OAUTH2 EXERCISE

- ***Implement oauth 2 using spring security and authenticate a user which is saved in database using spring data jpa.***
- ***Grant different Roles to different users and make sure that only authorized users of a given type can access the resource.***

AppUser.java

```
package com.springbootcamp.springsecurity;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import javax.persistence.Entity;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;
public class AppUser implements UserDetails {
    private String username;
    private String password;
    List<GrantedAuthority> grantAuthorities;
    public AppUser(String username, String password, List<GrantedAuthority>
grantAuthorities) {
        this.username = username;
        this.password = password;
        this.grantAuthorities = grantAuthorities;
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities()
    {
        List<GrantedAuthority> listAuthorities = new LinkedList<>();
        listAuthorities.addAll(grantAuthorities);
        for (GrantedAuthority auth: listAuthorities
        ) {
            System.out.println("................."+auth.getAuthority());
        }
        return listAuthorities;
    }
    @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;
    }
}
```

AppUserDetailService.java

```
package com.springbootcamp.springsecurity;
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.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
@Service
public class AppUserDetailsService implements UserDetailsService {
    @Autowired
    PasswordEncoder passwordEncoder;
    @Autowired
    UserDao userDao;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        String encryptedPassword = passwordEncoder.encode("pass");
        System.out.println("Trying to authenticate user ::" + username);
        System.out.println("Encrypted Password ::"+encryptedPassword);
        UserDetails userDetails = userDao.loadUserByUsername(username);
        return userDetails;
    }
}
```

AuthenticatioManagerProvider.java

```
package com.springbootcamp.springsecurity;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.web.configuration.WebSecurity
ConfigurerAdapter;
@Configuration
public class AuthenticationManagerProvider extends
WebSecurityConfigurerAdapter {
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws
Exception {
        return super.authenticationManagerBean();
    }
}
```

AuthorizationServerConfiguration.java

```
package com.springbootcamp.springsecurity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.oauth2.config.annotation.configurers.ClientDeta
ilsServiceConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Auth
orizationServerConfigurerAdapter;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Enab
leAuthorizationServer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Author
izationServerEndpointsConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Author
izationServerSecurityConfigurer;
import
org.springframework.security.oauth2.provider.token.DefaultTokenServices;
import org.springframework.security.oauth2.provider.token.TokenStore;
import
org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;
import
org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConve
rter;
import
org.springframework.security.oauth2.provider.token.store.JwtTokenStore;
@Configuration
@EnableAuthorizationServer
```

```
public class AuthorizationServerConfiguration extends
AuthorizationServerConfigurerAdapter {
    @Autowired
    PasswordEncoder passwordEncoder;
    @Autowired
    AuthenticationManager authenticationManager;
    @Autowired
    UserDetailsService userDetailsService;
    public AuthorizationServerConfiguration() {
        super();
    }
    @Bean
    @Primary
    DefaultTokenServices tokenServices(){
        DefaultTokenServices defaultTokenServices = new
DefaultTokenServices();
        defaultTokenServices.setTokenStore(tokenStore());
        defaultTokenServices.setSupportRefreshToken(true);
        return defaultTokenServices;
    }
    @Override
    public void configure(final AuthorizationServerEndpointsConfigurer
endpoints) {

endpoints.tokenStore(tokenStore()).userDetailsService(userDetailsService)
                .authenticationManager(authenticationManager)
//                .accessTokenConverter(accessTokenConverter())
        ;
    }
//    @Bean
//    JwtAccessTokenConverter accessTokenConverter(){
//        JwtAccessTokenConverter jwtAccessTokenConverter = new
JwtAccessTokenConverter();
//        jwtAccessTokenConverter.setSigningKey("1234");
//        return jwtAccessTokenConverter;
//    }
    @Bean
    public TokenStore tokenStore() {
        return new InMemoryTokenStore();
//        return new JwtTokenStore(accessTokenConverter());
    }
    @Override
    public void configure(final ClientDetailsServiceConfigurer clients)
throws Exception {
        clients.inMemory()
                .withClient("live-test")
                .secret(passwordEncoder.encode("abcde"))
                .authorizedGrantTypes("password","refresh_token")
                .refreshTokenValiditySeconds(30 * 24 * 3600)
                .scopes("app")
                .accessTokenValiditySeconds(7*24*60);
    }
    @Override
    public void configure(AuthorizationServerSecurityConfigurer
authorizationServerSecurityConfigurer) throws Exception {

authorizationServerSecurityConfigurer.allowFormAuthenticationForClients();
    }
}
```

Bootstrap

```
package com.springbootcamp.springsecurity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;
import java.util.LinkedList;
import java.util.List;
@Component
public class Bootstrap implements ApplicationRunner {
    @Autowired
    UserRepository userRepository;
    @Override
    public void run(ApplicationArguments args) throws Exception {
        if(userRepository.count()<1){
            PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
            User user1 = new User();
            user1.setUsername("user");
            user1.setPassword(passwordEncoder.encode("pass"));
            List<Role> roles = new LinkedList<>();
            roles.add(new Role("ROLE_USER"));
            roles.add(new Role("ROLE_ADMIN"));
            user1.setRole(roles);
            User user2 = new User();
            user2.setUsername("admin");
            user2.setPassword(passwordEncoder.encode("pass"));
            List<Role> roles1 = new LinkedList<>();
            roles1.add(new Role("ROLE_ADMIN"));
            //roles1.add(new Role("ROLE_USER"));
            user2.setRole(roles1);
            userRepository.save(user1);
            userRepository.save(user2);
            System.out.println("Total users
saved::"+userRepository.count());
        }
    }
}
```

GrantAuthorityImpl.java

```
package com.springbootcamp.springsecurity;
import org.springframework.security.core.GrantedAuthority;
import java.util.List;
public class GrantAuthorityImpl implements GrantedAuthority
{
        List<Role> authority;
        public GrantAuthorityImpl(List<Role> authority) {
            this.authority = authority;
        }
        @Override
        public String getAuthority() {
            for (Role auth :authority)
            {
                System.out.println("...../////////"+auth.getAuthority());
                return String.valueOf(auth.getAuthority());
            }
            return null;
        }
    }
```

ResourceServerConfiguration.java

```
package com.springbootcamp.springsecurity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authe
nticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGl
obalMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSe
curity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Enab
leResourceServer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Reso
urceServerConfigurerAdapter;
@Configuration
```

```java
@EnableResourceServer
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class ResourceServerConfiguration extends
ResourceServerConfigurerAdapter {
    @Autowired
    AppUserDetailsService userDetailsService;
    public ResourceServerConfiguration() {
        super();
    }
    @Bean
    public static BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public AuthenticationProvider authenticationProvider() {
        final DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService);
        authenticationProvider.setPasswordEncoder(bCryptPasswordEncoder());
        return authenticationProvider;
    }
    @Autowired
    public void configureGlobal(final AuthenticationManagerBuilder
authenticationManagerBuilder) {

authenticationManagerBuilder.authenticationProvider(authenticationProvider()
);
    }
    @Override
    public void configure(final HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/").anonymous()
                .antMatchers("/admin/home").hasAnyRole("ADMIN")
                .antMatchers("/user/home").hasAnyRole("USER")
                .antMatchers("/doLogout").hasAnyRole("ADMIN","USER")
                .anyRequest().authenticated()
                .and()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
                .csrf().disable();
    }
}
```

Role.java

```java
package com.springbootcamp.springsecurity;
import javax.persistence.*;
import java.util.Set;
@Entity
public class Role
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String authority;
    @ManyToMany(mappedBy = "roles",cascade = CascadeType.ALL)
    private Set<User> user;
    public Role() {
    }
    public Role(String authority) {
        this.authority = authority;
    }
    public Set<User> getUser() {
        return user;
    }
    public void setUser(Set<User> user) {
        this.user = user;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getAuthority() {
        return authority;
    }
    public void setAuthority(String authority) {
        this.authority = authority;
    }
}
```

SpringSecurityApplication.java

```
package com.springbootcamp.springsecurity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.oauth2.common.OAuth2AccessToken;
import org.springframework.security.oauth2.provider.token.TokenStore;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import javax.servlet.http.HttpServletRequest;
@RestController
@SpringBootApplication
public class SpringSecurityApplication {
   @Autowired
   private TokenStore tokenStore;
   @GetMapping("/doLogout")
   public String logout(HttpServletRequest request){
      String authHeader = request.getHeader("Authorization");
      if (authHeader != null) {
         String tokenValue = authHeader.replace("Bearer", "").trim();
         OAuth2AccessToken accessToken =
tokenStore.readAccessToken(tokenValue);
         tokenStore.removeAccessToken(accessToken);
      }
      return "Logged out successfully";
   }
   @GetMapping("/")
   public String index(){
      return "index";
   }
   @GetMapping("/admin/home")
   public String adminHome(){
      return "Admin home";
   }
   @GetMapping("/user/home")
   public String userHome(){
      return "User home";
   }
   public static void main(String[] args) {
      SpringApplication.run(SpringSecurityApplication.class, args);
   }
}
```

User.java

```java
package com.springbootcamp.springsecurity;
import javax.persistence.*;
import java.util.List;
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username;
    private String password;
    @ManyToMany(cascade = CascadeType.ALL,fetch = FetchType.EAGER)
    @JoinTable(
            name = "user_role",
            joinColumns = @JoinColumn(
                    name = "user_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(
                    name = "role_id", referencedColumnName = "id"))
    private List<Role> roles;
    public List<Role> getRole() {
        return roles;
    }
    public void setRole(List<Role> roles) {
        this.roles = roles;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", role='" + roles + '\'' +
                '}';
    }
}
```

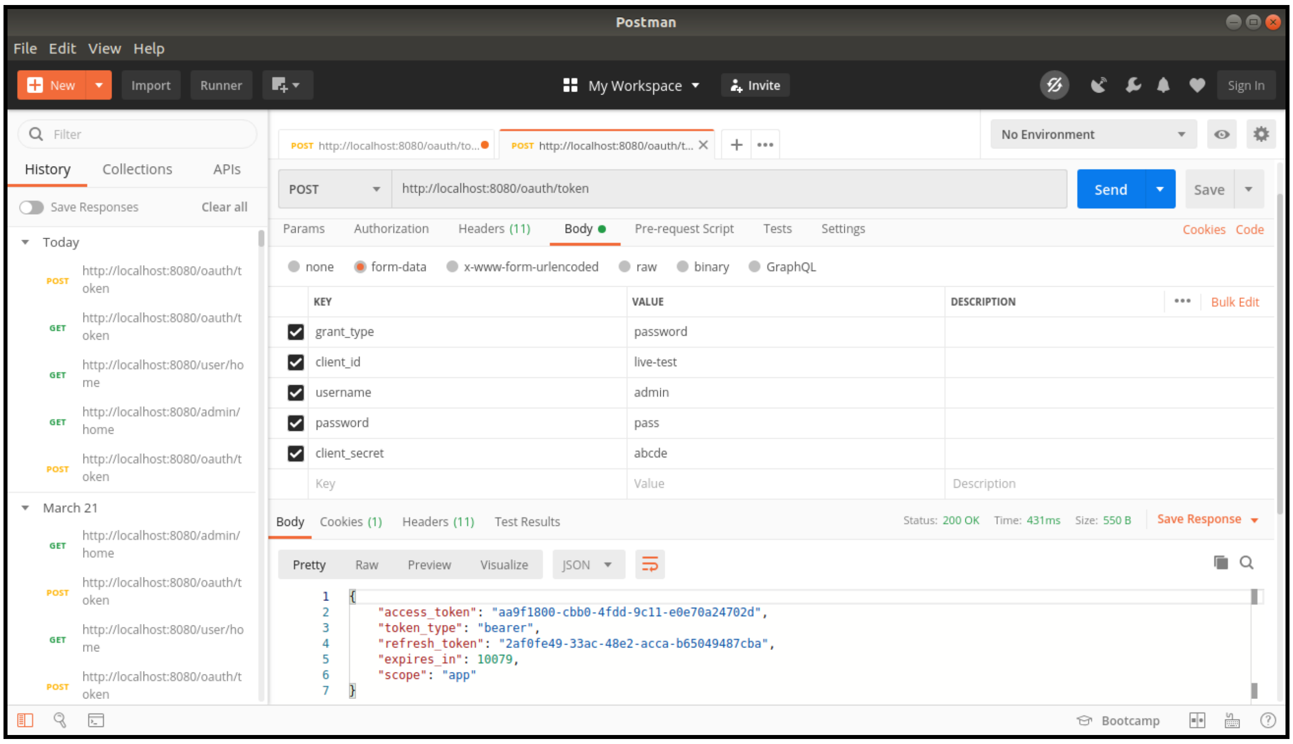
UserDao.java

```
package com.springbootcamp.springsecurity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Repository;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Optional;
@Repository
public class UserDao {
    @Autowired
    UserRepository userRepository;
        AppUser loadUserByUsername(String username) {
            User user = userRepository.findByUsername(username);
            List<GrantedAuthority> grantedAuthorities = new LinkedList<>();
            for (Role role : user.getRole()){
                grantedAuthorities.add(new
SimpleGrantedAuthority(role.getAuthority()));
            }
            System.out.println(user);
            if (username != null) {
                return new AppUser(user.getUsername(), user.getPassword(),
grantedAuthorities);
            } else {
                throw new RuntimeException();
            }
        }
}
```

UserRepository(Inteface)

```
package com.springbootcamp.springsecurity;
import org.springframework.data.repository.CrudRepository;
public interface UserRepository extends CrudRepository<User,Integer> {
    User findByUsername(String username);
}
```

(*When acess token received*)



Today
POST http://localhost:8080/oauth/token
GET http://localhost:8080/oauth/token
GET http://localhost:8080/user/home
GET http://localhost:8080/admin/home
POST http://localhost:8080/oauth/token

March 21
GET http://localhost:8080/admin/home
POST http://localhost:8080/oauth/token
GET http://localhost:8080/user/home
POST http://localhost:8080/oauth/token

POST http://localhost:8080/oauth/token

Params | Authorization | Headers (11) | Body ● | Pre-request Script | Tests | Settings

none | ● form-data | x-www-form-urlencoded | raw | binary | GraphQL

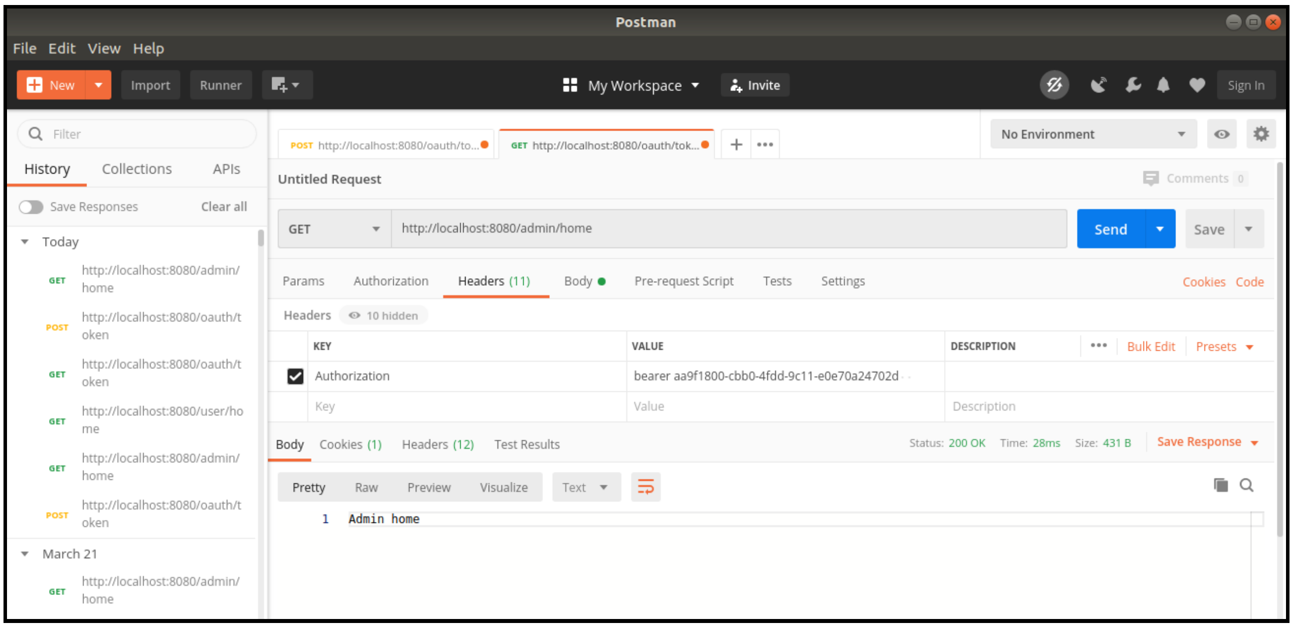
	KEY	VALUE	DESCRIPTION
✓	grant_type	password	
✓	client_id	live-test	
✓	username	admin	
✓	password	pass	
✓	client_secret	abcde	
	Key	Value	Description

Body | Cookies (1) | Headers (11) | Test Results — Status: 200 OK Time: 431ms Size: 550 B

Pretty | Raw | Preview | Visualize | JSON

```
{
    "access_token": "aa9f1800-cbb0-4fdd-9c11-e0e70a24702d",
    "token_type": "bearer",
    "refresh_token": "2af0fe49-33ac-48e2-acca-b65049487cba",
    "expires_in": 10079,
    "scope": "app"
}
```

(*As the details were given for the admin, the admin could access it*)



Today
GET http://localhost:8080/admin/home
POST http://localhost:8080/oauth/token
GET http://localhost:8080/oauth/token
GET http://localhost:8080/user/home
GET http://localhost:8080/admin/home
POST http://localhost:8080/oauth/token

March 21
GET http://localhost:8080/admin/home

Untitled Request

GET http://localhost:8080/admin/home

Params | Authorization | Headers (11) | Body ● | Pre-request Script | Tests | Settings

Headers 10 hidden

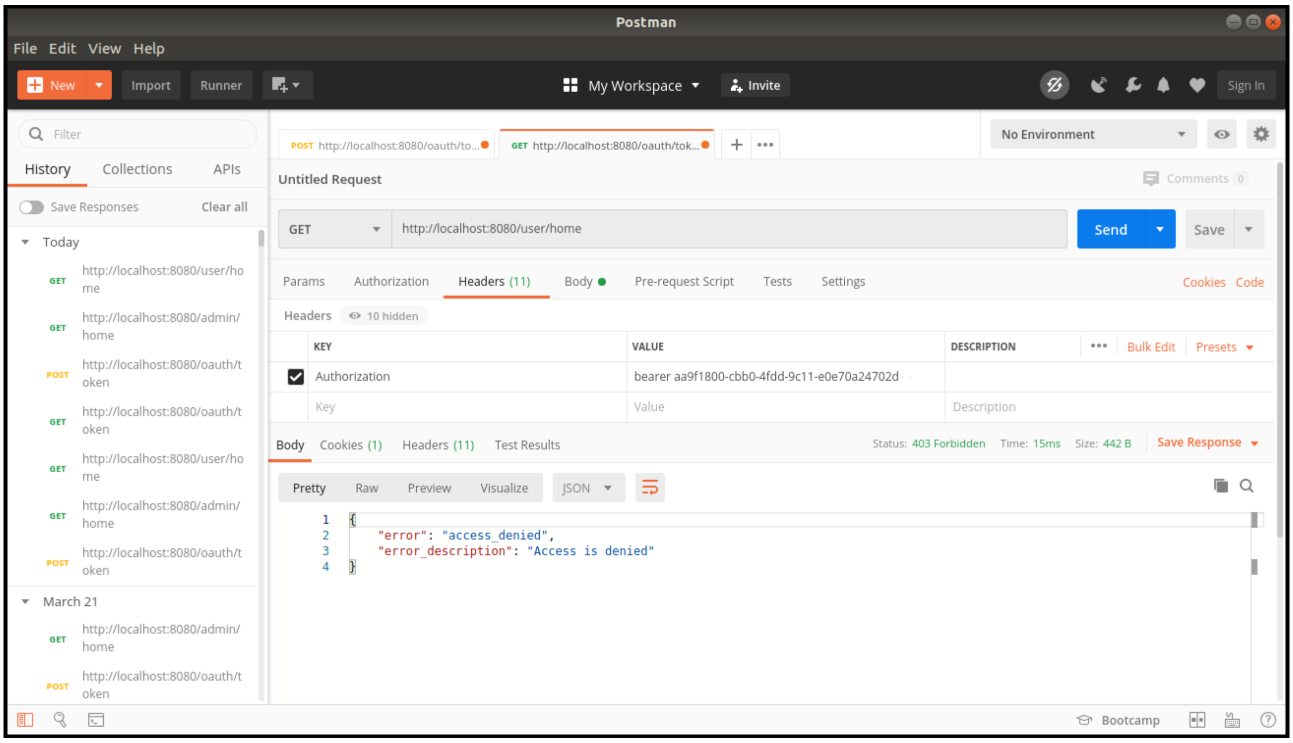
	KEY	VALUE	DESCRIPTION
✓	Authorization	bearer aa9f1800-cbb0-4fdd-9c11-e0e70a24702d	
	Key	Value	Description

Body | Cookies (1) | Headers (12) | Test Results — Status: 200 OK Time: 28ms Size: 431 B

Pretty | Raw | Preview | Visualize | Text

```
Admin home
```

(*As the authority is not given to the user, the user won't be able to access it*)



Untitled Request

GET http://localhost:8080/user/home

Headers (11)

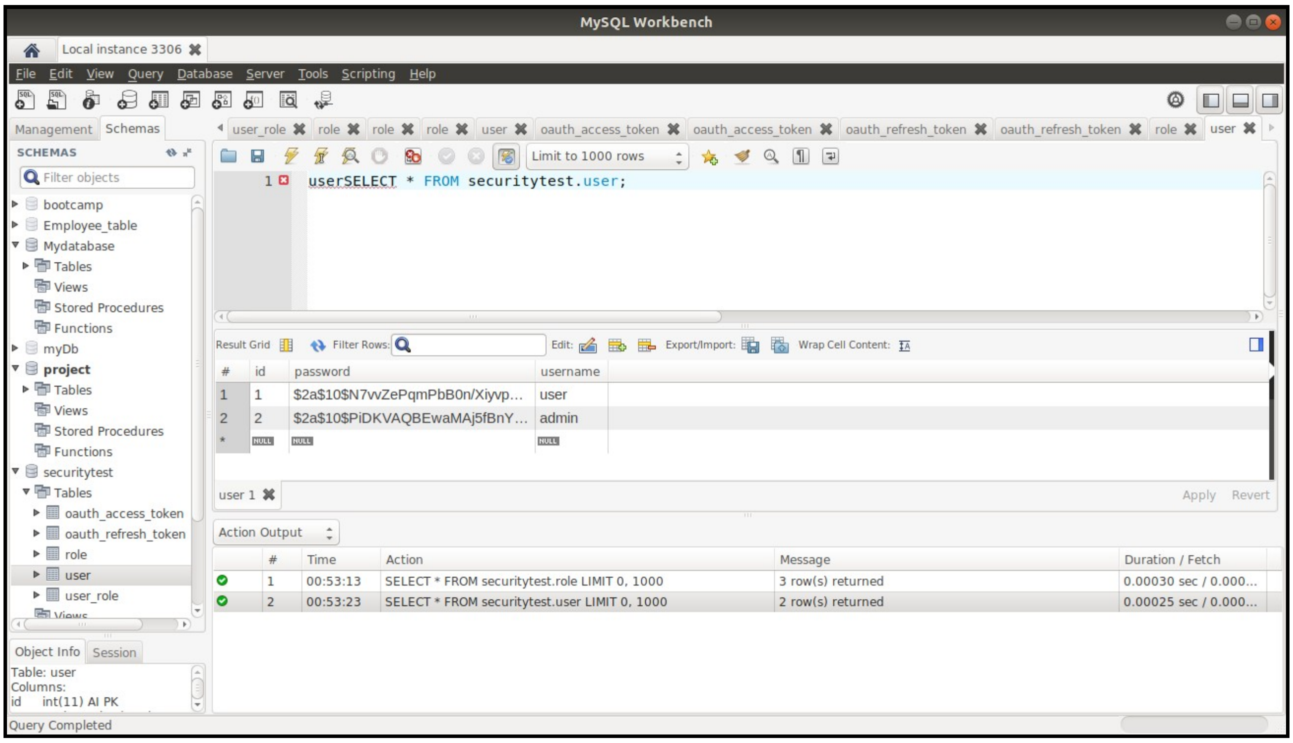
KEY	VALUE	DESCRIPTION
Authorization	bearer aa9f1800-cbb0-4fdd-9c11-e0e70a24702d	

Status: 403 Forbidden Time: 15ms Size: 442 B

```
{
    "error": "access_denied",
    "error_description": "Access is denied"
}
```

DATABASE TABLES

(User table)



```
userSELECT * FROM securitytest.user;
```

#	id	password	username
1	1	$2a$10$N7vvZePqmPbB0n/Xiyvp...	user
2	2	$2a$10$PiDKVAQBEwaMAj5fBnY...	admin
*	NULL	NULL	NULL

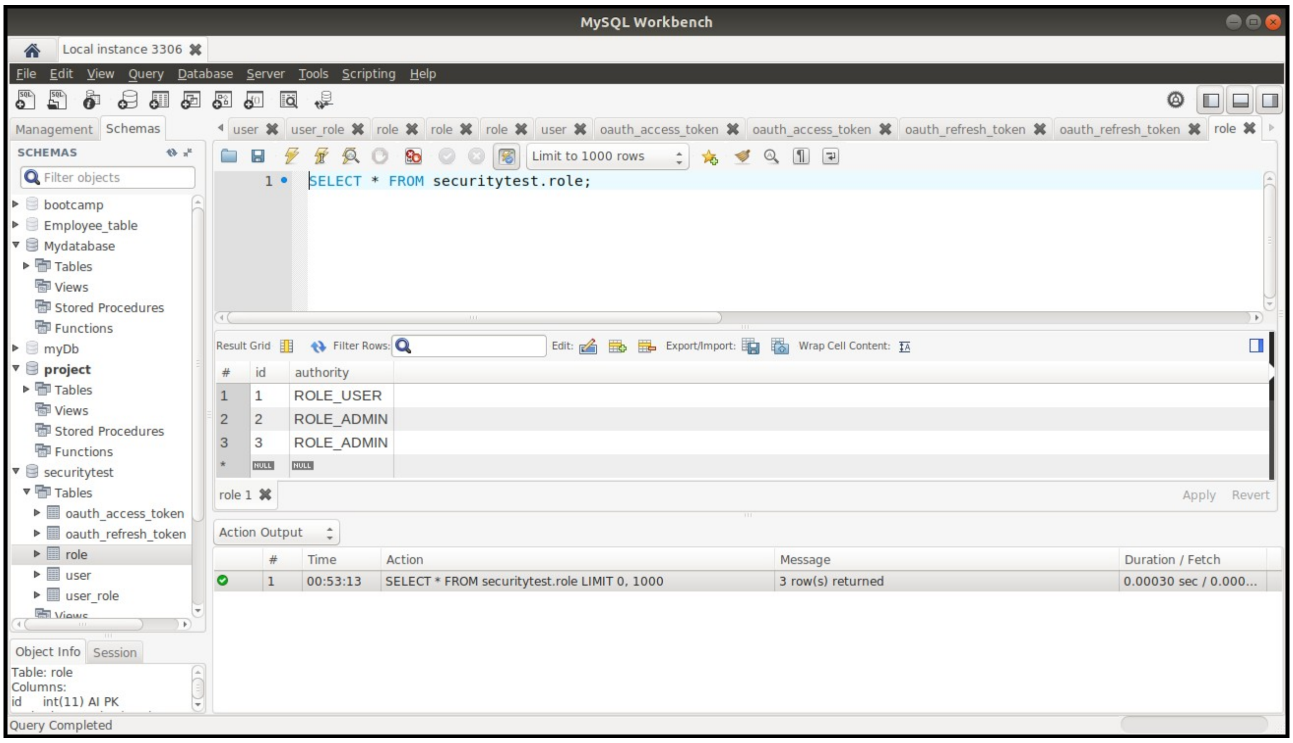
Action Output

	#	Time	Action	Message	Duration / Fetch
✓	1	00:53:13	SELECT * FROM securitytest.role LIMIT 0, 1000	3 row(s) returned	0.00030 sec / 0.000...
✓	2	00:53:23	SELECT * FROM securitytest.user LIMIT 0, 1000	2 row(s) returned	0.00025 sec / 0.000...

Table: user
Columns:
id int(11) AI PK

Query Completed

(Role table)

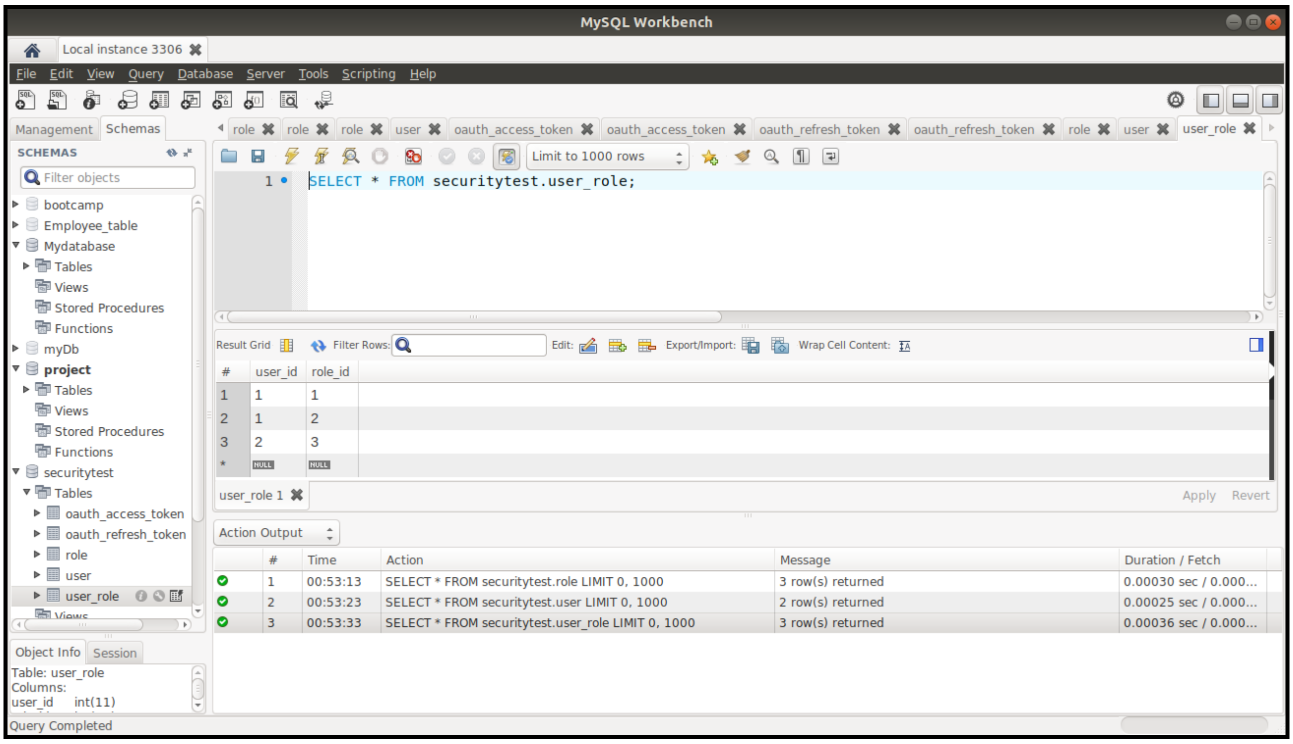


```
SELECT * FROM securitytest.role;
```

#	id	authority
1	1	ROLE_USER
2	2	ROLE_ADMIN
3	3	ROLE_ADMIN
*	NULL	NULL

	#	Time	Action	Message	Duration / Fetch
✓	1	00:53:13	SELECT * FROM securitytest.role LIMIT 0, 1000	3 row(s) returned	0.00030 sec / 0.000...

(As, many-to-many mappping, so another tablw will be formed)



```
SELECT * FROM securitytest.user_role;
```

#	user_id	role_id
1	1	1
2	1	2
3	2	3
*	NULL	NULL

	#	Time	Action	Message	Duration / Fetch
✓	1	00:53:13	SELECT * FROM securitytest.role LIMIT 0, 1000	3 row(s) returned	0.00030 sec / 0.000...
✓	2	00:53:23	SELECT * FROM securitytest.user LIMIT 0, 1000	2 row(s) returned	0.00025 sec / 0.000...
✓	3	00:53:33	SELECT * FROM securitytest.user_role LIMIT 0, 1000	3 row(s) returned	0.00036 sec / 0.000...