and maintain equipment effectively, contributing to the overall success of the gym.

Section 6.3: Realized or contemplated alternative design / approaches to the project

(a) Frontend Framework Alternatives:
While Angular was chosen for its robust features, other frontend frameworks like React or Vue.js could have been considered. The choice depends on factors like team expertise, project requirements, and the specific advantages each framework offers in terms of performance and flexibility.

(b) Backend Framework Alternatives:
Instead of Node.js, other backend frameworks such as Django (Python) or Ruby on Rails (Ruby) could be considered. The decision would hinge on factors like language preference, ecosystem support, and the specific needs of the project.

(c) Database Alternatives:
While SQL databases like MySQL were chosen, NoSQL databases like MongoDB could be contemplated for their scalability and flexibility, especially if the project involves handling large volumes of unstructured data.

(d) Authentication and Authorization Methods:
The project could explore alternative authentication and authorization methods. For instance, implementing OAuth or JWT (JSON Web Tokens) for enhanced security and user authentication.

(e) Alternative UI Libraries:
Besides Bootstrap, other UI libraries like Materialize or Tailwind CSS could be evaluated based on design preferences, ease of integration, and the overall visual aesthetics they offer.

(f) Machine Learning Integration:
Future enhancements might include the integration of machine learning algorithms for personalized workout recommendations, member engagement predictions, or automated feedback analysis.

These alternative design considerations depend on project goals, scalability requirements, and the evolving needs of the users and the fitness industry.

Section 6.4: Document any code not working in this section

The project is currently in a stable state, with all code sections functioning as expected. Rigorous testing has been conducted to ensure robustness and accuracy in implementation. While the current implementation is robust, there is always room for enhancement. Future iterations may focus on optimizing performance, introducing new features, and refining existing functionalities.

Section 7: Future work

Section 7.1: Planned uses of the database

(a) Integration with Wearable Devices:
- Objective: Explore integration with popular wearable fitness devices.
- Justification: Enhance user experience by allowing seamless tracking of workouts and health metrics directly from wearable devices. The database can store and analyze this additional data for personalized fitness recommendations.

(b) Enhanced Analytics and Reporting:
- Objective: Expand analytics capabilities for more in-depth insights.
- Justification: Provide management with comprehensive reports on user engagement, class popularity, and financial performance. This will aid in strategic decision-making and business planning.

(c) Automated Class Recommendations:
- Objective: Develop an algorithm for recommending classes to users.
- Justification: Utilize data on user preferences, workout history, and feedback to suggest personalized class schedules. This can enhance user engagement and encourage participation in diverse fitness programs.

(d) Mobile Application Development:
- Objective: Create a dedicated mobile app for the Gym Management System.
- Justification: Enhance accessibility and convenience for users and trainers. The app can leverage the database for real-time updates, push notifications, and an intuitive mobile-friendly interface.

Section 7.2: Potential areas for added functionality

(a) Online Class Booking:
- Objective: Enable users to book classes online.
- Justification: Provide users with the flexibility to reserve their spot in a class through the system. This requires additional functionality for managing class capacities and real-time updates.

(b) Nutrition Tracking:
- Objective: Integrate a nutrition tracking system.
- Justification: Allow users to log and track their dietary habits. The database can store nutritional information and correlate it with fitness data to provide holistic health insights.

(c) Virtual Fitness Challenges:
- Objective: Implement virtual fitness challenges for users.
- Justification: Engage users with gamified challenges, encouraging friendly competition and collaboration. The database can store challenge details, participant progress, and outcomes.

(d) AI-Powered Personalized Workouts:
- Objective: Explore AI algorithms for generating personalized workout plans.
- Justification: Leverage artificial intelligence to analyze user preferences, fitness levels, and goals. The database can store AI-generated workout plans for users and adapt them over time.

(e) Facial Recognition for Attendance:
- Objective: Integrate facial recognition technology for automated check-ins.

- Justification: Enhance user convenience by replacing traditional check-in methods. The database can store facial recognition data securely.

Section 7.3: No future uses or work can be documented if justification is provided

As of the current project scope, all anticipated features and functionalities have been identified, and the outlined future work aligns with industry trends and user expectations. Continuous improvement and adaptation to emerging technologies are essential for the sustained success of the Gym Management System.

Bonus criteria fulfilled:

1) Additional front end functionality such as a website or a GUI:

The usage of Angular Framework (version 12.2.0) is evident throughout the frontend codebase. Angular components, services, and modules are utilized to create a dynamic and responsive user interface.

2) Complicated schema – user data pull requires multi-joins, or many tables (> 10) due to the complexity of the data domain:

The complicated schema is reflected in SQL queries where multi-joins are necessary to fetch data spanning multiple tables. For example, the following query involves four tables to fetch the workout class.

```
" SELECT * FROM gymManagement.workout_class_instance w
JOIN gymManagement.trainer t on t.trainer_id = w.trainer_id
JOIN gymManagement.workoutclass_uses_gymequipment wg on wg.workout_class_id = w.workout_class_id
JOIN gymManagement.gym_equipment g ON g.equipment_id = wg.equipment_id; "
```

We also have more than 10 tables to represent the complexity of the Gym Management System

3) Application supports multiple user roles:

The application caters to different user roles, such as Member, Trainer, and Management. Each role has distinct functionalities and interactions with the system.