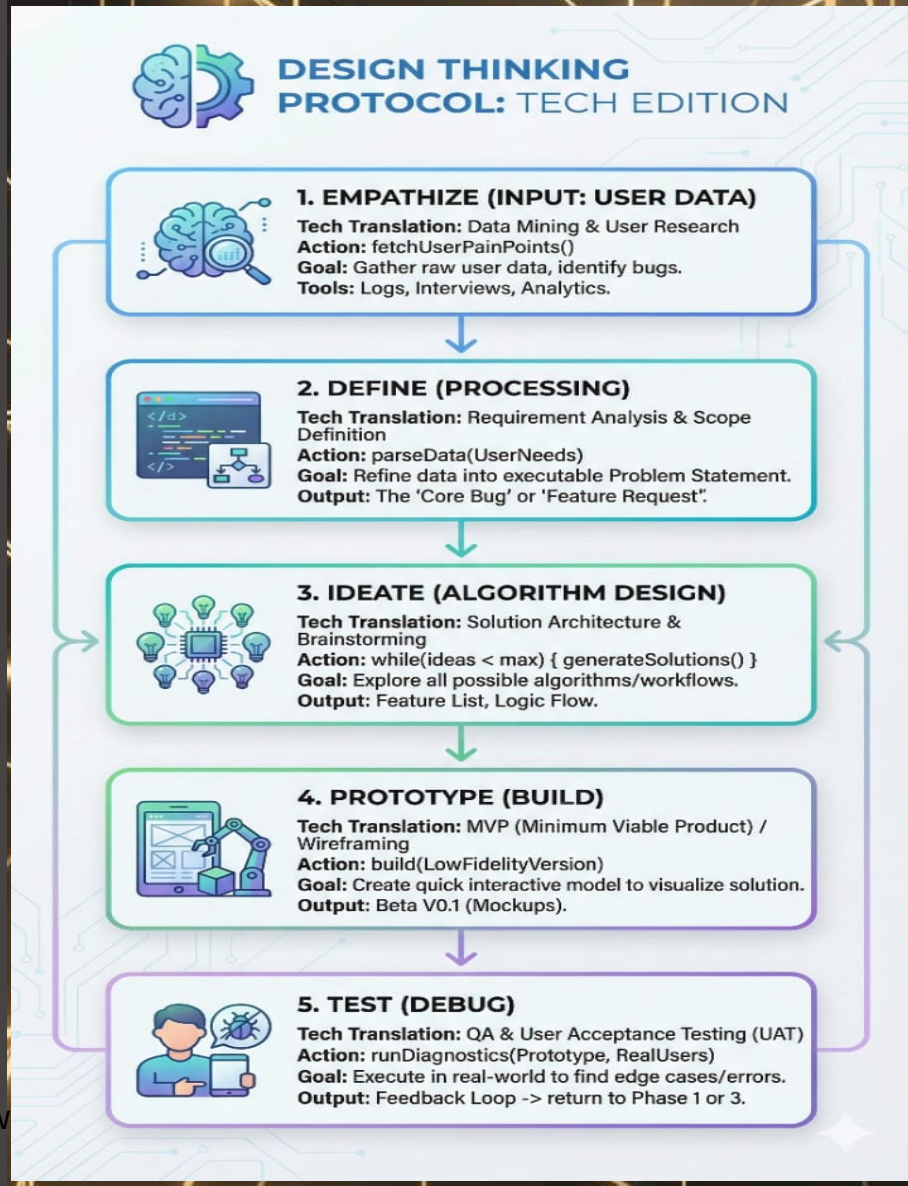
23ITT204 - COMPUTER NETWORK

UNIT 1 - INTRODUCTION AND APPLICATION LAYER

Introduction to network sockets

Introduction to Network Sockets

Network sockets are the fundamental building blocks that enable communication across theinternet.Theyserve as the invisible bridges connecting applications, devices, and services worldwide powering everything from web browsing to video streaming, instant messaging to cloud computing.



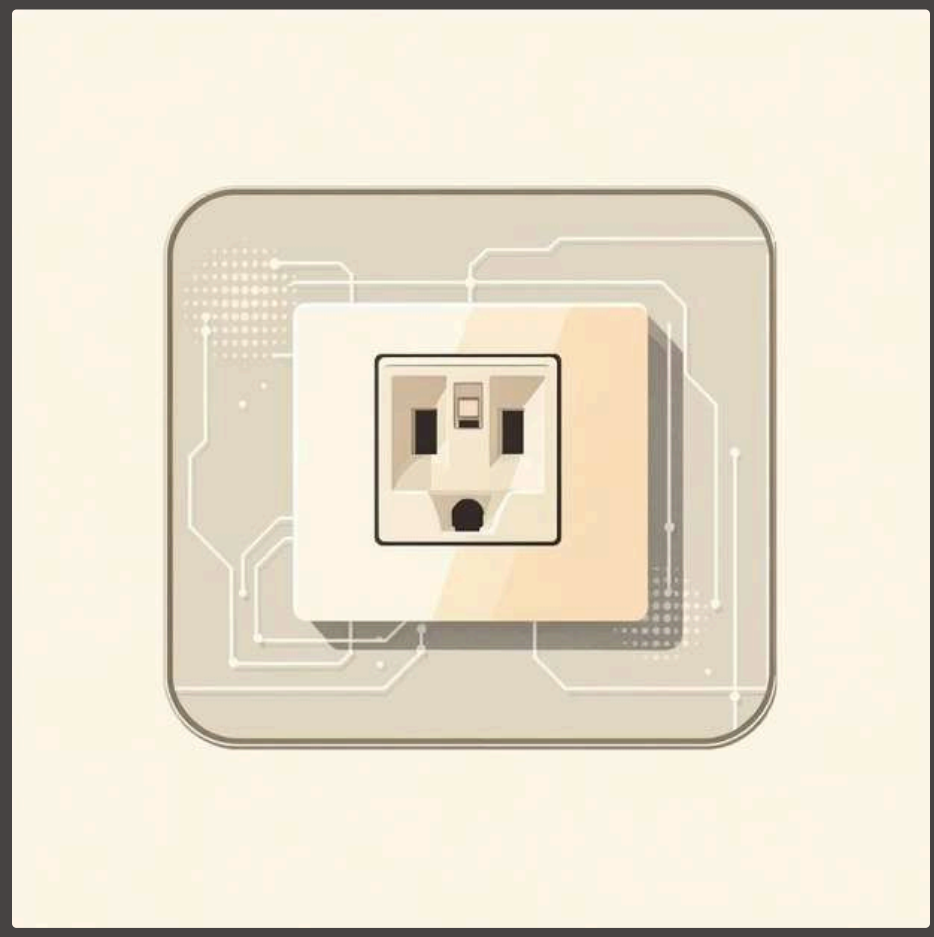
Empathize

CORE CONCEPTS

What are Sockets?

A socket is a software endpoint that establishes a bidirectional communication channel between applications running on different devices. Think of it as a virtual "plug" that connects programs across networks, enabling data exchange through standardized protocols.

Sockets provide an **Application Programming Interface (API)** that abstracts the complexity of network layers, allowing developers to send and receive data without managing low-level network details.



1

Socket Creation

Initialize a communication endpoint with specific protocol parameters

2

Connection Setup

Establish a link between client and server through binding and listening

3

Data Exchange

Send and receive information packets across the network connection

4

Connection Closure

Properly terminate the socket to free system resources

Define

PROTOCOL TYPES

Socket Types and Communication Models

Different applications require different communication characteristics. Network sockets support various protocols, each optimized for specific use cases and performance requirements.

TCP Sockets

Stream-Oriented

Transmission Control Protocol provides reliable, ordered, and error-checked delivery of data streams between applications.

- Connection-oriented with handshake
- Guaranteed delivery and ordering
- Flow and congestion control
- Ideal for web, email, file transfers

UDP Sockets

Datagram-Oriented

User Datagram Protocol offers fast, lightweight communication without reliability guarantees or connection establishment.

- Connectionless messaging
- No delivery guarantees
- Lower latency and overhead
- Perfect for gaming, video streaming, DNS

Client-Server Architecture

The most common socket communication pattern involves a **server** that listens for incoming connections and multiple **clients** that initiate requests. This asymmetric model powers the modern internet, from web servers responding to browser requests to database servers handling application queries.

Practical Socket Programming

Python TCP Server

```
import socket

server = socket.socket(

    socket.AF_INET,
    socket.SOCK_STREAM
     )
     server.bind(('localhost', 8080))
     server.listen(5)
     while True:

    client, addr = server.accept()
    data = client.recv(1024)
    client.send(b'Hello Client!')
    client.close()
```

Python TCP Client

```
import socket

client = socket.socket(

    socket.AF_INET,
 socket.SOCK_STREAM
)
client.connect(('localhost', 8080))
client.send(b'Hello Server!')
response = client.recv(1024)
print(response.decode())

client.close()
```

01

Import Socket Library

Access built-in networking functionality provided by your programming language

02

Create Socket Object

Specify address family (IPv4/IPv6) and socket type (TCP/UDP)

03

Bind and Listen (Server)

Associate socket with IP address and port, then wait for connections

04

Connect (Client)

Establish connection to server using destination IP and port

05

Exchange Data

Use send() and recv() methods to transmit information bidirectionally

06

Close Connection

Release socket resources and terminate the network connection gracefully

Key Takeaway: Socket programming follows a consistent pattern across languages. Master the fundamentals4creation, binding, connecting, data exchange, and closure4and you'll be able to build networked applications in any environment. Start experimenting with simple client-server programs to solidify your understanding!