

UNIT 2

Issues of Web Technology

(3 Hours) — Web Technology (CACS205)

2.1 Architectural Issues of the Web Layer

A web application is a distributed system: the user works in a browser, the request travels over the Internet using HTTP, and a server produces the response. Designing such a system means deciding *where* each responsibility lives. Every application, however small, performs three kinds of work:

Logical Layer	Responsibility	Typical Technology
Presentation layer	Displays information and captures user input (UI).	HTML, CSS, JavaScript, browser
Application (business) layer	Implements rules, calculations, workflow, validation, security decisions.	Servlets/JSP, PHP, ASP.NET, Node.js
Data layer	Stores, retrieves and protects persistent data.	MySQL, Oracle, SQL Server, files

The main architectural issues that a designer must resolve are:

- **Scalability** — can the system serve more users by adding hardware or servers?
- **Performance** — response time, network round trips, caching, load on the database.
- **Availability and reliability** — the ability to keep running despite failures (redundancy, fail-over).
- **Security** — authentication, authorisation, protecting data in transit and at rest.
- **Maintainability** — separating concerns so that one layer can change without rewriting others.
- **Interoperability** — different platforms, languages and vendors must work together through open standards (HTTP, XML, SOAP/REST).
- **State management** — HTTP is *stateless*, so sessions, cookies or databases are needed to remember users.
- **Cost and deployment** — hardware, licences, installation and upgrade effort for clients and servers.

*Note: The words **layer** and **tier** are often used together, but a layer is a logical division of code, whereas a tier is a physical division — a separate machine or process where a layer runs.*

2.2 Tier Technology

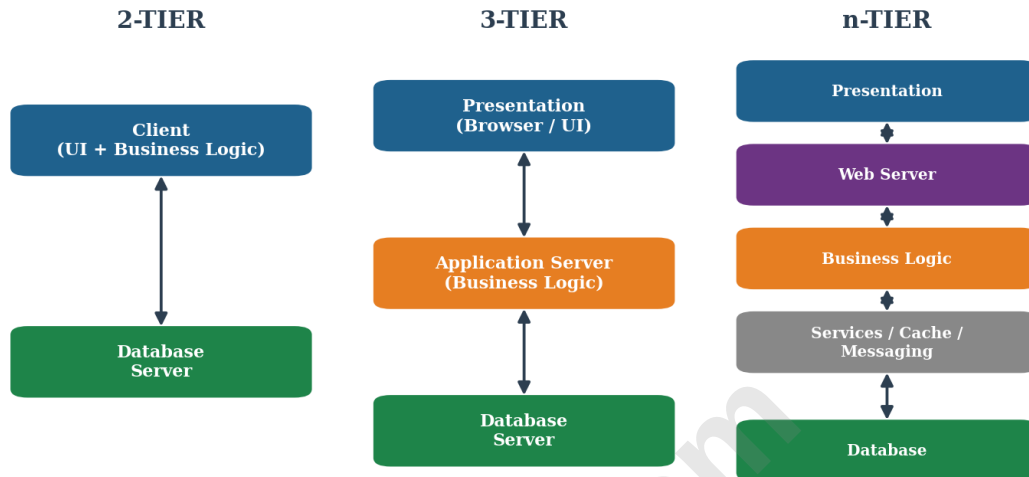


Figure 2.1 — 2-tier, 3-tier and n-tier architectures

2.2.1 Two-Tier Architecture

In a **2-tier (client–server)** architecture the client machine runs the user interface *and* the business logic (a “fat” or “thick” client) and talks directly to a database server. A desktop application connecting to a shared SQL Server is a typical example.

Advantages	Disadvantages
Simple to design and develop quickly	Business logic duplicated on every client; hard to update
Fast for small, LAN-based user groups	Poor scalability — each client holds a database connection
Rich, responsive user interface	Weak security — clients access the database directly
Fewer components to administer	Tight coupling of UI and logic reduces reuse

2.2.2 Three-Tier Architecture

A **3-tier** architecture inserts a middle **application server** between the client and the database. The client becomes a “thin” client (usually just a web browser) that handles only presentation. Business logic runs centrally on the application server, which alone talks to the database.

Tier	Role	Example
1. Presentation (client) tier	User interface, input, display	Browser with HTML/CSS/JavaScript
2. Application (middle) tier	Business logic, transaction control, security	Tomcat / IIS / Apache + PHP
3. Data tier	Persistent storage and queries	MySQL, Oracle

Advantages	Disadvantages
Thin clients — no installation, easy updates on the server only	More complex to design, deploy and test
Better scalability: add or cluster application servers	Extra network hop can add latency
Better security: the database is hidden behind the middle tier	Requires skilled administration of several servers
Reusable business logic; tiers developed independently	Higher initial hardware and software cost

2.2.3 n-Tier Architecture

An **n-tier (multi-tier)** architecture generalises the idea: the application is split into n separately deployable tiers, where n is three or more. Additional tiers may include a dedicated **web server** tier, a **business-logic/service** tier, a **caching** tier, a **messaging** tier, an **integration/API gateway** tier and one or more **database** tiers. Enterprise Java (Java EE), .NET and modern micro-services applications follow this style.

- Each tier can be **scaled, replaced or clustered independently**.
- Load balancers distribute requests among identical servers of the same tier.
- Improves fault tolerance: a failed server is replaced without stopping the whole system.
- Complexity, monitoring and network latency increase with every added tier.

2.2.4 Comparison

Feature	2-Tier	3-Tier	n-Tier
Number of tiers	2	3	3 or more
Client type	Thick / fat	Thin (browser)	Thin (browser/mobile)
Business logic location	On the client	Application server	One or more service tiers
Database access	Direct from client	Only via middle tier	Only via data-access tier

Feature	2-Tier	3-Tier	n-Tier
Scalability	Low	Medium–High	Very high
Security	Low	Good	Best (many controls)
Maintenance	Hard (update every client)	Easy (update server)	Easy but many parts
Cost / complexity	Low	Medium	High
Typical use	Small office/LAN systems	Web applications	Large enterprise, cloud systems

Example 2.1 — Choosing an architecture

A college wants an online admission system used by thousands of students across the country, with sensitive personal data. Which architecture is most suitable and why?

Solution

Three-tier (or n-tier). Students need only a browser (thin client, nothing to install); business rules such as fee and eligibility checks run on the application server, and the database is never exposed directly. Additional web servers can be added behind a load balancer as the number of applicants grows. A 2-tier design would need software installed on every student's PC and would expose the database.

Key Takeaways — Unit 2

- ✓ Every web application has presentation, business-logic and data layers; a tier is a physical deployment of a layer.
- ✓ Key issues: scalability, performance, security, availability, maintainability, interoperability, state.
- ✓ 2-tier = thick client + database server: simple but hard to maintain and scale.
- ✓ 3-tier = browser + application server + database: the standard web design; secure and scalable.
- ✓ n-tier = more specialised tiers (web, service, cache, messaging, data): maximum flexibility at the cost of complexity.

— End of Unit 2 —