Low Level Design

MakeMyTrip

Disclaimer

This is a high level overview of MakeMyTrip with comprehensive system requirements.

Before diving into the design part.Let us have some basic understanding of LLD and how to start with LLD.

What is LLD?

The name LLD stands for Low Level Design. This is one of the key steps in the development of scalable,available and highly maintainable software. It mainly deals with the class diagrams and object creations and the relations among the classes in a very detailed manner. All the successful applications that we are currently using such as stackoverflow, linked in, whatsapp and makemytrip ,are having good low level designs.

Why is LLD important?

Full Stack Development is one of the highly demanding domains. If a candidate is able to solve an LLD problem,then it's an indication that he can solve all the real time client or product manager requirements with ease. It’s an indication that the candidate is able to understand the business requirements and be able to convert them into code.

In most of the companies, the roles will be like **SDE1 -> SDE2 -> SDE3 -> Architect** as your seniority increases. As your seniority increases, you will be designing systems more rather than writing code.

How to Start with LLD?

Below are some prerequisites to get started with LLD.(*You find resources at the end of the doc*).

1. Learn most popular design patterns such as singleton and fascade patterns.(Design Patterns (refactoring.guru))
2. Learn SOLID and DRY principles (A Solid Guide to SOLID Principles | Baeldung ,baeldung.com/cs/dry-software-design-principle)
3. Try to code the patterns in any of the OOP supported language such as c#, c++, java.

How to Solve L.L.D. problems?

To solve L.L.D. problems you need to follow the below steps.

1. Find out all the requirements of the system.
2. Clarify all the requirements and write them on paper.
3. Create a Use Case diagram that tells all the activities done in the system by all the actors and system. Use Case Diagrams | Unified Modeling Language (UML) - GeeksforGeeks
4. Using use case diagram, try to create a class diagram that gives the blue picture of the system in terms of OOPs. Class Diagram | Unified Modeling Language (UML) - GeeksforGeeks
5. By seeing the class diagram try to find out which patterns can be applied among classes and objects, then start coding.

How to Know Which Design Pattern to Use?

To be honest, it all comes with practice. The more and more situations you see and the more and more problems you solve, the more and more ideas you get to select patterns.

Looking at different open source systems helps us to find the common problems and patterns to be used.

Design Problem

Create a flight booking system that lets users see all the available flights and tickets, choose a ticket and book a ticket. If the user wants, he should be able to cancel the tickets.

Solution

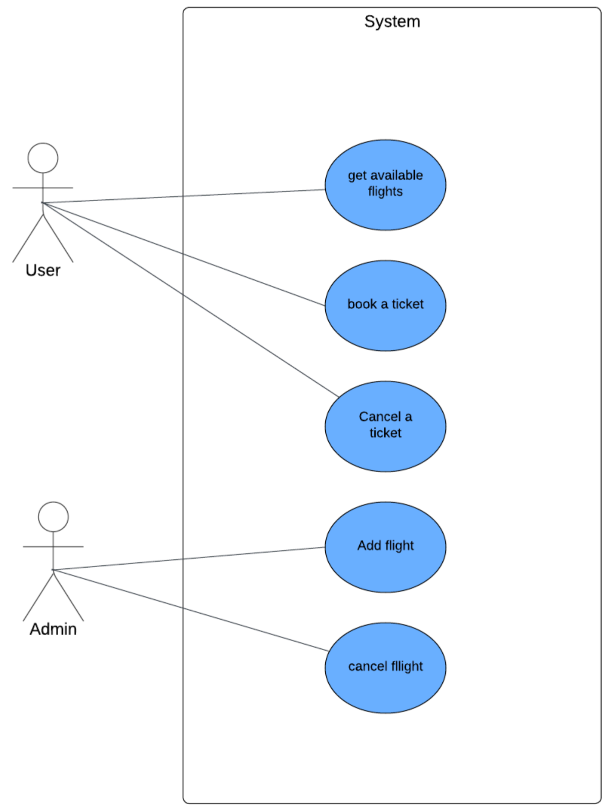
We follow the below steps to solve the problem.

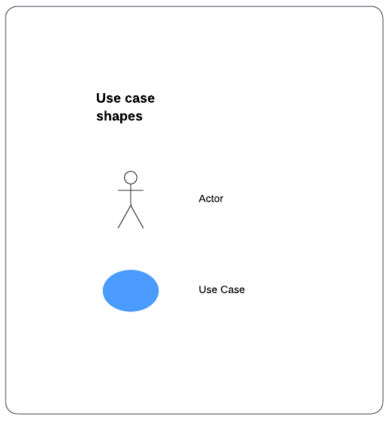
1. List down the requirements.
2. Make a use case diagram.
3. Make a class diagram.
4. Find the patterns and code.

List Down requirements

➔ Users should be able to fetch the flight details for a particular date with source and destination.
➔ Users should be able to reserve a number of tickets as per availability.
➔ Users should be able to cancel a particular ticket or all the tickets.
➔ Admin should be able to add flights.
➔ Admin should be able to cancel flights in case of emergency.

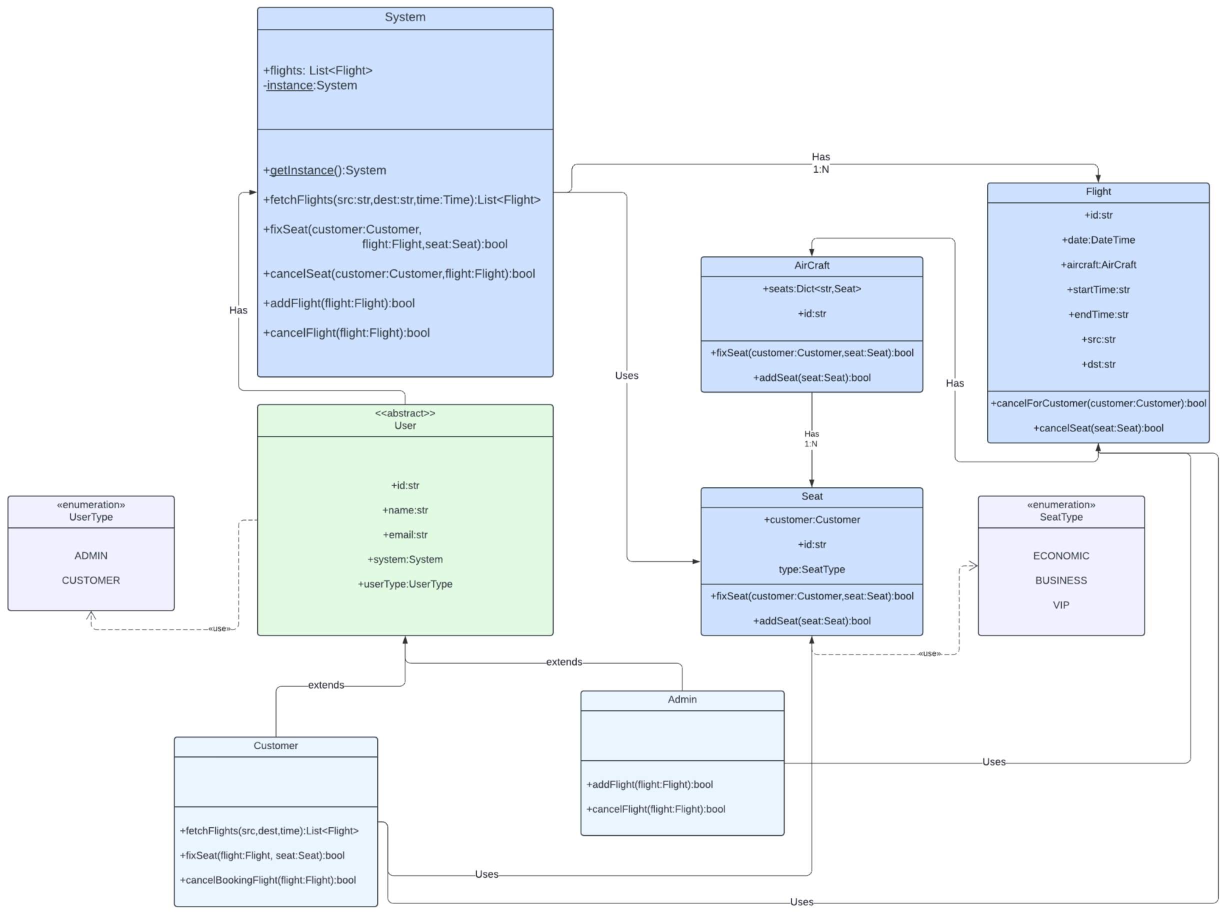
Use Case Diagram





Note: we are using an actor called system that provides all the APIs required by user and admin. So whenever an admin or user wants to do the operations they call those APIs. You will understand it well in the class diagrams.

Class Diagram


System
+flights: List<Flight>
-instance:System
+getInstance():System
+fetchFlights(src:str,dest:str,time:Time):List<Flight>
+fixSeat(customer:Customer, flight:Flight,seat:Seat):bool
+cancelSeat(customer:Customer,flight:Flight):bool
+addFlight(flight:Flight):bool
+cancelFlight(flight:Flight):bool
Has
1:N
Flight
+id:str
+date:DateTime
+aircraft:AirCraft
+startTime:str
+endTime:str
+src:str
+dst:str
+cancelForCustomer(customer:Customer):bool
+cancelSeat(seat:Seat):bool
AirCraft
+seats:Dict<str,Seat>
+id:str
+fixSeat(customer:Customer,seat:Seat):bool
+addSeat(seat:Seat):bool
Has
Uses
Has
Has
1:N
<<abstract>>
User
+id:str
+name:str
+email:str
+system:System
+userType:UserType
«enumeration»
UserType
ADMIN
CUSTOMER
«use»
Seat
+customer:Customer
+id:str
type:SeatType
+fixSeat(customer:Customer,seat:Seat):bool
+addSeat(seat:Seat):bool
«enumeration»
SeatType
ECONOMIC
BUSINESS
VIP
«use»
extends
extends
Admin
+addFlight(flight:Flight):bool
+cancelFlight(flight:Flight):bool
Uses
Customer
+fetchFlights(src,dest,time):List<Flight>
+fixSeat(flight:Flight, seat:Seat):bool
+cancelBookingFlight(flight:Flight):bool
Uses
Uses

Finding Patterns and Code

- If You see the above class diagram, as I am having common properties between Admin and Customer, I make an abstract class and inherit them.
- The class System should be singleton,because it is maintaining the state of the whole application and all the operations are happening through this System class only.
- The c# code is provided in this repository. You can have a look at it.

Conclusion

➔ Low level design problems are open ended. There is no perfect solution to it.

➔ 80% percent of solutions for the problems are acceptable.

➔ As it is my first Low level design problem, I might have made some mistakes.

Resources and References

Class Diagram | Unified Modeling Language (UML) - GeeksforGeeks

Use Case Diagrams | Unified Modeling Language (UML) - GeeksforGeeks

DRY Software Design Principle | Baeldung on Computer Science

A Solid Guide to SOLID Principles | Baeldung

Design Patterns (refactoring.guru)

https://www.lucidchart.com/pages/

Object Oriented Programming OOPs in C# - Dot Net Tutorials

(33) Low Level System Design: MakeMyTrip Booking Management System with Arun Goel - YouTube