## BOAT HOUSE BOOKING PORTAL: A Chess Admission Portal

## ADVANCED APPLICATION DEVELOPMENT

**APROJECTREPORT**

***Submittedby***

**KAVIN KUMAR K 727821TUIT049**

**AARTHI DJ 727821TUIT002**

**HARISH RAGAVENDRA A 727821TUIT033**

***In partial fulfillment for the award of the degree***

Of

**BACHELOR OF TECHNOLOGY IN**

**INFORMATIONTECHNOLOGY**

**APRIL 2024**

# BONAFIDE CERTIFICATE

Certified that this project report **“AQUAMARINA : BOAT HOUSE BOOKING APP ”**is the Bonafide work of**“KAVIN KUMAR K,AARTHI DJ, HARISH RAGAVENDRA A** who carried out the project work under my supervision.

## SIGNATURE SIGNATURE

**MENTOR NAME Dr. S. SIAMALA DEVI**

**SUPERVISOR HEAD OF THEDEPARTMENT**

Associate Professor, Associate Professor,

Department of Information Department of Information

Technology, Technology,

Sri Krishna College of Technology,Sri Krishna College of Technology,

Kovaipudur, Kovaipudur,

Coimbatore-641042. Coimbatore-641042.

Certified that the candidates were examined by us in the Project Work viva- voice examination held on ............................... at Sri Krishna College of Technology, Kovaipudur, Coimbatore -641 042.

**INTERNALEXAMINAR EXTERNALEXAMINAR**

# ACKNOWLEDGEMENT

Dedicating this project to the **ALMIGHTY GOD** whose abundant grace andmercies 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 would like to express our deep gratitude to **Dr.J.SHANTHINI**, Head of Computing Science and **Dr.S.SIAMALA DEVI**, Head of Information Technology for her exceptional dedication and care towards success of this project.

We would like to extend our heartfelt gratitude to our Project guide **Dr.A.CHRISTY JEBA MALAR,** Associate Professor, Department of Information Technology 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 us in completing this project work successfully.

# ABSTRACT

This abstract presents a summary of a project focused on the development of a backend system for Boat House Booking Portal: Boat\_House App with functionalities for implementing CRUD (Create, Read, Update, Delete) operations and enhancing security through microservices. Boat House Booking Portal has discovered the joy of personalized gifting with bespoke gift portal, where every present tells a unique story.Theprojectcenters arounddesigningandimplementing a robust backend systemthat enables users toperform CRUD operations, allowing them to create, read, update, and delete video within the application. This backend system leverages microservices to enhance scalability, modularity, and security. It emphasizes the importance of a well-structured API and data management to facilitate these actions while maintaining ease of use and minimizing the learning curve for users.It also underscores the need for comprehensive data validation and robust error handling mechanisms to ensure the accuracy and reliability of information management. Security is a central focus, and the project implements microservices to manage access controls, user authentication, and authorization measures. This approach helps protect sensitive data and restrict unauthorized access, safeguarding the integrity of the product database within the Boat House Booking Portal :Boat House portal.

# TABLEOFCONTETS

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER.NO** | **TITLE** | **PAGE.NO** | |
|  | **ABSTRACT** | **IV** | |
|  | **LISTOFFIGURES** | **VI** | |
|  | **LISTOFABBREVIATIONS** | **VII** | |
| 1 | INTRODUCTION | 01 | |
| 1.1 | PROBLEMSTATEMENT | 02 | |
| 1.2 | OVERVIEW | 02 | |
| 1.3 | OBJECTIVE | 02 | |
| 2 | BACKENDSYSTEMSPECIFICATION | 04 | |
| 2.1 | POSTGRESQL | 04 | |
| 2.2 | RESTAPI | 06 | |
| 2.3 | SPRINGBOOT | 06 | |
| 3. | SYSTEMARCHITECTURE | 08 | |
| 3.1 | BACKEND | 08 | |
| 4 IMPLEMENTATION&FUNCTIONALITIES | | | 12 |
| 4.1 APIREQUEST | | | 12 |
| 4.2 CRUDOPERATIONS | | | 14 |
| 4.3 DATARETRIEVELPROCESS | | | 19 |
| 4.4 DATAUPDATEPROCESS | | | 20 |
| 4.5 SECURITYANDAUTHENTICATION | | | 21 |
| 5 CONCLUSIONANDFUTURESCOPE | | | 27 |
| 5.1 CONCLUSION | | | 27 |
| 5.2 FUTURESCOPE | | | 28 |
| 6 REFERENCES | | | 29 |

|  |  |  |
| --- | --- | --- |
|  | **LISTOFFIGURES** |  |
| **FIGURENO** | **FIGURENAME PAGE** | **NO** |
| 2.1 | PostgreSQL | 04 |
| 3.1 | BackendSystemArchitecture | 08 |
| 4.1 | APIRequestFlowchart | 12 |
| 4.2 | DataRetrievalProcess | 19 |
| 4.3 | DataUpdateFlowchart | 20 |
| 4.4 | SecurityandAuthenticationFlowchart | 21 |

**LISTOFABBREVIATIONS**

### ABBREVIATIVE ABBERVIATION

REST RepresentationalStateTransfer

API ApplicationProgrammingInterface

JSON JavaScriptObjectNotation

CA Boat House Booking Portal

# CHAPTER 1 INTRODUCTION

The **Boat House Booking Portal Admission Portal** stands as a gateway to a world of possibilities, where aspiring chess enthusiasts can embark on their journey. Much like a well-played opening move, our portal is designed to facilitate seamless admissions for students who seek to hone their chess skills.

At the **Boat House Booking Portal**, we recognize that admission is not merely about paperwork and formalities. It’s an opportunity to express passion, dedication, and potential. With this ethos at our core, we have meticulously crafted an admission process that transcends the mundane.

Our bespoke approach ensures that each applicant’s journey is unique. Here’s how we stand apart:

1. **Tailored Pathways**: Just as a chess player strategizes their moves, we offer customized pathways for different skill levels. Whether you’re a novice or a seasoned player, our admission options cater to your individual needs.
2. **Personalized Guidance**: Our team of chess experts serves as your mentors, guiding you through the process. From selecting the right program to understanding prerequisites, we’re here to ensure your admission journey is smooth.
3. **Memorable Moments**: Chess is more than a game; it’s a collection of memories. Our admission portal captures these moments. Whether it’s sharing your chess achievements or expressing your aspirations, we celebrate your uniqueness.
4. **Depth of Commitment**: We don’t just admit students; we invest in their growth. Our commitment extends beyond the admission process. As you step onto the board, we’re there to support your progress.
5. **Individuality Reigns**: In a world of standardized forms, we celebrate individuality. Your application isn’t just paperwork; it’s a canvas where you paint your chess story.

So, welcome to the **Boat House Booking Portal Admission Portal**—where every move matters, and every student is a grandmaster in the making!

**1.1 Overview**

The **Boat House Booking Portal Admission Portal** stands as a premier gateway for aspiring chess enthusiasts. Our mission is to facilitate seamless admissions, much like a well-executed opening move on the chessboard. Here’s a glimpse of what sets us apart:

* **Tailored Pathways**: Just as a chess player strategizes their moves, we offer customized admission pathways for different skill levels. Whether you’re a novice or a seasoned player, our options cater to your individual needs.
* **Personalized Guidance**: Our team of chess experts serves as your mentors, guiding you through the process. From selecting the right program to understanding prerequisites, we’re here to ensure your admission journey is smooth.
* **Memorable Moments**: Chess is more than a game; it’s a collection of memories. Our admission portal captures these moments. Whether it’s sharing your chess achievements or expressing your aspirations, we celebrate your uniqueness.
* **Depth of Commitment**: We don’t just admit students; we invest in their growth. Our commitment extends beyond the admission process. As you step onto the board, we’re there to support your progress.
* **Individuality Reigns**: In a world of standardized forms, we celebrate individuality. Your application isn’t just paperwork; it’s a canvas where you paint your chess story.

**1.2 Objective**

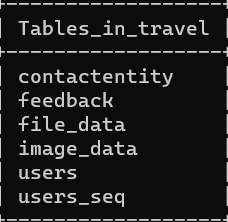
The **Boat House Booking Portal Admission Portal** aims to revolutionize the admission experience. Our comprehensive online platform empowers applicants to express their passion, dedication, and potential. Through a user-friendly interface and personalized options, we create a space where every move matters. Our commitment to exceptional service ensures that each student’s journey—from application to acceptance—is memorable and gratifying. Welcome to a world where chess dreams come alive!

# CHAPTER2

**BACKENDSYSTEM 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.

### POSTGRESQL



#### Fig2.1MySQL

Local storage is a type ofweb storagefor storing data on the client side ofaweb browser. It allows websites to store data ona 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 informationsuch as preferences and settings, or to store data that is not meant to be shared with other websites.

It is also used to cache data to improve the performance of a website. Local storageissupportedbyall modernwebbrowsers,includingchrome, Firefox,Safari, and Edge. It is accessible through the browser’sJavaScriptAPI. 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 Storageand Indexed. The data stored in local storageis persistent and can be accessedby 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 localstorage is its reliability. Unlike server-side storage, which can be affected by network outages or other server issues, local storageis 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.

### RESTAPI

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 andserver to provide additional functionality, such as load balancing, caching, or security.

### SPRINGBOOT

Spring Boot is an open-source Java framework that simplifies the developmentof 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 theneed 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 classpath 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.

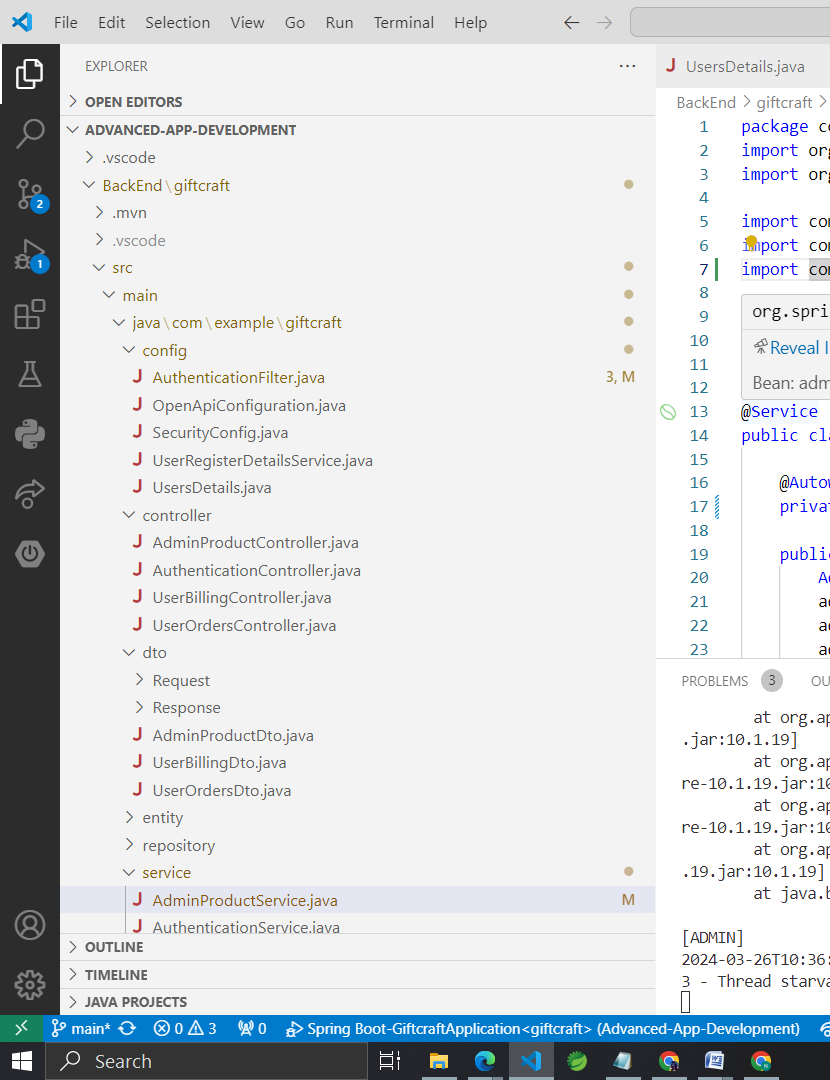
Actuator: Spring Boot Actuator provides out-of-the-box monitoring and management endpoints for the application. It offers metrics, health checks, logging, and other management features, making it easier to monitor and manage the application in production environments.

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.

# CHAPTER 3 SYSTEMARCHITECTURE

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.

### BACKEND

****

#### Fig3.1BackendSystemArchitecture

Boat House Booking Portal :Customized Gif application, the backend layer is built upon the Spring Boot framework, a Java-based solution renowned for its capacity to simplify the development of resilient and scalable web applications. Spring Boot brings to the tablea comprehensive array of features and libraries, offering streamlined solutions for managing HTTP requests,datapersistence, implementing robust securitymeasures, and seamlessly integrating with external systems. Within the application, the backend takes on the primary responsibility of crafting RESTful APIs. These APIs are meticulously designed to empower CRUD (Create, Read, Update, Delete) operations, catering to the dynamic world of photograph details management. Furthermore, the backend is equipped to handle user management and authentication, ensuring a secure and personalized user experience for both administrators and customers. In pursuit of enhanced security and modularity, the backend is strategically structured according to the principles of springboot architecture.

#### SpringBoot:

Spring Boot is a Java framework that simplifies the process of building enterprise-grade applications. It provides a robust set of features and conventions for developing backend systems, including dependency management, configuration, and automatic setup. Spring Boot follows the principle of convention over configuration, reducing the amount of boilerplate code required.

### RESTAPI:

The backend of the Open Library systemexposes a RESTful API that allows the frontend to communicate with the server. REST (Representational State Transfer) is an architecturalstylefordesigningnetworkedapplications.ItusesstandardHTTPmethods (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 orchestratethe 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:

Serviceswithinthe 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 repositoriesare instrumental in storing and retrieving customize gift and orders-related data from the database, ensuring the persistence and accessibility of vital information.

#### DataTransferObjects(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:

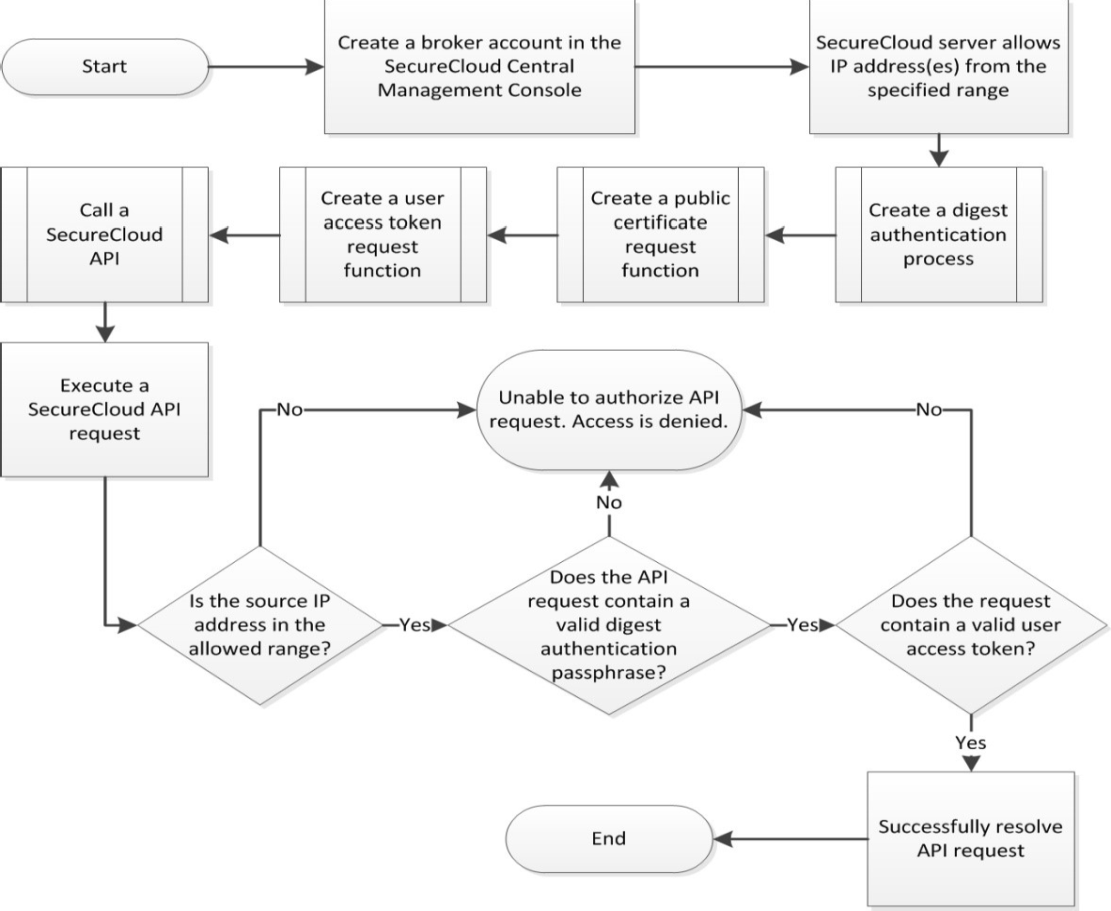
Security measures are a top priority within the application. Authentication and authorization protocols are diligently implemented to safeguard user data and system integrity. A robust security framework, such as Spring Security, is employed to manage user authentication and access control. It offers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

# CHAPTER 4

# IMPLEMETATIONANDFUNCTIONALITY

The Boat House Booking Portal :Customized Gifapplication'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 user-friendly interface simplifiestheir tasks, reducing the learning curve. Securityisatpriority,withstringentaccesscontrolsanduserauthenticationmeasures protecting sensitive data.

#### APIRequest



**Fig4.1 RESTAPIflowchart**

A Representational State Transfer (REST) API plays a pivotal role in the architecture of theApplication, 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 theapplication, 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 corefunctionalities are exposed and extended, ultimately contributing to a seamless and versatile user experience.

### CRUDOPERATION

Intheapplication,theimplementationofCRUD(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 comprehensivedetails,whichare then validated for accuracy andcompleteness 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:

**ControllerClass:**

#### package com.example.giftcraft.controller;

#### import org.springframework.beans.factory.annotation.Autowired;

#### import org.springframework.http.ResponseEntity;

#### import org.springframework.web.bind.annotation.GetMapping;

#### import org.springframework.web.bind.annotation.PostMapping;

#### import org.springframework.web.bind.annotation.RequestBody;

#### import org.springframework.web.bind.annotation.RequestMapping;

#### import org.springframework.web.bind.annotation.RestController;

#### import com.example.giftcraft.dto.Request.AuthenticationRequest;

#### import com.example.giftcraft.dto.Request.RegisterRequest;

#### import com.example.giftcraft.dto.Response.AuthenticationResponse;

#### import com.example.giftcraft.entity.UserRegister;

#### @RestController

#### @RequestMapping("/auth")

#### @RequiredArgsConstructor

#### public class AuthenticationController {

#### @Autowired

#### private UserRegisterRepository userRegisterRepository;

#### private final AuthenticationService service;

#### @PostMapping("/register")

#### public ResponseEntity<String> register(

#### @RequestBody RegisterRequest request

#### ) {

#### service.register(request);

#### return ResponseEntity.ok("Registered Successfully");

#### }

#### @GetMapping("/getregister")

#### public List<UserRegister> getregister()

#### {

#### return userRegisterRepository.findAll();

#### 

#### }

#### @PostMapping("/authentication")

#### public ResponseEntity<AuthenticationResponse> authenticate(@RequestBody AuthenticationRequest request)

#### {

#### return ResponseEntity.ok(service.authenticate(request));

#### 

#### }

#### EntityClass:

#### package com.example.giftcraft.entity;

#### import com.example.giftcraft.entity.enumerate.Role;

#### import jakarta.persistence.Entity;

#### import jakarta.persistence.EnumType;

#### import jakarta.persistence.Enumerated;

#### import jakarta.persistence.GeneratedValue;

#### import jakarta.persistence.GenerationType;

#### import jakarta.persistence.Id;

#### import jakarta.persistence.Table;

#### import lombok.AllArgsConstructor;

#### import lombok.Builder;

#### import lombok.Data;

#### import lombok.NoArgsConstructor;

#### @Entity

#### @Data

#### @AllArgsConstructor

#### @NoArgsConstructor

#### @Builder

#### @Table(name = "\_user")

#### public class UserRegister {

#### @Id

#### @GeneratedValue(strategy = GenerationType.IDENTITY)

#### private Long userid;

#### private String username;

#### private String emailid;

#### private String password;

#### private String mobileno;

#### @Enumerated(EnumType.STRING)

#### private Role roles;

#### }

#### ServiceClass:

package com.example.giftcraft.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import com.example.giftcraft.dto.Request.AuthenticationRequest;

import com.example.giftcraft.dto.Request.RegisterRequest;

import com.example.giftcraft.dto.Response.AuthenticationResponse;

import com.example.giftcraft.entity.UserRegister;

import com.example.giftcraft.entity.enumerate.Role;

import com.example.giftcraft.repository.UserRegisterRepository;

import lombok.RequiredArgsConstructor;

@Service

@RequiredArgsConstructor

public class AuthenticationService {

@Autowired

private final UserRegisterRepository userRegisterRepository;

private final PasswordEncoder passwordEncoder;

private final JwtService jwtService;

private final AuthenticationManager authenticationManager;

@SuppressWarnings("null")

public AuthenticationResponse register(RegisterRequest request) {

Role role;

if (request.getRoles() != null) {

role = request.getRoles();

} else {

role = Role.USER;

}

UserRegister user = UserRegister.builder()

.username(request.getUsername())

.emailid(request.getEmailid())

.mobileno(request.getMobileno())

.password(passwordEncoder.encode(request.getPassword()))

.roles(role)

.build();

userRegisterRepository.save(user);

String jwtToken = jwtService.generateToken(request.getUsername());

return AuthenticationResponse.builder()

.token(jwtToken)

.build();

}

public AuthenticationResponse authenticate(AuthenticationRequest request) {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(

request.getUsername(),

request.getPassword()

)

);

String jwtToken = jwtService.generateToken(request.getUsername());

return AuthenticationResponse.builder()

.token(jwtToken)

.build();

}

}

#### RepositoryInterface:

package com.example.giftcraft.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.stereotype.Repository;

import com.example.demo.entity.Users;

import java.util.List;

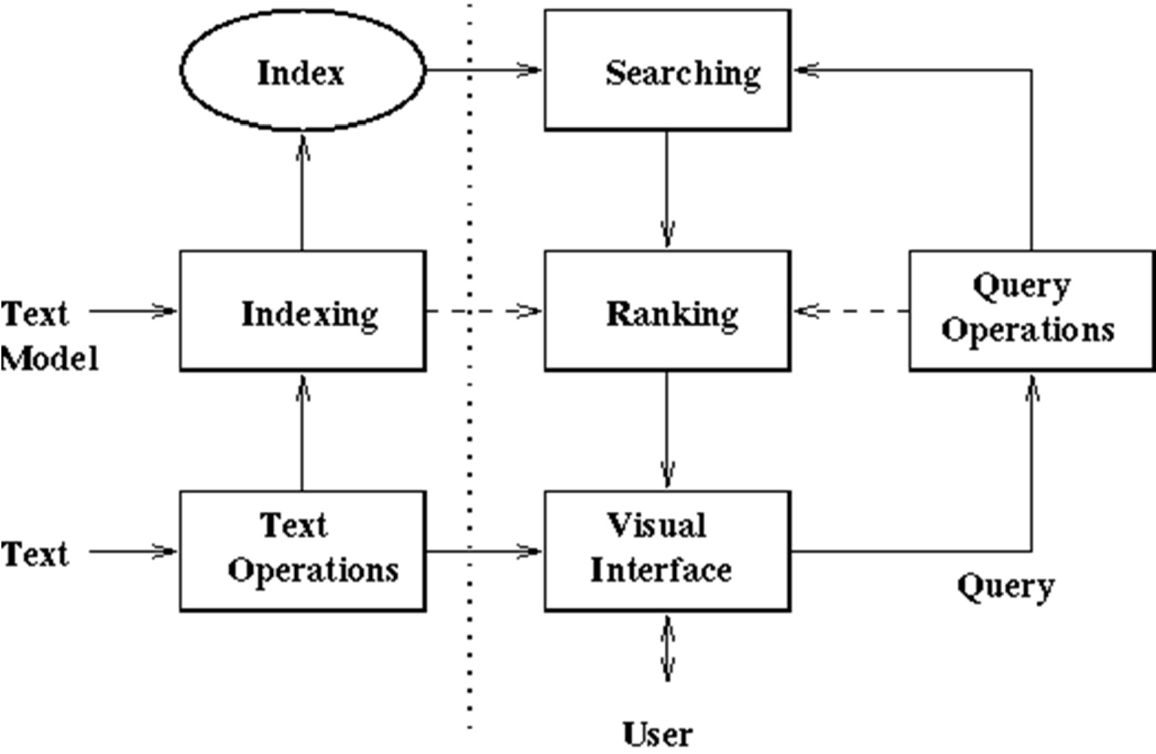
@Repository

public interface UsersRepository extends JpaRepository<Users, Integer> {

Optional<Users>findByEmail(String email);

}

### DATARETRIEVELPROCESS

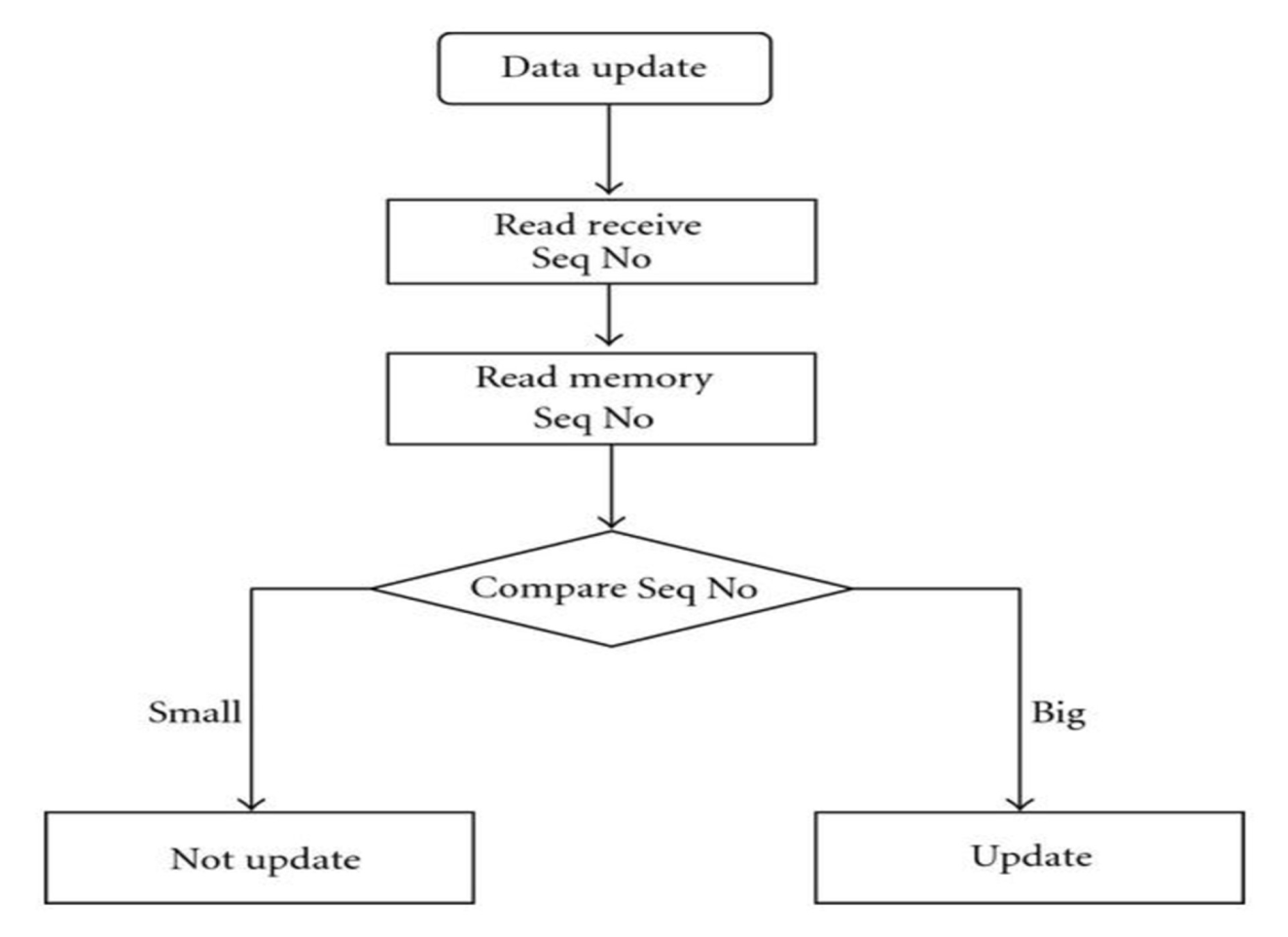


#### Fig4.2DataRetrievalProcess

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 backends routing system directs it to the appropriate endpoint handler based on the URL and HTTP method.

Before proceeding, the backend verifies user authentication and confirms if the necessary permissions exist to access the requested data. Following authorization, a database query is generated based on the request, specifying the criteria for dataretrieval.

* 1. **DATAUPDATEPROCESS**



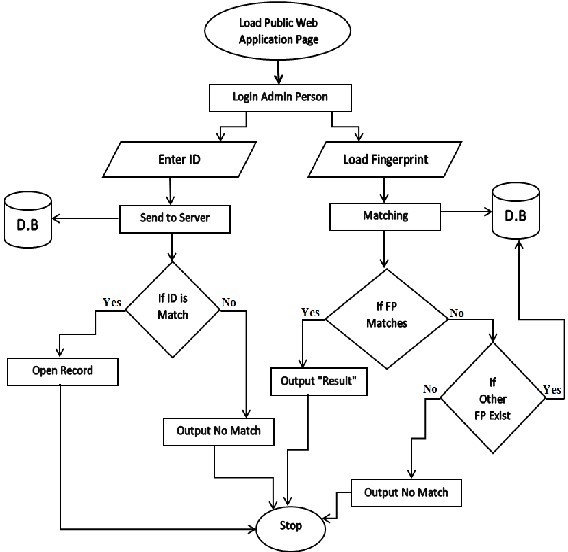
## Fig4.3DataUpdateFlowchart

The data update process within the backend of Applicationisacrucialmechanismthat allowsusersandadministrators to enactchanges 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 phaseplays apivotal role in maintaining data integrity.

### SECURITYANDAUTHENTICATION



#### Fig4.4SecurityAndAuthenticationFlowchart

Security and authentication lie at the heart of theapplication'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 againstcommon security threats, such as cross-site scripting and SQL injection, is in place.

Userawareness 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:

**JwtAuthenticationFilterClass:**

#### 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);

#### }

#### JwtServiceClass:

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);

}

}

# CHAPTER 5 CONCLUSION

This chapter discusses the project's conclusions and the learning we learnt by taking over the project.

### 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 Auth, to protect the 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 the 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 the API effectively. Optimize the 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 the API using tools like monitoring dashboards, log analysis, or API analytics platforms. Consider implementing versioning strategies for the API to allow for backward compatibility and smooth upgrades. This ensures that clients can continue using older versions of the 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).

### FUTURESCOPE

The future scope of the project is incredibly promising, poised for transformative growth and innovation. Technological advancements are set to revolutionize the industry, with the integration of virtual and augmented reality offering participants unparalleled immersive experiences, bringing them closer to the natural world than ever before.

## CHAPTER 6

## REFERENCES

* + 1. ExpertPhotography, “The Ultimate Guide to Travel Photography (113 Best Tips)”, provides a comprehensive guide on travel photography, covering everything from what travel photography is, how to start, planning for road trip photography, and more1.
    2. NOMADasaurus, “21 Travel Photography Tips (Secrets from the Pros)”, offers 21 practical tips for beginners and intermediates based on the authors’ experience of working in the industry for 8 years2.
    3. My Global Viewpoint, “25 Travel Photography Tips for Beginners to Take Travel Photos”, provides general travel photography tips for beginners3.
    4. Pixpa, “Travel Photography - How to Become a Travel Photographer”, shares four top useful tips on how to get started in travel photography4.
    5. National Geographic, “Travel Photography Tips”, provides tips from photographers for National Geographic who spend a lot of time doing research5.
    6. Digital Photography School, “13 Travel Photography Tips for Breathtaking Images”, offers tips on capturing travel shots around the world6.
    7. The Wandering Lens, “24 Destinations To Ignite Love of Photography”, lists 24 destinations around the world where it can take the camera on a romantic escape7.