Securing RESTful API with Spring Boot, Security, and Data MongoDB

by Didin J. on Mar 08, 2019

A comprehensive step by step tutorial on securing or authentication RESTful API with Spring Boot, Security, and Data MongoDB

A comprehensive step by step tutorial on securing or authentication RESTful API with Spring Boot, Security, and Data MongoDB. Previously, we have shown you how to securing Spring Boot, MVC and MongoDB web application (https://www.djamware.com/post/5b2f000880aca77b083240b2/spring-boot-security-and-data-mongodb-authentication-example). In this tutorial, the secure endpoint will restrict the access from an unauthorized request. Every request to secure endpoint should bring authorization token with it. Of course, there will be an endpoint for login which will get authorization token after successful login.

Table of Contents:

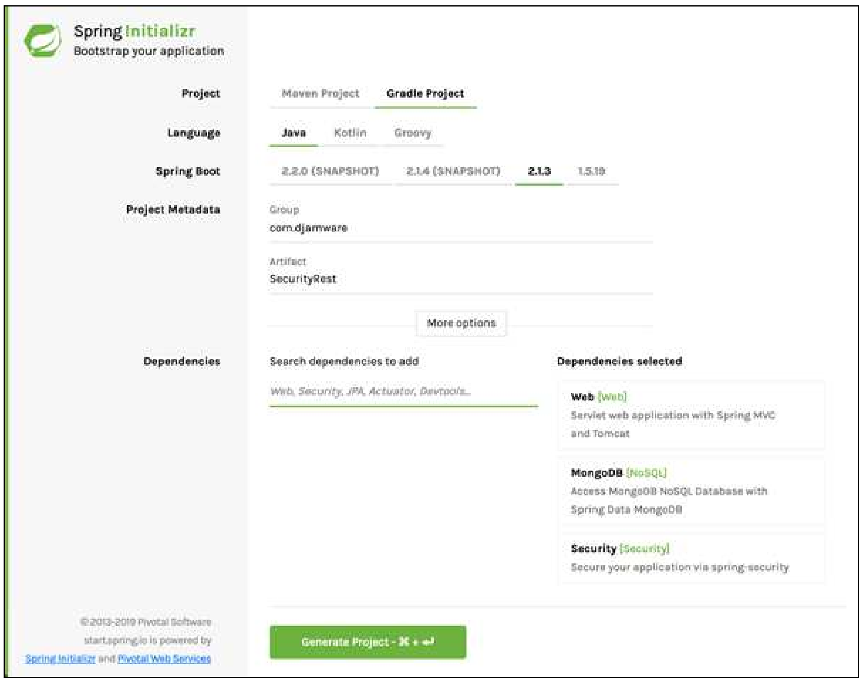
The following software, tools, and frameworks are required for this tutorial:

- Java Development Kit (JDK) 8 (https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
- Gradle (https://gradle.org/)
- Spring Boot (https://projects.spring.io/spring-boot/)
- Spring Data MongoDB (https://projects.spring.io/spring-data-mongodb/)
- MongoDB (https://www.mongodb.com/)
- Spring Security (https://spring.io/projects/spring-security)
- Spring Initializer (https://start.spring.io/)
- IDE or Text Editor (We are using Spring Tool Suite (https://spring.io/tools/sts))
- Terminal or cmd

We assume that you already installed all required software, tools, and frameworks. So, we will not cover how to install that software, tools, and frameworks.

1. Generate a New Spring Boot Gradle Project

To create or generate a new Spring Boot Application or Project, simply go to Spring Initializer (https://start.spring.io/). Fill all required fields as below then click on Generate Project button.



Project: Maven Project [✓ Gradle Project]

Language: [✓ Java] Kotlin Groovy

Spring Boot: 2.2.0 (SNAPSHOT) 2.1.4 (SNAPSHOT) [✓ 2.1.3] 1.5.19

Project Metadata

Group: com.djamware

Artifact: SecurityRest

Dependencies

Search dependencies to add: Web, Security, JPA, Actuator, Devtools...

Dependencies selected

Web [Web]
Servlet web application with Spring MVC and Tomcat

MongoDB [NoSQL]
Access MongoDB NoSQL Database with Spring Data MongoDB

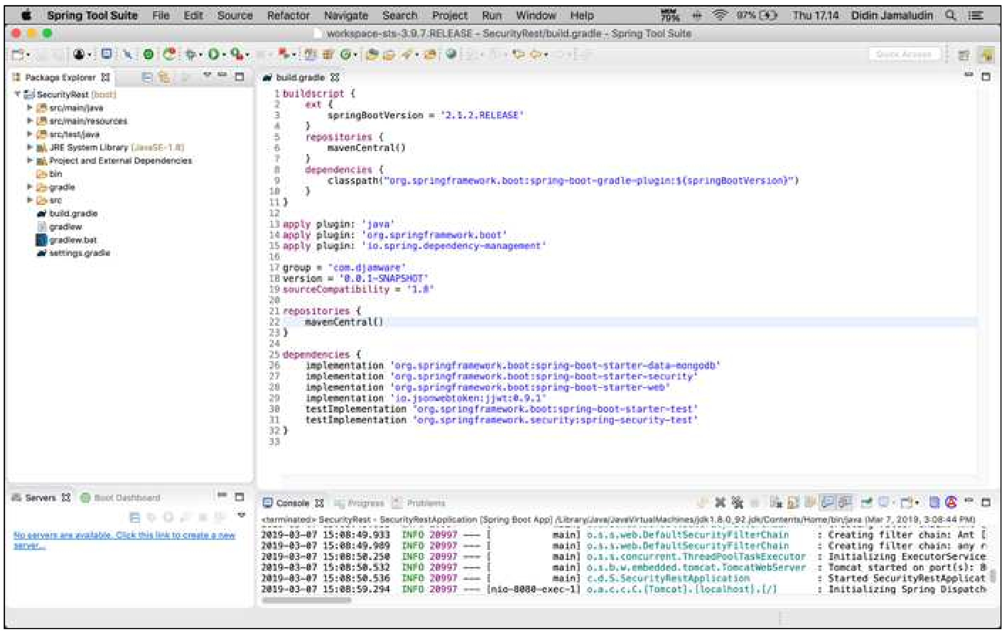
Security [Security]
Secure your application via spring-security

Generate Project - ⌘ + ↵
©2013-2019 Pivotal Software
start.spring.io is powered by
Spring Initializr and Pivotal Web Services

The project will automatically be downloaded as a Zip file. Next, extract the zipped project to your java projects folder. On the project folder root, you will find `build.gradle` file for register dependencies, initially it looks like this.

```
buildscript {
    ext {
        springBootVersion = '2.1.2.RELEASE'
    }
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

apply plugin: 'java'
apply plugin: 'org.springframework.boot'
apply plugin: 'io.spring.dependency-management'

group = 'com.djamware'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
}
```

Now, you can work with the source code of this Spring Boot Project using your own IDE or Text Editor. We are using Spring Tool Suite (STS). In STS, import the extracted zipped file as Existing Gradle Project.



```
buildscript {
    ext {
        springBootVersion = '2.1.2.RELEASE'
    }
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

apply plugin: 'java'
apply plugin: 'org.springframmwork.boot'
apply plugin: 'io.spring.dependency-management'

group = 'com.djamware'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'io.jsonwebtoken:jjwt:0.9.1'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
}
```

```
<terminated> SecurityRest - SecurityRestApplication [Spring Boot App] /Library/Java/JavaVirtualMachines/jdk1.8.0_92.jdk/Contents/Home/bin/java (Mar 7, 2019, 3:08:44 PM)
2019-03-07 15:08:49.933  INFO 20997 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: Ant [
2019-03-07 15:08:49.989  INFO 20997 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: any r
2019-03-07 15:08:50.250  INFO 20997 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService
2019-03-07 15:08:50.532  INFO 20997 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8
2019-03-07 15:08:50.536  INFO 20997 --- [           main] c.d.S.SecurityRestApplication            : Started SecurityRestApplicat
2019-03-07 15:08:59.294  INFO 20997 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispatch
```

Next, we have to add the JWT library to the `build.gradle` as the dependency. Open and edit `build.gradle` then add this line to dependencies after other implementation.

```
implementation 'io.jsonwebtoken:jjwt:0.9.1'
```

Next, compile the Gradle Project by type this command from Terminal or CMD.

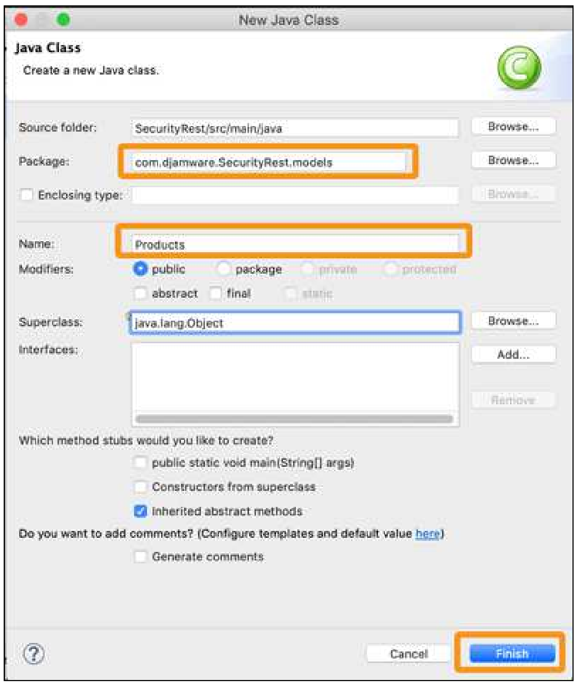
```
./gradlew compile
```

Or you can compile directly from STS by right-clicking the project name then choose Gradle -> Refresh Gradle Project. Next, open and edit `src/main/resources/application.properties` then add these lines.

```
spring.data.mongodb.database=springmongodb
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
```

2. Create Product, User and Role Model or Entity Classes

We will be creating all required models or entities for products, user and role. In STS, right-click the project name -> New -> Class. Fill the package with `com.djamware.SecurityRest.models`, the name with `Product`, and leave other fields and checkbox as default then click Finish Button.



New Java Class

Java Class

Create a new Java class.

Source folder: SecurityRest/src/main/java

Package: com.djamware.SecurityRest.models

Enclosing type:

Name: Products

Modifiers: [✓ public] package private protected

abstract final static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

[✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

Generate comments

Next, open and edit `src/main/java/com/djamware/SecurityRest/models/Product.java` then add this annotation above the class name that will point to MongoDB collection.

```
@Document(collection = "products")
```

Inside Product class, add these variables.

```
@Id
String id;
String prodName;
String prodDesc;
Double prodPrice;
String prodImage;
```

Add constructors after the variable or fields.

```
public Product() {
}

public Product(String prodName, String prodDesc, Double prodPrice, String prodImage) {
    super();
    this.prodName = prodName;
    this.prodDesc = prodDesc;
    this.prodPrice = prodPrice;
    this.prodImage = prodImage;
}
```

Generate or create Getter and Setter for each field.

```
public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getProdName() {
    return prodName;
}

public void setProdName(String prodName) {
    this.prodName = prodName;
}

public String getProdDesc() {
    return prodDesc;
}

public void setProdDesc(String prodDesc) {
    this.prodDesc = prodDesc;
}

public Double getProdPrice() {
    return prodPrice;
}

public void setProdPrice(Double prodPrice) {
    this.prodPrice = prodPrice;
}

public String getProdImage() {
    return prodImage;
}

public void setProdImage(String prodImage) {
    this.prodImage = prodImage;
}
```

Using STS you can organize imports automatically from the menu Source -> Organize Imports then you can see the imports after the package name.

```
package com.djamware.SecurityRest.models;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
```

You can do the same way as the above step for User and Role class. Here's the User class looks like.

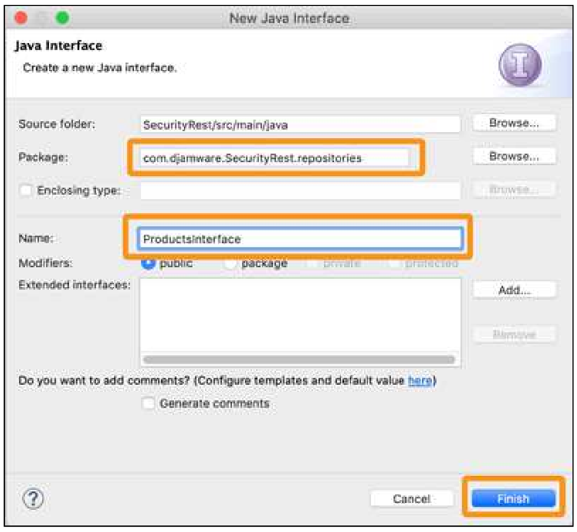
```
package com.djamware.SecurityRest.models;

import java.util.Set;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.IndexDirection;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "users")
public class User {

    @Id
    private String id;
    @Indexed(unique = true, direction = IndexDirection.DESCENDING, dropDups = true)
    private String email;
    private String password;
    private String fullname;
    private boolean enabled;
    @DBRef
    private Set<Role> roles;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getFullname() {
        return fullname;
    }
    public void setFullname(String fullname) {
        this.fullname = fullname;
    }
    public boolean isEnabled() {
        return enabled;
    }
    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }
    public Set<Role> getRoles() {
        return roles;
    }
    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

}
```

And the Role class will be like this.

```
package com.djamware.SecurityRest.models;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.IndexDirection;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "roles")
public class Role {

    @Id
    private String id;
    @Indexed(unique = true, direction = IndexDirection.DESCENDING, dropDups = true)

    private String role;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }

}
```

3. Create Product, User and Role Repository Interfaces

Next steps to create Product, User, and Role Repository Interfaces. From the STS, right-click the project name -> New -> Interface then fill all required fields and checkboxes as below before click Finish button.



New Java Interface

Java Interface

Create a new Java interface.

Source folder: SecurityRest/src/main/java

Package: com.djamware.SecurityRest.repositories

Enclosing type:

Name: ProductsInterface

Modifiers: [✓ public] package private protected

Extended interfaces:

Do you want to add comments? (Configure templates and default value here)

Generate comments

Next, open and edit `src/main/java/com/djamware/SecurityRest/repositories/ProductRepository.java` then add extends to MongoDB CRUD Repository.

```
public interface ProductRepository extends CrudRepository<Product, String> {

}
```

Inside the class name add this method.

```
@Override
void delete(Product deleted);
```

Organize all required imports.

```
import org.springframework.data.repository.CrudRepository;
import com.djamware.SecurityRest.models.Product;
```

The same way can be applied to User and Role repositories. So, the User Repository Interface will look like this.

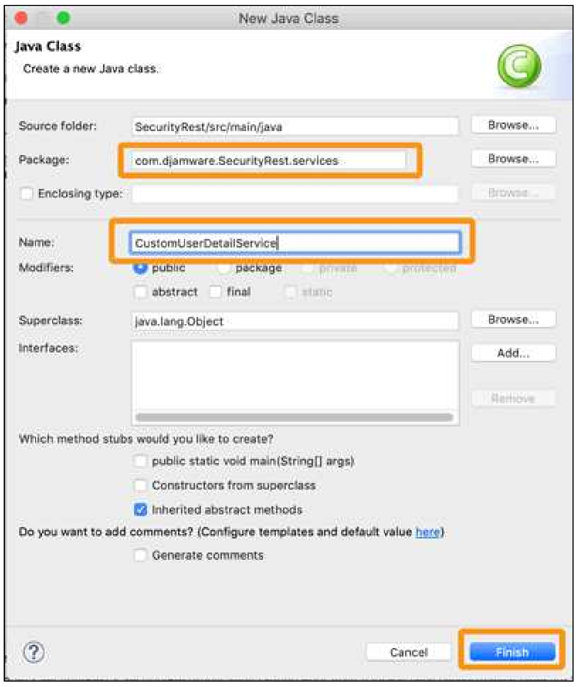
```
package com.djamware.SecurityRest.repositories;

import org.springframework.data.mongodb.repository.MongoRepository;
import com.djamware.SecurityRest.models.User;

public interface UserRepository extends MongoRepository<User, String> {

    User findByEmail(String email);
}
```

And the Role Repository Interface will look like this.

```
package com.djamware.SecurityRest.repositories;

import org.springframework.data.mongodb.repository.MongoRepository;
import com.djamware.SecurityRest.models.Role;

public interface RoleRepository extends MongoRepository<Role, String> {

    Role findByRole(String role);
}
```

4. Create a Custom User Details Service

To implements authentication using existing User and Role from MongoDB, we have to create a custom user details service. From the STS, right-click the project name -> New -> Class File then fill all required fields and checkboxes as below before clicking the finish button.



Source folder: SecurityRest/src/main/java

Package: com.djamware.SecurityRest.services

☐ Enclosing type:

Name: CustomUserDetailService

Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

Next, open and edit `src/main/java/com/djamware/SecurityRest/services/CustomUserDetailsService.java` then give an annotation above the class name and implement the Spring Security User Details Service.

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
}
```

Next, inject all required beans at the first line of the class bracket.

```
@Autowired
private UserRepository userRepository;

@Autowired
private RoleRepository roleRepository;

@Autowired
private PasswordEncoder bCryptPasswordEncoder;
```

Add a method to find a user by email field.

```
public User findUserByEmail(String email) {
    return userRepository.findByEmail(email);
}
```

Add a method to save a new user.

```
public void saveUser(User user) {
    user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
    user.setEnabled(true);
    Role userRole = roleRepository.findByRole("ADMIN");
    user.setRoles(new HashSet<>(Arrays.asList(userRole)));
    userRepository.save(user);
}
```

Override the Spring Security User Details to load User by email.

```
@Override
public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

    User user = userRepository.findByEmail(email);
    if(user != null) {
        List<GrantedAuthority> authorities = getUserAuthority(user.getRoles());
        return buildUserForAuthentication(user, authorities);
    } else {
        throw new UsernameNotFoundException("username not found");
    }
}
```

Add a method to get a set of Roles that related to a user.

Huippumallin salaisuus, miten laihtua 5 kiloa yhdessä viikossa
Tunnettu huippumalli paljastaa salaisuutensa – miten saada hoikka vartalo il ponnistu meneteln yllättävän riitää min päivässä.
suomi.slimmingdetox.eu

```
private List<GrantedAuthority> getUserAuthority(Set<Role> userRoles) {
    Set<GrantedAuthority> roles = new HashSet<>();
    userRoles.forEach((role) -> {
        roles.add(new SimpleGrantedAuthority(role.getRole()));
    });

    List<GrantedAuthority> grantedAuthorities = new ArrayList<>(roles);
    return grantedAuthorities;
}
```

Add a method for authentication purpose.

```
private UserDetails buildUserForAuthentication(User user, List<GrantedAuthority> authorities) {
    return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), authorities);
}
```

5. Configure Spring Boot Security Rest

Now, the main purpose of this tutorial is configuring Spring Security Rest. First, we have to create a bean for JWT token generation and validation. Right-click the project name -> New -> Class File. Fill the package name as `com.djamware.SecurityRest.configs` and the Class name as `JwtTokenProvider` then click the Finish button. Next, open and edit that newly created class file then give it an annotation above the class name.

```
@Component
public class JwtTokenProvider {
}
```

Add variables and injected bean inside the class bracket at the top lines.

```
@Value("${security.jwt.token.secret-key:secret}")
private String secretKey = "secret";

@Value("${security.jwt.token.expire-length:3600000}")
private long validityInMilliseconds = 3600000; // 1h

@Autowired
private CustomUserDetailsService userDetailsService;
```

Add a method for initialization.

```
@PostConstruct
protected void init() {
    secretKey = Base64.getEncoder().encodeToString(secretKey.getBytes());
}
```

Add a method to create a JWT token.

```
public String createToken(String username, Set<Role> set) {
    Claims claims = Jwts.claims().setSubject(username);
    claims.put("roles", set);
    Date now = new Date();
    Date validity = new Date(now.getTime() + validityInMilliseconds);
    return Jwts.builder()//
        .setClaims(claims)//
        .setIssuedAt(now)//
        .setExpiration(validity)//
        .signWith(SignatureAlgorithm.HS256, secretKey)//
        .compact();
}
```

Add a method to load User by username.

```
public Authentication getAuthentication(String token) {
    UserDetails userDetails = this.userDetailsService.loadUserByUsername(getUsername(token));
    return new UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());
}
```

Add a method to get the username by JWT token.

```
public String getUsername(String token) {
    return Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token).getBody().getSubject();
}
```

Add a method to resolve JWT token from request headers of Authorization that has a Bearer prefix.

```
public String resolveToken(HttpServletRequest req) {
    String bearerToken = req.getHeader("Authorization");
    if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
        return bearerToken.substring(7, bearerToken.length());
    }
    return null;
}
```

Add a method to validate a JWT token.

```
public boolean validateToken(String token) {
    try {
        Jws<Claims> claims = Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);
        if (claims.getBody().getExpiration().before(new Date())) {
            return false;
        }
        return true;
    } catch (JwtException | IllegalArgumentException e) {
        throw new JwtException("Expired or invalid JWT token");
    }
}
```

Finally, organize imports like below.

```
package com.djamware.SecurityRest.configs;

import java.util.Base64;
import java.util.Date;
import java.util.Set;

import javax.annotation.PostConstruct;
import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import com.djamware.SecurityRest.models.Role;
import com.djamware.SecurityRest.services.CustomUserDetailsService;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.JwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
```

Next, create a JWT filter class with the name `JwtTokenFilter` in configs package that extends Spring GenericFilterBean. Replace all Java codes with these lines of codes.

```
package com.djamware.SecurityRest.configs;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.GenericFilterBean;

public class JwtTokenFilter extends GenericFilterBean {

    private JwtTokenProvider jwtTokenProvider;

    public JwtTokenFilter(JwtTokenProvider jwtTokenProvider) {
        this.jwtTokenProvider = jwtTokenProvider;
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain filterChain)
        throws IOException, ServletException {
        String token = jwtTokenProvider.resolveToken((HttpServletRequest) req);
        if (token != null && jwtTokenProvider.validateToken(token)) {
            Authentication auth = token != null ? jwtTokenProvider.getAuthentication(token) : null;
            SecurityContextHolder.getContext().setAuthentication(auth);
        }
        filterChain.doFilter(req, res);
    }
}
```

Next, create a class with the name `JwtConfigurer` for JWT configuration in configs package then replace all codes with these lines of codes.

```
package com.djamware.SecurityRest.configs;

import org.springframework.security.config.annotation.SecurityConfigurerAdapter;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.DefaultSecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

public class JwtConfigurer extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

    private JwtTokenProvider jwtTokenProvider;

    public JwtConfigurer(JwtTokenProvider jwtTokenProvider) {
        this.jwtTokenProvider = jwtTokenProvider;
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        JwtTokenFilter customFilter = new JwtTokenFilter(jwtTokenProvider);
        http.addFilterBefore(customFilter, UsernamePasswordAuthenticationFilter.class);
    }
}
```

Finally, we have to configure the Spring Security by creating a Java class file inside configs package with the name \`WebSecurityConfig\`. Give annotations to this class and extends Spring WebSecurityConfigurerAdapter.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
}
```

Inject JWT token provider inside this class.

```
@Autowired
JwtTokenProvider jwtTokenProvider;
```

Add an override method to configure Authentication Manager Builder.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    UserDetailsService userDetailsService = mongoUserDetails();
    auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder());

}
```

Next, add an override method to configure Spring Security Http Security.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.httpBasic().disable().csrf().disable().sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS).and().authorizeRequests()
            .antMatchers("/api/auth/login").permitAll().antMatchers("/api/auth/register").permitAll()
            .antMatchers("/api/products/**").hasAuthority("ADMIN").anyRequest().authenticated().and().csrf()
            .disable().exceptionHandling().authenticationEntryPoint(unauthorizedEntryPoint()).and()
            .apply(new JwtConfigurer(jwtTokenProvider));
}
```

Next, declare all required beans for this configuration.

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

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

@Bean
public AuthenticationEntryPoint unauthorizedEntryPoint() {
    return (request, response, authException) -> response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
            "Unauthorized");
}

@Bean
public UserDetailsService mongoUserDetails() {
    return new CustomUserDetailsService();
}
```

6. Create Product and Authentication Controllers

Now it's time for REST API endpoint. All RESTful API will be created from each controller. Product controller will handle CRUD endpoint of product and Authentication controller will handle login and register endpoint. Right-click project name -> New -> Class then fill the package with `com.djamware.SecurityRest.controllers` and the class name as `ProductController`. Open and edit the newly created class file then replace all codes with these lines of codes.

```java
package com.djamware.SecurityRest.controllers;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.djamware.SecurityRest.models.Product;
import com.djamware.SecurityRest.repositories.ProductRepository;

@RestController
public class ProductController {

    @Autowired
    ProductRepository productRepository;

    @RequestMapping(method=RequestMethod.GET, value="/api/products")
    public Iterable<Product> product() {
        return productRepository.findAll();
    }

    @RequestMapping(method=RequestMethod.POST, value="/api/products")
    public String save(@RequestBody Product product) {
        productRepository.save(product);

        return product.getId();
    }

    @RequestMapping(method=RequestMethod.GET, value="/api/products/{id}")
    public Optional<Product> show(@PathVariable String id) {
        return productRepository.findById(id);
    }

    @RequestMapping(method=RequestMethod.PUT, value="/api/products/{id}")
    public Product update(@PathVariable String id, @RequestBody Product product) {
        Optional<Product> prod = productRepository.findById(id);
        if(product.getProdName() != null)
            prod.get().setProdName(product.getProdName());
        if(product.getProdDesc() != null)
            prod.get().setProdDesc(product.getProdDesc());
        if(product.getProdPrice() != null)
            prod.get().setProdPrice(product.getProdPrice());
        if(product.getProdImage() != null)
            prod.get().setProdImage(product.getProdImage());
        productRepository.save(prod.get());
        return prod.get();
    }

    @RequestMapping(method=RequestMethod.DELETE, value="/api/products/{id}")
    public String delete(@PathVariable String id) {
        Optional<Product> product = productRepository.findById(id);
        productRepository.delete(product.get());

        return "product deleted";
    }
}
```

For login, we need to create a POJO to mapping required fields of User. Create a new class file with the name `AuthBody` inside controllers package then replace all Java codes with these lines of codes.

```
package com.djamware.SecurityRest.controllers;

public class AuthBody {

    private String email;
    private String password;

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }


}
```

Finally, create a controller for authentication with the name `AuthController` inside the controllers' package. Open and edit that newly created file then replace all Java codes with these lines of codes.

```
package com.djamware.SecurityRest.controllers;

import static org.springframework.http.ResponseEntity.ok;

import java.util.HashMap;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.AuthenticationException;
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.djamware.SecurityRest.configs.JwtTokenProvider;
import com.djamware.SecurityRest.models.User;
import com.djamware.SecurityRest.repositories.UserRepository;
import com.djamware.SecurityRest.services.CustomUserDetailsService;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    JwtTokenProvider jwtTokenProvider;

    @Autowired
    UserRepository users;

    @Autowired
    private CustomUserDetailsService userService;

    @SuppressWarnings("rawtypes")
    @PostMapping("/login")
    public ResponseEntity login(@RequestBody AuthBody data) {
        try {
            String username = data.getEmail();
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, data.getPassword()));
            String token = jwtTokenProvider.createToken(username, this.users.findByEmail(username).getRoles());
            Map<Object, Object> model = new HashMap<>();
            model.put("username", username);
            model.put("token", token);
            return ok(model);
        } catch (AuthenticationException e) {
            throw new BadCredentialsException("Invalid email/password supplied");
        }
    }

    @SuppressWarnings("rawtypes")
    @PostMapping("/register")
    public ResponseEntity register(@RequestBody User user) {
        User userExists = userService.findUserByEmail(user.getEmail());
        if (userExists != null) {
            throw new BadCredentialsException("User with username: " + user.getEmail() + " already exists");
        }
        userService.saveUser(user);
        Map<Object, Object> model = new HashMap<>();
        model.put("message", "User registered successfully");
        return ok(model);
    }
}
```

7. Run and Test Spring Boot Security Rest using Postman

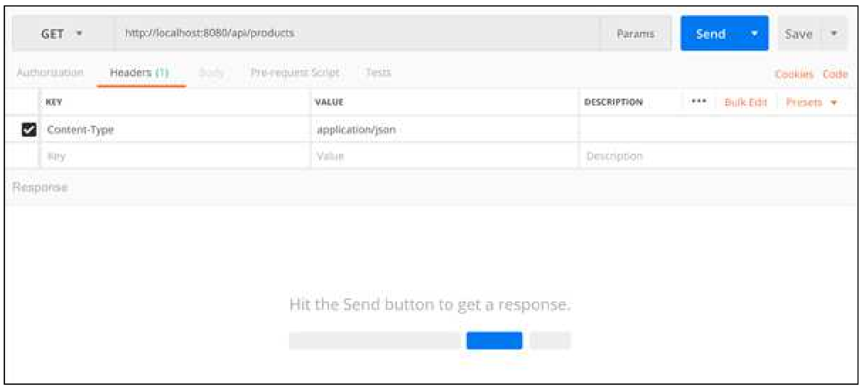
Before run and test the application, we have to populate a Role data first. Open and edit `src/main/java/com/djamware/SecurityRest/SecurityRestApplication.java` then add these lines of codes inside the initialization method.

```
@Bean
CommandLineRunner init(RoleRepository roleRepository) {

    return args -> {

        Role adminRole = roleRepository.findByRole("ADMIN");
        if (adminRole == null) {
            Role newAdminRole = new Role();
            newAdminRole.setRole("ADMIN");
            roleRepository.save(newAdminRole);
        }
    };

}
```

Next, make sure you have run the MongoDB server on your local machine then run the Gradle application using this command.

```
./gradlew bootRun
```

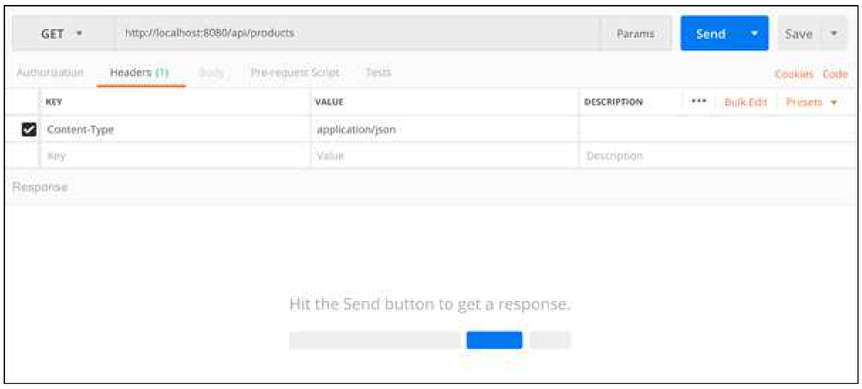
Or in STS just right-click the project name -> Run As -> Spring Boot App. Next, open the Postman application then change the method to `GET` and address to `localhost:8080/api/products` then click Send button.



You will see this response in the bottom panel of Postman.

```
{
    "timestamp": "2019-03-07T13:16:34.935+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "Unauthorized",
    "path": "/api/products"
}
```

Next, change the method to POST then address to `localhost:8080/api/auth/register` then fill the body with raw data as below image then click Send button.



Content-Type: application/json

You will get the response in the bottom panel of Postman.

```
{
    "message": "User registered successfully"
}
```

Next, change the address to `localhost:8080/api/auth/login` and change the body as below then click Send button.

```
{ "email":"info@djamware.com", "password": "q1w2we3r4" }
```

You will see this response in the bottom panel of Postman.

```
{
    "username": "info@djamware.com",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJpbmZvQGRqYW13YXJlLmNvbSIsInJvbGVzIjpbeyJpZCI6IjVjODBjNjIzYjIwMTkxNGIyYTY5N2U4ZCIsInJvbG
UiOiJBRE1JTiJ9XSwiaWF0IjoxNTUxOTY0OTc3LCJleHAiOjE1NTE5Njg1Nzd9.j5CHZ_LCmeQtdxQeH9eluxVXcOsHPWV1p8WnBn0CULo"
}
```

Copy the token then back to the GET product. Add a header with the name `Authorization` and the value that paste from a token that gets by login with additional `Bearer ` prefix (with space) as below.

```
Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJpbmZvQGRqYW13YXJlLmNvbSIsInJvbGVzIjpbeyJpZCI6IjVjODBjNjIzYjIwMTkxNGIyYTY5N2U4ZCIsInJvbGUiOiJBR
E1JTiJ9XSwiaWF0IjoxNTUxOTY0OTc3LCJleHAiOjE1NTE5Njg1Nzd9.j5CHZ_LCmeQtdxQeH9eluxVXcOsHPWV1p8WnBn0CULo
```

You should see this response after clicking the Send button.

```
[
    {
        "id": "5c80dc6cb20191520567b68a",
        "prodName": "Dummy Product 1",
        "prodDesc": "The Fresh Dummy Product in The world part 1",
        "prodPrice": 100,
        "prodImage": "https://dummyimage.com/600x400/000/fff"
    }
]
```

You can test the POST product with the token in headers using the same way.

That it's, the Securing RESTful API with Spring Boot, Security, and Data MongoDB tutorial. You can get the full source code from our GitHub (https://github.com/didinj/spring-boot-security-rest.git).

That just the basic. If you need more deep learning about Java and Spring Framework you can take the following cheap course:

- Master Java Web Services and REST API with Spring Boot (https://click.linksynergy.com/link?id=6nYo96*QrJE&offerid=358574.1302610&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fspring-web-services-tutorial%2F)
- JDBC Servlets and JSP - Java Web Development Fundamentals (https://click.linksynergy.com/link?id=6nYo96*QrJE&offerid=358574.603778&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fjdbcservletsandjsp%2F)
- The Complete Java Web Development Course (https://click.linksynergy.com/link?id=6nYo96*QrJE&offerid=358574.1522560&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fjava-web-development-course%2F)
- Spring MVC For Beginners: Build Java Web App in 25 Steps (https://click.linksynergy.com/link?id=6nYo96*QrJE&offerid=358574.732464&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fspring-mvc-tutorial-for-beginners-step-by-step%2F)
- Practical RESTful Web Services with Java EE 8 (JAX-RS 2.1) (https://click.linksynergy.com/link?id=6nYo96*QrJE&offerid=358574.1257336&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fjava-ee-8-tutorials-restful-web-services-with-jax-rs-21-for-java-devs%2F)

Thanks!

Follow

(http://www.facebook.com/djamwarecom)

(http://twitter.com/intent/follow?source=followbutton&variant=1.0&screen_name=djamware)

(http://www.pinterest.com/djamware) (http://www.linkedin.com/in/didin-jamaludin-7a530351)

(http://www.youtube.com/channel/UCtI81hYLh2Ae_45KHkyy0vw?sub_confirmation=1)

(https://github.com/didinj)

Previous Article

Spring Boot, Security, and Data MongoDB Authentication Example

(/post/5b2f000880aca77b083240b2/spring-boot-security-and-data-mongodb-authentication-example)

Related Articles

- Spring Boot, Security, and Data MongoDB Authentication Example (/post/5b2f000880aca77b083240b2/spring-boot-security-and-data-mongodb-authentication-example)
- Building Spring Boot, MongoDB and React.js CRUD Web Application (/post/5ab6397c80aca714d19d5b9c/building-spring-boot-mongodb-and-reactjs-crud-web-application)
- Spring Boot, MongoDB and Angular 5 CRUD Java Web Application (/post/5a792ecb80aca7059c142978/spring-boot-mongodb-and-angular-5-crud-java-web-application)
- Spring Boot + MongoDB Slack Bot Example (/post/5a44631080aca7059c142971/spring-boot-mongodb-slack-bot-example)
- Tutorial of Building Java REST API using Spring Boot and MongoDB (/post/59be51e780aca768e4d2b140/tutorial-of-building-java-rest-api-using-spring-boot-and-mongodb)
- Spring Boot, MVC, Data and MongoDB CRUD Java Web Application (/post/59b606e280aca768e4d2b13b/spring-boot-mvc-data-and-mongodb-crud-java-web-application)
- Ubuntu 16.04: Install SSL on Nginx and Tomcat 7 (/post/5895452f80aca70683707ee3/ubuntu-1604-install-ssl-on-nginx-and-tomcat-7)
- Install Nginx, Tomcat 7 and Java 8 on Ubuntu 16.04 (/post/588df76680aca722878a364a/install-nginx-tomcat-7-and-java-8-on-ubuntu-1604)
- How to Create Java Web Application using Netbeans 8.2 (/post/58721cdb80aca723c115bead/how-to-create-java-web-application-using-netbeans-82)
- Parse and Format Date Time in Java 8 (/post/585bb21d80aca73b19a2efd6/parse-and-format-date-time-in-java-8)
- Java Class and Object Example (/post/584a792a80aca72c1ecc65be/java-class-and-object-example)
- How to Create Java Application Using Netbeans 8.2 (/post/58494f8780aca721005e8677/how-to-create-java-application-using-netbeans-82)

0 Comments **Djamware - Fullstack Programming Tutorials** 1 **Login**

♡ **Recommend** 1 Tweet Share **Sort by Best**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON **DJAMWARE - FULLSTACK PROGRAMMING TUTORIALS**

Angular 7 Tutorial: Create Angular Material CDK Virtual Scroll

1 comment • 4 months ago

Zane — Is there instructions on how to do this with the cdk Table or Material Design table? Those tables just take a [DataSource] and not an ngFor, so i wasn't quite sure how to convert my cdk Table to …

Secure Node.js, Express.js and PostgreSQL API using Passport.js

16 comments • 4 months ago

Didin Jamaludin — check the port that use by express, or might be the express not running properly. otherwise, compare with working source code on github https://github.com/didinj/s...

Node, Express, Sequelize, and PostgreSQL Association Example

2 comments • 5 months ago

BMad — When you create the initial models with the model:create, it creates the corresponding migrations. After you then go in and add associations to the model files, in the example I dont see that you …

Angular 6 Firebase Tutorial: Firestore CRUD Web Application

3 comments • 5 months ago

Didin Jamaludin — In your firebase console project, under Develop side menu -> Authentication then you will see Web Setup button at the top right of the screen. Just click that button and you will get all …

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Disqus' Privacy PolicyPrivacy PolicyPrivacy**

ezoic (https://www.ezoic.com/what-is-ezoic/)

report this ad

ezoic (https://www.ezoic.com/what-is-ezoic/)

report this ad

Programming Blog

(/post-category/595f867edbd39e7571e183dc/programming-blog)

CSS 3

(/post-sub-category/584249bde4d5d303658d1ecf/css-3)

React Native
(/post-sub-category/5b4aa0b480aca707dd4f65a6/react-native)
HTML 5 Tutorial
(/post-sub-category/584209dffcbe618f680bdc5c/html-5-tutorial)
MongoDB
(/post-sub-category/5845677b80aca7763489d871/mongodb)
Ionic Framework
(/post-sub-category/5845691a80aca7763489d872/ionic-framework)
ASP.NET Core
(/post-sub-category/5c50643780aca754f7a9d1e9/aspnet-core)
Node.js
(/post-sub-category/58a9196f80aca748640ce352/nodejs)
Groovy and Grails
(/post-sub-category/585b3fa380aca73b19a2efd4/groovy-and-grails)
Javascript
(/post-sub-category/583d6d30fcbe614473a4c4e9/javascript)
Java
(/post-sub-category/583d6c37fcbe614473a4c4e8/java)
All Articles
(/public/allArticles)

Popular Articles:

- Angular 7 Tutorial: Building CRUD Web Application (/post/5bca67d780aca7466989441f/angular-7-tutorial-building-crud-web-application)
- Angular 6 HttpClient: Consume RESTful API Example (/post/5b87894280aca74669894414/angular-6-httpclient-consume-restful-api-example)
- Ionic 4, Angular 7 and Cordova Tutorial: Build CRUD Mobile Apps (/post/5be52ce280aca72b942e31bc/ionic-4-angular-7-and-cordova-tutorial-build-crud-mobile-apps)
- Node, Express, Angular 7, GraphQL and MongoDB CRUD Web App (/post/5c75d68880aca754f7a9d1ed/node-express-angular-7-graphql-and-mongodb-crud-web-app)
- Building Web App using ASP.NET Web API Angular 7 and SQL Server (/post/5c50e5f280aca754f7a9d1eb/building-web-app-using-aspnet-web-api-angular-7-and-sql-server)
- Building CRUD Mobile App using Ionic 4, Angular 6 and Cordova (/post/5b5cffaf80aca707dd4f65aa/building-crud-mobile-app-using-ionic-4-angular-6-and-cordova)
- Ionic 3, Angular 4 and SQLite CRUD Offline Mobile App (/post/59c53a1280aca768e4d2b143/ionic-3-angular-4-and-sqlite-crud-offline-mobile-app)
- MEAN Stack Angular 6 CRUD Web Application (/post/5b00bb9180aca726dee1fd6d/mean-stack-angular-6-crud-web-application)
- Ionic 3 Consuming REST API using New Angular 4.3 HttpClient (/post/59924f9080aca768e4d2b12e/ionic-3-consuming-rest-api-using-new-angular-43-httpclient)
- How to Upload File on Ionic 3 using Native File Transfer Plugin (/post/599da16580aca768e4d2b130/how-to-upload-file-on-ionic-3-using-native-file-transfer-plugin)