GYM MANAGEMENT SYSTEM

WebApp for Gym Management System

Claudia Giannelli Taccarino

claudia.giannelli2@studio.unibo.it

2024

INTRODUCTION

The purpose of the system developed in this project is to manage the operations of a gym through a distributed system, involving three types of users: administrators, personal trainers, and clients.

Each personal trainer will have access to their own reserved area through which they can view all their courses and appointments booked by clients for private sessions. Additionally, personal trainers will be able to view the list of attendees for their courses or private sessions and access the profiles of individual clients, which will include personal information.

Clients, on the other hand, will access their private area where they can view their profile, including personal data and bookings made for courses and private sessions. They will have the ability to enroll in available courses and sessions. Sessions are booked by consulting the trainers' calendars, checking for available slots, and, if necessary, canceling a booking. In case of a cancellation, the slot will be made available again. It should be noted that clients cannot enroll in a course that has reached its maximum capacity, nor book a private session already assigned to another client.

Administrators have access to tools for creating and managing courses, clients, and personal trainers within the database. They can also modify or delete courses, clients, and trainers from the database.

The overall aim of this project is to facilitate the work of both personal trainers and clients by streamlining the scheduling process and ensuring efficient management of gym activities.

REQUIREMENTS

1. Functional requirements

- **Administrators**

  - Can log in to the system.
  - Can register and save client profiles in the database.
  - Can register and save personal trainer profiles in the database.
  - Can create courses and manage course details.
  - Can modify course details, client profiles, and personal trainer profiles.
  - Can delete courses, client profiles, and personal trainer profiles.

- **Personal Trainers**

  - Can log in to access their personal information and calendar of appointments.
  - Can view the list of courses they are teaching.
  - Can view the list of private sessions booked for them.
  - Can view course details and the list of participants for each course.
  - Can view participant details by clicking on a participant's name in the course or session list
  - Trainers do not have direct permission to modify their data; they must request the administrator to make changes

- **Clients**

  - Can log in to view their personal profile and bookings.
  - Can unsubscribe from enrolled courses or private sessions, making slots available.
  - Can view and enroll in available courses. If a course is full, they cannot enroll.
  - Can book private lessons by selecting an available trainer and time slot.
  - Clients do not have direct permission to modify their data; they must request the administrator to make changes
  - Clients do not have direct permission to modify their data; they must request the administrator to make changes

Design requirements

- The administrator is the initial user and must access the system using predefined credentials (e.g., "admin/admin"), which should be changed for security reasons.
- The administrator stores client and trainer usernames and passwords in the database, but it is recommended that clients and trainers choose their own credentials for better security.
- The system automatically recognizes trainers based on the username and password entered during login, assigning them the appropriate role and access permissions.
- The system automatically recognizes clients based on the username and password entered during login, assigning them the appropriate role and access permissions.
- Each client, trainer, course, and session is identified by a unique identifier.
- The gym manager (administrator) assigns identifiers for clients, trainers, and courses, while sessions are identified by a sequentially increasing number starting from 1.
- Weekly courses are assigned unique IDs, even if they have the same name but occur on different days.

2. Non functional requirements

- **Concurrency Control and Data Consistency:**
    - **Description:** The system must ensure that simultaneous modifications to different data do not cause conflicts or inconsistencies.
    - **Objective:** Ensure that concurrent operations are handled without data loss or corruption, maintaining data integrity.
- **Fault Tolerance and Reliability:**
    - **Description:** The system must be able to function even in the presence of hardware or software failures.
    - **Objective:** Guarantee high availability and rapid data recovery in case of malfunction of a system component.

DESIGN

The system is designed as a web application using modern technologies to ensure efficiency and robustness.

Technologies Used

- **Backend**: Node.js and Express.js
  - **Node.js**: Allows for server-side execution of JavaScript code.
  - **Express.js**: Simplifies the handling of HTTP requests and responses.
- **Frontend**: JavaScript and HTML
  - Together, they create a dynamic and interactive user interface.
- **Database**: MongoDB
  - Offers flexible and scalable storage solutions.

Architectural Approach

The architectural approaches employed include the REST model. The system implements a RESTful CRUD API to provide an intuitive, scalable, and interoperable interface for accessing and manipulating resources. This approach is essential for the following reasons:

1. **Intuitive Interface**:
   - Adhering to REST principles and utilizing standard HTTP methods ensures an intuitive and uniform interface, simplifying interaction with the system for developers and clients.
2. **Scalability**:
   - The stateless nature of RESTful architecture facilitates horizontal scalability, allowing the system to handle increasing load by distributing requests across multiple servers without shared session state.
3. **Interoperability**:
   - Using standard representation formats such as JSON or XML enables seamless interoperability with a wide range of clients and services, promoting integration and compatibility.

4. **Reliability**:
    - The RESTful CRUD API guarantees reliable execution of operations by adhering to consistent and standardized methods, ensuring correct processing of requests and generation of compliant responses.
5. **Maintainability**:
    - Providing a well-documented and structured API enhances maintainability, enabling developers to easily understand and extend functionality, thereby reducing the complexity of maintenance tasks over time.

System Entities:

1. Admin

- **Role:** The system administrator is responsible for managing user profiles and gym courses.
- **Responsibilities:**
    - Manages user access and privileges.
    - Creates, modifies, and deletes client and personal trainer profiles.
    - Creates and manages the gym's course offerings.
- **Interactions:** The admin interacts with client profiles, personal trainer profiles, and courses, having full control to view, modify, update, and delete them.

2. Client

- **Role:** Registered gym users who can book courses and private sessions.
- **Responsibilities:**
    - Book courses and private sessions.
    - Can cancel their enrollment in courses and private sessions.
    - Manage their bookings and view their personal profile.
- **Interactions:** Clients interact with the system to sign up for courses and private sessions. They have access to their profile and information related to their booked activities. Clients cannot modify their profiles without the administrator's assistance.

3. Personal Trainer

- **Role:** Gym instructors responsible for conducting courses and private sessions for clients.
- **Responsibilities:**
  - View and manage their appointments: courses and private lessons.
  - Access the profiles of clients enrolled in their courses and private sessions.
- **Interactions:** Personal trainers interact with clients through booked sessions and courses, viewing participant details. They have access to their personal profile and information regarding the courses they teach and the private sessions booked by clients. Trainers cannot modify their profiles without the administrator's assistance.

4. Course

- **Description:** Courses are scheduled activities offered by the gym that clients can enroll in.
- **Interactions:** Courses are created and managed by the administrator and taught by personal trainers. Clients enroll in courses through the system and can view their enrollments in their personal area. Trainers can view the courses they are teaching in their personal area.

5. Private Session

- **Description:** Private sessions offer personalized services to clients, allowing them to book one-on-one sessions with a personal trainer.
- **Interactions:** Clients book private sessions with a specific personal trainer based on the trainer’s availability. Personal trainers can view the private sessions booked by clients in their personal area.

The following UML diagram shows the entities with their attributes and relationships:

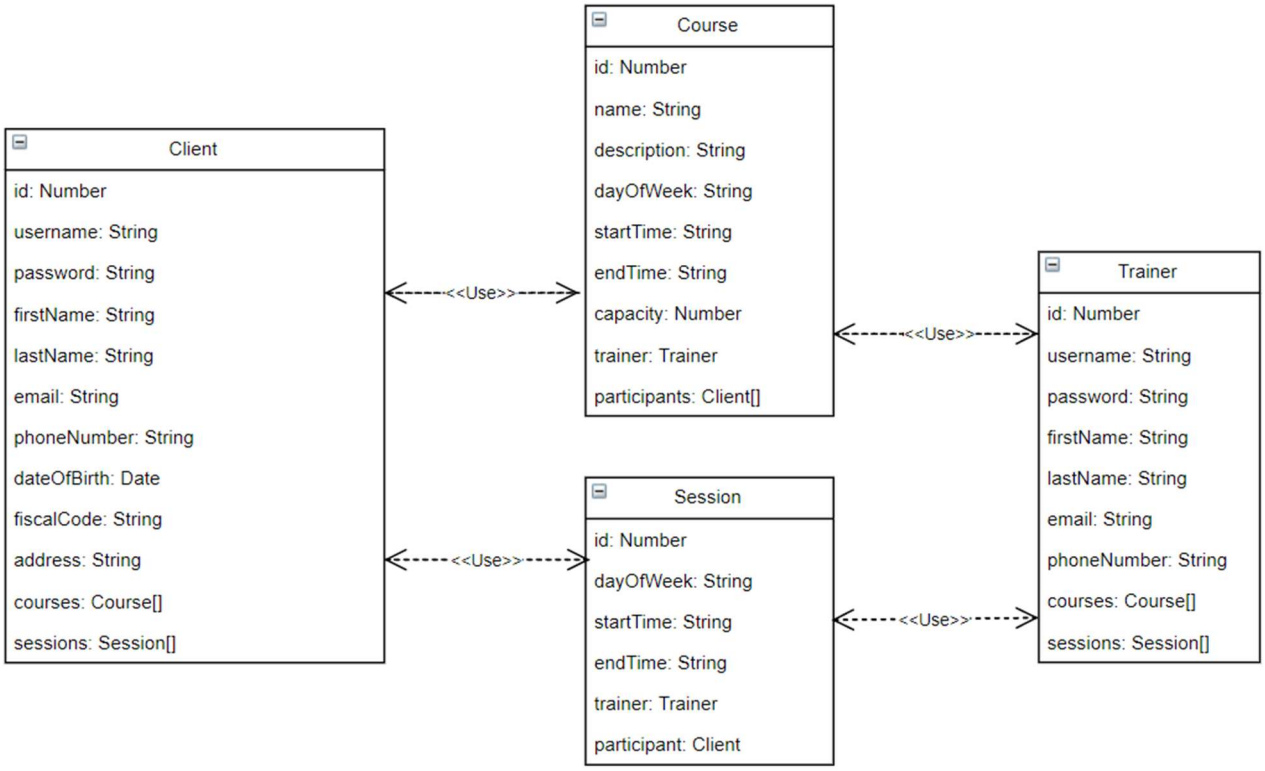


Figure 1: Entities

The following sequence diagram shows the booking of a course:

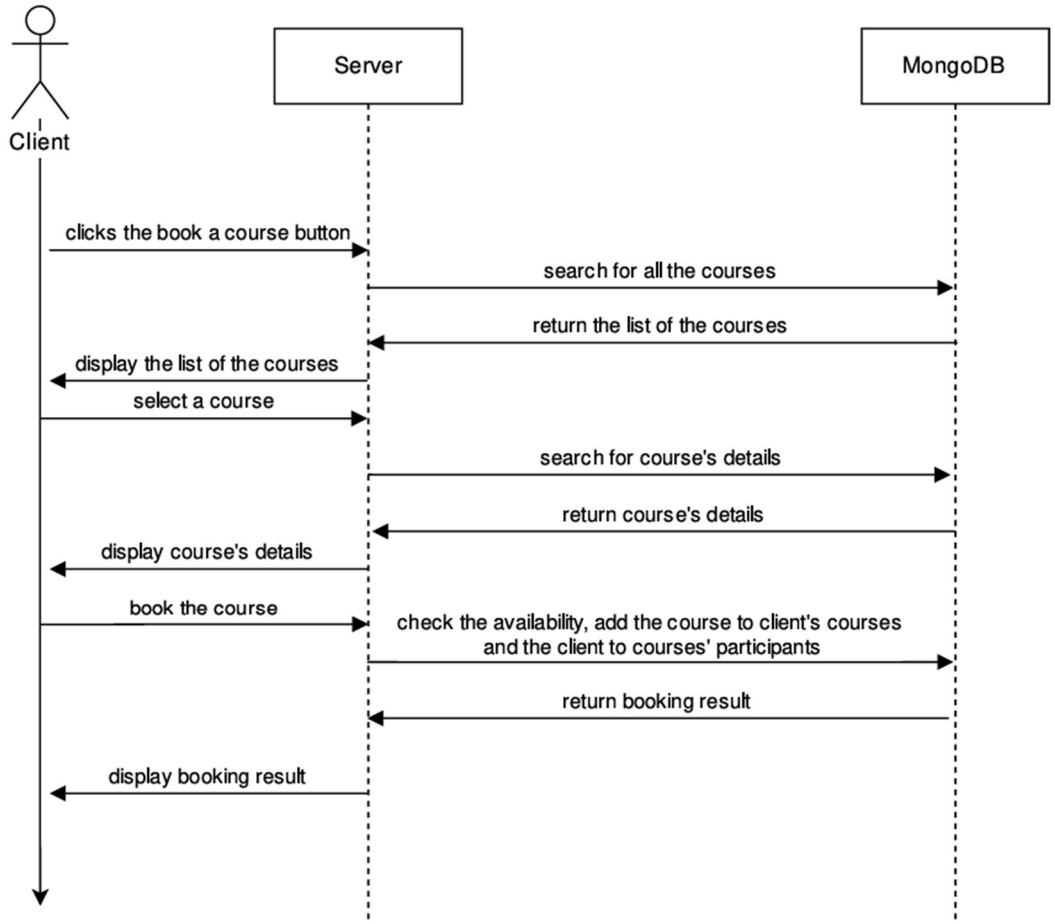


Figure 2: Course booking sequence diagram

The following sequence diagram shows the visualization of trainer's courses details:

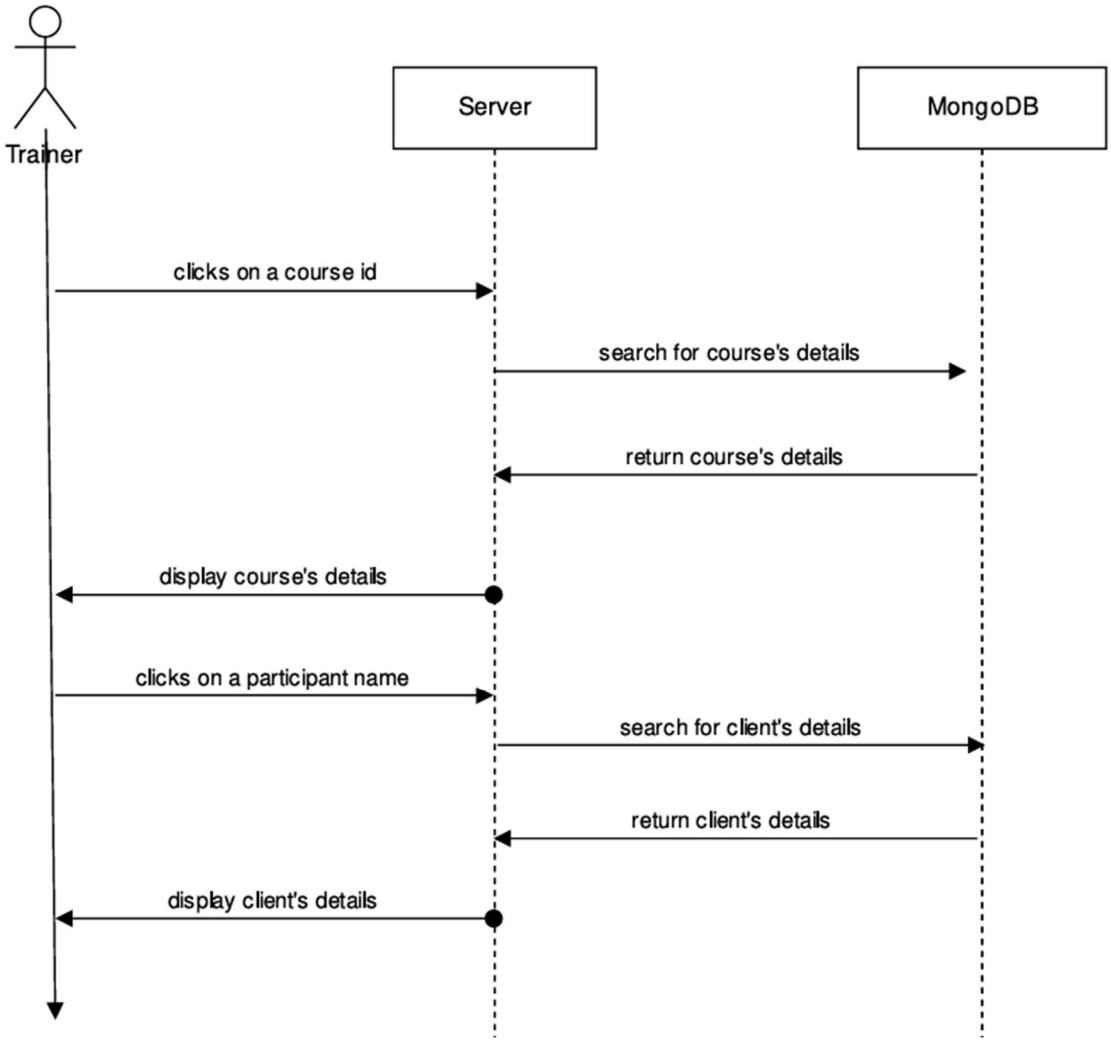


Figure 3: Visualization of trainer's courses details sequence diagram

TECHNOLOGIES USED

In the gym management system, a variety of modern technologies have been employed to ensure efficiency, scalability, and ease of use.

Frontend:

- **HTML and CSS:**
  - **HTML (HyperText Markup Language):** HTML provides the structure by defining elements such as headings, paragraphs, forms, and buttons. This markup language allows the creation of a semantically meaningful and accessible web layout.
  - **CSS (Cascading Style Sheets):** Used alongside HTML to style and lay out web pages, CSS controls the presentation aspect, including layout, colors, fonts, and overall visual appearance. By separating content (HTML) from design (CSS), it ensures a clean and maintainable codebase that can be easily updated or modified.
- **JavaScript:**
  - **Dynamic Interactions:** JavaScript enables interactive elements on the web pages, such as form validations, real-time updates, animations, and event handling. This interactivity improves the user experience by making the web pages more responsive and engaging.
  - **Client-Side Logic:** JavaScript can execute logic directly in the user's browser, reducing server load and providing instant feedback to users without the need for full page reloads.

Backend:

- **Node.js:**
  - **JavaScript Runtime:** Node.js allows JavaScript to be used for server-side scripting, providing a unified language across both the client and server sides. This consistency simplifies development and allows for sharing code between the frontend and backend.

- **Non-Blocking I/O:** Node.js uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient, particularly suitable for data-intensive real-time applications.

- **Express.js:**
  - **Web Application Framework:** Express.js is a minimal and flexible framework for Node.js that provides a robust set of features for web and mobile applications. It simplifies the process of routing, middleware management, and the creation of RESTful APIs.

Database:

- **MongoDB**
  - **Document-Oriented Storage:** MongoDB is a NoSQL database that stores data in flexible, JSON-like documents (BSON format). This document model is inherently different from the traditional row-column model used by relational databases. Each document can have a unique structure, which allows for a more natural representation of hierarchical data and simplifies the mapping between application objects and database entities.

Benefits of using MongoDB in the gym management system

Scalability:

- **Horizontal Scaling:** MongoDB is designed for high performance and scalability. Its distributed architecture supports horizontal scaling through a process called sharding. Sharding involves partitioning data across multiple servers (shards), each acting as an independent database. This approach allows MongoDB to handle large volumes of data and high-throughput operations by distributing the load across multiple nodes.
- **Replica Sets for Fault Tolerance:** MongoDB ensures fault tolerance and high availability using replica sets. A replica set is a group of MongoDB instances that maintain the same dataset. A primary node receives all write operations, while secondary nodes replicate the data from the primary. If the primary node fails, an automatic election process promotes one of the secondary nodes to become the new

primary, ensuring continuous availability of the database. This redundancy safeguards against data loss and downtime, providing robust fault tolerance.

Concurrency and Data Consistency:

- **Atomic Operations and Mutual Exclusion:** MongoDB provides support for multi-document transactions, ensuring ACID (Atomicity, Consistency, Isolation, Durability) properties for operations that span multiple documents. Within a single document, MongoDB guarantees atomic updates, meaning all modifications within a single update operation are applied simultaneously. This atomicity prevents partial updates, maintaining data consistency and integrity.
- **Locks and Isolation:** To ensure mutual exclusion, MongoDB uses locks at the database and collection levels. While earlier versions of MongoDB used a single global lock, more recent versions have implemented more granular locking mechanisms, such as document-level locking, to improve concurrency. These locks ensure that multiple operations do not interfere with each other, preserving data consistency during concurrent accesses.
- **Isolation Levels:** MongoDB provides snapshot isolation for transactions, ensuring that all reads within a transaction see a consistent view of the data as it existed at the start of the transaction. This level of isolation prevents phenomena like dirty reads and non-repeatable reads, which can compromise data integrity in concurrent environments.

Replication and High Availability:

- **Data Redundancy:** By maintaining multiple copies of the data across different nodes in a replica set, MongoDB provides data redundancy. This redundancy ensures that even if one node fails, the data remains accessible from other nodes, enhancing the reliability of the database.
- **Automatic Failover:** The automatic failover mechanism in MongoDB's replica sets ensures that if the primary node goes down, one of the secondary nodes is automatically elected as the new primary. This process is transparent to the application and minimizes downtime, ensuring high availability of the database.

To implement these capabilities effectively in a Node.js environment, developers often use Mongoose. Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js. Mongoose schemas are a fundamental part of this library, defining the structure of the documents within a MongoDB collection. This schema specify the fields, their data types, and any constraints or validations on those fields, providing a formal definition of how documents should appear in MongoDB.

Here is an example of a Mongoose schema used in the gym management system:

```
const mongoose = require('mongoose');

const sessionSchema = new mongoose.Schema({
    id: {
        type: Number,
        required: true
    },
    trainer: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Trainer',
        required: true
    },

    dayOfWeek: {
        type: String,
        enum: ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday', 'monday', 'tuesday',
'wednesday', 'thursday', 'friday', 'saturday', 'sunday'],
        required: true
    },
    startTime: {
        type: String,
        required: true,
        match: /^(09|1[0-8]):00$/ // Regex for the "HH:00"
format and hours between 09 and 18
    },
    endTime: {
        type: String,
        required: true,
        match: /^(1[0-9]):00$/ // Regex for the "HH:00"
format and hours between 10 and 19
    },
    participant: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Client',
```

```
        required: true
    },

});

module.exports = mongoose.model("Session", sessionSchema)
```

Figure 4: Example of a session Mongoose schema

For each entity mentioned above, a Mongoose schema has been implemented.

API:

- **REST API:**
    - **Communication Protocol:** The REST (Representational State Transfer) API facilitates communication between the frontend and backend. It uses standard HTTP methods (GET, POST, PUT, DELETE) for CRUD (Create, Read, Update, Delete) operations.
    - **Stateless Interactions:** Each API call from the client to the server is stateless, meaning it contains all the information needed to process the request. This statelessness simplifies the server design and improves scalability.

Containerization:

- **Docker:**
    - **Consistency Across Environments:** Docker is a leading containerization platform that packages an application and all its dependencies into a container, ensuring that it runs consistently across various environments, whether development, staging, or production. This packaging eliminates the discrepancies between different environments and guarantees that the application behaves the same way everywhere.

Benefits of Using Docker in the gym management system:

Isolation and Scalability:

- **Process Isolation:** Docker containers encapsulate applications in isolated environments, which includes their dependencies, libraries, and configuration files. This isolation helps manage dependencies and prevents conflicts between applications running on the same host. Each container has its own filesystem, network stack, and process space, ensuring that they do not interfere with one another.
- **Resource Management:** Docker provides tools to manage and allocate resources like CPU and memory to containers, ensuring that no single container consumes too many resources, leading to predictable performance and efficient utilization.
- **Scalability and High Availability:** Docker enables horizontal scaling by allowing multiple instances of containers to run concurrently. This capability ensures high availability and load balancing. Orchestration tools like Docker Swarm or Kubernetes can manage container clusters, scaling them up or down based on demand.
- **Facilitates Distributed Environment:** Docker enables the creation of a distributed environment by encapsulating each component (MongoDB replica set, MongoExpress, application replicas) within separate containers. This architecture supports scalability and fault tolerance by distributing workload and isolating resources effectively.
- **Fault Tolerance with MongoDB Replica Set:** The MongoDB replica set configured within Docker ensures fault tolerance by maintaining multiple copies of data across different nodes. If one node fails, others seamlessly take over, ensuring continuous availability and data integrity. This redundancy minimizes the risk of data loss and downtime.

Docker in the gym management system:

- **Replica Set with MongoDB:** In my application, Docker is used to start three MongoDB databases, each in its own container, forming a replica set. This setup enhances fault tolerance and data redundancy. A MongoDB replica set consists of a primary node that handles all write operations and secondary nodes that replicate the primary's data. If the primary fails, one of the secondaries is promoted to primary, ensuring continuous availability and data integrity. Docker simplifies the configuration

and management of these instances, providing isolated environments for each MongoDB instance.

- **MongoExpress:** Alongside the MongoDB replica set, a MongoExpress container is deployed. MongoExpress is a web-based MongoDB administration interface that provides a user-friendly way to interact with MongoDB databases. It allows users to perform CRUD operations, visualize data, and manage collections and databases through a web browser. Running MongoExpress in a Docker container ensures that the interface is easily accessible and consistently configured across environments.
- **Application Replicas:** Additionally, Docker is used to run two replicas of the main application. These application replicas ensure high availability and load balancing. By running multiple instances of the application, Docker distributes incoming traffic among them, which enhances performance and reliability. If one instance fails, the others continue to handle requests, providing uninterrupted service.

Here is the Docker Compose file used for setting up the application:

```
version: '3.8'
services:
  web:
    image: applicazione
    build:
      context: ./applicazione
      dockerfile: Dockerfile
    restart: always
    ports:
      - "3000:3000"
    environment:
      NODE_ENV: development
      MONGO_HOST: mongodb
      MONGO_PORT: 27017
      DB_URI:
mongodb://mongodb1:27017,mongodb2:27017,mongodb3:27017/gym?re
plicaSet=rs0
    depends_on:
      - mongodb1
    deploy:
      replicas: 2
```

```
  mongodb1:
    image: mongo
    restart: always
    volumes:
     - mongodata1:/data/db
    ports:
     - "27017:27017"

  mongodb2:
    image: mongo
    restart: always
    volumes:
      - mongodata2:/data/db
    ports:
      - "27018:27017"

  mongodb3:
    image: mongo
    restart: always
        volumes:
      - mongodata3:/data/db

    ports:
      - "27019:27017"

  mongoexpress:
    image: mongo-express
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: admin
      ME_CONFIG_MONGODB_ADMINPASSWORD: pass
      ME_CONFIG_MONGODB_URL:
mongodb://mongodb1:27017,mongodb2:27017,mongodb3:27017/gym?re
plicaSet=rs0
    restart: always
    ports:
      - "8081:8081"

volumes:
  mongodata1:
  mongodata2:
  mongodata3:
```

Figure 5: docker-compose.yaml

TESTING

The application's functionality and performance underwent rigorous testing across major web browsers, ensuring responsiveness and reliability. Tests were conducted on Chrome, Firefox, Edge, and Safari to evaluate the system's performance under diverse conditions.

Within the application's Node.js environment, routes are defined as API endpoints managed through the Express framework. Each route serves as an entry point through which HTTP requests interact with the application's features. These APIs were thoroughly tested using the desktop application "Postman". Postman provides an intuitive graphical interface for simulating various types of HTTP requests to the endpoints defined by the routes. For each route, different request types (GET, POST, etc.) can be configured and sent to the server to validate the API's behavior effectively.

Here is an example of routes used in the gym management system:

```
const express = require('express')
const router = express.Router()
const API = require('../controller/clientApi')

// in the path, before these, there must be /clients
router.get("/", API.fetchAllClients)
router.get("/username/:username", API.fetchClientByUsername)
router.post("/", API.createClient)
router.post("/update", API.updateClient)
router.post("/delete", API.deleteClient)
router.get("/courses/:username", API.fetchAllClientCourses)
router.get("/sessions/:username", API.fetchAllClientSessions)
router.get("/courses/delete/:username/:id", API.deleteClientCourse)
router.get("/sessions/delete/:username/:id", API.deleteClientSession)
router.get("/courses/add/:username/:id", API.addClientCourseById)
router.get("/sessions/add/:username/:id", API.addClientSessionById)
router.get("/sessions/:username/:id", API.checkClientSessions)
router.get("/id/:username", API.fetchClient_IdByUsername)
router.get("/checkId/:id", API.isClientIdPresent)


module.exports = router
```

Figure 6: clients' routes

DEPLOYMENT STEPS WITH DOCKER

1. Start Docker: Open Docker Desktop and ensure the Docker daemon is running.

2. **Initialize Docker Swarm:** Open Docker Desktop and ensure Docker daemon is running.

   Write in the command prompt:
   docker swarm init

   Note: Docker Swarm is a built-in orchestration tool in Docker that allows you to manage a cluster of Docker engines, called nodes. It enables you to deploy and manage services across multiple Docker hosts, providing built-in scaling, load balancing, and fault-tolerant features. This ensures applications can handle increased load or recover from node failures without downtime.

3. **Managing Containers and Volumes:**
   - **Persistent Data:** MongoDB replica set data volumes (mongodata1, mongodata2, mongodata3) are persisted across container restarts, ensuring that the replica set configuration does not need to be recreated each time.

     The volumes for the persistent data of the MongoDB replica set are created automatically through the docker-compose.yaml file. It is not necessary to create them manually.

4. **Building Application Image:** Before deployment, build the application image named applicazione using Docker.

   Write in the command prompt:
   docker image build -t applicazione .

5. **Deploy Application Stack:** Navigate to your project directory containing docker-compose.yaml and deploy the stack.

   Write in the command prompt:

   docker stack deploy -c ./docker-compose.yaml stackName

   Note: When deploying with Docker Compose in Swarm mode (docker stack deploy), services defined in the docker-compose.yaml file are orchestrated across the Swarm cluster. Stack is a group of services (or applications) that are managed together as a logical unit.

6. **Configuring the Replica Set**
   - After deploying the stack, you can proceed to configure the replica set. First, list the running containers with:

     docker ps

   - Access one of the MongoDB containers using the command:

     docker exec -it <container_name> mongosh

   - Initialize the replica set with the following command:

     rs.initiate({_id: "rs0", members: [{_id: 0, host: "mongodb1"}, {_id: 1, host: "mongodb2"}, {_id: 2, host: "mongodb3"}]})

   - Check the status of the replica set by running:

     rs.status()

7. **Accessing the Application and Database:**
   - Open a web browser and navigate to:

    - localhost:3000 to access the initial page of the application.
    - localhost:8081 to access the MongoDB database via MongoExpress (username: admin, password: pass).

- If making changes to the application, rebuild the Docker image (docker image build) to reflect updates in the deployed containers.
- The MongoDB replica set does not need to be recreated each time Docker system starts, unless Docker volumes associated (such as mongodata1, mongodata2, mongodata3) are deleted or removed.

USAGE EXAMPLES

Once the application is launched, users are directed to the login page. The manager accesses the admin area using the credentials username: admin and password: admin, where they can create, update, and delete clients, trainers, and courses. Clients and trainers, once registered by the admin, log in using their respective credentials to access their own areas—the client area and the trainer area, respectively.

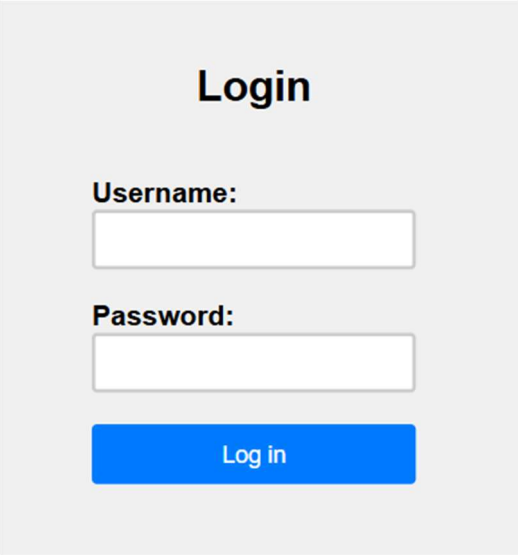


Figure 7: Login page

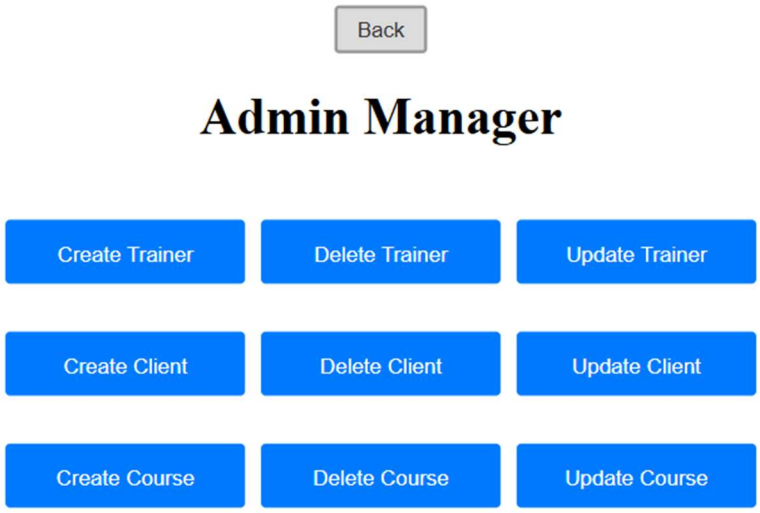


Figure 8: Admin area

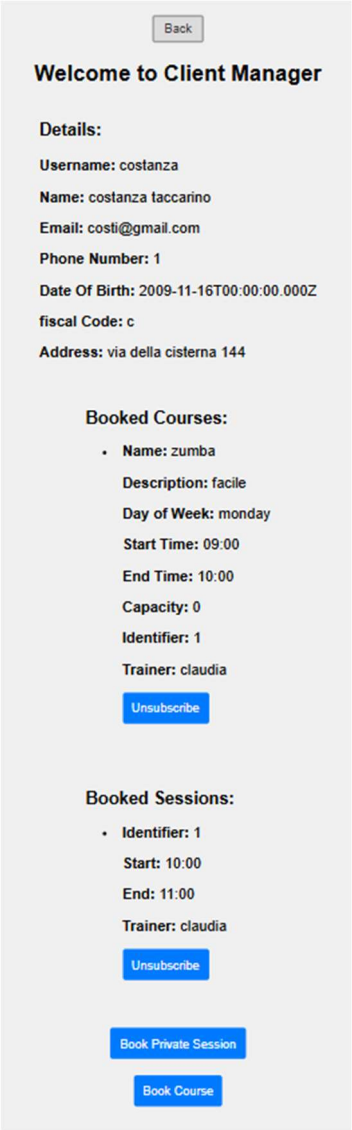


Figure 9: Client area

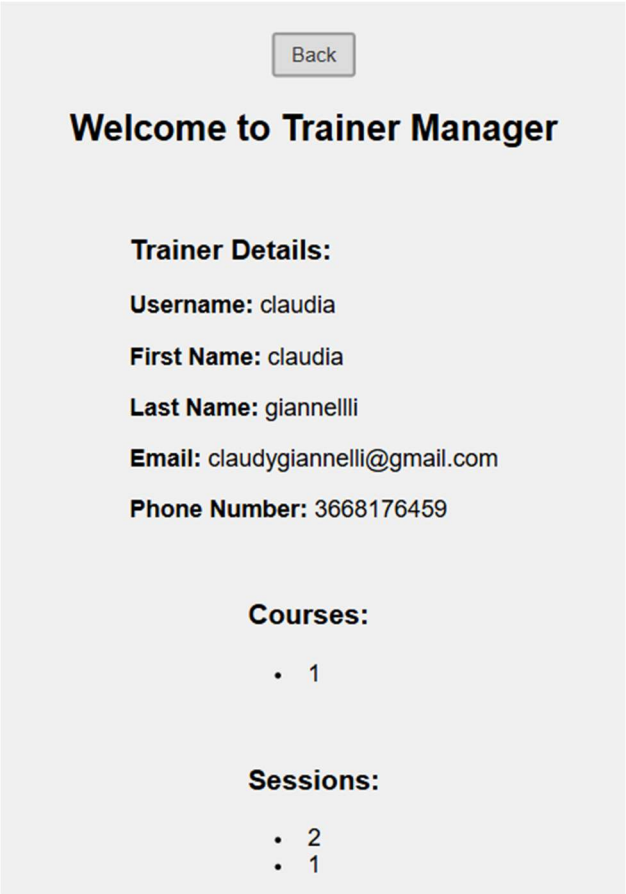
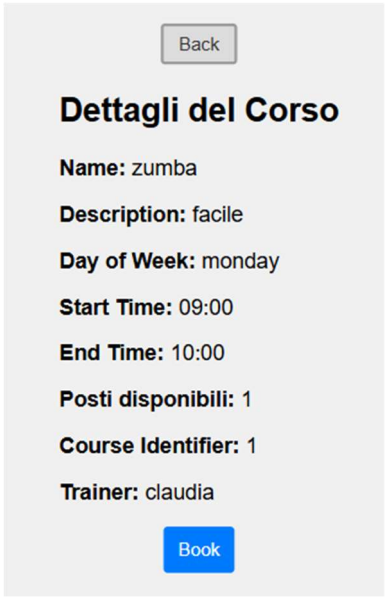


Figure 10: Trainer area

In their client area, clients can view their personal data along with the IDs of booked courses and private sessions. They can book courses or private sessions by clicking on the corresponding buttons. Clicking "Book Course" displays a list of all available courses, while clicking "Book Private Session" shows a list of all trainers. Selecting a course allows clients to view details and book it by clicking the "Book" button (they can later cancel by clicking the "Unbook" button associated with the course in their client area). Selecting a trainer enables clients to choose an available time slot and book the session by clicking the "Book" button (they can later cancel by clicking the "Unbook" button associated with the session in their client area).



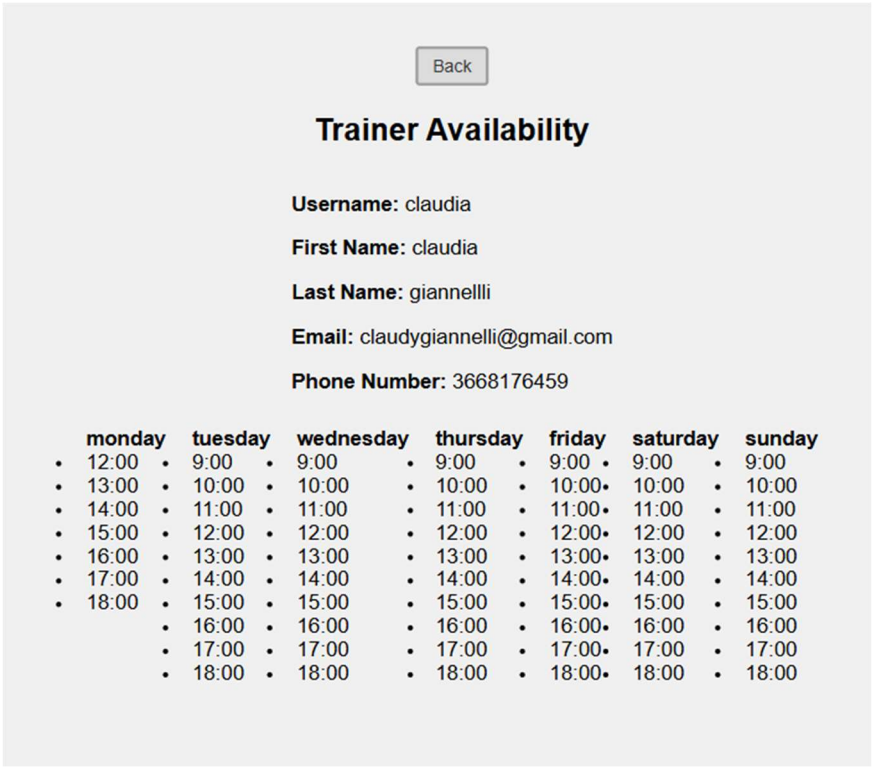


Figure 11: Book courses page Figure 12: Book sessions page

In their trainer area, trainers can access their personal data along with the IDs of courses assigned to them and booked sessions. Clicking on a course ID allows trainers to view detailed information about the course, including the names of participants. By clicking on a participant's name, trainers can navigate to a page displaying detailed information about that participant. Similarly, by clicking on session IDs, trainers can view detailed information about the sessions, and clicking on the participant's name allows them to see the data of the client who booked the session.

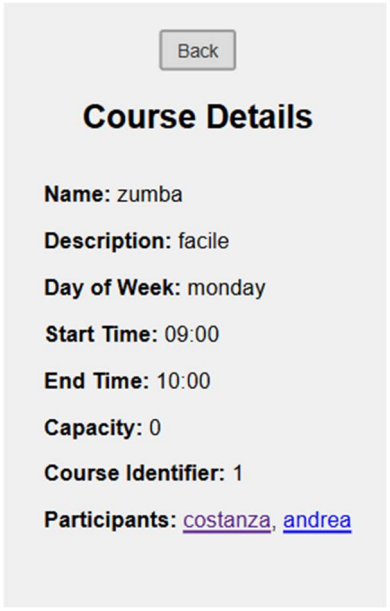


Figure 13: Course details

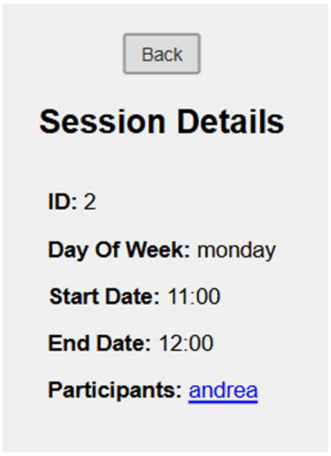


Figure 14: Session details

CONCLUSIONS

The gym management project has proven to be a tangible example of how implementing modern technologies like Docker and MongoDB can significantly enhance operational efficiency and reliability of a complex system. Docker's usage facilitated the creation of a distributed and highly scalable environment, ensuring continuous availability of the application even during hardware failures or sudden spikes in load.

Configuring a MongoDB replica set provided a robust fault tolerance solution, ensuring data redundancy and crucial operational continuity for a distributed system. Integration of tools such as MongoExpress greatly simplified database administration, enabling efficient management of collections and CRUD operations through an intuitive web interface.

Looking forward, the system is designed to evolve further. Upon request from the gym managers, enhancements can be implemented to allow clients and personal trainers to independently manage their data in the database without administrative intervention. Additionally, it will be possible to schedule courses and private sessions with more flexible timings, no longer restricted to hourly durations. Advanced validation checks will be implemented for data entered via forms during client, trainer, and course creation and modification. Furthermore, the system will introduce the management of variable weekly course schedules, departing from the current model of weekly repetitive courses. Advanced cybersecurity techniques will also be adopted to ensure passwords are not stored in plaintext in the database, thereby enhancing overall system security.

In conclusion the project is a reliable solution for gym management designed to adapt to evolving needs in the dynamic context of distributed systems.