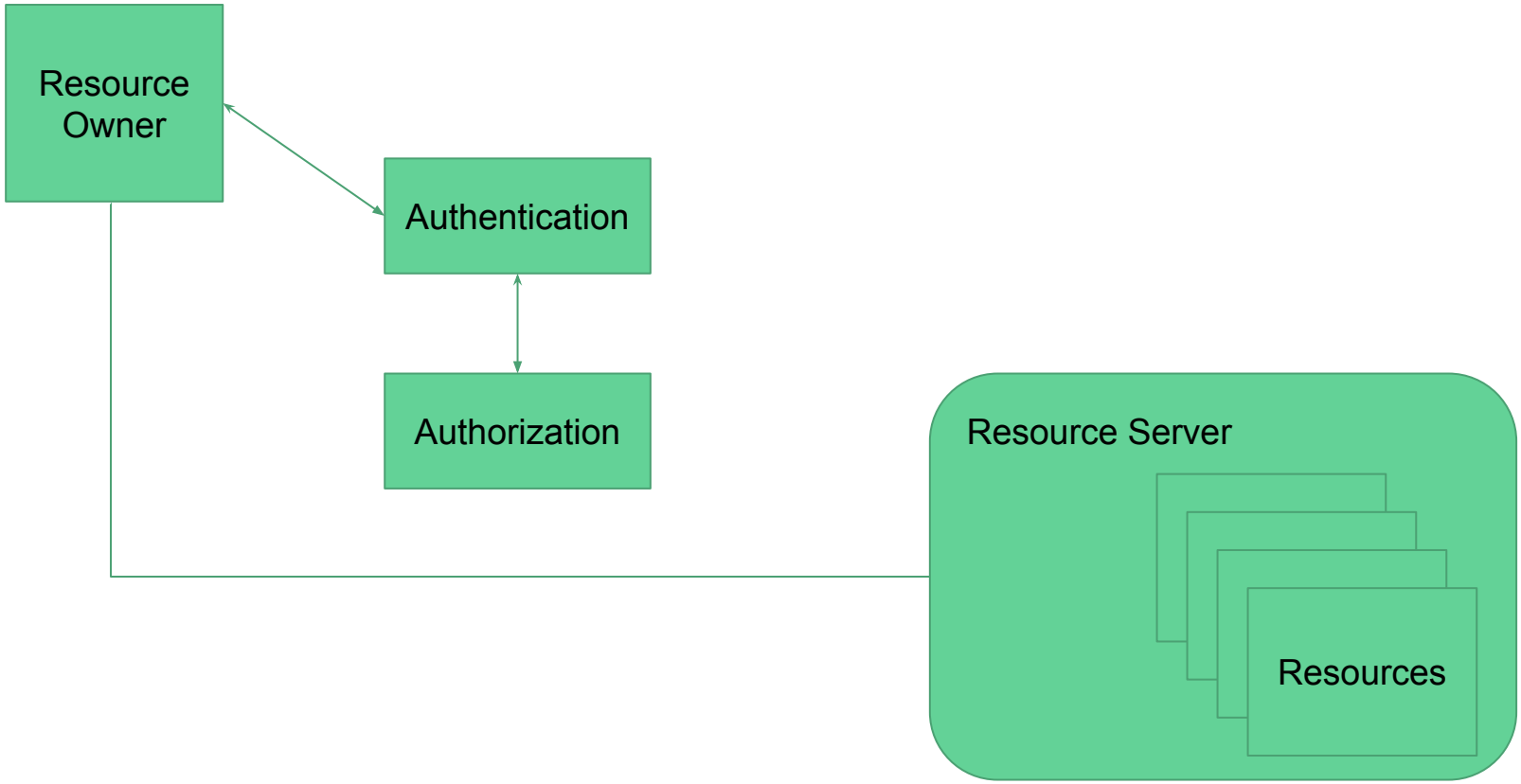
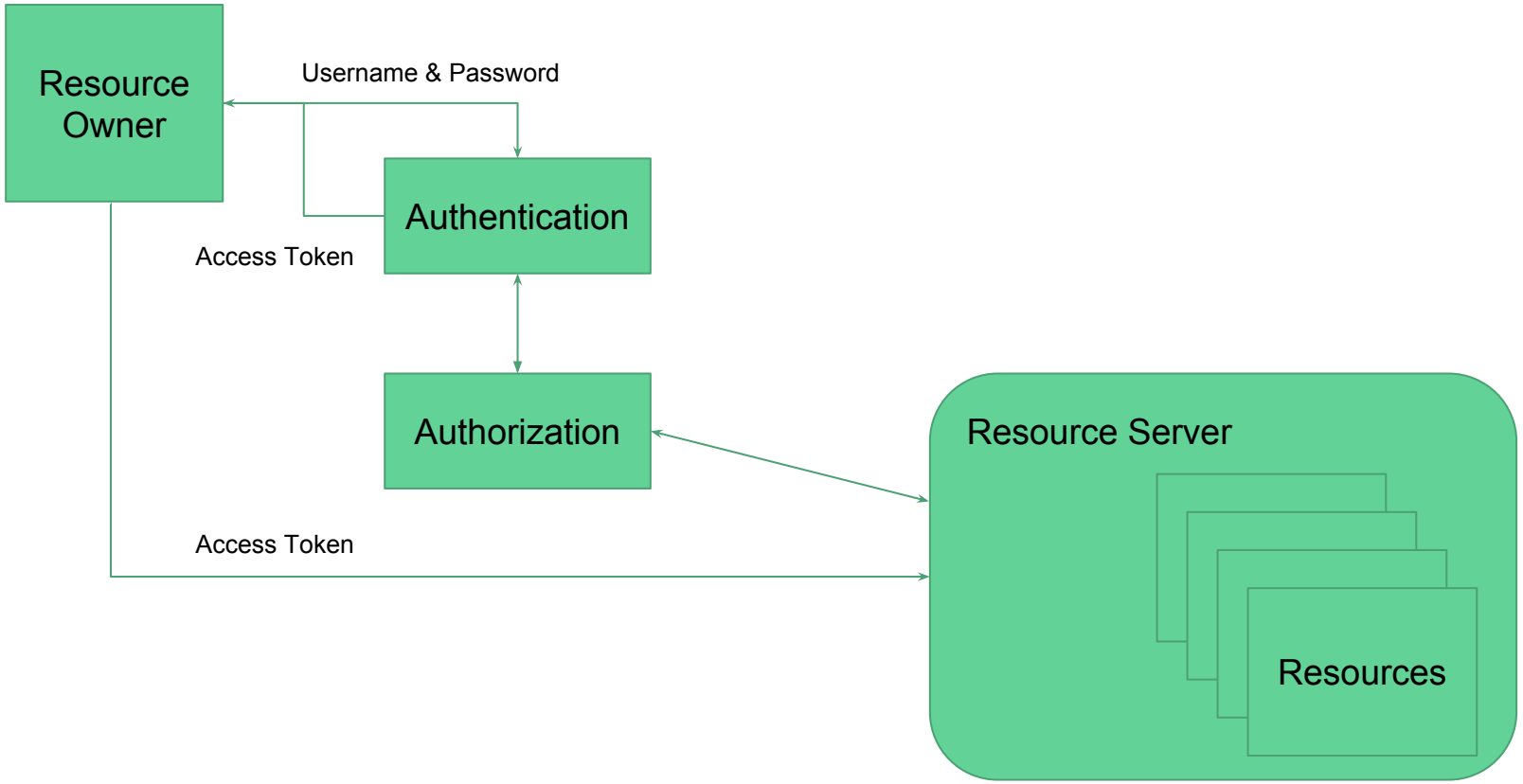
Spring Boot Security with OAuth2



Spring Boot Security with OAuth



User Authentication

- UserDetailsDetails Interface - Defines username, password, enabled etc.
- UserDetailsService @Service(“userDetailsService”) - provides ability to load a user by username so that password and credentials can be checked.
- AuthorizationServerConfigurerAdapter base class @EnableAuthorizationServer
  - define passwordEncoder
  - Hook up passwordEncoder
  - Configure authentication manager and userDetailsService
  - Configure clients
- Can be outside resource (i.e. Facebook, Google, GitHub, etc)
- Users Passes Credentials and Receives and receives an Access Token, Refresh Token, Expiration, and scope.

Authorization

- Restrict which Authenticated users can do what with the resources
- Defined in a Resource Server configuration class. @EnableResourceServer that extends ResourceServerConfigurerAdapter
- Configure HttpSecurity - Create rules for requests that match URLs
- Configure Resource IDs
- Implement AuthorizationServerConfigurerAdapter - Define clients, secret codes, expiration, scope, grant types, and resourceIds

Demo 17: Spring Security with OAuth2

1. Add `compile('org.springframework.boot:org.springframework.security.oauth')` and `compile('org.springframework.boot:spring-boot-starter-security')` dependencies to build.gradle
2. Create a OAuth2Config that extends AuthorizationServiceConfigurerAdapter
3. Create CustomerSecurity that implements CustomerDetailsService interface
4. Add findOneByUsername to CustomerRepository
5. Make Customer implement UserDetails, add new columns for security
6. Add ResourceServerConfig that extends ResourceServerConfigurerAdapter
7. Add WebSecurityConfig that extends WebSecurityConfigurerAdapter
8. Add @EnableResourceServer to GadgetsApplication
9. Create SQL resources with users and passwords, schema.sql and data.sql

Demo 17: Rest JSON Output

POST
http://localhost:8080/oauth/token?grant_type=password&username=geoff@example.com&password=password
Accept: application/json
Authorization: Basic Y29ycDpzZWNyZXQ=


```
{
  "access_token": "1f3f68ae-78dc-4fdd-bcf7-f48af3fa1fd7",
  "token_type": "bearer",
  "refresh_token": "3e1efefa-1b45-4261-8738-a6af919e0462",
  "expires_in": 3482,
  "scope": "read write"
}
```

Demo 17: Authentication Code

```
public class Customer implements UserDetails {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(nullable = false, updatable = false)
   private Long id;
```

```
@Service("userDetailsService")
public class CustomerSecurityService implements CustomerDetailsService {

   @Autowired
   private CustomerRepository userRepository;

   @Override
   public CustomerDetails loadCustomerByCustomername(String username) throws
CustomernameNotFoundException {
       return userRepository.findOneByUsername(username);
   }
}
```

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

  /**
   * Constructor disables the default security settings
   */
  public WebSecurityConfig() {
     super(true);
  }


  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
     return super.authenticationManagerBean();
  }

}
```

Demo 17: Authorization Code 1 of 2

```
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {

   @Override
   public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
      resources.resourceId("resource");
   }

   @Override
   public void configure(HttpSecurity http) throws Exception {
      http.requestMatchers().antMatchers("/users/**")
            .and()
            .authorizeRequests()
            .anyRequest().authenticated();
   }
}
```

Demo 17: Authorization Code 2 of 2

```
@Configuration
@EnableAuthorizationServer
public class OAuth2Config extends AuthorizationServerConfigurerAdapter {

  // secret = secret
  private static final String CORP_SECRET_BCRYPT =
"$2a$04$DQjbLE9xtfkN3T1cq3QL.u3OKhSrstz7wbywx9kyzraOwKJXM8Y9e";

  @Autowired
  @Qualifier("userDetailsService")
  private CustomerDetailsService userDetailsService;

  @Autowired
  private AuthenticationManager authenticationManager;

  @Value("${corp.oauth.tokentTimeout:3600}")
  private int expiration;

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

  @Override
  public void configure(AuthorizationServerEndpointsConfigurer configurer) {
     configurer.authenticationManager(authenticationManager);
     configurer.userDetailsService(userDetailsService);
  }

  @Override
  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
     clients.inMemory()
           .withClient("corp")
           .secret(CORP_SECRET_BCRYPT)
           .accessTokenValiditySeconds(expiration)
           .scopes("read", "write")
           .authorizedGrantTypes("password", "refresh_token")
           .resourceIds("resource");
  }
}
```

Lab 17: Spring Security using OAuth

1. Add compile('org.springframework.boot:org.springframework.security.oauth') and compile('org.springframework.boot:spring-boot-starter-security') dependencies to build.gradle
2. Create a OAuth2Config that extends AuthorizationServiceConfigurerAdapter
3. Create CustomerSecurity that implements CustomerDetailsService interface
4. Add User and UserRepository
5. Make User implement UserDetails, add columns for security
6. Add ResourceServerConfig that extends ResourceServerConfigurerAdapter
7. Add WebSecurityConfig that extends WebSecurityConfigurerAdapter
8. Add @EnableResourceServer to IndicatorApplication
9. Create SQL resources with users and passwords, schema.sql and data.sql
10. Protect the /indicator resources
11. Allow the corp client to access indicator
12. Create an additional client and allow access to just the /gadgets resources