Modulidentifikation hier öffnen

Spring Boot Setup and Implementation Checklist

1. Initial Setup

Project Creation

- [ ] Go to [Spring Initializr](https://start.spring.io/)
- [ ] Configure project:
  - Choose Maven or Gradle (Maven recommended for beginners)
  - Select Java version (17+ recommended)
  - Choose Spring Boot version (3.x recommended)

- [ ] Add essential dependencies:
  - `spring-boot-starter-web`
  - `spring-boot-starter-data-jpa`
  - `spring-boot-starter-security`
  - `spring-boot-starter-validation`
  - `lombok` (optional but recommended)
  - `mysql-connector-java` or `postgresql` (depending on your database choice)
  - `jjwt` for JWT token handling

IDE Setup

- [ ] Import project into IDE (IntelliJ IDEA recommended)
- [ ] Verify Maven/Gradle builds successfully
- [ ] Configure IDE hot reload (DevTools)

2. Database Setup

Local Database

- [ ] Install MySQL/PostgreSQL locally
- [ ] Create database for the project
- [ ] Configure `application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/superherodb
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Web Database (e.g., Railway, Heroku)

- [ ] Create database service account
- [ ] Get connection credentials
- [ ] Configure environment variables
- [ ] Update `application.properties` with environment variables:

```
spring.datasource.url=${DATABASE_URL}
```

```
spring.datasource.username=${DATABASE_USERNAME}
spring.datasource.password=${DATABASE_PASSWORD}
```

3. Project Structure Implementation (In Order)

1. Domain Models

- [ ] Create `model` package
- [ ] Implement entity classes with JPA annotations:

```java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    private String email;

    private String password;

    // other fields, getters, setters
}
```

- [ ] Create necessary DTOs (Data Transfer Objects)

2. Repositories

- [ ] Create `repository` package
- [ ] Create interfaces extending JpaRepository:

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

3. Services

- [ ] Create `service` package
- [ ] Define service interfaces
- [ ] Implement service classes:

```java
@Service
public class UserService {
    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

     // Service methods
}
```

4. Controllers

- [ ] Create `controller` package
- [ ] Implement REST controllers:

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    private final UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

     // Controller endpoints
}
```

4. Security Implementation

Basic Security Setup

- [ ] Create security configuration class:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
    Exception {
        // Security rules
    }
 }
```

JWT Implementation

- [ ] Create JWT utility class
- [ ] Implement JWT token generation and validation
- [ ] Create JWT filter for request authentication

Security Checklist

- [ ] Implement password encryption (BCrypt)
- [ ] Configure CORS
- [ ] Set up CSRF protection
- [ ] Implement rate limiting
- [ ] Set up request validation
- [ ] Configure security headers
- [ ] Implement role-based authorization
- [ ] Set up error handling
- [ ] Configure logout mechanism

5. Frontend Integration

CORS Configuration

- [ ] Configure CORS in Spring Boot:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
            .allowedOrigins("http://localhost:5173")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*");
    }
}
```

Documentation

- [ ] Add GitHub Pages documentation
- [ ] Document all endpoints
- [ ] Add Class Diagrams / images of Unit Test Results

Environment Configuration

- [ ] Set up different profiles (dev/prod)
- [ ] Configure environment-specific properties
- [ ] Set up logging

6. Testing

Unit Tests

- [ ] Test service layer
- [ ] Test repositories
- [ ] Test security configurations

Integration Tests

- [ ] Test API endpoints
- [ ] Test database operations
- [ ] Test security features

Frequent Gotchas and Solutions

- Remember to use @CrossOrigin for development
- Always use DTOs for request/response
- Never expose entity objects directly
- Always validate input data
- Use proper exception handling
- Implement proper logging
- Use environment variables for sensitive data
- Implement proper error responses
- Use proper HTTP status codes