1. Introduction

The backend of the Job Searching Portal is a critical component of the application, responsible for handling data management, user authentication, job postings, and secure communication between the client and server. The backend is built using Spring Boot, a popular Java framework for building microservices and RESTful APIs. This report provides an in-depth analysis of the backend architecture, including configuration, controller services, and security mechanisms.

2. Architecture and Components

The backend architecture is composed of several key components, including:

- **Spring Boot Framework**: Utilized for building a robust RESTful API with support for dependency injection, security, and data management.
- **MySQL Database**: Used as the primary relational database for storing user data, job postings, and other application-related information.
- **Spring Security**: Provides authentication and authorization functionalities, ensuring secure access to the application's resources.
- **JWT (JSON Web Token)**: Used for user authentication, providing a secure and scalable solution for managing user sessions.
- **Service Layer**: Handles business logic and interaction between the controllers and repositories.
- **Repository Layer**: Responsible for data access and persistence, leveraging JPA (Java Persistence API) for database interactions.

3. Components Description

3.1. Configuration

The configuration files define the overall behavior of the backend, including CORS (Cross-Origin Resource Sharing) settings, security configurations, and database connections.

- **CorsConfig**: Manages CORS settings, allowing the frontend application hosted on http://localhost:3000 to interact with the backend services.
- **SecurityConfig**: Configures the security settings, permitting public access to specific endpoints such as user registration, login, and job application APIs, while restricting access to other resources.

3.2. Controllers

Controllers are responsible for handling HTTP requests, processing data, and returning appropriate responses.

- **JobController**: Manages job postings, allowing users to create, view, and delete job listings.
- **UserController**: Handles user-related operations, including registration, login, and user management.

3.3. Services

The service layer contains business logic and interacts with the repository layer to perform CRUD (Create, Read, Update, Delete) operations.

- **JobService**: Provides methods to add, retrieve, and delete job postings.
- **UserService**: Manages user registration, authentication, and updates, utilizing password encryption and JWT generation.

3.4. Models

The models represent the database entities and are annotated with JPA annotations for ORM (Object-Relational Mapping).

- **Job**: Represents a job posting entity, with attributes like title, description, location, company name, salary range, job type, benefits, and job level.
- **User**: Represents a user entity, containing attributes such as username, password, email, phone, and address.

3.5. Repositories

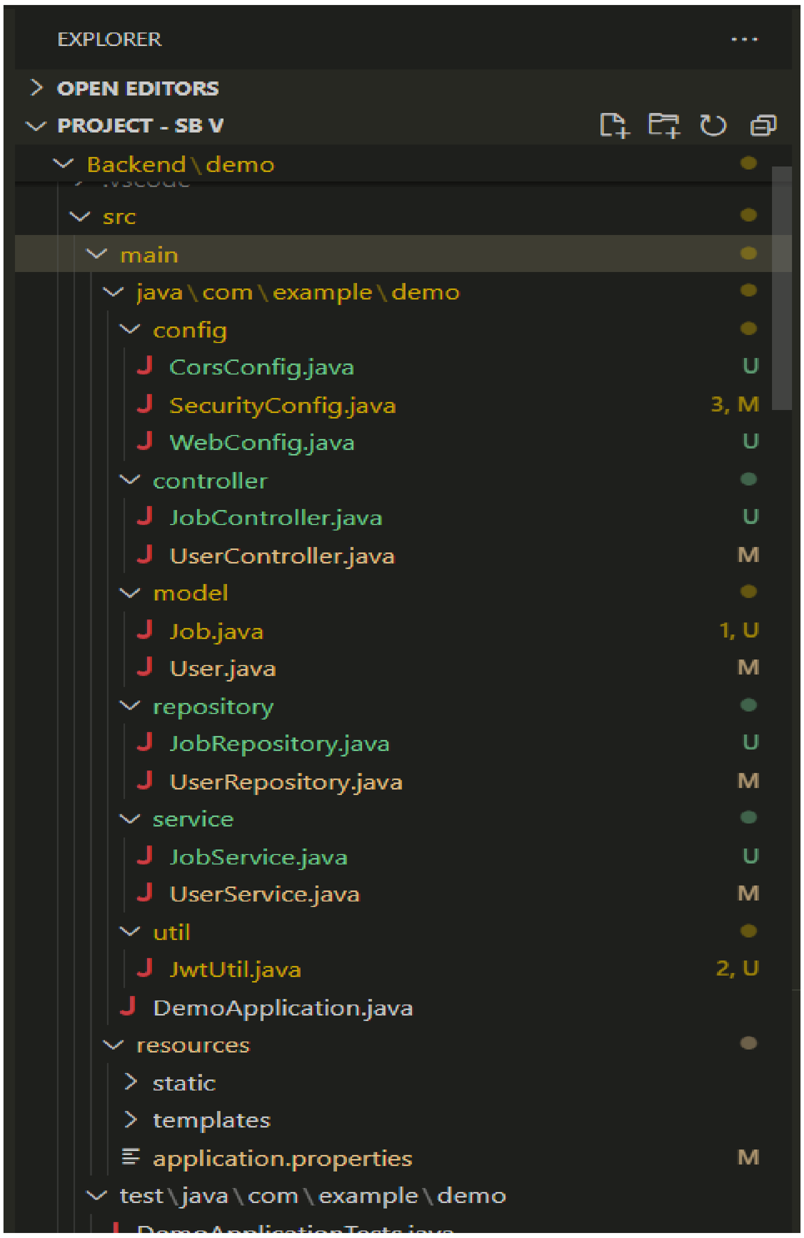
Repositories interface with the database, allowing the application to perform CRUD operations seamlessly.

- **JobRepository**: Extends JpaRepository to provide data access for job postings.
- **UserRepository**: Extends JpaRepository to provide data access for user information.

3.6. Security

Security is implemented using Spring Security and JWT for authentication.

- **BCryptPasswordEncoder**: Encrypts user passwords before storing them in the database.
- **JwtUtil**: Handles JWT generation, extraction of claims, and token validation.



```
EXPLORER                                        ...
> OPEN EDITORS
v PROJECT - SB V
  v Backend\demo                                  ●
    > .vscode
    v src                                         ●
      v main                                      ●
        v java\com\example\demo                   ●
          v config                                ●
            CorsConfig.java                       U
            SecurityConfig.java                3, M
            WebConfig.java                        U
          v controller                            ●
            JobController.java                    U
            UserController.java                   M
          v model                                 ●
            Job.java                           1, U
            User.java                             M
          v repository                            ●
            JobRepository.java                    U
            UserRepository.java                   M
          v service                               ●
            JobService.java                       U
            UserService.java                      M
          v util                                  ●
            JwtUtil.java                       2, U
          DemoApplication.java
        v resources                               ●
          > static
          > templates
          application.properties                  M
      v test\java\com\example\demo
        DemoApplicationTests.java
```

Fig 3.6.1 Backend System Architecture

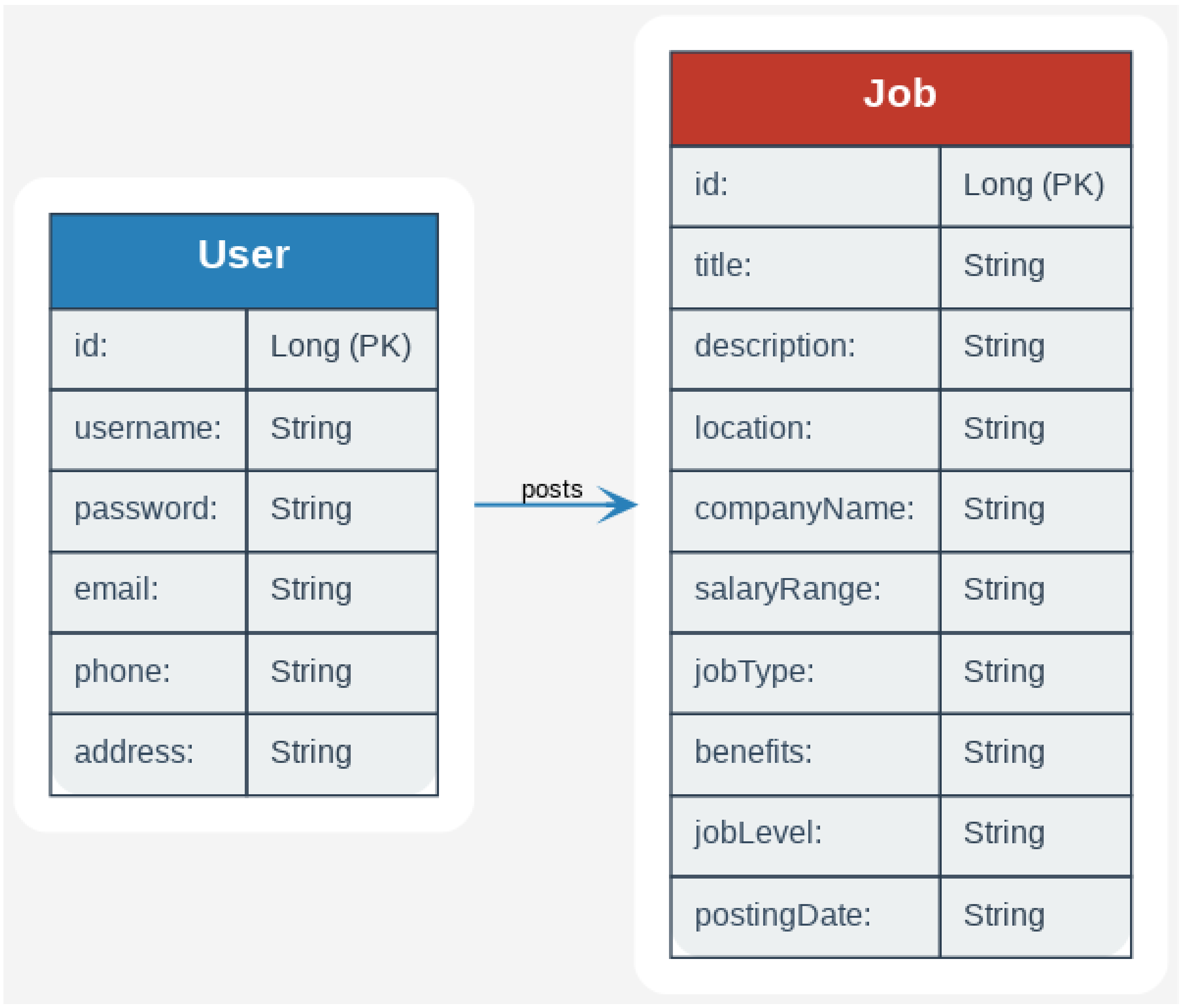
4. Entity-Relationship

4.1. Entities

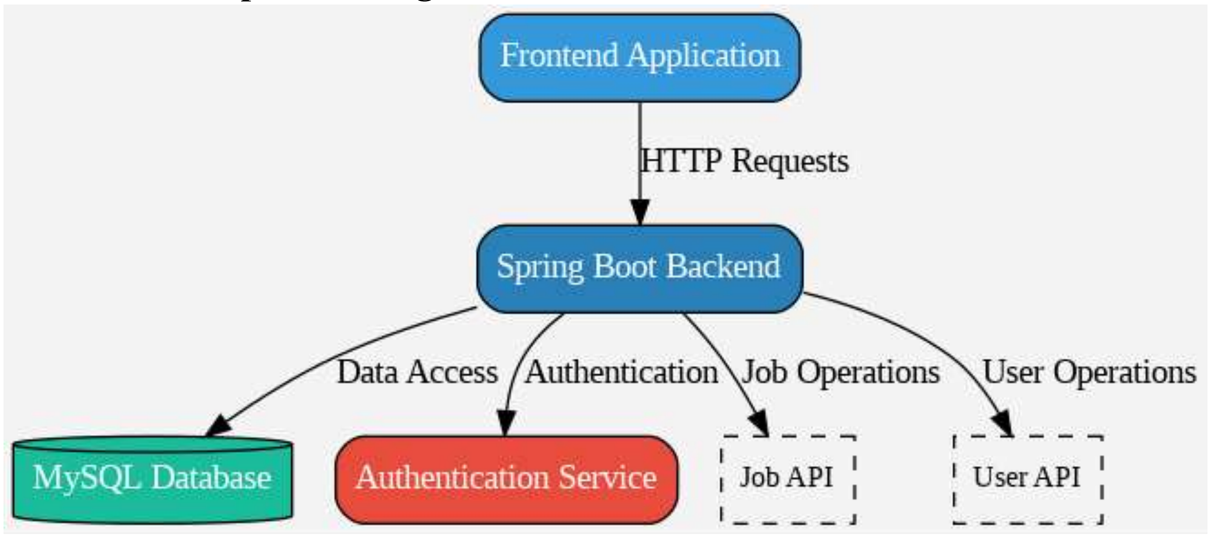
- **User**
    - id: Long (Primary Key)
    - username: String
    - password: String
    - email: String
    - phone: String
    - address: String
- **Job**
    - id: Long (Primary Key)
    - title: String
    - description: String
    - location: String
    - companyName: String
    - salaryRange: String
    - jobType: String
    - benefits: String
    - jobLevel: String
    - postingDate: String

Diagram

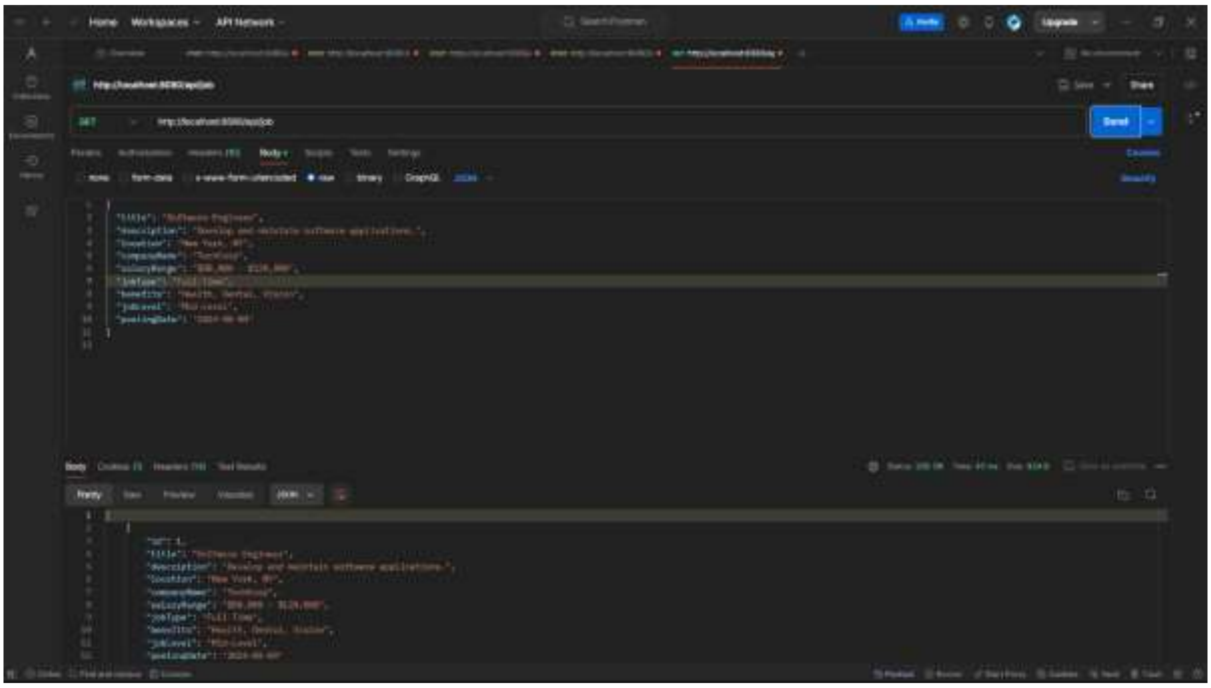
4.3. Entity Diagram



4.4. UML Component Diagram



4.5. Postman Api

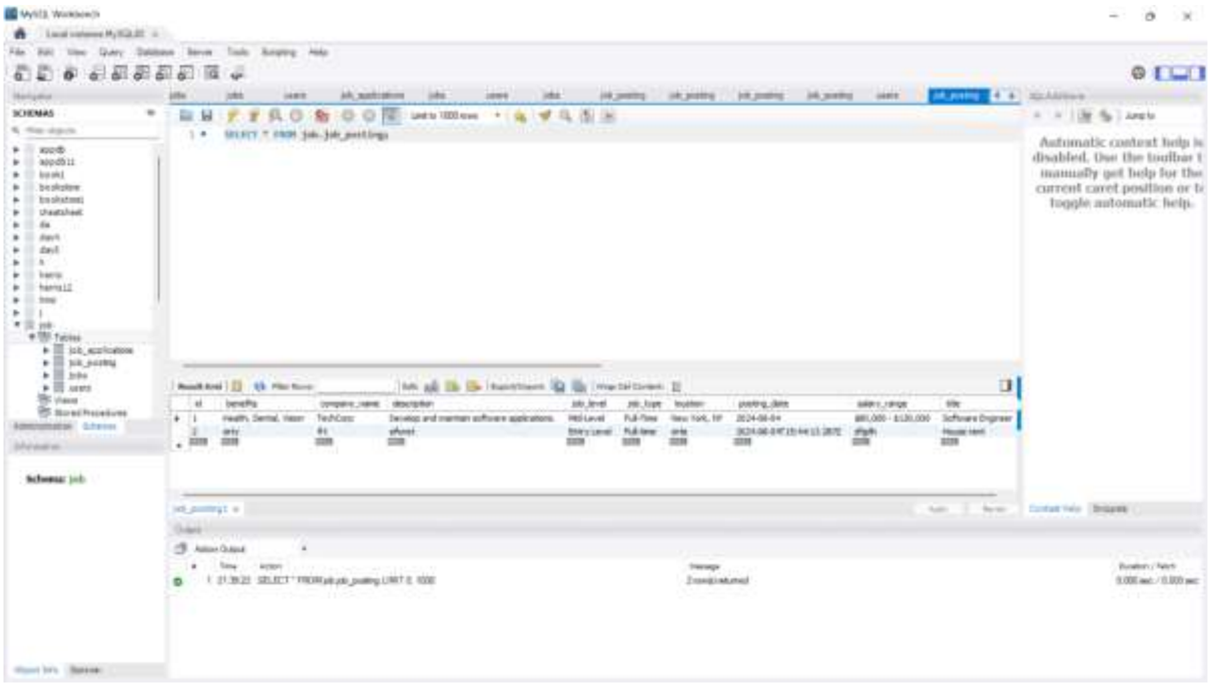


GET http://localhost:8080/api/job

```
{
"title": "Software Engineer",
"description": "Develop and maintain software applications.",
"location": "New York, NY",
"companyName": "TechCorp",
"salaryRange": "$80,000 - $120,000",
"jobType": "Full-Time",
"benefits": "Health, Dental, Vision",
"jobLevel": "Mid-Level",
"postingDate": "2024-08-04"
}
```

```
[
    {
        "id": 1,
        "title": "Software Engineer",
        "description": "Develop and maintain software applications.",
        "location": "New York, NY",
        "companyName": "TechCorp",
        "salaryRange": "$80,000 - $120,000",
        "jobType": "Full-Time",
        "benefits": "Health, Dental, Vision",
        "jobLevel": "Mid-Level",
        "postingDate": "2024-08-04"
```

4.6. Mysql Database



```
SELECT * FROM job.job_posting;
```

id	benefits	company_name	description	job_level	job_type	location	posting_date	salary_range	title
1	Health, Dental, Vision	TechCorp	Develop and maintain software applications.	Mid-Level	Full-Time	New York, NY	2024-08-04	$80,000 - $120,000	Software Engineer
2	[illegible]	[illegible]	[illegible]	Entry Level	Full-time	[illegible]	[illegible]	[illegible]	[illegible]
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
1 21:35:23 SELECT * FROM job.job_posting LIMIT 0, 1000    2 row(s) returned    0.000 sec / 0.000 sec
```

5. Coding

5.1. UserController.java

```
package com.example.demo.controller;
import com.example.demo.model.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:3000")// frontend oda localhost
public class UserController {

   @Autowired
   private UserService userService;

   @PostMapping("/register")
   public ResponseEntity<User> registerUser(@RequestBody User user) {
      return ResponseEntity.ok(userService.registerUser(user));
   }

   @PostMapping("/login")
public ResponseEntity<String> loginUser(@RequestBody User loginUser) {
   String token = userService.loginUser(loginUser.getUsername(),
loginUser.getPassword());
   if (token != null) {
      return ResponseEntity.ok(token);
   } else {
      return ResponseEntity.status(401).build();
   }
}

   @GetMapping
   public ResponseEntity<List<User>> getAllUsers() {
      return ResponseEntity.ok(userService.getAllUsers());
   }

   @GetMapping("/{id}")
```

```
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        Optional<User> user = userService.getUserById(id);
        return user.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody
User user) {
        Optional<User> updatedUser = userService.updateUser(id, user);
        return updatedUser.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        if (userService.deleteUser(id)) {
            return ResponseEntity.noContent().build();
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

5.2. JobController.java

```
package com.example.demo.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import com.example.demo.model.Job;
import com.example.demo.service.JobService;

import java.util.List;

@RestController
@RequestMapping("/api/job")
@CrossOrigin(origins = "http://localhost:3000")

public class JobController {

    @Autowired
    private JobService jobService;

    @PostMapping
    public ResponseEntity<Job> addJob(@RequestBody Job job) {
        Job newJob = jobService.addJob(job);
        return new ResponseEntity<>(newJob, HttpStatus.CREATED);
    }
    @GetMapping
    public ResponseEntity<List<Job>> getAllJobs() {
        List<Job> jobs = jobService.getAllJobs();
        return new ResponseEntity<>(jobs, HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteJob(@PathVariable Long id) {
        jobService.deleteJob(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
```

5.3. Job.java

```
package com.example.demo.model;
import jakarta.persistence.*;
import java.util.Date;

@Entity
@Table(name = "job_posting")
public class Job {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private String location;
    private String companyName;
    private String salaryRange;
    private String jobType;
    private String benefits;
    private String jobLevel;
    private String postingDate;

    public Job(){
        super();
    }
    public Job(Long id, String title, String description, String location, String
companyName, String salaryRange,
            String jobType, String benefits, String jobLevel, String postingDate) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.location = location;
        this.companyName = companyName;
        this.salaryRange = salaryRange;
        this.jobType = jobType;
        this.benefits = benefits;
        this.jobLevel = jobLevel;
        this.postingDate = postingDate;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public String getLocation() {
        return location;
    }
    public void setLocation(String location) {
        this.location = location;
    }
    public String getCompanyName() {
        return companyName;
    }
    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }
    public String getSalaryRange() {
        return salaryRange;
    }
    public void setSalaryRange(String salaryRange) {
        this.salaryRange = salaryRange;
    }
    public String getJobType() {
        return jobType;
    }
    public void setJobType(String jobType) {
        this.jobType = jobType;
    }
    public String getBenefits() {
        return benefits;
    }
    public void setBenefits(String benefits) {
        this.benefits = benefits;
    }
    public String getJobLevel() {
        return jobLevel;
    }
    public void setJobLevel(String jobLevel) {
        this.jobLevel = jobLevel;
    }
    public String getPostingDate() {
        return postingDate;
    }
    public void setPostingDate(String postingDate) {
        this.postingDate = postingDate;
    }
}
```

5.4. User.java

```
package com.example.demo.model;
import jakarta.persistence.*;
import lombok.Data;
@Entity
@Data
@Table(name = "users")//table table name
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;
    private String password;
    private String email;
    private String phone;
    private String address;

    public User() {
        super();
    }
    public User(Long id, String username, String password, String email, String phone,
String address) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.email = email;
        this.phone = phone;
        this.address = address;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }

    // Getters and setters
    // ...
}
```

5.5. UserRepository.java

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

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

5.6. JobRepository.java

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

@Repository
public interface JobRepository extends JpaRepository<Job, Long> {
}
```

5.7. UserService.java

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

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import com.example.demo.util.JwtUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

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

```
@Service
public class UserService {

   @Autowired
   private UserRepository userRepository;

   private final BCryptPasswordEncoder passwordEncoder = new
BCryptPasswordEncoder();

   public User registerUser(User user) {
      user.setPassword(passwordEncoder.encode(user.getPassword()));
      return userRepository.save(user);
   }

   public String loginUser(String username, String password) {
      User user = userRepository.findByUsername(username);
      if (user != null && passwordEncoder.matches(password, user.getPassword())) {
         return JwtUtil.generateToken(username);
      }
      return null;
   }

   public List<User> getAllUsers() {
      return userRepository.findAll();
   }

   public Optional<User> getUserById(Long id) {
      return userRepository.findById(id);
   }

   public Optional<User> updateUser(Long id, User userDetails) {
      return userRepository.findById(id).map(user -> {
         user.setUsername(userDetails.getUsername());
         user.setPassword(passwordEncoder.encode(userDetails.getPassword()));
         user.setEmail(userDetails.getEmail());
         user.setPhone(userDetails.getPhone());
         user.setAddress(userDetails.getAddress());
         return userRepository.save(user);
      });
   }

   public boolean deleteUser(Long id) {
      if (userRepository.existsById(id)) {
         userRepository.deleteById(id);
         return true;
      }
```

```
        return false;
    }
}
```

5.8 .JobService.java

```
package com.example.demo.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.demo.model.Job;
import com.example.demo.repository.JobRepository;

import java.util.List;

@Service
public class JobService {

    @Autowired
    private JobRepository jobRepository;

    public Job addJob(Job job) {
        return jobRepository.save(job);
    }
    public List<Job> getAllJobs() {
        return jobRepository.findAll();
    }

    public void deleteJob(Long id) {
        jobRepository.deleteById(id);
    }
}
```

5.9. JWTUtil.java

```
package com.example.demo.util;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.Claims;

import java.util.Date;

public class JwtUtil {

    private static final String SECRET_KEY =
"DWnN2JPlmIimWXd3ZJWJtQ9mQOggGynoZpLCtvrGr/M=";
```

```
    private static final long EXPIRATION_TIME = 1000 * 60 * 60; // 1 hour

    public static String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() +
EXPIRATION_TIME))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }

    public static Claims extractClaims(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(SECRET_KEY)
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    public static String extractUsername(String token) {
        return extractClaims(token).getSubject();
    }

    public static boolean isTokenExpired(String token) {
        return extractClaims(token).getExpiration().before(new Date());
    }

    public static boolean validateToken(String token, String username) {
        return (username.equals(extractUsername(token)) && !isTokenExpired(token));
    }
}
```

5.10. SecurityConfig.java

```
package com.example.demo.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
;
import org.springframework.security.web.SecurityFilterChain;
@Configuration
@EnableWebSecurity
public class SecurityConfig {
   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
       http
          .csrf(csrf -> csrf.disable())
          .authorizeRequests(auth -> auth
             .requestMatchers("/api/users/register", "/api/users/login",
"/api/applications/**", "/api/job/**").permitAll()  // Allow public access to /api/job/**
             .anyRequest().authenticated());
       return http.build();
   }
}
```

5.11. DataBase

```
spring.application.name=demo
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost/Job
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql= true
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

6. Conclusion

The backend of the Job Searching Portal is built with a focus on security, scalability, and efficient data management. By leveraging Spring Boot and related technologies, the application provides a robust and secure platform for job seekers and employers. The integration of JWT for authentication and Spring Security for access control ensures that user data is protected and that only authorized users can access sensitive information. The architecture is designed to be extendable, allowing for future enhancements and scalability as the application grows.

In conclusion, the backend system is well-architected to support the application's requirements, providing a solid foundation for further development and deployment.