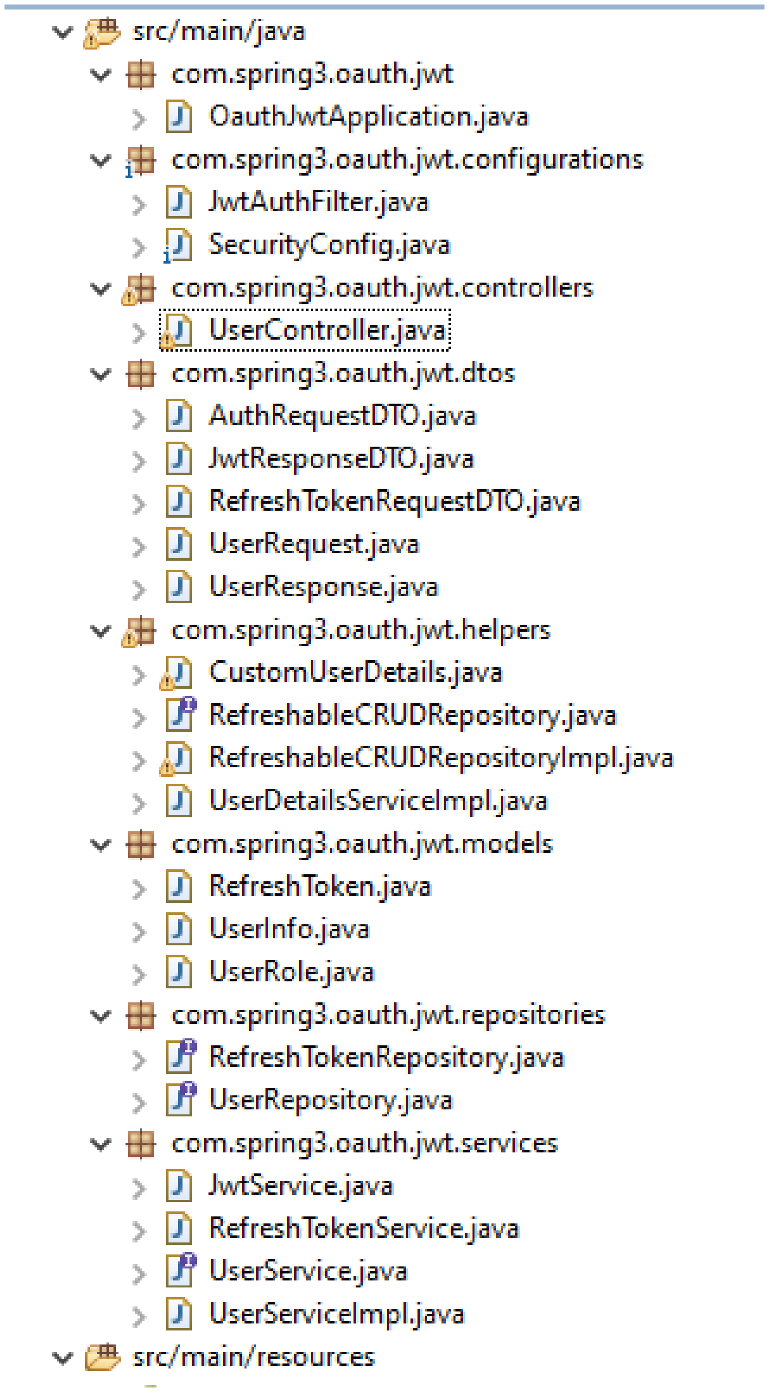
Implementing JWT in Java Spring

1. New Spring boot project
2. Add the maven dependencies

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.spring3.oauth.jwt</groupId>
  <artifactId>oauth-jwt</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>oauth-jwt</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->
    <dependency>
      <groupId>org.modelmapper</groupId>
      <artifactId>modelmapper</artifactId>
      <version>3.1.1</version>
    </dependency>

    <dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-api</artifactId>
      <version>0.11.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-impl</artifactId>
      <version>0.11.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-jackson</artifactId>
      <version>0.11.5</version>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
           <groupId>javax.xml.bind</groupId>
                  <artifactId>jaxb-api</artifactId>
                  <version>2.3.0</version>
                  </dependency>
  </dependencies>

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

</project>
```

3. Construct project hierarchy



```
src/main/java
  com.spring3.oauth.jwt
    OauthJwtApplication.java
  com.spring3.oauth.jwt.configurations
    JwtAuthFilter.java
    SecurityConfig.java
  com.spring3.oauth.jwt.controllers
    UserController.java
  com.spring3.oauth.jwt.dtos
    AuthRequestDTO.java
    JwtResponseDTO.java
    RefreshTokenRequestDTO.java
    UserRequest.java
    UserResponse.java
  com.spring3.oauth.jwt.helpers
    CustomUserDetails.java
    RefreshableCRUDRepository.java
    RefreshableCRUDRepositoryImpl.java
    UserDetailsServiceImpl.java
  com.spring3.oauth.jwt.models
    RefreshToken.java
    UserInfo.java
    UserRole.java
  com.spring3.oauth.jwt.repositories
    RefreshTokenRepository.java
    UserRepository.java
  com.spring3.oauth.jwt.services
    JwtService.java
    RefreshTokenService.java
    UserService.java
    UserServiceImpl.java
src/main/resources
```

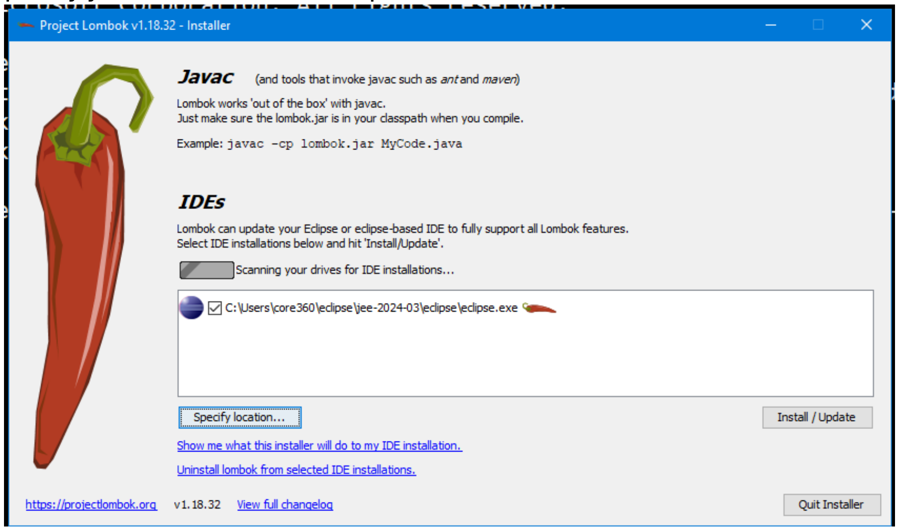
4. Note: make sure Lombok is working
    a. Test Lombok in a class
    b. If not working, we have to run Lombok from the .m2 folder

```
C:\Users\core360\.m2\repository\org\projectlombok\lombok\1.18.32
```

    c. Open a terminal in that directory and run the Lombok-x.x.x.jar in terminal

```
java -jar Lombok-1.x.x.x.jar
```

    d. Then specify your IDE where to setup Lombok



Project Lombok v1.18.32 - Installer

Javac (and tools that invoke javac such as *ant* and *maven*)

Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.

Example: `javac -cp lombok.jar MyCode.java`

IDEs

Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.

Scanning your drives for IDE installations...

[✓] C:\Users\core360\eclipse\jee-2024-03\eclipse\eclipse.exe

Specify location...
Install / Update
Show me what this installer will do to my IDE installation.
Uninstall lombok from selected IDE installations.
https://projectlombok.org
v1.18.32
View full changelog
Quit Installer

    e. Close and re-open your IDE. Do not use Eclipse "Restart".

5. Create a database (ie, MySQL)
6. Modify application properties

application.properties

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
server.port=9898

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/psaspringdb1
spring.datasource.username=psaadmin
spring.datasource.password=123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA Configuration
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update
```

7. Spring boot application (entry point)

OauthJwtApplication.java

```
package com.spring3.oauth.jwt;

import com.spring3.oauth.jwt.helpers.RefreshableCRUDRepositoryImpl;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories(repositoryBaseClass = RefreshableCRUDRepositoryImpl.class)
@SpringBootApplication
public class OauthJwtApplication {

  public static void main(String[] args) {
    SpringApplication.run(OauthJwtApplication.class, args);
  }

}
```

8. Create models

UserInfo.java

```
package com.spring3.oauth.jwt.models;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.HashSet;
import java.util.Set;


@Entity
@Data
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "USERS")
public class UserInfo {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name = "ID")
  private long id;
  private String username;
```

```
   @JsonIgnore
   private String password;

   @ManyToMany(fetch = FetchType.EAGER)
   private Set<UserRole> roles = new HashSet<>();

    //getters and setters as necessary
}
```

UserRole.java

```
package com.spring3.oauth.jwt.models;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Entity
@Data
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "ROLES")
public class UserRole {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = "ID")
   private long id;
   private String name;

    //getter and setters as necessary
}
```

RefreshToken.java

```
package com.spring3.oauth.jwt.models;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.Instant;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
```

```
@Builder
@Table(name = "REFRESH_TOKENS")
public class RefreshToken {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private String token;

   private Instant expiryDate;

   @OneToOne
   @JoinColumn(name = "user_id", referencedColumnName = "id")
   private UserInfo userInfo;

    //getters and setters
}
```

9. Make Data Transfer Objects (DTOs)

AuthRequestDTO.java

```
package com.spring3.oauth.jwt.dtos;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class AuthRequestDTO {

   private String username;
   private String password;

   //getters and setters
}
```

JwtResponseDTO.java

```
package com.spring3.oauth.jwt.dtos;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
```

```
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class JwtResponseDTO {

   private String accessToken;
   private String token;

    // getters and setters as necessary
}
```

RefreshTokenRequestDTO.java

```
package com.spring3.oauth.jwt.dtos;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class RefreshTokenRequestDTO {
   private String token;

    //getters and setters as necessary
}
```

UserRequest.java

```
package com.spring3.oauth.jwt.dtos;

import com.spring3.oauth.jwt.models.UserRole;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
import java.util.Set;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class UserRequest {

   private Long id;
   private String username;
   private String password;
   private Set<UserRole> roles;
```

```
    //getters and setters as necessary

}
```

UserResponse.java

```
package com.spring3.oauth.jwt.dtos;

import com.spring3.oauth.jwt.models.UserRole;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
import java.util.Set;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class UserResponse {

    private Long id;
    private String username;
    private Set<UserRole> roles;

    //getters and setters as necessary

}
```

10. Create main repositories

RefreshTokenRepository.java

```
package com.spring3.oauth.jwt.repositories;

import com.spring3.oauth.jwt.helpers.RefreshableCRUDRepository;
import com.spring3.oauth.jwt.models.RefreshToken;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface RefreshTokenRepository extends RefreshableCRUDRepository<RefreshToken, Integer> {

    Optional<RefreshToken> findByToken(String token);
}
```

UserRepository.java

```
package com.spring3.oauth.jwt.repositories;

import com.spring3.oauth.jwt.helpers.RefreshableCRUDRepository;
```

```java
import com.spring3.oauth.jwt.models.UserInfo;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends RefreshableCRUDRepository<UserInfo, Long> {

  public UserInfo findByUsername(String username);
  UserInfo findFirstById(Long id);

}
```

11. Create helpers

CustomUserDetails.java

```java
package com.spring3.oauth.jwt.helpers;

import com.spring3.oauth.jwt.models.UserInfo;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import com.spring3.oauth.jwt.models.UserRole;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public class CustomUserDetails extends UserInfo implements UserDetails {

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

   public CustomUserDetails(UserInfo byUsername) {
      this.username = byUsername.getUsername();
      this.password= byUsername.getPassword();
      List<GrantedAuthority> auths = new ArrayList<>();

      for(UserRole role : byUsername.getRoles()){
         auths.add(new SimpleGrantedAuthority(role.getName().toUpperCase()));
      }
      this.authorities = auths;
   }

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

   @Override
```

```
  public String getPassword() {
    return password;
  }

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

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

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

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

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

RefreshableCRUDRepository.java

```
package com.spring3.oauth.jwt.helpers;

import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.NoRepositoryBean;
import java.util.Collection;

@NoRepositoryBean
public interface RefreshableCRUDRepository<T, ID> extends CrudRepository<T, ID> {
  void refresh(T t);
  void refresh(Collection<T> s);
  void flush();
}
```

RefreshableCRUDRepositoryImpl.java

```
package com.spring3.oauth.jwt.helpers;

import jakarta.persistence.EntityManager;
import org.springframework.data.jpa.repository.support.JpaEntityInformation;
import org.springframework.data.jpa.repository.support.SimpleJpaRepository;
import org.springframework.data.repository.NoRepositoryBean;
import org.springframework.transaction.annotation.Transactional;

import java.util.Collection;

@NoRepositoryBean
public class RefreshableCRUDRepositoryImpl<T, ID> extends SimpleJpaRepository<T, ID> implements
RefreshableCRUDRepository<T, ID> {

  private final EntityManager entityManager;

  public RefreshableCRUDRepositoryImpl(JpaEntityInformation entityInformation, EntityManager entityManager){
    super(entityInformation, entityManager);
    this.entityManager = entityManager;
  }

  @Override
  @Transactional
  public void flush(){
    entityManager.flush();
  }

  @Override
  @Transactional
  public void refresh(T t) {
    entityManager.refresh(t);
  }

  @Override
  @Transactional
  public void refresh(Collection<T> s) {
    for (T t: s){
      entityManager.refresh(t);
    }
  }
}
```

UserDetailsServiceImpl.java

```
package com.spring3.oauth.jwt.helpers;

import com.spring3.oauth.jwt.models.UserInfo;
import com.spring3.oauth.jwt.repositories.UserRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
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.Component;

@Component
public class UserDetailsServiceImpl implements UserDetailsService {

  @Autowired
  private UserRepository userRepository;

  private static final Logger logger = LoggerFactory.getLogger(UserDetailsServiceImpl.class);

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

    logger.debug("Entering in loadUserByUsername Method...");
    UserInfo user = userRepository.findByUsername(username);
    if(user == null){
      logger.error("Username not found: " + username);
      throw new UsernameNotFoundException("could not found user..!!");
    }
    logger.info("User Authenticated Successfully..!!!");
    return new CustomUserDetails(user);
  }
}
```

12. Create services

JwtService.java

```
package com.spring3.oauth.jwt.services;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class JwtService {

  public static final String SECRET =
"357638792F423F4428472B4B6250655368566D597133743677397A2443264629";
```

```java
public String extractUsername(String token) {
  return extractClaim(token, Claims::getSubject);
}

public Date extractExpiration(String token) {
  return extractClaim(token, Claims::getExpiration);
}

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

private Claims extractAllClaims(String token) {
  return Jwts
      .parserBuilder()
      .setSigningKey(getSignKey())
      .build()
      .parseClaimsJws(token)
      .getBody();
}

private Boolean isTokenExpired(String token) {
  return extractExpiration(token).before(new Date());
}

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

public String GenerateToken(String username){
  Map<String, Object> claims = new HashMap<>();
  return createToken(claims, username);
}

private String createToken(Map<String, Object> claims, String username) {

  return Jwts.builder()
      .setClaims(claims)
      .setSubject(username)
      .setIssuedAt(new Date(System.currentTimeMillis()))
      .setExpiration(new Date(System.currentTimeMillis()+1000*60*10))
      .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
}

private Key getSignKey() {
  byte[] keyBytes = Decoders.BASE64.decode(SECRET);
  return Keys.hmacShaKeyFor(keyBytes);
```

```
    }
}
```

RefreshTokenService.java

```
package com.spring3.oauth.jwt.services;

import com.spring3.oauth.jwt.models.RefreshToken;
import com.spring3.oauth.jwt.repositories.RefreshTokenRepository;
import com.spring3.oauth.jwt.repositories.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.Instant;
import java.util.Optional;
import java.util.UUID;

@Service
public class RefreshTokenService {

    @Autowired
    RefreshTokenRepository refreshTokenRepository;

    @Autowired
    UserRepository userRepository;

    public RefreshToken createRefreshToken(String username){
        RefreshToken refreshToken = RefreshToken.builder()
                .userInfo(userRepository.findByUsername(username))
                .token(UUID.randomUUID().toString())
                .expiryDate(Instant.now().plusMillis(600000))
                .build();
        return refreshTokenRepository.save(refreshToken);
    }

    public Optional<RefreshToken> findByToken(String token){
        return refreshTokenRepository.findByToken(token);
    }

    public RefreshToken verifyExpiration(RefreshToken token){
        if(token.getExpiryDate().compareTo(Instant.now())<0){
            refreshTokenRepository.delete(token);
            throw new RuntimeException(token.getToken() + " Refresh token is expired. Please make a new login..!");
        }
        return token;

    }

}
```

UserService.java

```
package com.spring3.oauth.jwt.services;

import com.spring3.oauth.jwt.dtos.UserRequest;
import com.spring3.oauth.jwt.dtos.UserResponse;
import java.util.List;

public interface UserService {
   UserResponse saveUser(UserRequest userRequest);
   UserResponse getUser();
   List<UserResponse> getAllUser();
}
```

UserServiceImpl.java

```
package com.spring3.oauth.jwt.services;

import com.spring3.oauth.jwt.dtos.UserRequest;
import com.spring3.oauth.jwt.dtos.UserResponse;
import com.spring3.oauth.jwt.models.UserInfo;
import com.spring3.oauth.jwt.repositories.UserRepository;
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import java.lang.reflect.Type;
import java.util.List;

@Service
public class UserServiceImpl implements UserService {

   @Autowired
   UserRepository userRepository;

   ModelMapper modelMapper = new ModelMapper();

   @Override
   public UserResponse saveUser(UserRequest userRequest) {
      if(userRequest.getUsername() == null){
         throw new RuntimeException("Parameter username is not found in request..!!");
      } else if(userRequest.getPassword() == null){
         throw new RuntimeException("Parameter password is not found in request..!!");
      }


//       Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
//      UserDetails userDetail = (UserDetails) authentication.getPrincipal();
//      String usernameFromAccessToken = userDetail.getUsername();
//
//      UserInfo currentUser = userRepository.findByUsername(usernameFromAccessToken);

    UserInfo savedUser = null;

    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
    String rawPassword = userRequest.getPassword();
    String encodedPassword = encoder.encode(rawPassword);

    UserInfo user = modelMapper.map(userRequest, UserInfo.class);
    user.setPassword(encodedPassword);
    if(userRequest.getId() != null){
      UserInfo oldUser = userRepository.findFirstById(userRequest.getId());
      if(oldUser != null){
        oldUser.setId(user.getId());
        oldUser.setPassword(user.getPassword());
        oldUser.setUsername(user.getUsername());
        oldUser.setRoles(user.getRoles());

        savedUser = userRepository.save(oldUser);
        userRepository.refresh(savedUser);
      } else {
        throw new RuntimeException("Can't find record with identifier: " + userRequest.getId());
      }
    } else {
//        user.setCreatedBy(currentUser);
      savedUser = userRepository.save(user);
    }
    userRepository.refresh(savedUser);
    UserResponse userResponse = modelMapper.map(savedUser, UserResponse.class);
    return userResponse;
  }

  @Override
  public UserResponse getUser() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    UserDetails userDetail = (UserDetails) authentication.getPrincipal();
    String usernameFromAccessToken = userDetail.getUsername();
    UserInfo user = userRepository.findByUsername(usernameFromAccessToken);
    UserResponse userResponse = modelMapper.map(user, UserResponse.class);
    return userResponse;
  }

  @Override
  public List<UserResponse> getAllUser() {
    List<UserInfo> users = (List<UserInfo>) userRepository.findAll();
    Type setOfDTOsType = new TypeToken<List<UserResponse>>(){}.getType();
    List<UserResponse> userResponses = modelMapper.map(users, setOfDTOsType);
    return userResponses;
  }


}
```

13. Create configurations

JwtAuthFilter.java

```
package com.spring3.oauth.jwt.configurations;

import com.spring3.oauth.jwt.helpers.UserDetailsServiceImpl;
import com.spring3.oauth.jwt.services.JwtService;
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 JwtAuthFilter extends OncePerRequestFilter {

  @Autowired
  private JwtService jwtService;

  @Autowired
  UserDetailsServiceImpl userDetailsServiceImpl;

  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain) throws ServletException, IOException {

    String authHeader = request.getHeader("Authorization");
    String token = null;
    String username = null;
    if(authHeader != null && authHeader.startsWith("Bearer ")){
      token = authHeader.substring(7);
      username = jwtService.extractUsername(token);
    }

```

```
      if(username != null && SecurityContextHolder.getContext().getAuthentication() == null){
         UserDetails userDetails = userDetailsServiceImpl.loadUserByUsername(username);
         if(jwtService.validateToken(token, userDetails)){
            UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
            authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
         }

      }

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

SecurityConfig.java

```
package com.spring3.oauth.jwt.configurations;

import com.spring3.oauth.jwt.helpers.UserDetailsServiceImpl;
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.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
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
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

   @Autowired
   JwtAuthFilter jwtAuthFilter;

   @Bean
   public UserDetailsService userDetailsService(){
      return new UserDetailsServiceImpl();
   }
```

```
  @SuppressWarnings("removal")
        @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.csrf().disable()
        .authorizeHttpRequests()
        .requestMatchers("/api/v1/save", "/api/v1/login", "/api/v1/refreshToken").permitAll()
        .and()
        .authorizeHttpRequests().requestMatchers("/api/v1/**")
        .authenticated()
        .and()
        .sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .authenticationProvider(authenticationProvider())
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class).build();

  }

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

  @Bean
  public AuthenticationProvider authenticationProvider(){
    DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(userDetailsService());
    authenticationProvider.setPasswordEncoder(passwordEncoder());
    return authenticationProvider;

  }

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

}
```

14. Create controller(s)

UserController.java

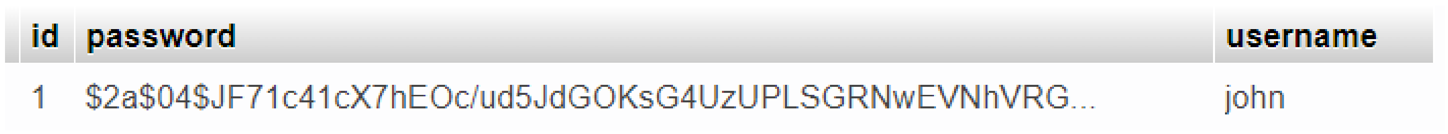
```
package com.spring3.oauth.jwt.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.spring3.oauth.jwt.dtos.AuthRequestDTO;
import com.spring3.oauth.jwt.dtos.JwtResponseDTO;
import com.spring3.oauth.jwt.dtos.RefreshTokenRequestDTO;
import com.spring3.oauth.jwt.dtos.UserRequest;
import com.spring3.oauth.jwt.dtos.UserResponse;
import com.spring3.oauth.jwt.models.RefreshToken;
import com.spring3.oauth.jwt.services.JwtService;
import com.spring3.oauth.jwt.services.RefreshTokenService;
import com.spring3.oauth.jwt.services.UserService;


@RestController
@RequestMapping("/api/v1")
public class UserController {

  @Autowired
  UserService userService;

  @Autowired
  private JwtService jwtService;

  @Autowired
  RefreshTokenService refreshTokenService;


  @Autowired
  private  AuthenticationManager authenticationManager;

  @PostMapping(value = "/save")
  public ResponseEntity saveUser(@RequestBody UserRequest userRequest) {
    try {
      UserResponse userResponse = userService.saveUser(userRequest);
      return ResponseEntity.ok(userResponse);
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
  @GetMapping("/users")
  public ResponseEntity getAllUsers() {
    try {
      List<UserResponse> userResponses = userService.getAllUser();
      return ResponseEntity.ok(userResponses);
    } catch (Exception e){
      throw new RuntimeException(e);
    }
  }


  @PostMapping("/profile")
  public ResponseEntity<UserResponse> getUserProfile() {
    try {
    UserResponse userResponse = userService.getUser();
    return ResponseEntity.ok().body(userResponse);
    } catch (Exception e){
      throw new RuntimeException(e);
    }
  }

  @PreAuthorize("hasAuthority('ROLE_ADMIN')")
  @GetMapping("/test")
  public String test() {
    try {
      return "Welcome";
    } catch (Exception e){
      throw new RuntimeException(e);
    }
  }

  @PostMapping("/login")
  public JwtResponseDTO AuthenticateAndGetToken(@RequestBody AuthRequestDTO authRequestDTO){
    Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(authRequestDTO.getUsername(), authRequestDTO.getPassword()));
    if(authentication.isAuthenticated()){
      RefreshToken refreshToken = refreshTokenService.createRefreshToken(authRequestDTO.getUsername());
     return JwtResponseDTO.builder()
           .accessToken(jwtService.GenerateToken(authRequestDTO.getUsername()))
           .token(refreshToken.getToken()).build();

    } else {
      throw new UsernameNotFoundException("invalid user request..!!");
    }

  }


  @PostMapping("/refreshToken")
  public JwtResponseDTO refreshToken(@RequestBody RefreshTokenRequestDTO refreshTokenRequestDTO){
    return refreshTokenService.findByToken(refreshTokenRequestDTO.getToken())
        .map(refreshTokenService::verifyExpiration)
        .map(RefreshToken::getUserInfo)
        .map(userInfo -> {
          String accessToken = jwtService.GenerateToken(userInfo.getUsername());
          return JwtResponseDTO.builder()
              .accessToken(accessToken)
              .token(refreshTokenRequestDTO.getToken()).build();
        }).orElseThrow(() ->new RuntimeException("Refresh Token is not in DB..!!"));
  }

}
```

15. Add a user directly into the database, for the password, use a bcrypted password. You may use an online service to convert a plaintext password to bcrypt:

```
https://www.devglan.com/online-tools/bcrypt-hash-generator
```

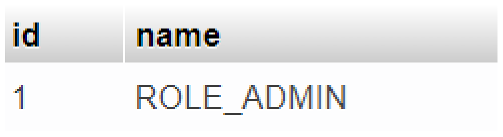
Example:

<u>**users table**</u>
user="john", password="123"



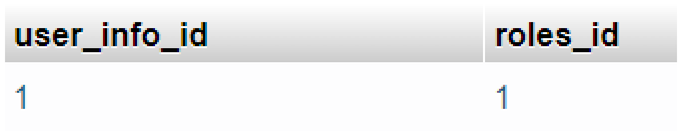
id	password	username
1	$2a$04$JF71c41cX7hEOc/ud5JdGOKsG4UzUPLSGRNwEVNhVRG...	john

<u>**roles table**</u>



id	name
1	ROLE_ADMIN

<u>**users_roles table**</u>

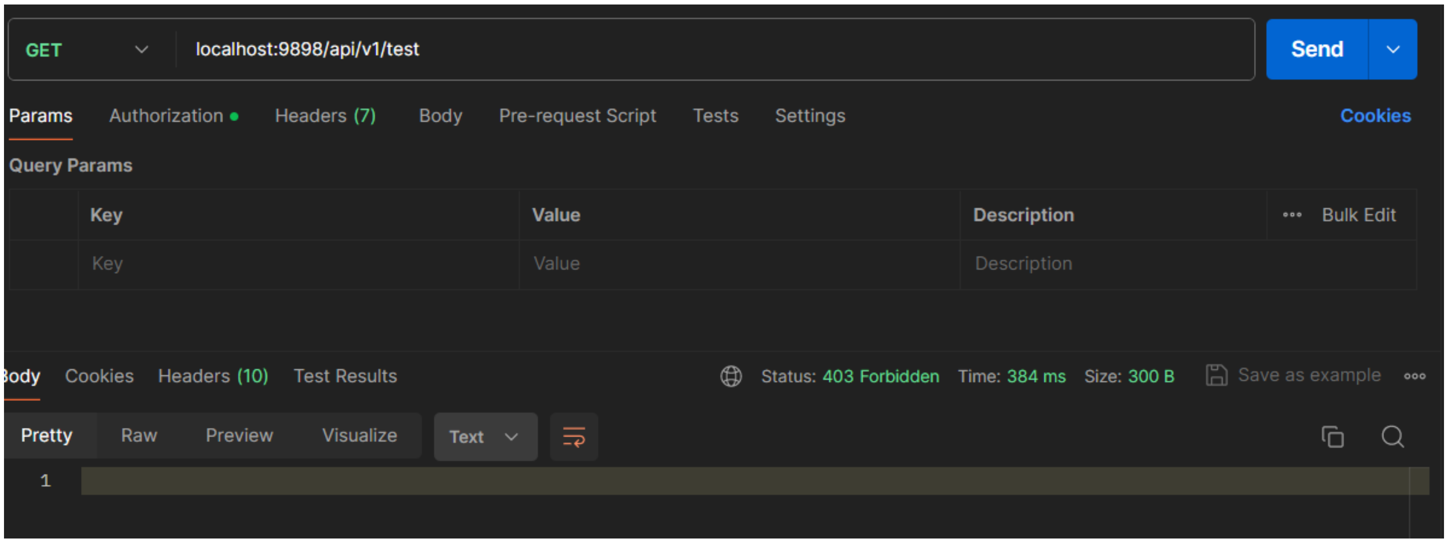


user_info_id	roles_id
1	1

16. Test

Try to access a secured path, example:

```
http://localhost:9898/api/v1/test
```

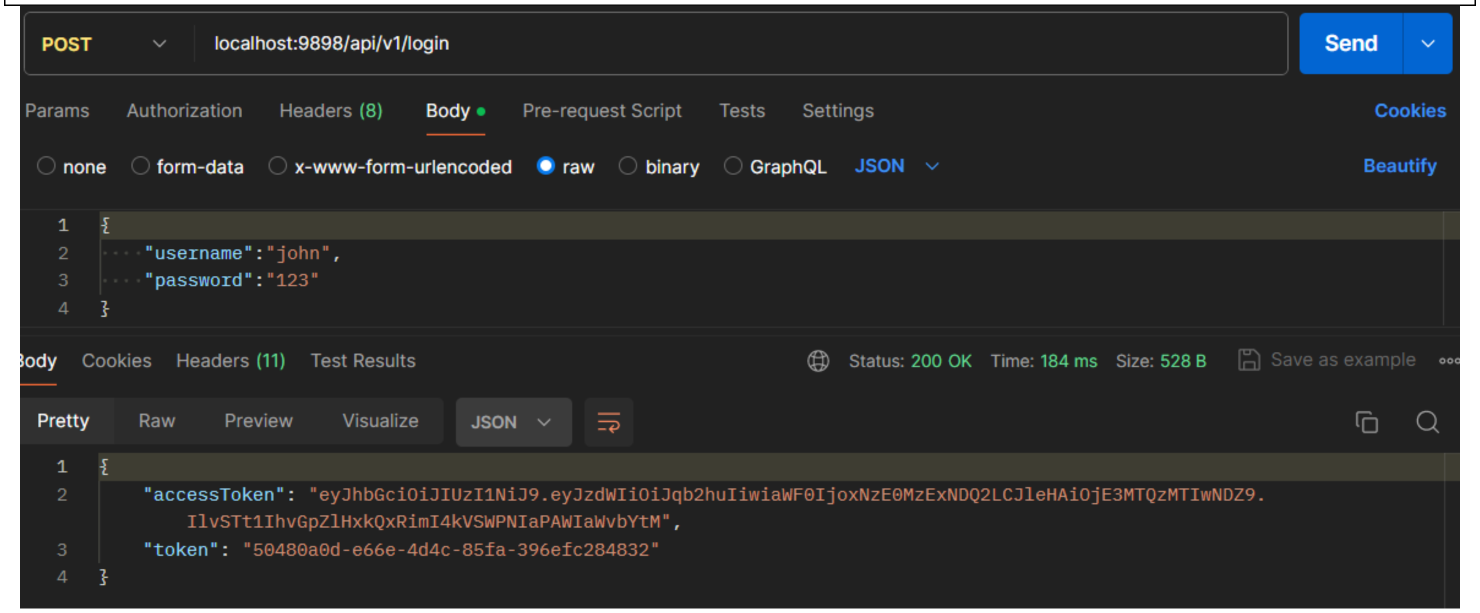


GET localhost:9898/api/v1/test

Status: 403 Forbidden Time: 384 ms Size: 300 B

We can get an access token by logging in with a correct username and password. Send a json post request to:

http://localhost:9898/api/v1/login



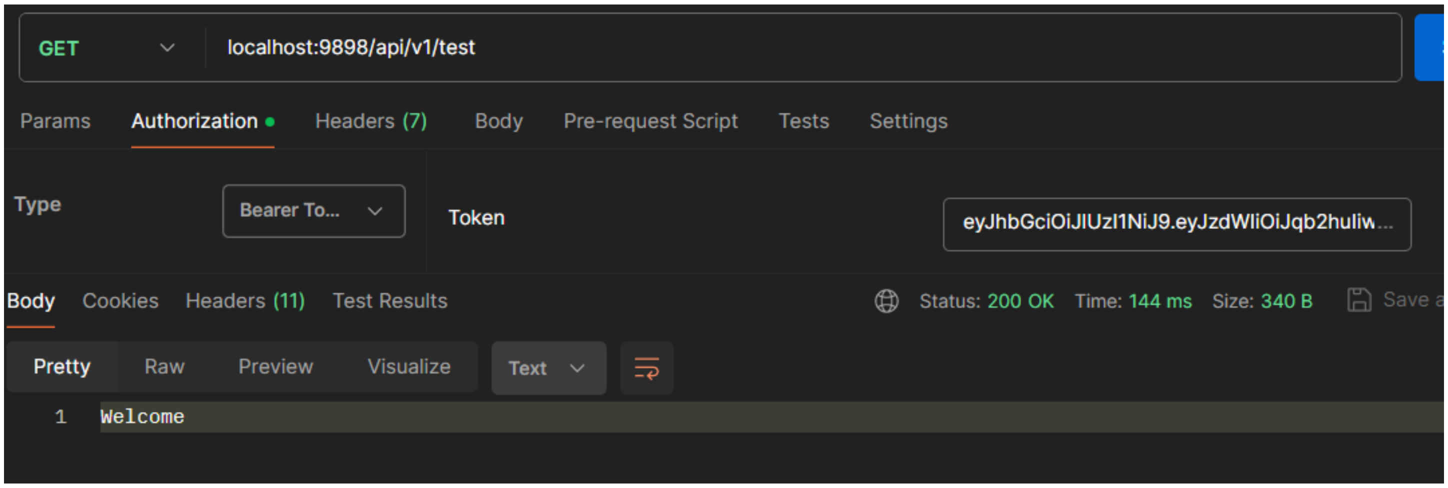
POST localhost:9898/api/v1/login

```
{
    "username":"john",
    "password":"123"
}
```

Status: 200 OK Time: 184 ms Size: 528 B

```
{
    "accessToken": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huIiwiaWF0IjoxNzE0MzExNDQ2LCJleHAiOjE3MTQzMTIwNDZ9.
        IlvSTt1IhvGpZlHxkQxRimI4kVSWPNIaPAWIaWvbYtM",
    "token": "50480a0d-e66e-4d4c-85fa-396efc284832"
}
```

We can use that access token by adding **Authorization (Bearer xxxxxxxxxxxx)** in our header when sending a request to our secured route.



GET localhost:9898/api/v1/test

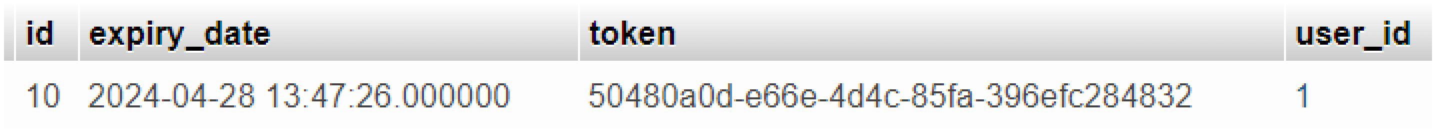
Type: Bearer To...

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huliw...

Status: 200 OK Time: 144 ms Size: 340 B

```
Welcome
```

It will also add an entry in our **refresh_token** table



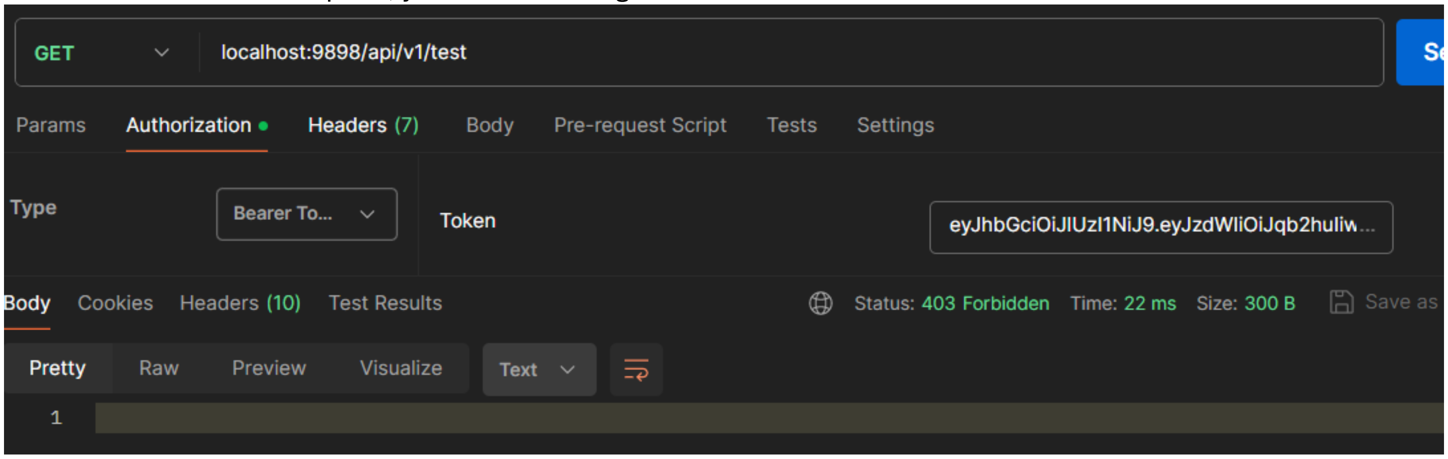
id	expiry_date	token	user_id
10	2024-04-28 13:47:26.000000	50480a0d-e66e-4d4c-85fa-396efc284832	1

The access token has an expiry and you can set it in the JwtService.java file

JwtService.java

```
...
  private String createToken(Map<String, Object> claims, String username) {

      return Jwts.builder()
              .setClaims(claims)
              .setSubject(username)
              .setIssuedAt(new Date(System.currentTimeMillis()))
              .setExpiration(new Date(System.currentTimeMillis()+1000*60*10))
              .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
  }
...
```

Once the access token expires, you would no longer be able to access the secured routes



GET localhost:9898/api/v1/test

Type: Bearer To...

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huliw...

Status: 403 Forbidden Time: 22 ms Size: 300 B

We can also whitelist and blacklist routes in SecurityConfig.java

SecurityConfig.java

```
...
  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.csrf().disable()
          .authorizeHttpRequests()
          .requestMatchers("/api/v1/save", "/api/v1/login", "/api/v1/refreshToken").permitAll()
          .and()
          .authorizeHttpRequests().requestMatchers("/api/v1/**")
          .authenticated()
          .and()
          .sessionManagement()
          .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
          .and()
          .authenticationProvider(authenticationProvider())
          .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class).build();

  }
...
```