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

#### ZENYOGA

**ADVANCED APPLICATION DEVELOPMENT**

**A PROJECT REPORT**

***Submitted by***

**IBRAHIMI (Register No: 727821TUIT037)**

**MANOJ KUMARN (Register No: 727821TUIT57)**

**MOHAMED NOUFAL ANSARIT (Register No: 727821TUIT060)**

***in partial fulfilment for the award of the degree***

***of***

#### BACHELOR OF TECHNOLOGY

**IN**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**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 **“ZENYOGA”** is the Bonafide work of **“IBRAHIMI, MANOJ KUMARN, MOHAMED NOUFAL ANSARIT”**who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Ms. S. MADUMIDHA Dr. SIAMALA DEVI S**

**SKILL DEVELOPMENT ENGINEER HEAD OF THE DEPARTMENT**

Assistant Professor, Associate Professor,

Department of Information Department of Information

Technology, Technology,

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

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 -641042.

**INTERNAL EXAMINAR EXTERNAL EXAMINAR**

**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 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 their exceptional dedication and care towards success of this project.

We would like to extend our heartfelt gratitude to our Project guide **Ms.S.Madumidha,** Assistant 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 to 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 a Voting & Polling System with functionalities for implementing CRUD (Create, Read, Update, Delete) operations and enhancing security through microservices. The project aims to provide an efficient and secure platform for managing voting and polling processes within the context of online elections and surveys. The abstract begins by highlighting the significance of effective Yoga and poll management in the context of online democratic processes, emphasizing the challenges associated with manual methods and the need for an automated system to streamline operations and bolster security through a microservices architecture. The project centers around designing and implementing a robust backend system that enables users to perform CRUD operations, allowing them to create, read, update, and delete polls and voting data 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. The abstract discusses the key features and functionalities of the backend system, including the ability to handle poll and Yoga details such as questions, options, and participant data. 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 voting and polling data and restrict unauthorized access, safeguarding the integrity of the system within the online voting and polling application.

**TABLE OF CONTETS**

**CHAPTER.NO TITLE PAGE.NO**

**ABSTRACT IV**

**LIST OF FIGURES VI**

**LIST OF ABBREVIATIONS VII**

1 INTRODUCTION 01

1.1 PROBLEM STATEMENT 02

1.2 OVERVIEW 02

1.3 OBJECTIVE 03

2 BACKEND SYSTEM SPECIFICATION 04

2.1 MYSQL 04

2.2 REST API 05

2.3 SPRING BOOT 06

3 SYSTEM ARCHITECTURE 08

3.1 BACKEND 08

4 IMPLEMENTATION AND FUNCTIONALITIES 12

4.1 API REQUEST 12

4.2 CRUD OPERATIONS 13

4.3 DATA RETRIEVEL PROCESS 19

4.4 DATA UPDATE PROCESS 20

4.5 SECURITY AND AUTHENTICATION 21

4.6 MICROSERVICE 27

5 CONCLUSION AND FUTURE SCOPE 28

5.1 CONCLUSION 28

5.2 FUTURE SCOPE 29

6 REFERENCES 30

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **FIGURE NAME** | **PAGE NO** |

2.1 Mysql 04

3.1 Backend System Architecture 08

4.1 API Request Flowchart 12

4.2 Data Retrieval Process 19

4.3 Data Update Flowchart 20

4.4 Security and Authentication Flowchart 21

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| **ABBREVIATIVE** | **ABBERVIATION** |
| REST | Representational State Transfer |
| POJO | Plain Old Java Object |
| API | Application Programming Interface |
| JSON | JavaScript Object Notation |

**CHAPTER 1**

**INTRODUCTION**

In today's rapidly evolving digital landscape, the realm of democratic participation and public opinion gathering has seen a substantial transformation with the widespread adoption of zenyogas. These systems have become a cornerstone for engaging citizens, organizations, and communities in a dynamic and accessible digital arena, where their voices and preferences can be expressed and recorded with ease. As the demand for efficient, secure, and user-friendly methods of conducting elections and surveys continues to rise, the role of a robust backend application for online voting and polling becomes increasingly pivotal.

This introduction sets the stage for a comprehensive exploration of the backend application's central role in ensuring the seamless operation and secure management of critical data within the zenyoga. By delving into the intricacies of backend functionalities, which encompass data processing, user authentication, and stringent security protocols, this study aims to provide a comprehensive understanding of the fundamental components and mechanisms that underpin the smooth functioning of the online voting and polling backend application.

With a keen focus on enhancing user experience, preserving the integrity of voting and polling data, and optimizing the processes of democratic engagement, this exploration emphasizes the critical role of the backend application in establishing a resilient and efficient infrastructure for the online voting and polling ecosystem. In response to the growing demand for innovative and accessible methods of gauging public opinion and conducting elections, zenyogas have emerged as ensuring the security and confidentiality of data, while fostering a user-centric environment that encourages meaningful democratic participation and decision-making.

This introduction aims to emphasize the crucial role of the zenyoga's backend application in streamlining the electoral and survey processes, safeguarding the privacy of nurturing a user-centric environment that encourages meaningful participation and decision-making among citizens and organizations alike.

* 1. **PROBLEMSTATEMENT**

An zenyoga is a digital platform designed to enable the conduct of elections and surveys over the internet. It offers a virtual environment for citizens, organizations, and polling administrators to engage, share opinions, and participate in the electoral and survey processes?

**1.2 OVERVIEW**

This report delves into the outcomes of the development efforts focused on the backend of the Zenyoga. The project harnesses advanced backend technologies to establish a robust platform that simplifies the management of elections, updates polling details, and enhances the overall user experience. The Zenyoga, built with a modern technology stack, aims to provide a seamless and efficient online platform for citizens, organizations, and polling administrators. With a primary focus on backend development, the team has strived to create a cohesive ecosystem that streamlines the electoral and survey processes. The key objectives revolved around developing a reliable and scalable backend infrastructure that powers the Zenyoga. This infrastructure encompasses various functionalities, such as Yoga and poll management, data control, and user interactions, ultimately resulting in a user-friendly experience. The backend work for the Zenyoga includes the implementation of advanced database systems, robust API endpoints, and secure authentication mechanisms. These components work in harmony to provide a dynamic platform that allows administrators to effortlessly conduct and manage elections and surveys while ensuring that participants receive a seamless and informative experience. In summary, the backend development work for the Zenyoga has been instrumental in shaping a user-friendly and feature-rich platform. The system's success can be attributed to the meticulous approach to database management, API design, and security, all of which contribute to a superior user experience. Anticipating further improvements to the system, the aim is to provide users with an even more delightful and efficient platform for all their democratic engagement and survey needs.

**1.3 OBJECTIVE**

Crafting an intuitive and user-friendly interface to enable users, whether citizens or administrators, to navigate and interact with the Zenyoga effortlessly. The design approach focuses on minimizing the learning curve and enhancing the overall user experience. Empowering users to conduct elections and surveys with ease, providing essential details like questions, options, and participant data in a straightforward manner. Implementing validation mechanisms to ensure the accuracy and completeness of data during the creation and management of polls and Yogas.

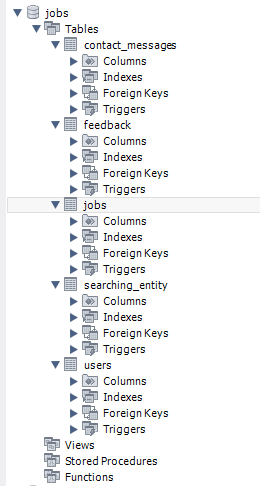
The Zenyoga offers users the capability to update polling and voting information as required. This functionality allows for the modification of details such as questions and options, ensuring that changes in survey content are seamlessly reflected in real-time. A mechanism has been developed to enable the straightforward and controlled removal of polls and Yogas from the system when necessary. Users can efficiently delete polls and Yogas, ensuring precise tracking of survey availability and election data.

**CHAPTER 2**

**BACKEND SYSTEM SPCIFICATION**

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.

**2**.**1 MYSQL**

****

**Fig 2.1 MySQL**

Local storage is a type of web storage for storing data on the client side of a webbrowser.Itallowswebsitestostoredataona user’s computer, which can then beaccessed by the website again when the user returns. Local storage is a more securealternative to cookies because it allows websites to store data without having to send itback and forth with each request. Local storage is a key-value pair storage mechanism,meaning it stores data in the form of a key and corresponding value. It is similar to adatabase table in that it stores data in columns and rows, except that local storage storesthe data in the browser rather than in a database.

Local storage is often used to store userinformationsuch as preferences and settings, or to store data that is not meant to beshared with other websites.

It is also used to cache data to improve the performance of awebsite.Localstorageissupportedbyallmodernwebbrowsers,including chrome, Firefox, Safari, and Edge. It is accessible through the browser’s JavaScript API. Localstorage 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 otherwebsites.

Local Storage is a great way to improve the performance of a website by cachingdata. Local storage in web browsers allows website data to be stored locally on the user’scomputer.Itisawayofpersistentlystoringdataontheclientside,whichisnotsenttothe server with each request. This allows users to store data such as preferences, logininformation,andformdatawithoutneedingtosendittoaserver.

Itistypicallystoredinabrowser’s cookie file, but it can also be stored in other locations such as HTML5 LocalStorage and Indexed. The data stored in local storage is persistent and can be accessedby the website even if the user closes the browser or navigates to another page. It is agreat way for websites to store user-specific data, as it is secure, reliable, and fast. It isalso a great way for developers to store data that does not need to be sent to the serverwitheachrequest.

One of the key benefits of using local storage is its reliability. Unlike server-sidestorage, which can be affected by network outages or other server issues, local storage isstored locally on the user’s machine, and so is not affected by these issues. Anotheradvantage of local storage is its speed. Because the data is stored locally, it is accessedquickly, as there is no need to send requests to a server.

This makes it ideal for storingdata that needs to be accessed quickly, such as user preferences or session data. Localstorage is also secure, as the data is stored on the user’s machine and not on a server. This means that the data is not accessible by anyone other than the user, making it a goodchoiceforstoringsensitiveinformation.

**2.2 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.

**2.3 SPRING BOOT**

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 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.

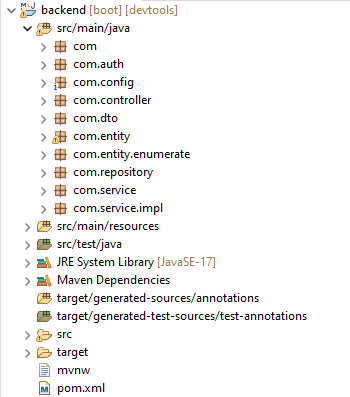
Community Support: Spring Boot has a large and active community of developers, which means there is extensive documentation, tutorials, and community-driven support available.

**CHAPTER 3**

**SYSTEMARCHITECTURE**

The Zenyoga 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.

**3.1 BACKEND**

****

**Fig 3.1 Backend System Architecture**

Zenyoga, 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 table a comprehensive array of features and libraries, offering streamlined solutions for managing HTTP requests, data persistence, implementing robust security measures, and seamlessly integrating with external systems. Within the Zenyoga, the backend takes on the primary responsibility of crafting RESTful APIs.These APIs are meticulously designed to facilitate CRUD (Create, Read, Update, Delete) operations, catering to the dynamic realm of zenyoga management. Furthermore, the backend is equipped to manage user interactions and authentication, ensuring a secure and personalized user experience for both administrators and participants. In pursuit of enhanced security and modularity, the backend is strategically structured based on the principles of microservices architecture. This approach enables the creation of self-contained, independently deployable services that work together seamlessly to support various aspects of the Zenyoga. This design inherently promotes scalability, adaptability, and long-term system stability.

**Spring Boot:**

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.

**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 Zenyoga, 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 Zenyoga 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 products, handle user authentication, and other application-specific functionalities, ensuring a seamless user experience.

**Repositories:**

In the Zenyoga, 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 product-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 Zenyoga. These objects define the structure and format of data shared in API requests and responses. DTO’s are employed to represent product details, user information, and other relevant data that is transferred between the frontend and backend, ensuring seamless communication and data consistency.

**Security:**

Security measures are a top priority within the Zenyoga. 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.

**Database:**

The Zenyoga utilizes a MySQL database management system. This relational database is responsible for storing and managing persistent data associated with polls, Yogas, users, and other relevant entities within the voting and polling domain. The database schema is meticulously designed to align with the system's data model, guaranteeing efficient data storage, retrieval, and query capabilities.

**Tables and Relationships:**

The database comprises multiple tables, with each one representing distinct entities or data structures within the Zenyoga. These tables establish relationships to uphold data consistency and integrity. Common relationship types include one-to-one, one-to-many, and many-to-many, which contribute to efficient data storage and retrieval within the system.

**Schema Design and SQL Queries:**

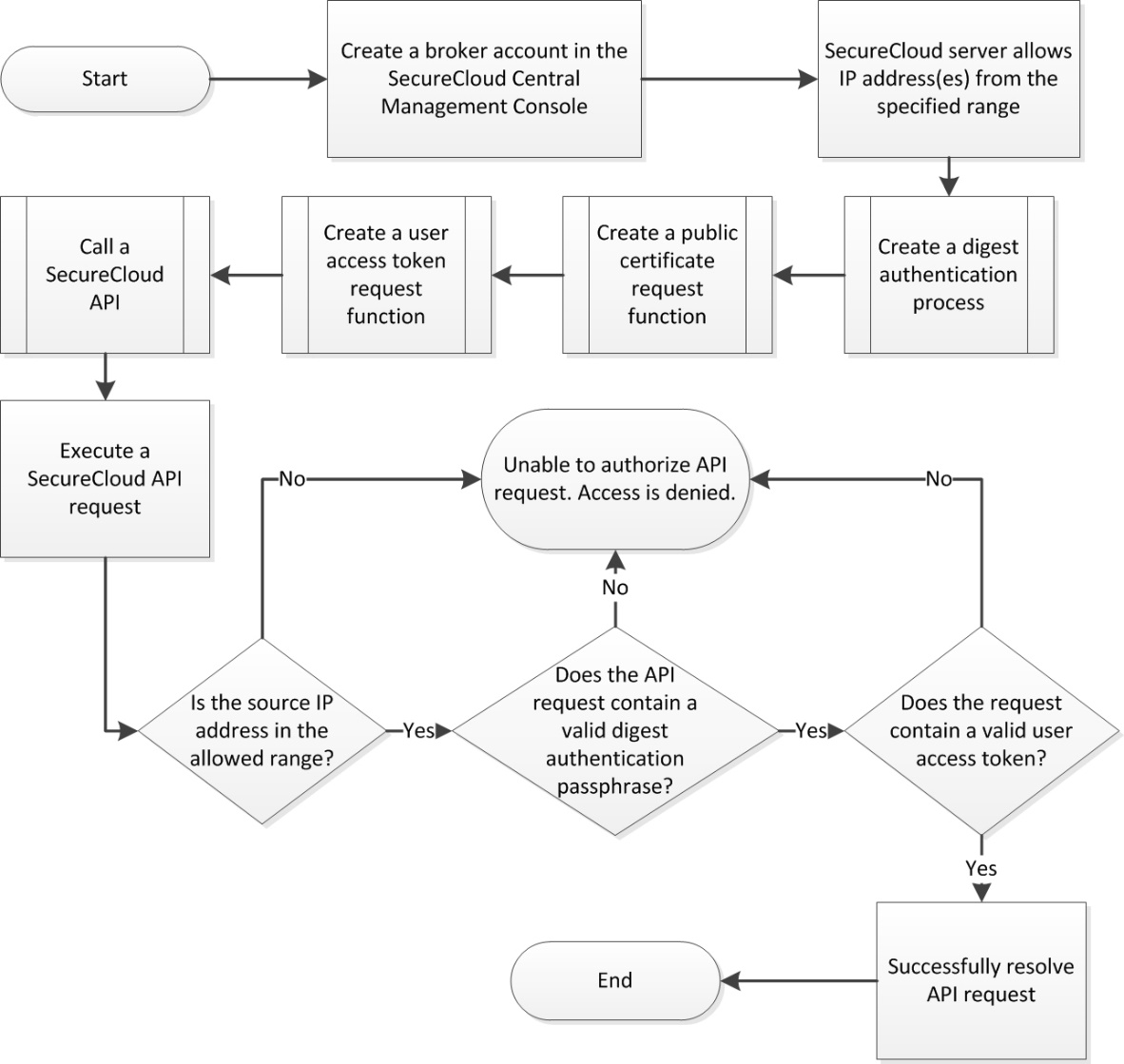
The database schema is thoughtfully designed to represent data relationships logically. It defines tables, columns, primary keys, foreign keys, constraints, and indexes for data integrity and performance.

**CHAPTER 4**

**IMPLEMETATION AND FUNCTIONALITY**

The backend of the Zenyoga serves as the core of efficient polling and voting management. It offers administrators a comprehensive set of tools for creating, updating, and deleting polls and Yogas, fostering a user-friendly experience and minimizing the learning curve. Stringent data validation and error-handling mechanisms are implemented to ensure the precision and reliability of stored data. Security remains a paramount concern, with robust access controls and user authentication measures in place to safeguard sensitive voting and polling information. The system architecture is designed with scalability in mind, capable of efficiently managing increased data loads and user traffic as the system grows.

**4.1 API Request**

****

**Fig 4.1. REST API flowchart**

A Representational State Transfer (REST) API is a foundational element in the architecture of the Voting and Polling System, providing an organized 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 system's functionality.

The REST API of the Voting and Polling System adheres to RESTful principles, which revolve around a set of stateless operations for creating, retrieving, updating, and deleting data. It defines a structured format for the endpoints, with each endpoint corresponding to a specific resource or action. For example, endpoints might include "GET /poll" to retrieve a list of available polls, "POST /poll" to create new polls, and "DELETE /poll/{pollID}" to remove specific polls.

By embracing RESTful design, the API simplifies interactions with the system, ensuring that users can easily access poll and Yoga information, participate in surveys, and manage their engagement seamlessly. It provides data in a widely understood format, typically in JSON, allowing for smooth integration with various client applications, including web and mobile interfaces.

In addition to its role in facilitating user interactions, the REST API is a foundational element for potential future developments. It opens doors to third-party integrations, such as authentication services, external data analytics tools, or performance monitoring systems, which can enhance the system's functionality and broaden its capabilities. Furthermore, the API empowers the system to be scalable, ensuring it can adapt to growing user bases and evolving features.

The REST API in the Voting and Polling System is not just a technical component; it serves as the conduit through which the system's core functionalities are exposed and extended, ultimately contributing to a seamless and adaptable user experience.

**4.2 CRUD OPERATION**

In the context of the Voting and Polling System, the implementation of CRUD (Create, Read, Update, Delete) operations is fundamental to the effective management of polls and Yogas. The "Create" operation enables administrators to create new polls, providing comprehensive details that are subject to validation for accuracy and completeness before secure storage in the database.

The "Read" operations empower participants to effortlessly browse and explore the catalog of available polls and Yogas. Enhanced search and filtering options further assist participants in locating specific polls and accessing detailed information about them.

The "Update" operation allows administrators to modify poll and Yoga information through an intuitive user interface. Stringent data validation criteria are upheld to ensure the precision and reliability of the edited information, which is subsequently updated in the database.

The "Delete" operation provides administrators with the capability to remove polls and Yogas from the system, incorporating a confirmation prompt to prevent accidental deletion. Once confirmed, the information associated with the poll or Yoga is securely deleted from the database.

The successful implementation of these CRUD operations is paramount for ensuring that poll and Yoga management within the Voting and Polling System is both efficient and user-friendly. These operations, coupled with robust data validation, security measures, and intuitive interfaces, collectively contribute to a seamless user experience, simplifying the process for administrators to conduct surveys and for participants to engage with the voting and polling functionalities.

**Coding:**

**Yoga Controller Class:**

package com.Controller;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import com.model.online\_voting.Model.Yoga;

import com.model.online\_voting.Service.YogaService;

import java.util.List;

@RestController

@RequestMapping("/api/Yoga")

@CrossOrigin("\*")

public class YogaController {

private final YogaService YogaService;

@Autowired

public YogaController(YogaService YogaService) {

this.YogaService = YogaService;

}

@GetMapping

public List<Yoga> getAllItems() {

return YogaService.getAllItems();

}

@PostMapping

public Yoga addItem(@RequestBody Yoga Yoga) {

return YogaService.addItem(Yoga);

}

@PutMapping("/{itemId}")

public Yoga updateItem(@PathVariable Long itemId, @RequestBody Yoga Yoga) {

return YogaService.updateItem(itemId, Yoga);

}

@DeleteMapping("/{itemId}")

public void removeItem(@PathVariable Long itemId) {

YogaService.removeItem(itemId);

}

@DeleteMapping("/clear")

public void clearResult() {

YogaService.clearResult();

}

}

**Yoga Entity Class:**

package com.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "Yoga")

public class Yoga {

@Id

@Column(unique = true)

private Long id;

@Column(unique = true)

private String name;

private double Yoga;

public Yoga() {

}

public Yoga(String name, double Yoga) {

this.name = name;

this.Yoga = Yoga;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getYoga() {

return Yoga;

}

public void setYoga(double Yoga) {

this.Yoga = Yoga;

}

} **Yoga Service Class:**

package com.service;

import com.model.online\_voting.Model.Yoga;

import com.model.online\_voting.Repository.YogaRepository;

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

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class YogaService {

private final YogaRepository YogaRepository;

@Autowired

public YogaService(YogaRepository YogaRepository) {

this.YogaRepository = YogaRepository;

}

public List<Yoga> getAllItems() {

return YogaRepository.findAll();

}

public Yoga addItem(Yoga Yoga) {

return YogaRepository.save(Yoga);

}

public Yoga updateItem(Long itemId, Yoga Yoga) {

if (YogaRepository.existsById(itemId)) {

Yoga.setId(itemId);

return YogaRepository.save(Yoga);

}

return null;

}

public void removeItem(Long itemId) {

YogaRepository.deleteById(itemId);

}

public void clearResult() {

YogaRepository.deleteAll();

}}

**Yoga Repository Interface:**

package com.repository;

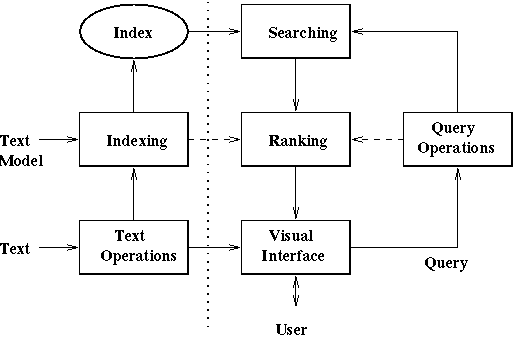
import org.springframework.data.jpa.repository.JpaRepository;

import com.model.online\_voting.Model.Yoga;

public interface YogaRepository extends JpaRepository<Yoga, Long> {

}

**4.3 DATA RETRIEVEL PROCESS**

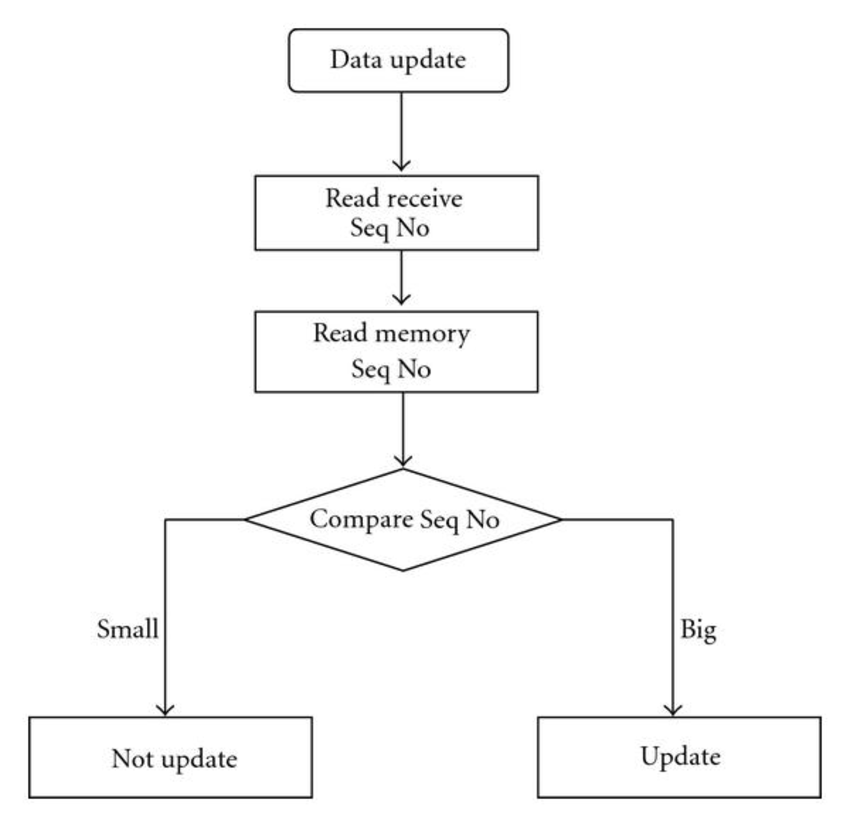
****

**Fig 4.2 Data Retrieval Process**

The data retrieval process in the backend of the Voting and Polling System is a crucial component that facilitates the efficient and secure retrieval of data from the database and its transmission to the frontend or client applications. This process initiates with a client request made to a specific API endpoint on the backend. Upon receiving the request, the backend 's routing system directs it to the appropriate endpoint handler, determined by the URL and HTTP method.

Before proceeding, the backend validates user authentication and confirms the presence of necessary permissions to access the requested data. Once authorization is established, a database query is generated based on the request, specifying the criteria for data retrieval.

**4.4 DATA UPDATE PROCESS**

****

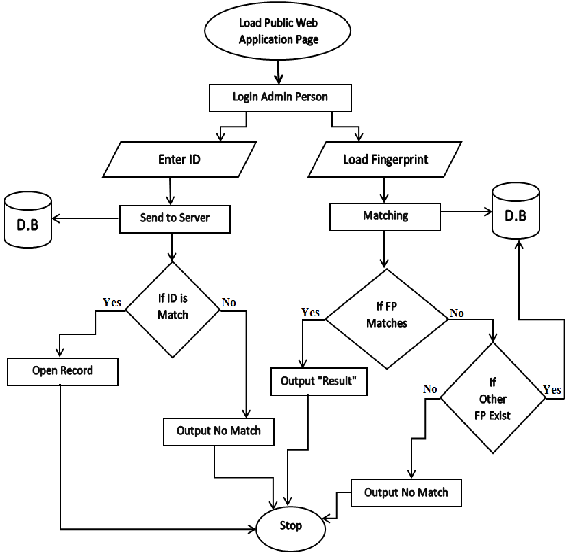
**Fig 4.3 Data Update Flowchart**

The data update process within the backend of the Voting and Polling System is a vital mechanism that empowers users and administrators to enact changes in the system's database.

Authentication and authorization are fundamental checkpoints in this process, ensuring that the user's identity is confirmed and that the necessary permissions are in place for executing the data update operation. Unauthorized requests are diligently restricted.

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

**4.5 SECURITY AND AUTHENTICATION**

****

**Fig 4.4 Security And Authentication Flowchart**

Security and authentication are fundamental to the robust infrastructure of the Voting and Polling System. A comprehensive set of security measures and authentication protocols have been put in place to safeguard sensitive data and uphold the integrity of the system.

User authentication is a foundational element, enabling users, including administrators and participants, to securely register using email and strong, hashed passwords. A robust login system verifies user credentials and controls access to the system. Role-based access assigns specific permissions based on user roles, providing administrators with control over system management functions and restricting participants primarily to read-only access.

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

The use of security libraries and frameworks, such as Spring Security, enhances the efficiency of authentication and access control. Additionally, protection against common security threats, including 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 proactively to identify and address potential weaknesses, maintaining the system's resilience against emerging security risks. In summary, these measures collectively create a secure and trustworthy environment within the Voting and Polling System, protecting user data and maintaining the system's integrity.

**Coding:**

**Jwt Authentication Filter Class:**

package com.config;

import java.io.IOException;

importorg.springframework.security.authentication.UsernamePasswordAuthenticationToken;import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

importorg.springframework.security.web.authentication.WebAuthenticationDetailsSource;import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import lombok.RequiredArgsConstructor;

@Component

@RequiredArgsConstructor

public class JwtAuthenticationFilter extends OncePerRequestFilter{

private final JwtService jwtService;

private final UserDetailsService userDetailsService;

@Override

protected void doFilterInternal(

HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain

)throws ServletException, IOException{

final String authHeader = request.getHeader("Authorization");

final String jwt;

final String username;

if(authHeader == null || !authHeader.startsWith("Bearer ")){

filterChain.doFilter(request, response);

return;

}

jwt = authHeader.substring(7);

username = jwtService.extractUsername(jwt);

if(username != null && SecurityContextHolder.getContext().getAuthentication()==null){

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if(jwtService.isTokenValid(jwt, 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.config;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

import javax.crypto.SecretKey;

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

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Service;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.io.Decoders;

import io.jsonwebtoken.security.Keys;

@Service

public class JwtService {

@Value("${application.jwt.secret-key}")

private String SECRET\_KEY;

public <T> T extractClaim(String token, Function<Claims, T>claimsResolver){

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

public String generateToken(UserDetails userDetails){

return generateToken(new HashMap<>(), userDetails);

}

private String generateToken(

Map<String, Object> extraClaims,

UserDetails userDetails

) {

return Jwts

.builder()

.setClaims(extraClaims)

.setSubject(userDetails.getUsername())

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 43200000))

.signWith(getSignInKey(), SignatureAlgorithm.HS256)

.compact();

}

public boolean isTokenValid(String token, UserDetails userDetails){

final String username = extractUsername(token);

return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);

}

private boolean isTokenExpired(String token){

return extractExpiration(token).before(new Date());

}

private Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

private Claims extractAllClaims(String token){

return Jwts

.parserBuilder()

.setSigningKey(getSignInKey())

.build()

.parseClaimsJws(token)

.getBody();

}

private SecretKey getSignInKey(){

byte[] keybytes = Decoders.BASE64.decode(SECRET\_KEY);

return Keys.hmacShaKeyFor(keybytes);

}}

**4.6 MICROSERVICE**

Microservices with Spring is a robust and efficient approach to constructing microservices-based applications using the Spring Framework, with a specific emphasis on Spring Boot. Spring's comprehensive ecosystem offers a wealth of tools and libraries that streamline the development, deployment, and management of microservices. Spring Boot, in particular, simplifies microservices development by providing a pre-configured environment for creating production-ready applications, reducing the burden of infrastructure setup and allowing developers to concentrate on core business logic. Spring Cloud complements this with a suite of tools for building distributed systems, encompassing service discovery, configuration management, API gateways, and circuit breakers.

Describe the architectural decisions made for the microservice, including technology choices, databases, communication protocols, and any other design considerations that impact how the microservice functions. Consider including diagrams illustrating the microservice's place within the broader system architecture.

**CHAPTER 5**

**CONCLUSION**

This chapter discusses the project's conclusions and the knowledge gained from its management.

**5.1 CONCLUSION**

In conclusion, the development and implementation of a robust and secure backend application for the Zenyoga have played a pivotal role in redefining the landscape of democratic engagement and facilitating a seamless and efficient experience for both citizens and organizations.

The thorough examination of backend functionalities, encompassing data management, user authentication, and security measures, underscores the vital role of the backend application in safeguarding the integrity and confidentiality of sensitive voting and polling data. Emphasizing scalability, data integrity, and user-centric design, it has effectively enhanced democratic engagement, user experience, and provided a secure infrastructure for the seamless operation of the Zenyoga.

As the digital democracy landscape continues to evolve, the insights derived from this study serve as a testament to the indispensable role of the Zenyoga's backend application in driving innovation, fostering participation, and empowering citizens and organizations through a dynamic and accessible online voting and polling environment.

Security and data integrity remain paramount, with stringent authentication measures, access controls, and encryption mechanisms in place to protect sensitive information. The backend architecture is designed for scalability, ensuring that the system can adapt to growing user bases and increasing survey and election catalogs. Continuous improvement, guided by user feedback and emerging trends, is a cornerstone of the Zenyoga's commitment to staying at the forefront of its domain.

**5.2 FUTURESCOPE**

Looking ahead, the Zenyoga's backend offers substantial potential for further enhancements and advancements in line with the evolving demands of the digital democratic landscape. One promising avenue for future development involves the integration of advanced AI and machine learning capabilities to facilitate more intelligent and tailored participant-engagement, thereby streamlining the democratic engagement processes and enhancing the overall quality of engagement.

Additionally, the implementation of sophisticated data analytics tools within the backend can offer valuable insights into participation trends, system performance metrics, and democratic efficacy, enabling administrators to make data-driven decisions and continually optimize the platform's functionalities.

Embracing the integration of blockchain technology presents another compelling opportunity, enhancing data security, transparency, and integrity within the online voting and polling ecosystem. Furthermore, prioritizing the expansion of networking and collaboration features within the backend can foster more interactive and engaging democratic experiences, promoting seamless communication and relationship-building between participants and administrators.

Continuous efforts to improve user experience through intuitive interface design, user-friendly navigation, and responsive support services will remain integral to the future evolution of the Zenyoga's backend.

Finally, maintaining stringent security measures and ensuring compliance with industry regulations will be paramount to mitigate potential cybersecurity risks and safeguard the confidentiality of sensitive data within the system. As the digital democracy landscape continues to evolve, these proactive initiatives and forward-thinking strategies will play a pivotal role in shaping the future of the Zenyoga's backend, fostering a more efficient, secure, and user-centric democratic environment for all stakeholders involved.

**CHAPTER 6**

**REFERENCES**

1. "Spring Boot Official Documentation" - The official documentation is a valuable resource for Spring Boot, which is commonly used for backend development in the context of the Voting & Polling System.

2. "Spring Boot 2 Recipes: A Problem-Solution Approach" by Marten Deinum, Daniel Rubio, Josh Long, and Iuliana Cosmina - This resource offers solutions to common problems encountered in Spring Boot 2, which is applicable to the Voting & Polling System's backend development.

3. "RESTful Web Services with Spring Framework" - This resource provides insights into building RESTful services with the Spring Framework, which is relevant to the development of the Voting & Polling System.

4. "Pro Spring Boot 2" by Felipe Gutierrez - This book offers in-depth insights into Spring Boot, a widely used framework for building the backend of the Voting & Polling System.

5. "Spring Boot in Action" by Craig Walls - This book provides practical guidance on using Spring Boot for the development of the Voting & Polling System's backend.