SRI KRISHNA COLLEGE OF TECHNOLOGY An
Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042

GYM TRAINER AND PROGRESS TRACKER APPLICATION

Software Design Pattern

A PROJECT REPORT

Submitted by

KRISHNA PRAJVIN P: 727822TUCS061

in partial fulfilment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

Computer Science & Engineering

JULY - 2024

GYM TRAINER AND PROGRESS TRACKER APPLICATION

INTRODUCTION:

In today's digital age, fitness enthusiasts of all levels are increasingly turning to electronic devices to track their workouts and monitor their progress. This trend necessitates the adaptation of fitness resources to digital formats. Our project aims to meet this need by developing a comprehensive gym tracking application that not only replicates the depth and effectiveness of traditional gym sessions but also incorporates interactive elements to enhance the fitness experience.

The development of this gym tracking application involves several critical stages, including requirement analysis, design, implementation, and testing. Each stage is essential in ensuring that the final product is both user-friendly and effective in helping users achieve their fitness goals. Our goal is to create an application that is intuitive for users of all fitness levels, yet robust and scalable enough to accommodate a wide range of workout plans and interactive features we plan to include. This project seeks to provide a dynamic and engaging fitness environment, making quality training and progress tracking accessible to all in the digital era.

.

OBJECTIVE:

The primary objective of a gym tracking application is to provide an optimized platform for fitness engagement and physical well-being, efficiently supporting personal and athletic growth. This involves prioritizing the most relevant and high-quality workout programs, ensuring they align with users' fitness goals and schedules. The application emphasizes efficiency by leveraging advanced technologies and intuitive user interfaces, enabling users to maximize their workout experience in a time-efficient manner without compromising on exercise quality. Organization is key, with structured workout plans and clear timelines to guide users through their fitness journey smoothly. The platform also aims to reduce stress by offering flexible workout schedules and a

variety of resources, allowing users to manage their exercise routines effectively. Productivity is enhanced by breaking down fitness programs into manageable units with clear instructions, preventing procrastination and keeping users motivated. The gym tracking application supports work-life balance by accommodating diverse schedules and providing customizable workout plans, enabling users to integrate fitness seamlessly into their daily lives. Continuous improvement is a core principle, with regular updates to the application's features and content based on user feedback and fitness trends, ensuring an ever-evolving and effective fitness experience.

BACKEND SYSTEM SPECIFICATION

In this chapter, the content discusses the software employed for constructing the website. This chapter provides a brief description of the software utilized in the project.

1.1 POSTGRESQL

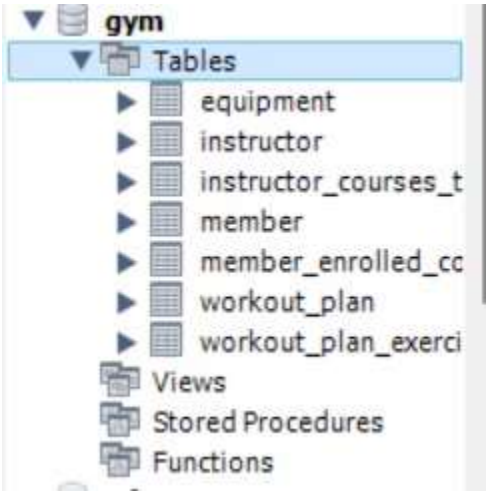

Fig1.1 MySQL

Local storage is a type of web storage for storing data on the client side of a web browser. It allows websites to store data on a user's computer, which can then be accessed by the website again when the user returns. Local storage is a more secure alternative to cookies because it allows websites to store data without having to send it back and forth with each request. It is similar to a database table in that it stores data in columns and rows, except that local storage stores the data in the browser rather than in a database.

Local storage is often used to store user information such as preferences and settings, or o store data that is not meant to be shared with other websitesIt is also used to cache data to improve the performance of a website. Local storage is supported by all modern web browsers ,including chrome, Firefox ,Safari, and Edge. It is accessible through the browser's JavaScriptAPI. Local storage is a powerful tool for websites to store data on the client side. It is secure, efficient ,and can be used to store data that does not need to be shared with other websites.

Local Storage is a great way to improve the performance of a website by caching data. Local storage in web browsers allows website data to be stored locally on the user's computer. It is a way of persistently storing data on the client side, which is not sent to the server with each request. This allows users to store data such as preferences, login information, and form data without needing to send it to a server.

It is typically stored in a browser's cookie file, but it can also be stored in other locations such as HTML5 Local Storage and Indexed. The data stored in local storage is persistent and can be accessed by the website even if the user closes the browser or navigates to another page. It is a great way for websites to store user-specific data, as it is secure, reliable, and fast. It is also a great way for developers to store data that does not need to be sent to the server with each request.

One of the key benefits of using local storage is its reliability. Unlike server-side storage, which can be affected by network outages or other server issues, local storage is

stored locally on the user's machine, and so is not affected by these issues. Another advantage of local storage is its speed. Because the data is stored locally, it is accessed quickly, as there is no need to send requests to a server..

1.2 REST API

A REST API (Representational State Transfer Application Programming Interface) is a popular architectural style for designing networked applications. It is based on a set of principles and constraints that allow for scalability, simplicity, and interoperability between systems.

Client-Server: Separated entities communicate over HTTP or a similar protocol, with distinct responsibilities and the ability to evolve independently.

Stateless: Each request from the client to the server must contain all the necessary information to understand and process the request. The server does not maintain any client state between requests.

Uniform Interface: The API exposes a uniform interface, typically using HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources. Resources are identified by URLs (Uniform Resource Locators).

Cacheable: Responses can be cached by the client or intermediaries to improve performance and reduce the load on the server.

Layered System: Intermediary servers can be placed between the client and server to provide additional functionality, such as load balancing, caching, or security.

1.3 SPRINGBOOT

Spring Boot is an open-source Java framework that simplifies the development of standalone, production-ready applications. It offers several advantages for building robust and scalable applications.

Simplified Configuration: Spring Boot eliminates the need for complex XML configuration files by leveraging sensible default configurations and annotations.

Embedded Server: Spring Boot includes an embedded server (e.g., Apache Tomcat, Jetty) that allows developers to create self-contained applications. This eliminates the need for external server installation and configuration, making it easier to package and deploy the application.

Dependency Management: Spring Boot incorporates the concept of starter dependencies, which are curated sets of libraries that provide commonly used functionalities. It simplifies dependency management and ensures that all required dependencies are included automatically, reducing configuration issues and potential conflicts.

Auto-Configuration: Spring Boot's auto-configuration feature analyzes the class path and automatically configures the application based on the detected dependencies. It saves developers from writing boilerplate configuration code, resulting in faster development and reduced code clutter.

Actuator: Spring Boot Actuator provides out-of-the-box monitoring and management endpoints for the application. It offers metrics, health checks, logging, and other management features, making it easier to monitor and manage the application in production environments.

DevOps Friendliness: Spring Boot's emphasis on simplicity and ease of use makes it DevOps friendly. It supports various deployment options, including traditional servers, cloud platforms, and containerization technologies like Docker. It also provides features for externalized configuration, making it easier to manage different environments.

SYSTEM ARCHITECTURE

The application adopts a contemporary and scalable three-tier architecture. It comprises the frontend layer, backend layer, and the database layer. Each of these

layers fulfills a pivotal role in the application's comprehensive functionality, facilitating seamless communication and efficient data management.

2.1 BACKEND

Fig2.1 Backend System Architecture

In the gym tracking application, the backend layer is built on the robust Spring Boot framework, a Java-based technology renowned for its ability to streamline the development of resilient and scalable web applications. Spring Boot offers a comprehensive suite of features and libraries, simplifying the management of HTTP requests, data persistence, security, and seamless integration with external systems. In this application, the backend's primary role is to create RESTful APIs that support CRUD (Create, Read, Update, Delete) operations, facilitating the dynamic management of workout routines, user progress, and other fitness-related resources. Additionally, the backend handles user management and authentication, ensuring a secure and personalized experience for both trainers and users. The backend architecture is meticulously structured following Spring Boot principles, enhancing security, modularity, and maintainability. This strategic setup allows the gym tracking application to provide a seamless, efficient, and secure fitness experience for all users.REST API:

REST API:

The backend of the Open Library system exposes a RESTful API that allows the frontend to communicate with the server. REST (Representational State Transfer) is an architectural style for designing networked applications. It uses standard HTTP methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources. The API endpoints define the URLs and request/response formats for interacting with the system.

Controller:

In the application, controllers have a crucial role in managing incoming HTTP requests. Controllers map these requests to the appropriate methods within the system. API endpoints are defined by controllers, and orchestrate the processing logic of incoming requests. Controllers act as the gateway between the frontend and backend, receiving user inputs, validating and processing data, interacting with services, and returning the relevant responses.

Services:

Services within the application encapsulate the essential business logic. Responsible for orchestrating complex operations and facilitating interactions between different system components, these operations encompass data retrieval, validation, transformation, and storage. In this context, services manage orders, handle user authentication, and other application-specific functionalities, ensuring a seamless user experience.

Repositories:

In the application, repositories serve as an abstraction layer for interacting with the database. Define the methods required for executing CRUD (Create, Read, Update, Delete) operations and querying the database using SQL or Object-Relational Mapping (ORM) frameworks like Hibernate. These repositories are instrumental in storing and retrieving customize gift and orders-related data from the database, ensuring the persistence and accessibility of vital information.

Data Transfer Objects(DTOs):

Data Transfer Objects (DTOs) play a pivotal role in enabling data exchange between the frontend and backend layers of the application. These objects define the structure and format of data shared in API requests and responses. DTOs are employed to represent destination and order details, user information, and other relevant data that is transferred between the frontend and backend, ensuring seamless .

Security:

Security measures are a top priority within the application. Authentication and authorization protocols are diligently implemented to safeguard user data and system integrity. A robust security framework, such as Spring Security, is employed to manage user authentication and access control. It offers features such as user registration, login, password hashing, and role-based permissions, contributing to a secure and reliable application.

IMPLEMETATION AND FUNCTIONALITY

The Children's interactive story book and educational application's backend is the linchpin of effective place details management. It provides administrators with a comprehensive set of tools to seamlessly add, update, and remove places. The userfriendly interface simplifies their tasks, reducing the learning curve. Security is at priority ,with stringent access controls and user authentication measures protecting sensitive data.

3.1 API Request

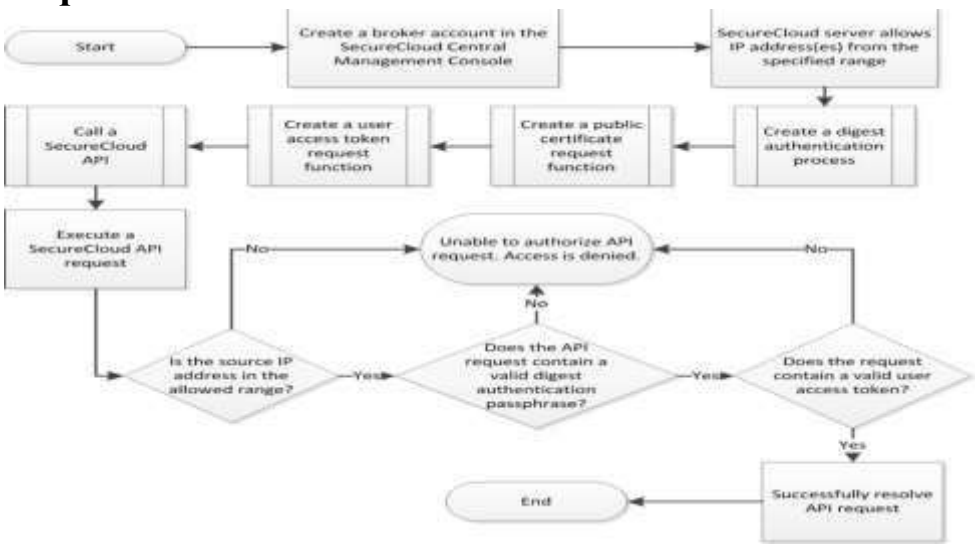

Fig3.1 REST API flow chart

A Representational State Transfer (REST) API plays a pivotal role in the architecture of the Application, providing a structured and efficient means of communication between the frontend and backend components. In this context, the REST API serves as the intermediary that enables the exchange of data and requests, making it a cornerstone of the application's functionality.

The REST API of the Application adheres to RESTful principles, which are centered around a set of stateless operations for creating, retrieving, updating,anddeletingdata.Itdefinesaclearstructurefortheendpoints,with each endpoint corresponding to a specific resource or action. For instance, endpoints might include "GET /user" to retrieve a list of available places, "POST /user" to add places, and "DELETE /auth/{user ID}" to remove places.

By adopting RESTful design, the API simplifies interactions with the application, ensuring that users can easily access place information and complete transactions. It provides data in a format that is widely understood, typically in JSON, allowing for seamless integration with various client applications, including web and mobile interfaces.

In addition to its role in enabling user interactions, the REST API is a fundamental component for potential future developments. It opens the door to third- party integrations, such as payment gateways, external inventory systems, or analytics services, that can enhance the application's functionality and expand its capabilities. Furthermore, the API empowers the application to be scalable, ensuring that it can accommodate growing user bases and evolving feature sets.

The REST API in the Application is not merely a technical component; it's the conduit through which the application's core functionalities are exposed and extended, ultimately contributing to a seamless and versatile user experience.

3.2 CRUD OPERATION

In the application ,the implementation of CRUD(Create, Read, Update, Delete) operations is fundamental to the efficient management of the product inventory. The "Create" operation allows administrators to add new places by entering comprehensive details ,which are then validated for accuracy and completeness before being securely stored in the database. "

The "Update" operation empowers administrators to modify place information, and a user-friendly interface ensures that this process is intuitive. Stringent data validation criteria are maintained to guarantee the accuracy and reliability of the edited information, which is subsequently updated in the database.

The "Delete" operation provides administrators with the means to remove places from the inventory, incorporating a confirmation prompt to prevent accidental deletion. Once confirmed, the place’s information is securely deleted from the database.

Coding:

Entity Class:

```
package com.example.demo.controller;


import com.example.demo.model.Equipment;
import com.example.demo.service.EquipmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/equipments")
public class EquipmentController {

   @Autowired
   private EquipmentService equipmentService;

   @GetMapping
   public List<Equipment> getAllEquipments() {
      return equipmentService.getAllEquipments();
   }

   @GetMapping("/{id}")
   public ResponseEntity<Equipment> getEquipmentById(@PathVariable(value = "id")
String equipmentID) {
      Optional<Equipment> equipment =
equipmentService.getEquipmentById(equipmentID);
      return equipment.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
   }

   @PostMapping
   public Equipment createEquipment(@RequestBody Equipment equipment) {
      return equipmentService.saveEquipment(equipment);
   }
```

```
    @PutMapping("/{id}")
    public ResponseEntity<Equipment> updateEquipment(@PathVariable(value = "id")
String equipmentID, @RequestBody Equipment equipmentDetails) {
        Equipment updatedEquipment = equipmentService.updateEquipment(equipmentID,
equipmentDetails);
        return updatedEquipment != null ? ResponseEntity.ok(updatedEquipment) :
ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteEquipment(@PathVariable(value = "id") String
equipmentID) {
        equipmentService.deleteEquipment(equipmentID);
        return ResponseEntity.noContent().build();
    }
}
```

Service Class:

```
package com.example.demo.service;


import com.example.demo.model.Equipment;
import com.example.demo.repository.EquipmentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class EquipmentService {

    @Autowired
    private EquipmentRepository equipmentRepository;

    public List<Equipment> getAllEquipments() {
        return equipmentRepository.findAll();
    }
```

```
    public Optional<Equipment> getEquipmentById(String equipmentID) {
        return equipmentRepository.findById(equipmentID);
    }

    public Equipment saveEquipment(Equipment equipment) {
        return equipmentRepository.save(equipment);
    }

    public void deleteEquipment(String equipmentID) {
        equipmentRepository.deleteById(equipmentID);
    }

    public Equipment updateEquipment(String equipmentID, Equipment equipmentDetails) {
        Optional<Equipment> optionalEquipment = equipmentRepository.findById(equipmentID);
        if (optionalEquipment.isPresent()) {
            Equipment equipment = optionalEquipment.get();
            equipment.setName(equipmentDetails.getName());
            equipment.setDescription(equipmentDetails.getDescription());
            equipment.setStatus(equipmentDetails.getStatus());
            equipment.setLocation(equipmentDetails.getLocation());
            equipment.setLastMaintenanceDate(equipmentDetails.getLastMaintenanceDate());
            return equipmentRepository.save(equipment);
        } else {
            return null; // or throw an exception
        }
    }
}
```

Repository Interface:

```
package com.example.demo.repository;
import com.example.demo.model.Equipment;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EquipmentRepository extends JpaRepository<Equipment, String> {
}
```

3.3 DATA RETRIEVEL PROCESS

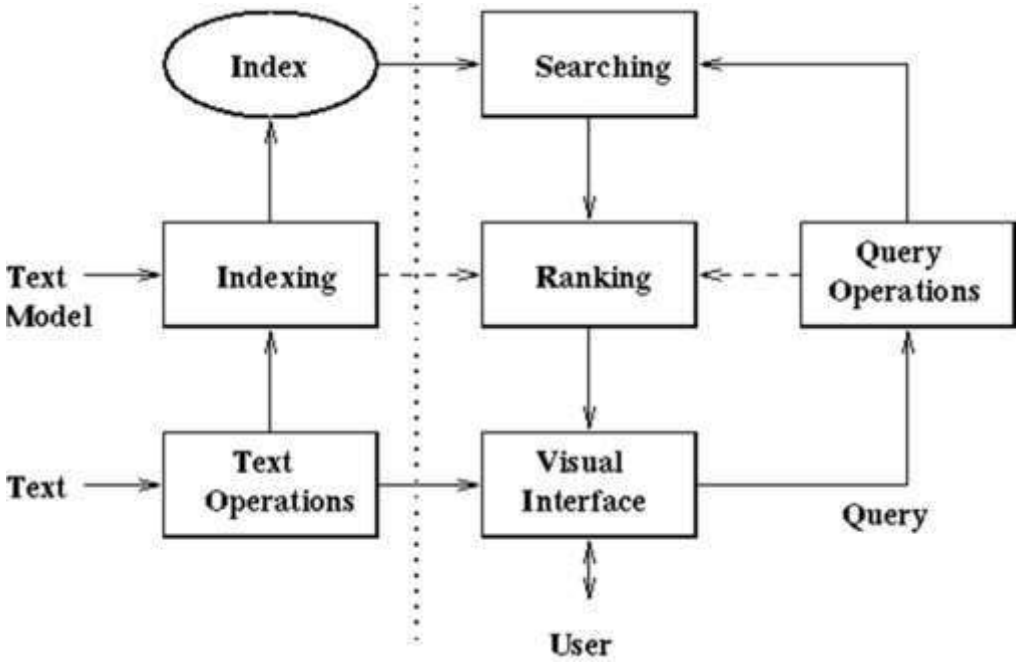

Fig3.2 Data Retrieval Process

The data retrieval process in the backend of the Application is a pivotal component that facilitates the efficient and secure retrieval of data from the database and its transmission to the frontend or client applications. This process begins with a client request made to a specific API endpoint on the backend. Upon receiving the request, the back ends routing system directs it to the appropriate endpoint handler based on the URL and HTTP method.

3.4 DATA UPDATE PROCESS

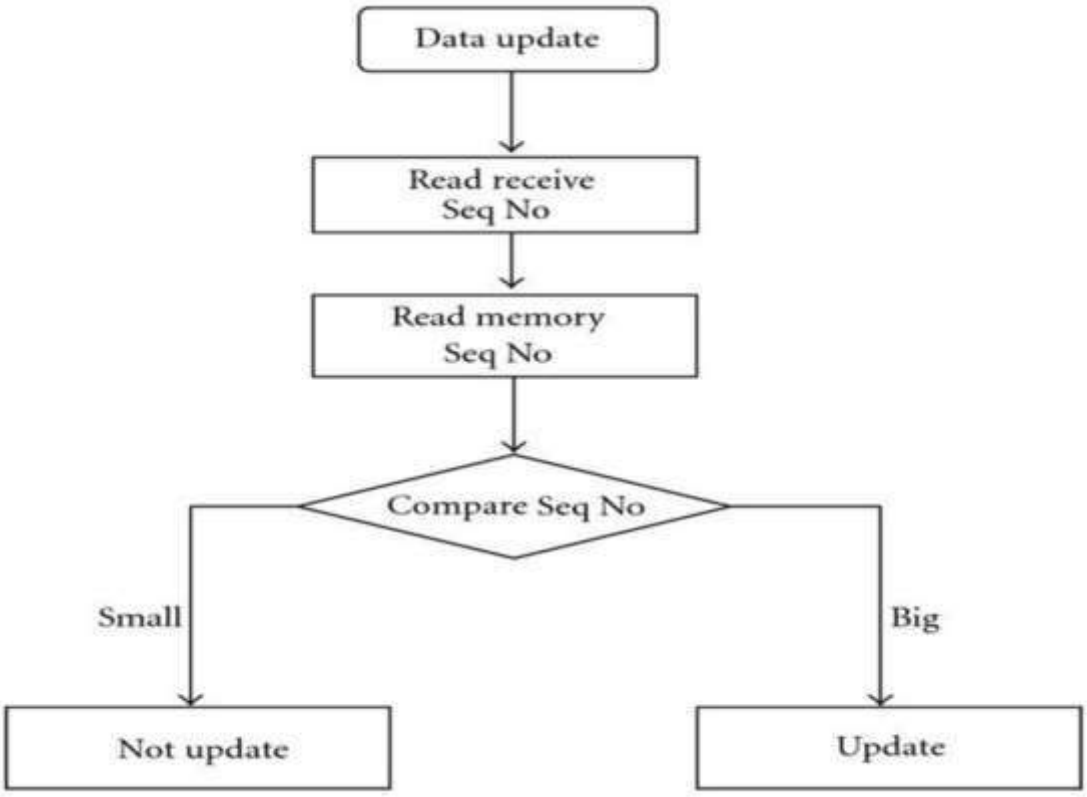

Fig3.3 Data Update Flowchart

The data update process within the backend of Application is a crucial mechanism that allows users and administrators to enact changes in the application's database.

Authentication and authorization are fundamental checkpoints in this process. The backend confirms the user's identity and verifies if the user possesses the necessary permissions to execute the data update operation. Unauthorized requests are diligently restricted.

Data validation is a subsequent step, whereby the backend scrutinizes the provided data to ensure that it adheres to the correct format and complies with established business rules and constraints .This phase plays a pivotal role in maintaining data integrity.

3.5 SECURITY AND AUTHENTICATION

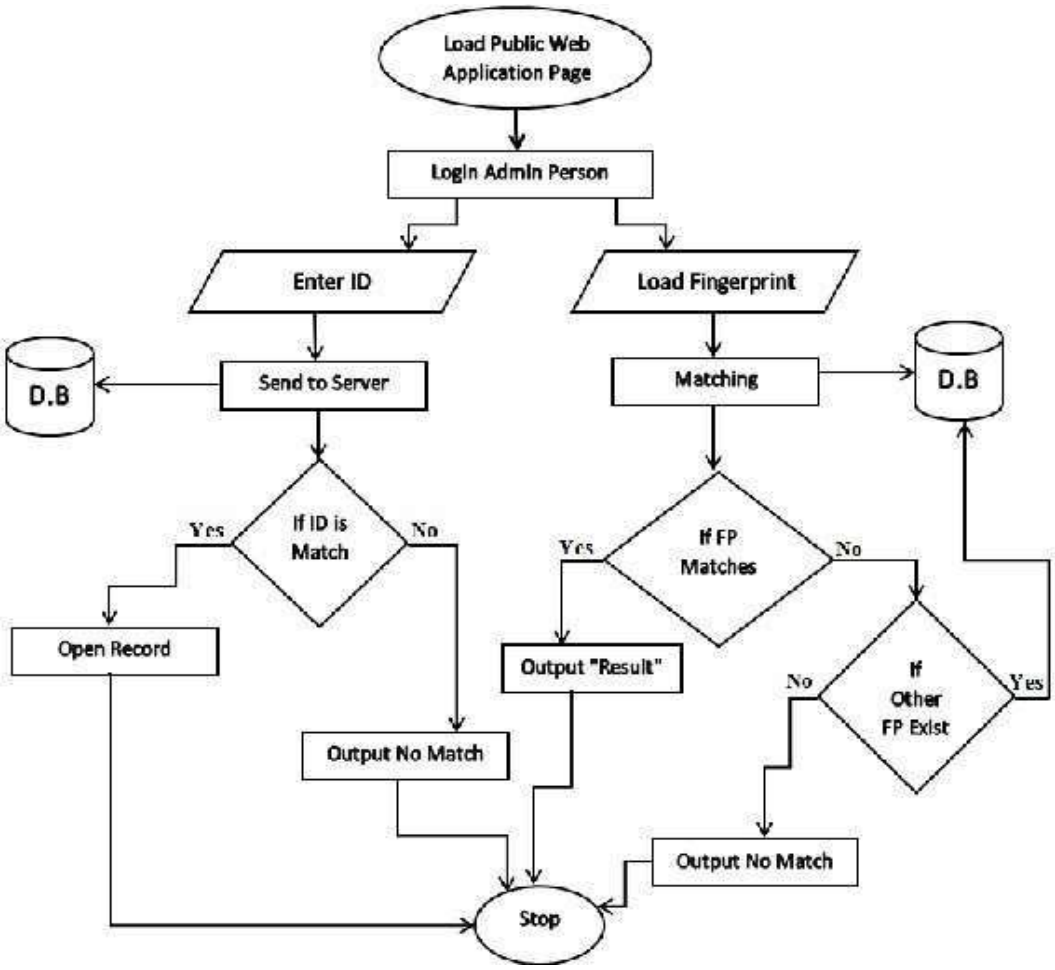

Fig3.4 Security And Authentication Flowchart

Security and authentication lie at the heart of the application's robust infrastructure. To safeguard sensitive data and uphold the integrity of the system .

User authentication is a fundamental pillar, allowing users, including administrators and customers, to register securely, leveraging email and strong, hashed passwords. A robust login system verifies user credentials and controls access to the application.

Data encryption, both in transit and at rest, is a core component of the security framework. Secure communication channels protect data during interactions, while encryption of sensitive data in the database safeguards information in the event of a breach. Session management maintains secure user sessions, preventing unauthorized access or data exposure.

Utilizing security libraries and frameworks, like Spring Security, enhances the efficiency of authentication and access control. Additionally, protection against common security threats, such as cross-site scripting and SQL injection, is in place.

User awareness and education contribute to the overall security posture, ensuring that users are informed and capable of recognizing potential threats. Regular security audits and vulnerability assessments are conducted to proactively identify and address potential weaknesses, keeping the application resilient against emerging security risks. In sum, these measures collectively create a secure and trustworthy environment within the the application, protecting user data and maintaining the application's integrity.

Coding:

Jwt Authentication Filter Class:

```
package com.example.giftcraft.config; import java.io.IOException;
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails; import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import
org.springframework.web.filter.OncePerRequestFilter;
import com.example.giftcraft.service.JwtService; import
jakarta.servlet.FilterChain; import
jakarta.servlet.ServletException; import
jakarta.servlet.http.HttpServletRequest; import
jakarta.servlet.http.HttpServletResponse;

@Component
```

```
public class AuthenticationFilter extends OncePerRequestFilter{
   @Autowired                private
JwtService jwtService;
   @Autowired                    private    UserRegisterDetailsService
userRegisterDetailsService;


   @Override
   protected                void                doFilterInternal(
HttpServletRequest request,
       HttpServletResponse                    response,
FilterChain filterChain)
           throws ServletException, IOException {
       String authHeader = request.getHeader("Authorization");
       String token = null;         String username = null;         if
(authHeader != null && authHeader.startsWith("Bearer ")) {
token = authHeader.substring(7);              username =
jwtService.extractUsername(token);
       }

           if (username != null &&
   SecurityContextHolder.getContext().getAuthentication() == null) {
               UserDetails userDetails =
   userRegisterDetailsService.loadUserByUsername(username);
   if (jwtService.validateToken(token, userDetails)) {
                   UsernamePasswordAuthenticationToken authToken = new
   UsernamePasswordAuthenticationToken(userDetails,
                           null, userDetails.getAuthorities());
                   authToken.setDetails(new
   WebAuthenticationDetailsSource().buildDetails(request));
                   SecurityContextHolder.getContext().setAuthentication(authToken);
               }
```

```
        }
        filterChain.doFilter(request, response);
    }
```

Jwt Service Class:

```
package com.example.giftcraft.service;
import java.util.Date; import
java.util.HashMap; import
java.util.Map; import
java.util.function.Function;
import org.springframework.security.core.userdetails.UserDetails; import
org.springframework.stereotype.Component;
import java.security.Key; import
io.jsonwebtoken.Claims; import
io.jsonwebtoken.Jwts; import
io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;

@Component
public class JwtService {
   @Value("${application.jwt.secret-key}")
   private String secretKey;

   @Value("${application.jwt.token-expiration:1800000}") // Default token
expiration: 30 minutes     private long tokenExpiration;

   public String extractUsername(String token) {
      return extractClaim(token, Claims::getSubject);
   }

   public Date extractExpiration(String token) {
      return extractClaim(token, Claims::getExpiration);
   }

   public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
final Claims claims = extractAllClaims(token);        return
claimsResolver.apply(claims);
```

```
    }

    private Claims extractAllClaims(String token) {
        return                                    Jwts.parserBuilder()
.setSigningKey(getSignKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private   Boolean   isTokenExpired(String   token)   {
return extractExpiration(token).before(new Date());
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
final String username = extractUsername(token);         return
(username.equals(userDetails.getUsername()) &&
!isTokenExpired(token));
    }

    public String generateToken(String username) {
Map<String, Object> claims = new HashMap<>();         // You
can add additional claims here if needed         return
createToken(claims, username);
    }

    private String createToken(Map<String, Object> claims, String username) {
return Jwts.builder()                .setClaims(claims)
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + tokenExpiration))
                .signWith(getSignKey(),                  SignatureAlgorithm.HS256)
.compact();
    }

    private Key getSignKey() {
        byte[]    keyBytes    =    Decoders.BASE64.decode(secretKey);
return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

CONCLUSION:

In conclusion, the development of an online platform dedicated to gym tracking offers a transformative solution to the increasing demand for accessible, efficient, and personalized fitness management in today's health-conscious society. By leveraging technology to streamline the fitness journey, this platform empowers users to monitor and optimize their workouts, track progress, and achieve their health and fitness goals effectively. Through the seamless integration of workout logging, progress tracking, and personalized fitness plans, it promises to revolutionize the way fitness is approached and experienced, ensuring that every user's journey is engaging, motivating, and impactful. With this innovative tool at their fingertips, fitness enthusiasts and trainers alike can look forward to a more dynamic and fulfilling fitness experience, marking the beginning of a new era in health and wellness.

Overall, an online platform for gym tracking not only revolutionizes the way individuals manage their fitness but also enhances the overall experience for users. By utilizing technology to streamline workout planning, enhance motivation, and provide personalized insights, it promises to usher in a new era of effortless and enjoyable fitness management. This ensures that every workout is a resounding success and a vital step towards achieving optimal health and well-being.