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

FITZONE

A PROJECT REPORT

Submitted by

KISHORE S **727821TUAD027**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

NOVEMBER 2023

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

BONAFIDE CERTIFICATE

Certified that this project report **"FITZONE"** is the bonafide work of "**KISHORE S"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. R.KARTHIK	**Dr. C. P. MAHESWARAN**
SUPERVISOR	**HEAD OF THE DEPARTMENT**
Associate Professor, Department of Computer Science and Engineering (Cyber Security), Sri Krishna College of Technology, Coimbatore-641042.	Associate Professor, Department of Artificial Intelligence and Data Science, Sri Krishna College of Technology, Coimbatore-641042.

Certified that the candidates were examined by us in the Project Viva Voce examination held on______________________at Sri Krishna College of Technology, Kovaipudur, Coimbatore -641042.

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

Dedicating this project to the **ALMIGHTY GOD** whose abundant grace and mercies enabled its successful completion.

We extend our deep gratitude to our beloved Principal, **Dr. M. G. Sumithra,** for her kindness and unwavering support throughout the project work.

We are grateful to our beloved Dean Academics affairs and assessment **Dr. R. Rameshkumar,** for his tireless and relentless support.

We extend our heartfelt thanks to our beloved Dean Accreditation and Ranking **Dr. P. Manju,** for her advice and ethics inculcated during the entire period of our study.

We would like to express our deep gratitude to **Dr. J. Shanthini,** Head of Computing Science and **Dr. C. P. Maheswaran,** Programme Coordinator of Artificial Intelligence and Data Science, for their exceptional dedication and care towards success of this project.

We would like to extend our heartfelt gratitude to our Project guide **Dr.Suma Sira Jacob** Assistant Professor, Department of Artificial Intelligence and Data Science for her valuable guidance and suggestions in all aspects that aided us to ameliorate our skills.

We are thankful to all those who have directly and indirectly extended their help to us in completing this project work successfully.

ABSTRACT

In a world increasingly conscious of health and fitness, the need for a comprehensive and user-friendly gym application has never been more apparent. This report outlines the process of conceptualizing, designing, and developing a state-of-the-art Gym Application, catering to the needs of fitness enthusiasts, gym owners, and trainers. The Gym Application is envisioned as a versatile tool, encompassing features such as workout tracking, personalized training plans, nutrition guidance, and social connectivity, making it a one-stop solution for the fitness community. The project's scope ranges from defining the application's core functionality to designing an intuitive user interface, choosing the right technology stack, and implementing a secure and scalable infrastructure.The Gym Application's development process is documented in stages, from initial ideation and requirement gathering to coding, testing, and deployment. Emphasis is placed on the agile development methodology, ensuring flexibility and adaptability to evolving user needs and industry trends.The Gym Application's ultimate goal is to contribute to a healthier society by making fitness accessible and enjoyable, promoting a lifestyle that embraces well-being. The report provides an in-depth view of this innovative project, shedding light on the intersection of technology, health, and human motivation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIVE	**ABBREVIATION**
JSON	JAVASCRIPT OBJECT NOTATION
JWT	JAVASCRIPT WEB TOKEN
HTTP	HYPER TEXT TRANSFER PROTOCOL
API	APPLICATION PROGRAMMING INTERFACE
SQL	STRUCTURED QUERY LANGUAGE

CHAPTER 1
INTRODUCTION

The Gym Application is a complete fitness ecosystem, designed to adapt to your unique needs and goals. Whether you're an early bird hitting the gym, a busy professional seeking quick home workouts, or a weekend warrior exploring new routines, this app accommodates your lifestyle. Our seamless wearable integration allows you to sync your favorite fitness trackers, ensuring that you get the most accurate and comprehensive data on your physical activity and health stats.

Plan your workouts in advance, set reminders, and establish a consistent routine with the Workout Scheduler feature. It's the key to maintaining your commitment and achieving long-term success. Moreover, our achievement badge system rewards your dedication and perseverance, giving you a sense of accomplishment with each milestone you reach. The Ultimate Gym Application isn't just an app; it's your personal fitness coach, your nutritionist, your community, and your source of inspiration. It's time to embark on your fitness journey with confidence, knowing that success is within reach. Let's make your health and fitness dreams a reality.

1.1 PROBLEM STATEMENT

To build a Gym Trainer and Process Tracker Application usingReact (Frontend), Spring Boot Microservices (Backend), and MySQL (database).

1.2 OVERVIEW

The Gym Application is your all-in-one solution for achieving your fitness goals and embracing a healthier lifestyle. With its personalized workouts, exercise instructions, and a vast array of nutrition guidance, it ensures you have the tools and knowledge to succeed in your fitness journey. Real-time progress tracking keeps you accountable and motivated, allowing you to see your achievements unfold before your eyes.It will make your health and wellness aspirations a reality.

1.3 OBJECTIVE

The primary objective of the Gym Application is to empower individuals to achieve their fitness and wellness goals effectively and efficiently by providing a comprehensive, user-friendly, and personalized fitness platform. The application aims to offer tailored workout routines, expert guidance on exercise and nutrition, real-time progress tracking, a supportive community, and a seamless integration of wearable technology to create a holistic fitness experience. The ultimate goal is to inspire and assist users, regardless of their fitness level or background, in their pursuit of a healthier and more active lifestyle..

CHAPTER 2

LITERATURE SURVEY

In Gym applications have become a cornerstone of the digital fitness revolution, offering users an array of tools and resources that cater to their individual health and wellness needs. They empower individuals by granting them instant access to personalized workout routines, nutritional guidance, and the ability to track their fitness progress.

The reviewed literature consistently highlights the numerous advantages of these applications, including their convenience, flexibility, and potential to foster motivation. Users can access expert advice, engage with fitness communities, and experience a sense of achievement through workout tracking and achievement badges. However, challenges such as user engagement and adherence pose noteworthy hurdles. As the fitness app landscape continues to evolve, it becomes increasingly evident that in-depth research is essential to refine and enhance the user experience while addressing the identified gaps.

In this context, gym applications have the potential not only to revolutionize individual fitness journeys but also to transform the broader health and wellness landscape. The implications of these findings underscore the importance of user engagement and adherence strategies for both application developers and users alike, as we collectively pursue healthier, more active lifestyles.

The literature survey on gym applications underscores their pivotal role in modern fitness and wellness. Gym applications offer a convenient and accessible means for individuals to engage in tailored workouts, access nutritional guidance, and track their fitness progress.

Research suggests that these apps hold substantial potential for motivating users and helping them reach their fitness goals. However, they also face challenges related to user engagement and long-term adherence. This survey identifies a critical need for further research in understanding and enhancing user engagement strategies and in assessing the efficacy of gym applications for specific fitness objectives and diverse user populations. In conclusion, gym applications are poised to reshape the fitness landscape, but their continued success hinges on addressing the highlighted gaps and optimizing user experiences. This literature survey sheds light on the path forward, emphasizing the importance of effective engagement and adherence strategies for developers and users alike.

A gym application encompasses a comprehensive review of existing research, studies, and literature related to the development and usage of fitness and workout applications. Numerous studies have examined the benefits of mobile applications in promoting physical activity and healthy lifestyle choices. Research has shown that such applications can significantly enhance user engagement, motivation, and adherence to fitness routines. Additionally, investigations into the design and user experience aspects of gym apps have highlighted the importance of intuitive interfaces, gamification elements, and personalized features to maximize user satisfaction and long-term usage. Furthermore, studies have explored the impact of wearable fitness technology integration with gym applications, shedding light on the potential for real-time tracking and data analysis to improve exercise outcomes. By examining this body of literature, one can gain valuable insights into the latest trends and best practices in the development of gym applications, ultimately contributing to the creation of more effective and user-friendly fitness tools.

Gym applications have the potential to significantly impact the fitness and wellness landscape by providing accessible and personalized fitness solutions. Understanding the factors that influence user engagement and adherence is essential for both developers and users to fully realize the benefits of these applications.

CHAPTER 3
SYSTEM SPECIFICATIONS

3.1 BACK-END DEVELOPMENT

Back-End Development refers to the server-side development. It focuses on databases, scripting, and website architecture. It contains behind-the-scenes activities that occur when performing any action on a website. It can be an account login or making to fit for a healthy life . Code written by back-end developers helps browsers to communicate with database information.

3.1.1 REST API – SPRING BOOT

Representational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. **REST API** is a way of accessing web services in a simple and flexible way without having any processing. REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.

Working: A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON. But now JSON is the most popular format being used in Web Services.

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**.

You can get started with minimum configurations without the need for an entire Spring configuration setup. Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method. Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation.

3.1.2 MICROSERVICES

Microservices are a software development approach that structures an application as a collection of loosely coupled services. Each service is self-contained and performs a specific task. Microservices communicate with each other through well-defined APIs.

Embedded servers: Spring Boot can embed a web server in the application, so there is no need to deploy the application to a separate server. This makes it easy to develop and deploy microservices.

Autoconfiguration: Spring Boot can automatically configure many common Spring features, such as database connections and web servers. This reduces the boilerplate code that developers need to write and makes it easier to get started with microservices.

Starters: Spring Boot starters are groups of dependencies that are pre-configured for specific tasks, such as developing microservices with REST APIs or messaging queues. Starters make it easy to get started with microservices and avoid common configuration mistakes.

3.1.3 EUREKA

Eureka is one of the great libraries of Netflix OSS particularly focused on service discovery and Spring has a fluent integration with it. In this story, I will share a very basic example containing 4 modules:

Eureka-server: The service registry

Eureka-dto: The DTO container module

Eureka-client: A microservice containing a sample REST endpoint.

Eureka-feign-client: A sample application consuming the REST endpoint provided by the eureka-client.

Note that all these modules share the same Maven parent project in which project lombok is added under the dependencies section. Therefore, you'll not see it in the dependencies section of any module. By default, Eureka server is also a Eureka client and it can register itself to other Eureka server so that they can work as peers. This way your service registry becomes more resilient to failures. But, if you want to switch off the client behavior of your service registry and want a single registry instance, then you should disable register-with-eureka and fetch-registry configurations

3.1.4 JWT

JWT, or JSON Web Token, is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

A JWT consists of three parts:

Header: The header contains information about the token, such as the type of token and the signing algorithm used.

Payload: The payload contains the claims, which are statements about the user or the application.

Signature: The signature is used to verify the integrity of the token.

JWTs can be used for a variety of purposes, such as:

Authentication: JWTs can be used to authenticate users. When a user successfully logs in, the server can generate a JWT and send it to the user. The user can then store the JWT in their browser or in local storage. When the user needs to access a protected resource, they can send the JWT to the server. The server can then verify the JWT and authenticate the user.

Authorization: JWTs can be used to authorize users to perform specific actions. For example, a JWT can contain a claim that gives the user permission to access a certain resource. When the user needs to access the resource, they can send the JWT to the server. The server can then verify the JWT and authorize the user to access the resource.

Information exchange: JWTs can be used to securely exchange information between parties. For example, a JWT can contain information about a user, such as their name and email address. When a user needs to share this information with another party, they can send the JWT to the other party. The other party can then verify the JWT and trust the information in the payload.

3.2 DATA BASE MANAGEMENT

Database management refers to the actions a business takes to manipulate and control data to meet necessary conditions throughout the entire data lifecycle. Database management has become more important as the volume of business data has grown. Rapid data growth creates a wide variety of negative conditions, including poor application performance and compliance risk, to name a few. Database management comprises a number of proactive techniques to prevent the deleterious effects of data growth. A database management task is any task that protects the organization’s data, prevents legal and compliance risk, and keeps data-driven applications performing at their best. This includes performance monitoring and tuning, storage and capacity planning, backup and recovery, data archiving, data partitioning, replication, masking, and retirement.

3.2.1 MYSQL

MYSQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co- founder Wideness's daughter "My" and "SQL", the acronym for Structured Query Language. A relational database organizes data into one or more data tables in whichdata may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

CHAPTER 4

SOFTWARE REQUIREMENTS

Operating System	**:**	Runs in any operating system (Not Specific)
Front End	**:**	React JS
Back End	**:**	REST API - SPRING BOOT
Data Base Management	**:**	MYSQL
I**DE Used**	**:**	Visual Studios Code (VS-Code) , Spring Tool Suite (STS) , Eclipse.
RAM	**:**	2 to 3 times the size of the database.
Storage	**:**	Minimum of 50 GB Storage
Processor	**:**	Any Quad Core Processors (3.3Ghz)
Connectivity	**:**	Internet speed of min.2Mbps .

CHAPTER 5

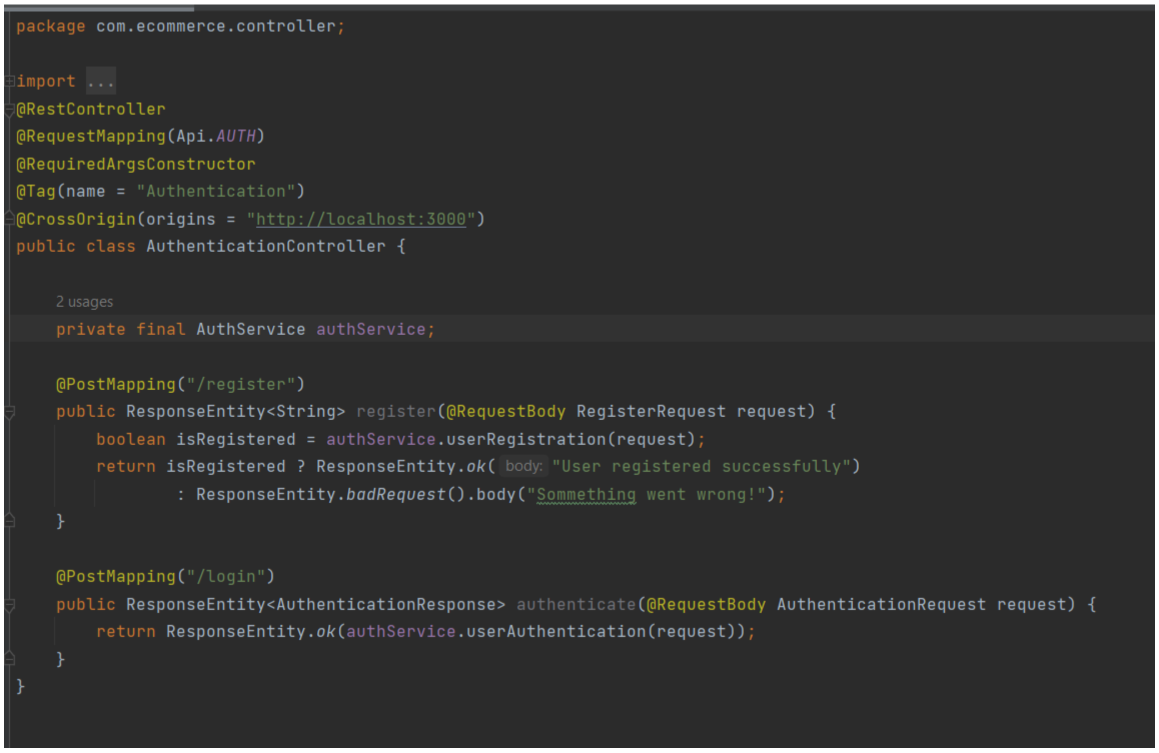
BACKEND IMPLEMENTATION

5.1 MAIN CONTROLLER

The Main Controller of our project consists of 4 files namely Gymuser Controller, Authentication Controller , Trainer Controller and Microserver Controller.

5.1.1 AUTHENTICATION CONTROLLER

User Controller consists of functions which carry on the process of login / signup, signup as a seller and to get all the products from the database.

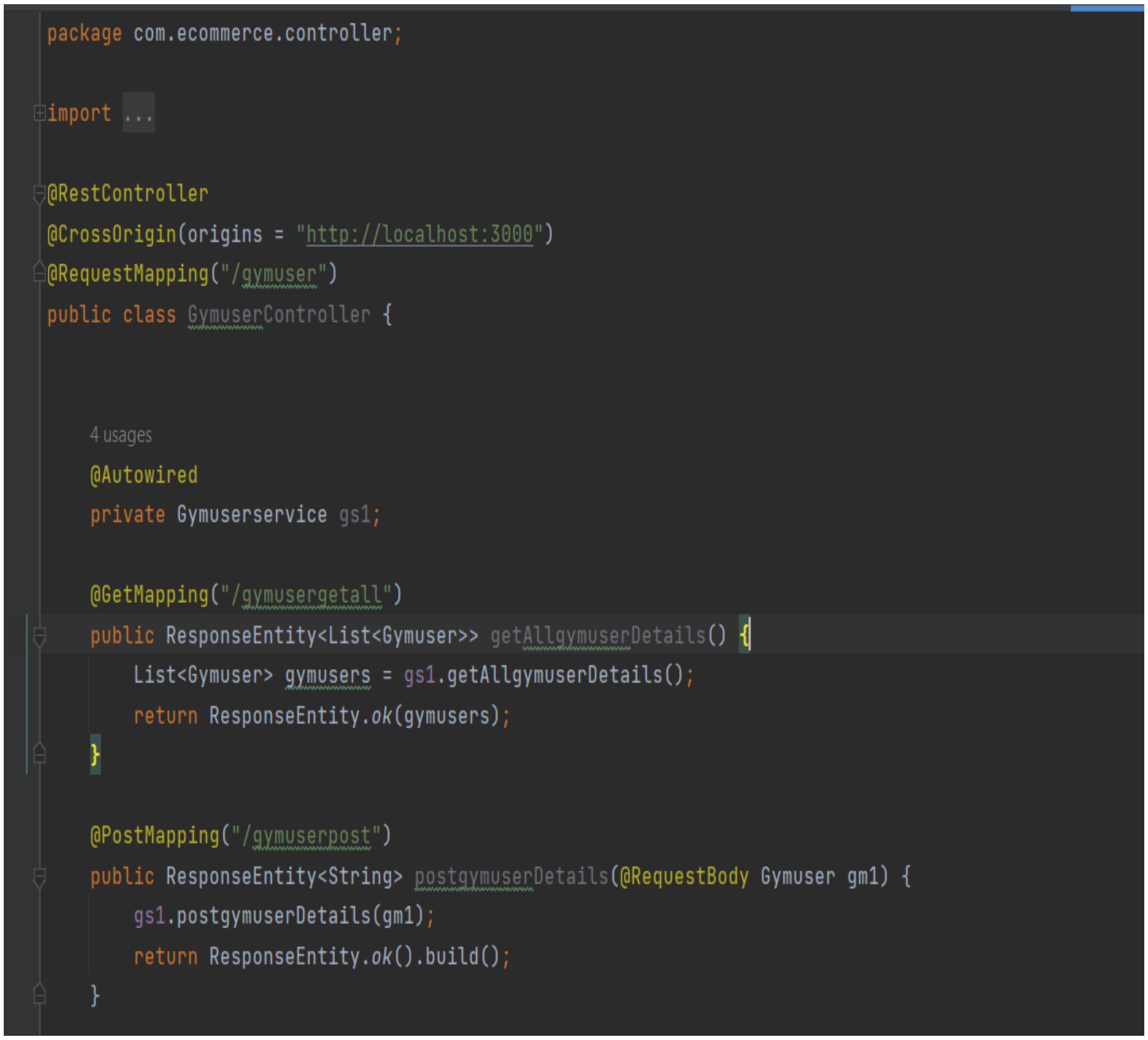


```
package com.ecommerce.controller;

import ...
@RestController
@RequestMapping(Api.AUTH)
@RequiredArgsConstructor
@Tag(name = "Authentication")
@CrossOrigin(origins = "http://localhost:3000")
public class AuthenticationController {

    2 usages
    private final AuthService authService;

    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestBody RegisterRequest request) {
        boolean isRegistered = authService.userRegistration(request);
        return isRegistered ? ResponseEntity.ok( body: "User registered successfully")
                : ResponseEntity.badRequest().body("Sommething went wrong!");
    }

    @PostMapping("/login")
    public ResponseEntity<AuthenticationResponse> authenticate(@RequestBody AuthenticationRequest request) {
        return ResponseEntity.ok(authService.userAuthentication(request));
    }
}
```

Fig 5.1.1 – Authentication Controller

5.1.2 GYMUSER CONTROLLER

The Gymuser Controller consists of functions where a specific user can to add the additional information fot fitness. Admin can able to delete the user information

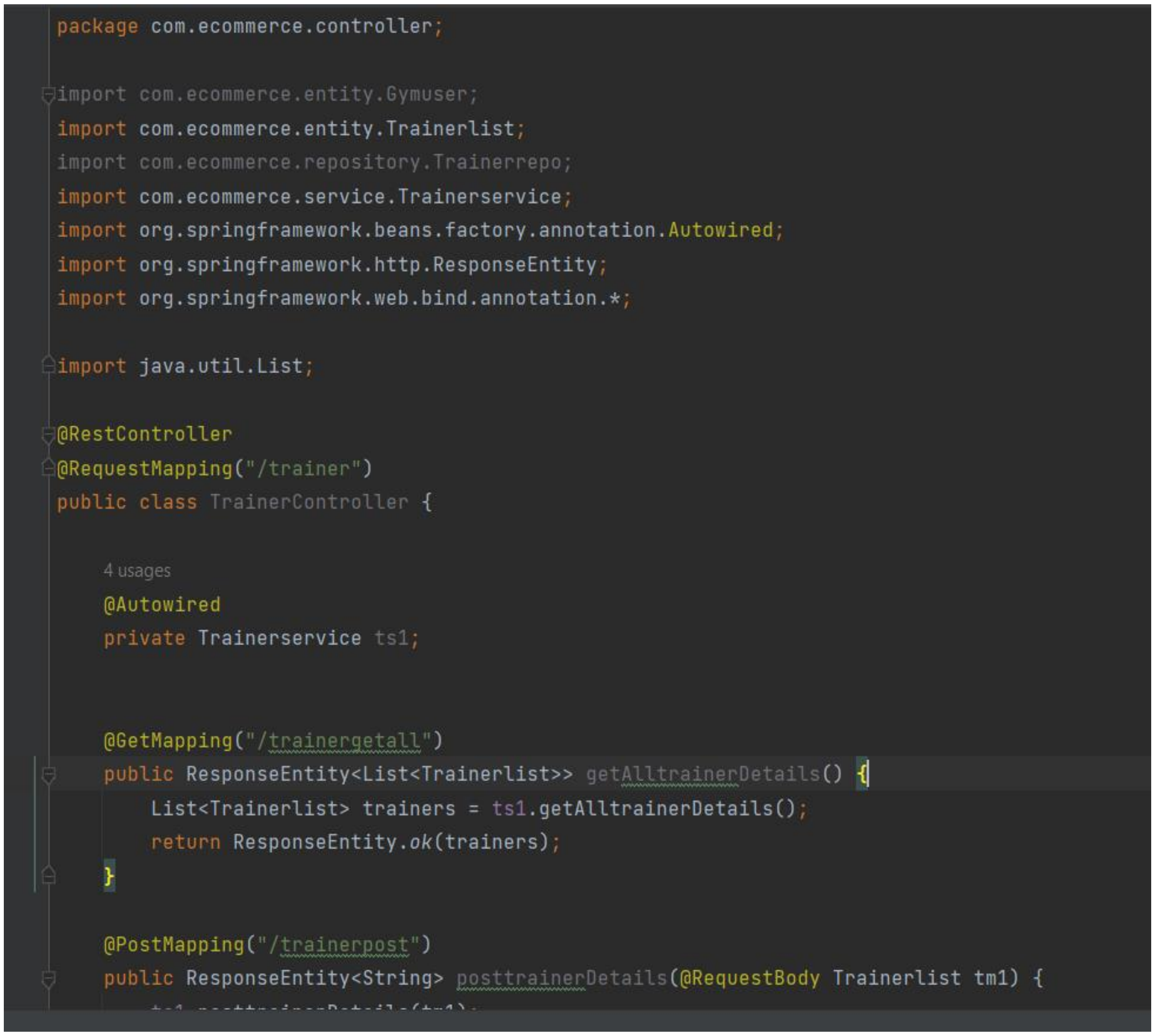


```
package com.ecommerce.controller;

import ...

@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/gymuser")
public class GymuserController {



    4 usages
    @Autowired
    private Gymuserservice gs1;

    @GetMapping("/gymusergetall")
    public ResponseEntity<List<Gymuser>> getAllgymuserDetails() {
        List<Gymuser> gymusers = gs1.getAllgymuserDetails();
        return ResponseEntity.ok(gymusers);
    }

    @PostMapping("/gymuserpost")
    public ResponseEntity<String> postgymuserDetails(@RequestBody Gymuser gm1) {
        gs1.postgymuserDetails(gm1);
        return ResponseEntity.ok().build();
    }
```

Fig 5.1.2 – Gymuser Controller

5.1.3 TRAINER CONTROLLER

Trainer Controller consists of functions that can able to check the user progress about their fitness.



```java
package com.ecommerce.controller;

import com.ecommerce.entity.Gymuser;
import com.ecommerce.entity.Trainerlist;
import com.ecommerce.repository.Trainerrepo;
import com.ecommerce.service.Trainerservice;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/trainer")
public class TrainerController {

    4 usages
    @Autowired
    private Trainerservice ts1;


    @GetMapping("/trainergetall")
    public ResponseEntity<List<Trainerlist>> getAlltrainerDetails() {
        List<Trainerlist> trainers = ts1.getAlltrainerDetails();
        return ResponseEntity.ok(trainers);
    }

    @PostMapping("/trainerpost")
    public ResponseEntity<String> posttrainerDetails(@RequestBody Trainerlist tm1) {
```

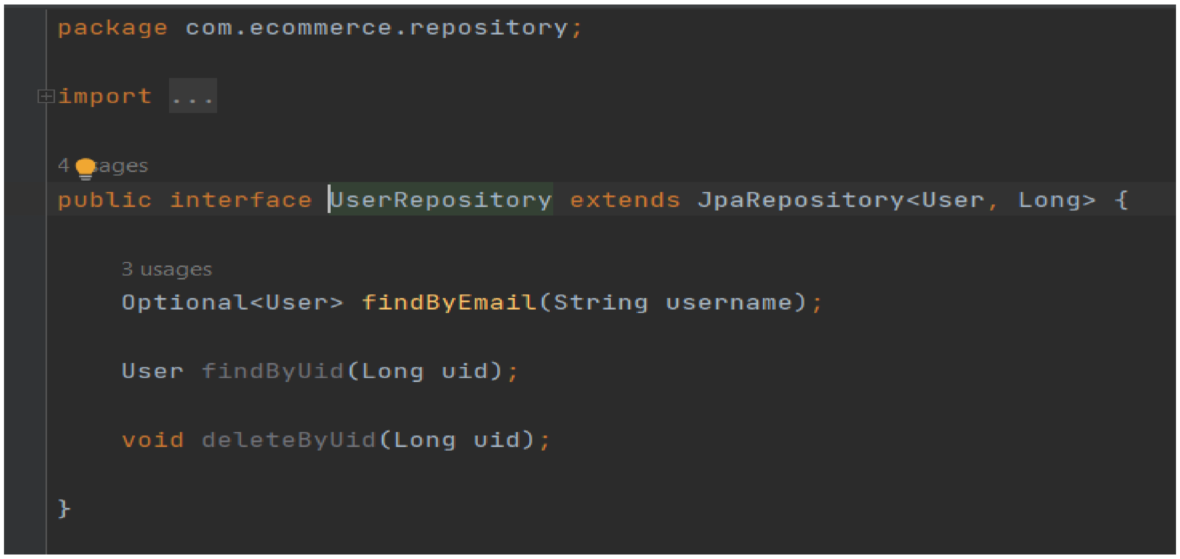
Fig 5.1.3 – Trainer Controller

5.1.4 MICROSERVER CONTROLLER

Microserver Controller is used to connect with other services using RestTemplate class. The Microservice includes , comparing two products and prociding feedback to a specific products. This controller will invoke another function which will be in the controller of support server which is running in another port.

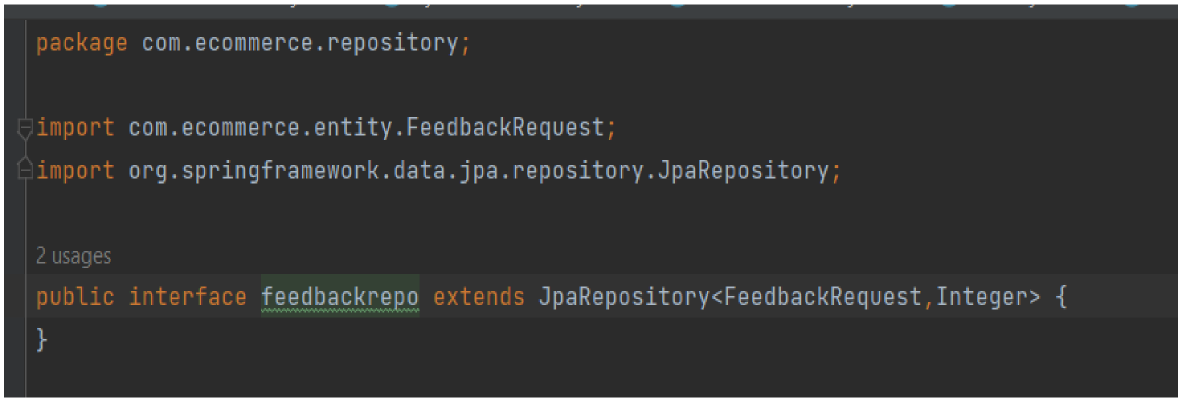
5.2 MAIN REPOSITORY

The Main Repository consists of 4 files namely CrudDb, ProductDb, SellerDb and UserDb. These files has native SQL queries which performs the operations of controllers.



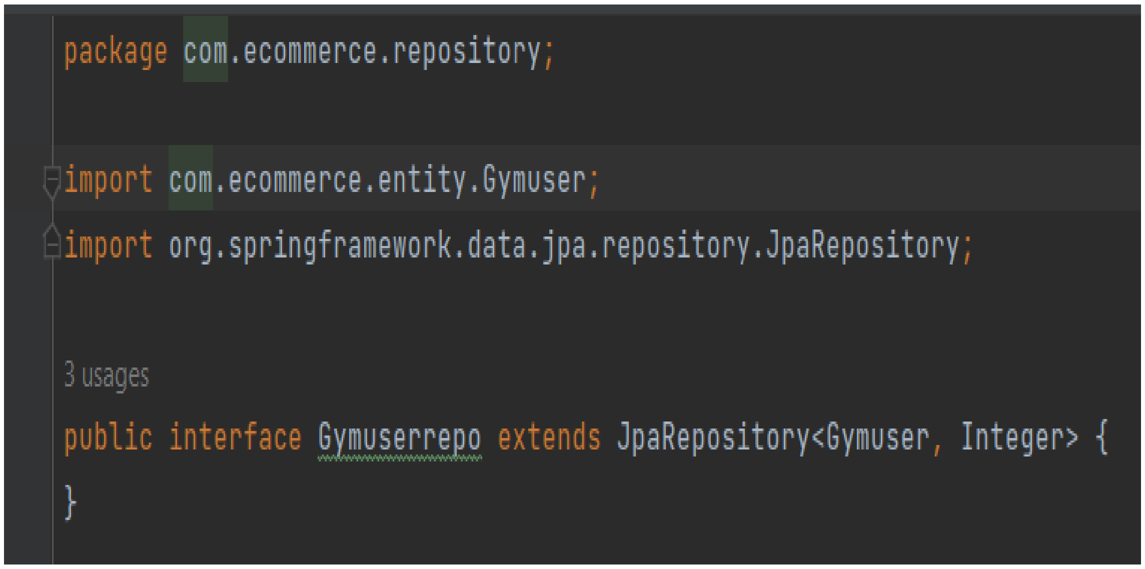
```
package com.ecommerce.repository;

import ...

4 usages
public interface UserRepository extends JpaRepository<User, Long> {

    3 usages
    Optional<User> findByEmail(String username);

    User findByUid(Long uid);

    void deleteByUid(Long uid);

}
```

Fig 5.2.1 – User Repository



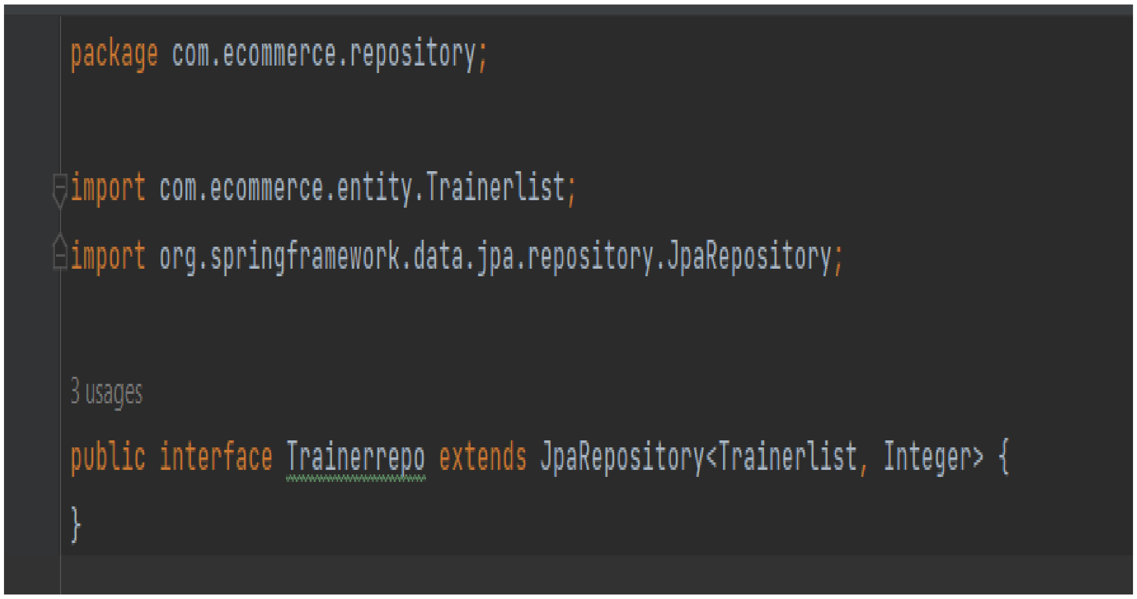
```
package com.ecommerce.repository;

import com.ecommerce.entity.FeedbackRequest;
import org.springframework.data.jpa.repository.JpaRepository;

2 usages
public interface feedbackrepo extends JpaRepository<FeedbackRequest,Integer> {
}
```

Fig 5.2 – Microserver Repository



```java
package com.ecommerce.repository;

import com.ecommerce.entity.Gymuser;
import org.springframework.data.jpa.repository.JpaRepository;

3 usages
public interface Gymuserrepo extends JpaRepository<Gymuser, Integer> {
}
```

Fig 5.2.3 – Gymuser Repository



```java
package com.ecommerce.repository;

import com.ecommerce.entity.Trainerlist;
import org.springframework.data.jpa.repository.JpaRepository;

3 usages
public interface Trainerrepo extends JpaRepository<Trainerlist, Integer> {
}
```

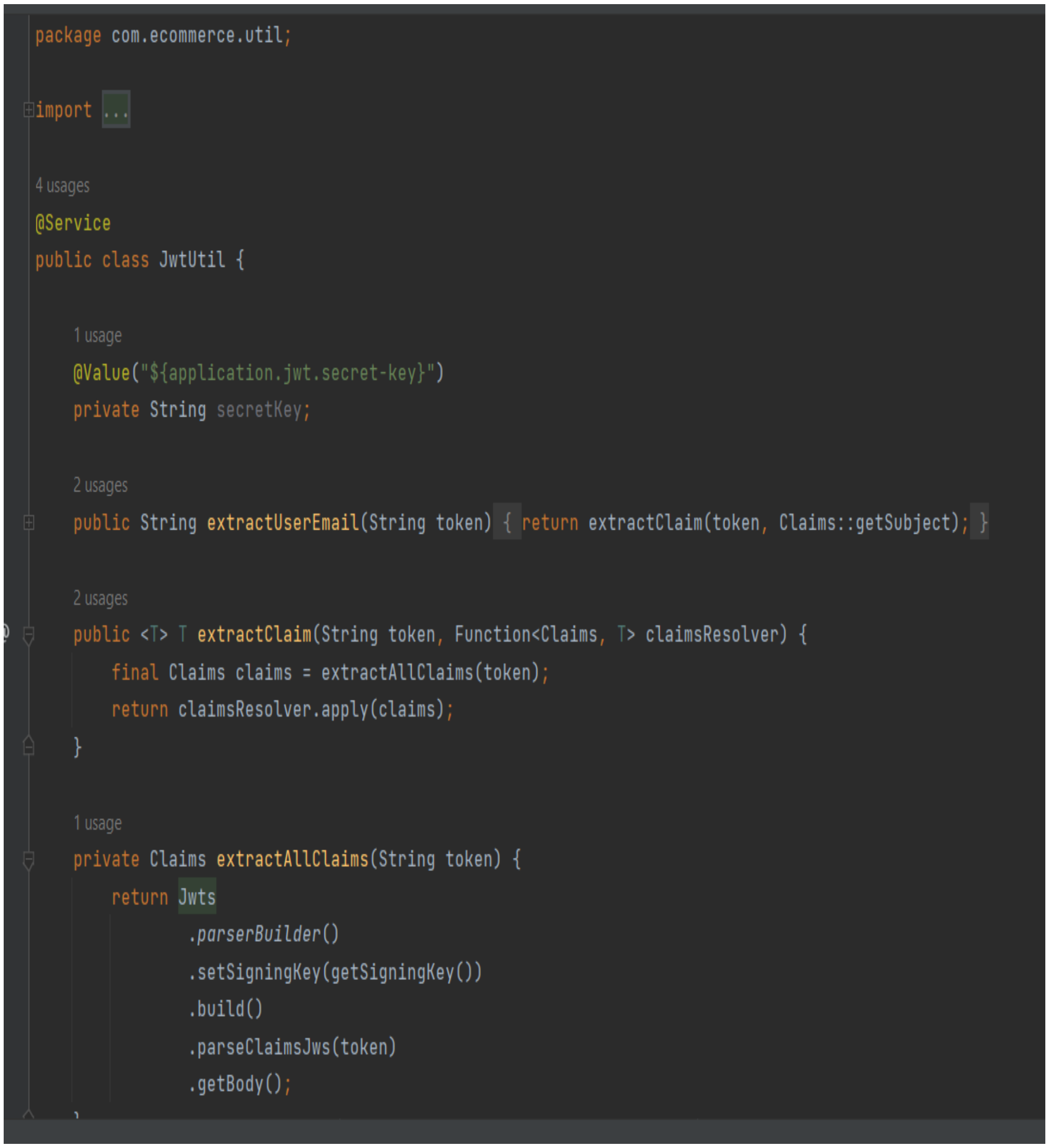
Fig 5.2.4 – Trainer Repository

5.3 SECURITY

5.3.1 JWT TOKEN

The JWT(Json Web Token) is an encrypted signature generated for a specific user logged in. All the further operations are carried on with the help of this token.

If someone get this token they can’t misuse it as it is encrypted strongly and the cannot get the username and password.

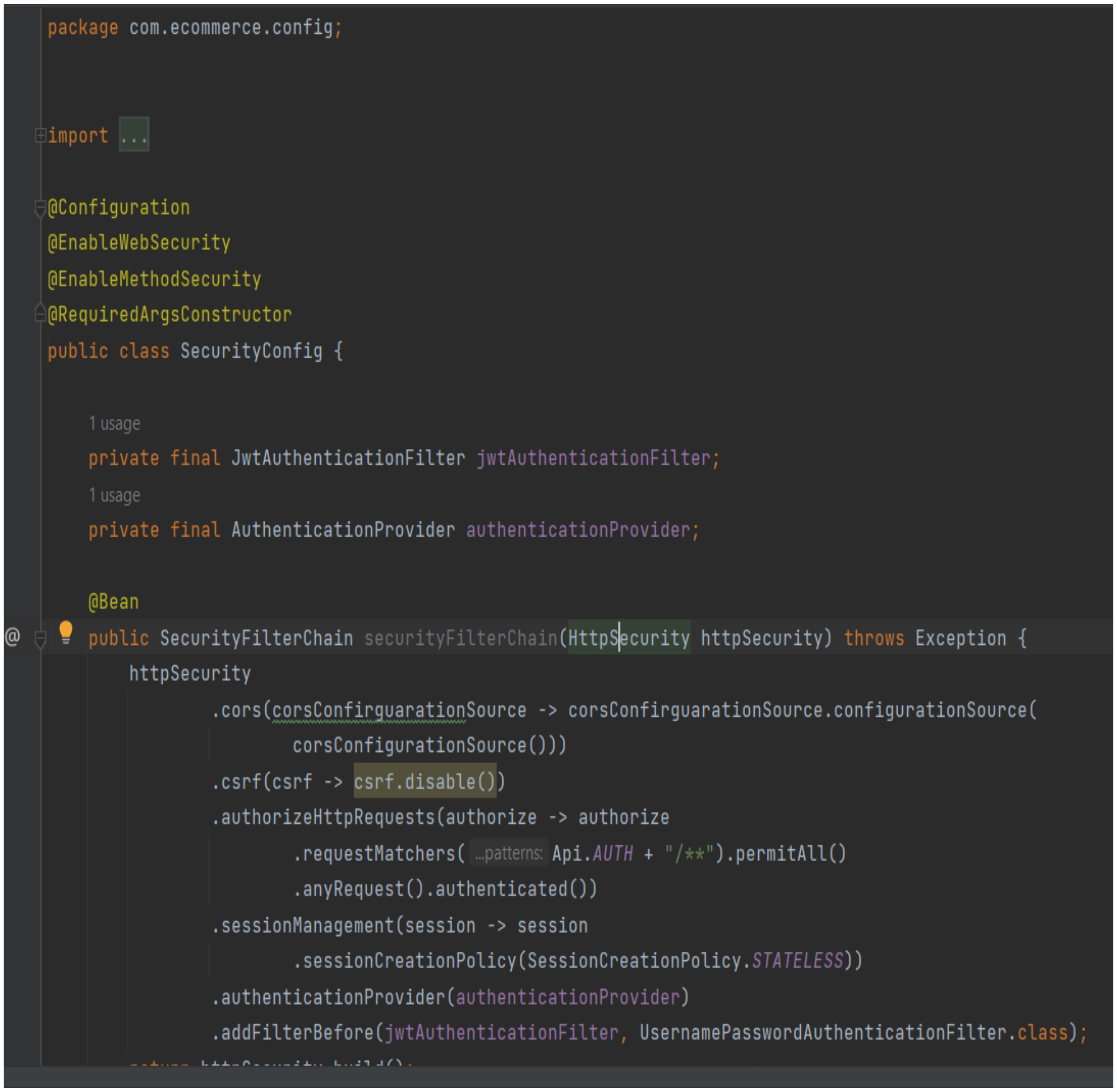


```
package com.ecommerce.util;

import ...

4 usages
@Service
public class JwtUtil {

    1 usage
    @Value("${application.jwt.secret-key}")
    private String secretKey;

    2 usages
    public String extractUserEmail(String token) { return extractClaim(token, Claims::getSubject); }

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

    1 usage
    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSigningKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
```

Fig 5.3.1 – Token Generation

5.3.2 SECURITY CONFIGURATION

Security Configuration file is used to specify which HTTP requests need to be allowed and which one should be denied. With the help of JWTfilter file it will extract the value from the token and authenticate the routes that are specified.

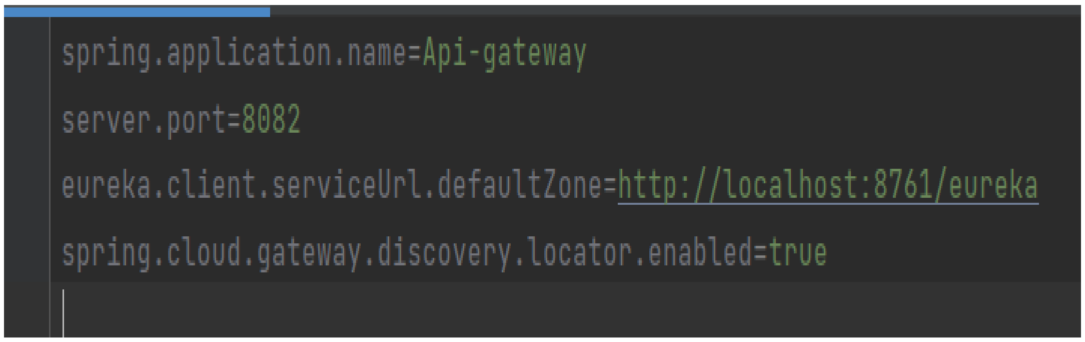


```java
package com.ecommerce.config;


import ...

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    1 usage
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    1 usage
    private final AuthenticationProvider authenticationProvider;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
                .cors(corsConfirguarationSource -> corsConfirguarationSource.configurationSource(
                        corsConfigurationSource()))
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers( ...patterns: Api.AUTH + "/**").permitAll()
                        .anyRequest().authenticated())
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
```

Fig 5.3.2 – Security Configuration

5.4 API GATEWAY

An API gateway is a component of the app-delivery infrastructure that sits between clients and services and provides centralized handling of API communication between them. It also delivers security, policy enforcement, and monitoring and visibility across on-premises, multi-cloud, and hybrid environments. It is where the request for a website is comes in and response for that particular request is sent out.

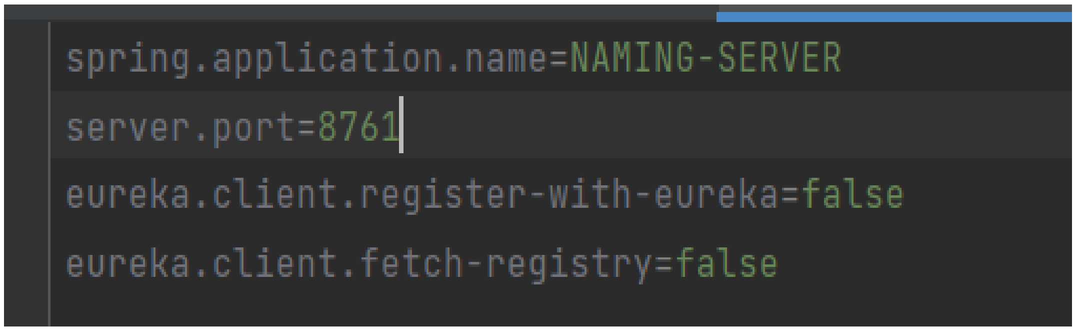


```
spring.application.name=Api-gateway
server.port=8082
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
spring.cloud.gateway.discovery.locator.enabled=true
```

Fig 5.4 – API Gateway

5.1 EUREKA

The Eureka Server acts as a centralized registry, where microservices can register themselves and advertise their availability. It maintains a dynamic directory of available services, allowing other services to locate and communicate with each other. It is used to know the status of our website whether it is in Up or Down.



```
spring.application.name=NAMING-SERVER
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Fig 5.5 – Eureka Server

CHAPTER 6

CONCLUSION

In conclusion, building a REST API project involves careful consideration of various components and principles to ensure a well-designed and functional API. Here are some key points to remember. Follow a consistent and intuitive URL structure, utilize appropriate HTTP methods, and ensure meaningful and consistent response formats. Implement secure authentication mechanisms, such as token-based authentication or OAuth, to protect your API endpoints from unauthorized access. Apply proper authorization rules to control the actions that different users or roles can perform. Validation and Error Handling: Implement thorough input validation to ensure the integrity and consistency of data. Handle errors gracefully by providing meaningful error messages and appropriate HTTP status codes in response to client requests. Include tests for various scenarios, such as positive and negative cases, edge cases, and performance testing, to ensure robustness. Document your API endpoints, including their purpose, input/output formats, and any required headers or parameters. Provide clear and concise examples to guide developers on how to interact with your API effectively. Optimize your API for performance by implementing caching mechanisms, proper database indexing, and efficient query execution. Consider horizontal scaling options, such as load balancing and clustering, to handle increased traffic or future growth. Implement logging to record important events and errors for debugging purposes. Monitor the performance and usage of your API using tools like monitoring dashboards, log analysis, or API analytics platforms. Consider implementing versioning strategies for your API to allow for backward compatibility and smooth upgrades. This ensures that clients can continue using older versions of your API while new versions are introduced. Implement security best practices, such as input validation, secure communication over HTTPS, and protection against common vulnerabilities like SQL injection or cross-site scripting (XSS).

REFERENCES

[1] A. Goldberg and D. Robson, Smalltalk-80 The Language and Its Implemeniaiion, Addison-Wesley, 1983.

[2] B. C. Schwab, K. Hemmaplardh, S. A. Sackett, C. M. Kitto, D. J. Inglis and C. D. Meier, "A Developed Database Management System For Operation and Planning Applications", IEEE Transactions on Power Apparatus and Systems, vol. PAS-104, no. 7, pp. 1637-1643, July 2002 .

[3] H. Daniels and N. Mayur, "More Than Mainframes", IEEE Spectrum, pp. 54-61, August 2007.

[4] W. G. Aref and I. F. Ilyas. "SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees". J. Intell. Inf. Syst., 17(2-3):215-240, 2011.

[5] V. N. Anh and A. Moffat. "Inverted Index Compression Using Word-Aligned Binary Codes". Information Retrieval, 8(1):151-166, 2016.

[6] Fang Kong, Research on the Management of Data System and Honor Archives in China, Nanjing University, 2017.