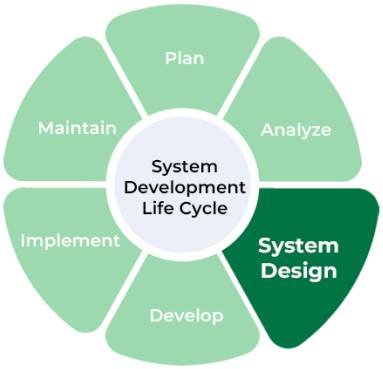
What is System Design?

- System design is the process of defining the ***overall architecture***, ***components***, ***interfaces***, and ***data flow*** of a system to meet specific requirements.
- It involves translating ***business*** and ***technical needs*** into a ***detailed blueprint*** that guides the ***development***, ***implementation***, and ***maintenance*** of the system.
- Essentially, it's about figuring out ***how all the pieces of a system fit together*** and ***how they interact to achieve a desired outcome***.



System design involves two main levels of design:

1. ***High-Level Design*** (HLD)
2. ***Low-Level Design*** (LLD)

What is High-Level Design (HLD)?

High-Level Design (HLD) is like ***planning*** the ***overall structure*** and ***layout of a building***, including the ***main components*** and ***their interactions***. It focuses on the ***high-level architecture*** of a ***system***, such as the different ***services***, ***databases***, and ***how they communicate***, without going into the detailed implementation.

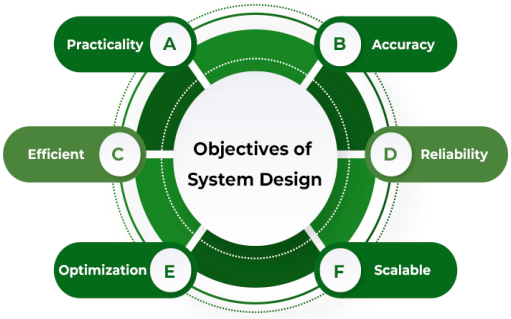
- HLD defines the ***system's architecture***, including the ***major components*** and ***how they work together***.
- It outlines ***how different parts of the system*** (services) ***communicate with each other***, including ***what data they exchange***.
- It considers the ***database design*** and ***how the system interacts with the database***.
- HLD also considers ***how the system will scale*** and ***perform under load***, including decisions about ***caching***, ***load balancing***, and *other performance optimizations*.
- HLD ***doesn't go*** into the ***specific implementation details*** of ***each component***, like the exact data structures or algorithms.

What is Low-Level Design (LLD)?

Low-Level Design (LLD) is like the ***detailed blueprint*** of a software system's components. It dives into the specifics of ***how each part of the system***, like ***classes***, ***methods***, and ***interfaces***, are ***implemented*** to achieve the overall functionality.

- Designing ***classes***, ***methods***, ***data structures***, and their ***interactions***.
- Specifying ***how to implement algorithms***, ***data structures***, and ***interfaces***.
- LLD provides a ***detailed blueprint*** that can be ***directly used by developers*** to ***implement the code***.
- LLD dives deep into the ***details of how each component is designed*** and ***interacts***, ensuring the system is built to the desired specifications.

Pillars of a good LLD:



Scalability: A scalable LLD ensures that ***the system can handle increasing workloads*** or ***user traffic without compromising performance*** or ***functionality***.

Maintainability: A maintainable LLD makes it ***easier to debug*** and ***make changes to the system*** over time. This is achieved by using ***clear*** and ***well-documented code***, following ***coding standards***, and ***minimizing complexity***.

Reusability: A reusable LLD allows ***components to be used in different parts of the system*** or ***in other projects***, ***reducing development effort*** and ***promoting code consistency***. This often involves ***designing components*** with ***well-defined interfaces*** and ***functionalities that can be easily plugged into other systems***.

Comparison of LLD, HLD and DSA:

Aspect	LLD	HLD	DSA
Purpose	Detailing the implementation	Outlining system architecture	Solving algorithmic problems
Level	Micro (code-level)	Macro (system-level)	Low-level (logic and data structure)
Focus	Classes, methods, object relations	Components, modules, integrations	Efficiency, data manipulation
Artifacts	Class diagrams, sequence diagrams	Architecture diagrams, flowcharts	Code, pseudocode, complexity analysis
Use Case	Developing specific modules	Designing entire systems	Solving specific problems or optimizing code
When	After HLD, before implementation	At the beginning of system design	During coding challenges or optimization tasks