# PROJECT DOCUMENTATION:EYALTY SERVICES

**Project Title: Eyalty Services for User Management System**

**Submitted By:T.Monica**

## Project Overview

### Description:

The project is a User Management System built using Java and Spring Boot. It allows users to register, log in, and manage their profiles. The application module has User service-core and Organisation service-core is modularized into multiple components: Core, Commons, DTO and Exception handling.

### Technologies Used:

- Java version:Java 17
- Framework:Spring Boot 3
- Development Tool:Intellij Idea
- Build Automation Tool:Maven

### Modules Overview:

| MODULE | PURPOSE |
|---|---|
| **Core** | Contains business logic for user registration and login. |
| **Commons** | Contains reusable utilities, constants, and DTOs |
| **DTO** | Data transfer objects used across modules |

| | |
|---|---|
| **Exception** | Exceptions for error handling |

## User Service-Core Module

### Purpose:

Here I have implemented the Commons,DTO and Exception Handling to handle the business logic for registration and login. It Validates inputs, checks user existence and handling the errors.

### 1.Commons

☐ **Purpose**: To hold **common utility code or shared components** that can be used across modules.In this module,I have implemented some Java classes which is based on rules of Business details rules that is given for user login and registration purpose.

### Dependency implementation in Commons:

- Mostly each fields initialized using Lombok dependency to generate getter,setter methods and constructors automatically.
- Lombok dependency is mainly used to reduce writing larger and repetitive codes.Instead of writing 30-50 getter and setter methods manually,lombok generate methods at compile time.
- To utilize the full potential of Lombok,needs to include dependency in Pom.xml is essential along with this getter,setter annotations are added in code.

### Implementation in Commons:

#### 1.Business Details:

- In this Java class,it represents the Business related fields.
- The Fields includes userUUID,trading name,UUID,regionid,address1,address2,registration number,name of Business,Category.

- These fields are used for user profile registration.

**Sample code for Registration:**

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class BusinessDetails
{
    private String userUUID;
    private String category;
    private String region;
    private String uuid;
    private String nameOfBusiness;
    private String registrationNumber;
    private String address1;
    private String address2;
}
```

## 2.EMAIL Payload and Reset Password:

- This class is used as a Data Transfer Object(DTO) for email payload field.
- I have used Validation constraints library to validate user input in this class so it is used to check whether the email is valid or not.

- To ensure validation,characters should not be empty and characters size should be within the range according to validation constraints.

**Sample code for Email payload:**

```
import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

public class Email

{

    @Size(max = 100, message = "The field email must be less than 100 "
+"character")

     @.Email(message = "Email should be valid")

    @NotNull(message = "Email should not be null")

    private String email;

     @Size(max = 6)

    @NotNull(message = "Token should not be null")

    private String token;

}
```

**Reset Password:**

```
public class ResetPassword

{

     @ValidPassword(message = "Password should be valid")

    private String password;

     private String confirmPassword;

     @NotNull(message = "Token should not be null")      //@Size(max = 6)

    private String token;
```

}

**Interest Details and Mobile number:**

To ensure the user interest in registration process.For mobile number,validation constraints is used to validate mobile number.

**Code for interest:**

```
public class InterestDetails{
private String interest1;
private String interest2;
private String interest3;
}
```

**MAIL:**

The class used for ensuring mail sending format.

```
public class Mail{
    private String from;
    private String mailTo;
    private String subject;
    private List<Object> attachments;
    private Map<String, Object> props;
 }
```

**Region details,Notification payload and Token Response:**

- The class is used to ensuring the location details,notification enabling for region payload.
- For enabling notification,token access and token response fileds are used.

**Token enabling:**

```
public class TokenResponse
{
    private AccessTokenResponse tokenDetails;
    private String userUUID;
private Date sessionTimestamp;
}
```

**Location Details:**

```
public class RegionWithCountryDetails

{

    private String countryCode;

    private Long postalCode;

    private String telephonePrefix;

    private String currency;

    private String regionName;

    private String regionUUID;

}
```

## User Registration:

This class is used for login and registration purpose for user.

```
public class UserObjectArrangement

{

        /*Used to arrange object for the given user details.*/

        public UserLogin arrangeObject(UserProfile user)

        {

                UserLogin newLoginUser = new UserLogin();

                newLoginUser.setUserId(user.getUuid());

                newLoginUser.setCreatedBy(user.getCreatedBy());
```

```
                    newLoginUser.setUpdatedBy(user.getUpdatedBy());

            return newLoginUser;
                }
        }
```

## 2.DTO-Data Transfer Object:

**Purpose:** DTO carries only the required data from Service to Controller.It transfers the data between all the layers and mainly used to hold the data in variables.

**Business Notification and Business Profile Payload:**

This class is used to get business notification for users.Business profile field is used to set Profile name using validation constraints library where fields contains **first name,last name and email**.

**Code:**

```
public class BusinessNotificationList
{
        private List<BusinessNotification> businessNotifications;
}
public class BusinessProfilePayload
{
@Size(max = 50,message = "The field firstName "+ "must be less than 50
character")
@Pattern(regexp = "^[a-zA-Z]*", message = "First Name must not " +
   "contain any special characters or numbers")
@NotNull(message = "FirstName should not be null")
```

```
        private String firstName;

           @Size(max = 50,message = "The field lastName "+ "must be less
than 50     character")

}
```

**Login and Userchecklogin Modules:**

- This class is initialized the variables which contains **email,ssotype,jwt token.** UserProfileDetails class contains variables of **currentPage,totalPages and totalItems.**
- **ContactUsResponse and EmailResponse classes are initialized the variable called details to get contact related details**
- Some of the modules used in **Commons** modules are again reusable into DTO module which contains of about **NotificationDetails,LocationDetails,TokenResponse and Userchecklogin,userProfilePayload.**

**3.EXCEPTION HANDLING:**

**Purpose:** Centralized Error Handling which means One place to catch and manage all exceptions.

**Implemented exception classes :**

✓ **BusinessNotFoundException** is used for when user is not found.

✓ **InternalServerDownException** is used for when internal server is down.

✓ **InvalidCountryCodeException** is used for when country code is invalid.

✓ **InvalidTokenException and InvalidUUID** are used for when token and is invalid.

✓ **KeyCloakException** is thrown when wrong client credentials to KC.

- ✓ **KeyCloakSameUserException** is thrown from KC when the user enters same email.
- ✓ **NotificationNotFound and Region NotFoundException** are thrown when notification and region are not found.
- ✓ **TokenExpired and TokenMismatchException** are used when token has mismatch characters and expired token.
- ✓ **SSOLoginFailed and UserAuthenticationFailedException** are thrown when Login and User authentication has been failed.

**Sample code for Exception:**

```
public class InvalidCountryCodeException extends CoreException

{

private static HttpStatus statusCode = HttpStatus.NOT_FOUND;

public InvalidCountryCodeException(String message, String details,

                                                   String parameter,
String pointer)

{

        super(details, statusCode, message, parameter, pointer);

}

}
```

**2.Organisation Service-Core Module:**

**Purpose:**Organisation service is used for Subscription and payment related details.

**1.Commons:**

Some classes are implemented which contains **ChargeRequest for charging payment,RegionWithCountry for Country details and SubscriptionObject for customer subscriptions.**

**Sample code:**

```
public enum SubscriptionObject
{
        program, customer, special, promotion
}
```

## 2.DTO-Data Transfer Object:

## 1.Business Details:

- In this Java class,it represents the Business related fields.
- The Fields includes userUUID,trading name,UUID,regionid,address1,address2,registration number,name of Business,Category.
- These fields are used for user profile registration for organisation that the same implemented in user service.

### 2.Notification Details:

- This class is used to initialize the business related notification
- The fields contains of about id,userid.programdetails,targetuserid,targetuserid,programid,programname,status,title,body,action,type,page and createdAt.
- **Sample code:**

```
public class NotificationDetails
{
      private String uuid;
      private String userUuid;
      private String programmeUUID;
```

}

**3.Organisation Document:**

- This class is used to initialize the customer subscription details.
- The fields contains of about **id,businessName,registrationNumber,Category,tradingName,memberStatus,registrationDate,programmCount,customerCount,promotionCount,regionDate,regionId,postalCode,address1 and address2.**
- **Sample code:**

```
public class OrganisationDocument
{
    private String registrationNumber;
    private String tradingName;
    private String category;
    private String memberStatus;
    private Integer customerCount;
}
```

**4.Payment Audit:**

- This class is used to initialize the payment related details
- The fields contains of about **currency,amount and clientSecret.**

**Sample code:**

```
public class PaymentDetails
{
        private float floatAmount;
        private String currency;
        private String client_secret;
    }
```

**5.Reference Details:**

- This class is used to initialize the getting customer's another reference details.
- The fields contains of about **telephonePrefix,regionName,state,currency,postalCode,lattitude,longitude and regionID.**

6.StripeResponse:

- This class is used to initialize the getting response from server about subscription.
- The fields contains of about **status,message,sessioniD,sessionurl.**

**Sample code:**

```
public class StripeResponse {
      private String status;
    private String message;
    private String sessionId;
    private String sessionUrl;
}
```

7.**Subscription Details and Subscription Info:**

- This class is used to initialize subscription info about customer.
- The fields contains of about **image,title,content,price and opageContent.**

**Sample code:**

```
public class SubscriptionDetails
{
    private float price;
    private List<SubscriptionInfo> pageContent;
    private String image;
    private String title;
    private String content;
}
```

**3.Exceptions:**

In Organisation-core module,**userNotFoundException** is created which is used when we identify no user is found.

## FUTURE ENHANCEMENT:

- As part of the future enhancement, a repository module will be introduced to manage database interactions with the persistence layer.
- Additionally, a security module is planned to provide authentication and authorization features, safeguarding the system through role-based access control and secure endpoints.
- This phased development approach ensures that the application is both modular and scalable, with a strong foundation for future improvements.