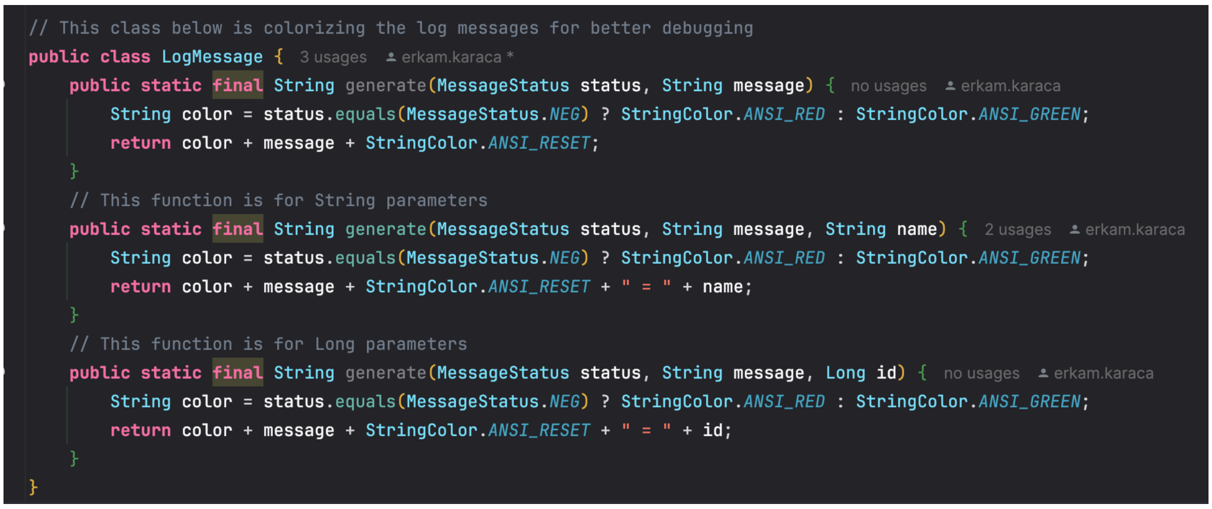
Final Project Development Journal

(This document is a journal of the development process that I made day by day.)

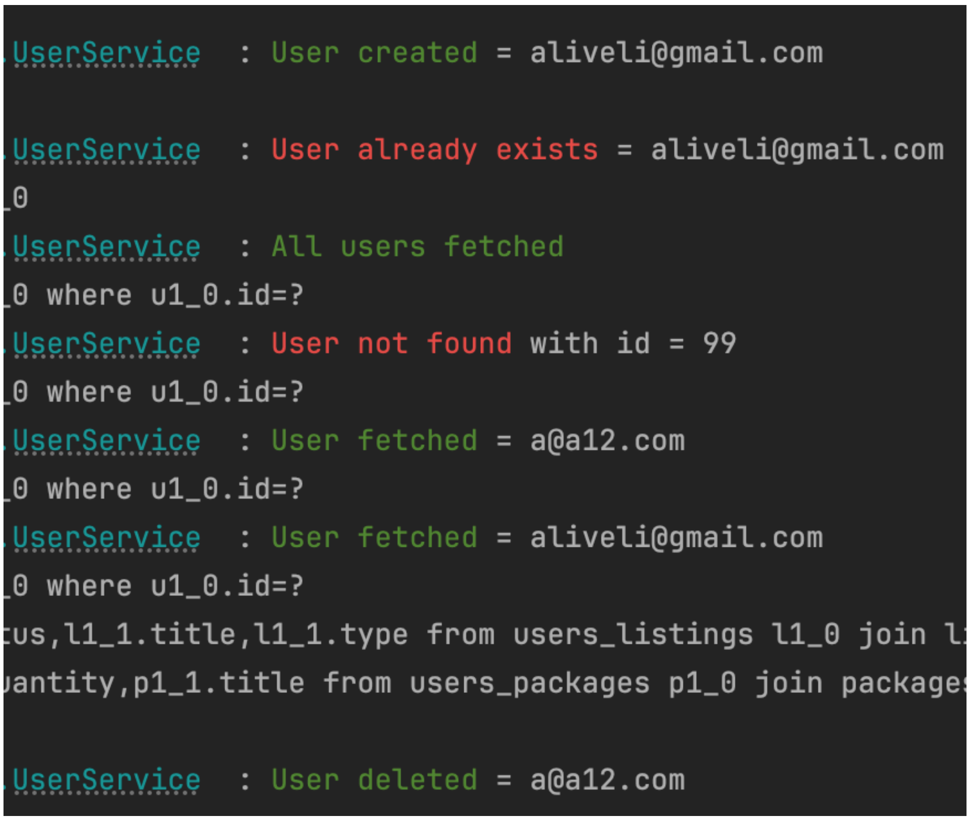
First Week

04.07.2024

LogMessage class implemented for logging and debugging purposes. This class is colorizing the errors and exceptions as red, and success messages or info as green.

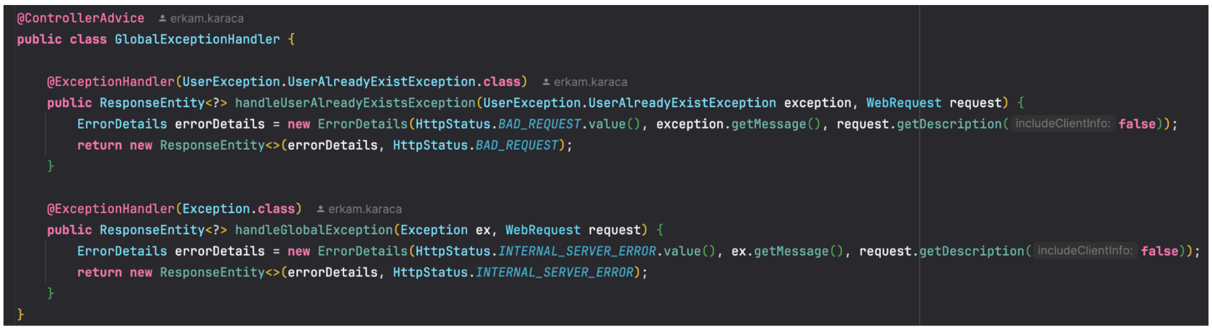


```
// This class below is colorizing the log messages for better debugging
public class LogMessage {  3 usages  erkam.karaca *
    public static final String generate(MessageStatus status, String message) {  no usages  erkam.karaca
        String color = status.equals(MessageStatus.NEG) ? StringColor.ANSI_RED : StringColor.ANSI_GREEN;
        return color + message + StringColor.ANSI_RESET;
    }
    // This function is for String parameters
    public static final String generate(MessageStatus status, String message, String name) {  2 usages  erkam.karaca
        String color = status.equals(MessageStatus.NEG) ? StringColor.ANSI_RED : StringColor.ANSI_GREEN;
        return color + message + StringColor.ANSI_RESET + " = " + name;
    }
    // This function is for Long parameters
    public static final String generate(MessageStatus status, String message, Long id) {  no usages  erkam.karaca
        String color = status.equals(MessageStatus.NEG) ? StringColor.ANSI_RED : StringColor.ANSI_GREEN;
        return color + message + StringColor.ANSI_RESET + " = " + id;
    }
}
```



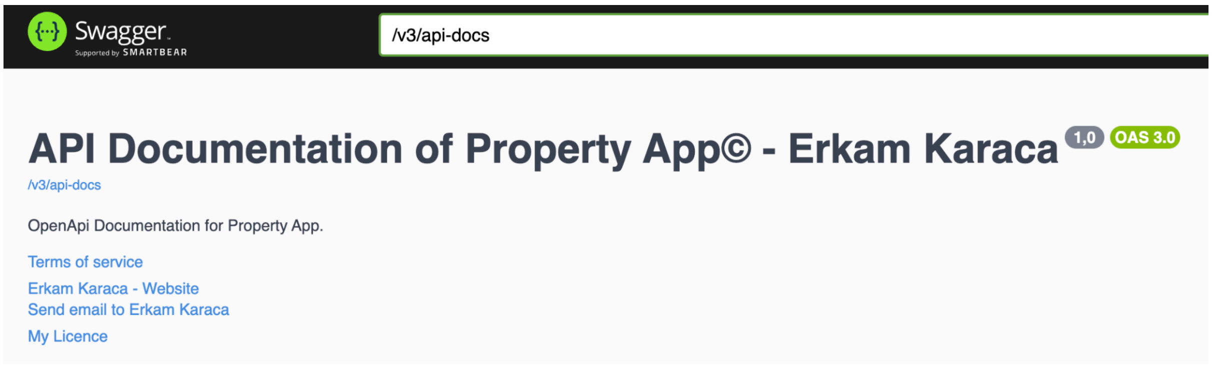
```
UserService   : User created = aliveli@gmail.com

UserService   : User already exists = aliveli@gmail.com
0
UserService   : All users fetched
0 where u1_0.id=?
UserService   : User not found with id = 99
0 where u1_0.id=?
UserService   : User fetched = a@a12.com
0 where u1_0.id=?
UserService   : User fetched = aliveli@gmail.com
0 where u1_0.id=?
us,l1_1.title,l1_1.type from users_listings l1_0 join l
antity,p1_1.title from users_packages p1_0 join package

UserService   : User deleted = a@a12.com
```

Global Exception Handler and Error Details classes implemented.



```
@ControllerAdvice  erkam.karaca
public class GlobalExceptionHandler {

    @ExceptionHandler(UserException.UserAlreadyExistException.class)  erkam.karaca
    public ResponseEntity<?> handleUserAlreadyExistsException(UserException.UserAlreadyExistException exception, WebRequest request) {
        ErrorDetails errorDetails = new ErrorDetails(HttpStatus.BAD_REQUEST.value(), exception.getMessage(), request.getDescription( includeClientInfo: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(Exception.class)  erkam.karaca
    public ResponseEntity<?> handleGlobalException(Exception ex, WebRequest request) {
        ErrorDetails errorDetails = new ErrorDetails(HttpStatus.INTERNAL_SERVER_ERROR.value(), ex.getMessage(), request.getDescription( includeClientInfo: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

SwaggerUI implemented.



API Documentation of Property App© - Erkam Karaca 1,0 OAS 3.0

/v3/api-docs

OpenApi Documentation for Property App.

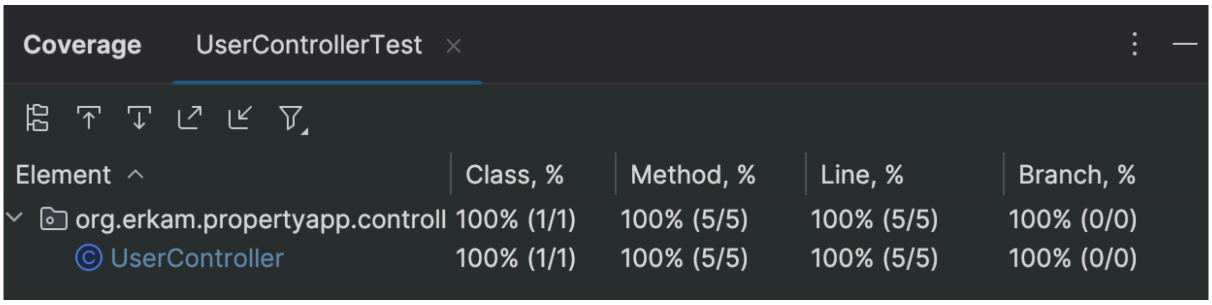
Terms of service

Erkam Karaca - Website

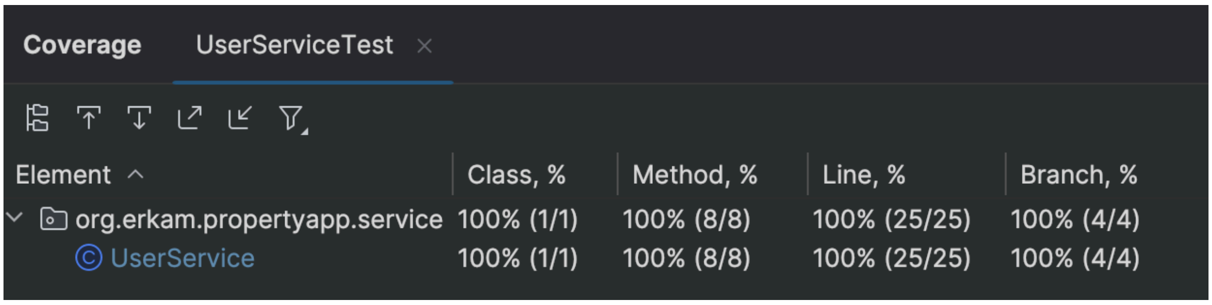
Send email to Erkam Karaca

My Licence

Unit tests of User Service and User Controller added at the beginnings of the project.



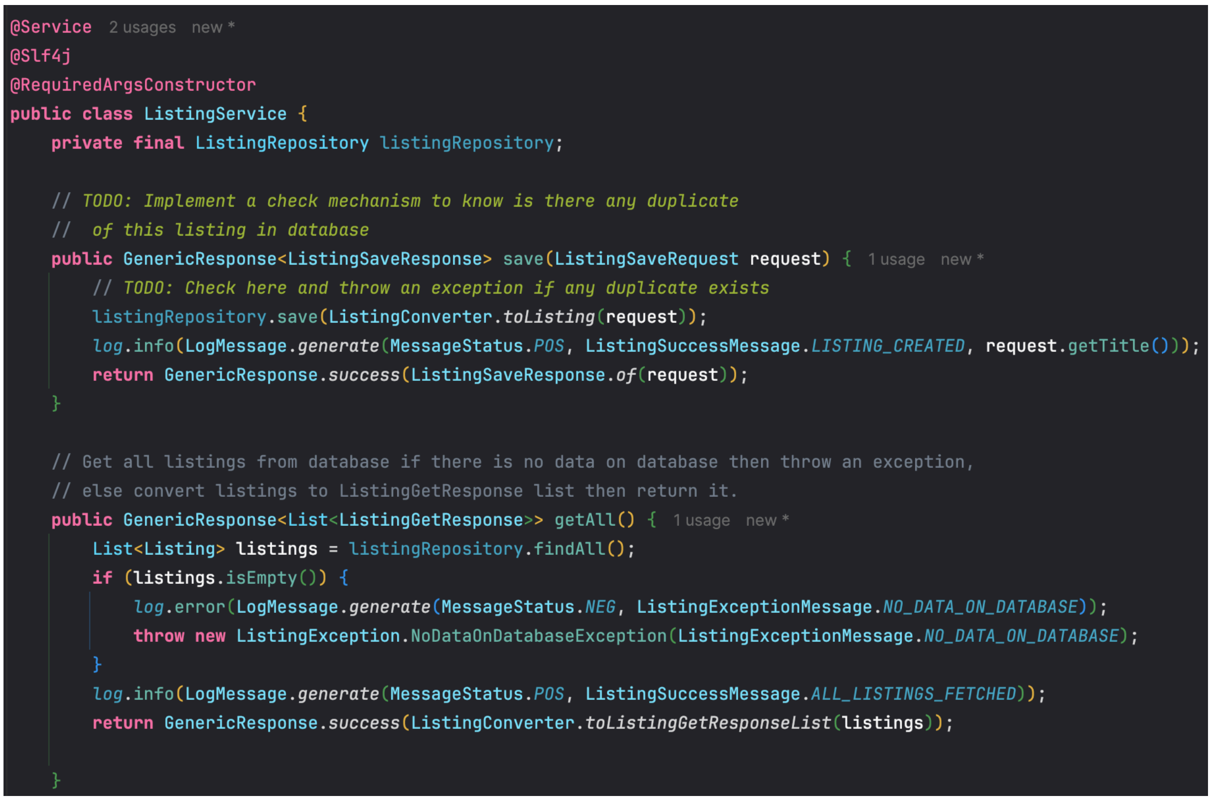
Element	Class, %	Method, %	Line, %	Branch, %
org.erkam.propertyapp.controll	100% (1/1)	100% (5/5)	100% (5/5)	100% (0/0)
UserController	100% (1/1)	100% (5/5)	100% (5/5)	100% (0/0)



Element	Class, %	Method, %	Line, %	Branch, %
org.erkam.propertyapp.service	100% (1/1)	100% (8/8)	100% (25/25)	100% (4/4)
UserService	100% (1/1)	100% (8/8)	100% (25/25)	100% (4/4)

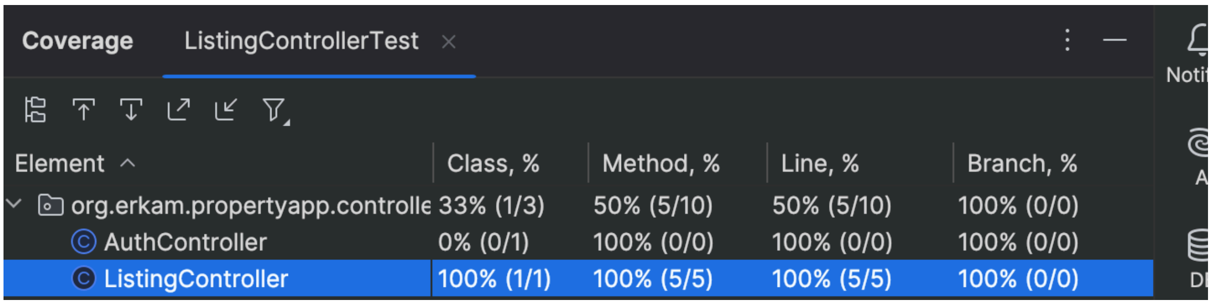
07.07.2024

Listing controller, service and repository added.

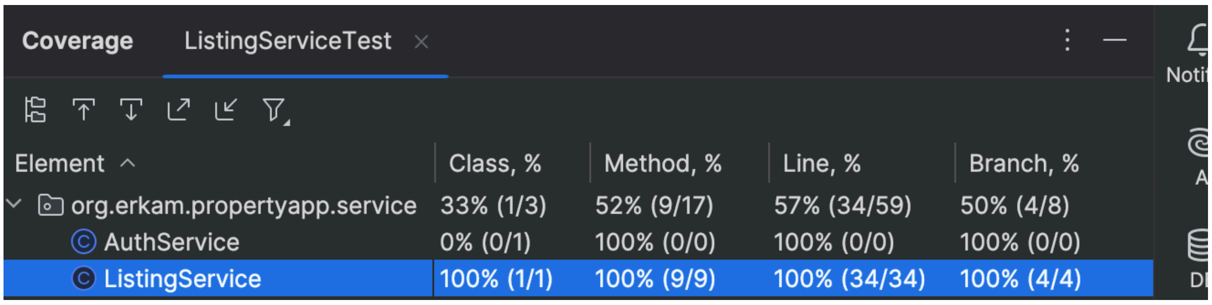


```java
@Service  2 usages  new *
@Slf4j
@RequiredArgsConstructor
public class ListingService {
    private final ListingRepository listingRepository;

    // TODO: Implement a check mechanism to know is there any duplicate
    //  of this listing in database
    public GenericResponse<ListingSaveResponse> save(ListingSaveRequest request) {  1 usage  new *
        // TODO: Check here and throw an exception if any duplicate exists
        listingRepository.save(ListingConverter.toListing(request));
        log.info(LogMessage.generate(MessageStatus.POS, ListingSuccessMessage.LISTING_CREATED, request.getTitle()));
        return GenericResponse.success(ListingSaveResponse.of(request));
    }

    // Get all listings from database if there is no data on database then throw an exception,
    // else convert listings to ListingGetResponse list then return it.
    public GenericResponse<List<ListingGetResponse>> getAll() {  1 usage  new *
        List<Listing> listings = listingRepository.findAll();
        if (listings.isEmpty()) {
            log.error(LogMessage.generate(MessageStatus.NEG, ListingExceptionMessage.NO_DATA_ON_DATABASE));
            throw new ListingException.NoDataOnDatabaseException(ListingExceptionMessage.NO_DATA_ON_DATABASE);
        }
        log.info(LogMessage.generate(MessageStatus.POS, ListingSuccessMessage.ALL_LISTINGS_FETCHED));
        return GenericResponse.success(ListingConverter.toListingGetResponseList(listings));

    }
```

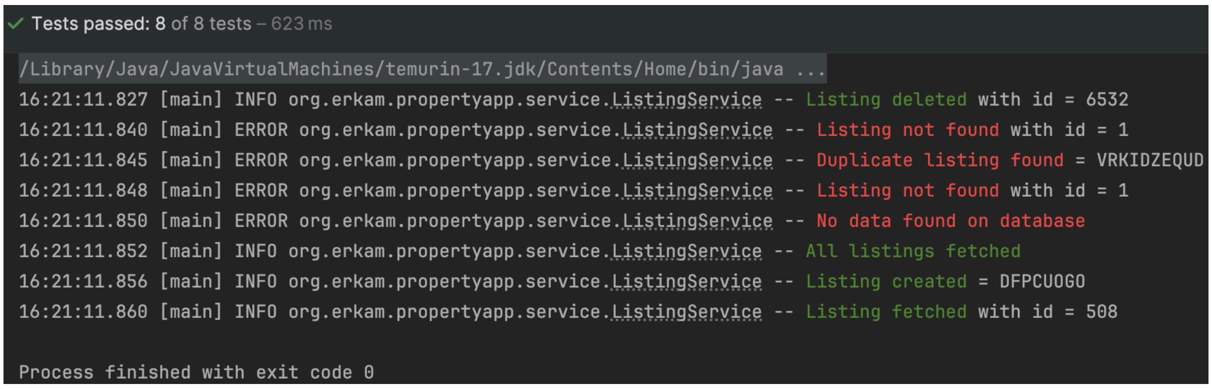
Listing controller and service unit tests added.



Element	Class, %	Method, %	Line, %	Branch, %
org.erkam.propertyapp.controlle	33% (1/3)	50% (5/10)	50% (5/10)	100% (0/0)
AuthController	0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
ListingController	100% (1/1)	100% (5/5)	100% (5/5)	100% (0/0)

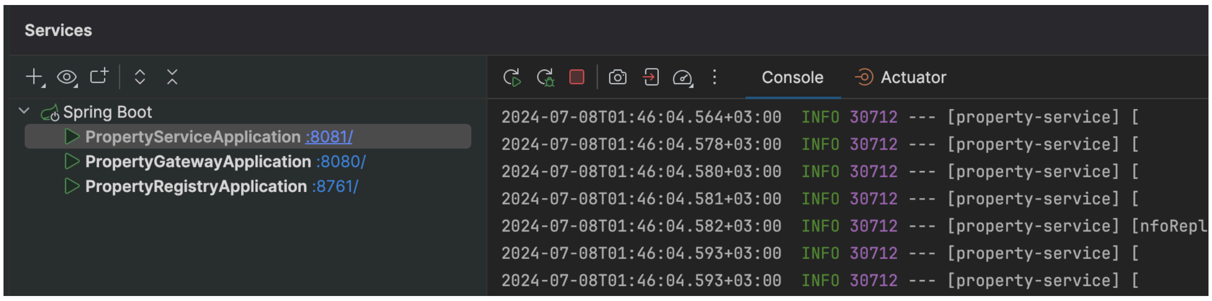


Element	Class, %	Method, %	Line, %	Branch, %
org.erkam.propertyapp.service	33% (1/3)	52% (9/17)	57% (34/59)	50% (4/8)
AuthService	0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
ListingService	100% (1/1)	100% (9/9)	100% (34/34)	100% (4/4)



```
✓ Tests passed: 8 of 8 tests – 623 ms
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java ...
16:21:11.827 [main] INFO org.erkam.propertyapp.service.ListingService -- Listing deleted with id = 6532
16:21:11.840 [main] ERROR org.erkam.propertyapp.service.ListingService -- Listing not found with id = 1
16:21:11.845 [main] ERROR org.erkam.propertyapp.service.ListingService -- Duplicate listing found = VRKIDZEQUD
16:21:11.848 [main] ERROR org.erkam.propertyapp.service.ListingService -- Listing not found with id = 1
16:21:11.850 [main] ERROR org.erkam.propertyapp.service.ListingService -- No data found on database
16:21:11.852 [main] INFO org.erkam.propertyapp.service.ListingService -- All listings fetched
16:21:11.856 [main] INFO org.erkam.propertyapp.service.ListingService -- Listing created = DFPCUOGO
16:21:11.860 [main] INFO org.erkam.propertyapp.service.ListingService -- Listing fetched with id = 508

Process finished with exit code 0
```

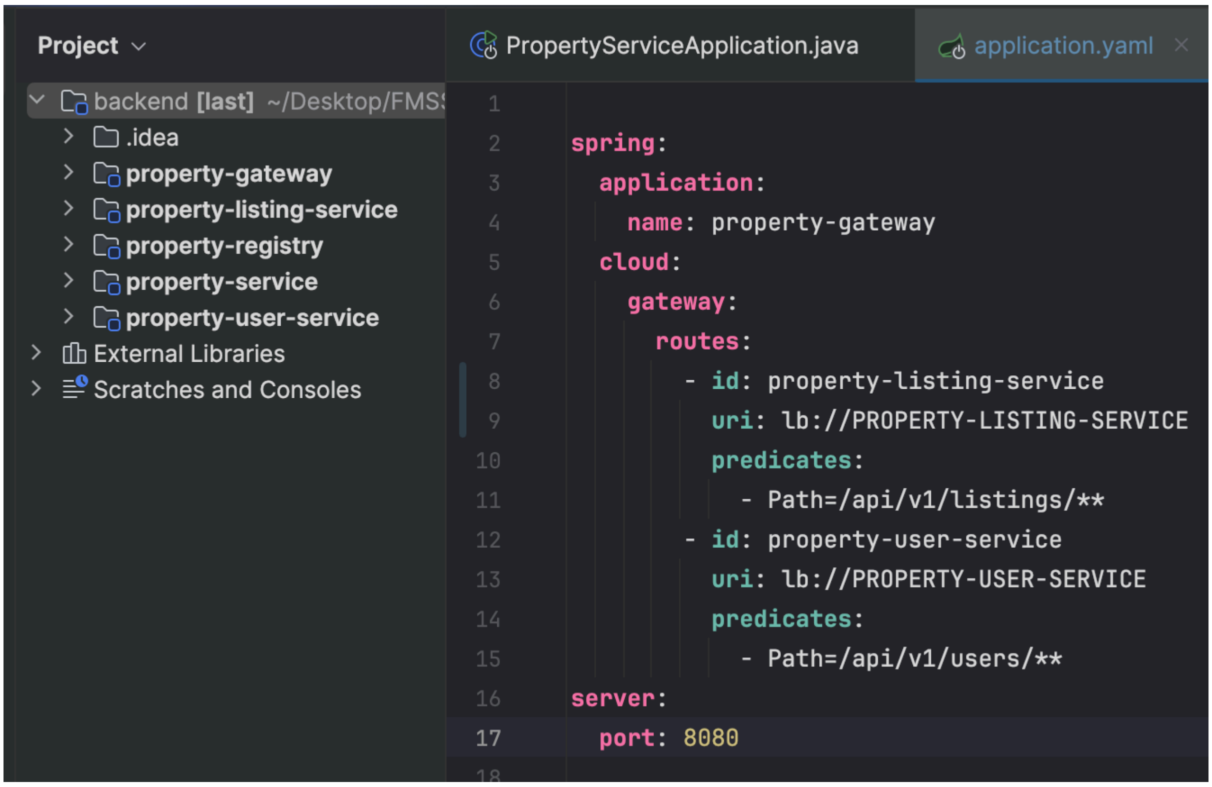
Gateway and Registry added.



```
Services
Spring Boot
  PropertyServiceApplication :8081/
  PropertyGatewayApplication :8080/
  PropertyRegistryApplication :8761/

Console    Actuator
2024-07-08T01:46:04.564+03:00  INFO 30712 --- [property-service] [
2024-07-08T01:46:04.578+03:00  INFO 30712 --- [property-service] [
2024-07-08T01:46:04.580+03:00  INFO 30712 --- [property-service] [
2024-07-08T01:46:04.581+03:00  INFO 30712 --- [property-service] [
2024-07-08T01:46:04.582+03:00  INFO 30712 --- [property-service] [nfoRepl
2024-07-08T01:46:04.593+03:00  INFO 30712 --- [property-service] [
2024-07-08T01:46:04.593+03:00  INFO 30712 --- [property-service] [
```

08.07.2024

Listing Service and User Service turned into microservices.



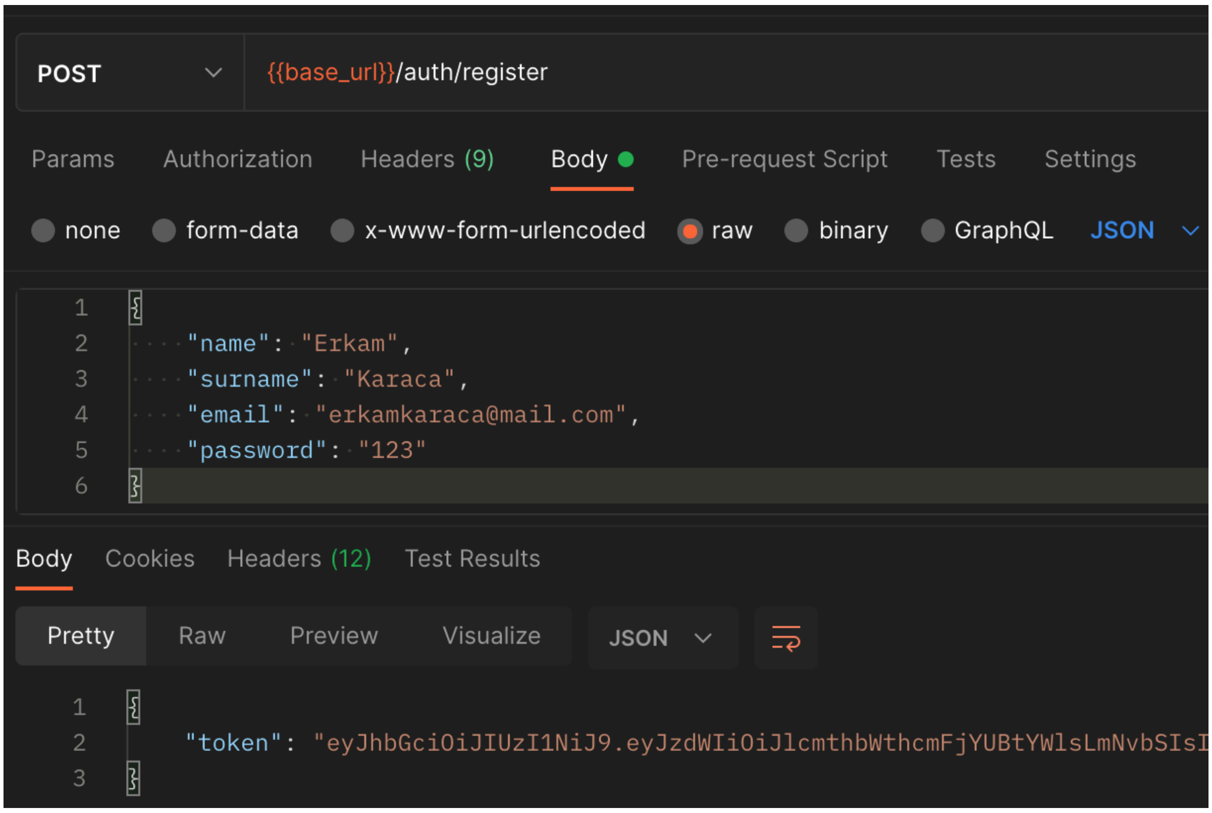
```
Project
backend [last] ~/Desktop/FMS
  .idea
  property-gateway
  property-listing-service
  property-registry
  property-service
  property-user-service
External Libraries
Scratches and Consoles
```

PropertyServiceApplication.java   application.yaml

```yaml
spring:
  application:
    name: property-gateway
  cloud:
    gateway:
      routes:
        - id: property-listing-service
          uri: lb://PROPERTY-LISTING-SERVICE
          predicates:
            - Path=/api/v1/listings/**
        - id: property-user-service
          uri: lb://PROPERTY-USER-SERVICE
          predicates:
            - Path=/api/v1/users/**
server:
  port: 8080
```

09.07.2024

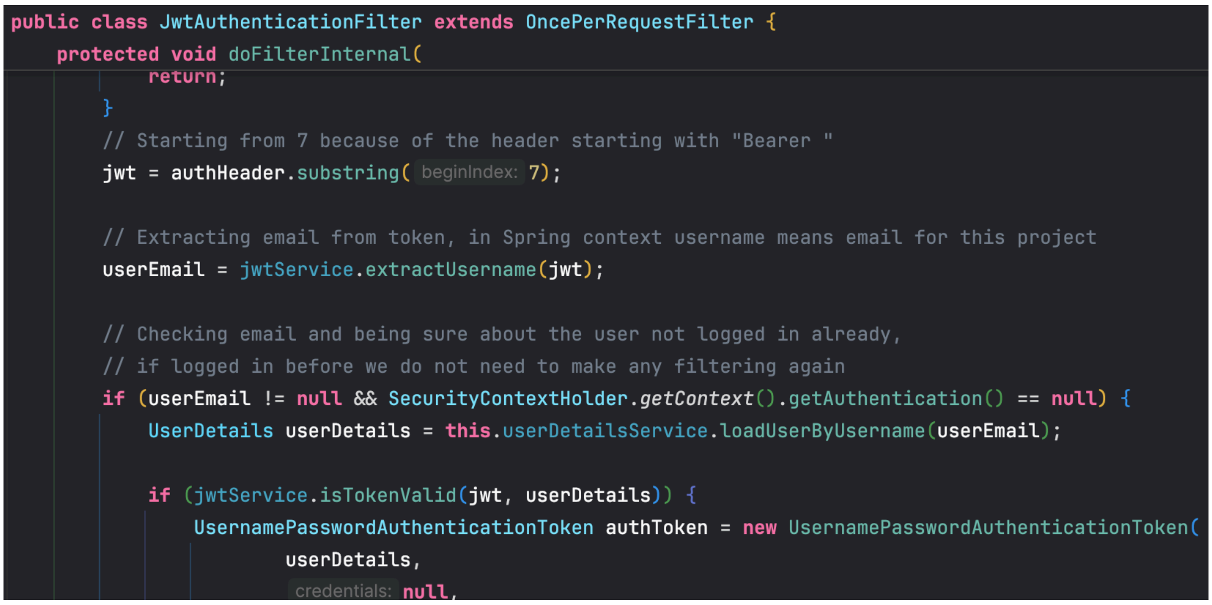
JWT Authentication implemented. Auth Controller and Auth Service added.



{{base_url}}/auth/register

```
{
    "name": "Erkam",
    "surname": "Karaca",
    "email": "erkamkaraca@mail.com",
    "password": "123"
}
```

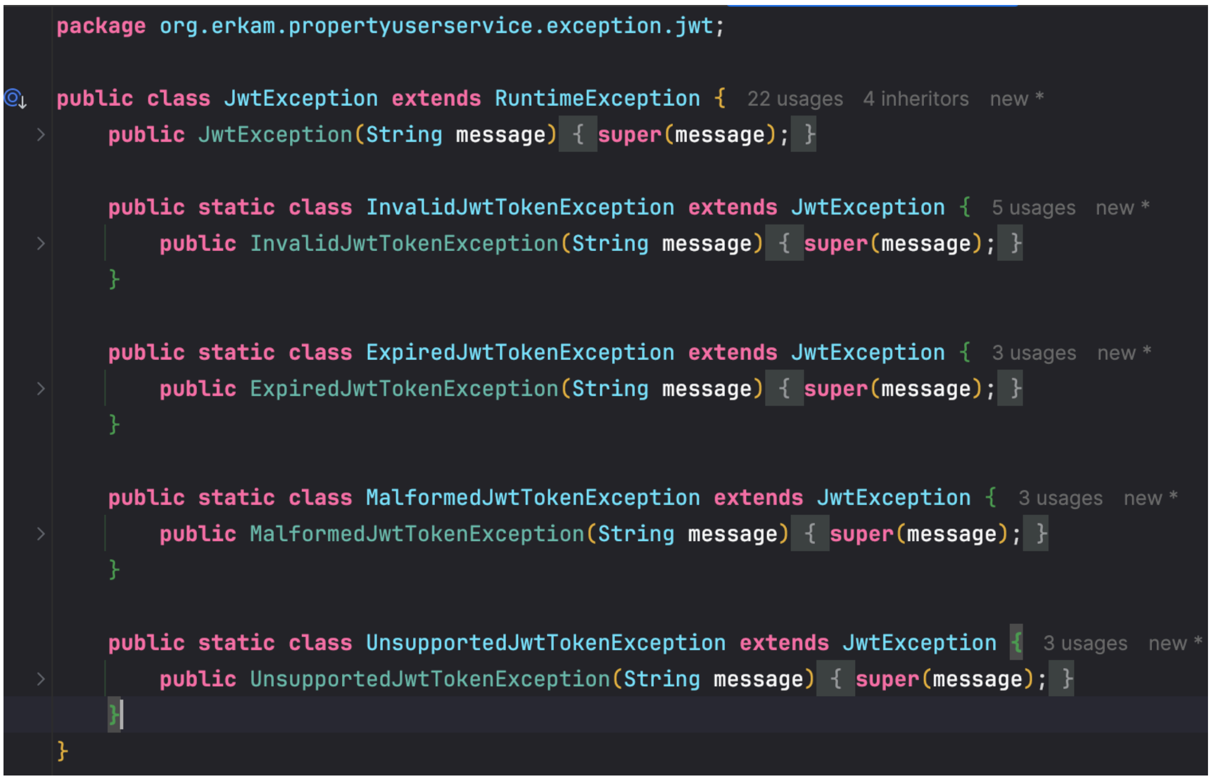
```
{
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlcmthbWthcmFjYUBtYWlsLmNvbSIsI
}
```



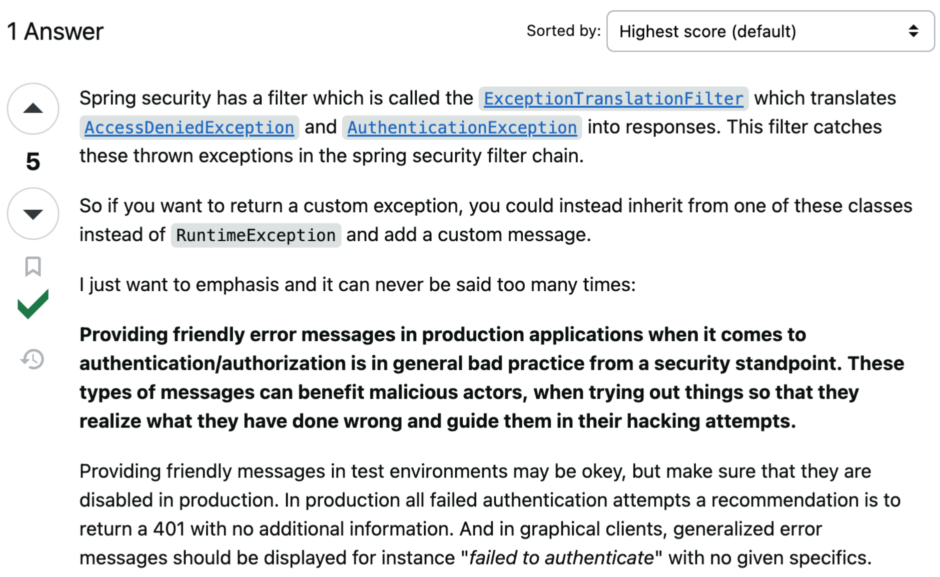
```
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    protected void doFilterInternal(
            return;
        }
        // Starting from 7 because of the header starting with "Bearer "
        jwt = authHeader.substring( beginIndex: 7);

        // Extracting email from token, in Spring context username means email for this project
        userEmail = jwtService.extractUsername(jwt);

        // Checking email and being sure about the user not logged in already,
        // if logged in before we do not need to make any filtering again
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);

            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails,
                        credentials: null,
```

10.07.2024

I have implemented a feign client of ListingService in UserService, and I thought about returning meaningful response messages to the client for example I thought to return "JWT is expired" or "JWT is invalid" or "JWT signature is malformed" but while I was researching I saw a stackoverflow ticket like below, it was saying that "In security related issues, it is not a secure thing to give specific information to clients in responses." then I decided to send a response only like "You must login first" and 401 Unauthorized, 403 Forbidden. But I still throwing those custom exceptions below only at the development environment for debugging purposes



```java
package org.erkam.propertyuserservice.exception.jwt;

public class JwtException extends RuntimeException {  22 usages   4 inheritors   new *
    public JwtException(String message) { super(message); }

    public static class InvalidJwtTokenException extends JwtException {  5 usages   new *
        public InvalidJwtTokenException(String message) { super(message); }
    }

    public static class ExpiredJwtTokenException extends JwtException {  3 usages   new *
        public ExpiredJwtTokenException(String message) { super(message); }
    }

    public static class MalformedJwtTokenException extends JwtException {  3 usages   new *
        public MalformedJwtTokenException(String message) { super(message); }
    }

    public static class UnsupportedJwtTokenException extends JwtException {  3 usages   new *
        public UnsupportedJwtTokenException(String message) { super(message); }
    }
}
```


1 Answer

Sorted by: Highest score (default)

5

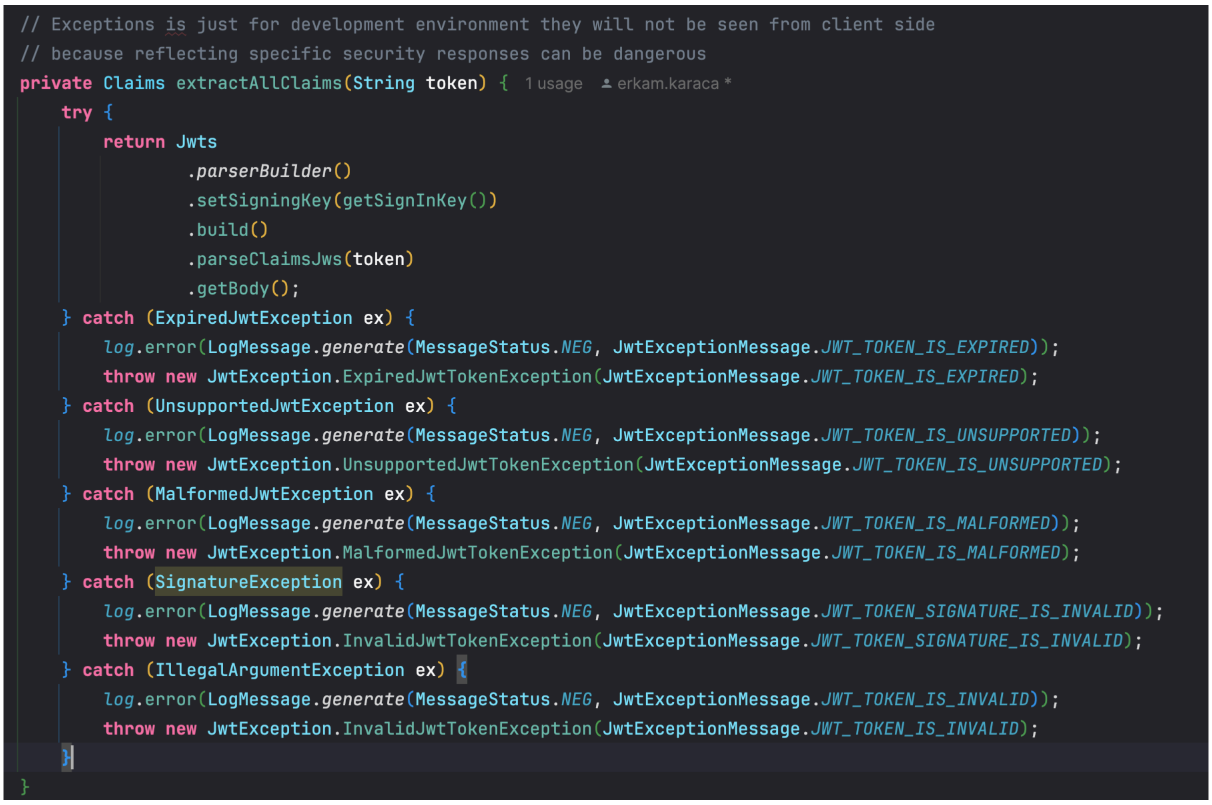
Spring security has a filter which is called the `ExceptionTranslationFilter` which translates `AccessDeniedException` and `AuthenticationException` into responses. This filter catches these thrown exceptions in the spring security filter chain.

So if you want to return a custom exception, you could instead inherit from one of these classes instead of `RuntimeException` and add a custom message.

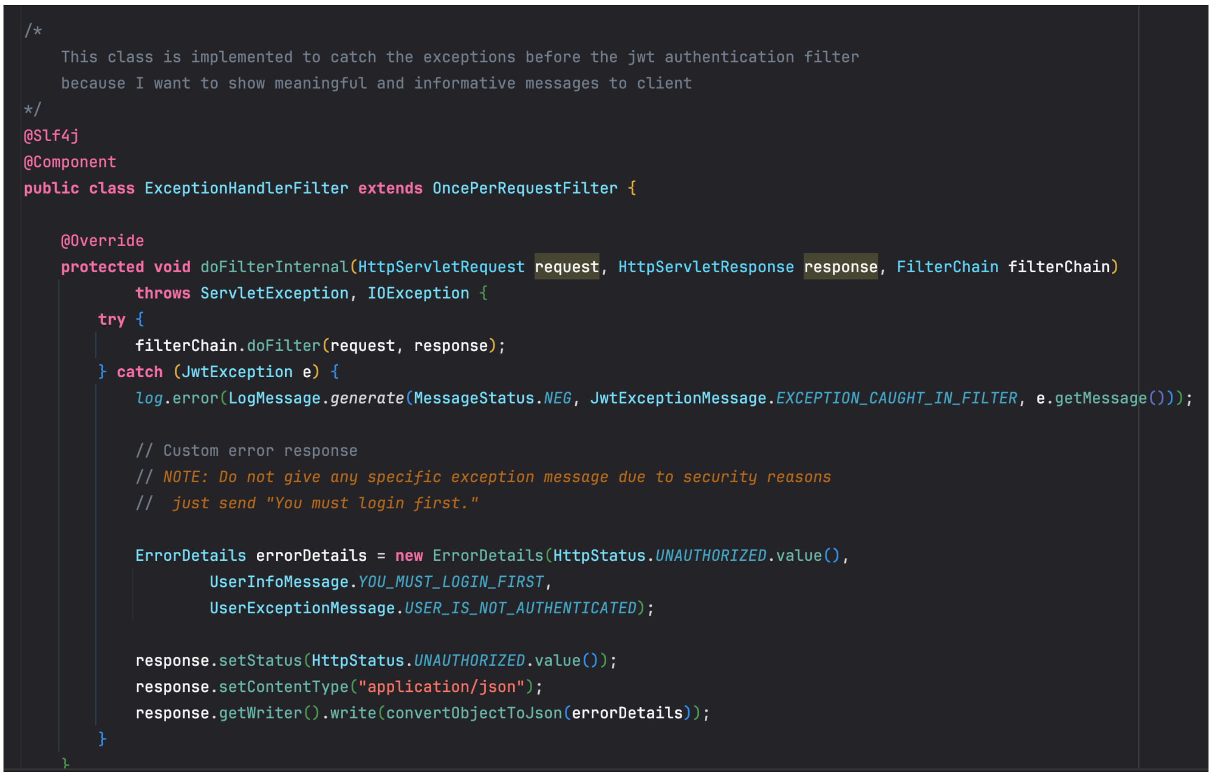
I just want to emphasis and it can never be said too many times:

Providing friendly error messages in production applications when it comes to authentication/authorization is in general bad practice from a security standpoint. These types of messages can benefit malicious actors, when trying out things so that they realize what they have done wrong and guide them in their hacking attempts.

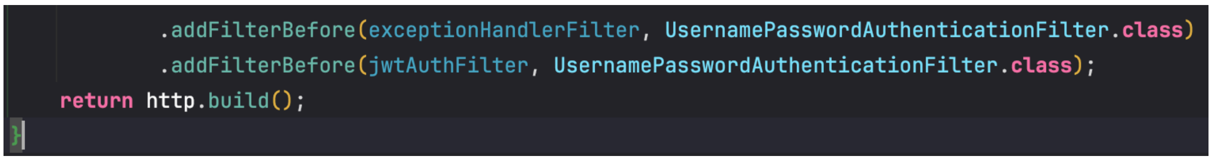
Providing friendly messages in test environments may be okey, but make sure that they are disabled in production. In production all failed authentication attempts a recommendation is to return a 401 with no additional information. And in graphical clients, generalized error messages should be displayed for instance "*failed to authenticate*" with no given specifics.


```java
// Exceptions is just for development environment they will not be seen from client side
// because reflecting specific security responses can be dangerous
private Claims extractAllClaims(String token) {  1 usage  erkam.karaca *
    try {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    } catch (ExpiredJwtException ex) {
        log.error(LogMessage.generate(MessageStatus.NEG, JwtExceptionMessage.JWT_TOKEN_IS_EXPIRED));
        throw new JwtException.ExpiredJwtTokenException(JwtExceptionMessage.JWT_TOKEN_IS_EXPIRED);
    } catch (UnsupportedJwtException ex) {
        log.error(LogMessage.generate(MessageStatus.NEG, JwtExceptionMessage.JWT_TOKEN_IS_UNSUPPORTED));
        throw new JwtException.UnsupportedJwtTokenException(JwtExceptionMessage.JWT_TOKEN_IS_UNSUPPORTED);
    } catch (MalformedJwtException ex) {
        log.error(LogMessage.generate(MessageStatus.NEG, JwtExceptionMessage.JWT_TOKEN_IS_MALFORMED));
        throw new JwtException.MalformedJwtTokenException(JwtExceptionMessage.JWT_TOKEN_IS_MALFORMED);
    } catch (SignatureException ex) {
        log.error(LogMessage.generate(MessageStatus.NEG, JwtExceptionMessage.JWT_TOKEN_SIGNATURE_IS_INVALID));
        throw new JwtException.InvalidJwtTokenException(JwtExceptionMessage.JWT_TOKEN_SIGNATURE_IS_INVALID);
    } catch (IllegalArgumentException ex) {
        log.error(LogMessage.generate(MessageStatus.NEG, JwtExceptionMessage.JWT_TOKEN_IS_INVALID));
        throw new JwtException.InvalidJwtTokenException(JwtExceptionMessage.JWT_TOKEN_IS_INVALID);
    }
}
```

When I was debugging I realized that JWT related exceptions were being handled before my Global Exception Handler. I researched it and I found that they were being handled in the JWT Auth Filter. I did some more research and I created an Exception Handler Filter to handle the JWT Exceptions before the JWT Auth Filter, and placed it to the Security Filter Chain before JWT Auth Filter.



```java
/*
    This class is implemented to catch the exceptions before the jwt authentication filter
    because I want to show meaningful and informative messages to client
*/
@Slf4j
@Component
public class ExceptionHandlerFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        try {
            filterChain.doFilter(request, response);
        } catch (JwtException e) {
            log.error(LogMessage.generate(MessageStatus.NEG, JwtExceptionMessage.EXCEPTION_CAUGHT_IN_FILTER, e.getMessage()));

            // Custom error response
            // NOTE: Do not give any specific exception message due to security reasons
            //  just send "You must login first."

            ErrorDetails errorDetails = new ErrorDetails(HttpStatus.UNAUTHORIZED.value(),
                    UserInfoMessage.YOU_MUST_LOGIN_FIRST,
                    UserExceptionMessage.USER_IS_NOT_AUTHENTICATED);

            response.setStatus(HttpStatus.UNAUTHORIZED.value());
            response.setContentType("application/json");
            response.getWriter().write(convertObjectToJson(errorDetails));
        }
```



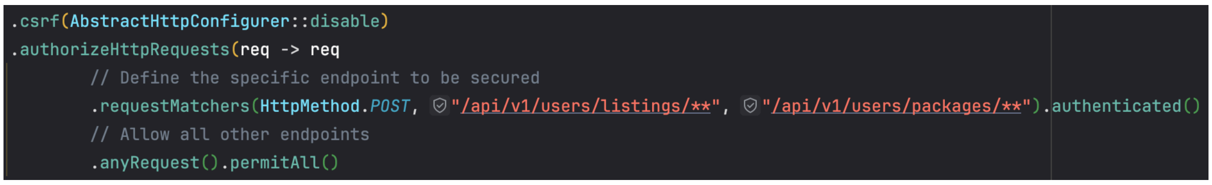
```java
                .addFilterBefore(exceptionHandlerFilter, UsernamePasswordAuthenticationFilter.class)
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
}
```

<u>*11.07.2024*</u>

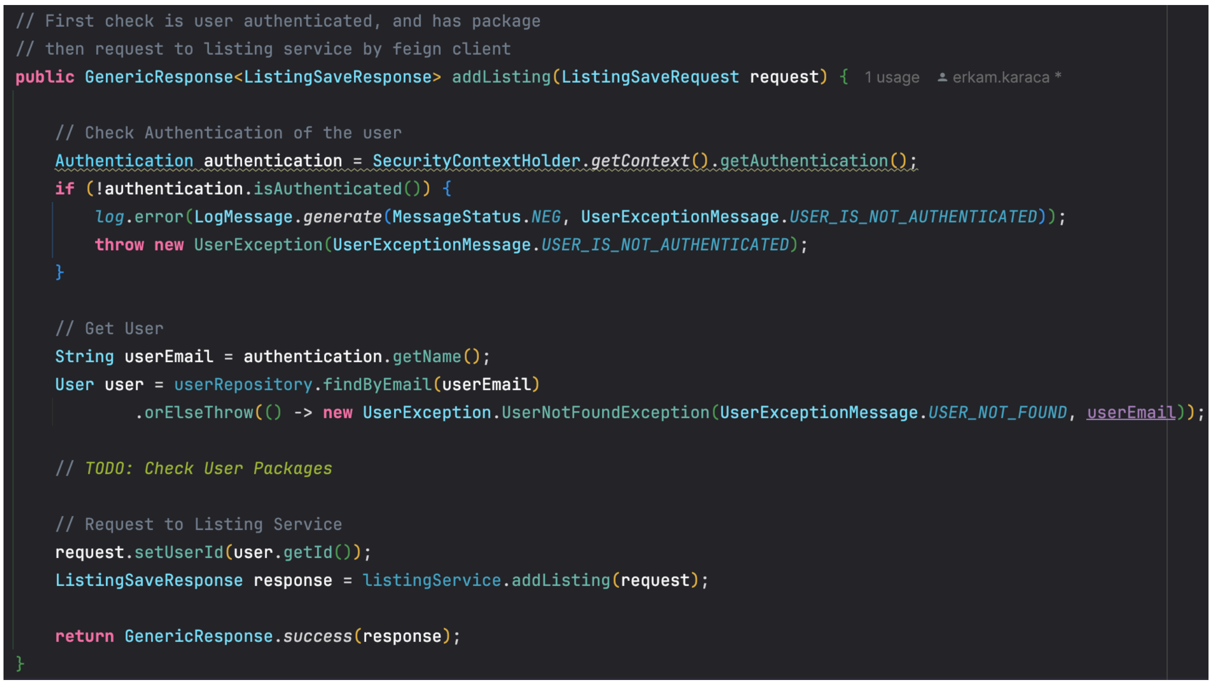
I have implemented a Payment Service to receive payments, but this is a mock service for now I am using its endpoint from User Service, if I have enough time I will make it real.

I implemented methods to purchase a package and add listings to the User Service. I have used Feign Client again for the interaction with the Listing Service.

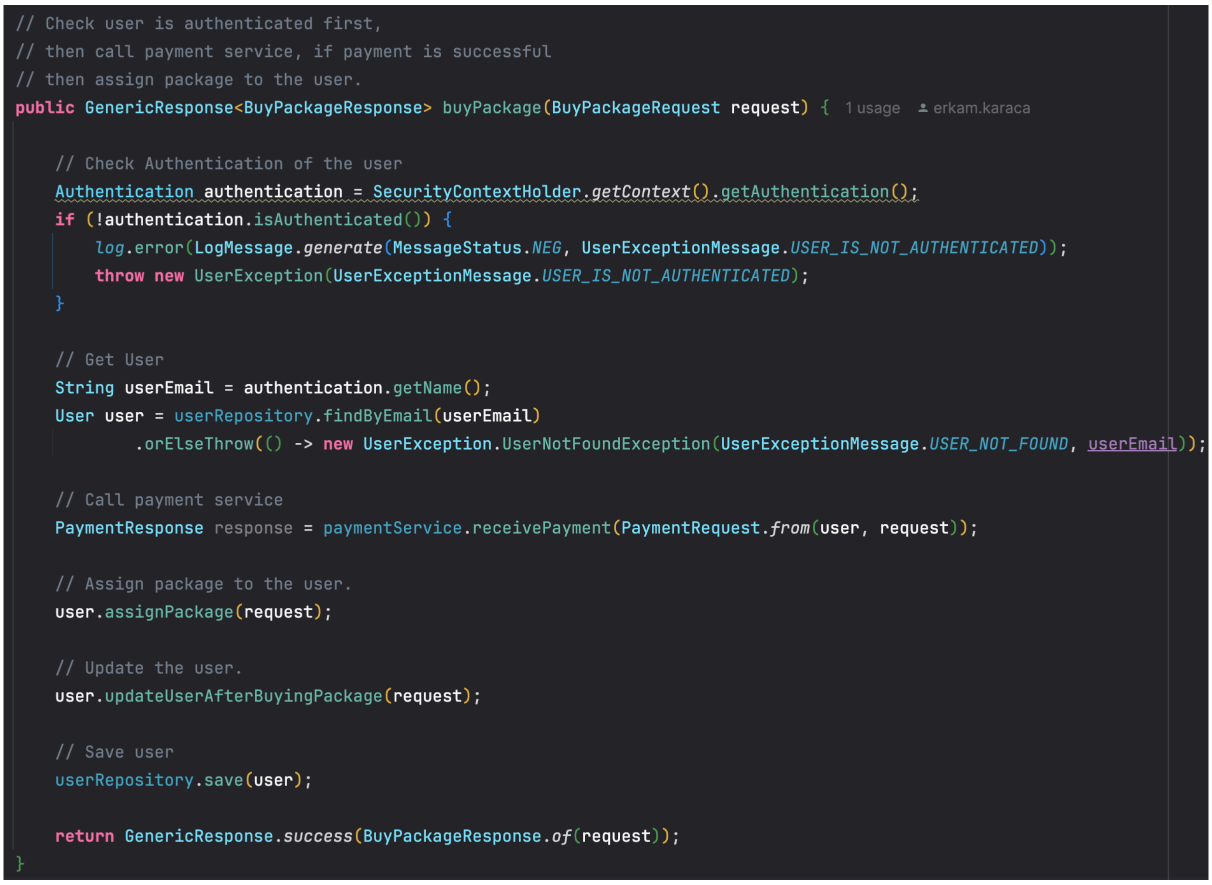
By the way I secured just two endpoints which are for purchasing a package and adding listings.



```java
.csrf(AbstractHttpConfigurer::disable)
.authorizeHttpRequests(req -> req
        // Define the specific endpoint to be secured
        .requestMatchers(HttpMethod.POST, "/api/v1/users/listings/**", "/api/v1/users/packages/**").authenticated()
        // Allow all other endpoints
        .anyRequest().permitAll()
```

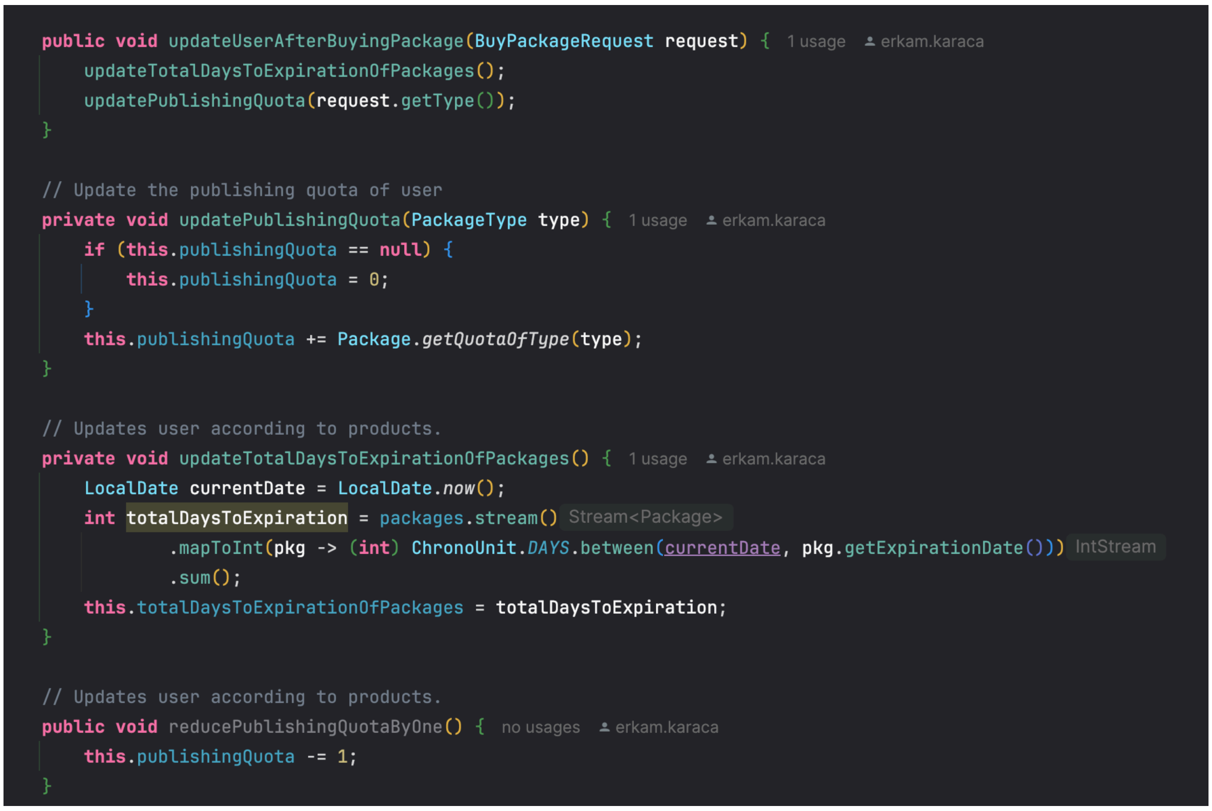


```java
// First check is user authenticated, and has package
// then request to listing service by feign client
public GenericResponse<ListingSaveResponse> addListing(ListingSaveRequest request) {  1 usage  erkam.karaca *

    // Check Authentication of the user
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (!authentication.isAuthenticated()) {
        log.error(LogMessage.generate(MessageStatus.NEG, UserExceptionMessage.USER_IS_NOT_AUTHENTICATED));
        throw new UserException(UserExceptionMessage.USER_IS_NOT_AUTHENTICATED);
    }

    // Get User
    String userEmail = authentication.getName();
    User user = userRepository.findByEmail(userEmail)
            .orElseThrow(() -> new UserException.UserNotFoundException(UserExceptionMessage.USER_NOT_FOUND, userEmail));

    // TODO: Check User Packages

    // Request to Listing Service
    request.setUserId(user.getId());
    ListingSaveResponse response = listingService.addListing(request);

    return GenericResponse.success(response);
}
```

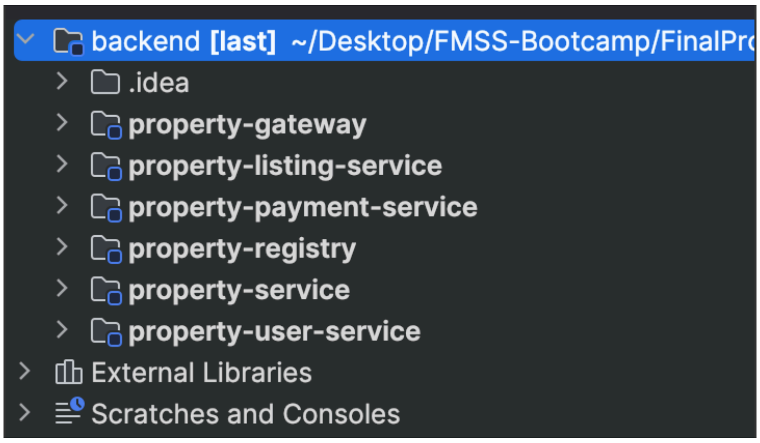


```java
// Check user is authenticated first,
// then call payment service, if payment is successful
// then assign package to the user.
public GenericResponse<BuyPackageResponse> buyPackage(BuyPackageRequest request) {  1 usage  erkam.karaca

    // Check Authentication of the user
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (!authentication.isAuthenticated()) {
        log.error(LogMessage.generate(MessageStatus.NEG, UserExceptionMessage.USER_IS_NOT_AUTHENTICATED));
        throw new UserException(UserExceptionMessage.USER_IS_NOT_AUTHENTICATED);
    }

    // Get User
    String userEmail = authentication.getName();
    User user = userRepository.findByEmail(userEmail)
            .orElseThrow(() -> new UserException.UserNotFoundException(UserExceptionMessage.USER_NOT_FOUND, userEmail));

    // Call payment service
    PaymentResponse response = paymentService.receivePayment(PaymentRequest.from(user, request));

    // Assign package to the user.
    user.assignPackage(request);

    // Update the user.
    user.updateUserAfterBuyingPackage(request);

    // Save user
    userRepository.save(user);

    return GenericResponse.success(BuyPackageResponse.of(request));
}
```

I update users quota to publish listing and expiration duration for users after buying a package. I will implement the reducing quota after adding a listing functionality too.



```
public void updateUserAfterBuyingPackage(BuyPackageRequest request) {  1 usage  erkam.karaca
    updateTotalDaysToExpirationOfPackages();
    updatePublishingQuota(request.getType());
}

// Update the publishing quota of user
private void updatePublishingQuota(PackageType type) {  1 usage  erkam.karaca
    if (this.publishingQuota == null) {
        this.publishingQuota = 0;
    }
    this.publishingQuota += Package.getQuotaOfType(type);
}

// Updates user according to products.
private void updateTotalDaysToExpirationOfPackages() {  1 usage  erkam.karaca
    LocalDate currentDate = LocalDate.now();
    int totalDaysToExpiration = packages.stream() Stream<Package>
            .mapToInt(pkg -> (int) ChronoUnit.DAYS.between(currentDate, pkg.getExpirationDate())) IntStream
            .sum();
    this.totalDaysToExpirationOfPackages = totalDaysToExpiration;
}

// Updates user according to products.
public void reducePublishingQuotaByOne() {  no usages  erkam.karaca
    this.publishingQuota -= 1;
}
```

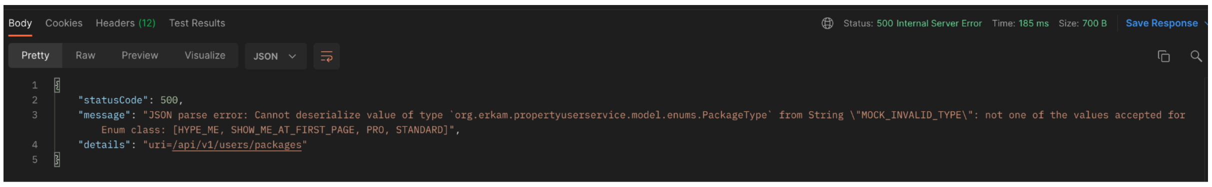


```
backend [last]  ~/Desktop/FMSS-Bootcamp/FinalPr
    .idea
    property-gateway
    property-listing-service
    property-payment-service
    property-registry
    property-service
    property-user-service
  External Libraries
  Scratches and Consoles
```

12.07.2024

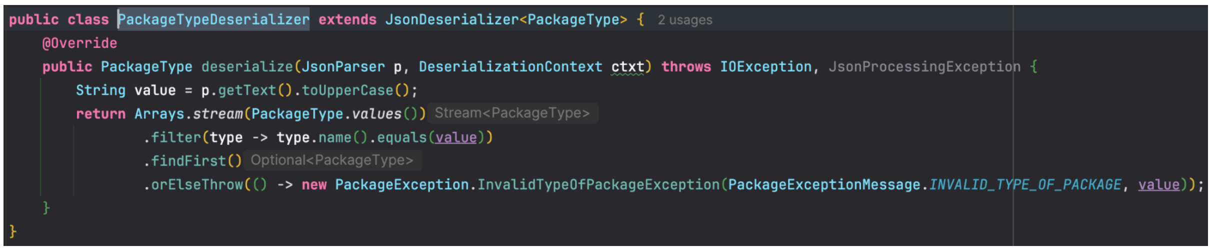
When I tried to buy a package with invalid type, I was getting a message like below, I wanted to show a meaningful message to the client, so I implemented my custom deserializer to throw my custom exception and message.

Before:

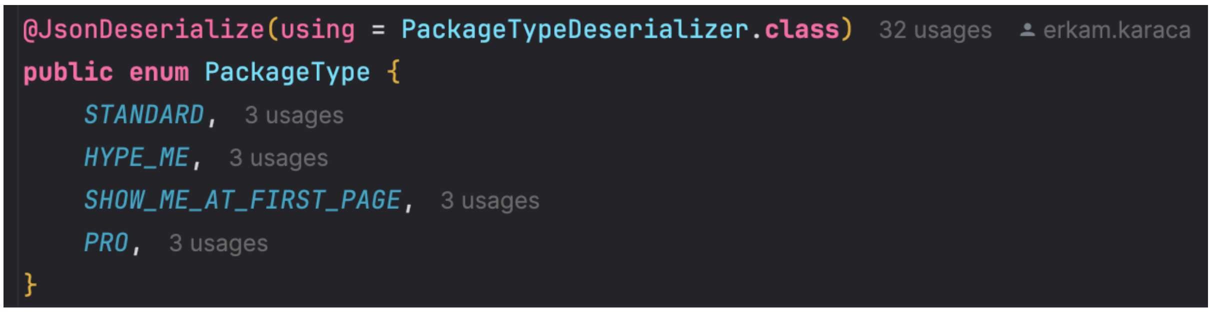


```
{
    "statusCode": 500,
    "message": "JSON parse error: Cannot deserialize value of type `org.erkam.propertyuserservice.model.enums.PackageType` from String \"MOCK_INVALID_TYPE\": not one of the values accepted for
        Enum class: [HYPE_ME, SHOW_ME_AT_FIRST_PAGE, PRO, STANDARD]",
    "details": "uri=/api/v1/users/packages"
}
```

Implementation:

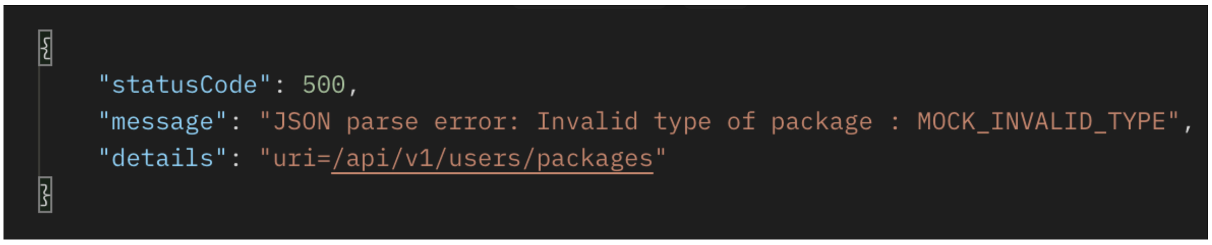


```
public class PackageTypeDeserializer extends JsonDeserializer<PackageType> {  2 usages
    @Override
    public PackageType deserialize(JsonParser p, DeserializationContext ctxt) throws IOException, JsonProcessingException {
        String value = p.getText().toUpperCase();
        return Arrays.stream(PackageType.values()) Stream<PackageType>
                .filter(type -> type.name().equals(value))
                .findFirst() Optional<PackageType>
                .orElseThrow(() -> new PackageException.InvalidTypeOfPackageException(PackageExceptionMessage.INVALID_TYPE_OF_PACKAGE, value));
    }
}
```



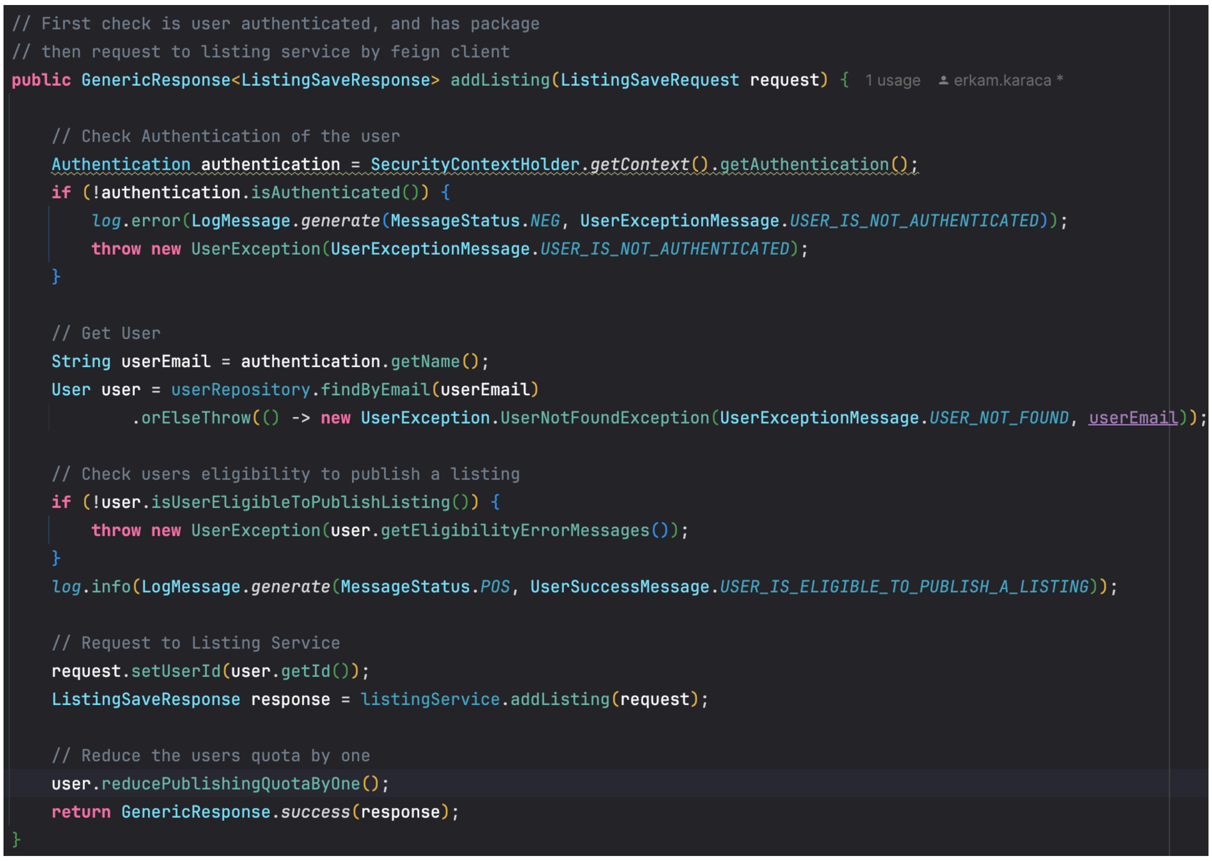
```
@JsonDeserialize(using = PackageTypeDeserializer.class)  32 usages  erkam.karaca
public enum PackageType {
    STANDARD,  3 usages
    HYPE_ME,  3 usages
    SHOW_ME_AT_FIRST_PAGE,  3 usages
    PRO,  3 usages
}
```

After:



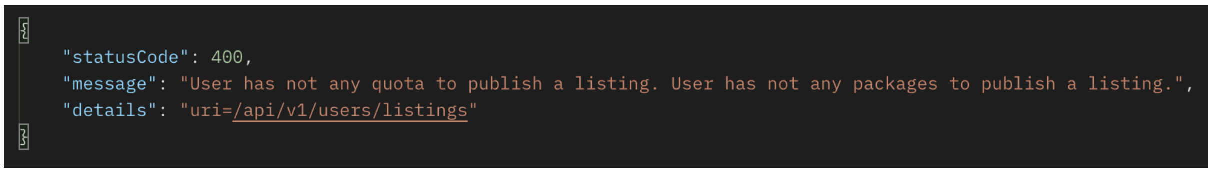
```
{
    "statusCode": 500,
    "message": "JSON parse error: Invalid type of package : MOCK_INVALID_TYPE",
    "details": "uri=/api/v1/users/packages"
}
```

Finished the addListing() method, with checking eligibility to publish a listing and reducing the publishing quota of the user.



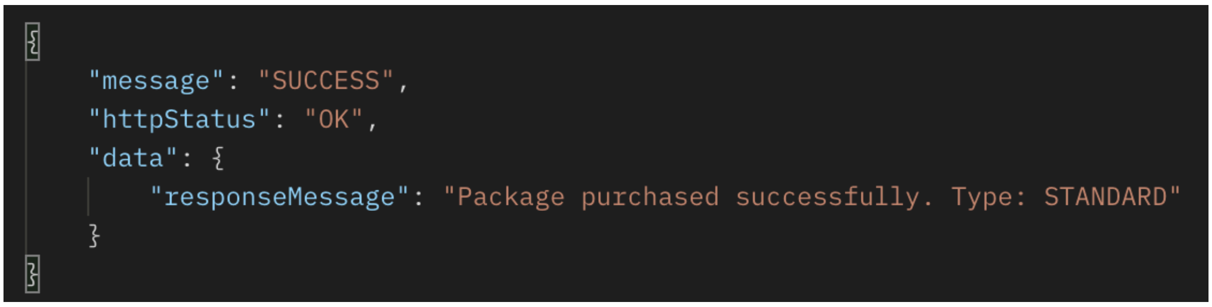
```java
// First check is user authenticated, and has package
// then request to listing service by feign client
public GenericResponse<ListingSaveResponse> addListing(ListingSaveRequest request) {   1 usage   erkam.karaca *

    // Check Authentication of the user
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (!authentication.isAuthenticated()) {
        log.error(LogMessage.generate(MessageStatus.NEG, UserExceptionMessage.USER_IS_NOT_AUTHENTICATED));
        throw new UserException(UserExceptionMessage.USER_IS_NOT_AUTHENTICATED);
    }

    // Get User
    String userEmail = authentication.getName();
    User user = userRepository.findByEmail(userEmail)
            .orElseThrow(() -> new UserException.UserNotFoundException(UserExceptionMessage.USER_NOT_FOUND, userEmail));

    // Check users eligibility to publish a listing
    if (!user.isUserEligibleToPublishListing()) {
        throw new UserException(user.getEligibilityErrorMessages());
    }
    log.info(LogMessage.generate(MessageStatus.POS, UserSuccessMessage.USER_IS_ELIGIBLE_TO_PUBLISH_A_LISTING));

    // Request to Listing Service
    request.setUserId(user.getId());
    ListingSaveResponse response = listingService.addListing(request);

    // Reduce the users quota by one
    user.reducePublishingQuotaByOne();
    return GenericResponse.success(response);
}
```

Testing add listing use case:

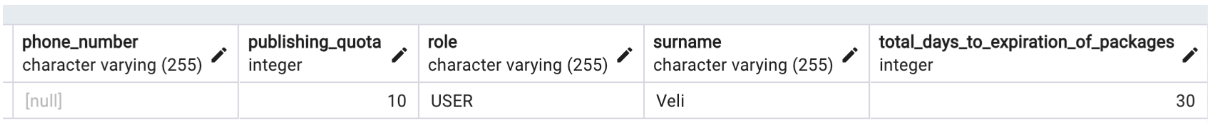


```
{
    "statusCode": 400,
    "message": "User has not any quota to publish a listing. User has not any packages to publish a listing.",
    "details": "uri=/api/v1/users/listings"
}
```

Purchasing STANDARD (quota: 10, duration: 30 days) package:

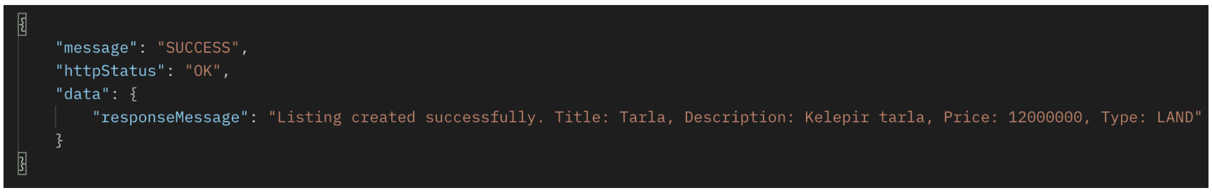


```
{
    "message": "SUCCESS",
    "httpStatus": "OK",
    "data": {
        "responseMessage": "Package purchased successfully. Type: STANDARD"
    }
}
```

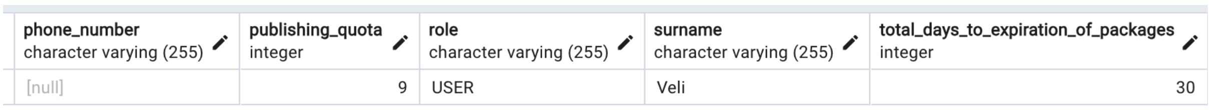


phone_number character varying (255)	publishing_quota integer	role character varying (255)	surname character varying (255)	total_days_to_expiration_of_packages integer
[null]	10	USER	Veli	30

Publishing a listing:

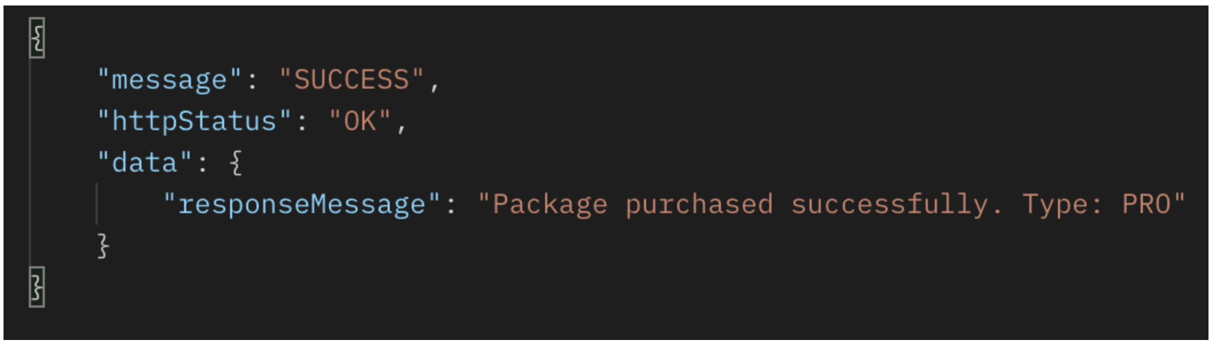


```
{
    "message": "SUCCESS",
    "httpStatus": "OK",
    "data": {
        "responseMessage": "Listing created successfully. Title: Tarla, Description: Kelepir tarla, Price: 12000000, Type: LAND"
    }
}
```

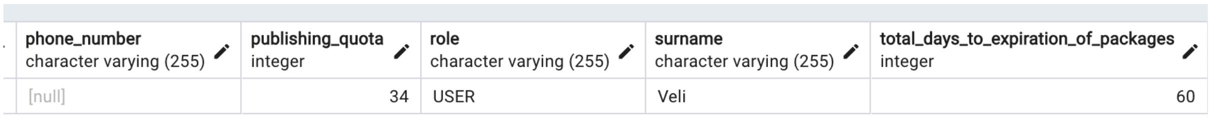


phone_number character varying (255)	publishing_quota integer	role character varying (255)	surname character varying (255)	total_days_to_expiration_of_packages integer
[null]	9	USER	Veli	30

One more package purchased PRO(quota: 25, duration: 30 days):



```
{
    "message": "SUCCESS",
    "httpStatus": "OK",
    "data": {
        "responseMessage": "Package purchased successfully. Type: PRO"
    }
}
```



phone_number character varying (255)	publishing_quota integer	role character varying (255)	surname character varying (255)	total_days_to_expiration_of_packages integer
[null]	34	USER	Veli	60