System design(HLD&LLD)

What is System Design?

System design is the process of defining the architecture, components, modules, interfaces,

and data for a system to satisfy specified requirements. It is a phase in the software

development life cycle that involves transforming the requirements gathered during the analysis

phase into a detailed and structured design.

On the start of the project we gather all requirement in say SRS document - Software Requirement Specification.

Now Its our task to convert this requirement to a working model Now comes the Task of Architects is to design overall system - Now Thats called a high level design.What is expected out of an architect is to design the system in such a way that its not only catering the clients requirement but also doing it with right set of softwares and tool to have performance

For example - we need to store user information in a database . Now its Architects task to decide which DB to use - Relational or NOSQL (on a very high level) .

Once we are done with all components , modules and their interactions then comes the task of developers.We divide these components, modules into individual subproblems and Start getting into whats called a Low Level design and machine coding.

Now is the time to implement the HLD to working software. Here comes designing of various ER diagrams, UML diagrams, class diagrams to show case how the problem is going to solved.

**HLD:**

HLD stands for High-Level Design. It is a phase in the software development process that comes after the system design phase. High-Level Design focuses on creating an abstract representation of the system's architecture, components, and their interactions. This design level provides a broad view of the system without delving into detailed specifications.

**Example of HLD:**

Let's consider a high-level design for a basic email system:

1. **Architecture:**
   * **Client-Side:** Web and mobile interfaces for users to send, receive, and manage emails.
   * **Server-Side:** Email server handling incoming and outgoing emails, user authentication, and communication with the database.
   * **Database:** Stores user account information, emails, and metadata.
2. **Modules/Components:**
   * **User Authentication Module:** Manages user login, logout, and account information.
   * **Email Composition Module:** Allows users to compose and send emails.
   * **Inbox Module:** Handles the retrieval and display of received emails.
   * **Sent Items Module:** Displays a list of sent emails.
   * **Email Storage Module:** Manages the storage and retrieval of emails from the database.
   * **Notification Module:** Notifies users about new emails or other relevant events.
3. **Data Flow:**
   * Users interact with the web or mobile interface to compose and send emails.
   * The email server processes incoming and outgoing emails, communicates with the database for storage, and manages user sessions.
   * The inbox module retrieves and displays emails in the user's inbox.
4. **Interfaces:**
   * Web and mobile interfaces for users.
   * APIs for communication between the client-side and server-side components.
   * Database interfaces for storing and retrieving user accounts and emails.
5. **Technology Choices:**
   * **Client-Side:** HTML, CSS, JavaScript for web interfaces.
   * **Server-Side:** Node.js for the email server.
   * **Database:** SQLite for storing user data and emails.
6. **Constraints and Assumptions:**
   * **Assumption:** Users have internet access for sending and receiving emails.
   * **Constraint:** The system should comply with email communication standards.

This high-level design for an email system provides a conceptual framework for the development team before moving on to detailed design and implementation.

Top of Form

**LLD:**

LLD stands for "Low-Level Design." It is a phase in the software development life cycle that follows the High-Level Design (HLD) phase. In Low-Level Design, the focus shifts from the broad conceptual view of the system to the detailed design of individual components or modules. It involves specifying how each module will be implemented, outlining algorithms, data structures, and defining the interfaces in greater detail.

illustrate Low-Level Design (LLD) for a specific module - the "Email Composition Module."

**Low-Level Design for Email Composition Module:**

1. **Module Purpose:**
   * **Email Composition:** Allows users to compose and send emails.
2. **Detailed Module Design:**
   * **User Interface (UI):** Specify the fields in the email composition form, including "To," "Subject," and "Message Body."
   * **Internal Logic:**
     + Define the logic for handling user input, such as validating email addresses and ensuring mandatory fields are filled.
     + Specify how attachments will be handled, including uploading, storing, and linking them to the email.
   * **Data Structures:**
     + Define the data structures for storing email details, including sender, recipient(s), subject, body, and attachments.
   * **Algorithms:**
     + Specify algorithms for encoding attachments and handling file uploads.
     + Define algorithms for sending emails, including addressing, formatting, and any encryption or encoding.
3. **Interface Definitions:**
   * **Input:**
     + Specify input parameters for the module, such as recipient email addresses, subject, and message body.
     + Define how the module will handle optional inputs, like attachments.
   * **Output:**
     + Specify the format of the output, which could include a confirmation message upon successful email composition.
     + Define any error messages or notifications for unsuccessful attempts.
4. **Error Handling and Exception Handling:**
   * Specify how the module will handle errors, such as invalid email addresses, file format errors for attachments, or failures in sending emails.
   * Define exception handling mechanisms to gracefully manage unexpected situations.
5. **Resource Management:**
   * Outline how resources will be managed within the module, including memory usage during the composition process.
   * Specify how temporary data (e.g., drafts) will be stored and managed.
6. **Technology-Specific Details:**
   * Define the programming language (e.g., JavaScript), framework (e.g., React for web interfaces), and any relevant libraries for implementing the module.
7. **Code-Level Details:**
   * Provide pseudocode or code snippets to illustrate key functions within the module, such as sending emails and handling attachments.

By detailing the Email Composition Module at a low level, the development team gains a comprehensive understanding of how to implement this specific functionality. This detailed design facilitates smoother and more accurate coding during the implementation phase. Each module in the system would undergo a similar low-level design process to ensure a systematic and well-documented approach to building the entire email system.

Top of Form