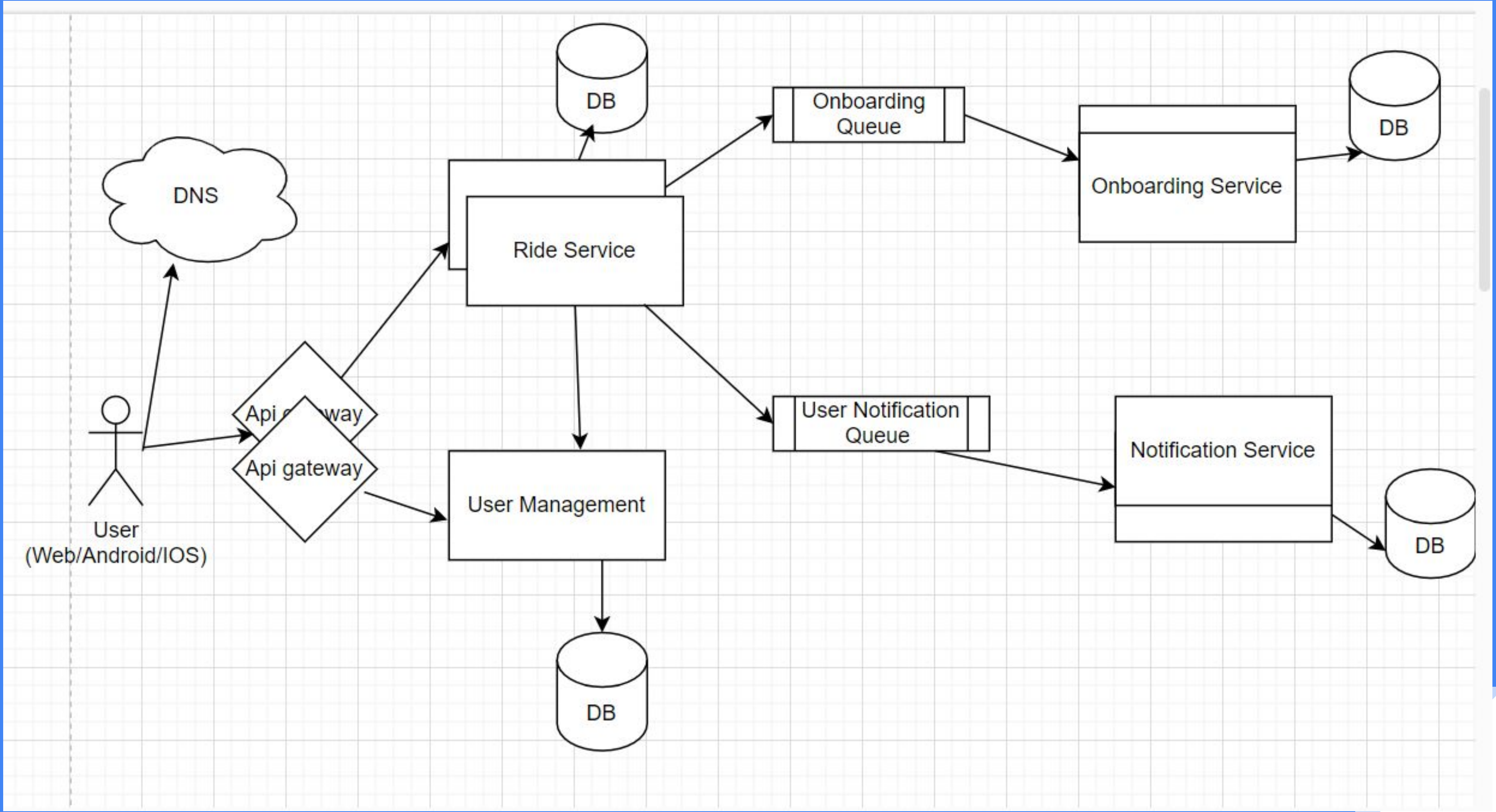
# Ride application

## Requirements

a) Allow a driver to sign-up and enter their profile information
b) Trigger onboarding processes like document collection, background verification, shipping of tracking device, etc.,
c) Allow a driver to mark when they are ready to take a ride

## High Level design  Components

1.) User/Client - Web/Android/IOS
2.) DNS(Domain name server)
3.) Api Gateway
4.) Ride service -It can handle all operations related to cab availability and bookings etc


DB
Onboarding Queue
DB
DNS
Onboarding Service
Ride Service
Api gateway
Api gateway
User Notification Queue
Notification Service
User (Web/Android/IOS)
User Management
DB
DB

# Assumptions & Implementations:

1.) **Onboarding service -** This service is already implemented that handles onboarding process(Document collection, Background verification & shipping of Tracking device) of Driver once signup is completed on ride service. Once onboarding is completed then that will be notified to ride service that now driver is onboarded.

2.) **User Management service -** This service is responsible for User signup,login.

3.) **SQSUtil -** SQSUtil class implemented as Queue that push messages to Queue so flow will work in async(background). Ex- Onboarding Queue & Notification Queue

4.) **Authentication** - Token based authentication is implemented with no expiry right now but we can add expiry as well as enhancement.  Rather than token based authentication jWT can also be used on microservice architecture.

**5.) User notification** - once user is signed up then a mail will be triggered to user to update password from random generated password by ride service, then that will be stored as encrypted in db.

**6.) MYSQL is used for Data storage.**

7.) **MD5 hash is used for password encryption**.

**8.) Logging - SL4J lombok**

**9.) Log tracking** - Sl4j MDC(Mapped diagnostic context) is used. Unique identifier for every thread in a application from end to end that helps to track each transaction flow.

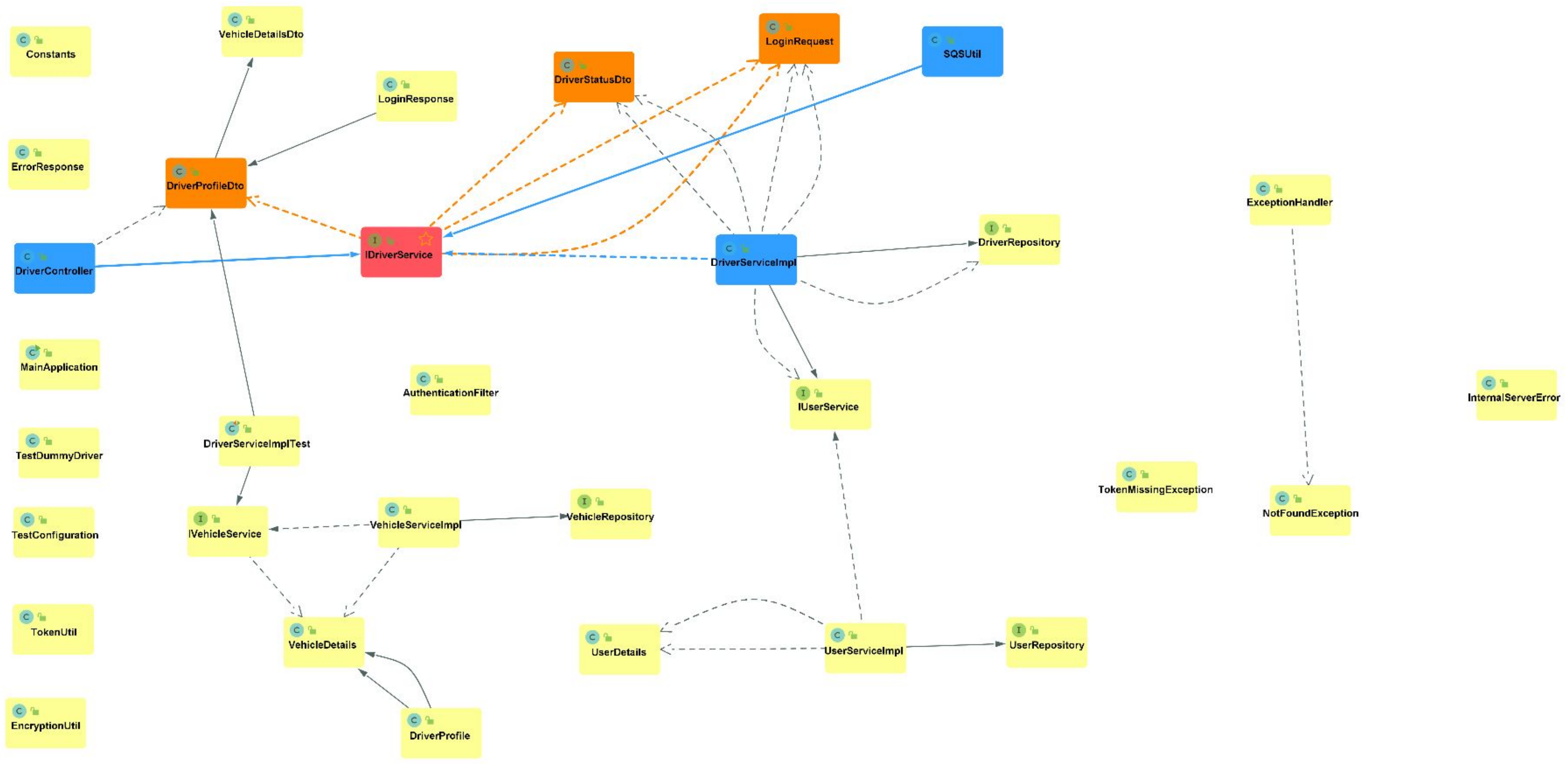
## Implementation

**10.) Application follows MVC architecture** - Controller, Service & Repository different layers

**11.) Used Builder design pattern**

**12.) Exception handling -** Custom exceptions with exception handler is implemented with handling of type of errorCode and errorMessage.

**13.) Junit test cases -** Junit is implemented as of now for register driver/signup so we can implement as same for other methods.


Constants
VehicleDetailsDto
LoginResponse
DriverStatusDto
LoginRequest
SQSUtil
ErrorResponse
DriverProfileDto
ExceptionHandler
DriverController
IDriverService
DriverServiceImpl
DriverRepository
MainApplication
AuthenticationFilter
IUserService
InternalServerError
TestDummyDriver
DriverServiceImplTest
TokenMissingException
NotFoundException
TestConfiguration
IVehicleService
VehicleServiceImpl
VehicleRepository
TokenUtil
VehicleDetails
UserDetails
UserServiceImpl
UserRepository
EncryptionUtil
DriverProfile

## Thanks !!