Spring Security

What Is Spring Security?

- Provides Jakarta EE (J2EE) application security services
- Designed for enterprise applications and internet-facing applications
- Provides the authentication (who) and authorization (what)

Authentication

- Determination of who
- Technically it is the determination if a principal is who they say they are
- Principals can be humans or machines

Authentication Support

- HTTP basic, digest, x509, and forms-based authentication
- LDAP and Active Directory
- OpenID, Jasig CAS (Central Authentication Service), and JAAS
- Kerberos and SAML

Authorization

- Determines what the principal can or cannot do
- Authorization is based on authentication
- Authorization is often called access control

Authorization Support

- Web request
- Method invocation
- Domain object instance access control

Most Common Projects

- spring-security-core
- spring-security-config
- spring-security-web
- spring-security-test

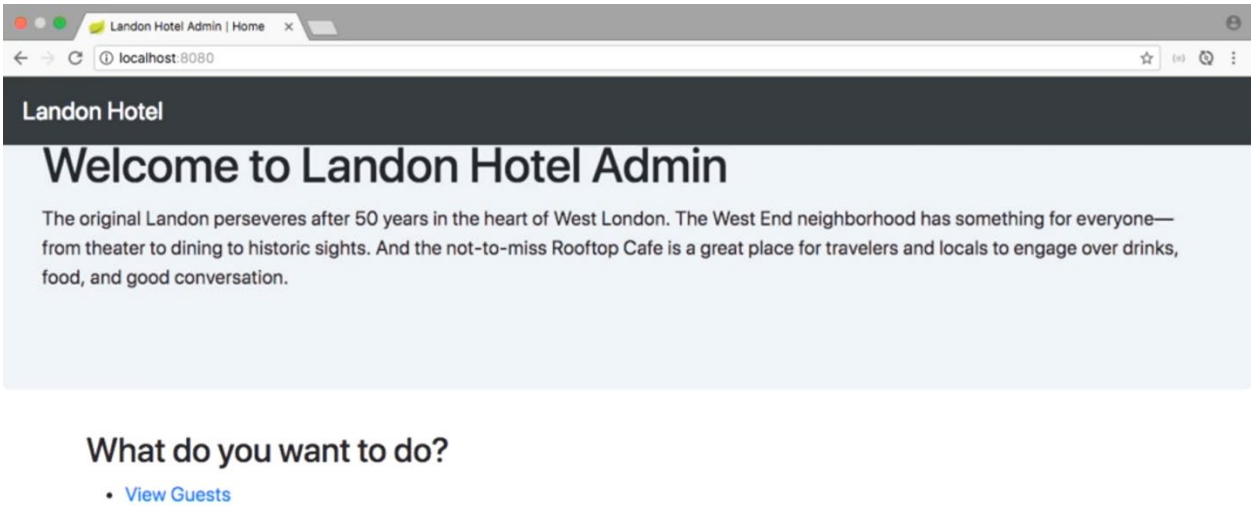
Specific Projects

- spring-security-ldap
- spring-security-oauth2-core
- spring-security-oauth2-client
- spring-security-openid

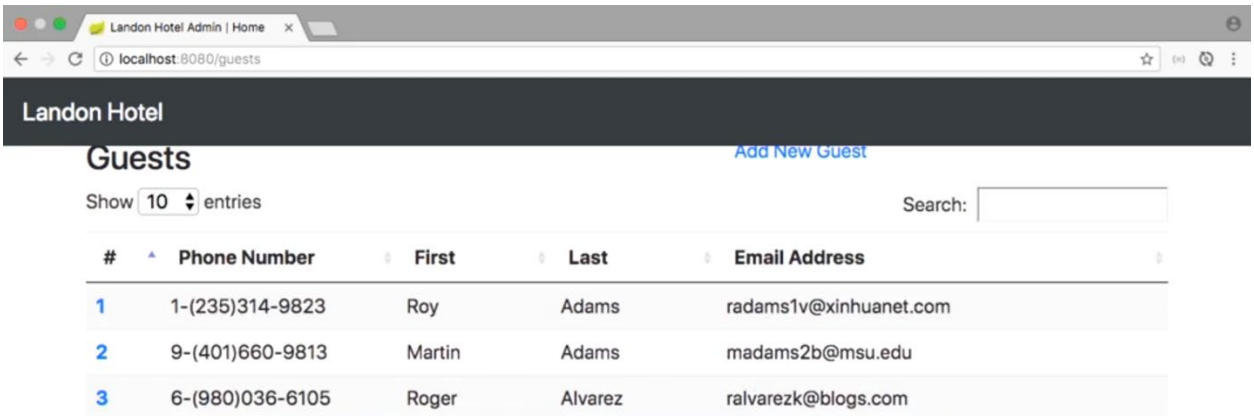
Less Common Projects

- spring-security-oauth2-jose
- spring-security-remoting
- spring-security-cas
- spring-security-acl

Now import the two projects guest-app and guest services. Build and run first service project and then build and run guest-app. Now open localhost:8080.



You can access the view Guests without any authentication

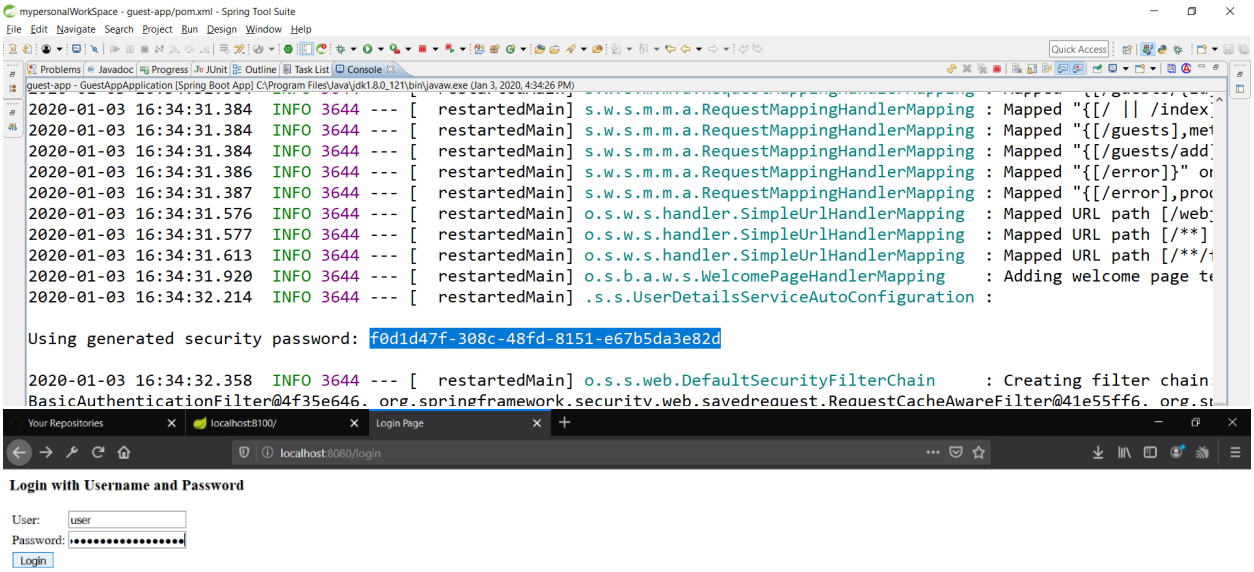


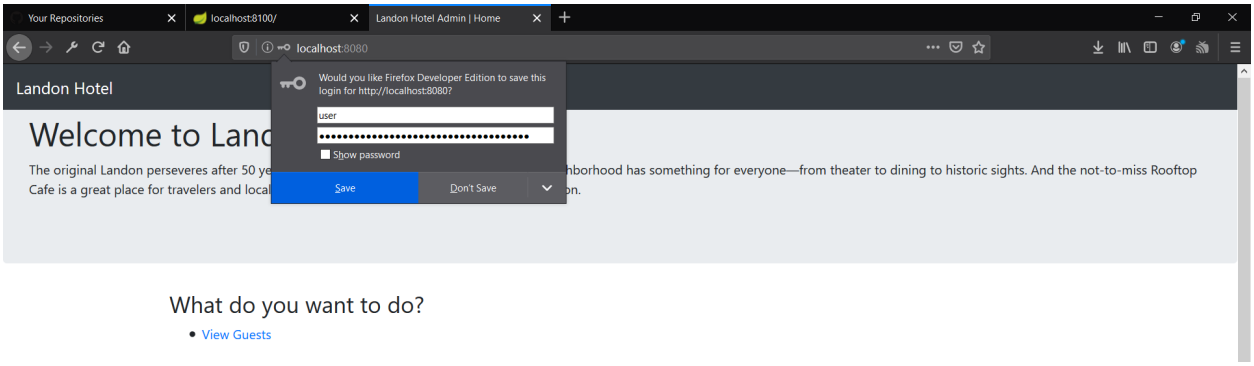
Add Spring security dependency to the web app

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Now by default spring-security adds form-based authentication.

Now check the console output in your ide and look to Using generated security password .This will be the password by default generated by spring and User name is user.





So all this security is Default Security provided by Spring security. Lets start implementing our own Basic HTTP based security.

Lets create a Configuration class to configure Security

ty/app/ApplicationSecurityConfiguration.java - Spring Tool Suite

Help

Dashboard | guest-app/pom.xml | ApplicationSecurityConfiguration.java

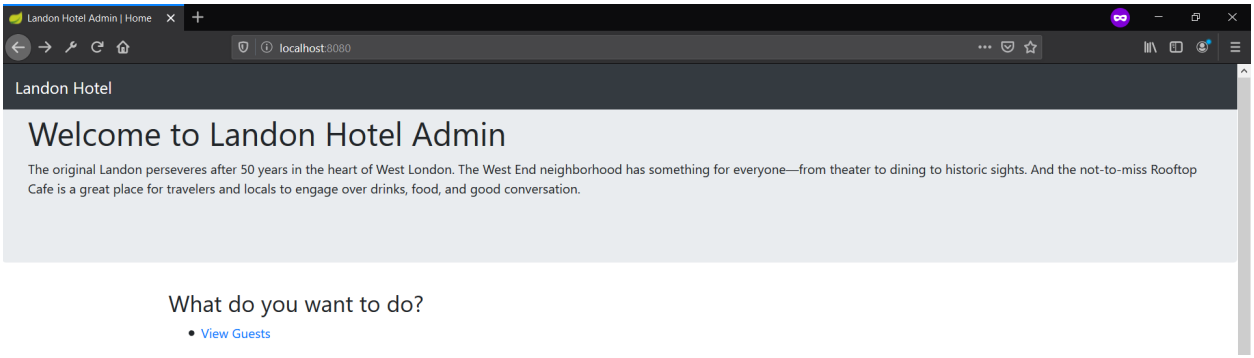
```java
@EnableWebSecurity
public class ApplicationSecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //By using http builder we will set up our authentication
        http
        .csrf().disable()//This will disable CSRF support
        .authorizeRequests()
        //Now we will add the patterns for which no authentication is required
        .antMatchers("/","/index","/css/*","/js/*").permitAll()
        //For any other URL pattern it should needed to be authenticated
        .anyRequest().authenticated()
        .and()
        //We set up authentication type to httpBasic by default it was form based
        .httpBasic();

    }
```

Now we have set security as below

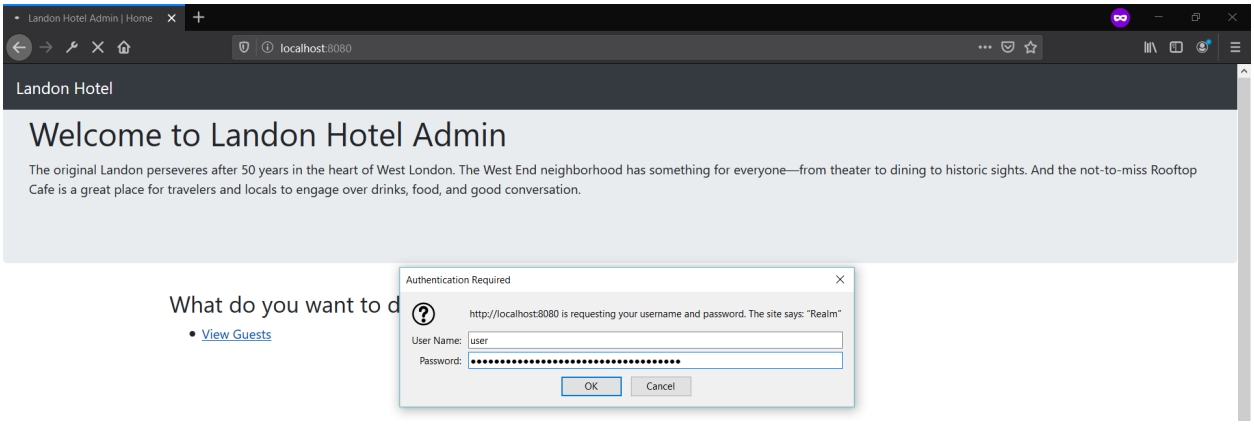
Allow / , /index , css and js without authentication.

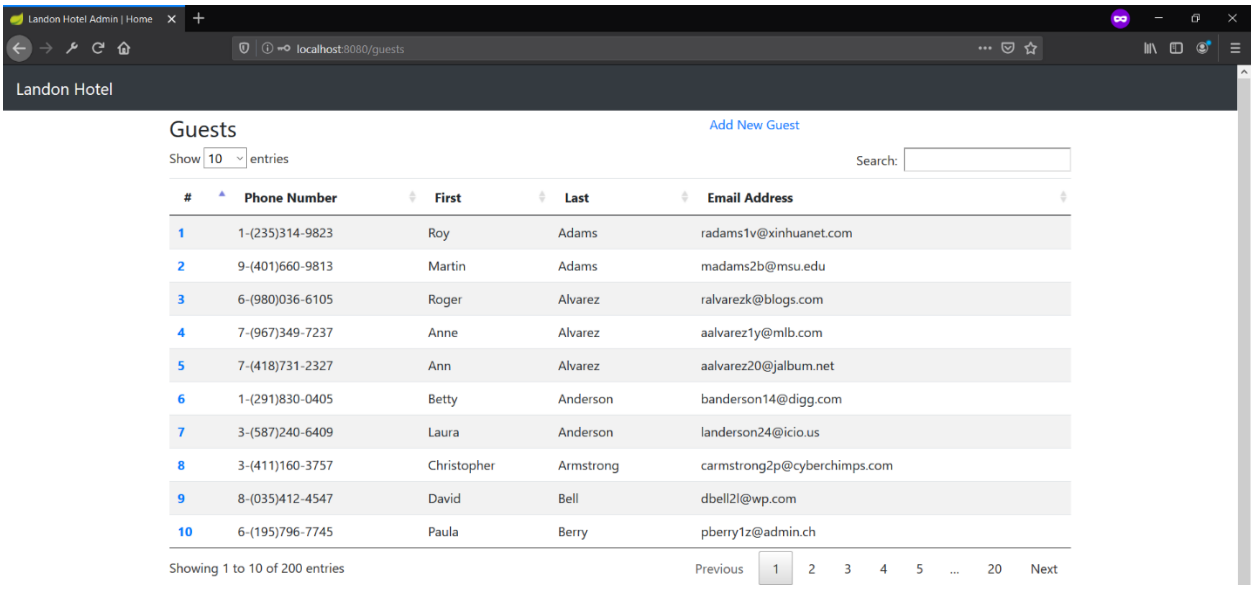
For all other url pattern we need access . Let check it out.

No access needed for http://localhost:8080/

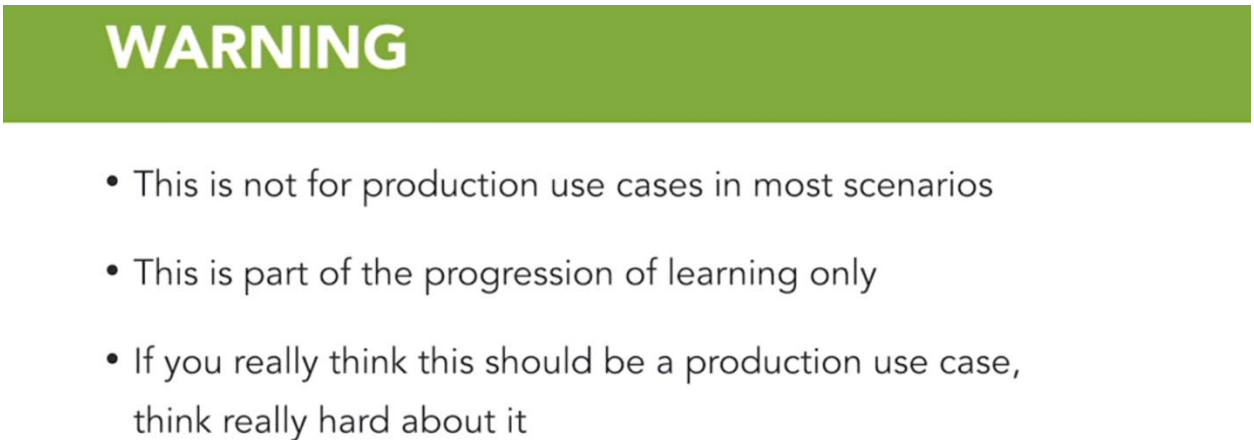


But we need http basic authentication to view the guests





#	Phone Number	First	Last	Email Address
1	1-(235)314-9823	Roy	Adams	radams1v@xinhuanet.com
2	9-(401)660-9813	Martin	Adams	madams2b@msu.edu
3	6-(980)036-6105	Roger	Alvarez	ralvarezk@blogs.com
4	7-(967)349-7237	Anne	Alvarez	aalvarez1y@mlb.com
5	7-(418)731-2327	Ann	Alvarez	aalvarez20@jalbum.net
6	1-(291)830-0405	Betty	Anderson	banderson14@digg.com
7	3-(587)240-6409	Laura	Anderson	landerson24@icio.us
8	3-(411)160-3757	Christopher	Armstrong	carmstrong2p@cyberchimps.com
9	8-(035)412-4547	David	Bell	dbell2l@wp.com
10	6-(195)796-7745	Paula	Berry	pberry1z@admin.ch



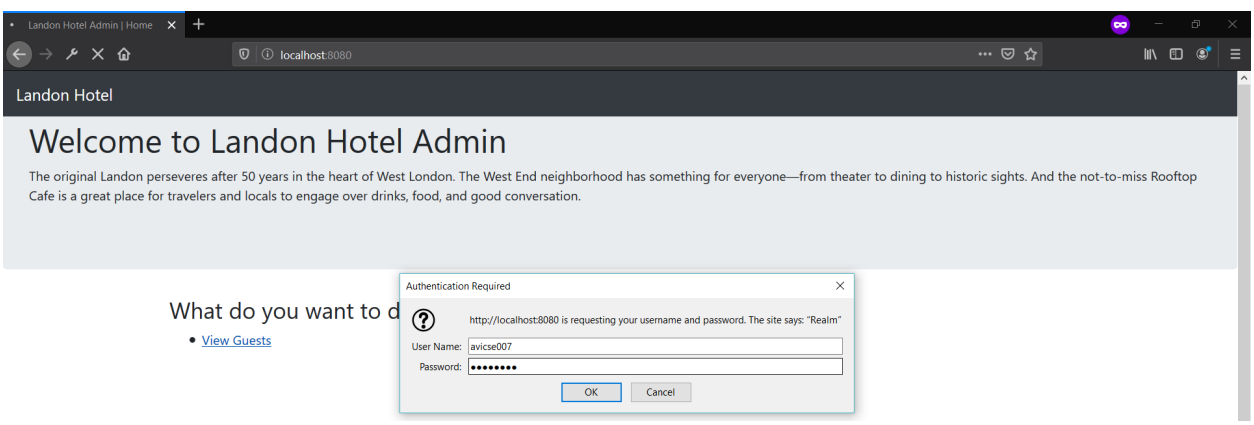
So lets start building our own IN Memory Authentication Engine. This will not generate any default username and password.

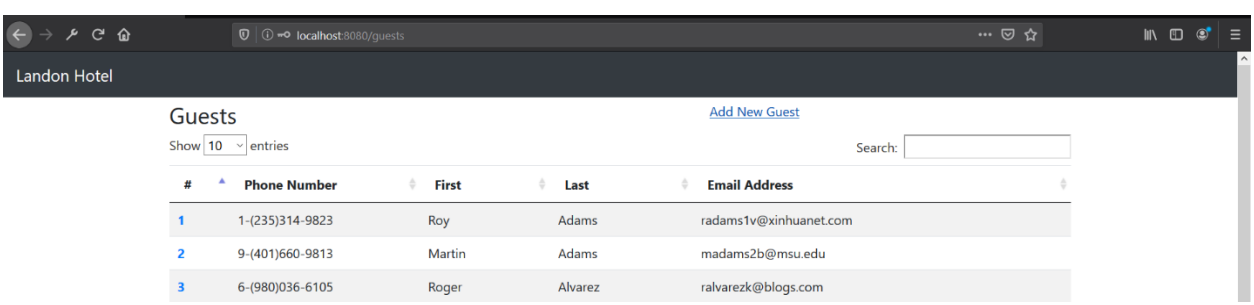
For this lets override another method in ApplicationSecurityConfiguration class

guest-app/pom.xml | ApplicationSecurityConfiguration.java

```java
@Bean
@Override
public UserDetailsService userDetailsService() {
    // TODO Auto-generated method stub
    List<UserDetails> users = new ArrayList<UserDetails>();
    users.add(User.withDefaultPasswordEncoder().
            username("avicse007")
            .password("password")
            .roles("USER","ADMIN")
            .build());
    users.add(User.withDefaultPasswordEncoder().
            username("avicse006")
            .password("password")
            .roles("USER")
            .build());

    return new InMemoryUserDetailsManager(users);
}
```

Let build and run our application





So this is not the industry standards of security. So lets look in industry standards of security with JDBC

JDBC-Based Authentication

Lets add few dependency to support this. One is for JPA and one for in memory database H2 database

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Lets create a User Entity class

y/app/auth/User.java - Spring Tool Suite

Help

Dashboard | guest-app/pom.xml | ApplicationSecurityConfiguration.java | User.java | data.sql | schen

```
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6 import javax.persistence.Table;
7
8 @Entity
9 @Table(name="USER")
10 public class User {
11     @Id
12     @Column(name="USER_ID")
13     private long id;
14
15     @Column(name="USERNAME",nullable=false,unique=true)
16     private String username;
17
18     @Column(name="PASSWORD")
19     private String password;
20
21     public String getUsername() {
```

Now create a Repository for the USER

y/app/auth/UserRepository.java - Spring Tool Suite

lelp

Dashboard | guest-app/pom.xml | ApplicationSecurityConfiguration.java | User.java | data.sql | schema.sql | application.pro

```java
package com.frankmoley.security.app.auth;

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

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

```

Now we have two SQL scripts one for Schema and one for Data

```sql
CREATE TABLE USER (
  USER_ID BIGINT AUTO_INCREMENT PRIMARY KEY,
  USERNAME VARCHAR(128) NOT NULL UNIQUE,
  PASSWORD VARCHAR(256) NOT NULL
);
```

Dashboard | guest-app/pom.xml | ApplicationSecurityConfiguration.java | User.java | data.sql | schema.sql | application.properties

```sql
INSERT INTO USER (USERNAME, PASSWORD) VALUES ('avicse007', 'password');
INSERT INTO USER (USERNAME, PASSWORD) VALUES ('avicse006', 'password');
```

Now set up the auto-ddl in application to false

Dashboard | guest-app/pom.xml | ApplicationSecurityConfiguration.java | User.java | da

```
landon.guest.service.url=http://localhost:8100
spring.jpa.hibernate.ddl-auto=none
```

Now spring security needs a Principal so lets create one

```java
public class PrincipalUser implements UserDetails{

    private User user;

    public PrincipalUser(User user) {
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singleton(new SimpleGrantedAuthority("USER"));
    }

    @Override
    public String getPassword() {
        return this.user.getPassword();
    }

    @Override
    public String getUsername() {
        return this.user.getUsername();
    }

    //For all these methods we shoud get its vlaue from DB. But for now
    // lets return true for all
    @Override
    public boolean isAccountNonExpired() {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        // TODO Auto-generated method stub
        return true;
```

Now we need one more class UserDetailService lets create one now

```java
@Service
public class UserDetailService implements UserDetailsService {

    private final UserRepository userRepository;

    public UserDetailService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundExcep

        User user = this.userRepository.findByUsername(username);
        if(user==null)
            throw new UsernameNotFoundException("User with username "+username+" is not

        return new PrincipalUser(user);
    }

}
```

One last thing we need to do is to configure it in

`ApplicationSecurityConfiguration class`

```java
/*Commenting it for getting user from DB

        @Bean
        @Override
        public UserDetailsService userDetailsService() {
                // TODO Auto-generated method stub
                List<UserDetails> users = new ArrayList<UserDetails>();
                users.add(User.withDefaultPasswordEncoder().
                                username("avicse007")
                                .password("password")
                                .roles("USER","ADMIN")
                                .build());
                users.add(User.withDefaultPasswordEncoder().
                                username("avicse006")
                                .password("password")
                                .roles("USER")
                                .build());

                return new InMemoryUserDetailsManager(users);
        }

        */
```

Autowired the userDetailService that we have created.

```
@Autowired
private UserDetailService userDetailService;
```

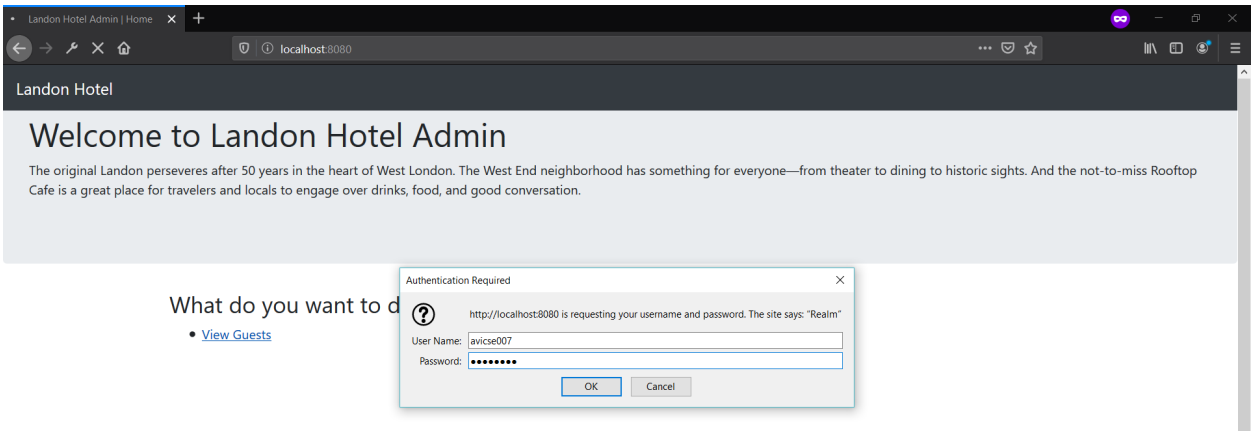
Create a Bean that returns a DAOAuthenticationProvider and set userDetailService to it

```
@Bean
public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailService);
        //Do not use NoOpPasswordEncoder in production code
        provider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());
        return provider;
}
```

Now override the configure method and provide the AuthenticationProvider returned by the bean.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authenticationProvider());
}
```

Now lets run the application



Now lets get rid of the plan text password encoder

Just add Bcrypt encoder in ApplicationSecurityConfiguration class

```
	@Bean
	public DaoAuthenticationProvider authenticationProvider() {
		DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();
		provider.setUserDetailsService(userDetailService);
		//Do not use NoOpPasswordEncoder in production code

	//provider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());
		//Lets use Bcrypt for password encodder
		provider.setPasswordEncoder(new BCryptPasswordEncoder(11));

		return provider;
	}
```

Now update the password in sql file

```
--For Bcrypt password
INSERT INTO USER (USERNAME, PASSWORD) VALUES ('avicse007',
'$2a$11$dp4wMyuqYE3KSwIyQmWZFeUb7jCsHAdk7ZhFc0qGw6i5J124imQBi');
INSERT INTO USER (USERNAME, PASSWORD) VALUES ('avicse006',
'$2a$11$dp4wMyuqYE3KSwIyQmWZFeUb7jCsHAdk7ZhFc0qGw6i5J124imQBi');
```

That all Build and run the application.

Authorizations

Now to provide authorizations first we need to update our database. So below are the two files

1. **Data.sql**

```
INSERT INTO USER (USERNAME, PASSWORD) VALUES ('avicse007',
'$2a$11$dp4wMyuqYE3KSwIyQmWZFeUb7jCsHAdk7ZhFc0qGw6i5J124imQBi');
INSERT INTO USER (USERNAME, PASSWORD) VALUES ('avicse006',
'$2a$11$dp4wMyuqYE3KSwIyQmWZFeUb7jCsHAdk7ZhFc0qGw6i5J124imQBi');
INSERT INTO AUTH_USER_GROUP (USERNAME, AUTH_GROUP) VALUES('avicse007',
'USER');
```

```
INSERT INTO AUTH_USER_GROUP (USERNAME, AUTH_GROUP) VALUES('avicse007',
'ADMIN');
INSERT INTO AUTH_USER_GROUP (USERNAME, AUTH_GROUP) VALUES('avicse006',
'USER');
```

2. **Schema.sql**

```
CREATE TABLE USER (
  USER_ID BIGINT AUTO_INCREMENT PRIMARY KEY,
  USERNAME VARCHAR(128) NOT NULL UNIQUE,
  PASSWORD VARCHAR(256) NOT NULL
);

CREATE TABLE AUTH_USER_GROUP (
  AUTH_USER_GROUP_ID BIGINT AUTO_INCREMENT PRIMARY KEY,
  USERNAME VARCHAR(128) NOT NULL,
  AUTH_GROUP VARCHAR(128) NOT NULL,
  CONSTRAINT USER_AUTH_USER_GROUP_FK FOREIGN KEY(USERNAME) REFERENCES
USER(USERNAME),
  UNIQUE (USERNAME, AUTH_GROUP)
)
```

Now lets create Entity and Repository for AUTH_USER_GROUP

Entity

```
@Entity
@Table(name="AUTH_USER_GROUP")
public class AuthGroup {

	@Id
	@Column(name="AUTH_USER_GROUP_ID")
	private Long id;

	@Column(name="USERNAME", nullable=false)
	private String username;

	@Column(name="AUTH_GROUP", nullable=false)
	private String authGroup;

	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 getAuthGroup() {
		return authGroup;
	}

	public void setAuthGroup(String authGroup) {
		this.authGroup = authGroup;
	}
```

Repository

```
@Repository
public interface AuthGroupRepository extends JpaRepository<AuthGroup,
Long> {
	List<AuthGroup> findByUsername(String username);

}
```

Now in PrincicalUser Class add AuthGroup

```
private List<AuthGroup> authGroups;

	public PrincipalUser(User user, List<AuthGroup> authGroups) {
		this.user = user;
		this.authGroups = authGroups;
	}
```

Now update the GrantAuthority method

```
	@Override
	public Collection<? extends GrantedAuthority> getAuthorities() {
		//return Collections.singleton(new
SimpleGrantedAuthority("USER"));
		if(authGroups==null)
			return Collections.emptySet();
		Set<SimpleGrantedAuthority> grantedAuthorities = new
HashSet<>();
		authGroups.forEach(group->{
			grantedAuthorities.add(new
SimpleGrantedAuthority(group.getAuthGroup()));
		});

		return grantedAuthorities;
	}
```

Then we need to make change in UserDetailService class

```
@Service
public class UserDetailService implements UserDetailsService {

	private final UserRepository userRepository;
	private final AuthGroupRepository authGroupRepository;

	public UserDetailService(UserRepository
userRepository,AuthGroupRepository authGroupRepository) {
		this.userRepository = userRepository;
		this.authGroupRepository = authGroupRepository;
	}

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

		User user = this.userRepository.findByUsername(username);
		if(user==null)
			throw new UsernameNotFoundException("User with
username "+username+" is not found");
		List<AuthGroup> authGroups =
authGroupRepository.findByUsername(username);
		return new PrincipalUser(user,authGroups);
```

```
    }

}
```

And finally, we need to make changes in our GuestController to add roles on each of the endpoints by using @PreAuthorize annotations

```
@Controller
@RequestMapping("/")
public class GuestController {

    private final GuestService guestService;

    public GuestController(GuestService guestService){
        super();
        this.guestService = guestService;
    }

    @GetMapping(value={"/", "/index"})
    public String getHomePage(Model model){

        return "index";
    }

    @GetMapping(value="/guests")
    @PreAuthorize("hasRole('ROLE_USER')")
    public String getGuests(Model model){
        List<Guest> guests = this.guestService.getAllGuests();
        model.addAttribute("guests", guests);
        return "guests-view";
    }

    @GetMapping(value="/guests/add")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String getAddGuestForm(Model model){
        return "guest-view";
    }

    @PostMapping(value="/guests")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public ModelAndView addGuest(HttpServletRequest request, Model
model, @ModelAttribute GuestModel guestModel){
        Guest guest = this.guestService.addGuest(guestModel);
        model.addAttribute("guest", guest);
```

```
        request.setAttribute(View.RESPONSE_STATUS_ATTRIBUTE,
HttpStatus.TEMPORARY_REDIRECT);
        return new ModelAndView("redirect:/guests/" + guest.getId());
    }

    @GetMapping(value="/guests/{id}")
    @PreAuthorize("hasRole('ROLE_USER')")
    public String getGuest(Model model, @PathVariable long id){
        Guest guest = this.guestService.getGuest(id);
        model.addAttribute("guest", guest);
        return "guest-view";
    }

    @PostMapping(value="/guests/{id}")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String updateGuest(Model model, @PathVariable long id,
@ModelAttribute GuestModel guestModel){
        Guest guest = this.guestService.updateGuest(id, guestModel);
        model.addAttribute("guest", guest);
        model.addAttribute("guestModel", new GuestModel());
        return "guest-view";
    }
}
```

Now lets just configure our ApplicationSecurityCongifuration. Fist we add @EnableGlobalMethodSecutrity this will turn our method level security.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class ApplicationSecurityConfiguration extends
WebSecurityConfigurerAdapter {
```

Now we need to map the roles that we have in our DB. So lets create a Bean

```
@Bean
     GrantedAuthoritiesMapper authoritiesMapper() {
          SimpleAuthorityMapper authorityMapper = new
SimpleAuthorityMapper();
          authorityMapper.setConvertToUpperCase(true);
          authorityMapper.setDefaultAuthority("USER");
```

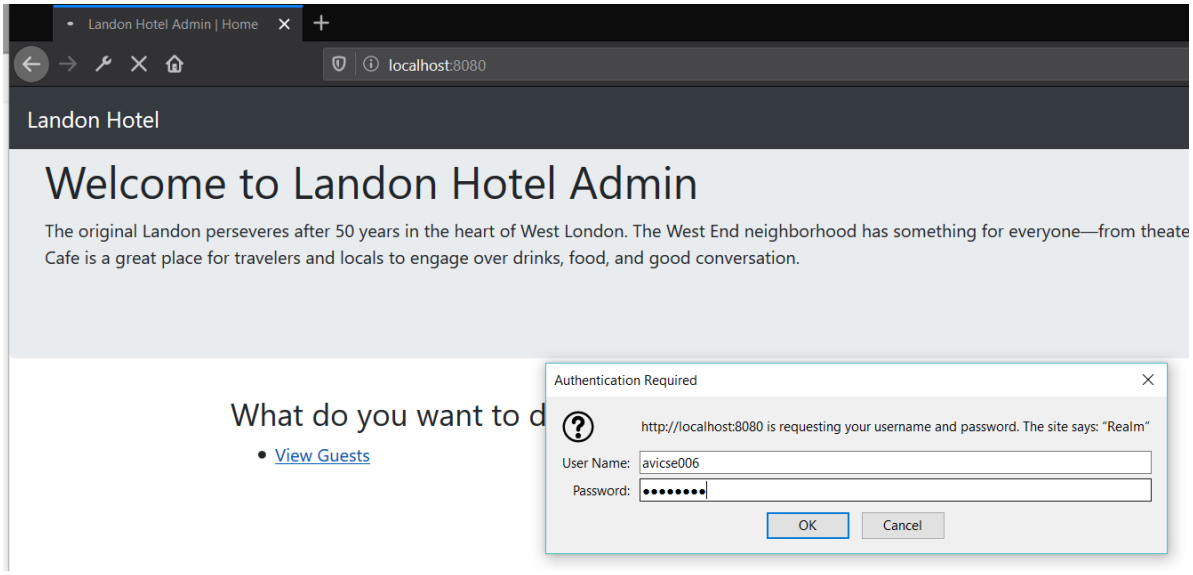
```
            return authorityMapper;
      }
```

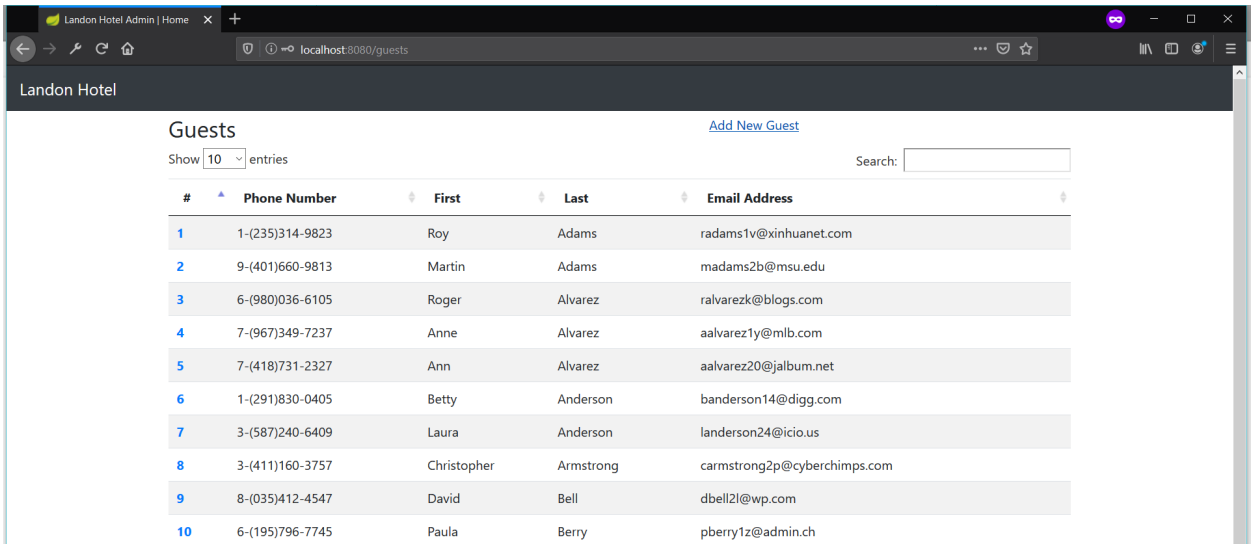
Now set this AuthorityMapper returned by this bean in authenticationProvider bean.

```
@Bean
      public DaoAuthenticationProvider authenticationProvider() {
            DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();
            provider.setUserDetailsService(userDetailService);
            //Do not use NoOpPasswordEncoder in production code

      //provider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());
            //Lets use Bcrypt for password encodder
            provider.setPasswordEncoder(new BCryptPasswordEncoder(11));
            provider.setAuthoritiesMapper(authoritiesMapper());
            return provider;
      }
```

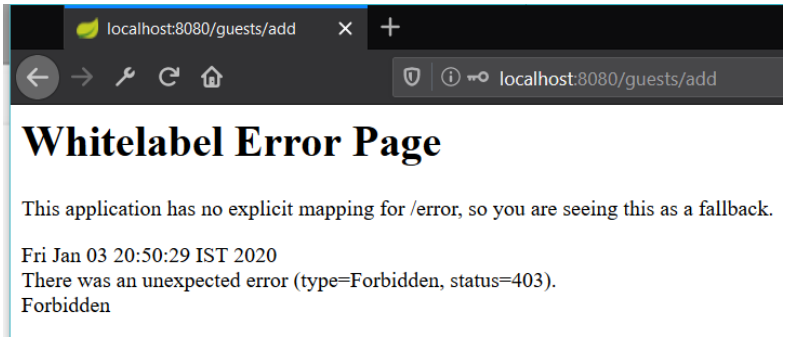
Now lets test

Login with USER role that is with user avicse006

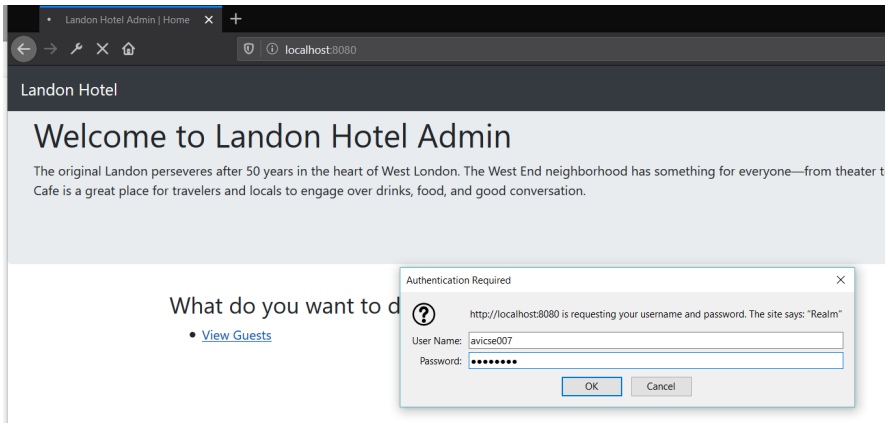


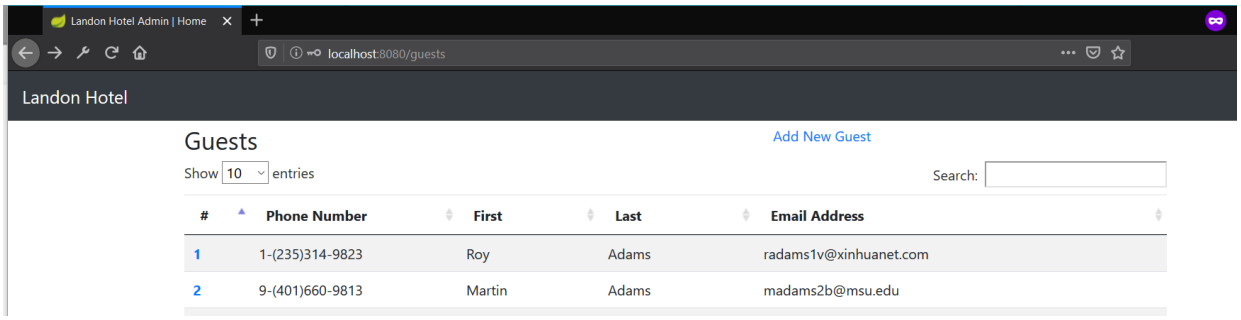


Now click on AddNew Guest

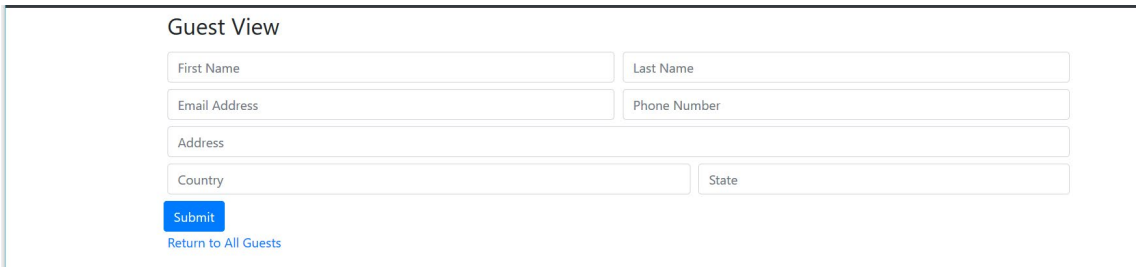


Do same with ADMIN ROLE user that is avicse007

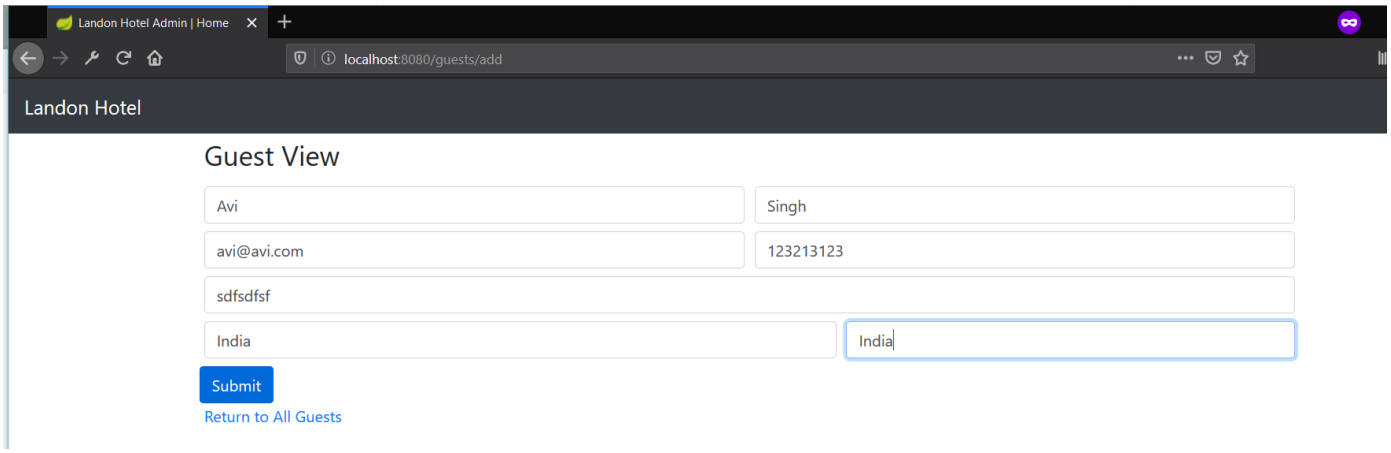




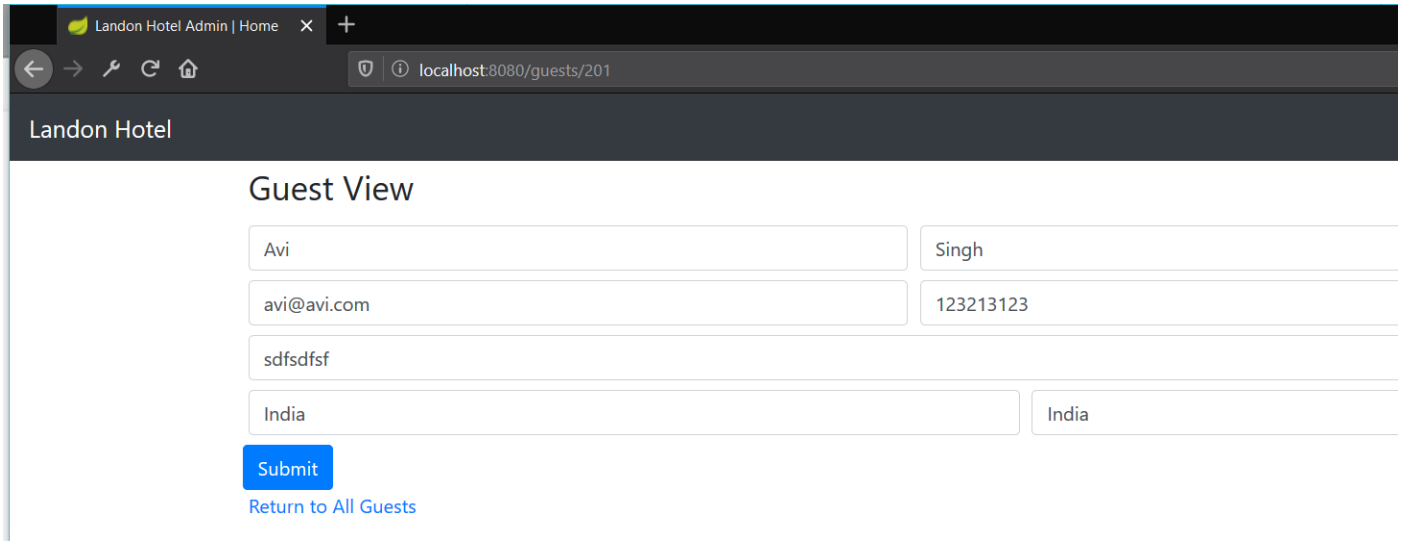
Click on Add New User



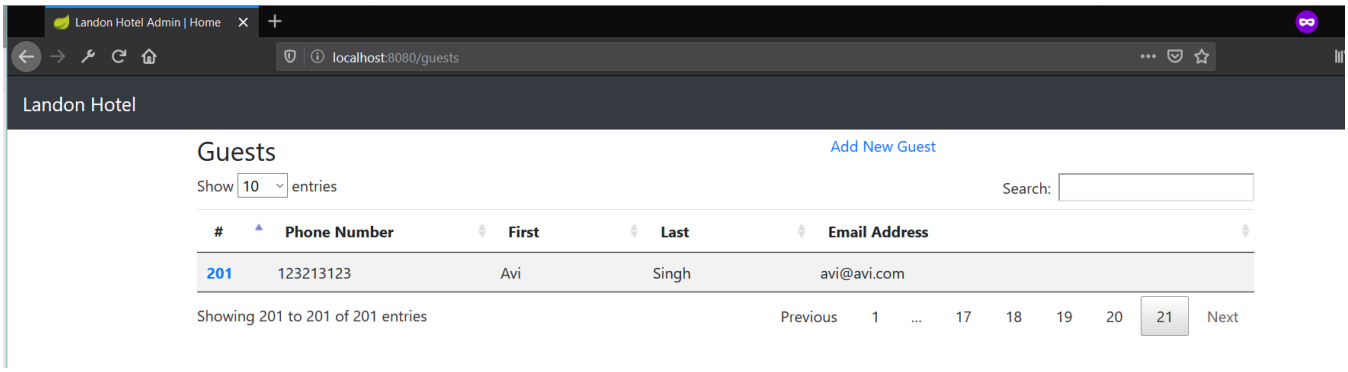
Fill form and hit submit



After submit note the url



Click on Return to all guests



Basic Authentication

- Basic is in the spec; forms isn't
- Cannot support logging off
- No real security implications with either, assuming TLS is used

Forms-Based Authentication

- Allows you to customize the form
- Allows for a more seamless view in your application
- Provides "remember me" options
- Provides logout

Steps

- Implement login form
- Implement logout page
- Turn it on

Implementing the Login Form

1. **Add Maven dependency for thymeleaf**

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity4</artifactId>
</dependency>
```

2. **Set up the html pages**

```
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
            <a class="navbar-brand" th:href="@{/index}">Landon Hotel</a>
            <div class="navbar-nav mr-auto">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link" th:href="@{/guests}">Guests</a>
                    </li>
                </ul>
            </div>
            <div class="navbar-nav">
                <ul class="navbar-nav">
                    <li class="nav-item" sec:authorize="isAnonymous()">
                            <a class="nav-link" th:href="@{/login}">Login</a>
                    </li>
                    <li class="nav-item" sec:authorize="isAuthenticated()">
                        <a class="nav-link" th:href="@{/logout}">Logout</a>
                    </li>
                </ul>
            </div>
        </nav>
    </header>
```

and we'll get a link to go to the logout page

3. **Create login and logout html pages**
4. **Make changes in Controller**

   Add login get mapping

```
@GetMapping(value="/login")
public String getLogin(Model model) {
  return "login";
}
```

Add logout mapping

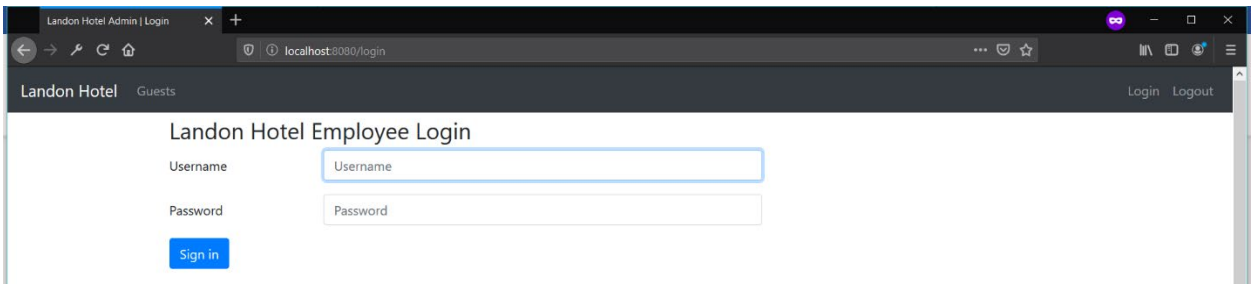
```
@GetMapping(value="/logout-success")
public String getLoginOut(Model model) {
  return "logout";
}
```

5. **Now made configuration changes in ApplicationSecurityConfiguration**

```
@Override
protected void configure(HttpSecurity http) throws Exception {
```

```
            //By using http builder we will set up our authentication
            http
            .csrf().disable()//This will disable CSRF support
            .authorizeRequests()
            //Now we will add the patterns for which no authentication
is required
            .antMatchers("/","/index","/css/*","/js/*").permitAll()
            //For any other URL pattern it should needed to be
authenticated
            .anyRequest().authenticated()
            .and()
            //We set up authentication type to httpBasic by default it
was basic http based
            //.httpBasic();
            //We set up authentication type to Form based
            .formLogin()
            .loginPage("/login").permitAll()
            .and()
            .logout().invalidateHttpSession(true)
            .clearAuthentication(true)
            .logoutRequestMatcher((new
AntPathRequestMatcher("/logout")))
            .logoutSuccessUrl("/logout-success").permitAll();
      }
```

6. **Now let's test the application**



LDAP Security

Why LDAP

- LDAP is lightweight, especially for user authentication—Lightweight Directory Access Protocol
- Built into many operating systems
- Interoperability
- Scalability

Spring Security LDAP

- spring-security-ldap project
- Full support for native LDAP operations
- Password-hashing algorithms included

Paradigm

- Very similar to basic and forms-based authentication
- Leverages AuthenticationManagerBuilder in the same manner

Internal LDAP

- Use embedded LDAP
- Can use OpenLDAP if you prefer
- AD is not LDAP, but can use LDAP for authentication

Configuring an Embedded LDAP Server

1. **Add Maven dependency**

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-ldap</artifactId>
</dependency>
<dependency>
    <groupId>com.unboundid</groupId>
    <artifactId>unboundid-ldapsdk</artifactId>
</dependency>
```

2. **Add Configuration to application.properties**

```
landon.guest.service.url=http://localhost:8100
spring.jpa.hibernate.ddl-auto=none
#logging.level.root=DEBUG
#Setting for lap directory
spring.ldap.embedded.ldif=classpath:landon.dif
spring.ldap.embedded.base-dn=dc=frankmoley,dc=com
spring.ldap.embedded.port=8389
```

3. **Add the ldif file in the resources folder**

Landon.ldif

```
dn: dc=frankmoley,dc=com
objectclass: top
objectclass: domain
objectclass: extensibleObject
dc: frankmoley
o: FrankMoley

dn: ou=people,dc=frankmoley,dc=com
objectclass: top
objectClass: organizationalUnit
ou: people
```

```
dn: ou=groups,dc=frankmoley,dc=com
objectClass: top
objectClass: organizationalUnit
ou: groups

dn: uid=fpmoles,ou=people,dc=frankmoley,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Frank Moley
sn: Moley
givenName: Frank
mail: fpmoles@frankmoley.com
uid: fpmoles
userPassword: {SHA}W6ph5Mm5Pz8GgiULbPgzG37mj9g=

dn: uid=jdoe,ou=people,dc=frankmoley,dc=com
objectClass: inetOrgPerson
cn: John Doe
sn: Doe
givenName: John
mail: jdoe@frankmoley.com
uid: jdoe
userPassword: {SHA}iEPX+SQWIR3p67lj/0zigSWTKHg=

dn: cn=admin,ou=groups,dc=frankmoley,dc=com
objectclass: top
objectclass: groupOfUniqueNames
cn: admins
uniqueMember: uid=fpmoles,ou=people,dc=frankmoley,dc=com

dn: cn=user,ou=groups,dc=frankmoley,dc=com
objectclass: top
objectclass: groupOfUniqueNames
cn: users
uniqueMember: uid=fpmoles,ou=people,dc=frankmoley,dc=com
uniqueMember: uid=jdoe,ou=people,dc=frankmoley,dc=com
```

Implementing LDAP Authentication

1. **Create another copy of guest-app**
2. **Delete the auth package**
3. **In pom.xml remove jpa and h2 dependency**
4. **Delete data.sql and schema.sql from resources folder**
5. **In ApplicationSecurityConfiguration class delete UserDetailService , Bean**
6. **Change the configure bean in the ApplicationSecurityConfiguration class**

```
@Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth
        .ldapAuthentication()
                .userDnPatterns("uid={0},ou=people")
                .groupSearchBase("ou=groups")
                .authoritiesMapper(authoritiesMapper())
                .contextSource()
                .url("ldap://localhost:8389/dc=frankmoley,dc=com")
                .and()
                .passwordCompare()
                .passwordEncoder(new LdapShaPasswordEncoder())
                .passwordAttribute("userPassword");
    }
```

Build and run the application

Login as user : fpmoles

Password : password

AD for Authentication

AD Is Not LDAP

- Active Directory implements an LDAP API
- Directory Services exposed via AD Lightweight Directory Services exposed as an LDAP interface
- AD provides many, many more services and looking at the port requirements you can get a real feel for the weight

Authentication with AD

- Uses standard and non-standard methods
- Configuration should be very familiar, however
- Nested groups -> SEC-1823

Authentication Provider

- ActiveDirectoryLdapAuthenticationProvider
- Utilized via configure method of WebSecurityConfigurerAdapter
- Mostly the same as standard LDAP
- Group to role matching work

Introduction to OAuth 2

What Is OAuth 2?

- Protocol and framework for providing access to HTTP services
- Often used for third-party access
- Can be used for system-to-system communications in standalone or on behalf of a user

Parts of OAuth 2

- Resource owner - often the user
- Client - application requesting access
- Resource server - hosts protected data and accounts
- Authorization server - service that grants tokens

Token Types

- Access token - the secret and often short-lived token that identifies a user
- Refresh token - longer-lived token used to renew access token when it expires
- Scopes provide for rights associated with the access token

Grants

- Several grant types that impact flows
- Authorization code grant is most common
- Implicit is common in web apps and mobile apps
- Client credentials grant is useful in system-to-system comms

OAuth 2 and Spring

CommonOAuth2Provider

- Provides native support for Okta, Google, GitHub, and Facebook
- Property-based configuration in Spring Boot
- Client-side OAuth integration

Authorization Server

- Provides authorization services to the system
- @EnableAuthorizationServer
- AuthorizationServerConfigurerAdapter used to configure it
- Supports various grant types

Resource Server

- Provides the resources being protected
- @EnableResourceServer

OAuth 2 Client

- Full client-side support
- @EnableOauth2Client
- Oauth2RestTemplate provides much of the scaffolding
- Supports various grant types

Authorization Server

1. **Open guest-services pom.xml and add the dependency.**

```
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
    <version>2.3.0.RELEASE</version>
</dependency>
```

2. **Create a package auth and create a new class**

```
@EnableAuthorizationServer
@Configuration
public class AuthorizationServerConfig extends
AuthorizationServerConfigurerAdapter{

    @Override
    public void configure(AuthorizationServerSecurityConfigurer
security) throws Exception {
        security
            .passwordEncoder(NoOpPasswordEncoder.getInstance())
            .checkTokenAccess("permitAll()")
```

```
                    .tokenKeyAccess("permitAll()");

        }

        @Override
        public void configure(ClientDetailsServiceConfigurer clients)
throws Exception {
            clients
            .inMemory()
            .withClient("guest_app")
            .scopes("READ_ALL_GUESTS","WRITE_GUEST","UPDATE_GUEST")
            .secret("secret")
            .autoApprove(true)
            .authorities("ROLE_GUESTS_AUTHORIZED_CLIENT")
            .authorizedGrantTypes("client_credentials")
            .and()
            .withClient("api_audit")
            .scopes("READ_ALL_GUESTS")
            .secret("secret")
            .autoApprove(true)
            .authorities("ROLE_GUESTS_AUTHORIZED_CLIENT")
            .authorizedGrantTypes("client_credentials");
        }

        @Override
        public void configure(AuthorizationServerEndpointsConfigurer
endpoints) throws Exception {
            endpoints
            .tokenStore(new InMemoryTokenStore());
        }
}
```

Resource Server

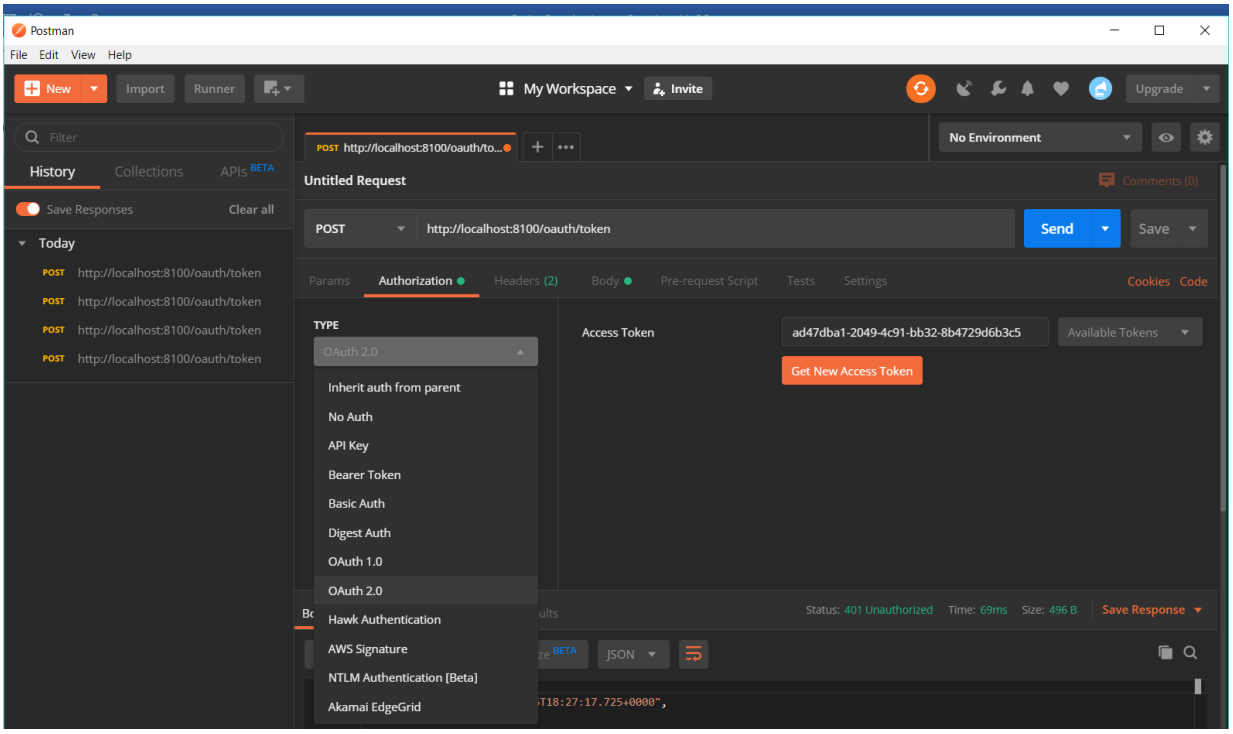
1. **Open GusetServicesApplication class in guest-services**
   Annotate with @EnableResouceServer

```
@SpringBootApplication
@EnableResourceServer
public class GuestServicesApplication {

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

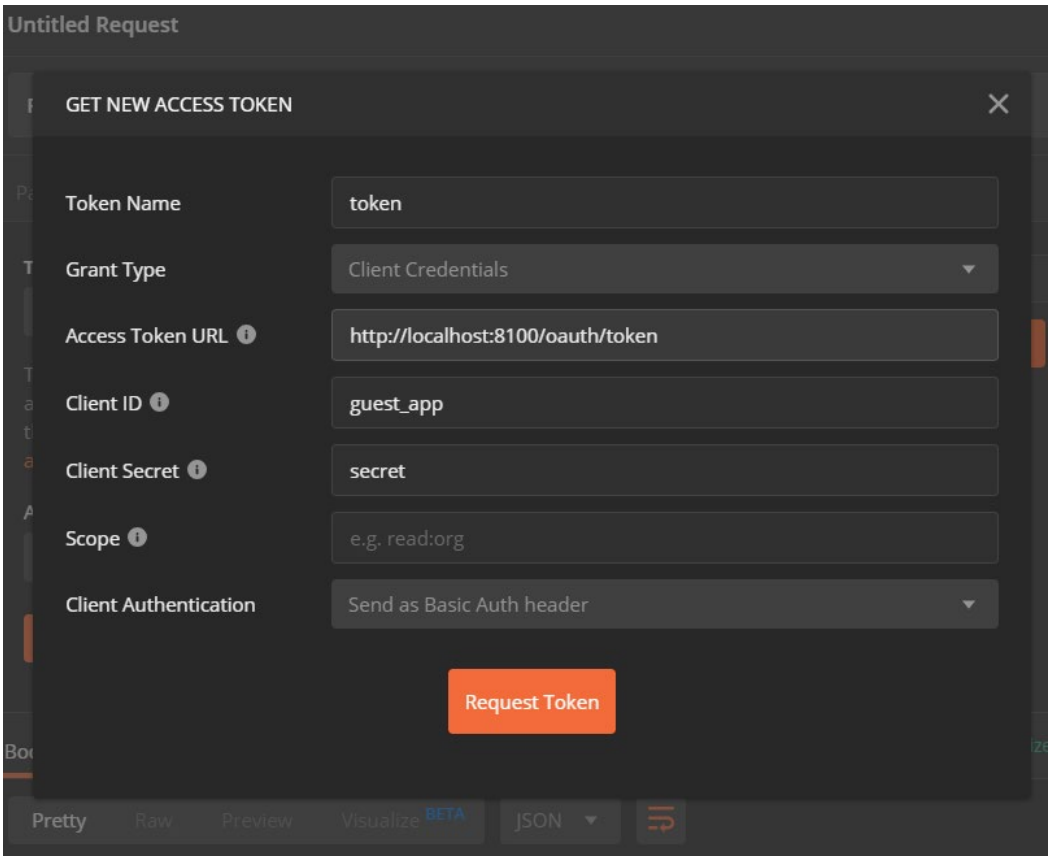
And we are done with the resource server.

Build and run the application

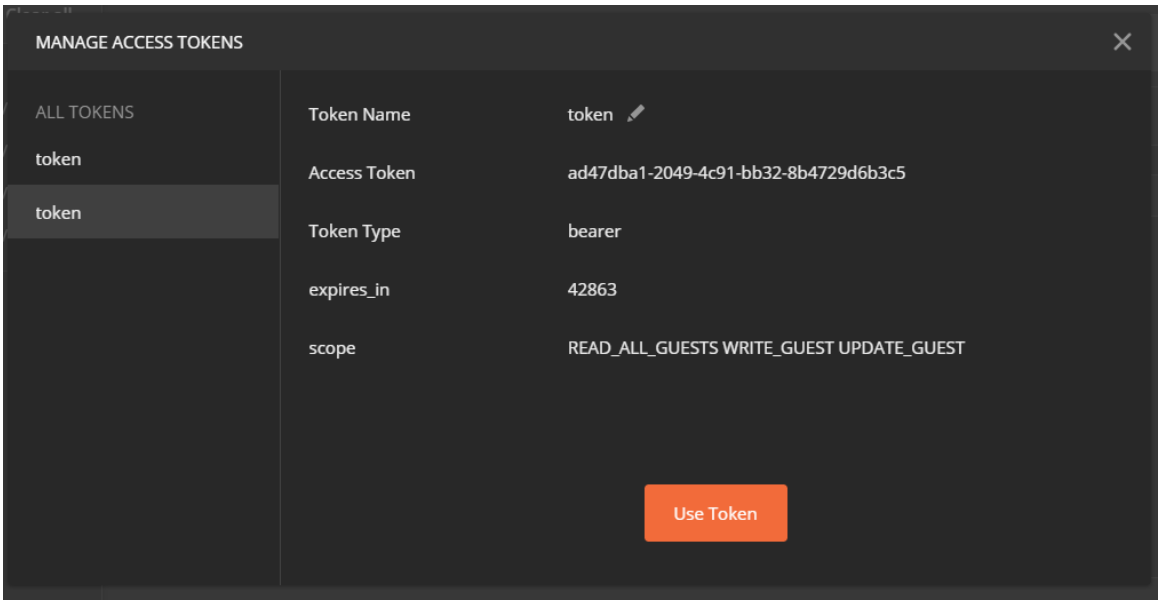
Now open postman and select Authorization tab and select OAuth2.0



Now click on Get Access Token and fill the fields

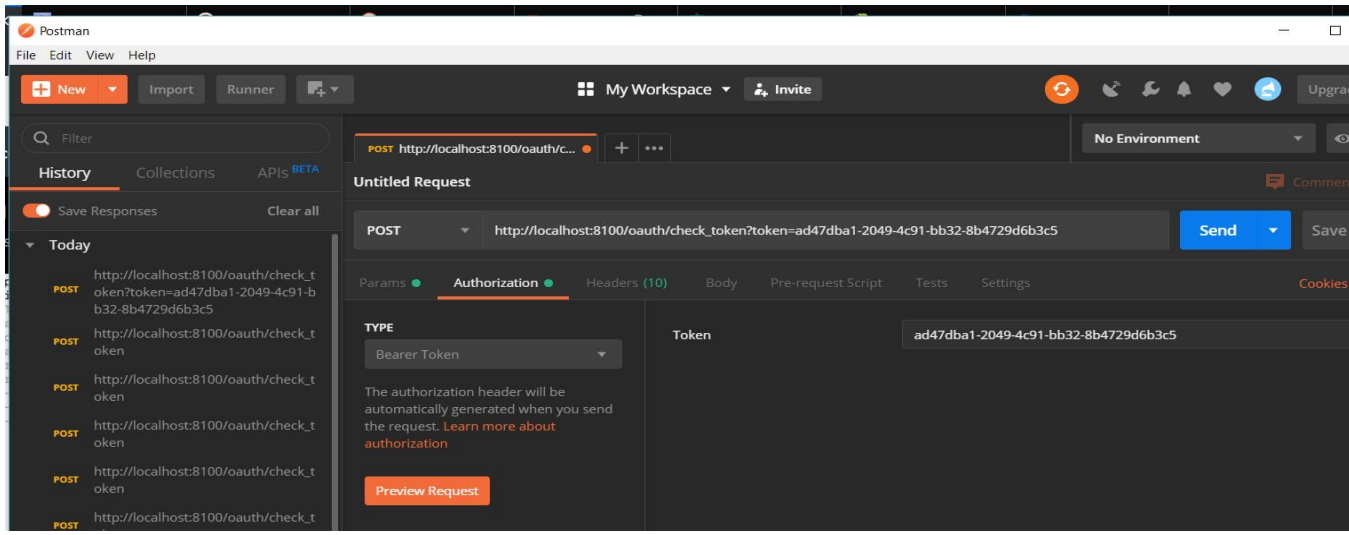


Click on request token



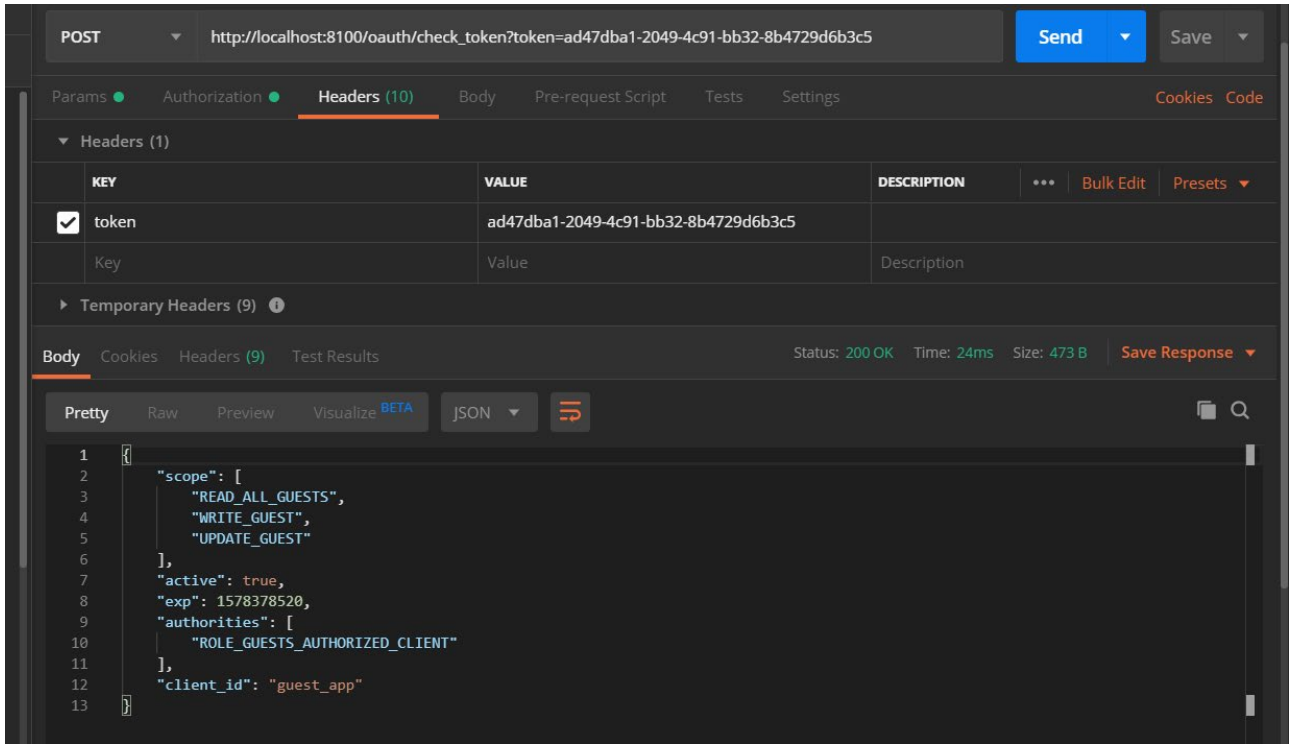
Now once you get the access token lets check the token

For this go to postman click on authorization tab and select Barer Token



Now click on Headers and select a key token and give the value of token look like this ad47dba1-2049-4c91-bb32-8b4729d6b3c5

Click on send



Client Side

Now if you run the guest-app2-ldap and login with fpmoles password you will get internal sever error in web and 401 in spring console log
So lets fix this

1. **Open pom.xml of guest_app2-ldap**

```
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
    <version>2.3.0.RELEASE</version>
</dependency>
```

2. **Open GuestService and create a constructor with RestTemplate**

```
@Service
public class GuestService {
    private static final String GUESTS = "/guests";
    private static final String SLASH = "/";

    @Value("${landon.guest.service.url}")
    private String guestServiceUrl;

    private final RestTemplate restTemplate;

    public GuestService(RestTemplate restTemplate) {
      super();
      this.restTemplate = restTemplate;
          }
```

3. **Now open GuestApplication class and Add annotation @EnableOAuthEnableClient and following Bean**

```
@SpringBootApplication
@EnableOAuth2Client
public class GuestAppApplication {
      public static void main(String[] args) {
            SpringApplication.run(GuestAppApplication.class, args);
      }
      private static final String AUTH_TOKEN_URL = "/oauth/token";
      @Value("${landon.guest.service.url}")
      private String serviceUrl;
      @Bean
```

```
      public OAuth2RestTemplate restTemplate() {
            ClientCredentialsResourceDetails resource =new
ClientCredentialsResourceDetails();
            resource.setAccessTokenUri(serviceUrl+AUTH_TOKEN_URL);
            resource.setClientId("guest_app");
            resource.setClientSecret("secret");
            resource.setGrantType("client_credentials");
            resource.setScope(new ArrayList<String>()
{{add("READ_ALL_GUESTS");add("WRITE_GUEST");add("UPDATE_GUEST");}});
            resource.setAuthenticationScheme(AuthenticationScheme.form);
            AccessTokenRequest request = new
DefaultAccessTokenRequest();

            return new OAuth2RestTemplate(resource,new
DefaultOAuth2ClientContext(request));
            }
```

4. **Now Build and run the guest-app2-ldap app**
   And login with fpmoles and password

- Just need authentication
- Don't want to manage
- Want further integration into remote platform
- Want social aspects

So Many Ways

- OAuth 2 is all over the place
- Working on unifying it all under Spring Security
- This video is based on spring-security-oauth2-client
- This should be considered the model to follow

Basic Flow

- Authorize application
- Set up redirect URL
- Configure application (application.yml)
- Run

Example Config

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            client-id: bb57...
            client-secret: 9dbc...
```

WebFlux Security

@EnableWebFluxSecurity

- Basic config maps everything to security
- SecurityWebFilterChain provides more fine-grained control
- MapReactiveUserDetailsService provides handle to UserDetailsServices

Principal

- Security model still based on principal
- Inject the Mono<Principal> into methods where you want a handle to it
- Still provides core functionality