Todos App - Getting started with the backend

Credentials

Only 3 users are added as part of the app. Usernames and passwords below. The use with username `jane` is a marked as an admin. However the app does not implement any role-based access (authentication only, no authorization implementation is provided).

- user / dummy
- john / dummy
- jane / dummy

Step 1: Create the Spring Boot starter project

- Go to https://start.spring.io/ and create a project with the following settings.

```
Group: com.greatlearning.security
Artifact: spring-boot-security
Description: Demo project for Spring Boot Security
```

- Also choose Maven Project, Java 17, and packaging as Jar.
- Download the generated project
- Import the project into IntelliJ (or you can use Eclipse/Netbeans etc. The steps explained here assume IntelliJ).

Step 2: Add the dependencies

- `pom.xml` - Add the following

```xml
<dependencies>
    <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-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>
    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.3.1</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
```

```xml
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.project-lombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
</build>
```

- Make sure to "Load Maven changes" (the button that appears, once you add the dependencies, on the top right corner of the open file in the editor).

Step 3: Run the app

- `SpringBootSecurityApplication` - Right click on the file and run. Enable annotations processing in the dialog that may pop up - Lombok requires this to work.
- Spring Security generates a default user with username `user`. Note the password displayed in the terminal at startup (It appears like so - Using generated security password: 00798578-f8e0-4ed1-96d9-b4ef157fbae6).
- Visit `http://localhost:8080` and login using this username and password.

Step 4: Add a home page

- `resources/application.properties`

```
spring.application.name=spring-boot-security

jwt.signing.key.secret=mySecret
jwt.get.token.uri=/authenticate
jwt.refresh.token.uri=/refresh
jwt.http.request.header=Authorization
jwt.token.expiration.in.seconds=604800

spring.datasource.url=jdbc:h2:mem:todolistDB
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=admin
spring.datasource.password=

spring.jpa.defer-datasource-initialization=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true

spring.h2.console.enabled=true
```

Step 5: Add a home page

- `resources/static/index.html`

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title>Todos API</title>
    </head>
    <body>
        <h1>Todos API</h1>
        <hr />
        This API server serves the todos data required by the Todos Application
    </body>
</html>
```

- Restart the app. Now you can login (using the `user` username and the new password in terminal), and view the home page on `http://localhost:8080`

Step 6: Add Todo POJO that defines the Todo Entity

- `todo` - Create a package called `todo`
- `todo/Todo.java` - Create a Todo POJO

```java
package com.gl.rest.webservices.restfulwebservices.todo;

import java.util.Date;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;

@Entity
public class Todo {
    @Id
    @GeneratedValue
    private Long id;
    private String username;
    private String description;
    private Date targetDate;

    public Todo() {

    }

    public Todo(long id, String username, String description, Date targetDate) {
        super();
        this.id = id;
        this.username = username;
        this.description = description;
        this.targetDate = targetDate;

    }

    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 getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Date getTargetDate() {
        return targetDate;
    }

    public void setTargetDate(Date targetDate) {
        this.targetDate = targetDate;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + (int) (id ^ (id >>> 32));
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Todo other = (Todo) obj;
        if (id != other.id)
            return false;
        return true;
    }
}
```

Step 7: Add TodoJpaRepository

- `todo/TodoJpaRepository.java`

```
package com.greatlearning.security.spring_boot_security.todo;

import java.util.List;

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

@Repository
public interface TodoJpaRepository extends JpaRepository<Todo, Long>{
    List<Todo> findByUsername(String username);
}
```

Step 8: Add TodoJpaResource

- `todo/TodoJpaResource.java`

```java
package com.greatlearning.security.spring_boot_security.todo;

import java.net.URI;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
public class TodoJpaResource {

    @Autowired
    private TodoJpaRepository todoJpaRepository;

    @GetMapping("/jpa/users/{username}/todos")
    public List<Todo> getAllTodos(@PathVariable String username) {
        return todoJpaRepository.findByUsername(username);
    }

    @GetMapping("/jpa/users/{username}/todos/{id}")
    public Todo getTodo(@PathVariable String username, @PathVariable long id) {
        return todoJpaRepository.findById(id).get();
    }

    @DeleteMapping("/jpa/users/{username}/todos/{id}")
    public ResponseEntity<Void> deleteTodo(@PathVariable String username,
@PathVariable long id) {
        todoJpaRepository.deleteById(id);

        return ResponseEntity.noContent().build();
    }

    @PutMapping("/jpa/users/{username}/todos/{id}")
    public ResponseEntity<Todo> updateTodo(@PathVariable String username,
@PathVariable long id,
                                             @RequestBody Todo todo) {
        todo.setUsername(username);

        todoJpaRepository.save(todo);

        return new ResponseEntity<Todo>(todo, HttpStatus.OK);
    }

    @PostMapping("/jpa/users/{username}/todos")
    public ResponseEntity<Void> createTodo(@PathVariable String username,
@RequestBody Todo todo) {
        todo.setUsername(username);

        Todo createdTodo = todoJpaRepository.save(todo);

        // Location
        // Get current resource url
        // {id}
        URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(creat
edTodo.getId())
                .toUri();

        return ResponseEntity.created(uri).build();
    }
}
```

- The todos (array of Todo objects) resource can now be served. Check `http://localhost:8080/jpa/users/user/todos` . However, we have no data right now.

Step 9: Add seed data and application properties

- `resources/data.sql`

```
insert into todo(id, username,description,target_date)
values(101, 'user', 'Learn Driving', NOW());

insert into todo(id, username,description,target_date)
values(102, 'user', 'Complete Reading Book', NOW());

insert into todo(id, username,description,target_date)
values(103, 'user', 'Run 5 Km', NOW());
```

- Check `http://localhost:8080/jpa/users/user/todos` . You have some data now.

Step 10: Add the `/authenticate` Request POJO

- `todo` - Create a package called `jwt` and within it another called `resource`
- `jwt/resource/JwtTokenRequest.java`

```java
package com.greatlearning.security.spring_boot_security.jwt.resource;

import java.io.Serializable;

public class  JwtTokenRequest implements Serializable {

    private static final long serialVersionUID = -5616176897013108345L;

    private String username;
    private String password;

    public JwtTokenRequest() {
        super();
    }

    public JwtTokenRequest(String username, String password) {
        this.setUsername(username);
        this.setPassword(password);
    }

    public String getUsername() {
        return this.username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return this.password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Step 11: Add the `authenticate` Response POJO

- `jwt/resource/JwtTokenResponse.java`

```java
package com.greatlearning.security.spring_boot_security.jwt.resource;

import java.io.Serializable;

public class JwtTokenResponse implements Serializable {

    private static final long serialVersionUID = 8317676219297719109L;

    private final String token;

    public JwtTokenResponse(String token) {
        this.token = token;
    }

    public String getToken() {
        return this.token;
    }
}
```

Step 12: Add the authentication controller that generates and returns a token on `authenticate` endpoint

- `jwt/resource/AuthenticationException.java`

```java
package com.greatlearning.security.spring_boot_security.jwt.resource;

public class AuthenticationException extends RuntimeException {
    public AuthenticationException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

- `jwt/resource/JwtAuthenticationRestController.java`

```java
package com.greatlearning.security.spring_boot_security.jwt.resource;

import java.util.Objects;

import jakarta.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.DisabledException;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
@CrossOrigin(origins="http://localhost:3000")
public class JwtAuthenticationRestController {

    @Value("${jwt.http.request.header}")
    private String tokenHeader;

    @RequestMapping(value = "${jwt.get.token.uri}", method = RequestMethod.POST)
    public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtTokenRequest
authenticationRequest)
            throws AuthenticationException {

        // Authenticate the user
        authenticate(authenticationRequest.getUsername(),
authenticationRequest.getPassword());

        // Generate a token
        final String token = "Here is a token for you";

        return ResponseEntity.ok(new JwtTokenResponse(token));
    }

    @ExceptionHandler({ AuthenticationException.class })
    public ResponseEntity<String>
handleAuthenticationException(AuthenticationException e) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(e.getMessage());
    }

    private void authenticate(String username, String password) {
        Objects.requireNonNull(username);
        Objects.requireNonNull(password);

        try {
            // authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(username, password));
            System.out.println( "User needs to be authenticated by Spring Security"
);
        } catch (DisabledException e) {
            throw new AuthenticationException("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new AuthenticationException("INVALID_CREDENTIALS", e);
        }
    }
}
```

- Make a POST request using Postman to `/authenticate` endpoint. Pass user data as shown.

```
POST http://localhost:8080/authenticate
```

Select: Body -> raw -> JSON and hit "Send"

```
{
    "username": "user",
    "password": "dummy"
}
```

- The endpoint is protected and needs a session cookie (`JSESSIONID`) to be added. Get it from the browser - inspect the response for the login request in the browser using the Network tab and get the cookie in `Set-Cookie` header. Alternatively check Application tab -> Cookies.
- Use Postman Cookie editor to set the cookie (which represents a logged in session, and hence allows access to protected resources). Make the request to `http://localhost:8080/jpa/users/user/todos` (allowed) or `http://localhost:8080/authenticate` (forbidden) to get responses.
- **NOTE**: Usually you can get the session cookie from a POST request to `http://localhost:8080/login` with the `username` and `password` fields in the form-data set, but Spring Security enables CSRF protection by default, and hence you will need a CSRF token, which is not available when making the call through Postman (but which is sent in the form on the Login page that open in the browser).

Step 13: Configure authentication for endpoints

- `jwt/JWTWebSecurityConfig.java`

```
package com.greatlearning.security.spring_boot_security.jwt;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCustomi
zer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class JwtWebSecurityConfig {
    @Value("${jwt.get.token.uri}")
    private String authenticationPath;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity)
throws Exception {
        return httpSecurity
                .csrf((csrf) -> csrf.disable())
                .sessionManagement((session) ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests(authorizeRequests ->
                        authorizeRequests
                                .anyRequest().authenticated()
                )
                .build();
    }

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web
                            .ignoring()
                            .requestMatchers(HttpMethod.GET, "/*")
                            .requestMatchers(HttpMethod.POST, authenticationPath)
                            .requestMatchers(HttpMethod.OPTIONS, "/**")
                            .requestMatchers("/h2-console/**");
    }
}
```

Step 14: Create a UserDetails POJO class as required by Spring Security

- `jwt/JwtUserDetails.java`

```java
package com.greatlearning.security.spring_boot_security.jwt;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import com.fasterxml.jackson.annotation.JsonIgnore;

public class JwtUserDetails implements UserDetails {
    private static final long serialVersionUID = 5155720064139820502L;

    private final Long id;
    private final String username;
    private final String password;
    private final Collection<? extends GrantedAuthority> authorities;

    public JwtUserDetails(Long id, String username, String password, String role) {
        this.id = id;
        this.username = username;
        this.password = password;

        List<SimpleGrantedAuthority> authorities = new
ArrayList<SimpleGrantedAuthority>();
        authorities.add(new SimpleGrantedAuthority(role));

        this.authorities = authorities;
    }

    @JsonIgnore
    public Long getId() {
        return id;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @JsonIgnore
    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

Step 15: Create a UserDetailsService to provide Spring Security with user's details (given a username)

- `jwt/JwtInMemoryUserDetailsService.java`

```java
package com.greatlearning.security.spring_boot_security.jwt;

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

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;

@Service
public class JwtInMemoryUserDetailsService implements UserDetailsService {
    static List<JwtUserDetails> inMemoryUserList = new ArrayList<>();

    static {
        inMemoryUserList.add(new JwtUserDetails(1L, "user",
"$2a$10$3zHzb.Npv1hfZbLEU5qsdOju/tk2je6W6PnNnY.c1ujWPcZh4PL6e", "ROLE_USER"));
        inMemoryUserList.add(new JwtUserDetails(2L, "john",
"$2a$10$3zHzb.Npv1hfZbLEU5qsdOju/tk2je6W6PnNnY.c1ujWPcZh4PL6e", "ROLE_USER"));
        inMemoryUserList.add(new JwtUserDetails(3L, "jane",
"$2a$10$3zHzb.Npv1hfZbLEU5qsdOju/tk2je6W6PnNnY.c1ujWPcZh4PL6e", "ROLE_ADMIN"));
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Optional<JwtUserDetails> findFirst = inMemoryUserList.stream()
                .filter(user -> user.getUsername().equals(username)).findFirst();

        if (!findFirst.isPresent()) {
            throw new UsernameNotFoundException(String.format("USER_NOT_FOUND
'%s'.", username));
        }

        return findFirst.get();
    }
}
```

Step 16: Configure the Spring Security AuthenticationManager using the BcryptPasswordEncoder and UserDetailsService

- `jwt/JwtWebSecurityConfig.java`

```
@Configuration
@EnableWebSecurity
public class JwtWebSecurityConfig {
    @Autowired
    private UserDetailsService jwtInMemoryUserDetailsService;

    // @Bean
    // public PasswordEncoder passwordEncoderBean() {
    //     return NoOpPasswordEncoder.getInstance();
    // }

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

    @Bean
    public AuthenticationManager authenticationManager(
            UserDetailsService userDetailsService,
            PasswordEncoder passwordEncoder) {
        DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService);
        authenticationProvider.setPasswordEncoder(passwordEncoder);

        return new ProviderManager(authenticationProvider);
    }

    // rest of code...
}
```

Step 17: Add a JWT Token utility

- `jwt/JwtTokenUtil.java`

```java
package com.greatlearning.security.spring_boot_security.jwt;

import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Clock;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.impl.DefaultClock;

@Component
public class JwtTokenUtil implements Serializable {

    static final String CLAIM_KEY_USERNAME = "sub";
    static final String CLAIM_KEY_CREATED = "iat";
    private static final long serialVersionUID = -3301605591108950415L;
    private Clock clock = DefaultClock.INSTANCE;

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

    @Value("${jwt.token.expiration.in.seconds}")
    private Long expiration;

    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }

    public Date getIssuedAtDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getIssuedAt);
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public <T> T getClaimFromToken(String token, Function<Claims, T>
claimsResolver) {
        final Claims claims = getAllClaimsFromToken(token);
        return claimsResolver.apply(claims);
    }

    private Claims getAllClaimsFromToken(String token) {
        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(clock.now());
    }

    private Boolean ignoreTokenExpiration(String token) {
        // here you specify tokens, for that the expiration is ignored
        return false;
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return doGenerateToken(claims, userDetails.getUsername());
    }

    private String doGenerateToken(Map<String, Object> claims, String subject) {
        final Date createdDate = clock.now();
        final Date expirationDate = calculateExpirationDate(createdDate);

        return
Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(createdDate)
                .setExpiration(expirationDate).signWith(SignatureAlgorithm.HS512,
secret).compact();
    }

    public Boolean canTokenBeRefreshed(String token) {
        return (!isTokenExpired(token) || ignoreTokenExpiration(token));
    }

    public String refreshToken(String token) {
        final Date createdDate = clock.now();
        final Date expirationDate = calculateExpirationDate(createdDate);

        final Claims claims = getAllClaimsFromToken(token);
        claims.setIssuedAt(createdDate);
        claims.setExpiration(expirationDate);

        return Jwts.builder().setClaims(claims).signWith(SignatureAlgorithm.HS512,
secret).compact();
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        JwtUserDetails user = (JwtUserDetails) userDetails;
        final String username = getUsernameFromToken(token);
        return (username.equals(user.getUsername()) && !isTokenExpired(token));
    }

    private Date calculateExpirationDate(Date createdDate) {
        return new Date(createdDate.getTime() + expiration * 1000);
    }
}
```

Step 18: Add a filter, that authenticates based on the JWT Bearer token, to the Spring Security filter chain

- `jwt/JwtTokenAuthorizationOncePerRequestFilter.java`

```
package com.greatlearning.security.spring_boot_security.jwt;

import java.io.IOException;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import
org.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;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import io.jsonwebtoken.ExpiredJwtException;

@Component
public class JwtTokenAuthorizationOncePerRequestFilter extends OncePerRequestFilter
{

    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    @Autowired
    private UserDetailsService jwtInMemoryUserDetailsService;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Value("${jwt.http.request.header}")
    private String tokenHeader;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain chain) throws ServletException, IOException {
        logger.debug("Authentication Request For '{}'", request.getRequestURL());

        final String requestTokenHeader = request.getHeader(this.tokenHeader);

        String username = null;
        String jwtToken = null;
        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer "))
{
            jwtToken = requestTokenHeader.substring(7);
            try {
                username = jwtTokenUtil.getUsernameFromToken(jwtToken);
            } catch (IllegalArgumentException e) {
                logger.error("JWT_TOKEN_UNABLE_TO_GET_USERNAME", e);
            } catch (ExpiredJwtException e) {
                logger.warn("JWT_TOKEN_EXPIRED", e);
            }
        } else {
            logger.warn("JWT_TOKEN_DOES_NOT_START_WITH_BEARER_STRING");
        }

        logger.debug("JWT_TOKEN_USERNAME_VALUE '{}'", username);
        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {

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

            if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {
                UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                usernamePasswordAuthenticationToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthentication
Token);
            }
        }

        chain.doFilter(request, response);
    }
}
```

- `jwt/JwtWebSecurityConfig.java`

```java
@Configuration
@EnableWebSecurity
public class JwtWebSecurityConfig {
    @Autowired
    private JwtTokenAuthorizationOncePerRequestFilter jwtAuthenticationTokenFilter;

    // rest of code...
    // ...

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity)
throws Exception {
        return httpSecurity
                .csrf((csrf) -> csrf.disable())
                .sessionManagement((session) ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests(authorizeRequests ->
                        authorizeRequests
                                // .requestMatchers("/", "/home").permitAll()
                                .anyRequest().authenticated()
                )
                .addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class)
                .build();
    }

    // rest of code...
    // ...
}
```

References

- https://bootify.io/spring-security/rest-api-spring-security-with-jwt.html
- https://spring.io/guides/topicals/spring-security-architecture
- https://stackoverflow.com/questions/56388865/spring-security-configuration-httpsecurity-vs-websecurity

Todos App - Getting started with the frontend

Step 1: Create the React project and start the development server

- Create the app using create-react-app. In the folder of your choice, run the following in the terminal

```
npx create-react-app --template typescript spring-boot-security
```

- Switch to the folder

```
cd spring-boot-security
```

- Start the app in the terminal

```
npm start
```

- View the app on `http://localhost:3000`
- Open another terminal. Keep it free for package installations.

Step 2: Include React Boostrap

- Install it

```
npm i react-bootstrap bootstrap
```

- `src/index.tsx` - Include it.
- You may remove the imports and code that aren't needed in essence (this is an optional step).

```
import React from "react";
import ReactDOM from "react-dom/client";

import App from "./App";

import "bootstrap/dist/css/bootstrap.min.css";

const root = ReactDOM.createRoot(
    document.getElementById("root") as HTMLElement
);

root.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>
);
```

- `src/App.tsx`

```
const App = () => {
    return <div>App component</div>;
};

export default App;
```

- You may remove `App.css` , `App.test.tsx` , `index.css` , `logo.svg` , `setupTests.ts` , `reportWebVitals.ts`

Step 3: Set up `components` and `pages` folders, and add the `Navigation` component

- Create `components` and `pages` folders
- Install React router

```
npm i react-router-dom
```

- `src/index.tsx` - Set up React Router for use in the app

```
import { BrowserRouter } from "react-router-dom";


root.render(
    <React.StrictMode>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </React.StrictMode>
);
```

- `src/components/Navigation/Navigation.tsx`

```
import { Container, Nav, Navbar } from "react-bootstrap";
import { NavLink } from "react-router-dom";

const Navigation = () => {
    return (
        <Navbar expand="lg" bg="primary" data-bs-theme="dark">
            <Container>
                <Navbar.Brand to="/todos" as={NavLink}>
                    Todos
                </Navbar.Brand>
                <Navbar.Toggle aria-controls="basic-navbar-nav" />
                <Navbar.Collapse id="basic-navbar-nav">
                    <Nav className="ms-auto">
                        <Nav.Link as={NavLink} to="/login">
                            Login
                        </Nav.Link>
                        <Nav.Link>Logout</Nav.Link>
                    </Nav>
                </Navbar.Collapse>
            </Container>
        </Navbar>
    );
};

export default Navigation;
```

- `src/components/App.tsx`

```
import { Container } from "react-bootstrap";

import Navigation from "./components/Navigation/Navigation";

const App = () => {
    return (
        <>
            <Navigation />

            <Container className="my-5">
                Page component shall be displayed here
            </Container>
        </>
    );
};

export default App;
```

Step 4: Login component / page

- `src/components/Login/Login.tsx`

```
const Login = () => {
    return <div>Login</div>;
};

export default Login;
```

- `src/pages/login/page.tsx`

```
import Login from "../../components/Login/Login";

const LoginPage = () => {
    return <Login />;
};

export default LoginPage;
```

- `src/App.tsx` - Set up the login route

```
import { Container } from "react-bootstrap";
import { Routes, Route } from "react-router-dom";

import Navigation from "./components/Navigation/Navigation";
import LoginPage from "./pages/login/page";

const App = () => {
    return (
        <>
            <Navigation />

            <Container className="my-5">
                <Routes>
                    <Route path="/" element={<LoginPage />} />
                    <Route path="/login" element={<LoginPage />} />
                </Routes>
            </Container>
        </>
    );
};

export default App;
```

Step 5: Add the login form

- `src/components/Login/Login.tsx`

```
import { Button, Form } from "react-bootstrap";

const Login = () => {
    return (
        <>
            <h1>Login</h1>
            <hr />
            <Form>
                <Form.Group
                    className="mb-4 col-12 col-sm-8 col-md-6 col-lg-4"
                    controlId="username"
                >
                    <Form.Label>Username</Form.Label>
                    <Form.Control type="text" placeholder="johndoe" />
                </Form.Group>
                <Form.Group
                    className="mb-4 col-12 col-sm-8 col-md-6 col-lg-4"
                    controlId="password"
                >
                    <Form.Label>Password</Form.Label>
                    <Form.Control type="password" />
                </Form.Group>
                <Button type="submit" variant="primary">
                    Login
                </Button>
            </Form>
        </>
    );
};

export default Login;
```

- Add state to make it a controlled component. Set it up for submission.

```
import { useState, FormEvent } from "react";
import { Alert, Button, Form } from "react-bootstrap";

const Login = () => {
    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");

    const login = async (event: FormEvent<HTMLFormElement>) => {
        event.preventDefault();

        try {
            // Tip: Never log the password. If ever you do, make sure you remove it
as soon as you can
            console.log(username, password);

            setUsername("");
            setPassword("");
        } catch (error) {}
    };

    return (
        <>
            <h1>Login</h1>
            <hr />
            <Form onSubmit={login}>
                <Form.Group
                    className="mb-4 col-12 col-sm-8 col-md-6 col-lg-4"
                    controlId="username"
                >
                    <Form.Label>Username</Form.Label>
                    <Form.Control
                        type="text"
                        placeholder="johndoe"
                        value={username}
                        onChange={(event) => setUsername(event.target.value)}
                    />
                </Form.Group>
                <Form.Group
                    className="mb-4 col-12 col-sm-8 col-md-6 col-lg-4"
                    controlId="password"
                >
                    <Form.Label>Password</Form.Label>
                    <Form.Control
                        type="password"
                        value={password}
                        onChange={(event) => setPassword(event.target.value)}
                    />
                </Form.Group>
                <Button type="submit" variant="primary">
                    Login
                </Button>
            </Form>
        </>
    );
};

export default Login;
```

Step 6: Set up a notifications service. Use it to show error on login.

- Install UUID to generate unique ids for notifications.

```
npm install uuid @types/uuid
```

- `src/contexts/notifications.tsx`

```
import { createContext, ReactNode, useContext, useState } from "react";
import { Toast, ToastContainer } from "react-bootstrap";
import { v4 as uuidv4 } from "uuid";

type Notification = {
    id?: string;
    header: ReactNode;
    body: ReactNode;
    variant?:
        | "primary"
        | "secondary"
        | "success"
        | "danger"
        | "warning"
        | "info"
        | "dark"
        | "light";
    autohide?: boolean;
    delay?: number;
    show?: boolean;
    onClose?: () => void;
};

type NotificationsContextType = {
    addNotification: (notification: Notification) => void;
};

const NotificationsContext = createContext<NotificationsContextType>({
    addNotification: () => {}, // default to noop
});

const NotificationsProvider = ({ children }: { children: ReactNode }) => {
    const [notifications, setNotifications] = useState<Notification[]>([]);

    const addNotification = (notification: Notification) => {
        setNotifications((notifications) => [
            ...notifications,
            {
                id: uuidv4(),
                show: true,
                variant: "primary",
                delay: 5000,
                ...notification,
            },
        ]);
    };

    const closeNotification = (notification: Notification) => {
        setNotifications((notifications) =>
            notifications.map((n) =>
                n === notification ? { ...n, show: false } : n
            )
        );
    };

    const value = {
        addNotification,
    };

    return (
        <NotificationsContext.Provider value={value}>
            {children}
            <div
                aria-live="polite"
                aria-atomic="true"
                className="position-fixed"
                style={{
                    minHeight: "320px",
                    minWidth: "300px",
                    top: 0,
                    right: 0,
                    pointerEvents: "none",
                }}
                key="notifications-wrapper"
            >
                <ToastContainer
                    position="top-end"
                    className="p-3"
                    style={{ zIndex: 1 }}
                >
                    {notifications.map((n) => {
                        const props = {
                            bg: n.variant,
                            show: n.show,
                            delay: n.delay,
                            autohide: n.autohide,
                            onClose: () => {
                                closeNotification(n);
                                n.onClose && n.onClose();
                            },
                        };

                        return (
                            <Toast key={n.id} {...props}>
                                <Toast.Header>{n.header}</Toast.Header>
                                <Toast.Body>{n.body}</Toast.Body>
                            </Toast>
                        );
                    })}
                </ToastContainer>
            </div>
        </NotificationsContext.Provider>
    );
};
```

```
const useNotifications = () => useContext(NotificationsContext);

export type { Notification, NotificationsContextType };
export { NotificationsProvider, useNotifications };
```

- `src/App.tsx`

```
import { NotificationsProvider } from "./contexts/notifications";


const App = () => {
    return (
        <NotificationsProvider>{/* rest of code... */}</NotificationsProvider>
    );
};

export default App;
```

- `src/components/Login/Login.tsx` - Set up error handling

```
import { useNotifications } from "../../contexts/notifications";

const Login = () => {
    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");

    const { addNotification } = useNotifications();

    const login = async (event: FormEvent<HTMLFormElement>) => {
        event.preventDefault();

        try {
            // Tip: Never log the password. If ever you do, make sure you remove it
as soon as you can
            console.log(username, password);

            setUsername("");
            setPassword("");

            addNotification({
                variant: "success",
                autohide: true,
                header: <strong className="me-auto">Success!</strong>,
                body: (
                    <span className="text-light">
                        Add things to do and track your todos based on their
                        deadlines.
                    </span>
                ),
            });
        } catch (error) {
            addNotification({
                variant: "danger",
                autohide: true,
                header: <strong className="me-auto">Error!</strong>,
                body: (
                    <span className="text-light">
                        {(error as Error).message}
                    </span>
                ),
            });
        }
    };

    // rest of code...
    // ...
};
```

- Alternatively, this complexity can be avoided by using any React notifications (aka toast) library like `react-toastify`

Step 7: Set up and call a service to make the authentication request, and redirect to a Welcome page on success

- Install `axios`

```
npm i axios
```

- `.env` - Add the base url for the service and for the API endpoints in particular.

```
REACT_APP_API_URL=http://localhost:8080
REACT_APP_JPA_API_URL=http://localhost:8080/jpa
```

- `src/services/auth.ts`

```
import axios from "axios";

const baseUrl = process.env.REACT_APP_API_URL;

export const LS_KEY_USERNAME = "username";
export const LS_KEY_TOKEN = "token";

axios.interceptors.request.use((config) => {
    if (isUserLoggedIn()) {
        config.headers.authorization = "Bearer " + getToken();
    }

    return config;
});

const login = async (username: string, password: string) => {
    const response = await axios.post(`${baseUrl}/authenticate`, {
        username,
        password,
    });

    const { token } = response.data;

    localStorage.setItem(LS_KEY_USERNAME, username);
    localStorage.setItem(LS_KEY_TOKEN, token);
};

const logout = () => localStorage.removeItem(LS_KEY_USERNAME);

const isUserLoggedIn = () => !!localStorage.getItem(LS_KEY_USERNAME);

const getUsername = () =>
    isUserLoggedIn() ? localStorage.getItem(LS_KEY_USERNAME) : "";

const getToken = () =>
    isUserLoggedIn() ? localStorage.getItem(LS_KEY_TOKEN) : "";

export { login, logout, isUserLoggedIn, getUsername, getToken };
```

- `src/components/Welcome/Welcome.tsx`

```
const WelcomeComponent = () => {
    return (
        <>
            <h1>Welcome!</h1>
            <p>Welcome John Doe. Click here to manage your todos </p>
        </>
    );
};

export default WelcomeComponent;
```

- `src/pages/welcome/page.tsx`

```
import Welcome from "../../components/Welcome/Welcome";

const WelcomePage = () => {
    return <Welcome />;
};

export default WelcomePage;
```

- `src/App.tsx` - Add a route to show the Welcome page

```
import WelcomePage from "./pages/welcome/page";


<Routes>
    <Route path="/" element={<LoginPage />} />
    <Route path="/login" element={<LoginPage />} />
    <Route path="/welcome/:username" element={<WelcomePage />} />
</Routes>
```

- `src/components/Login/Login.tsx`

```
import { useNavigate } from "react-router-dom";
import { login as LoginService } from "../../services/auth";
```

```
const Login = () => {
    // rest of code...

    const navigate = useNavigate();

    const login = async (event: FormEvent<HTMLFormElement>) => {
        event.preventDefault();

        try {
            await LoginService(username, password);

            navigate(`/welcome/${username}`);

            setUsername("");
            setPassword("");

            addNotification({
                variant: "success",
                autohide: true,
                header: <strong className="me-auto">Success!</strong>,
                body: (
                    <span className="text-light">
                        Add things to do and track your todos based on their
                        deadlines.
                    </span>
                ),
            });
        } catch (error) {
            // rest of code...
        }
    };

    // rest of code...
};

// rest of code...
```

- **IMPORTANT**: Make sure to restart the dev server - the changes in `.env` file are read only at startup time.

Step 8: Show the username, and set up link to navigate to TodosList page

- `src/components/Welcome/Welcome.tsx`

```tsx
import { Link, useParams } from "react-router-dom";

type Params = {
    username: string | undefined;
};

const WelcomeComponent = () => {
    const { username } = useParams<Params>();

    return (
        <>
            <h1>Welcome!</h1>
            <p>
                Welcome {username}. Click <Link to="/todos">here</Link> to
                manage your todos
            </p>
        </>
    );
};

export default WelcomeComponent;
```

- `src/components/todos/TodosList/TodosList.tsx`

```tsx
const TodosList = () => {
    return (
        <>
            <h1>Todos</h1>
            <hr />
        </>
    );
};

export default TodosList;
```

- `src/pages/todos/page.tsx`

```tsx
import TodosList from "../../components/todos/TodosList/TodosList";

const TodosListPage = () => {
    return <TodosList />;
};

export default TodosListPage;
```

- `src/App.tsx`

```tsx
import TodosListPage from "./pages/todos/page";
```

```
<Routes>
    <Route path="/" element={<LoginPage />} />
    <Route path="/login" element={<LoginPage />} />
    <Route path="/welcome/:username" element={<WelcomePage />} />
    <Route path="/todos" element={<TodosListPage />} />
</Routes>
```

Step 9: Fetch the todos and show them

- `src/types/utils/ts` - Add utility types like `NS` here

```
export type NS = number | string;
```

- `src/services/todos.ts`

```
import axios from "axios";
import { getUsername } from "./auth";
import { NS } from "../types/utils";

type Todo = {
    id?: number;
    username: string;
    description: string;
    targetDate: string;
};

const jpaBaseUrl = process.env.REACT_APP_JPA_API_URL;

const getTodosBaseUrl = () => `${jpaBaseUrl}/users/${getUsername()}/todos`;

const getTodos = async () => {
    const response = await axios.get<Todo[]>(getTodosBaseUrl());
    return response.data;
};

const getTodoById = async (id: NS) => {
    const response = await axios.get<Todo>(`${getTodosBaseUrl()}/${id}`);
    return response.data;
};

const postTodo = async (todo: Todo) => {
    const response = await axios.post<Todo>(getTodosBaseUrl(), todo);
    return response.data;
};

const putTodo = async (id: NS, todo: Todo) => {
    const response = await axios.put(`${getTodosBaseUrl()}/${id}`, todo);
    return response.data;
};

const deleteTodo = async (id: NS) => {
    await axios.delete(`${getTodosBaseUrl()}/${id}`);
};

export type { Todo };
export { getTodos, getTodoById, postTodo, putTodo, deleteTodo };
```

- Install moment (we use it to format dates). You may also use an alternative library like date-fns.

```
npm i moment
```

- `src/components/todos/TodosList/TodosList.tsx`

```
import { useState, useEffect } from "react";
import { Button, Spinner, Table } from "react-bootstrap";
import moment from "moment";

import { useNotifications } from "../../../contexts/notifications";
import { Todo, getTodos } from "../../../services/todos";
import { NS } from "../../../types/utils";

const TodosList = () => {
    const [todos, setTodos] = useState<Todo[]>([]);
    const [loading, setLoading] = useState(true);

    const { addNotification } = useNotifications();

    const refreshTodos = async () => {
        try {
            const todos = await getTodos();
            setTodos(todos);
        } catch (error) {
            addNotification({
                variant: "danger",
                autohide: true,
                header: <strong className="me-auto">Error!</strong>,
                body: (
                    <span className="text-light">
                        {(error as Error).message}
                    </span>
                ),
            });
        } finally {
            setLoading(false);
        }
    };

    useEffect(() => {
        refreshTodos();
    }, []);

    return (
        <>
            <h1 className="d-flex justify-content-between align-items-center">
                Todos
                <Button variant="primary">Add</Button>
            </h1>
            <hr />
            {loading && (
                <div className="text-center my-5">
                    <Spinner />
                </div>
            )}
            <Table striped bordered hover>
                <thead>
                    <tr>
                        <th>Description</th>
                        <th>Target Date</th>
                        <th>Update</th>
                        <th>Delete</th>
                    </tr>
                </thead>
                <tbody>
                    {todos.map((todo) => (
                        <tr key={todo.id}>
                            <td>{todo.description}</td>
                            <td>
                                {moment(todo.targetDate).format("YYYY-MM-DD")}
                            </td>
                            <td>
                                <Button variant="primary">Update</Button>
                            </td>
                            <td>
                                <Button variant="warning">Delete</Button>
                            </td>
                        </tr>
                    ))}
                </tbody>
            </Table>
        </>
    );
};

export default TodosList;
```

Step 10: Delete todo feature

- `src/components/TodosList/TodosList.tsx`
- Import the deleteTodod service

```
import { Todo, getTodos, deleteTodo } from "../../../services/todos";
```

- Add this method within the component function

```
const deleteTodoClicked = async (id: NS) => {
    if (!window.confirm("Are you sure you want to proceed?")) {
        return;
    }

    try {
        await deleteTodo(id);

        addNotification({
            variant: "success",
            autohide: true,
            header: <strong className="me-auto">Success!</strong>,
            body: (
                <span className="text-light">
                    Successfully deleted the todo
                </span>
            ),
        });

        refreshTodos();
    } catch (error) {
        addNotification({
            variant: "danger",
            autohide: true,
            header: <strong className="me-auto">Error!</strong>,
            body: (
                <span className="text-light">{(error as Error).message}</span>
            ),
        });
    }
};
```

- Call this method on click of the Delete button for a todo.

```
<Button variant="warning" onClick={() => deleteTodoClicked(todo.id as NS)}>
    Delete
</Button>
```

Step 11: Set up the Add / Edit todo page

- `src/components/todos/TodoAddEdit/TodoAddEdit.tsx`

```
const TodoAddEdit = () => {
    return (
        <>
            <h1>Add / Edit a Todo</h1>
            <hr />
        </>
    );
};

export default TodoAddEdit;
```

- `src/pages/todos/[id]/page.tsx`

```
import TodoAddEdit from "../../../components/todos/TodoAddEdit/TodoAddEdit";

const TodoAddEditPage = () => {
    return <TodoAddEdit />;
};

export default TodoAddEditPage;
```

- `src/components/todos/TodosList/TodosList.tsx`
- Add the necessary import

```
import { useNavigate } from "react-router-dom";
```

- Add the call to the hook, and the following methods in the component function, and set up the methods as button click event handlers

```
const navigate = useNavigate();

const addTodoClicked = () => {
    navigate(`/todos/-1`);
};

const updateTodoClicked = (id: NS) => {
    navigate(`/todos/${id}`);
};


<h1 className="d-flex justify-content-between align-items-center">
    Todos
    <Button variant="primary" onClick={addTodoClicked}>
        Add
    </Button>
</h1>
```

```
<Button variant="primary" onClick={() => updateTodoClicked(todo.id as NS)}>
    Update
</Button>
```

- `src/App.tsx`
- Add the necessary import

```
import TodoAddEditPage from "./pages/todos/[id]/page";
```

- Add the new Route

```
<Routes>
    <Route path="/" element={<LoginPage />} />
    <Route path="/login" element={<LoginPage />} />
    <Route path="/welcome/:username" element={<WelcomePage />} />
    <Route path="/todos" element={<TodosListPage />} />
    <Route path="/todos/:id" element={<TodoAddEditPage />} />
</Routes>
```

Step 12: Add the Add / Edit todo feature (with form validation using Formik)

- Install Formik for form validations. An alternative is react-hook-form

```
npm i formik
```

- `src/components/todos/TodoAddEdit/TodoAddEdit.tsx`

```
import { useState, useEffect } from "react";
import { useParams, useNavigate } from "react-router-dom";
import { Button } from "react-bootstrap";
import { Formik, Form, Field, ErrorMessage } from "formik";
import moment from "moment";

import { Todo, getTodoById, postTodo, putTodo } from "../../../services/todos";
import { getUsername } from "../../../services/auth";

type Params = {
    id: string | undefined;
};

const TodoComponent = () => {
    const { id } = useParams<Params>();
    const navigate = useNavigate();

    const [description, setDescription] = useState("");
    const [targetDate, setTargetDate] = useState(() =>
        moment(new Date()).format("YYYY-MM-DD")
    );

    useEffect(() => {
        if (id === "-1") {
            return;
        }

        const helper = async () => {
            const data = await getTodoById(id as string);

            setDescription(data.description);
            setTargetDate(moment(data.targetDate).format("YYYY-MM-DD"));
        };

        helper();
    }, []);

    const validate = (values: Omit<Todo, "id" | "username">) => {
        let errors = {} as Todo;

        if (!values.description) {
            errors.description = "Enter a Description";
        } else if (values.description.length < 5) {
            errors.description = "Enter atleast 5 Characters in Description";
        }

        if (!moment(values.targetDate).isValid()) {
            errors.targetDate = "Enter a valid Target Date";
        }

        return errors;
    };
```

```
const onSubmit = async (values: Omit<Todo, "id" | "username">) => {
    const todo = {
        id: id,
        username: getUsername(),
        description: values.description,
        targetDate: values.targetDate,
    } as Todo;

    if (id === "-1") {
        await postTodo(todo);
    } else {
        await putTodo(id as string, todo);
    }

    navigate("/todos");
};

return (
    <>
        <h1>Add / Edit a Todo</h1>
        <hr />
        <Formik
            initialValues={{ description, targetDate }}
            onSubmit={onSubmit}
            validateOnChange={true}
            validateOnBlur={true}
            validate={validate}
            enableReinitialize={true}
        >
            {() => (
                <Form noValidate>
                    <ErrorMessage
                        name="description"
                        component="div"
                        className="alert alert-warning"
                    />
                    <ErrorMessage
                        name="targetDate"
                        component="div"
                        className="alert alert-warning"
                    />
                    <fieldset className="form-group mb-3">
                        <label>Description</label>
                        <Field
                            className="form-control"
                            type="text"
                            name="description"
                        />
                    </fieldset>
                    <fieldset className="form-group mb-3">
                        <label>Target Date</label>
```

```
                                        <Field
                                            className="form-control"
                                            type="date"
                                            name="targetDate"
                                        />
                                    </fieldset>
                                    <Button variant="primary" type="submit">
                                        Save
                                    </Button>
                                </Form>
                            )}
                        </Formik>
                    </>
                );
            };

export default TodoComponent;
```

Step 13: Add logout functionality and update the Navigation component

- `src/components/Navigation/Navigation.tsx`

```
import { Container, Nav, Navbar } from "react-bootstrap";
import { NavLink, useNavigate } from "react-router-dom";
import { isUserLoggedIn, logout } from "../../services/auth";

const Navigation = () => {
    const loggedIn = isUserLoggedIn();
    const navigate = useNavigate();

    return (
        <Navbar expand="lg" bg="primary" data-bs-theme="dark">
            <Container>
                {loggedIn && (
                    <Navbar.Brand to="/todos" as={NavLink}>
                        Todos
                    </Navbar.Brand>
                )}
                <Navbar.Toggle aria-controls="basic-navbar-nav" />
                <Navbar.Collapse id="basic-navbar-nav">
                    <Nav className="ms-auto">
                        {!loggedIn && (
                            <Nav.Link as={NavLink} to="/login">
                                Login
                            </Nav.Link>
                        )}
                        {loggedIn && (
                            <Nav.Link
                                onClick={() => {
                                    logout();
                                    navigate("/");
                                }}
                            >
                                Logout
                            </Nav.Link>
                        )}
                    </Nav>
                </Navbar.Collapse>
            </Container>
        </Navbar>
    );
};

export default Navigation;
```

Step 14: Protect the pages that should be inaccessible to users who are not logged in

- `src/components/AuthenticatedRoute/AuthenticatedRoute.tsx`

```
import { ReactElement } from "react";
import { Navigate } from "react-router-dom";
import { isUserLoggedIn } from "../../services/auth";

type Props = {
    children: ReactElement;
};

const AuthenticatedRoute = ({ children }: Props) => {
    if (isUserLoggedIn()) {
        return children;
    } else {
        return <Navigate to="/login" />;
    }
};

export default AuthenticatedRoute;
```

- `src/App.tsx` - Protect routes using this component

```
import AuthenticatedRoute from
"./components/AuthenticatedRoute/AuthenticatedRoute";
```

```
<Routes>
    <Route path="/" element={<LoginPage />} />
    <Route path="/login" element={<LoginPage />} />
    <Route
        path="/welcome/:username"
        element={
            <AuthenticatedRoute>
                <WelcomePage />
            </AuthenticatedRoute>
        }
    />
    <Route
        path="/todos"
        element={
            <AuthenticatedRoute>
                <TodosListPage />
            </AuthenticatedRoute>
        }
    />
    <Route
        path="/todos/:id"
        element={
            <AuthenticatedRoute>
                <TodoAddEditPage />
            </AuthenticatedRoute>
        }
    />
</Routes>
```

- Logout and try navigating to `http://localhost:3000/todos` - You will be redirected to the login page.