Smart Task Management System (STMS) — Full Project (Option B)

> **What this document contains:** - Complete, GitHub-ready project structure and all source files for a Spring Boot application with JWT auth, role-based access, pagination, search, global exception handling, and logging.

Project structure (maven)

```
smart-task-management/
├── pom.xml
├── README.md
├── src/main/java/com/example/stms/
│   ├── SmartTaskManagementApplication.java
│   ├── config/
│   │   └── SecurityConfig.java
│   ├── controller/
│   │   ├── AuthController.java
│   │   └── TaskController.java
│   ├── dto/
│   │   ├── AuthRequest.java
│   │   ├── AuthResponse.java
│   │   └── TaskDTO.java
│   ├── exception/
│   │   ├── ApiException.java
│   │   └── GlobalExceptionHandler.java
│   ├── model/
│   │   ├── Role.java
│   │   ├── Task.java
│   │   └── User.java
│   ├── repository/
│   │   ├── TaskRepository.java
│   │   └── UserRepository.java
│   ├── security/
│   │   ├── JwtFilter.java
│   │   ├── JwtUtil.java
│   │   └── CustomUserDetailsService.java
│   └── service/
│       ├── AuthService.java
│       └── TaskService.java
├── src/main/resources/
│   ├── application.properties
│   └── data.sql
└── .gitignore
```

1) `pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>smart-task-management</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <java.version>17</java.version>
        <spring.boot.version>3.1.3</spring.boot.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starter -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <!-- MySQL -->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!-- JWT -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-api</artifactId>
            <version>0.11.5</version>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-impl</artifactId>
            <version>0.11.5</version>
            <scope>runtime</scope>
        </dependency>
```

```
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-jackson</artifactId>
            <version>0.11.5</version>
            <scope>runtime</scope>
        </dependency>
        <!-- Lombok (optional but useful) -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!-- Validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <!-- Test -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

---

2) `src/main/resources/application.properties`

```
spring.datasource.url=jdbc:mysql://localhost:3306/stms_db?
useSSL=false&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.datasource.initialization-mode=always

# JWT secret (change before production)
jwt.secret=ReplaceThisSecretKeyForProd
jwt.expirationMs=86400000
```

```
server.port=8080
```

---

3) `SmartTaskManagementApplication.java`

```java
package com.example.stms;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SmartTaskManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(SmartTaskManagementApplication.class, args);
    }
}
```

---

4) Models (`model` package)

`Role.java`

```java
package com.example.stms.model;

public enum Role {
    ROLE_USER,
    ROLE_ADMIN
}
```

`User.java`

```java
package com.example.stms.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    private String name;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password; // store hashed

    @Enumerated(EnumType.STRING)
    private Role role = Role.ROLE_USER;
}
```

`Task.java`

```
package com.example.stms.model;

import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDate;

@Entity
@Table(name = "tasks")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private String status; // PENDING / COMPLETED
    private String priority; // LOW / MEDIUM / HIGH
    private LocalDate dueDate;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User user;
}
```

---

5) Repositories

`UserRepository.java`

```
package com.example.stms.repository;

import com.example.stms.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByEmail(String email);
    boolean existsByEmail(String email);
}
```

`TaskRepository.java`

```
package com.example.stms.repository;

import com.example.stms.model.Task;
import com.example.stms.model.User;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface TaskRepository extends JpaRepository<Task, Long> {
    List<Task> findByUser(User user);
    Page<Task> findByUserAndTitleContainingIgnoreCase(User user, String
title, Pageable pageable);
}
```

---

6) DTOs (`dto` package)

`AuthRequest.java`

```
package com.example.stms.dto;

import lombok.Data;

@Data
public class AuthRequest {
```

```
    private String email;
    private String password;
}
```

`AuthResponse.java`

```
package com.example.stms.dto;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class AuthResponse {
    private String token;
}
```

`TaskDTO.java`

```
package com.example.stms.dto;

import lombok.Data;

import java.time.LocalDate;

@Data
public class TaskDTO {
    private String title;
    private String description;
    private String priority;
    private String status;
    private LocalDate dueDate;
}
```

---

7) Security (`security` package)

`JwtUtil.java`

```
package com.example.stms.security;

import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.Date;
```

```
@Component
public class JwtUtil {

    @Value("${jwt.secret}")
    private String jwtSecret;

    @Value("${jwt.expirationMs}")
    private int jwtExpirationMs;

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

    public String getUsernameFromToken(String token) {
        return
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token);
            return true;
        } catch (JwtException | IllegalArgumentException e) {
            return false;
        }
    }
}
```

`JwtFilter.java`

```
package com.example.stms.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
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 java.io.IOException;

@Component
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        final String header = request.getHeader("Authorization");
        String token = null;
        String username = null;

        if (header != null && header.startsWith("Bearer ")) {
            token = header.substring(7);
            try {
                username = jwtUtil.getUsernameFromToken(token);
            } catch (Exception e) {
                // ignore
            }
        }

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

SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

`CustomUserDetailsService.java`

```
package com.example.stms.security;

import com.example.stms.model.User;
import com.example.stms.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collections;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByEmail(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        return new
org.springframework.security.core.userdetails.User(user.getEmail(),
user.getPassword(), Collections.singletonList(new
SimpleGrantedAuthority(user.getRole().name())));
    }
}
```

---

8) Security Config (`config` package)

`SecurityConfig.java`

```
package com.example.stms.config;

import com.example.stms.security.CustomUserDetailsService;
import com.example.stms.security.JwtFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConf
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Autowired
    private JwtFilter jwtFilter;

    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http.csrf().disable()
                .authorizeHttpRequests()
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration configuration) throws
Exception {
        return configuration.getAuthenticationManager();
    }
}
```

9) Services (`service` package)

`AuthService.java`

```
package com.example.stms.service;

import com.example.stms.dto.AuthRequest;
import com.example.stms.dto.AuthResponse;
import com.example.stms.model.Role;
import com.example.stms.model.User;
import com.example.stms.repository.UserRepository;
import com.example.stms.security.JwtUtil;
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;

@Service
public class AuthService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private AuthenticationManager authenticationManager;

    public AuthResponse register(User user) {
        if (userRepository.existsByEmail(user.getEmail())) {
            throw new RuntimeException("Email already in use");
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        if (user.getRole() == null) user.setRole(Role.ROLE_USER);
        User saved = userRepository.save(user);
        String token = jwtUtil.generateToken(saved.getEmail());
        return new AuthResponse(token);
    }

    public AuthResponse login(AuthRequest request) {
        authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword()));
```

```
        String token = jwtUtil.generateToken(request.getEmail());
        return new AuthResponse(token);
    }
}
```

TaskService.java

```
package com.example.stms.service;

import com.example.stms.dto.TaskDTO;
import com.example.stms.exception.ApiException;
import com.example.stms.model.Task;
import com.example.stms.model.User;
import com.example.stms.repository.TaskRepository;
import com.example.stms.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class TaskService {

    @Autowired
    private TaskRepository taskRepository;

    @Autowired
    private UserRepository userRepository;

    public Task createTask(TaskDTO dto, String userEmail) {
        User user = userRepository.findByEmail(userEmail);
        if (user == null) throw new ApiException("User not found");
        Task task = new Task();
        task.setTitle(dto.getTitle());
        task.setDescription(dto.getDescription());
        task.setPriority(dto.getPriority());
        task.setStatus(dto.getStatus() == null ? "PENDING" :
dto.getStatus());
        task.setDueDate(dto.getDueDate());
        task.setUser(user);
        return taskRepository.save(task);
    }

    public Page<Task> getTasks(String userEmail, int page, int size, String
search) {
```

```
        User user = userRepository.findByEmail(userEmail);
        if (user == null) throw new ApiException("User not found");
        Pageable pageable = PageRequest.of(page, size);
        if (search == null || search.isBlank()) {
            return taskRepository.findAll(pageable).map(t -> t);
        } else {
            return
taskRepository.findByUserAndTitleContainingIgnoreCase(user, search,
pageable);
        }
    }

    public Task updateTask(Long id, TaskDTO dto, String userEmail) {
        Optional<Task> ot = taskRepository.findById(id);
        if (ot.isEmpty()) throw new ApiException("Task not found");
        Task task = ot.get();
        if (!task.getUser().getEmail().equals(userEmail)) throw new
ApiException("Not authorized");
        task.setTitle(dto.getTitle());
        task.setDescription(dto.getDescription());
        task.setPriority(dto.getPriority());
        task.setStatus(dto.getStatus());
        task.setDueDate(dto.getDueDate());
        return taskRepository.save(task);
    }

    public void deleteTask(Long id, String userEmail) {
        Optional<Task> ot = taskRepository.findById(id);
        if (ot.isEmpty()) throw new ApiException("Task not found");
        Task task = ot.get();
        if (!task.getUser().getEmail().equals(userEmail)) throw new
ApiException("Not authorized");
        taskRepository.delete(task);
    }
}
```

---

10) Controllers (`controller` package)

`AuthController.java`

```
package com.example.stms.controller;

import com.example.stms.dto.AuthRequest;
import com.example.stms.dto.AuthResponse;
import com.example.stms.model.User;
import com.example.stms.service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
```

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

import jakarta.validation.Valid;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthService authService;

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

    @PostMapping("/login")
    public ResponseEntity<AuthResponse> login(@Valid @RequestBody
AuthRequest req) {
        return ResponseEntity.ok(authService.login(req));
    }
}
```

`TaskController.java`

```
package com.example.stms.controller;

import com.example.stms.dto.TaskDTO;
import com.example.stms.model.Task;
import com.example.stms.service.TaskService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import jakarta.validation.Valid;

@RestController
@RequestMapping("/api/tasks")
public class TaskController {

    @Autowired
    private TaskService taskService;

    @PostMapping
    public ResponseEntity<Task> createTask(@Valid @RequestBody TaskDTO dto,
Authentication auth) {
        String email = auth.getName();
```

```
        return ResponseEntity.ok(taskService.createTask(dto, email));
    }

    @GetMapping
    public ResponseEntity<Page<Task>> getTasks(@RequestParam(defaultValue =
"0") int page,
                                               @RequestParam(defaultValue =
"10") int size,
                                               @RequestParam(required =
false) String search,
                                               Authentication auth) {
        String email = auth.getName();
        return ResponseEntity.ok(taskService.getTasks(email, page, size,
search));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Task> updateTask(@PathVariable Long id, @Valid
@RequestBody TaskDTO dto, Authentication auth) {
        String email = auth.getName();
        return ResponseEntity.ok(taskService.updateTask(id, dto, email));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteTask(@PathVariable Long id,
Authentication auth) {
        String email = auth.getName();
        taskService.deleteTask(id, email);
        return ResponseEntity.ok("Deleted");
    }
}
```

---

11) Exception handling (`exception` package)

`ApiException.java`

```
package com.example.stms.exception;

public class ApiException extends RuntimeException {
    public ApiException(String message) {
        super(message);
    }
}
```

`GlobalExceptionHandler.java`

```
package com.example.stms.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ApiException.class)
    public ResponseEntity<String> handleApiException(ApiException ex) {
        return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(ex.getMessage());
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>>
handleValidation(MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(err ->
errors.put(err.getField(), err.getDefaultMessage()));
        return ResponseEntity.badRequest().body(errors);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleAll(Exception ex) {
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Server error:
" + ex.getMessage());
    }
}
```

---

12) `src/main/resources/data.sql` (sample data)

```
INSERT INTO users (id, name, email, password, role) VALUES (1, 'Admin User',
'admin@example.com', '$2a$10$XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX', 'ROLE_ADMIN');
-- Note: replace hashed password with bcrypt hash of your own password.
```

---

13) `.gitignore`

```
/target
/.idea
*.iml
.DS_Store
application.properties
```

---

14) `README.md` (short ready-to-use)

```
# Smart Task Management System (STMS)

Spring Boot REST API with JWT authentication, role-based access, pagination &
search.

## Run locally
1. Create MySQL DB `stms_db`
2. Update `src/main/resources/application.properties` with DB credentials and
JWT secret
3. `mvn clean package`
4. `java -jar target/smart-task-management-1.0.0.jar`

## API endpoints
- `POST /api/auth/register` - register user
- `POST /api/auth/login` - login -> returns JWT
- `POST /api/tasks` - create task (auth)
- `GET /api/tasks` - list tasks (auth)
- `PUT /api/tasks/{id}` - update (auth)
- `DELETE /api/tasks/{id}` - delete (auth)

## Postman
Export Postman collection with above endpoints and Authorization: Bearer
<token>
```

How to use these files

1. Create a new GitHub repo.
2. Copy this folder structure and files into your local project directory.
3. Update `application.properties` with your DB credentials and a secure JWT secret.
4. Build and run using Maven.
5. Test APIs with Postman. Export collection and attach to repo.

If you want, I can now: - Provide the ready-to-download ZIP of this project, or - Create the GitHub repo for you (I will provide the exact README and git commands), or - Generate the Postman collection JSON.

Which would you like next? (Download ZIP / Create repo instructions / Postman collection)