

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) ? ConsoleColor.ANSI_RED : ConsoleColor.ANSI_GREEN;
        return color + message + ConsoleColor.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) ? ConsoleColor.ANSI_RED : ConsoleColor.ANSI_GREEN;
        return color + message + ConsoleColor.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) ? ConsoleColor.ANSI_RED : ConsoleColor.ANSI_GREEN;
        return color + message + ConsoleColor.ANSI_RESET + " = " + id;
    }
}
```

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

.UserService  : User already exists = alivelili@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 = alivelili@gmail.com
_0 where u1_0.id=?
us,l1_1.title,l1_1.type from users_listings l1_0 join l1_1
quantity,p1_1.title from users_packages p1_0 join packages p1_1
_0 where p1_0.id=?

.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.

The screenshot shows the Swagger UI homepage for the 'Property App' API. At the top, there's a navigation bar with the 'Swagger' logo and a link to '/v3/api-docs'. Below the header, the main title is 'API Documentation of Property App © - Erkam Karaca' with an 'OAS 3.0' badge. Underneath the title, there's a link to '/v3/api-docs'. The page contains several links for documentation and legal information: 'Terms of service', 'Erkam Karaca - Website', 'Send email to Erkam Karaca', and 'My Licence'.

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

The screenshot displays two coverage reports side-by-side. The top report is for 'UserControllerTest' and the bottom one is for 'UserServiceTest', both using the 'Coverage' tool. Each report has a header with the test name and a close button. Below the header is a toolbar with icons for file operations. The main area is a table showing coverage metrics for different elements. In 'UserControllerTest', the 'UserController' class is fully covered at 100% for all metrics. In 'UserServiceTest', the 'UserService' class is also fully covered at 100% for all metrics.

Element	Class, %	Method, %	Line, %	Branch, %
org.erkam.propertyapp.controll UserController	100% (1/1)	100% (5/5)	100% (5/5)	100% (0/0)
org.erkam.propertyapp.service UserService	100% (1/1)	100% (8/8)	100% (25/25)	100% (4/4)

07.07.2024

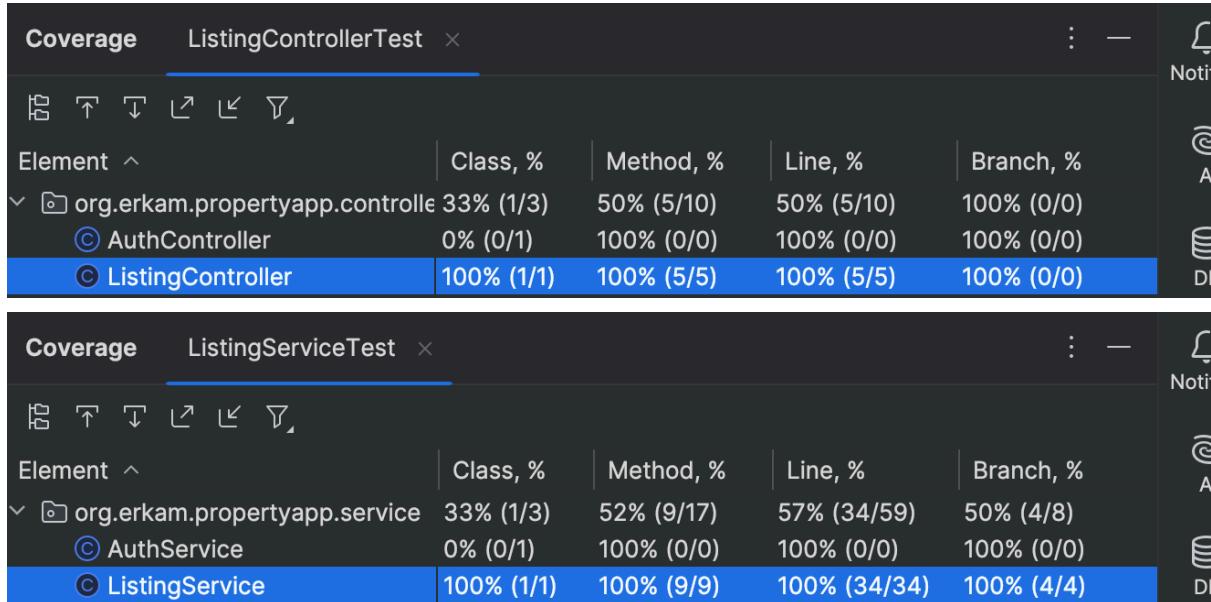
Listing controller, service and repository added.

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

    // TODO: Implement a check mechanism to know if there is 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.



```

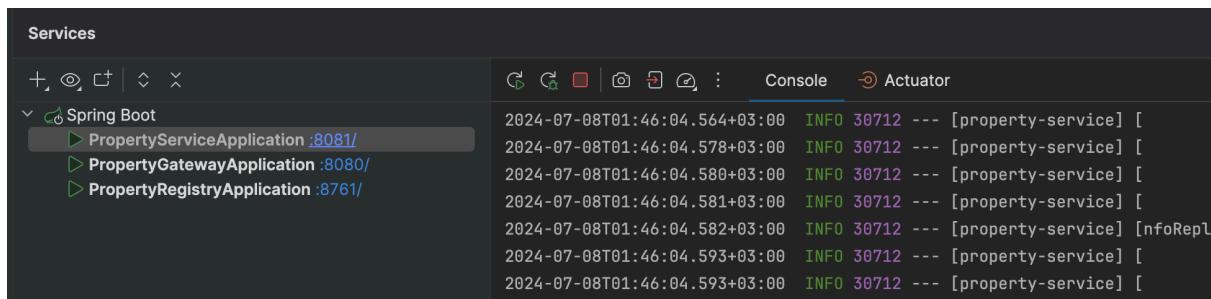
✓ Tests passed: 8 of 8 tests – 623 ms

/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

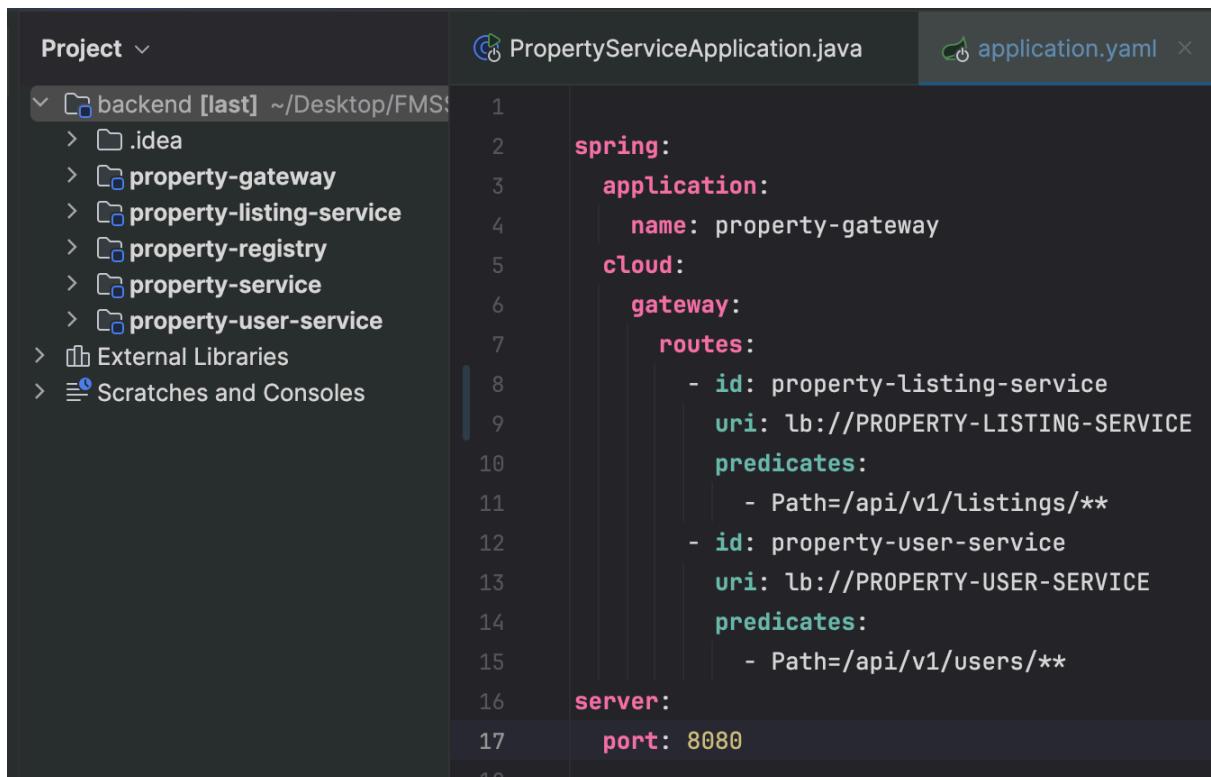
```

Gateway and Registry added.



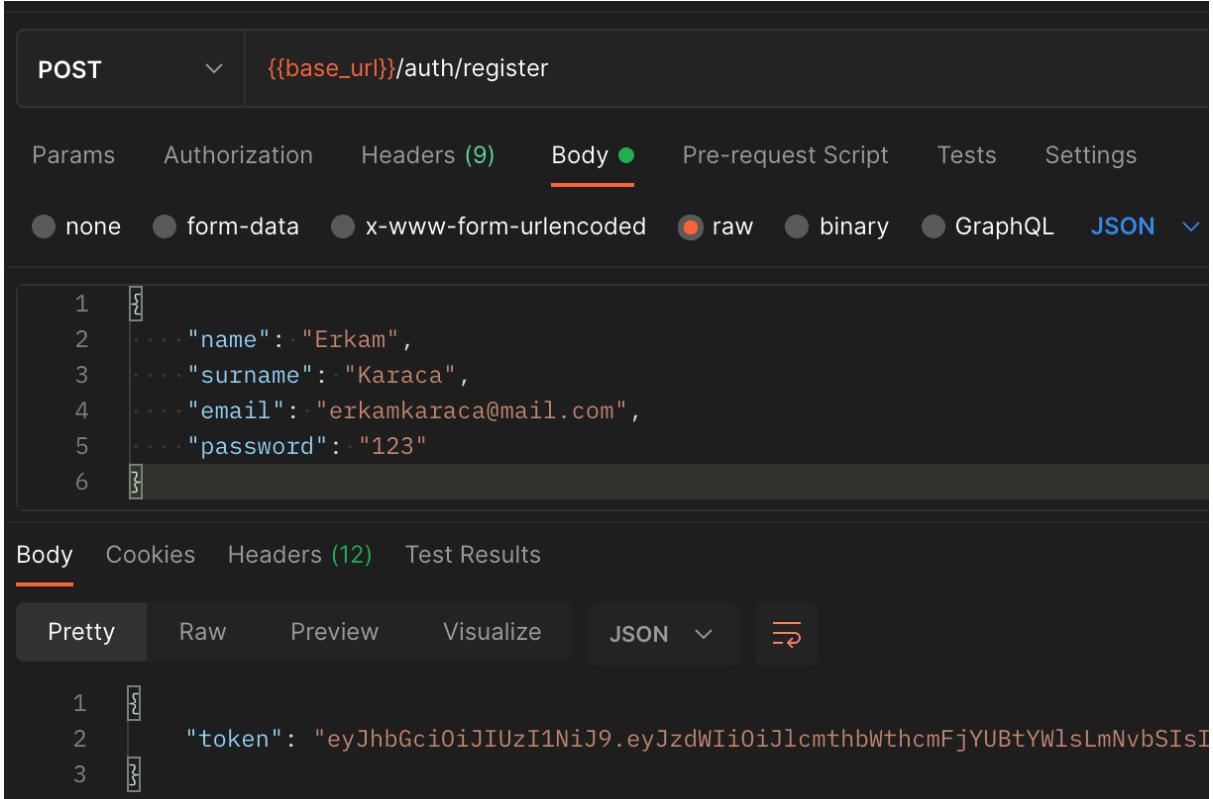
08.07.2024

Listing Service and User Service turned into microservices.



09.07.2024

JWT Authentication implemented. Auth Controller and Auth Service added.



POST {{base_url}}/auth/register

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body (JSON)

```
1 {  
2   "name": "Erkam",  
3   "surname": "Karaca",  
4   "email": "erkamkaraca@mail.com",  
5   "password": "123"  
6 }
```

Body Cookies Headers (12) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlcmttbWthcmFjYUBtYWlsLmNvbSIsI  
3 }
```

```
public class JwtAuthenticationFilter extends OncePerRequestFilter {  
    protected void doFilterInternal(  
        HttpServletRequest request,  
        HttpServletResponse response,  
        FilterChain filterChain) throws ServletException, IOException {  
        String jwt = request.getHeader("Authorization");  
        if (jwt != null) {  
            jwt = jwt.substring(7);  
            String userEmail = jwtService.extractUsername(jwt);  
            UserDetails userDetails = userDetailsService.loadUserByUsername(userEmail);  
            if (jwtService.isTokenValid(jwt, userDetails)) {  
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(  
                    userDetails,  
                    null,  
                    userDetails.getAuthorities());  
                authManager.setAuthenticatedUserPrincipal(authToken);  
            }  
        }  
        filterChain.doFilter(request, response);  
    }  
}
```

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

```
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)



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.

5



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 emphasize 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 okay, 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.

```
// 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.

```

/*
 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));
        }
    }

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

```

11.07.2024

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.

```

.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()

```

```

// 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);
}

```

```

// 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.

```

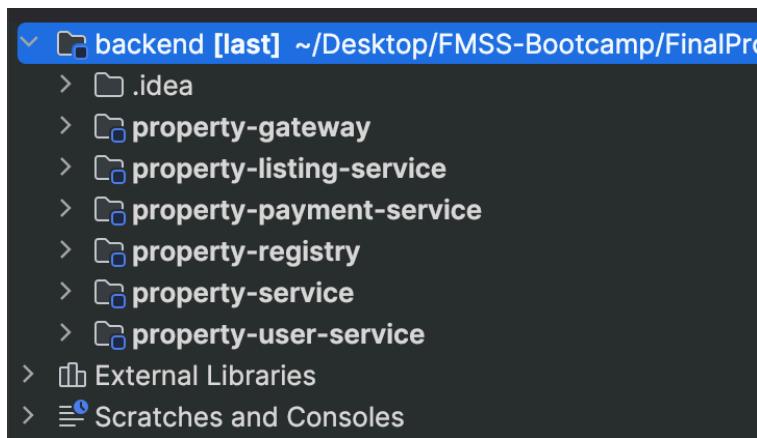
public void updateUserAfterBuyingPackage(BuyPackageRequest request) { 1 usage ▾ erkam.karaca
    updateTotalDaysToExpirationOfPackages();
    updatePublishingQuota(request.getType());
}

// 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;
}

```



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.

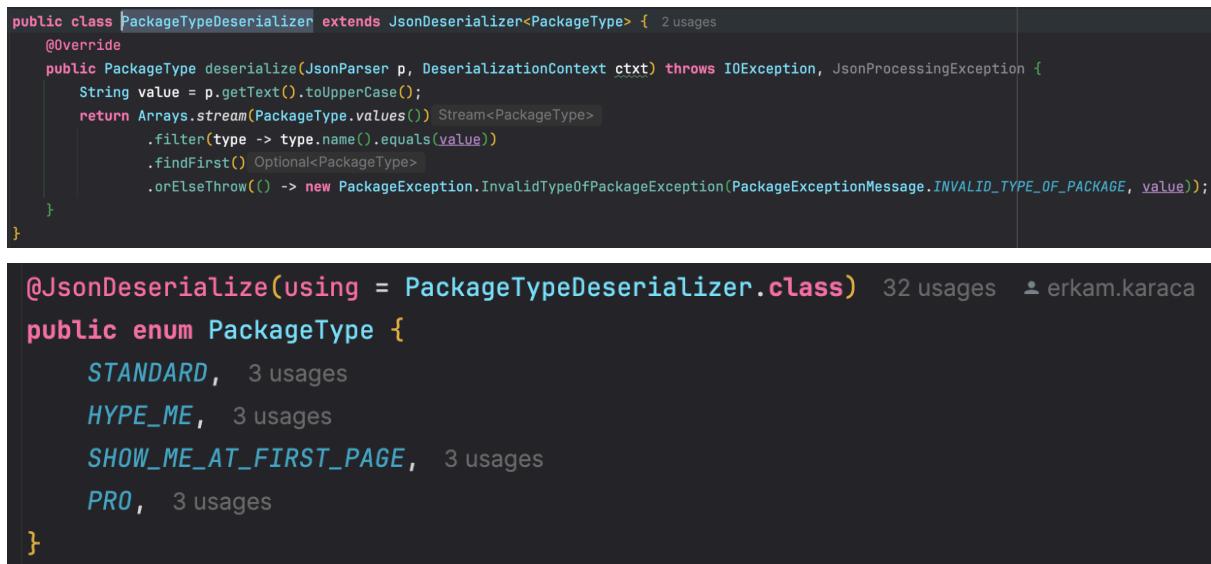
Before:



A screenshot of a REST API response in a browser. The status bar at the top shows "Status: 500 Internal Server Error Time: 185 ms Size: 700 B Save Response". The main content area has tabs for "Body", "Cookies", "Headers (12)", and "Test Results". The "Pretty" tab is selected, showing the following JSON response:

```
1 {"statusCode": 500,
2   "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
3     Enum class: [HYPE_ME, SHOW_ME_AT_FIRST_PAGE, PRO, STANDARD],
4   "details": "uri=/api/v1/users/packages"
5 }
```

Implementation:



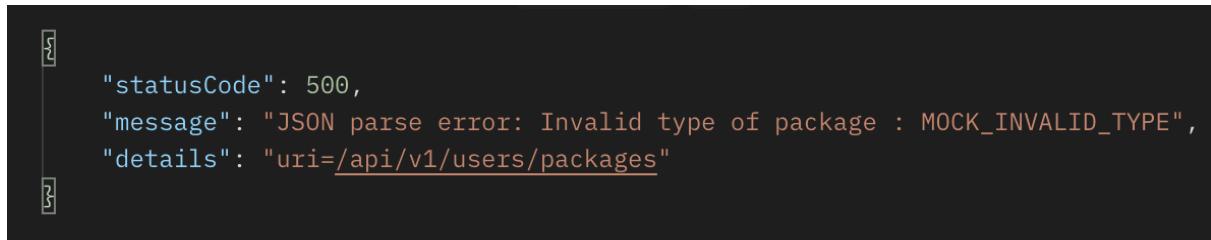
A screenshot of Java code. The top part shows a custom deserializer for the `PackageType` enum:

```
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()
            .orElseThrow(() -> new PackageException.InvalidTypeOfPackageException(PackageExceptionMessage.INVALID_TYPE_OF_PACKAGE, value));
    }
}
```

The bottom part shows the `PackageType` enum definition:

```
@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:



A screenshot of a simplified JSON response. It contains only the necessary fields: statusCode, message, and details.

```
{ "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.

```
// 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 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

13.07.2024

I secured the endpoints of both services (User, Listing) that returns PASSIVE listings and Packages of the user. To send the JWT in the headers of the requests from User Service to Listing Service I have implemented Feign Client Interceptor.

```
.authorizeHttpRequests(req -> req
    // NOTE: Secured the endpoints that returns the listing with PASSIVE status.
    .requestMatchers(HttpMethod.GET, ⑤ "/api/v1/listings/user/{userId}/passive",
        ⑤ "/api/v1/listings/user/{userId}").authenticated()
    .anyRequest().permitAll()
)

.authorizeHttpRequests(req -> req
    // NOTE: Endpoints which includes PASSIVE listings and Packages that user owned is secured.
    .requestMatchers(HttpMethod.POST, ⑤ "/api/v1/users/listings/**",
        ⑤ "/api/v1/users/packages/**").authenticated()
    .requestMatchers(HttpMethod.GET, ⑤ "/api/v1/users/listings",
        ⑤ "/api/v1/users/packages/**",
        ⑤ "api/v1/users/listings/passive").authenticated()

@Component
@RequiredArgsConstructor
public class FeignClientInterceptor implements RequestInterceptor {

    private final JwtService jwtService;

    @Override
    public void apply(RequestTemplate requestTemplate) {
        String jwtToken = getJwtFromSecurityContext();
        if (jwtToken != null) {
            requestTemplate.header(name: "Authorization", ...values: "Bearer " + jwtToken);
        }
    }

    private String getJwtFromSecurityContext() { 1 usage
        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        if (principal instanceof UserDetails) {
            UserDetails userDetails = (UserDetails) principal;
            return jwtService.generateToken(userDetails);
        }
        return null;
    }
}
```

I have changed the delete listing controller and service implementation, I updated the JWT payload and added userId to JWT to be sure the listing that requested to delete belongs to the authenticated user. I implement the logic in the listing service.

And I immediately made necessary changes at Unit Tests of Controller and Service layers.

The screenshot shows the JUNIT interface. At the top, there's a navigation bar with links for Debugger, Libraries, Introduction, Ask, and a Crafted by Auth0 by Okta logo. Below the navigation bar, there's a dropdown menu set to Algorithm HS256. The main area is divided into two sections: Encoded and Decoded. The Encoded section contains a long base64 string: eyJhbGciOiJIUzI1NiJ9.eyJc2VySWQiOjIsInN1YiI6ImF5c2V2ZWxpMUBnbWFpbC5jb20iLCJpYXQiOjE3MjA4ODk0MjksImV4cCI6MTcyMDg5MDMyOX0.IDEPLmYmA0RhpbWibJmVcG3WH2Dv6SY4XNnf12q3_Fk. The Decoded section shows the algorithm as HS256 and the payload as a JSON object:

```
{  
  "alg": "HS256"  
}  
  
PAYLOAD: DATA  


```
{
 "userId": 2,
 "sub": "ayseveli1@gmail.com",
 "iat": 1720889429,
 "exp": 1720890329
}
```



Below the decoded section, there's a code editor showing Java code for a ListingDeleteController. It has a DeleteMapping annotation with a path variable {id}. The controller method deleteById takes a Long id and an HttpServletRequest request, returning a ResponseEntity<GenericResponse<ListingDeleteResponse>>. The implementation returns a new ResponseEntity<listingService.deleteById(id, request), HttpStatus.OK>.



```
@DeleteMapping("/{id}")
public ResponseEntity<GenericResponse<ListingDeleteResponse>> deleteById(@PathVariable Long id, HttpServletRequest request) {
 return new ResponseEntity<listingService.deleteById(id, request), HttpStatus.OK>;
}
```



Below the controller code, there's another code block for a ListingDeleteService. It first checks if a listing exists by id. If it does, it deletes the listing and returns a ListingDeleteResponse. If it doesn't, it throws a ListingNotFoundException.



```
// First check by id, to find out whether listing exists or not,
// if not exists then throw an exception, else delete the listing and return ListingDeleteResponse
public GenericResponse<ListingDeleteResponse> deleteById(Long id, HttpServletRequest request) {
 Long userId = (Long) request.getAttribute("userId");

 Listing listing = listingRepository.findByIdAndUserId(id, userId).orElseThrow(() -> {
 log.error(LogMessage.generate(MessageStatus.NEG, ListingExceptionMessage.LISTING_NOT_FOUND_OR_YOU_DONT_HAVE_PERMISSION, id));
 return new ListingException.ListingNotFoundException(ListingExceptionMessage.LISTING_NOT_FOUND_OR_YOU_DONT_HAVE_PERMISSION, id);
 });
 listingRepository.delete(listing);
 log.info(LogMessage.generate(MessageStatus.POS, ListingSuccessMessage.LISTING_DELETED, listing.getId()));
 return GenericResponse.success(ListingDeleteResponse.of(listing));
}
```



At the bottom, there's a terminal window showing test results and logs. It says "Tests passed: 8 of 8 tests - 703 ms". The logs show various INFO and ERROR messages related to the application's database interactions and service logic.



```
✓ Tests passed: 8 of 8 tests - 703 ms
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java ...
20:19:16.078 [main] INFO org.erkam.propertylistingservice.service.ListingService -- Listing deleted with id = 592
20:19:16.092 [main] ERROR org.erkam.propertylistingservice.service.ListingService -- Listing not found with id = 1
20:19:16.099 [main] ERROR org.erkam.propertylistingservice.service.ListingService -- Duplicate listing found = KUYXATR
20:19:16.102 [main] ERROR org.erkam.propertylistingservice.service.ListingService -- Listing not found or you don't have permission with id = 1
20:19:16.105 [main] ERROR org.erkam.propertylistingservice.service.ListingService -- No data found on database
20:19:16.108 [main] INFO org.erkam.propertylistingservice.service.ListingService -- All listings fetched
20:19:16.113 [main] INFO org.erkam.propertylistingservice.service.ListingService -- Listing created = TGA
20:19:16.117 [main] INFO org.erkam.propertylistingservice.service.ListingService -- Listing fetched with id = 3620

Process finished with exit code 0
```


```

14.07.2024

I have implemented the updateStatusOfTheListing() method. And like I mentioned before I want to show meaningful messages to user and for that I have implemented another deserializer for Listing Status and I have checked the requested status type in here to fulfill the requirement "User can update the status as only ACTIVE or PASSIVE".

```
import java.io.IOException;

public class ListingStatusDeserializer extends JsonDeserializer<ListingStatus> { 2 usages ▾ erkam.karaca
    @Override ▾ erkam.karaca
    public ListingStatus deserialize(JsonParser p, DeserializationContext ctxt) throws IOException, JsonProcessingException {
        String value = p.getText().toUpperCase();

        switch (value) {
            case "ACTIVE":
                return ListingStatus.ACTIVE;
            case "PASSIVE":
                return ListingStatus.PASSIVE;
            default:
                throw new ListingException.InvalidTypeOfStatusException(ListingExceptionMessage.INVALID_TYPE_OF_STATUS, value);
        }
    }
}
```

```
{
    "message": "SUCCESS",
    "httpStatus": "OK",
    "data": {
        "responseMessage": "Listing status updated as ACTIVE successfully."
    }
}
```

```
{
    "statusCode": 500,
    "message": "JSON parse error: Invalid type of status : IN_REVIEW",
    "details": "uri=/api/v1/users/listings/status"
}
```

```
{
    "statusCode": 500,
    "message": "Listing not found or you don't have permission",
    "details": "uri=/api/v1/users/listings/status"
}
```

Update Listing functionality implemented.

```
    "message": "SUCCESS",
    "httpStatus": "OK",
    "data": {
        "responseMessage": "Listing updated successfully. , Title: Updated Title, Type: HOUSE, Desc
    }
```

In order to fulfill the requirement "Listings should be initially saved with an **IN_REVIEW** status and later must be asynchronously updated to **ACTIVE** by another service." I created a new service named **Listing Review Service**. This service is consuming the queue elements named "**listing.review.queue**", this queue is produced by **Listing Service**. I used **RabbitMQ** for this requirement.

```
public class ListingReviewConsumer {
    private final ListingReviewService listingReviewService;
    ...
    @RabbitListener(queues = "listing.review.queue")
    public void consumeListingReview(ListingDto listingDto) {
        log.info(LogMessage.generate(MessageStatus.POS, message: "Listing in review to change status:", listingDto.toString()));
        if (listingDto.getId() == null) {
            log.error("Listing id is null for listing review: {}", listingDto);
            throw new IllegalArgumentException("Listing id cannot be null");
        }
        listingReviewService.handleListingReview(listingDto);
    }
}

2023-07-10T10:45:23.123Z (?,?,?,?,?,?) returning id
ice.ListingService : Listing created = Tarla 412
user.ListingReviewProducer : Listing with id: 21 added to the listing review queue, exchange:
aws.ConfigClusterResolver : Resolving eureka endpoints via configuration

r.ListingReviewConsumer : Listing in review to change status: = org.erkam.propertylistingrevie
r.ListingReviewService : Handling listing review with id = 21
.type,l1_0.user_id from listings l1_0 where l1_0.id=?
.type,l1_0.user_id from listings l1_0 where l1_0.id=?
where id=?
r.ListingReviewService : Listing status updated with id = 21
```