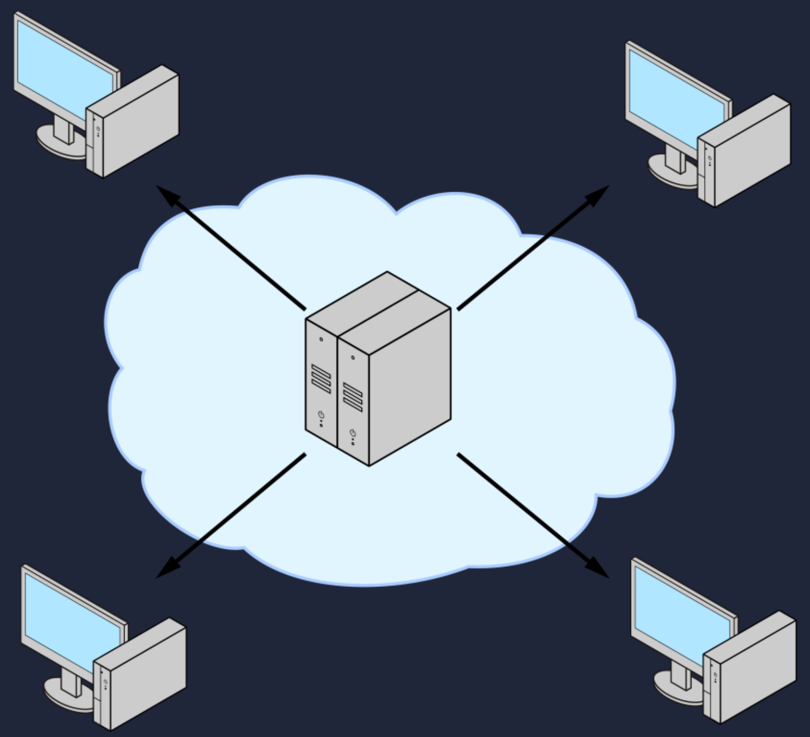
Web Programming

Week 1: Introduction

Fenerbahce University

Course Vision

- Build the competence to deliver a complete web service end-to-end.
- Understand how the browser, server, and database work together.
- Develop professional habits: clean code, testing mindset, and version control.
- Apply security and performance fundamentals from the start.



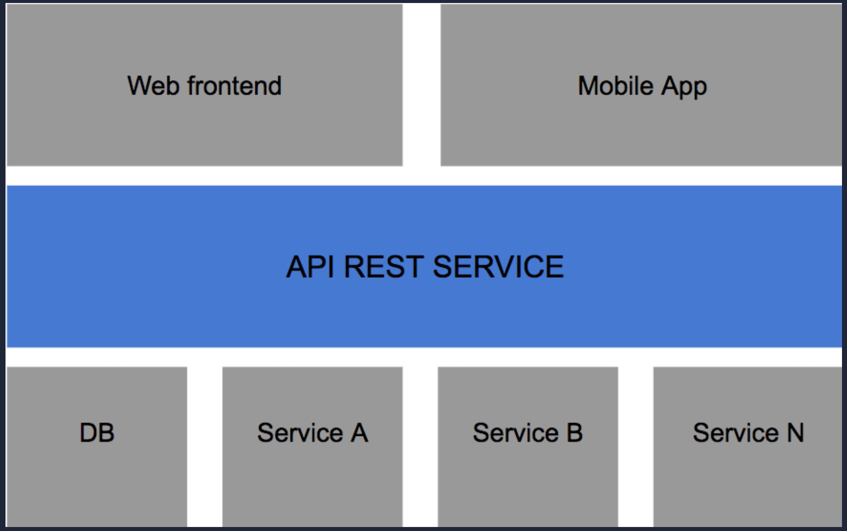
Client ⟷ Server (conceptual model)

High-Level Topics (Course Map)

- Course
    - Introduction
    - Web Fundamentals & Environment Setup
    - HTML + Modern CSS Foundations
    - PHP Basics (2 weeks)
    - MySQL & Database Design (2 weeks)
    - PHP + MySQL Connection (PDO, CRUD)
    - MVC Logic & Code Organization
    - Authentication & Authorization
    - REST API Development
    - Modern JavaScript
    - React & Modern Frontend Approach
    - Server Setup (Linux, Nginx, PHP-FPM)
    - Deployment & Security
    - Performance & Scalability

Capstone Deliverable

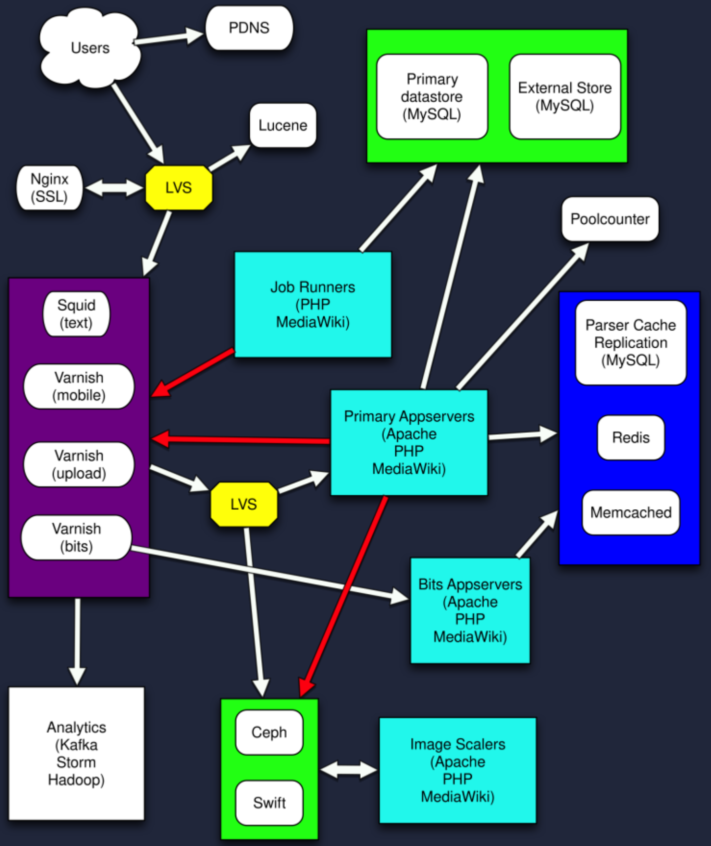
- A multi-user application with authentication and an admin panel.
- MySQL schema design with relationships, constraints, and indexes.
- PHP backend implementing business rules and data validation.
- Deployment to a Linux server with HTTPS.



Typical full-stack integration

Technology Stack Overview

- Frontend: HTML, CSS, JavaScript, React
- Backend: PHP (server-side runtime)
- Database: MySQL (relational data)
- Server: Linux + Apache + PHP



Users
PDNS
Primary datastore (MySQL)
External Store (MySQL)
Lucene
Nginx (SSL)
LVS
Poolcounter
Squid (text)
Varnish (mobile)
Varnish (upload)
Varnish (bits)
Job Runners (PHP MediaWiki)
Parser Cache Replication (MySQL)
Redis
Memcached
Primary Appservers (Apache PHP MediaWiki)
LVS
Bits Appservers (Apache PHP MediaWiki)
Analytics (Kafka Storm Hadoop)
Ceph
Swift
Image Scalers (Apache PHP MediaWiki)

Backend anchor: PHP

13-Week Roadmap

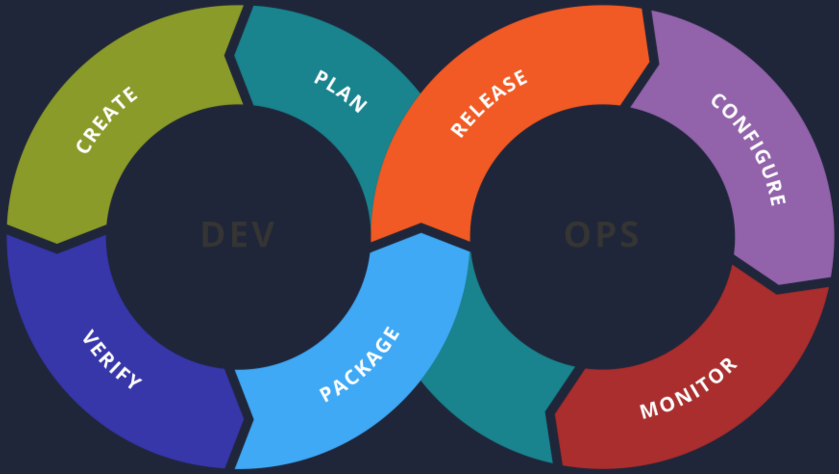
- Course Journey
- Weeks 1–3: Web fundamentals + HTML/CSS/JS foundations
- Weeks 4–5: PHP Basics I–II (forms, sessions, validation, file handling)
- Weeks 6–7: MySQL I–II (design, joins, indexing, optimization)
- Weeks 8–10: PHP+MySQL integration, MVC, authentication & security
- Weeks 11–12: REST API + React integration
- Week 13: Server setup, deployment, performance, and final delivery

Learning Outcomes

- Design a relational database schema for a real application.
- Implement secure backend features with PHP (auth, validation, CRUD).
- Build and consume REST APIs.
- Create modern responsive UIs and integrate React with APIs.
- Deploy and operate a basic production-like setup on Linux.

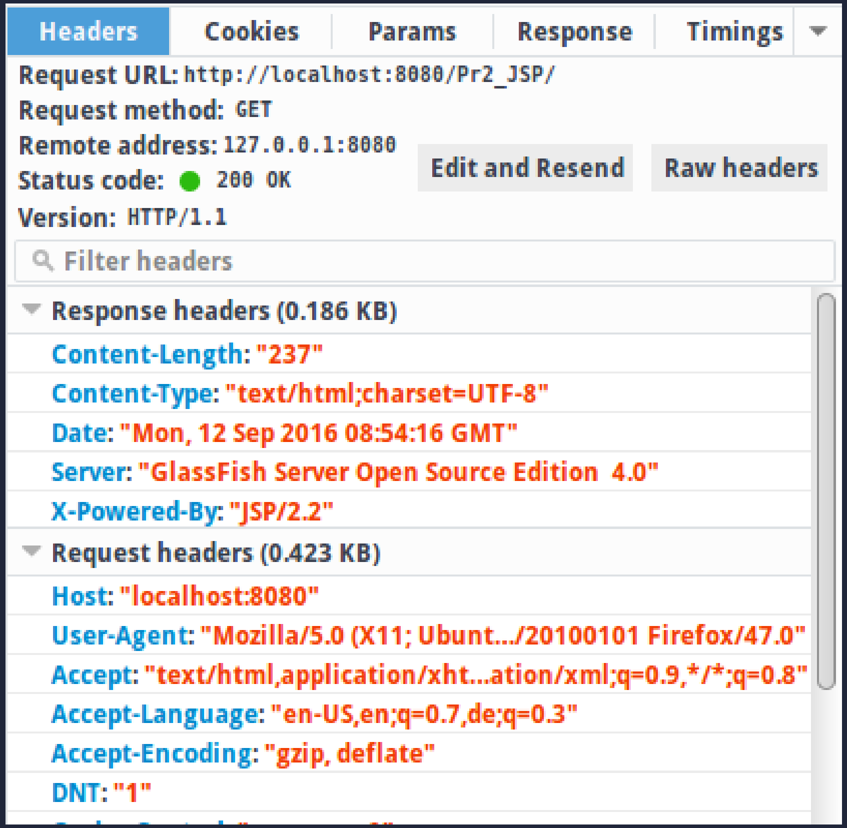
Primary Tools

- VS Code or Notepad++(editor, extensions, debugging).
- Terminal (run PHP, Composer, npm, server commands).
- Browser DevTools (inspect DOM/CSS/JS and network requests).
- MySQL Workbench or CLI for database work.



Browser DevTools (Why You Need Them)

- Inspect and modify HTML/CSS live.
- Understand JavaScript errors via stack traces and breakpoints.
- Monitor requests, responses, status codes, and payloads.
- Debug performance issues and caching behavior.



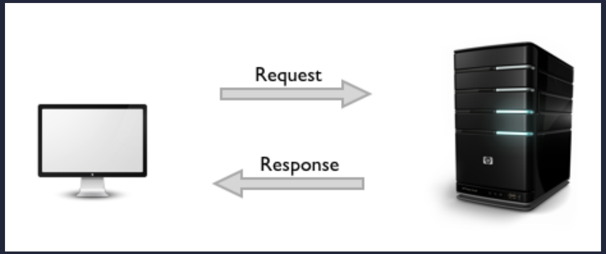
```
Request URL: http://localhost:8080/Pr2_JSP/
Request method: GET
Remote address: 127.0.0.1:8080
Status code: ● 200 OK
Version: HTTP/1.1

Response headers (0.186 KB)
  Content-Length: "237"
  Content-Type: "text/html;charset=UTF-8"
  Date: "Mon, 12 Sep 2016 08:54:16 GMT"
  Server: "GlassFish Server Open Source Edition  4.0"
  X-Powered-By: "JSP/2.2"
Request headers (0.423 KB)
  Host: "localhost:8080"
  User-Agent: "Mozilla/5.0 (X11; Ubunt.../20100101 Firefox/47.0"
  Accept: "text/html,application/xht...ation/xml;q=0.9,*/*;q=0.8"
  Accept-Language: "en-US,en;q=0.7,de;q=0.3"
  Accept-Encoding: "gzip, deflate"
  DNT: "1"
```

Chrome DevTools (Console)

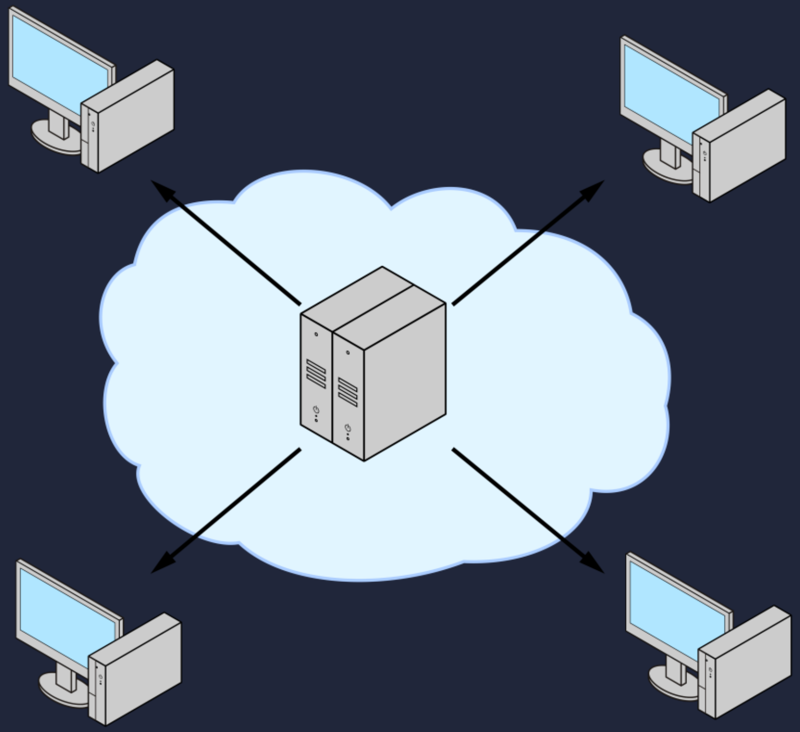
Web Fundamentals

How the Web Works (requests, responses, and networking basics)



What Is a Web Application?

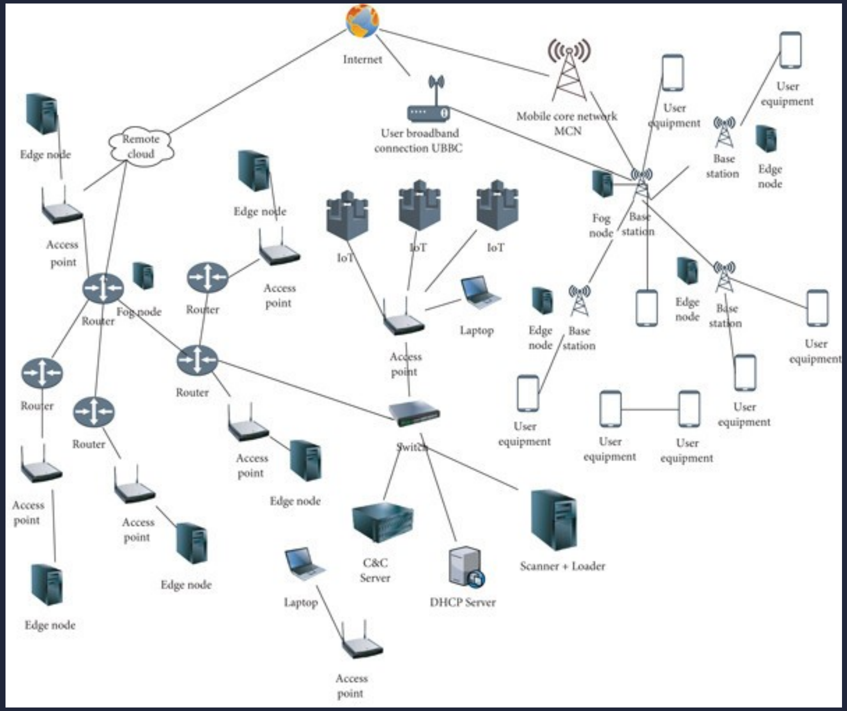
- A client-server system delivered over HTTP/HTTPS.
- Browser renders UI and executes client-side code.
- Server applies business logic and generates responses.
- Database persists structured data.



Browser as the client runtime

Client–Server Model

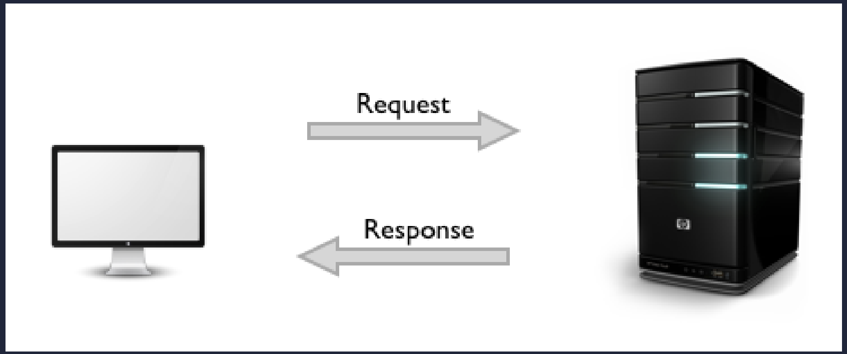
- Clients initiate requests; servers compute responses.
- HTTP is stateless by default: requests are independent.
- State is implemented via cookies, sessions, tokens, and storage.
- Understanding this model is essential for debugging.



Traditional client–server diagram

HTTP Request–Response Cycle

- Request: method, URL, headers, optional body.
- Response: status code, headers, body (HTML/JSON/etc.).
- Status codes communicate success and failure conditions.
- You will inspect these daily in DevTools.



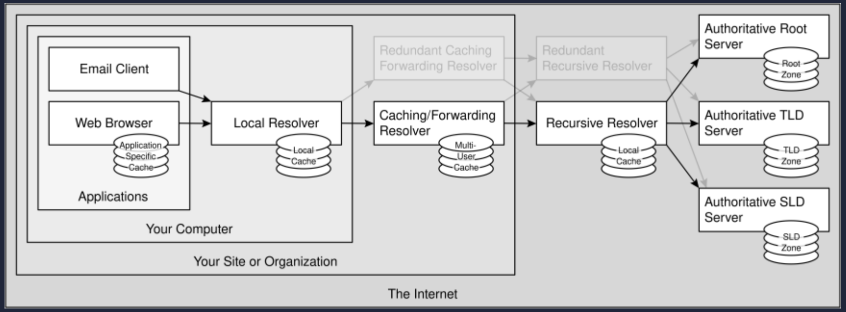
HTTP transaction (simplified)

URL Anatomy

- URL = scheme + host + path + query + fragment.
- Paths identify resources; queries parameterize them.
- Consistent URLs improve maintainability and API design.
- We will design URLs before writing code.

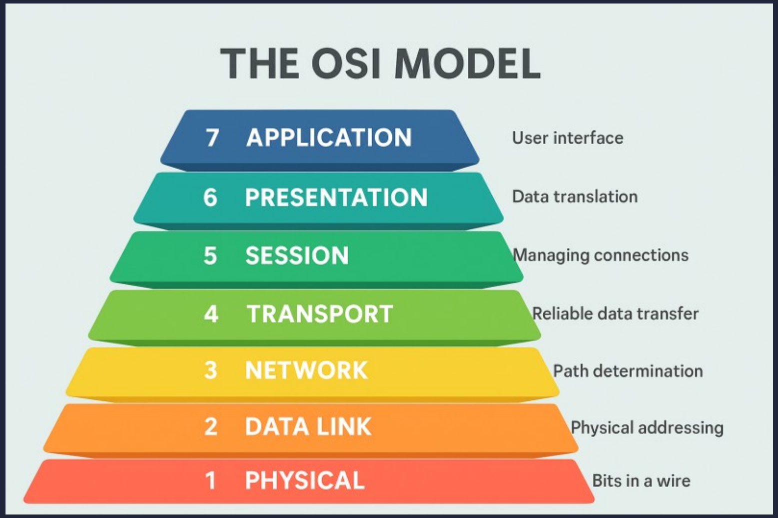
DNS: From Name to IP

- DNS resolves domain names into IP addresses.
- Resolvers cache results to improve performance.
- DNS becomes critical when deploying with custom domains.
- You will learn essential DNS records (A, CNAME).



Networking Layers (High-Level)

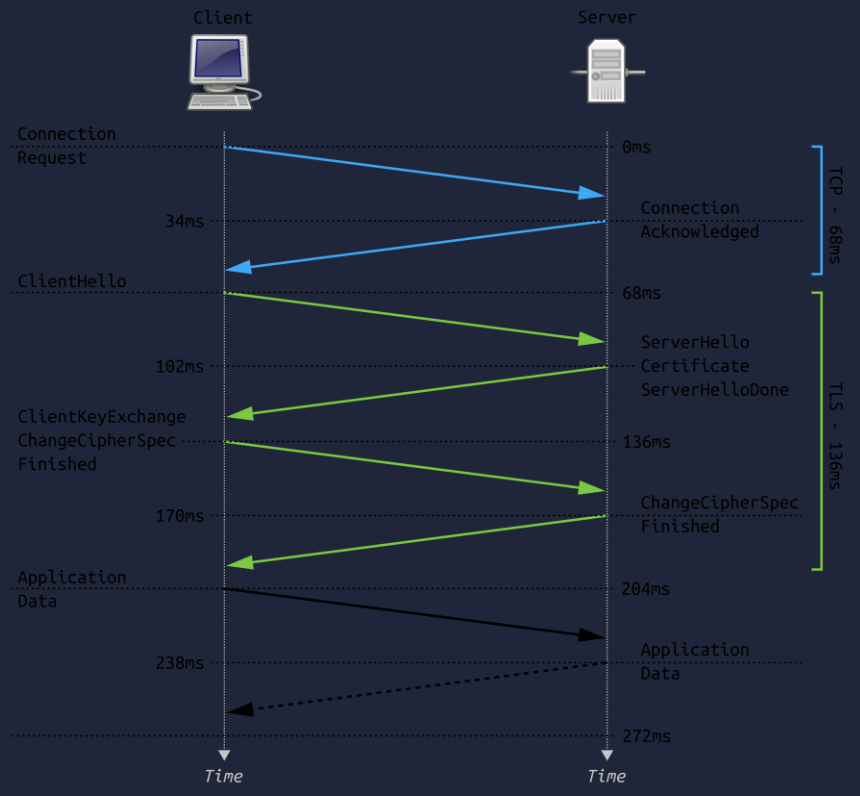
- Web apps sit on top of IP and TCP.
- TLS encrypts HTTP traffic (HTTPS).
- You do not need to memorize layers—only diagnose where problems live.
- Examples: DNS ≠ HTTP ≠ application bug.



OSI model

HTTPS & TLS

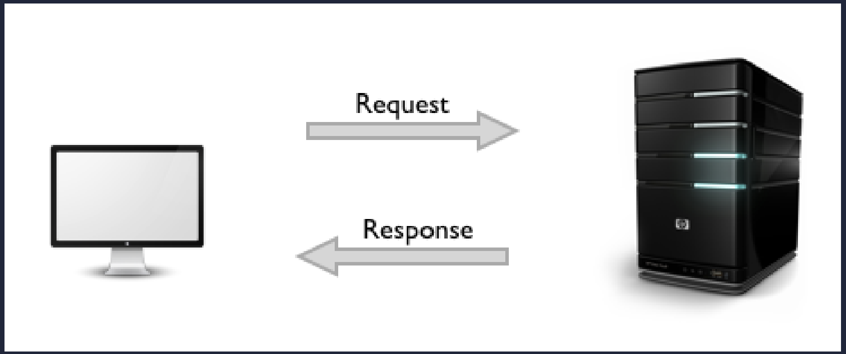
- HTTPS encrypts traffic and authenticates the server via certificates.
- Required for authentication and sensitive data.
- Modern browsers restrict or warn on insecure contexts.
- Deployment includes certificate provisioning and renewal.



Transport security

Methods, Status Codes, and Headers

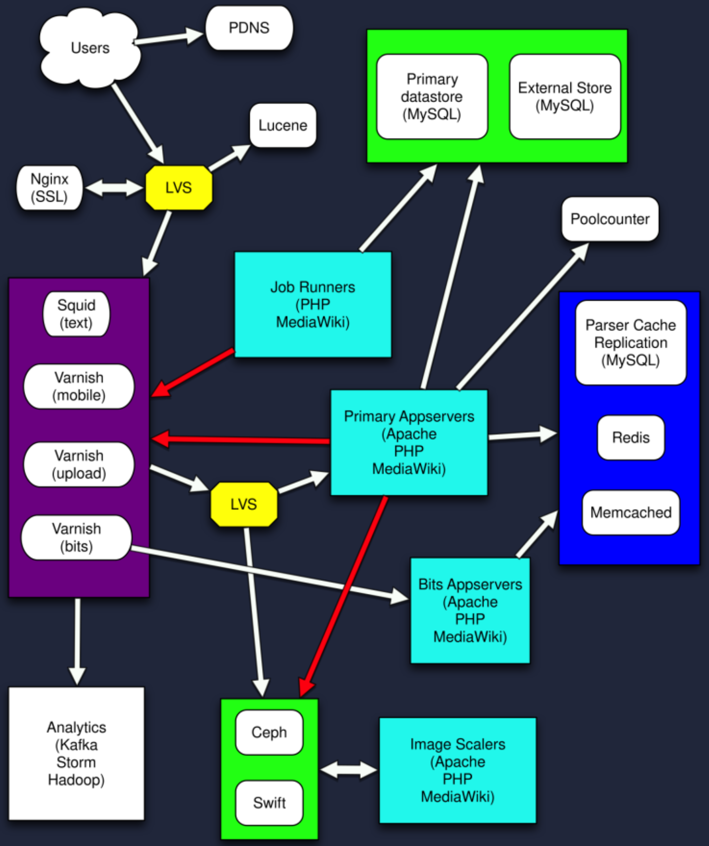
- GET/POST/PUT/PATCH/DELETE express intent.
- Status codes explain outcomes (200, 201, 400, 401, 403, 404, 500).
- Headers carry metadata (Content-Type, Authorization, Cache-Control).
- Correct HTTP usage makes your APIs and pages predictable.



HTTP fundamentals

Cookies & Sessions

- Cookies are stored in the browser and sent with requests.
- Sessions store state on the server (cookie holds only a session id).
- Secure, HttpOnly, SameSite flags reduce common risks.
- This is central to classic PHP authentication.



Users
PDNS
Lucene
Nginx (SSL)
LVS
Primary datastore (MySQL)
External Store (MySQL)
Poolcounter
Squid (text)
Varnish (mobile)
Varnish (upload)
Varnish (bits)
Job Runners (PHP MediaWiki)
Primary Appservers (Apache PHP MediaWiki)
LVS
Parser Cache Replication (MySQL)
Redis
Memcached
Bits Appservers (Apache PHP MediaWiki)
Analytics (Kafka Storm Hadoop)
Ceph
Swift
Image Scalers (Apache PHP MediaWiki)

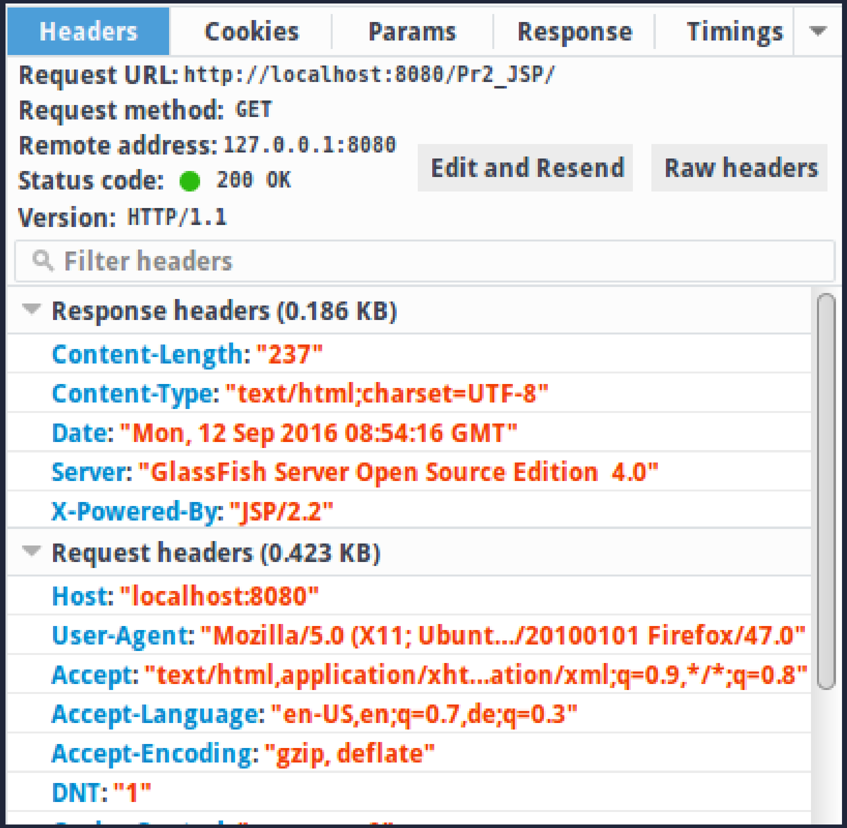
Server-side state via sessions

Caching Basics

- Caching avoids repeated work and speeds up applications.
- Browser caches static assets (CSS/JS/images).
- Server caches rendered pages or API results.
- Database performance depends heavily on indexes and caching.

Debugging Checklist

- Reproduce the issue and write down exact steps.
- Check the Network tab: request/response, status code, payload.
- Read error logs and stack traces carefully.
- Change one variable at a time and re-test.



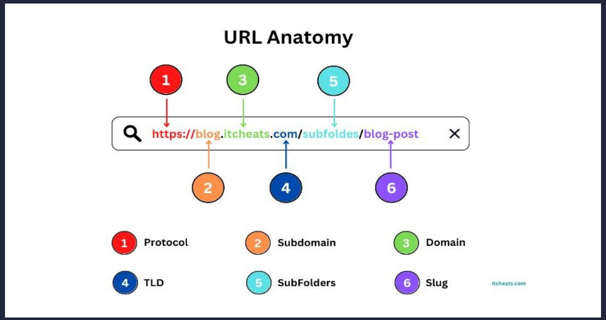
```
Request URL: http://localhost:8080/Pr2_JSP/
Request method: GET
Remote address: 127.0.0.1:8080
Status code: ● 200 OK
Version: HTTP/1.1

Response headers (0.186 KB)
  Content-Length: "237"
  Content-Type: "text/html;charset=UTF-8"
  Date: "Mon, 12 Sep 2016 08:54:16 GMT"
  Server: "GlassFish Server Open Source Edition  4.0"
  X-Powered-By: "JSP/2.2"
Request headers (0.423 KB)
  Host: "localhost:8080"
  User-Agent: "Mozilla/5.0 (X11; Ubunt.../20100101 Firefox/47.0"
  Accept: "text/html,application/xht...ation/xml;q=0.9,*/*;q=0.8"
  Accept-Language: "en-US,en;q=0.7,de;q=0.3"
  Accept-Encoding: "gzip, deflate"
  DNT: "1"
```

DevTools supports systematic debugging

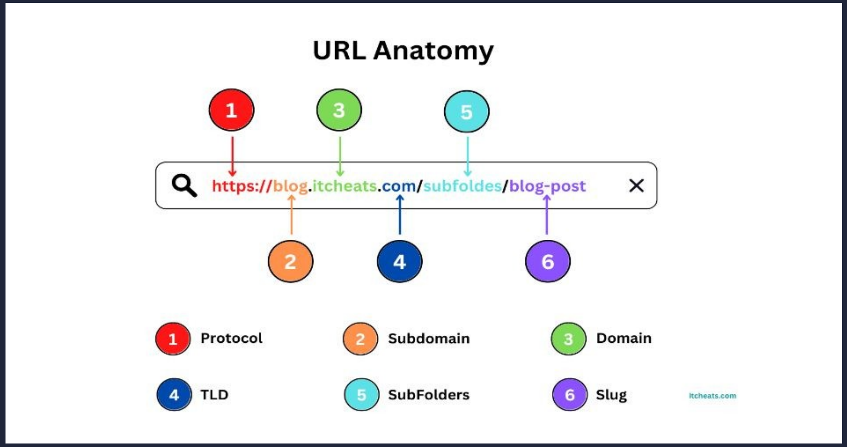
Frontend Foundations

HTML, CSS, JavaScript and React overview



HTML: Structure and Semantics

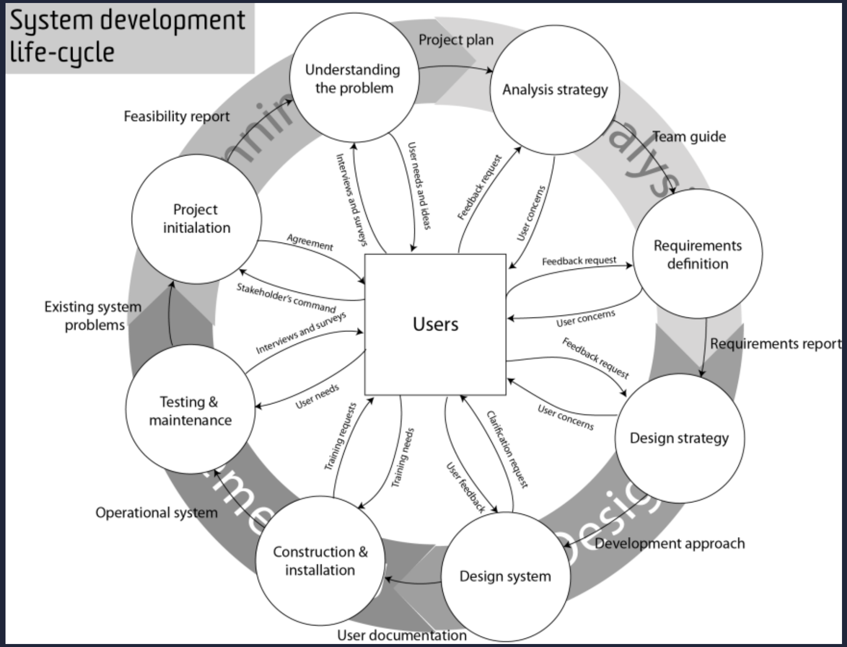
- HTML defines the structure of documents and UI elements.
- Semantic markup improves accessibility and maintainability.
- Forms provide user input that the server processes.
- Our baseline UI is server-rendered HTML from PHP.



HTML = structure

Modern CSS: Layout Systems

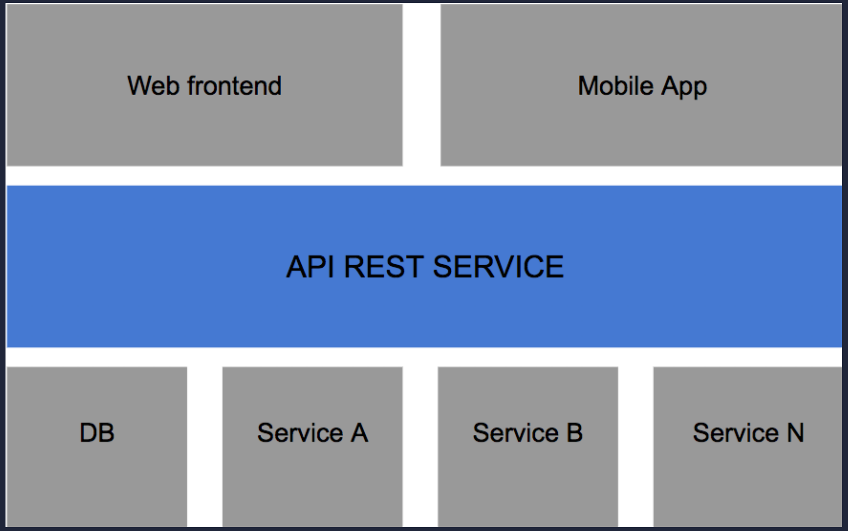
- Flexbox for one-dimensional layout (row OR column).
- Grid for two-dimensional layout (rows AND columns).
- Responsive design is non-optional for modern web services.
- We will practice readable CSS architecture and naming.



CSS = presentation

JavaScript: Interactivity and Networking

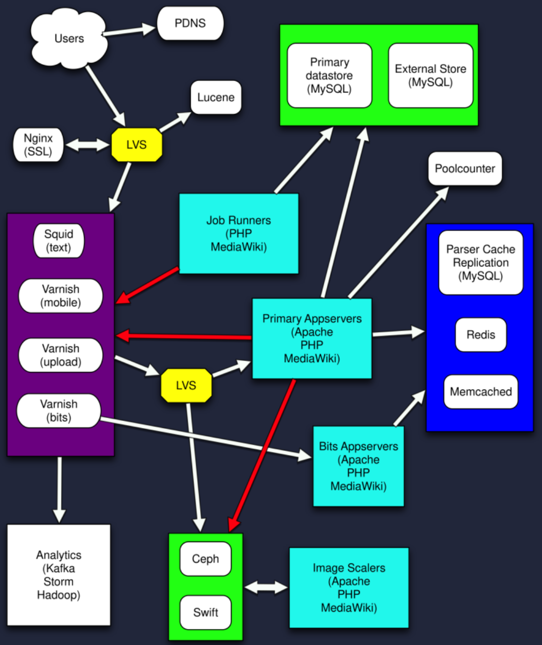
- DOM manipulation and event handling are core skills.
- fetch + async/await for calling APIs and updating UI.
- JSON is the default data format for APIs.
- We will structure JS code to remain maintainable.



JavaScript = behavior

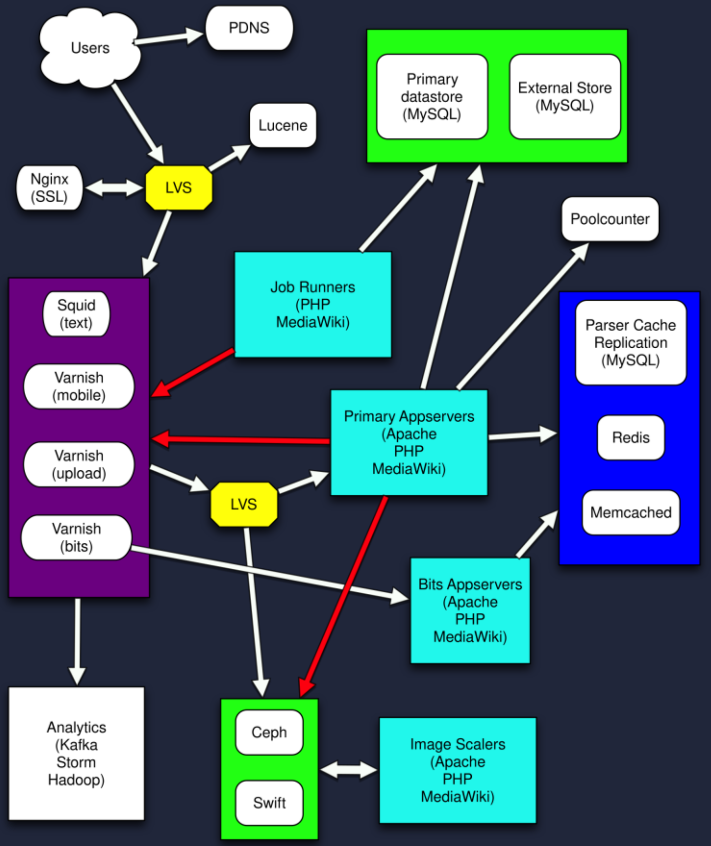
Backend Foundations

PHP request lifecycle, application structure, and APIs


Users
PDNS
Lucene
Nginx (SSL)
LVS
Primary datastore (MySQL)
External Store (MySQL)
Poolcounter
Job Runners (PHP MediaWiki)
Squid (text)
Varnish (mobile)
Varnish (upload)
Varnish (bits)
Primary Appservers (Apache PHP MediaWiki)
LVS
Parser Cache Replication (MySQL)
Redis
Memcached
Bits Appservers (Apache PHP MediaWiki)
Analytics (Kafka Storm Hadoop)
Ceph
Swift
Image Scalers (Apache PHP MediaWiki)

Backend Responsibilities

- Receive HTTP requests and route them to handlers.
- Validate inputs and enforce business rules.
- Read/write data through MySQL.
- Generate responses (HTML pages or JSON for APIs).

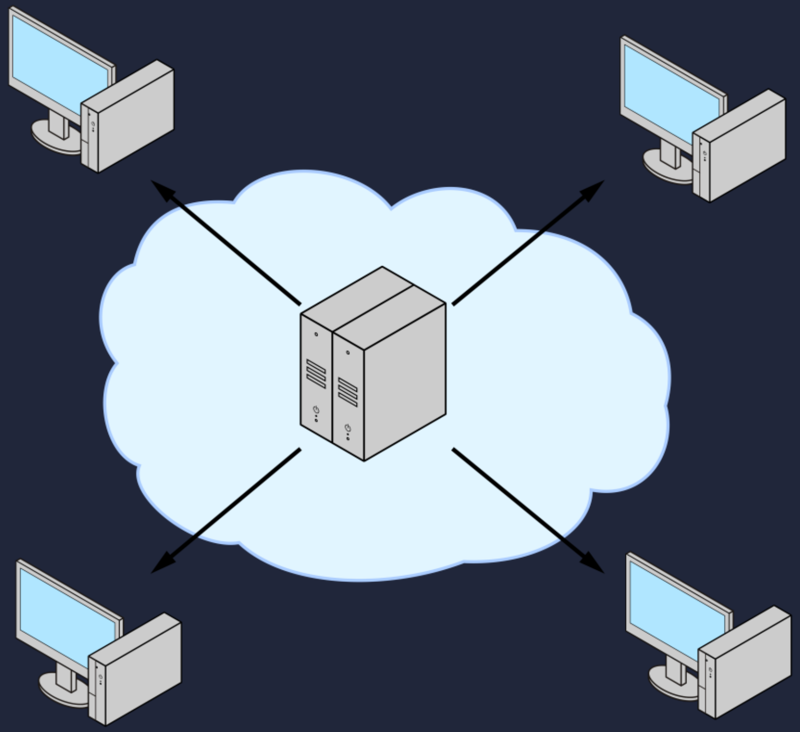


Users
PDNS
Primary datastore (MySQL)
External Store (MySQL)
Lucene
Nginx (SSL)
LVS
Poolcounter
Squid (text)
Varnish (mobile)
Varnish (upload)
Varnish (bits)
Job Runners (PHP MediaWiki)
Parser Cache Replication (MySQL)
Redis
Memcached
Primary Appservers (Apache PHP MediaWiki)
LVS
Bits Appservers (Apache PHP MediaWiki)
Analytics (Kafka Storm Hadoop)
Ceph
Swift
Image Scalers (Apache PHP MediaWiki)

Server-side processing

PHP in the Web Request Lifecycle

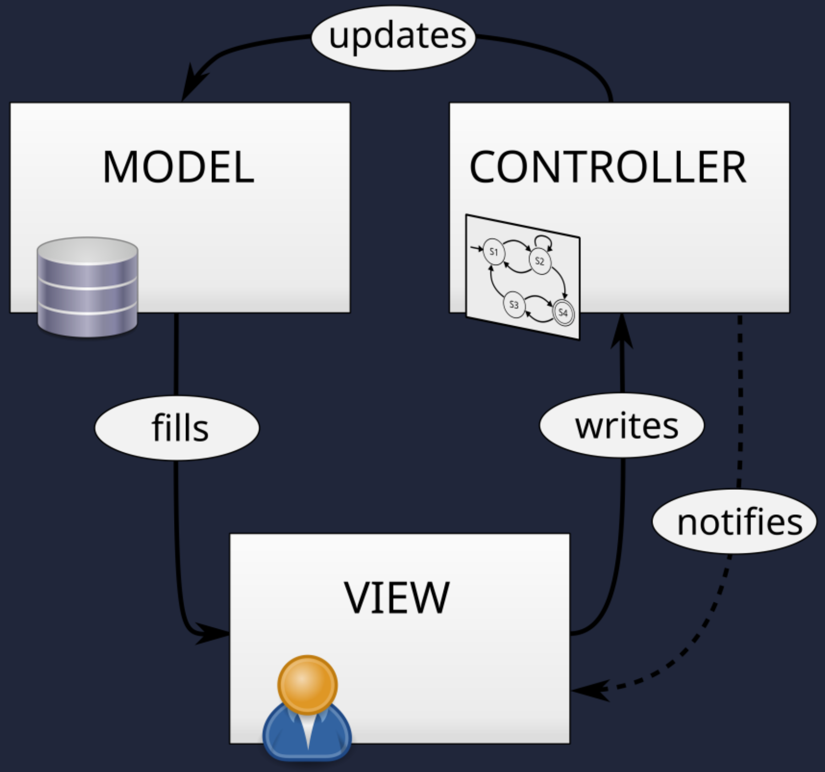
- A web server forwards a request to PHP-FPM.
- PHP executes code, often reading/writing the database.
- PHP returns HTML or JSON to the web server.
- The server sends the response back to the browser.



Request flow across components

MVC Logic & Code Organization

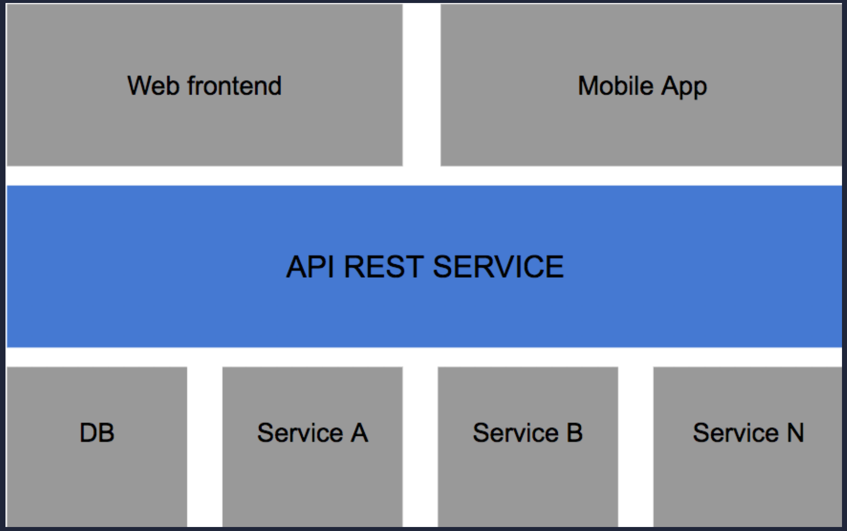
- Model: data access and domain logic.
- View: UI templates for server-rendered pages.
- Controller: request handling and orchestration.
- Separation improves testability and maintainability.



Model–View–Controller

REST APIs with PHP

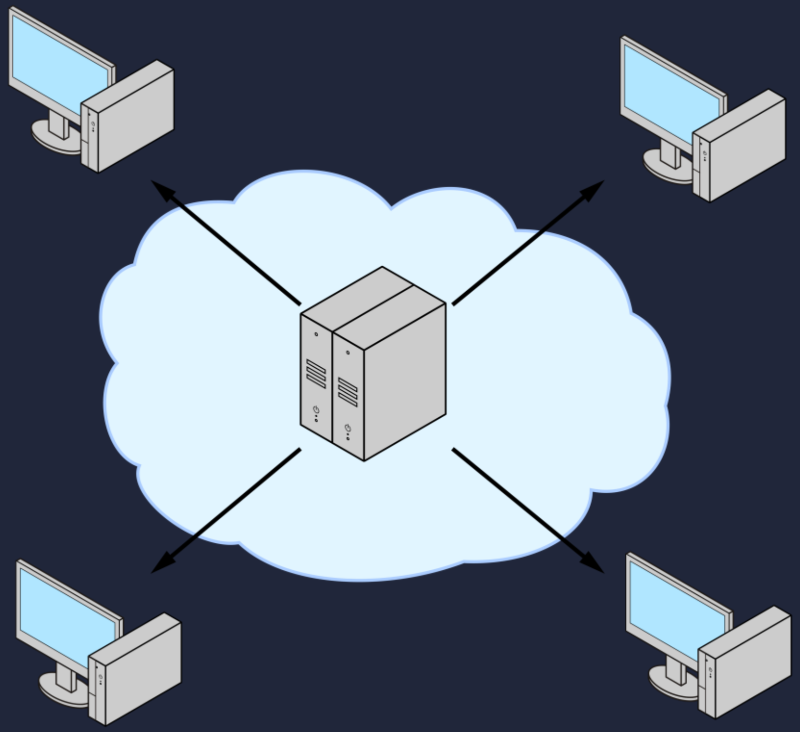
- REST endpoints expose resources via URLs and methods.
- JSON is the standard payload format.
- Status codes are part of the API contract.
- We will implement error responses consistently.



API structure

PHP Execution Model (What Actually Runs?)

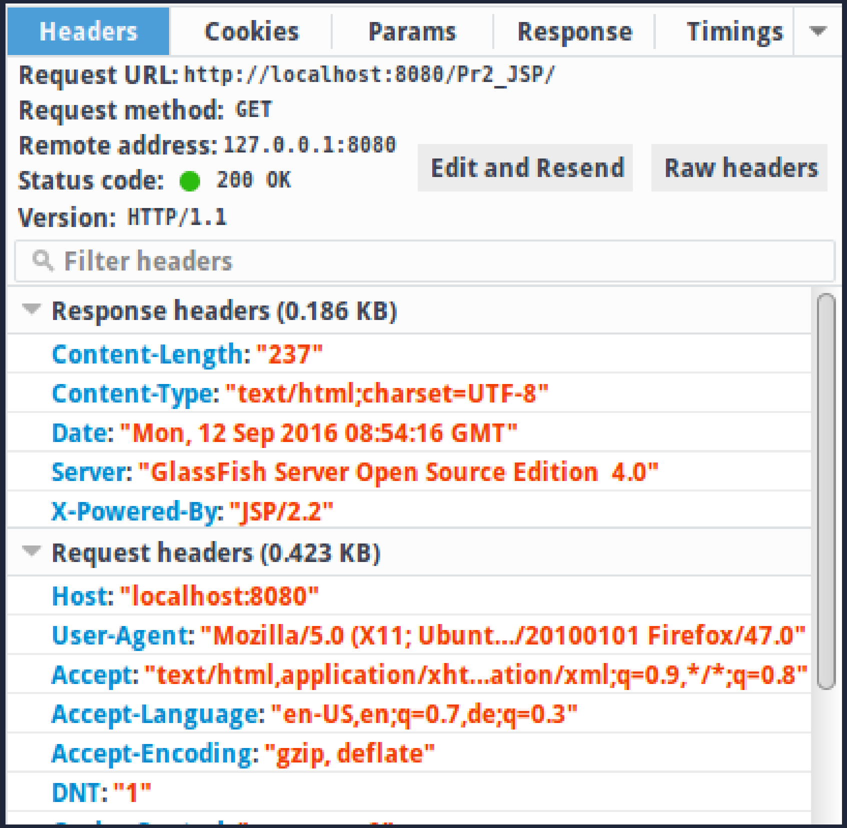
- PHP runs on the server, not in the browser.
- Each HTTP request triggers a fresh PHP execution context.
- You read input from superglobals ($_GET, $_POST, $_FILES, $_COOKIE).
- You write output as HTML or JSON (echo + headers + status codes).



Request-driven execution

Superglobals (Request Input)

- $_GET: query parameters (filters, search).
- $_POST: form body (create/update actions).
- $_FILES: uploaded files (multipart/form-data).
- $_SERVER: request metadata (method, URI, headers).



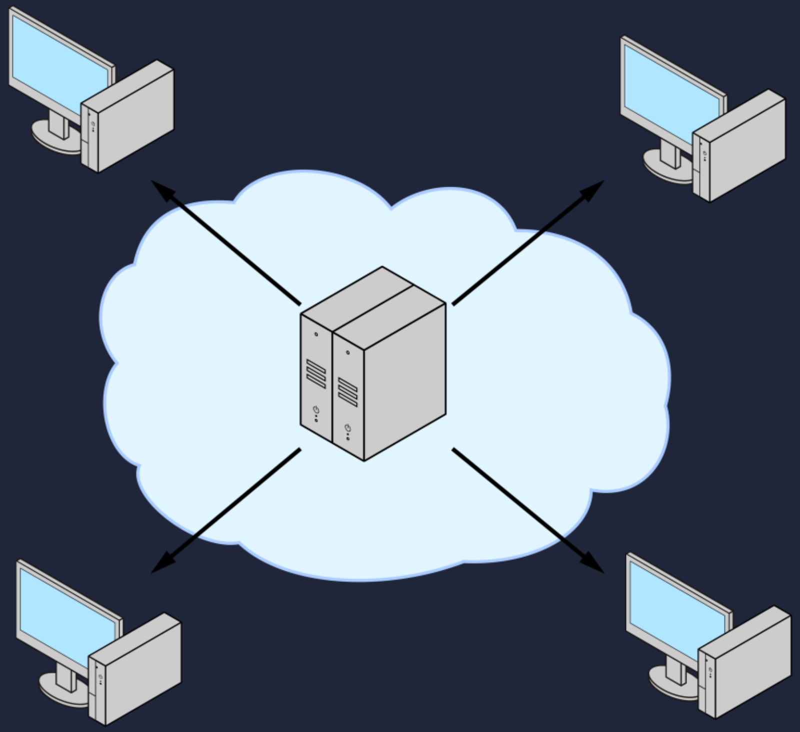
```
Request URL: http://localhost:8080/Pr2_JSP/
Request method: GET
Remote address: 127.0.0.1:8080
Status code: ● 200 OK
Version: HTTP/1.1

Response headers (0.186 KB)
  Content-Length: "237"
  Content-Type: "text/html;charset=UTF-8"
  Date: "Mon, 12 Sep 2016 08:54:16 GMT"
  Server: "GlassFish Server Open Source Edition  4.0"
  X-Powered-By: "JSP/2.2"
Request headers (0.423 KB)
  Host: "localhost:8080"
  User-Agent: "Mozilla/5.0 (X11; Ubunt.../20100101 Firefox/47.0"
  Accept: "text/html,application/xht...ation/xml;q=0.9,*/*;q=0.8"
  Accept-Language: "en-US,en;q=0.7,de;q=0.3"
  Accept-Encoding: "gzip, deflate"
  DNT: "1"
```

Requests are data + metadata

Output Escaping (Prevent XSS)

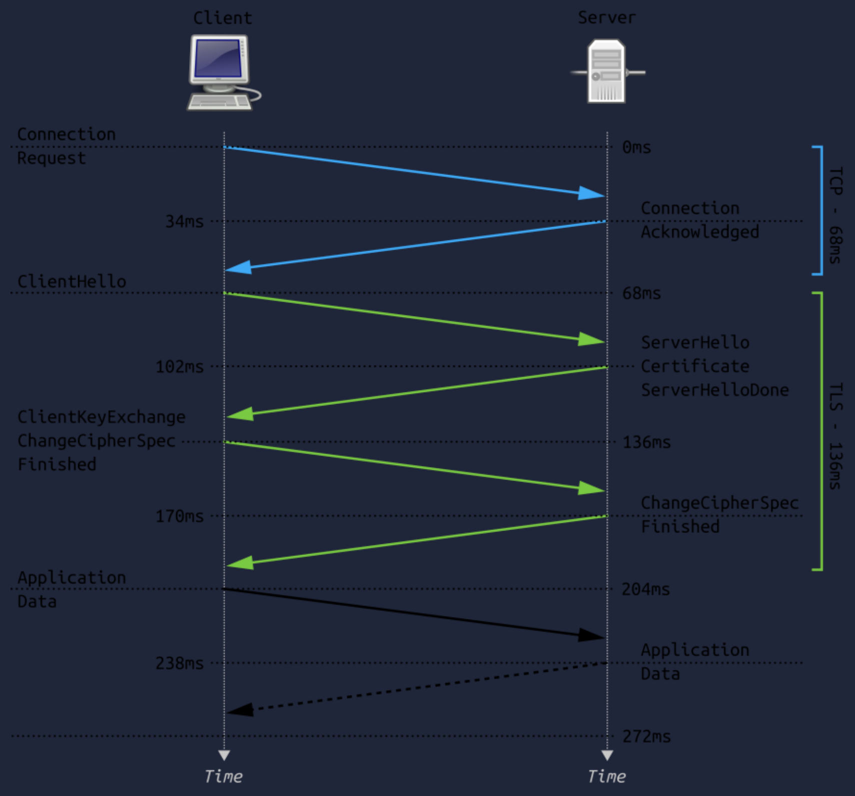
- Escape output when rendering untrusted data into HTML.
- Use htmlspecialchars(...) for HTML contexts.
- Different contexts require different escaping (HTML, JS, URL).
- Prefer templates that default to safe output practices.



Render safely

File Uploads (Preview)

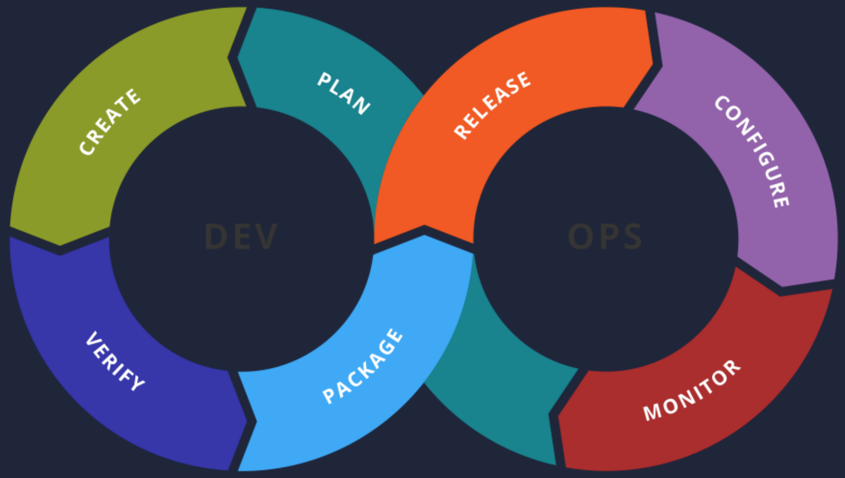
- Uploaded files arrive in $_FILES.
- Always validate file type, size, and filename.
- Store outside the web root or use randomized names.
- Scan/limit to reduce risk in real deployments.



Uploads are a high-risk surface

Error Handling & Logging

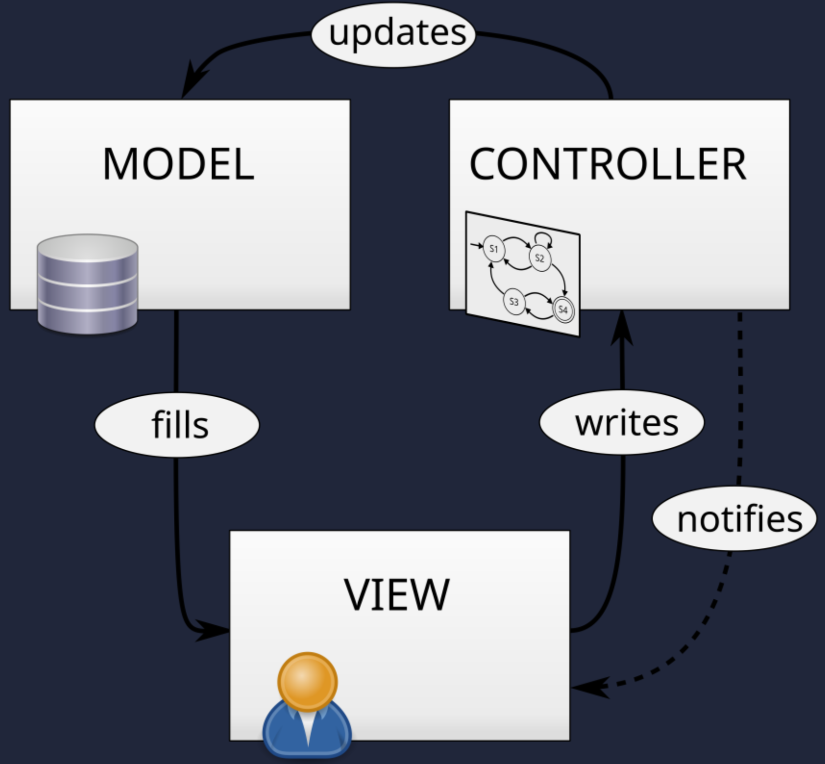
- Use exceptions for failures you cannot recover from locally.
- Log errors server-side; do not leak internals to users.
- Differentiate between 4xx (client) and 5xx (server).
- Logging becomes essential after deployment.



Operate what you build

Object-Oriented PHP (Why It Matters)

- Classes structure code for larger applications (services, repositories).
- Dependency injection reduces coupling and increases testability.
- Autoloading via Composer keeps imports clean.
- We will use OOP to build MVC layers.



Architecture enables growth

PHP Basics I — Syntax and Control Flow

- Variables and types are dynamically managed, but discipline matters.
- Use strict input validation; do not trust user data.
- Control flow: if/else, switch, loops.
- Build small functions rather than long scripts.

```php
<?php

$age = 21;

if ($age >= 18) {
    echo "Allowed";
} else {
    echo "Denied";
}

for ($i = 0; $i < 3; $i++) {
    echo $i;
}
```

PHP Basics II — Arrays and Functions

- Arrays are the workhorse data structure in PHP.
- Functions encapsulate logic and enable reuse.
- Prefer clear naming and predictable return values.
- We will gradually move from scripts to structured code.

```
<?php

$roles = ["user", "admin"];

function isAdmin(array $roles): bool
{
    return in_array("admin", $roles,
true);
}

if (isAdmin($roles)) {
    echo "Admin access";
}
```

Handling Forms — GET vs POST

- GET: parameters in the URL (safe for searches and filters).
- POST: data in the request body (forms, create actions).
- Always validate and sanitize server-side.
- Never trust “required” attributes alone.

```
<?php

$name = trim($_POST['name'] ?? '');

if ($name === '') {
    http_response_code(400);
    echo "Name is required.";
    exit;
}

echo "Hello, " .
htmlspecialchars($name, ENT_QUOTES,
'UTF-8');
```

Sessions in PHP (Authentication Foundation)

- Sessions store user state on the server.
- A cookie stores the session id in the browser.
- Regenerate session ids after login.
- Use secure cookie flags in production.

```php
<?php

session_start();

// after successful login
session_regenerate_id(true);
$_SESSION['user_id'] = $userId;

// logout
// session_destroy();
```

File Upload Handling (Safe Defaults)

- Check upload errors and size limits.
- Whitelist file types (do not trust extensions).
- Generate random file names; never use user-provided names.
- Store outside the web root when possible.

```php
<?php

if (!isset($_FILES['avatar']) ||
$_FILES['avatar']['error'] !==
UPLOAD_ERR_OK) {
    http_response_code(400);
    exit('Upload failed');
}

$tmp = $_FILES['avatar']['tmp_name'];
$size = $_FILES['avatar']['size'];

if ($size > 2 * 1024 * 1024) {
    http_response_code(413);
    exit('File too large');
}

$mime = mime_content_type($tmp);
$allowed = ['image/png', 'image/jpeg'];
if (!in_array($mime, $allowed, true)) {
    http_response_code(415);
    exit('Unsupported type');
}
```

JSON API Response (Consistent Contracts)

- APIs typically respond with JSON.
- Set Content-Type, status code, and a predictable structure.
- Avoid leaking internal errors; return safe messages.
- Standardize success and error shapes.

```
<?php

header('Content-Type:
application/json; charset=utf-8');

try {
    // ... business logic
    echo json_encode(['ok' => true,
'data' => $payload]);
} catch (Throwable $e) {
    http_response_code(500);
    echo json_encode(['ok' => false,
'error' => 'Internal error']);
}
```

Error Boundaries: 4xx vs 5xx

- Use 4xx for validation/input problems.
- Use 5xx for unexpected server failures.
- Return actionable messages for clients, details for logs.
- This separation improves debugging and client UX.

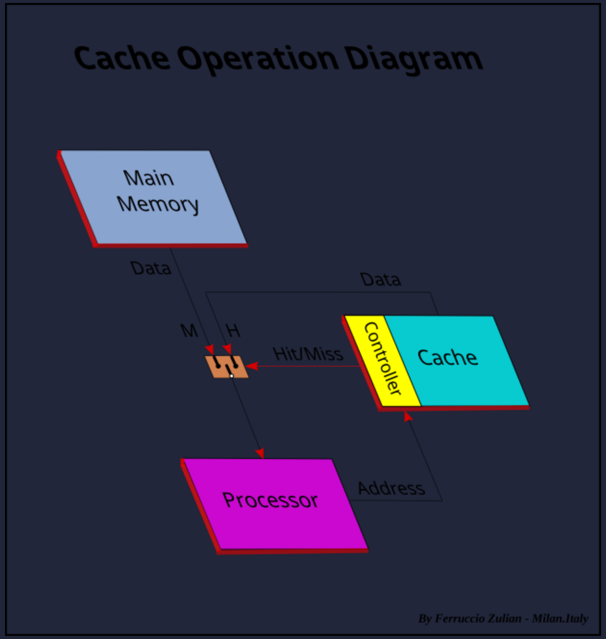
```php
<?php

function badRequest(string $msg):
void {
    http_response_code(400);
    echo $msg;
    exit;
}

try {
    if (trim($_POST['title'] ?? '')
=== '') {
        badRequest('title is
required');
    }
    // ... normal flow
} catch (Throwable $e) {
    http_response_code(500);
    echo 'Internal server error';
}
```

Data Layer (MySQL)

Database design, SQL, joins, indexes, transactions



Why a Relational Database?

- Relational databases store structured data reliably.
- Constraints maintain data integrity (keys, relations).
- SQL enables powerful queries and reporting.
- MySQL is widely used in production web systems.

Tables, Rows, Columns

- A table represents an entity (users, products, orders).
- Rows are records; columns are attributes.
- Types and constraints matter for correctness.
- Design choices impact performance and maintainability.

SQL Basics — SELECT

- SELECT retrieves rows from one or more tables.
- WHERE filters rows; ORDER BY sorts results.
- LIMIT supports pagination.
- We will read and write SQL daily in MySQL weeks.

```
SELECT id, name, price
FROM products
WHERE price >= 100
ORDER BY price DESC
LIMIT 20 OFFSET 0;
```

SQL Basics — INSERT / UPDATE / DELETE

- INSERT creates rows; UPDATE modifies; DELETE removes.
- Use transactions for multi-step operations.
- Use constraints to enforce correctness.
- We will build CRUD services around these operations.

```
INSERT INTO users (email,
password_hash)
VALUES ('a@b.com', '...');

UPDATE users
SET email = 'new@b.com'
WHERE id = 42;

DELETE FROM users
WHERE id = 42;
```

JOIN Example — Orders With Users

- Join related tables using foreign keys.
- Select only the columns you need.
- Indexes on join keys improve performance.
- This pattern appears throughout the capstone.

```
SELECT o.id, o.total, u.email
FROM orders AS o
JOIN users  AS u ON u.id = o.user_id
WHERE o.created_at >= '2026-01-01'
ORDER BY o.created_at DESC
LIMIT 50;
```

Aggregations — GROUP BY and COUNT

- Aggregations support reporting and dashboards.
- GROUP BY groups rows by a key.
- COUNT/SUM/AVG are common aggregate functions.
- Always consider indexes for GROUP BY keys.

```
SELECT u.id, u.email, COUNT(o.id) AS
order_count
FROM users AS u
LEFT JOIN orders AS o ON o.user_id =
u.id
GROUP BY u.id, u.email
ORDER BY order_count DESC
LIMIT 20;
```

PHP + MySQL — PDO Connection

- Use PDO for database access.
- Always use prepared statements to avoid SQL injection.
- Centralize connection configuration.
- Handle errors via exceptions.

```
<?php

$pdo = new PDO(

'mysql:host=localhost;dbname=app;cha
rset=utf8mb4',
  'user',
  'pass',
  [PDO::ATTR_ERRMODE =>
PDO::ERRMODE_EXCEPTION]
);
```

Prepared Statements (Security Baseline)

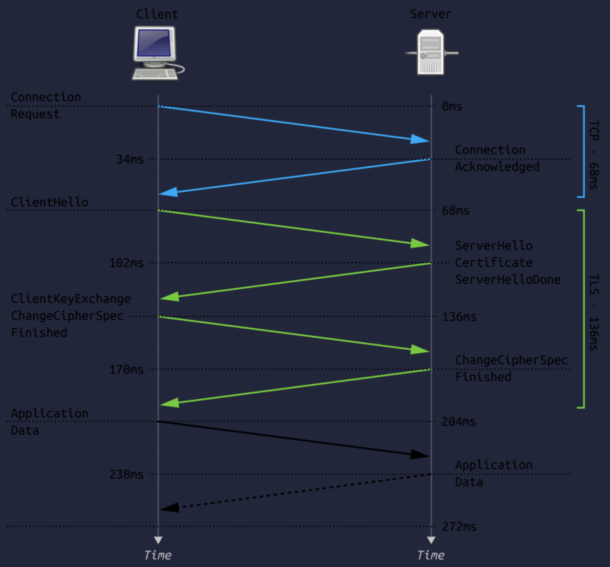
- Prepared statements separate SQL from user data.
- This prevents many injection attacks.
- Bind parameters with explicit types when possible.
- Never concatenate raw user input into SQL.

```
<?php

$stmt = $pdo->prepare('SELECT * FROM
users WHERE email = ?');
$stmt->execute([$email]);
$user = $stmt-
>fetch(PDO::FETCH_ASSOC);
```

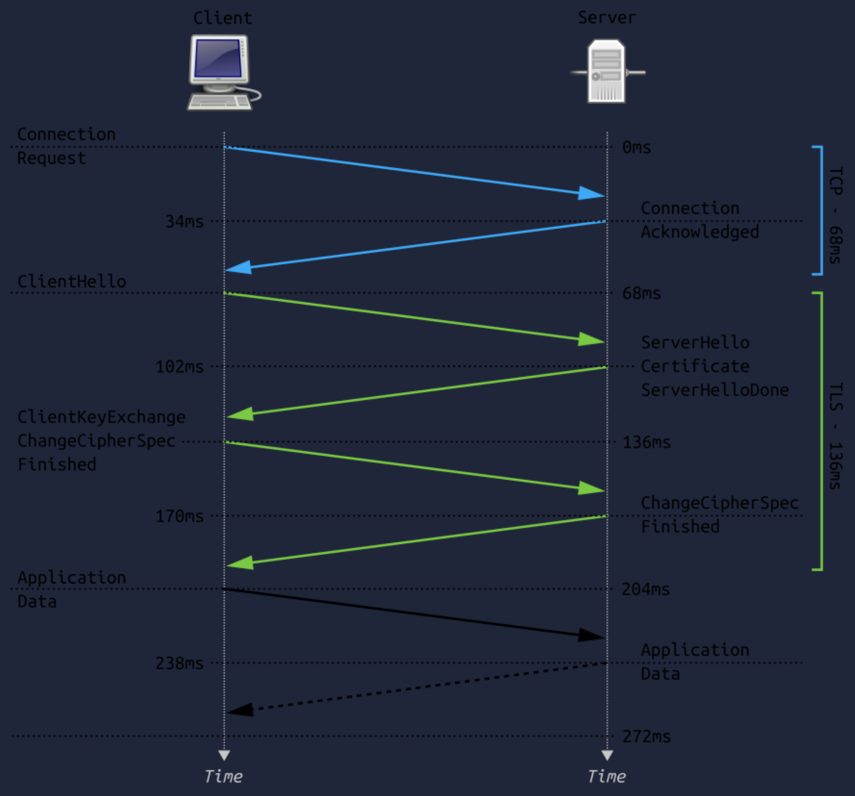
Security

Threats to expect
and how we mitigate them



OWASP Mindset

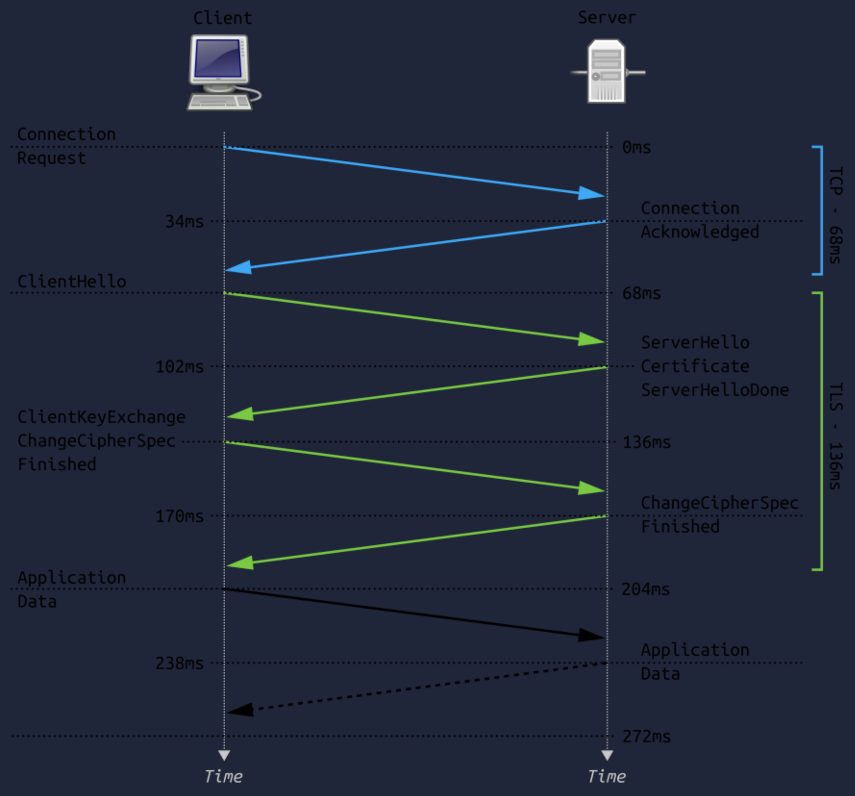
- Security is part of engineering, not an afterthought.
- We will discuss common web vulnerabilities (OWASP Top 10).
- Core practices: input validation, output escaping, least privilege.
- Secure defaults matter (cookies, headers, password storage).



OWASP tooling & culture

SQL Injection

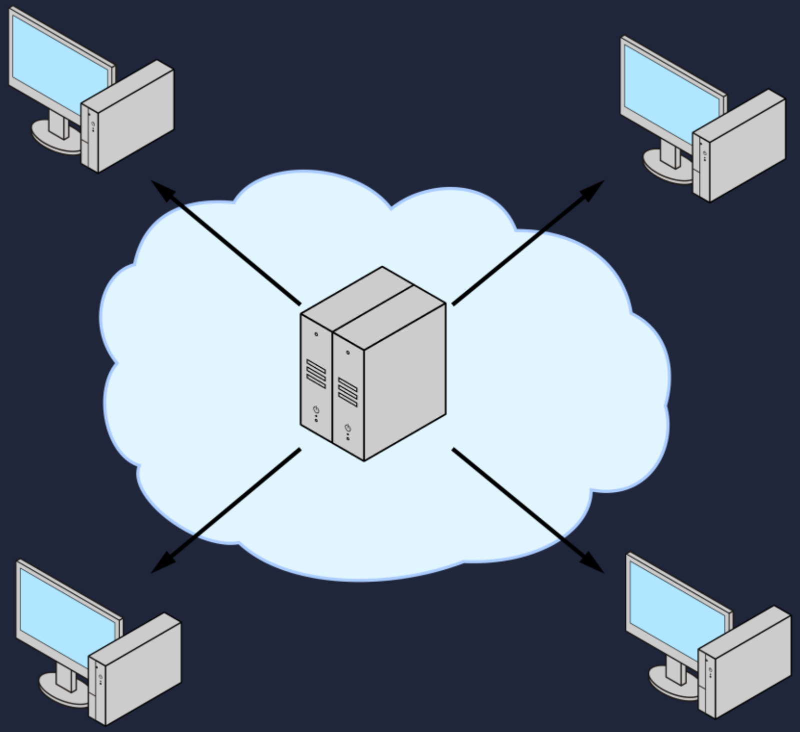
- Occurs when untrusted input changes SQL semantics.
- Prevent with prepared statements and strict validation.
- Use least-privilege database users.
- Log and monitor suspicious patterns.



Prepared statements mitigate SQLi

Cross-Site Scripting (XSS)

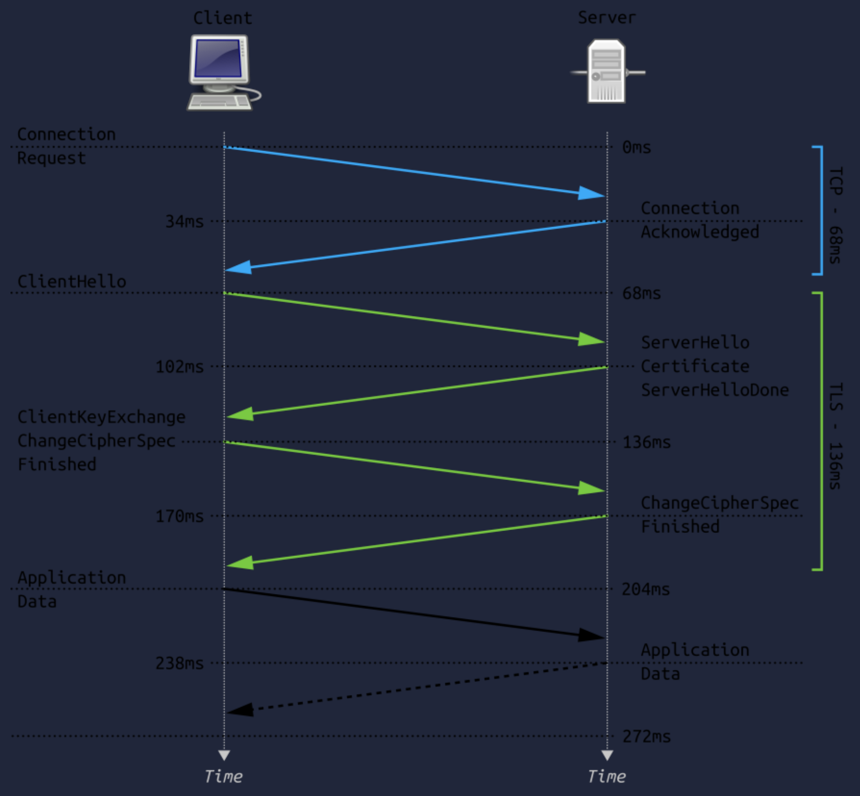
- Occurs when untrusted data is rendered as executable HTML/JS.
- Prevent with output escaping (e.g., htmlspecialchars in PHP).
- Use Content Security Policy where appropriate.
- Validate and sanitize inputs, but escape outputs.



Output encoding is essential

Cross-Site Request Forgery (CSRF)

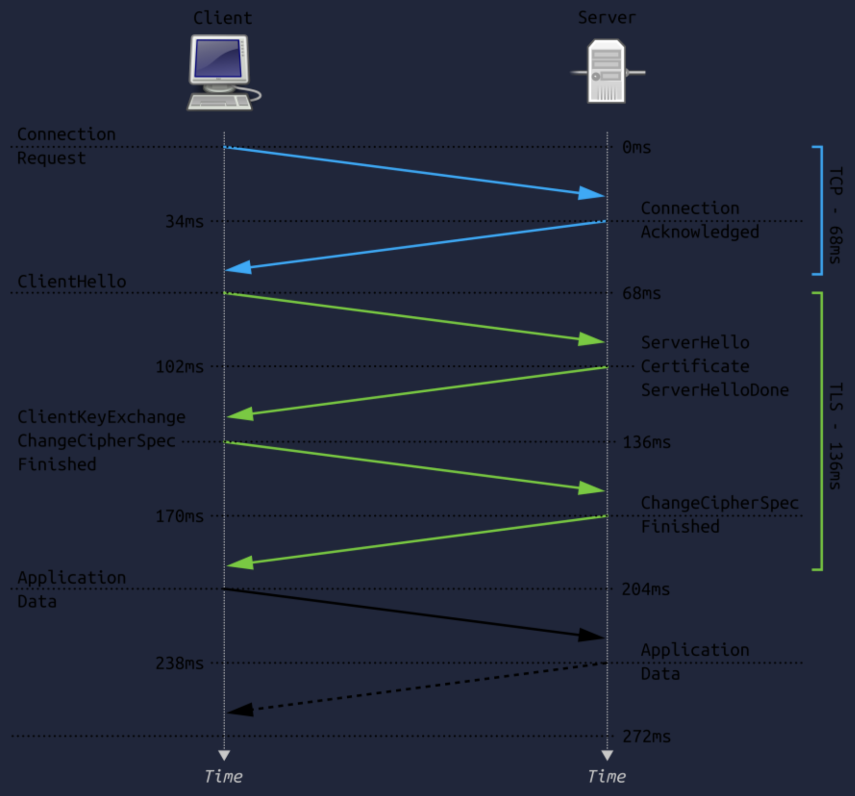
- Forces a logged-in browser to send unwanted requests.
- Prevent with CSRF tokens and SameSite cookies.
- Use proper method usage (avoid state changes in GET).
- Validate origin/referer where appropriate.



CSRF tokens protect state-changing actions

Password Storage

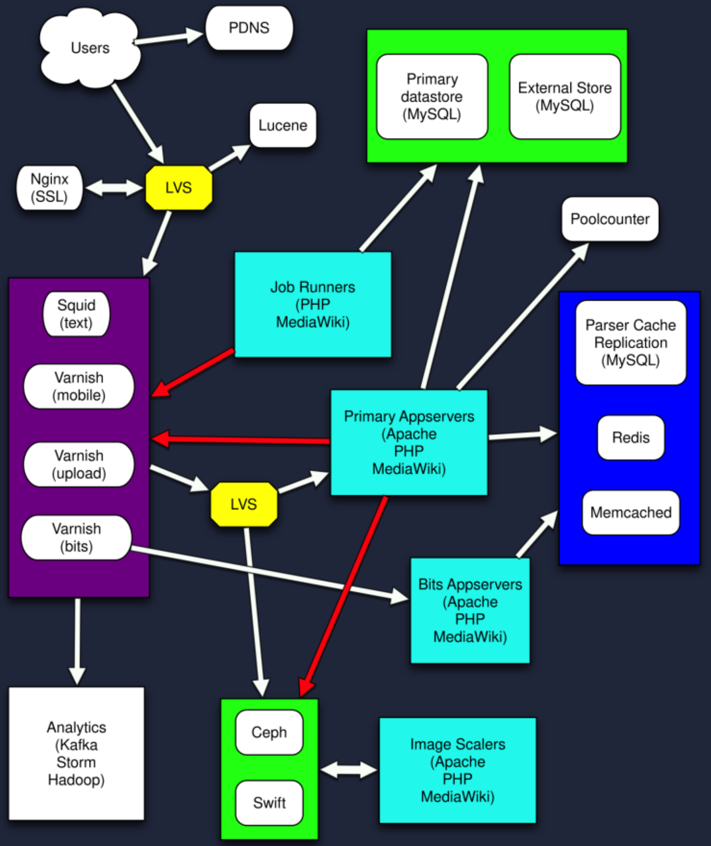
- Never store plain-text passwords.
- Use password_hash() and password_verify() in PHP.
- Use rate limiting and lockouts for brute-force mitigation.
- Prefer multi-factor authentication in real systems.



Hashing is mandatory

Authorization (Roles & Permissions)

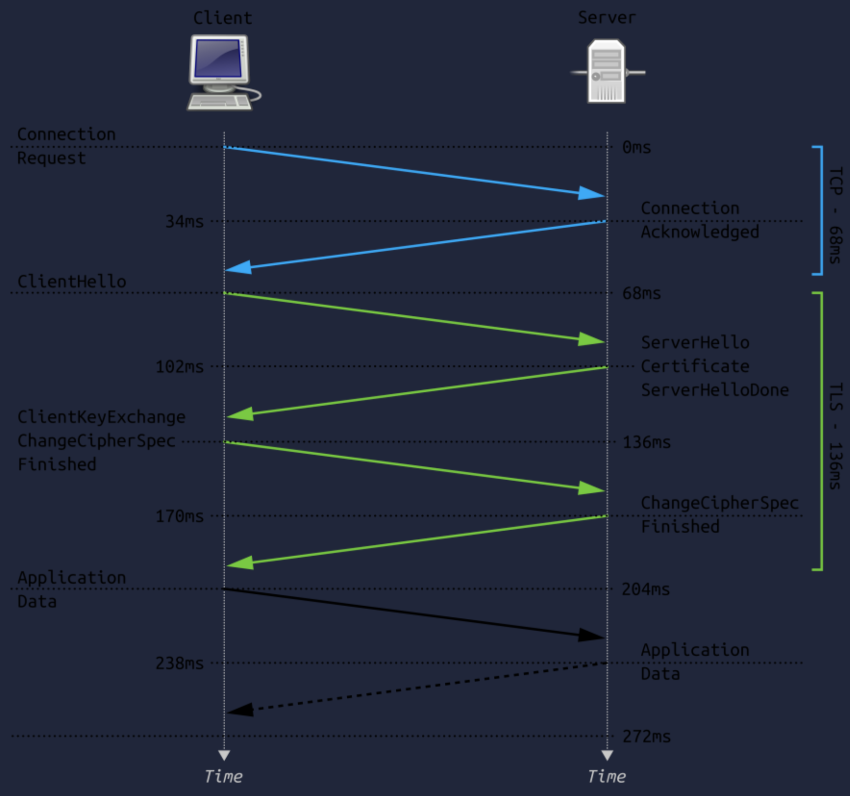
- Authentication = who you are; authorization = what you can do.
- Design roles (user/admin) and enforce checks server-side.
- Never rely on “hidden buttons” for security.
- Audit and log sensitive actions.


Users
PDNS
Primary datastore (MySQL)
External Store (MySQL)
Lucene
Nginx (SSL)
LVS
Poolcounter
Job Runners (PHP MediaWiki)
Squid (text)
Varnish (mobile)
Varnish (upload)
Varnish (bits)
Parser Cache Replication (MySQL)
Redis
Memcached
Primary Appservers (Apache PHP MediaWiki)
LVS
Bits Appservers (Apache PHP MediaWiki)
Analytics (Kafka Storm Hadoop)
Ceph
Swift
Image Scalers (Apache PHP MediaWiki)

Enforce permissions on the server

File Upload Security (Preview)

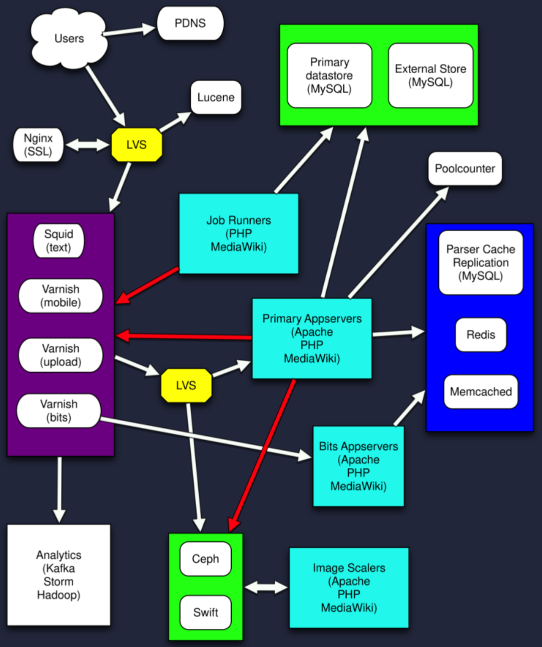
- Validate file type and size; never trust extensions.
- Store uploads outside the web root when possible.
- Rename files to avoid path traversal and collisions.
- Scan uploads when required in real deployments.



Uploads must be treated as untrusted input

Server & Deployment

Linux server setup
(Apache + PHP+ MySQL)



Users
PDNS
Lucene
Primary datastore (MySQL)
External Store (MySQL)
Nginx (SSL)
LVS
Poolcounter
Squid (text)
Varnish (mobile)
Varnish (upload)
Varnish (bits)
Job Runners (PHP MediaWiki)
Parser Cache Replication (MySQL)
Primary Appservers (Apache PHP MediaWiki)
Redis
LVS
Memcached
Bits Appservers (Apache PHP MediaWiki)
Analytics (Kafka Storm Hadoop)
Ceph
Swift
Image Scalers (Apache PHP MediaWiki)

Linux + SSH Basics

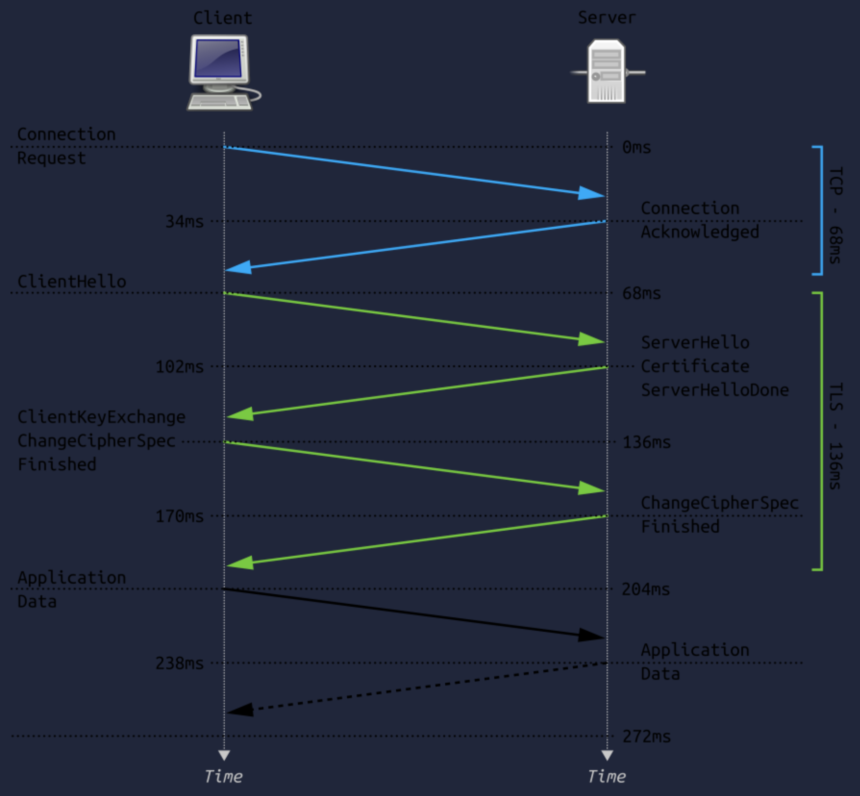
- You will connect to servers using SSH.
- Understand directories, permissions, and service management.
- Use logs to diagnose issues in production environments.
- Aim for repeatable setup steps (documentation matters).

Web Server: Apache

- Apache handles incoming HTTP requests and serves static files.

HTTPS Certificates (Let’s Encrypt Concept)

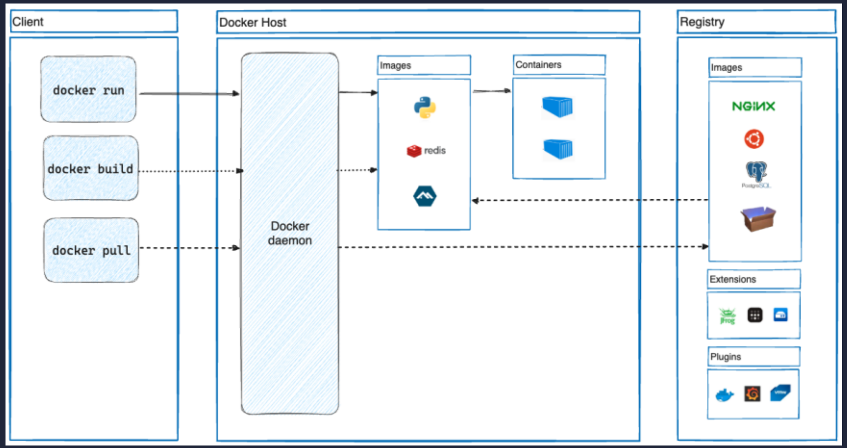
- TLS certificates enable HTTPS and prevent interception.
- Automated renewal prevents outages.
- Correct Nginx configuration is required.
- We will cover a pragmatic deployment checklist.



HTTPS is the default

Logging & Monitoring (Essentials)

- Logs are your primary debugging tool in production.
- Track errors, authentication events, and slow queries.
- A simple health endpoint supports monitoring.
- Measure before optimizing performance.



Operational visibility