كلية العلوم

قســـــم الامـــــــــــن الـــــــــــــــــسيبرانـــــــــــــــــي

Department of Cyber Security

Lec 1

Web Foundations: WWW, HTTP, URL, TCP/IP, Client/Server, browsing, website classification

Subject: Web programming

Class: third

Lecturer: Asst. Lecturer Qusai AL-Durrah

The Web Is More Than Websites

Understanding the Web requires looking beyond visual interfaces. At its core, the Web represents a sophisticated system of **network infrastructure, communication protocols, addressing mechanisms, and client/server interactions**. This foundational knowledge isn't optional – it's essential.

Network + Protocol

The Web operates on layered communication standards that enable reliable data exchange across global infrastructure.

Addresses + Behavior

URLs identify resources while HTTP defines how clients request them and servers respond – a dance of structured communication.

Security Implication

In cybersecurity, you can't defend what you don't understand. Web attacks are simply *abusing normal web behavior*.

If you misunderstand how requests work, you will write incorrect code later when building HTML, JavaScript, or ASP applications. Today's focus: the Web stack from **URL** → **HTTP** → **server response**.

Lecture Objectives

By the end of this session, you will understand the foundational concepts that underpin all web programming and cybersecurity work. These objectives establish the technical vocabulary and architectural patterns you'll use throughout the course.

01

Define Web Programming

Understand what web-based applications are and how they differ from traditional desktop software.

02

Differentiate Internet vs Web

Clarify the distinction between the global network infrastructure (Internet) and the application layer service (WWW).

03

Explain Client/Server Architecture

Describe the request/response model that governs all web communication.

04

Describe HTTP Mechanics

Break down HTTP request/response basics including methods, status codes, and message structure.

05

Analyze URL Components

Deconstruct URLs into their constituent parts and understand how each element functions.

06

Position TCP/IP

Place TCP/IP protocols within the broader web communication stack and understand their role.

07

Classify Website Types

Categorize websites by environment, approach, and complexity to understand the spectrum of web applications.

What is Web Programming?

Web programming involves creating systems and applications that users access through a **web browser** using standardized web technologies. Unlike traditional desktop applications, web programs operate across a network, separating the user interface (client) from the processing logic and data (server).

Client-Side Development

Focuses on presentation and user interaction – everything the user sees and interacts with in their browser.

- Visual layout and styling
- User interface components
- Interactive behavior
- Form validation

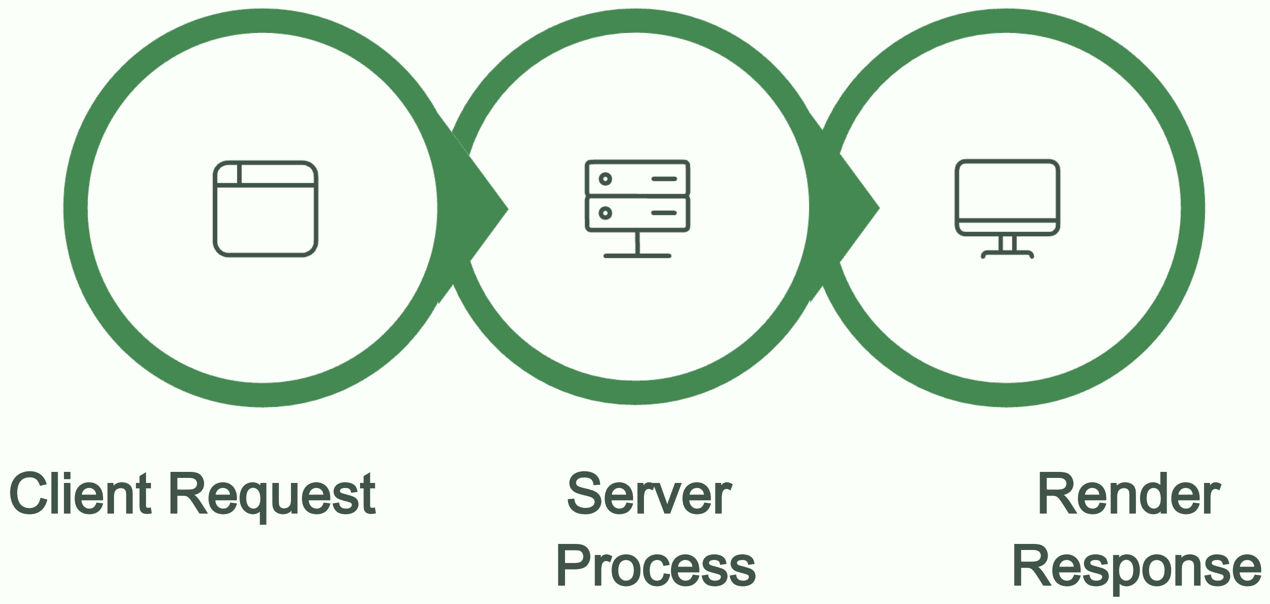
Server-Side Development

Handles business logic, data processing, authentication, and security – the invisible engine powering web applications.

- Data validation and processing
- Database operations
- Authentication and authorization
- API endpoints

This course progression: **HTML** (structure) → **CSS** (styling) → **JavaScript** (interactivity) → **ASP** (server logic) → **Database integration** (data persistence). Each layer builds on the previous, creating complete web-based systems.

Web-Based Application Concept



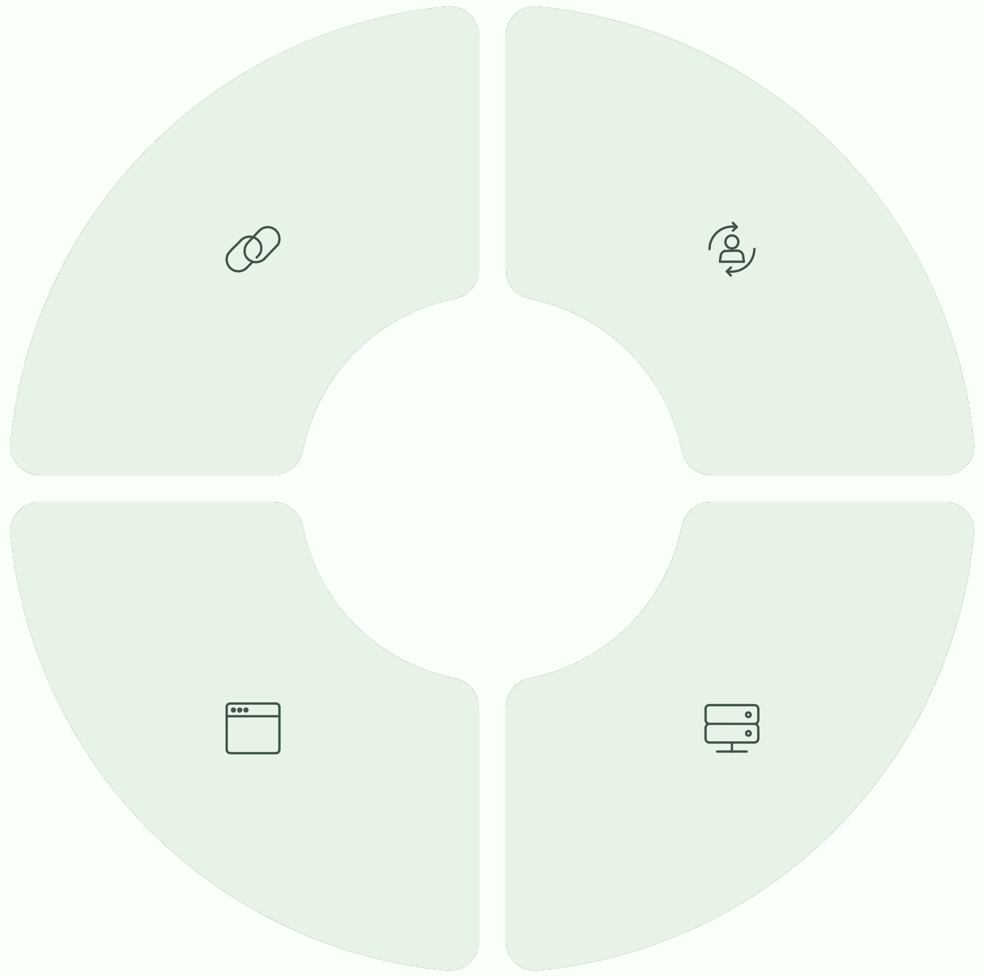
A web-based application leverages the browser-server model to deliver dynamic functionality rather than static content. The browser acts as a universal client platform, while the web server executes logic and manages data.

Web applications provide **behavior and interactivity**, not merely information pages. Users can authenticate, submit forms, search databases, manipulate data through CRUD operations (Create, Read, Update, Delete), and interact with real-time dashboards. The output format varies by design: traditional web apps return HTML pages for human viewing, while modern APIs return structured data (typically JSON) for consumption by other programs.

Example distinction: A static departmental information page displays fixed content. A student portal web application enables login, course registration, grade checking, and schedule management – all requiring server-side processing and database interaction.

WWW: What the Web Actually Is

The World Wide Web (WWW) represents a global system of **interconnected, hyperlinked resources** accessed through web browsers. It's important to understand that the Web is an *application-layer service*, not the physical network infrastructure itself.



URLs

Uniform Resource Locators provide standardized addresses for identifying and locating resources across the Web.

HTTP/HTTPS

Communication protocols defining how browsers request resources and servers respond, with HTTPS adding encryption.

Browsers

Client applications that request, receive, and render web content for users on various devices.

Web Servers

Computers running server software that host resources and respond to client requests continuously.

Key concept: "Web = a service running over the Internet." The Web couldn't exist without Internet infrastructure, but the Internet hosts many services beyond the Web (email, file transfer, streaming, etc.). The WWW specifically refers to the hyperlinked document system accessed via HTTP/HTTPS.

Internet vs Web (WWW)

Students often confuse these terms, but they represent fundamentally different layers of technology. Understanding this distinction is essential for both technical accuracy and exam preparation.

Internet

- **Physical + logical infrastructure:** The global network of interconnected networks
- **IP addressing and routing:** Systems for identifying devices and directing traffic
- **TCP/IP protocols:** Rules governing reliable data transmission
- **Multiple services:** Email, FTP, VoIP, gaming, streaming, and more

World Wide Web

- **Application-layer service:** One of many services using Internet infrastructure
- **HTTP/HTTPS protocols:** Specific to web resource retrieval
- **URLs and browsers:** Addressing and access mechanisms unique to the Web
- **Websites and web apps:** Hyperlinked resources forming an information space

> "Internet is the infrastructure; Web is a service built on it." – Remember this distinction for exams and technical discussions.

Evolution of the Web

1

Early Web (1990s)

Static HTML pages serving primarily as digital documents. Limited interactivity, mostly read-only content.

2

Dynamic Era (2000 s)

Server-generated content using technologies like PHP, ASP, JSP. Databases power personalized experiences.

3

Web 2.0 (2010s)

Rich client-side interactivity, AJAX, social platforms, user-generated content. Web becomes a platform for applications.

4

Modern Web (Present)

Web apps, RESTful APIs, cloud services, single-page applications, progressive web apps, microservices architecture.

This evolution reveals an important pattern: **increased complexity demands greater attention to security and correctness**. Early static pages had limited attack surfaces. Modern web applications handle sensitive data, financial transactions, personal information, and critical business operations. Each layer of functionality introduces new vulnerabilities that developers and security professionals must understand and mitigate.

As web systems grow more sophisticated, the gap between "it works" and "it works securely" becomes a critical concern for Computer Security professionals.

Internet Service Provider (ISP)

Your Internet Service Provider acts as the gateway connecting your device to the global Internet. Whether you're accessing the Web from home, mobile, or university networks, an ISP handles the routing of your traffic to destination servers and back.

Connectivity Provider

ISPs maintain the physical infrastructure and network routing that enable your devices to communicate with servers worldwide.

Traffic Routing

When you request a website, your ISP routes your packets through Internet backbone networks to reach the destination server.

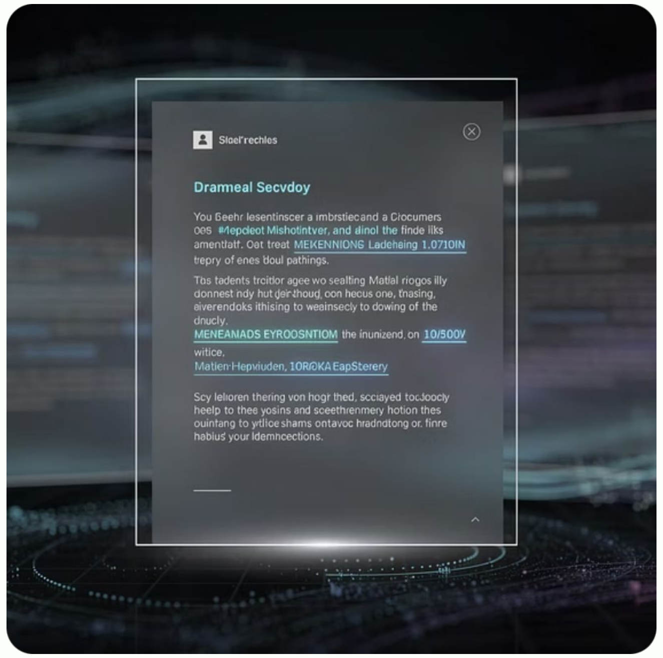
Security Consideration

Without encryption, your ISP can potentially observe your web traffic. HTTPS protects content in transit from inspection.

Cybersecurity note: HTTPS encryption matters even on trusted networks like campus Wi-Fi. Without it, anyone with network access (including compromised routers or malicious actors) can intercept sensitive data like passwords, session tokens, and personal information. Always verify HTTPS connections when handling sensitive operations.

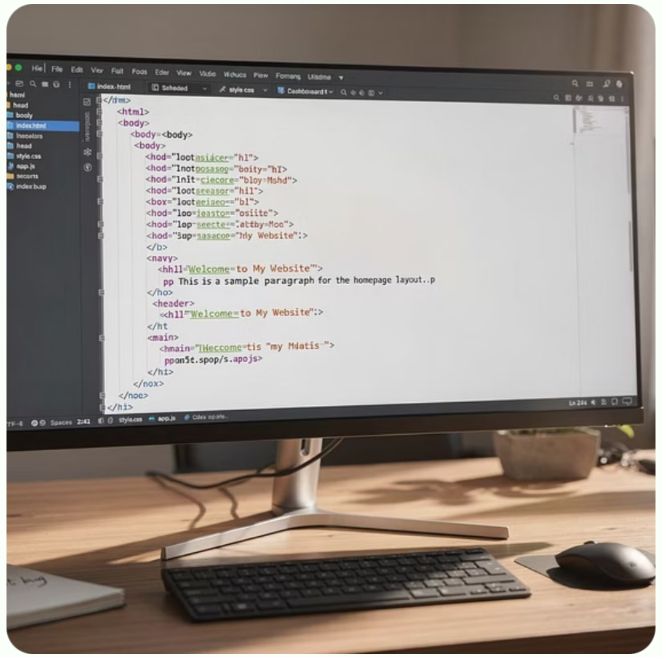
Core Web Concepts

Three fundamental terms form the building blocks of web architecture. Understanding these precisely prevents confusion as you progress through more complex topics.



Hypertext

Text containing embedded links to other resources, enabling non-linear navigation. This linking mechanism is the "web" in World Wide Web – documents connected by clickable references forming a vast information network.



Web Page

A single resource (typically an HTML document) identified by a unique URL. Web pages may include text, images, multimedia, and scripts that execute in the browser.

Website

A collection of related web pages and resources hosted under a common domain name, often sharing navigation, branding, and purpose. Websites range from single-page sites to massive applications with thousands of interconnected resources.

The hyperlink concept revolutionized information access. Before the Web, finding related information required manual searching through separate documents or databases. Hyperlinks create an interconnected information space where users navigate intuitively by following related concepts.

Web Page Addresses & URLs

Every resource on the Web requires a unique identifier so browsers can locate and request it. The Uniform Resource Locator (URL) provides this standardized addressing format, enabling consistent resource identification across the entire Web.

1

Unique Identification

URLs ensure each resource has an unambiguous address, preventing conflicts and enabling reliable access.

2

Standardized Format

The URL specification defines a uniform structure that all browsers and servers understand and process consistently.

3

Request Initiation

When you click a link or enter a URL, the browser parses it to determine where to send the HTTP request and what to request.

Every interaction on the Web begins with a URL. Clicking a link, submitting a form, loading an image – each triggers a URL-based request. Developer tools in modern browsers reveal this constant URL activity, showing every resource request with its full URL, method, status, and timing. Understanding URLs deeply is essential because **web security often involves analyzing, validating, and controlling URL-based access to resources**.

URL Structure Breakdown

Let's dissect a complete URL to understand how each component functions in directing web requests. Consider this example:

```
https://example.com:443/path/page?x=10&y=20#section
```

Scheme

`https`

The protocol used for communication. Common schemes: http, https, ftp, mailto. HTTPS adds TLS encryption.

Host/Domain

`example.com`

The server's domain name, resolved to an IP address via DNS. Identifies which server to contact.

Port

`:443`

Optional. Specifies the server port (443 for HTTPS, 80 for HTTP). Usually omitted when using standard ports.

Path

`/path/page`

The resource location on the server. Resembles a file system path but may represent routing logic, not actual files.

Query String

`?x=10&y=20`

Key-value parameters sent to the server. Used in GET requests to pass data. Multiple parameters separated by &.

Fragment

`#section`

Client-side anchor identifying a location within the page. Not sent to the server – handled by the browser only.

The query string deserves special attention. When you submit a form using the GET method or pass parameters in a link, those values appear in the URL as key-value pairs. For example, `search?q=security&filter=recent` passes `q=security` and `filter=recent` to the server. This visibility makes query strings convenient for bookmarking and sharing but inappropriate for sensitive data like passwords.

HTTP: The Web's Communication Protocol

HyperText Transfer Protocol (HTTP) defines the language browsers and servers use to communicate. It's an application-layer protocol specifying the format and sequence of messages exchanged during web interactions.

Request→ Response Model

Every web interaction follows a simple pattern: the client initiates by sending an HTTP request, and the server replies with an HTTP response. This stateless protocol treats each request independently.

HTTP Request contains:

- Method (GET, POST, etc.)
- Target URL/path
- Headers (metadata)
- Optional body (data payload)

Message Structure

HTTP messages are text-based and human-readable (though binary content can be transported). This simplicity aids debugging and understanding.

HTTP Response contains:

- Status code (200, 404, 500, etc.)
- Headers (content type, caching, etc.)
- Body (HTML, JSON, images, etc.)

> "HTTP defines the language; TCP/IP delivers it." HTTP provides the message format and meaning, while lower-layer protocols handle reliable transmission across networks.

HTTP Status Codes

Status codes are three-digit numbers in HTTP responses indicating the outcome of the request. They provide the server's "result message" to the browser, enabling proper handling of success, redirection, errors, and failures.

200 OK

Request succeeded. The server processed it successfully and returned the requested resource. This is the standard success response.

301/302 Redirect

Resource moved to a different URL. 301 indicates permanent relocation; 302 indicates temporary. Browsers automatically follow to the new location.

404 Not Found

Requested resource doesn't exist at the specified URL. Most common client error, often due to broken links or mistyped addresses.

500 Server Error

Server encountered an unexpected condition preventing it from fulfilling the request. Indicates a problem with server-side code or infrastructure.

Status codes are grouped by category: **2xx** (success), **3xx** (redirection), **4xx** (client errors), **5xx** (server errors). Understanding these codes is crucial for debugging web applications and interpreting network behavior in browser developer tools.

TCP/IP and Client/Server Architecture

Understanding where different technologies fit in the web communication stack prevents confusion about their roles and relationships.

Client/Server Model

An architectural pattern defining roles in networked communication:

- **Client:** Initiates requests, typically the user's browser or application
- **Server:** Waits for requests, processes them, and returns responses
- **Interaction:** Request-response cycle, with the client always initiating

This asymmetric relationship creates clear responsibilities and enables scalability (many clients, fewer servers).

TCP/IP Protocol Stack

The foundational Internet protocols handling reliable delivery:

- **IP (Internet Protocol):** Addressing and routing packets across networks
- **TCP (Transmission Control Protocol):** Reliable, ordered, error-checked delivery
- **Application Layer:** HTTP operates here, using TCP/IP services

TCP/IP ensures data reaches its destination correctly; HTTP defines what that data means.

Layer relationship: HTTP messages are application-level constructs that TCP/IP transports across networks. When you make an HTTP request, TCP establishes a connection, breaks the message into packets, sends them reliably, and reassembles them at the destination. IP handles routing those packets through intermediate networks. You work with HTTP; TCP/IP works behind the scenes.

HTTP Methods: Core Operations

HTTP methods (also called verbs) indicate the desired action on a resource. While HTTP defines many methods, two dominate web development and are essential to understand thoroughly.

GET Method

Purpose: Retrieve a resource without modifying server state (read operation)

Data transmission: Parameters appear in the URL query string, visible and bookmarkable

Characteristics: Idempotent (repeated requests produce same result), cacheable, length-limited by URL constraints

Use cases: Loading pages, search queries, filtering lists, fetching data

POST Method

Purpose: Submit data to the server for processing or creating resources

Data transmission: Parameters sent in the request body, not visible in URL

Characteristics: Not idempotent (repeated requests may create multiple resources), not cached, no practical size limit

Use cases: Form submissions, file uploads, creating records, authentication

Security clarification: GET parameters are visible in URLs, browser history, and server logs. POST hides data from the URL but does NOT encrypt it. Both methods transmit data in plain text unless HTTPS is used. Never rely on POST alone for security – always use HTTPS to protect sensitive information during transmission.

Web Browsing : The Complete Request Cycle

What actually happens when you type a URL and press Enter? Understanding this sequence reveals how all the components we've discussed work together.

Domain Resolution

Browser queries DNS (Domain Name System) to convert the domain name (e.g., example.com) into an IP address that identifies the server's network location.

Connection Establishment

Browser initiates a TCP connection to the server's IP address on the appropriate port (80 for HTTP, 443 for HTTPS). For HTTPS, TLS handshake encrypts the connection.

HTTP Request Sent

Browser constructs and sends an HTTP request containing the method (typically GET), path, headers (browser info, accepted formats, cookies), and optional body data.

Server Processing

Server receives the request, processes it (which may involve database queries, business logic, authentication checks), and generates an HTTP response with status code, headers, and content.

Response Received

Browser receives the response and begins parsing. HTML is processed, which reveals additional resources needed (CSS files, JavaScript, images, fonts).

Additional Requests

Browser makes separate HTTP requests for each additional resource referenced in the HTML. Each follows the same request-response cycle. Modern browsers parallelize these requests for efficiency.

A typical web page triggers dozens or hundreds of individual HTTP requests. Browser developer tools (Network tab) reveal every request with its method, URL, status code, size, and timing. This visibility is invaluable for debugging, performance optimization, and security analysis. In next week's lab, you'll observe this request cascade firsthand.

Classifying Websites by Environment

Websites operate in different network environments, each with distinct access patterns, security requirements, and architectural implications.

Internet (Public Web)

Accessible to anyone with an Internet connection. Public-facing websites hosted on publicly routable IP addresses. Examples: e-commerce, news, social media, marketing sites. Requires robust security due to exposure to threats worldwide.

Intranet (Internal Network)

Private network accessible only within an organization, typically behind firewalls. Uses web technologies for internal tools, employee portals, documentation, collaboration systems. Not accessible from outside the organization's network perimeter.

Extranet (Partner Network)

Controlled extension of an intranet to specific external parties – partners, suppliers, clients. Provides limited access to internal resources through authentication. Common in B2B scenarios: supply chain management, partner portals, client collaboration spaces.

Classifying Websites by General Approach

The technical approach to generating and delivering content fundamentally affects what a website can do and how it operates.

1

Static Websites

Content is pre-authored, stored as fixed HTML files, and served identically to all users. Fast and simple but cannot personalize or respond to user input dynamically. Best for content that rarely changes.

Examples: Brochure sites, documentation, landing pages

2

Dynamic Websites

Content generated by server-side code at request time, often pulling data from databases. Enables personalization, user accounts, content management, and real-time updates. Requires server-side processing.

Examples: News sites, blogs with CMS, user dashboards

3

Interactive Web Applications

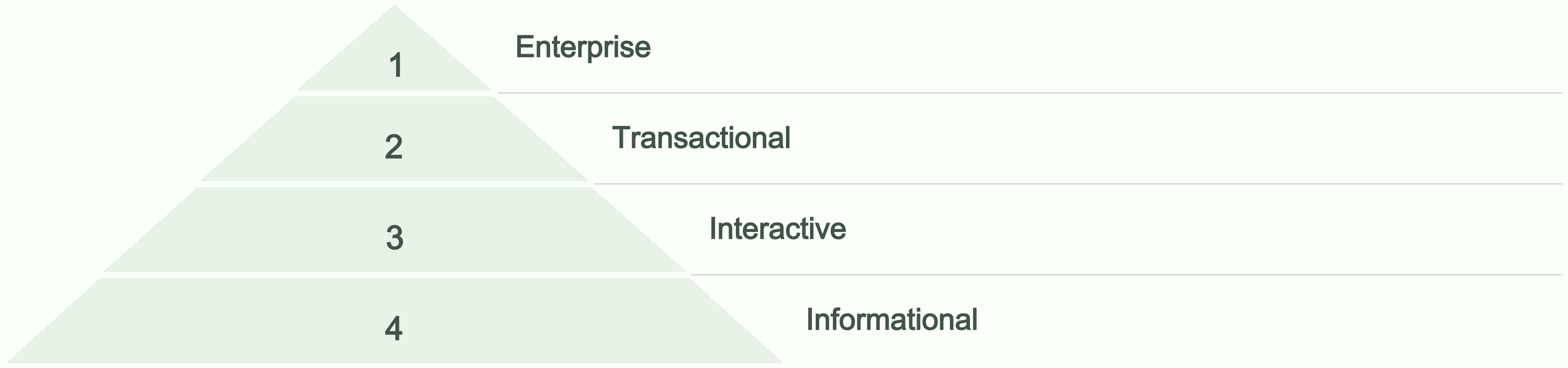
Rich client-side interactivity using JavaScript. Users manipulate data, trigger immediate responses, use complex interfaces without page reloads. Often combines dynamic server-side with interactive client-side.

Examples: Gmail, Google Docs, social platforms, SaaS tools

Modern websites often blend approaches. A site might serve static marketing pages for performance, dynamic personalized content for logged-in users, and highly interactive tools for specific features. The course progression mirrors this complexity: HTML/CSS builds static UI structure, JavaScript adds interactivity, ASP provides dynamic server-side generation, and databases enable data persistence.

Classifying Websites by Complexity Range

Websites vary dramatically in technical sophistication, from simple information displays to complex enterprise systems. Understanding this spectrum helps in planning, estimating effort, and choosing appropriate technologies.



This pyramid represents increasing complexity from bottom to top:

Website Complexity Levels Explained

Informational

Present information with minimal interaction. Static or simple dynamic content. Low complexity, limited functionality. Examples: company profiles, department pages, informational blogs.

Interactive

Users interact, submit forms, search, filter, customize views. Requires client and server logic. Examples: forums, directories, job boards, simple tools.

Transactional

Handle sensitive operations: payments, orders, account management, real-time inventory. Strong security, data integrity, audit trails essential. Examples: e-commerce, banking, booking systems.

Enterprise

Mission-critical systems integrating multiple subsystems, databases, external services. Scalability, reliability, security, compliance paramount. Examples: ERP systems, large-scale SaaS platforms, major financial systems.

As you progress in web development, you'll move up this complexity pyramid. This course provides foundations applicable at all levels, but enterprise systems require additional expertise in architecture, security, testing, and operations.

Why This Foundation Matters for Security

For Computer Security and Cybersecurity students, understanding web fundamentals isn't academic – it's operational. Web attacks exploit the normal behaviors we've studied today.

Attack Surface Understanding

Every HTTP request, URL parameter, and server response is a potential attack vector. Security professionals must understand normal web behavior to identify abnormal behavior indicating attacks.

Secure Development Practices

Common vulnerabilities stem from misunderstanding web mechanics: SQL injection from improper parameter handling, XSS from unsafe output rendering, CSRF from confused authorization models.

Forensics and Analysis

Security incident analysis requires reading HTTP logs, analyzing request patterns, reconstructing attack sequences. Understanding protocols enables effective investigation.

Defense Implementation

Implementing security controls – input validation, authentication, encryption, access control – requires deep knowledge of how requests flow and where vulnerabilities emerge.

As you build web applications in upcoming labs and assignments, apply a security mindset: validate inputs, encode outputs, use HTTPS, implement proper authentication, follow the principle of least privilege. Secure coding isn't a separate discipline – it's integrating security thinking into every development decision.

Course Trajectory: From Foundation to Implementation

Today's lecture established the conceptual foundation. The coming weeks build practical skills layer by layer, each depending on what came before.

1

Weeks 1-2: HTML

Structure and semantic markup. Building the skeleton of web pages. Understanding document hierarchy, forms, and accessibility. Next week's lab begins hands-on HTML development.

2

Weeks 3-4: CSS

Presentation and visual design. Styling HTML elements, layouts, responsive design. Separating structure from appearance. Creating professional, accessible interfaces.

3

Weeks 5-7: JavaScript

Client-side interactivity and dynamic behavior. Manipulating the DOM, handling events, validating inputs, making asynchronous requests. Building truly interactive experiences.

4

Weeks 8-11: ASP

Server-side logic and dynamic content generation. Processing forms, managing sessions, implementing authentication. Creating web applications that respond to user data and maintain state.

5

Weeks 12-14: Database Integration

Data persistence and retrieval. Connecting applications to databases, implementing CRUD operations securely. Building complete, data-driven web applications.

Each technology addresses a specific layer in the web architecture we've outlined today. HTML provides structure, CSS handles presentation, JavaScript adds client-side behavior, ASP implements server-side logic, and databases persist state. Together, they enable full-stack web development.

Homework Assignment

DUE NEXT CLASS

Assignment Details

- **Format:** 20 multiple-choice questions (A - E options)
- **Topics:** WWW, Internet vs Web, HTTP, URLs, TCP/IP, client/server, website classification

Preparation Tips

- Review slide content focusing on definitions and distinctions
- Understand *why* concepts differ, not just *what* they are
- Pay special attention to "Internet vs Web" and "GET vs POST"
- Memorize the four core HTTP status codes and their meanings

Study strategy: Don't just memorize answers. Understand the reasoning behind each concept. The exam questions are similar but not identical to homework. Conceptual understanding, not rote memorization, will serve you better.