Im folgenden habe ich versucht die aktuelle Übung von GPT lösen zu lassen. Es sind ein paar gute Codeteile rausgekommen. Jedoch hört die KI bei längeren Antworten nach einer bestimmten Zeit einfach auf. Mit Folgefragen lässt sich das Problem aber bestimmt lösen. Auch merkt man dass der Code nicht auf dem neuesten Stand ist.

Create a spring boot application that statifies the following requirements: - Has a controller for user authentication that returns a token - Stores user data utilizing the UserDetails interface - Has an article enity with title and description that each user can own - With an endpoint the user is able to grant the permissions read, write, delete of the article entiy to any other user. this user is then able to read, update or delete the article with appropiate controller endpoints

---

To satisfy the given requirements, you can create a Spring Boot application with the following steps:

Set up a new Spring Boot project with the necessary dependencies by using a build tool like Maven or Gradle. For example, you can use Spring Initializr to create a new project with Spring Web, Spring Data JPA, and Spring Security dependencies.

Create a User entity that implements the UserDetails interface. The User entity should have fields for username, password, and authorities (i.e., roles).

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

    @ElementCollection(fetch = FetchType.EAGER)
    private List<String> authorities;

    // constructors, getters and setters, and UserDetails methods
}
```

Create a UserRepository interface that extends the JpaRepository interface to handle CRUD operations for User entities.

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

Configure Spring Security to use the UserRepository to authenticate users.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserRepository userRepository;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(username ->
userRepository.findByUsername(username));
    }

    // other security configurations
}
```

Create a JWT authentication endpoint that returns a token when a user logs in. You can use the jjwt library to generate and verify JWT tokens.

```java
@RestController
@RequestMapping("/auth")
public class AuthController {
    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody JwtRequest jwtRequest) throws
Exception {
        authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(jwtRequest.getUsername(),
jwtRequest.getPassword()));
        final UserDetails userDetails =
userRepository.findByUsername(jwtRequest.getUsername());
        final String token = jwtTokenUtil.generateToken(userDetails);
        return ResponseEntity.ok(new JwtResponse(token));
    }
}
```

Create an Article entity with fields for title, description, and owner. You can use a ManyToOne relationship to link the Article entity to the User entity.

```java
@Entity
@Table(name = "articles")
public class Article {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String description;

    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User owner;

    // constructors, getters and setters
}
```

Create an ArticleRepository interface that extends the JpaRepository interface to handle CRUD operations for Article entities.

```java
@Repository
public interface ArticleRepository extends JpaRepository<Article, Long> {
}
```

Create a Controller for managing articles. This Controller should have endpoints for creating, reading, updating, and deleting articles. The endpoints should be secured with Spring Security.

```java
@RestController
@RequestMapping("/articles")
public class ArticleController {
    @Autowired
    private ArticleRepository articleRepository;

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/{id}")
    public ResponseEntity<Article> getArticle(@PathVariable Long id,
Authentication authentication) {
        Article article = articleRepository.findById(id).orElseThrow(() -> new
ResourceNotFoundException("Article", "id", id));
        User currentUser =
userRepository.findByUsername(authentication.getName());
        if (article.getOwner().getId().equals(currentUser.getId()) ||
currentUser.getAuthorities().contains("ROLE_ADMIN")) {
            return ResponseEntity.ok(article);
```

```
        } else {
            return
```