# **DOD ( Definition of Done ) :**

Simply Your core objective in this assignment is to write a simple service that is capable of inserting new records, modify or delete previously added records and fetch current version of records and currently implemented required things to persist datas and not missing them on service restart or any other kinds of problems. This service acts as an interface between end users using a web application, an RESTAPI just for now, and the database we persist records, postgreSQL for now.

The domain of this task is a “Book Store” that let us save required data of books ( like name, author, ISBN, price, etc. ) and let people to reserve books from their account using their account balance

The **minimum** **Required** data to handle

* Book data
  + keys
    - Title
    - ISBN
    - Price
    - genre
    - author(s)
    - Description
    - units
  + Considerations
    - A book may have one or more author(s)
    - Authors related to author table
* Author data
  + Keys
    - User
    - City
    - Goodreads link
    - Bank account number
  + Consideration
    - Authors are related to user table
* City data and genre data
  + Considerations
    - On your own.
* Customer data
  + Keys
    - User
    - Subscription model ( free - plus - premium )
    - Subscription end time
    - Wallet money amount
  + Consideration
    - Customers are related to user table
* User data
  + Keys
    - Username
    - First and last name
    - Phone
    - email
    - Password
    - User
  + Considerations
    - Users are either System Admins, Customers or Authors
* Reservation / order data
  + Keys
    - Customer id
    - Book id
    - Start of reservation
    - End of reservation
    - price
  + Consideration
    - Customer id is related to customer data table
    - Book id is related to book data table

The **minimum required** thing you should implements

* Complete CRUD service for books, user, customer and reservation data
  + No need to implement any crud things for author, genre and city tables
* General table model and relation and some sample data for all
* Api for every service you wrote, include correctly defined validation and serialization
* A service for book reservation

## **Checklist of DOD**

* Make required database table data model and relations between them and put sample ( at least­­­­ 10 rows of data ) in them
  + User
  + Author
  + Customer
  + Book
  + City
  + Genre
  + Reservation
* Make required CRUD service for :
  + Book
  + User
  + Customer
  + Reservation
* Make required general service ( that usually is something that is using some smaller service )
  + User auth - sign up, sign in, permission management functionalities
  + Reservation service - reservation checks and functionalities
  + purchasing service
* Make required apis for all crud services and general services
  + Required method to implement:
    - Create → POST
    - Read → GET
    - Update → PUT
    - Delete → DELETE
  + Required services
    - Simple cruds
      * Book
      * User
      * Customer
      * Reservation
    - General service
      * Auth
      * Reservation
      * purchasing
* General business logic consideration
  + Users login with password and then otp, instead of using an sms service, write an sample abstract version of sms service that instead of sending sms, print out otp code in the console.
  + A place for purchase plus ( monthly 50,000 Toman ) and premium ( monthly 200,000 Toman ) membership
  + Different membership status have different permissions
    - Free users :
      * cannot reserve books at all
      * just can fetch book data
    - Plus users can do anything in free plus :
      * Reserve books for at most 1 week ( 7 consecutive days )
      * Pay 1000 Toman for each reservation day
      * If read more than 3 different books in previous month ( 30 days ), get a 30% discount
      * If paid more than 300,000 Toman in previous two months ( 60 days ) get a 100% discount ( free reservation )
      * Can reserve up to 5 different book units simultaneously
    - Premium users can do anything in plus and :
      * Can reserve a book for at most 2 weeks ( 14 consecutive days )
      * In reservation queue for a book , they are prioritized
      * Can reserve up to 5 more different book units ( a total of 10 different book units ) simultaneously
  + Reservation
    - If there are more than 0 remaining units for a book to reserve, an instant reserve is issued on reserve request. Meaning another unit of book considered reserved, and the reservation time of that order starts at the moment of order placement
    - The number of books that instant reserved should not exeed the number of units of that book ( if we have 3 unit for a book, we cannot instant reserve it 4 times ), in other words, remaining unit for a book should not get negative
    - If there are 0 remaining units for a book to reserve, a scheduled reserve is issued on reserve request. Meaning the requested customer goes into a queue with custom ordering between customers and any time a unit of book released, we remove first customer from that queue and issue an instant reserve of that book for that customer issued on the time of book unit release
    - The order of customers in reservation queue defined in this way
      * Between all users, the ones with premium subscription model on the time of unit release have a higher priority and moves to the beginning of the queue before any other plus customer
      * Between all users with same subscription models, the one who scheduled a reserve sooner have a higher priority
    - Users can request to exit the queue any time they want
    - Users cannot instant reserve more than number of their remaining number of reservation limit. If the instant reserve request was from user it returns error. If it was from the queue, fails silently, user is removed from queue and not reservation issue for that user, we try to instant reserve that book for next ones in queue
    - If the current amount of wallet of user that issue a reserve, either instant or scheduled, is not enough for their order, return error, redirect to purchase api for the remaining required amount to charge their account.
    - On instant reserve the required amount of money is subtracted from their wallet. If on issuing instant reserve from queue the customer, the customer doesn’t have sufficient amount of charge, the request fails silently, user is removed from queue and not reservation issue for that user, we try to instant reserve that book for next ones in queue
  + Purchase
    - A simple place for people to charge their wallet
    - A place to request for upgrade / renew their subscription model. Remove the required amount from their wallet and change their subscription model and / or deadline
  + Other cruds
    - Each user - customer - author ‘s data can only be changed by him/herself
    - Each book’s data can only be changed by its author(s)
    - Reservation data cannot directly changed. It can only changed through reservation service
  + Auth
    - Implement handling auth, using jwt. This token used for auth process of users in any place. Only place that can be available without auth process is viewing book data and user sign up
    - Whole auth processes should implement as a service and get called from requested places
  + Admin
    - Admins can revoke any users token except admins
    - Admins can end reservations for a user before its end time
    - Admins can view a books current holders and scheduled reserver and thier orders and remove any one
* General technical consideration
  + All APIs have validation based on their data
  + Errors status codes handled currently based on their messages, reason and general guidelines for http errors status codes
  + You write services separated from apis. Apis are just something that calls those services.
  + Each data should be an entity object in system. These entities should validate themselves. Entity properties should not manipulated directly and can only be manipulated using its methods