**Code ví dụ Spring Boot JSON Web Token (JWT)**

Posted on *[Tháng Tư 4, 2018](https://stackjava.com/spring/code-vi-du-spring-boot-json-web-token-jwt.html)*

Code ví dụ Spring Boot JSON Web Token (JWT)

(Xem lại: [Code ví dụ Spring MVC – Spring Security JSON Web Token](https://stackjava.com/spring/code-vi-du-json-web-token-spring-security-restful-webservice.html). (XML config))

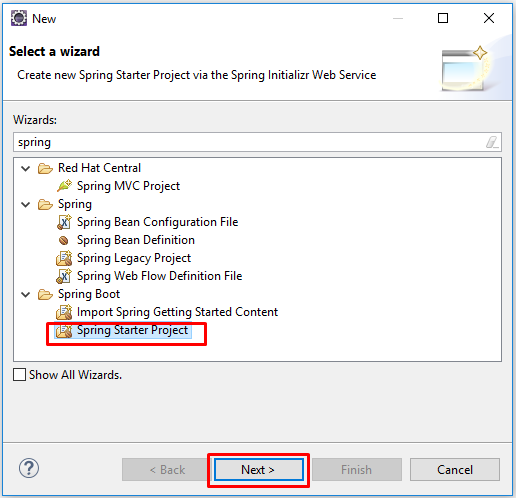
(Xem lại: [JSON Web Token là gì?](https://stackjava.com/spring/code-vi-du-json-web-token-voi-spring-security-jwt.html))

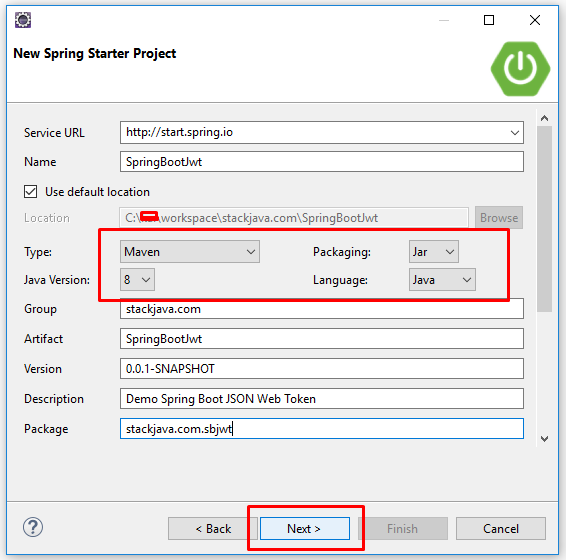
(Xem lại: [Code ví dụ Spring Boot RESTful Webservice (CRUD)](https://stackjava.com/spring/code-vi-du-spring-boot-restful-webservice-crud.html))

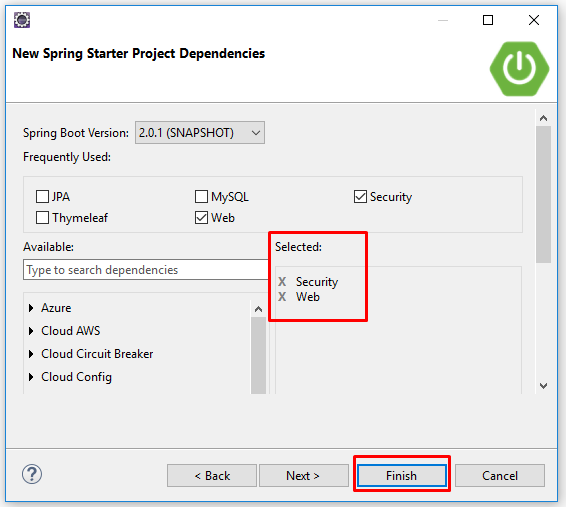
**Các công nghệ sử dụng:**

* Spring Boot 2.0.1 + Spring Security
* [Maven](https://stackjava.com/install/maven-phan-1-maven-la-gi-cai-dat-maven.html)
* [JDK 1.8](https://stackjava.com/install/cai-dat-java-9-cai-dat-bien-moi-truong-java.html)
* [Eclipse + Spring Tool Suite](https://stackjava.com/install/cai-dat-spring-tool-suite-cho-eclipse.html)

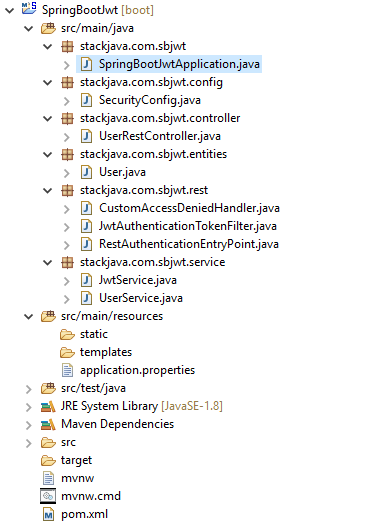
**Tạo Spring Boot Project**

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-multiple-form-login-1.png)

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-2.png)

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-5.png)

**Cấu trúc Project**

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-4.png)

Mình sử dụng thêm thư viện ***nimbus-jose-jwt*** để tạo token (decode/encode token, chữ ký số) và ***commons-lang3*** để xử lý Strings

* <!-- JWT -->
* <dependency>
* <groupId>com.nimbusds</groupId>
* <artifactId>nimbus-jose-jwt</artifactId>
* <version>4.20</version>
* </dependency>
* <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
* <dependency>
* <groupId>org.apache.commons</groupId>
* <artifactId>commons-lang3</artifactId>
* </dependency>

**File entity:**

* **User.java**
* package stackjava.com.sbjwt.entities;
* import java.util.ArrayList;
* import java.util.List;
* import org.springframework.security.core.GrantedAuthority;
* import org.springframework.security.core.authority.SimpleGrantedAuthority;
* import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
* import com.fasterxml.jackson.annotation.JsonInclude;
* @JsonInclude(JsonInclude.Include.NON\_NULL)
* @JsonIgnoreProperties(value = { "roles", "authorities" })
* public class User {
* private int id;
* private String username;
* private String password;
* private String[] roles;
* public User() {
* }
* public User(int id, String username, String password) {
* this.id = id;
* this.username = username;
* this.password = password;
* }
* public List<GrantedAuthority> getAuthorities() {
* List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
* for (String role : roles) {
* authorities.add(new SimpleGrantedAuthority(role));
* }
* return authorities;
* }
* // getter-setter
* }

**File Controller:**

* **UserRestController.java**
* package stackjava.com.sbjwt.controller;
* import java.util.List;
* import javax.servlet.http.HttpServletRequest;
* import org.springframework.beans.factory.annotation.Autowired;
* import org.springframework.http.HttpStatus;
* import org.springframework.http.ResponseEntity;
* 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 stackjava.com.sbjwt.entities.User;
* import stackjava.com.sbjwt.service.JwtService;
* import stackjava.com.sbjwt.service.UserService;
* @RestController
* @RequestMapping("/rest")
* public class UserRestController {
* @Autowired
* private JwtService jwtService;
* @Autowired
* private UserService userService;
* /\* ---------------- GET ALL USER ------------------------ \*/
* @RequestMapping(value = "/users", method = RequestMethod.GET)
* public ResponseEntity<List<User>> getAllUser() {
* return new ResponseEntity<List<User>>(userService.findAll(), HttpStatus.OK);
* }
* /\* ---------------- GET USER BY ID ------------------------ \*/
* @RequestMapping(value = "/users/{id}", method = RequestMethod.GET)
* public ResponseEntity<Object> getUserById(@PathVariable int id) {
* User user = userService.findById(id);
* if (user != null) {
* return new ResponseEntity<Object>(user, HttpStatus.OK);
* }
* return new ResponseEntity<Object>("Not Found User", HttpStatus.NO\_CONTENT);
* }
* /\* ---------------- CREATE NEW USER ------------------------ \*/
* @RequestMapping(value = "/users", method = RequestMethod.POST)
* public ResponseEntity<String> createUser(@RequestBody User user) {
* if (userService.add(user)) {
* return new ResponseEntity<String>("Created!", HttpStatus.CREATED);
* } else {
* return new ResponseEntity<String>("User Existed!", HttpStatus.BAD\_REQUEST);
* }
* }
* /\* ---------------- DELETE USER ------------------------ \*/
* @RequestMapping(value = "/users/{id}", method = RequestMethod.DELETE)
* public ResponseEntity<String> deleteUserById(@PathVariable int id) {
* userService.delete(id);
* return new ResponseEntity<String>("Deleted!", HttpStatus.OK);
* }
* @RequestMapping(value = "/login", method = RequestMethod.POST)
* public ResponseEntity<String> login(HttpServletRequest request, @RequestBody User user) {
* String result = "";
* HttpStatus httpStatus = null;
* try {
* if (userService.checkLogin(user)) {
* result = jwtService.generateTokenLogin(user.getUsername());
* httpStatus = HttpStatus.OK;
* } else {
* result = "Wrong userId and password";
* httpStatus = HttpStatus.BAD\_REQUEST;
* }
* } catch (Exception ex) {
* result = "Server Error";
* httpStatus = HttpStatus.INTERNAL\_SERVER\_ERROR;
* }
* return new ResponseEntity<String>(result, httpStatus);
* }
* }

**File Service:**

* **UserService.java**
* package stackjava.com.sbjwt.service;
* import java.util.ArrayList;
* import java.util.List;
* import org.apache.commons.lang3.StringUtils;
* import org.springframework.stereotype.Service;
* import stackjava.com.sbjwt.entities.User;
* @Service
* public class UserService {
* public static List<User> listUser = new ArrayList<User>();
* static {
* User userKai = new User(1, "kai", "123456");
* userKai.setRoles(new String[] { "ROLE\_ADMIN" });
* User userSena = new User(2, "sena", "123456");
* userSena.setRoles(new String[] { "ROLE\_USER" });
* listUser.add(userKai);
* listUser.add(userSena);
* }
* public List<User> findAll() {
* return listUser;
* }
* public User findById(int id) {
* for (User user : listUser) {
* if (user.getId() == id) {
* return user;
* }
* }
* return null;
* }
* public boolean add(User user) {
* for (User userExist : listUser) {
* if (user.getId() == userExist.getId() || StringUtils.equals(user.getUsername(), userExist.getUsername())) {
* return false;
* }
* }
* listUser.add(user);
* return true;
* }
* public void delete(int id) {
* listUser.removeIf(user -> user.getId() == id);
* }
* public User loadUserByUsername(String username) {
* for (User user : listUser) {
* if (user.getUsername().equals(username)) {
* return user;
* }
* }
* return null;
* }
* public boolean checkLogin(User user) {
* for (User userExist : listUser) {
* if (StringUtils.equals(user.getUsername(), userExist.getUsername())
* && StringUtils.equals(user.getPassword(), userExist.getPassword())) {
* return true;
* }
* }
* return false;
* }
* }

Mình khởi tạo ***listUser***và thực hiện truy vấn, login thêm/sửa/xóa user trên list này.

Ban đầu mình sẽ khởi tạo 2 tài khoản là kai/123456 với role ADMIN và sena/123456 với role là USER để đăng nhập và truy vấn.

(Các bạn có thể xem lại bài [Spring Boot Security Hibernate + MySQL + Eclipse + Maven](https://stackjava.com/spring/code-vi-du-spring-boot-security-hibernate-mysql-eclipse.html) để làm việc trực tiếp với database).

* **JwtService.java**
* package stackjava.com.sbjwt.service;
* import java.util.Date;
* import org.springframework.stereotype.Service;
* import com.nimbusds.jose.JWSAlgorithm;
* import com.nimbusds.jose.JWSHeader;
* import com.nimbusds.jose.JWSSigner;
* import com.nimbusds.jose.JWSVerifier;
* import com.nimbusds.jose.crypto.MACSigner;
* import com.nimbusds.jose.crypto.MACVerifier;
* import com.nimbusds.jwt.JWTClaimsSet;
* import com.nimbusds.jwt.SignedJWT;
* @Service
* public class JwtService {
* public static final String USERNAME = "username";
* public static final String SECRET\_KEY = "11111111111111111111111111111111";
* public static final int EXPIRE\_TIME = 86400000;
* public String generateTokenLogin(String username) {
* String token = null;
* try {
* // Create HMAC signer
* JWSSigner signer = new MACSigner(generateShareSecret());
* JWTClaimsSet.Builder builder = new JWTClaimsSet.Builder();
* builder.claim(USERNAME, username);
* builder.expirationTime(generateExpirationDate());
* JWTClaimsSet claimsSet = builder.build();
* SignedJWT signedJWT = new SignedJWT(new JWSHeader(JWSAlgorithm.HS256), claimsSet);
* // Apply the HMAC protection
* signedJWT.sign(signer);
* // Serialize to compact form, produces something like
* // eyJhbGciOiJIUzI1NiJ9.SGVsbG8sIHdvcmxkIQ.onO9Ihudz3WkiauDO2Uhyuz0Y18UASXlSc1eS0NkWyA
* token = signedJWT.serialize();
* } catch (Exception e) {
* e.printStackTrace();
* }
* return token;
* }
* private JWTClaimsSet getClaimsFromToken(String token) {
* JWTClaimsSet claims = null;
* try {
* SignedJWT signedJWT = SignedJWT.parse(token);
* JWSVerifier verifier = new MACVerifier(generateShareSecret());
* if (signedJWT.verify(verifier)) {
* claims = signedJWT.getJWTClaimsSet();
* }
* } catch (Exception e) {
* e.printStackTrace();
* }
* return claims;
* }
* private Date generateExpirationDate() {
* return new Date(System.currentTimeMillis() + EXPIRE\_TIME);
* }
* private Date getExpirationDateFromToken(String token) {
* Date expiration = null;
* JWTClaimsSet claims = getClaimsFromToken(token);
* expiration = claims.getExpirationTime();
* return expiration;
* }
* public String getUsernameFromToken(String token) {
* String username = null;
* try {
* JWTClaimsSet claims = getClaimsFromToken(token);
* username = claims.getStringClaim(USERNAME);
* } catch (Exception e) {
* e.printStackTrace();
* }
* return username;
* }
* private byte[] generateShareSecret() {
* // Generate 256-bit (32-byte) shared secret
* byte[] sharedSecret = new byte[32];
* sharedSecret = SECRET\_KEY.getBytes();
* return sharedSecret;
* }
* private Boolean isTokenExpired(String token) {
* Date expiration = getExpirationDateFromToken(token);
* return expiration.before(new Date());
* }
* public Boolean validateTokenLogin(String token) {
* if (token == null || token.trim().length() == 0) {
* return false;
* }
* String username = getUsernameFromToken(token);
* if (username == null || username.isEmpty()) {
* return false;
* }
* if (isTokenExpired(token)) {
* return false;
* }
* return true;
* }
* }

Class ***JwtService.java*** dùng để tạo và validate token. (tạo token với thông tin username, thời gian hết hạn; kiểm tra thời gian hết han, chữ ký có hợp lệ không…)

**File cấu hình security:**

* **SecurityConfig.java**
* package stackjava.com.sbjwt.config;
* import org.springframework.context.annotation.Bean;
* import org.springframework.context.annotation.Configuration;
* import org.springframework.http.HttpMethod;
* import org.springframework.security.authentication.AuthenticationManager;
* import org.springframework.security.config.annotation.web.builders.HttpSecurity;
* import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
* import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
* import org.springframework.security.config.http.SessionCreationPolicy;
* import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
* import stackjava.com.sbjwt.rest.CustomAccessDeniedHandler;
* import stackjava.com.sbjwt.rest.JwtAuthenticationTokenFilter;
* import stackjava.com.sbjwt.rest.RestAuthenticationEntryPoint;
* @Configuration
* @EnableWebSecurity
* public class SecurityConfig extends WebSecurityConfigurerAdapter {
* @Bean
* public JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter() throws Exception {
* JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter = new JwtAuthenticationTokenFilter();
* jwtAuthenticationTokenFilter.setAuthenticationManager(authenticationManager());
* return jwtAuthenticationTokenFilter;
* }
* @Bean
* public RestAuthenticationEntryPoint restServicesEntryPoint() {
* return new RestAuthenticationEntryPoint();
* }
* @Bean
* public CustomAccessDeniedHandler customAccessDeniedHandler() {
* return new CustomAccessDeniedHandler();
* }
* @Bean
* @Override
* protected AuthenticationManager authenticationManager() throws Exception {
* return super.authenticationManager();
* }
* protected void configure(HttpSecurity http) throws Exception {
* // Disable crsf cho đường dẫn /rest/\*\*
* http.csrf().ignoringAntMatchers("/rest/\*\*");
* http.authorizeRequests().antMatchers("/rest/login\*\*").permitAll();
* http.antMatcher("/rest/\*\*").httpBasic().authenticationEntryPoint(restServicesEntryPoint()).and()
* .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and().authorizeRequests()
* .antMatchers(HttpMethod.GET, "/rest/\*\*").access("hasRole('ROLE\_ADMIN') or hasRole('ROLE\_USER')")
* .antMatchers(HttpMethod.POST, "/rest/\*\*").access("hasRole('ROLE\_ADMIN')")
* .antMatchers(HttpMethod.DELETE, "/rest/\*\*").access("hasRole('ROLE\_ADMIN')").and()
* .addFilterBefore(jwtAuthenticationTokenFilter(), UsernamePasswordAuthenticationFilter.class)
* .exceptionHandling().accessDeniedHandler(customAccessDeniedHandler());
* }
* }

Các url /rest/\*\* với method GET (API lấy thông tin user) cho phép cả role ADMIN và USER truy cập, với các method “DELETE” và “POST” (xóa và tạo mới user) thì chỉ cho phép role ADMIN truy cập.

httpBasic().authenticationEntryPoint(restServicesEntryPoint()): bean restServicesEntryPoint sẽ xử lý những request chưa được xác thực.

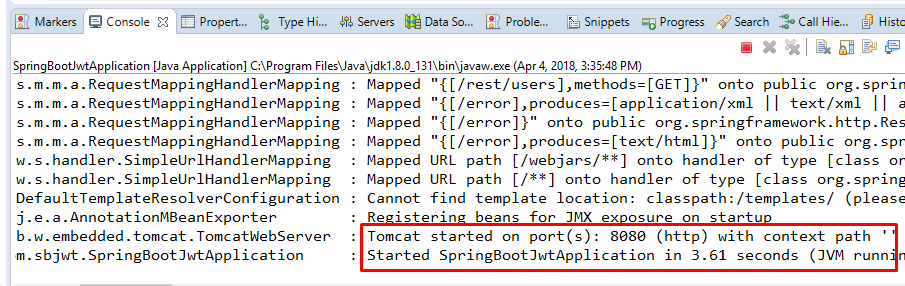
addFilterBefore(jwtAuthenticationTokenFilter(), UsernamePasswordAuthenticationFilter.class) : bean jwtAuthenticationTokenFilter sẽ thực hiện việc xác thực người dùng

exceptionHandling().accessDeniedHandler(customAccessDeniedHandler()): trường hợp người dùng gửi request mà không có quyền sẽ do bean customAccessDeniedHandlerxử lý (Ví dụ role USER nhưng gửi request xóa user)

**Các class filter, handler security:**

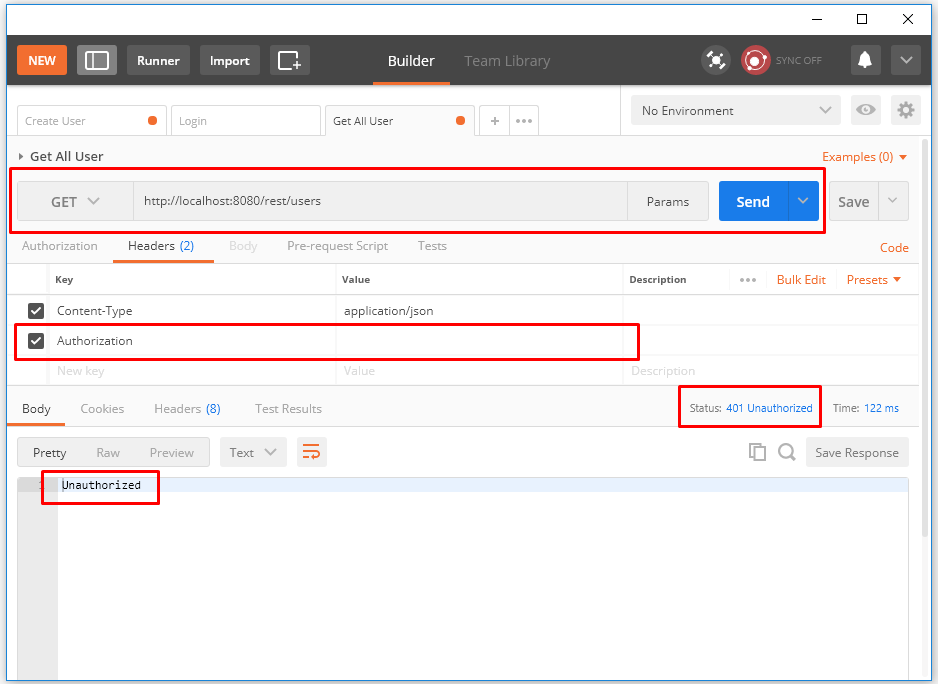
* **CustomAccessDeniedHandler.java**
* package stackjava.com.sbjwt.rest;
* import java.io.IOException;
* import javax.servlet.ServletException;
* import javax.servlet.http.HttpServletRequest;
* import javax.servlet.http.HttpServletResponse;
* import org.springframework.security.access.AccessDeniedException;
* import org.springframework.security.web.access.AccessDeniedHandler;
* public class CustomAccessDeniedHandler implements AccessDeniedHandler {
* @Override
* public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedException exc)
* throws IOException, ServletException {
* // Authentication auth = SecurityContextHolder.getContext().getAuthentication();
* response.setStatus(HttpServletResponse.SC\_FORBIDDEN);
* response.getWriter().write("Access Denied!");
* }
* }
* **JwtAuthenticationTokenFilter.java**
* package stackjava.com.sbjwt.rest;
* 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.beans.factory.annotation.Autowired;
* import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
* import org.springframework.security.core.context.SecurityContextHolder;
* import org.springframework.security.core.userdetails.User;
* import org.springframework.security.core.userdetails.UserDetails;
* import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
* import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
* import stackjava.com.sbjwt.service.JwtService;
* import stackjava.com.sbjwt.service.UserService;
* public class JwtAuthenticationTokenFilter extends UsernamePasswordAuthenticationFilter {
* private final static String TOKEN\_HEADER = "authorization";
* @Autowired
* private JwtService jwtService;
* @Autowired
* private UserService userService;
* @Override
* public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
* throws IOException, ServletException {
* HttpServletRequest httpRequest = (HttpServletRequest) request;
* String authToken = httpRequest.getHeader(TOKEN\_HEADER);
* if (jwtService.validateTokenLogin(authToken)) {
* String username = jwtService.getUsernameFromToken(authToken);
* stackjava.com.sbjwt.entities.User user = userService.loadUserByUsername(username);
* if (user != null) {
* boolean enabled = true;
* boolean accountNonExpired = true;
* boolean credentialsNonExpired = true;
* boolean accountNonLocked = true;
* UserDetails userDetail = new User(username, user.getPassword(), enabled, accountNonExpired,
* credentialsNonExpired, accountNonLocked, user.getAuthorities());
* UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetail,
* null, userDetail.getAuthorities());
* authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(httpRequest));
* SecurityContextHolder.getContext().setAuthentication(authentication);
* }
* }
* chain.doFilter(request, response);
* }
* }
* **RestAuthenticationEntryPoint.java**
* package stackjava.com.sbjwt.rest;
* import java.io.IOException;
* import javax.servlet.http.HttpServletRequest;
* import javax.servlet.http.HttpServletResponse;
* import org.springframework.security.core.AuthenticationException;
* import org.springframework.security.web.AuthenticationEntryPoint;
* public final class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {
* public void commence(HttpServletRequest request, HttpServletResponse response,
* AuthenticationException authException) throws IOException {
* // response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Unauthorized");
* response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);
* response.getWriter().write("Unauthorized");
* }
* }

**Demo:**

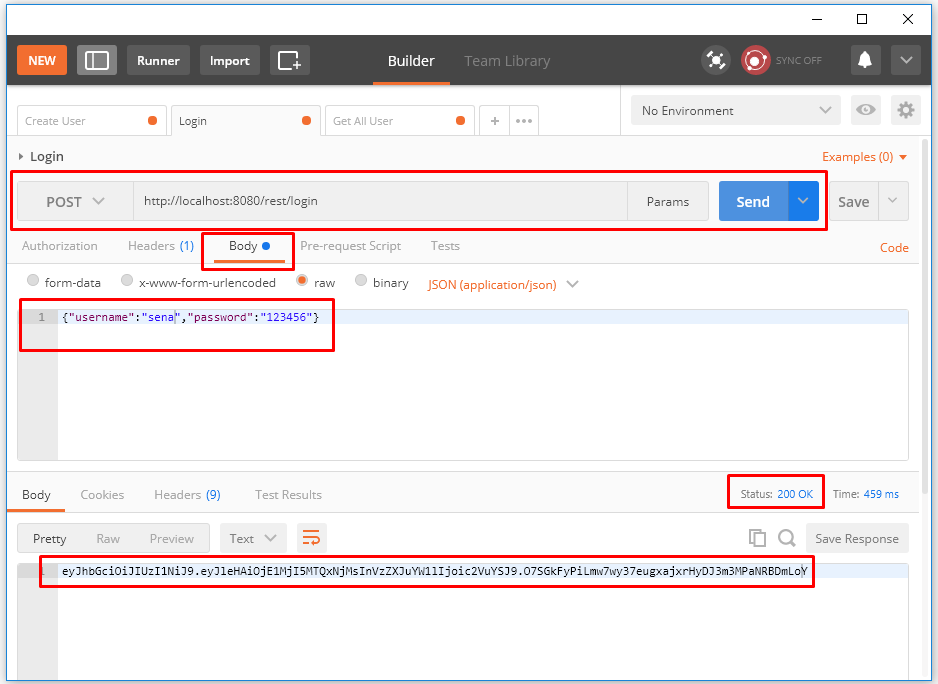
[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-6.png)

Với các URL Restful Web Service mình sẽ sử dụng [postman](https://stackjava.com/install/postman-la-gi-cai-dat-gui-request-api-voi-postman.html) extension trên chrome để gửi các request thêm, xóa, lấy thông tin đối tượng user.

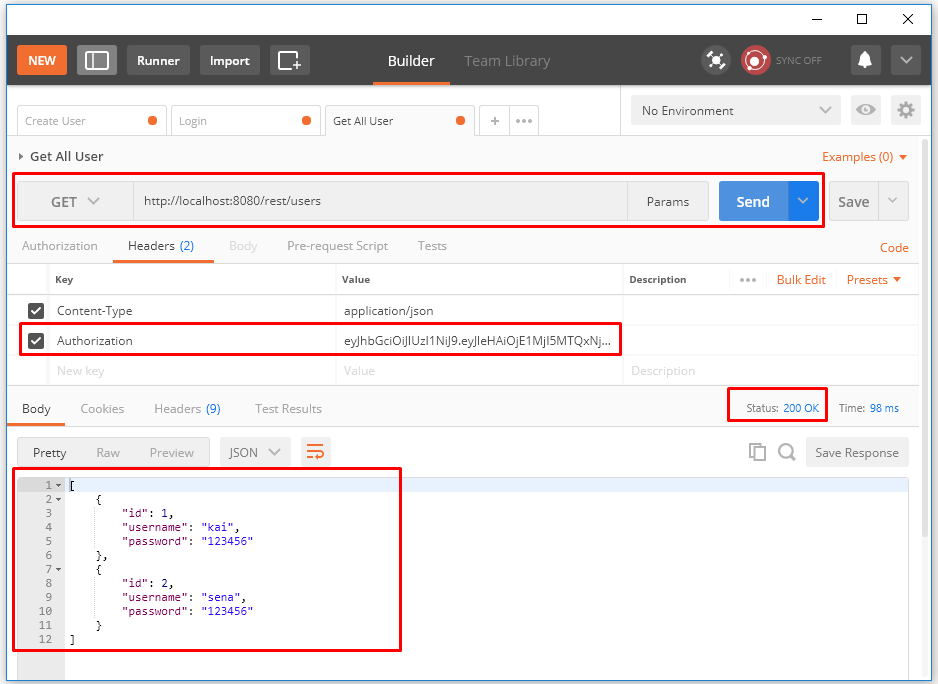
**Trường hợp gửi request lấy thông tin tất cả user mà không có token đính kèm (chưa đăng nhập) hoặc token đã hết hạn**

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-7.png)

Login với tài khoản sena/123456

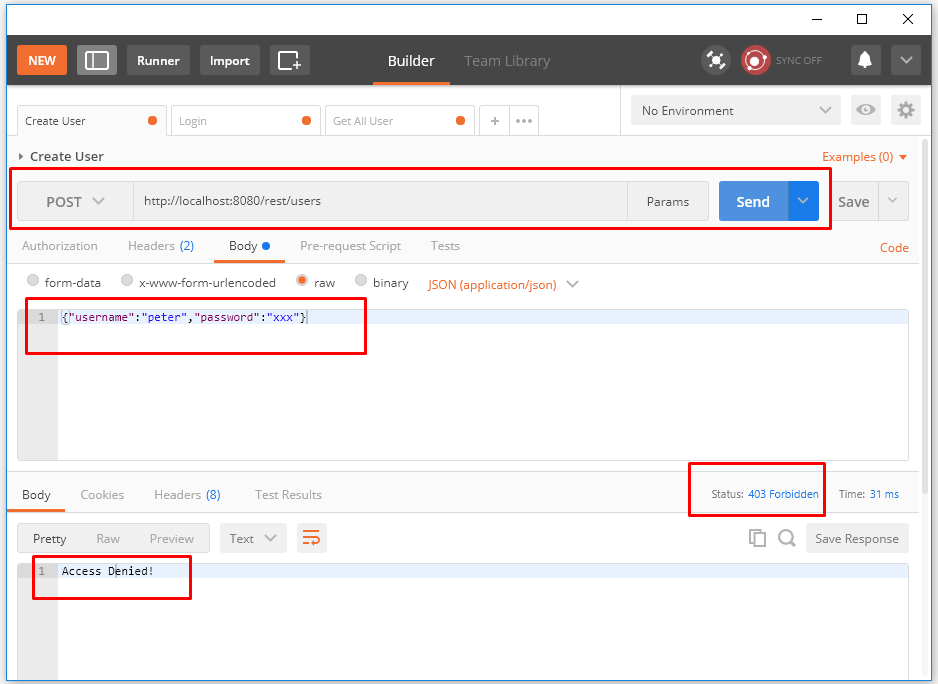
[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-8.png)

Đính kèm token vừa nhận được vào phần header rồi gửi request lấy thông tin tất cả user:

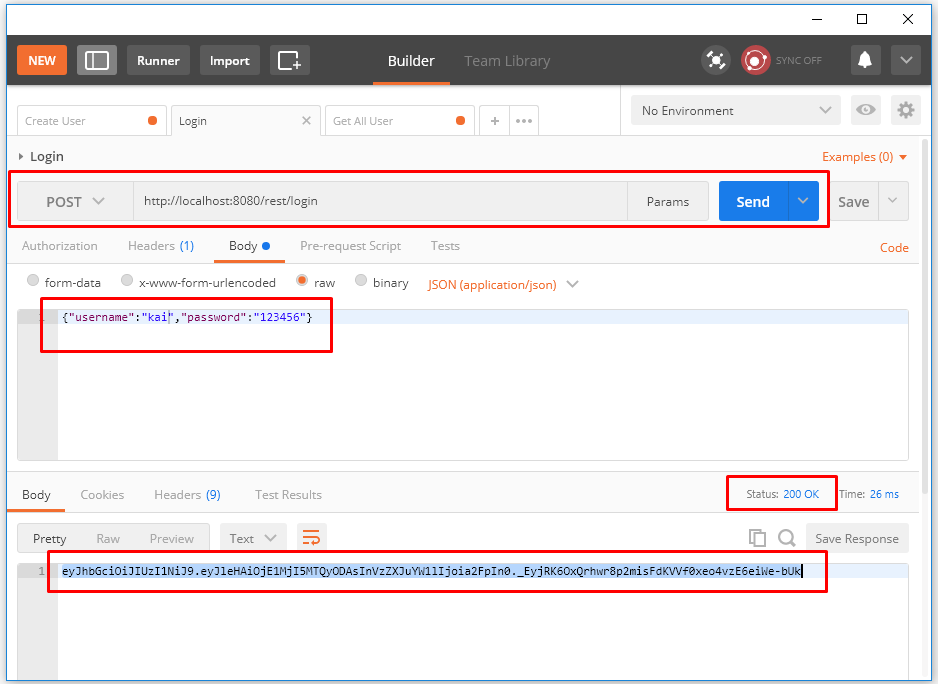
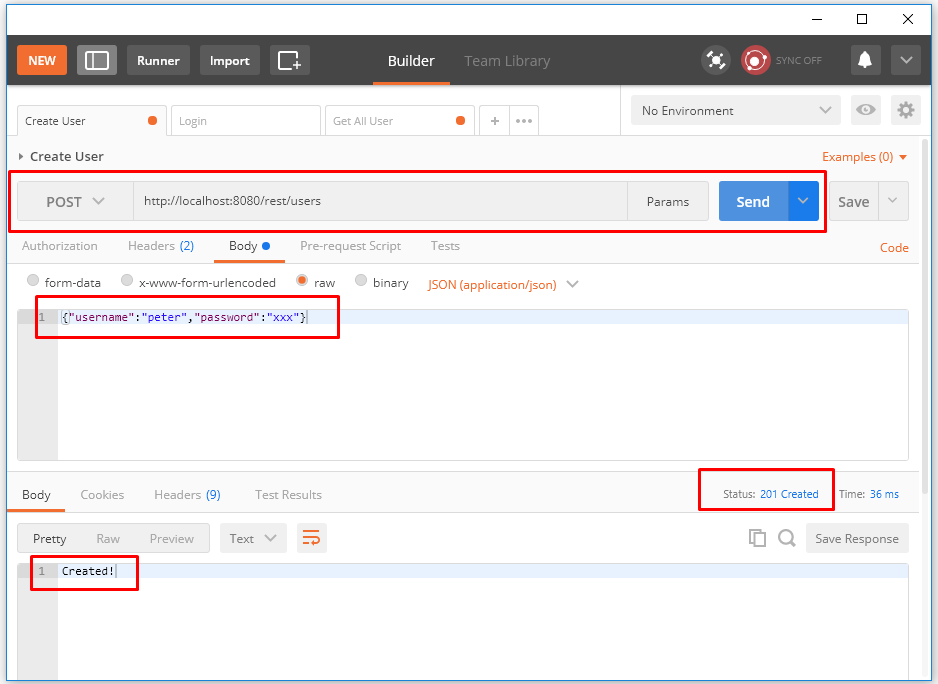
[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-9.png)

Gửi request tạo mới đối tượng user:

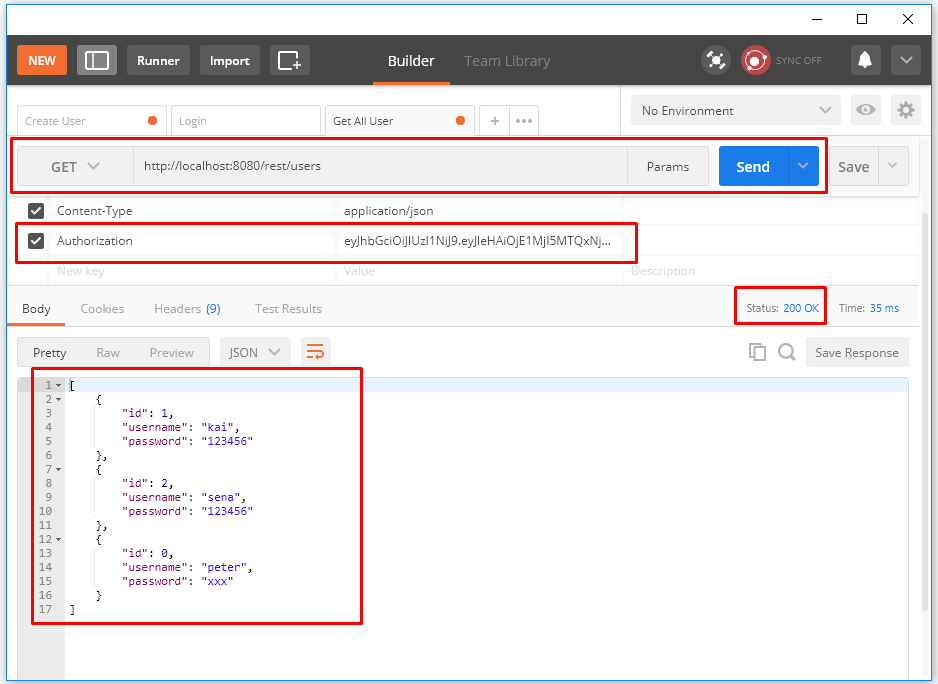
Vì tài khoản sena/123456 chỉ có role USER nên không thể truy cập được url /rest/users với method POST.

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-10.png)

Đăng nhập lại với tài khoản kai/123456 và thực hiện thêm mới user:

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-11.png) [](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-12.png)

Gửi lại request lấy danh sách tất cả các user ta sẽ thấy có thêm user “peter”

[](https://stackjava.com/wp-content/uploads/2018/04/spring-boot-jwt-13.png)