Client–Server Request Lifecycle (End-to-End Breakdown)

Think of this as the full journey from the moment a user clicks a link to the moment data appears on screen. We'll dissect it step-by-step and call out hidden bottlenecks where systems usually fail.

1. User Triggers a Request

Event: User types a URL or interacts with the UI (button click, form submit, API call).

Output: Browser prepares an HTTP/HTTPS request.

Key components the browser sets:

- Method: GET, POST, PUT, DELETE
- Headers: User-Agent, Accept, Cookies, Authorization
- Body (only for POST/PUT): JSON, form data, etc.

2. DNS Resolution

The browser needs the server's IP address.

Steps:

1. Check browser cache
2. Check OS cache
3. Check router cache
4. Ask ISP DNS
5. If needed → TLD → authoritative DNS

Output: Browser gets an IP (e.g., 142.250.184.78).

3. TCP Handshake (If HTTPS → TLS Handshake Too)

To communicate reliably, the client must establish a TCP connection.

TCP Three-Way Handshake:

1. SYN
2. SYN-ACK
3. ACK

If HTTPS:

- TLS handshake
- Certificate verification
- Session keys created

Blind spot: Slow handshake = slow site. Optimized servers use HTTP/2 or HTTP/3 to reduce latency.

4. Request is Sent to the Server

Browser sends the full HTTP request:

GET /products HTTP/1.1
Host: example.com
Accept: application/json
Cookie: sessionId=12345

5. Load Balancer Intercepts (Modern Architecture)

Most real systems don't let traffic hit the origin directly.

A Load Balancer:

- Distributes requests across servers
- Offloads SSL/TLS
- Applies rate limiting

- Sometimes performs routing logic (A/B tests, regional routing, microservice mapping)
- If traffic spikes, this is your first line of defense.

6. Application Server Processes the Request

This is where backend logic runs.

Examples:

- Node.js server
- Laravel / Django app
- Spring Boot service

Server responsibilities:

- Read request headers/body
- Authenticate (tokens, cookies, API keys)
- Validate user input
- Check permissions
- Execute business logic

Scenario: If the endpoint needs data, the server calls the database or another microservice.

7. Database Query or Internal Service Calls

Depending on the system design, the backend might:

- Query a SQL database
- Query NoSQL (Mongo, Redis)
- Call internal APIs (microservices)
- Fetch from a cache (Redis, Memcached)
- Use message queues (Kafka, RabbitMQ)

Performance insight:

90% of request latency often comes from database or microservice calls. This is where apps usually break under load.

8. Server Constructs a Response

The server builds an HTTP response:

HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: max-age=3600

{ "products": [...] }

Options:

- JSON (API)
- HTML (SSR)
- XML
- Files (images, PDFs)

9. Response Travels Back Through Load Balancer

Before reaching the client:

- Compression (gzip, brotli)
- Caching rules applied
- Security headers injected
- Logging/monitoring captured

10. Browser Receives the Response

Browser checks:

- Status code
- Content-Type
- Cookies
- Redirect instructions

If HTML:

- Browser begins rendering
- Additional files requested (CSS, JS, images)

If JSON:

- JS code (frontend) updates the UI.

11. Rendering Pipeline (Frontend Final Stage)

Browser:

1. Builds DOM
2. Builds CSSOM
3. Executes JavaScript
4. Applies layout
5. Paints pixels

Any heavy JS can block rendering, causing slow UI.

Final Summary (Compressed Version)

Full Lifecycle in one line:

DNS → TCP/TLS → Request → Load balancer → Backend → DB/cache → Backend → Load balancer → Response → Browser render

This is the canonical flow used by FAANG-scale systems.

Real-World Example: Logging Into Facebook

Let's dissect what actually happens behind the curtain when you log in to Facebook.

Step-by-step Breakdown

1. **User Action**

You open facebook.com and enter:

- email/ phone
- Password

Then click *Log in.*

2. **Browser Prepares the Request**

It builds a POST request:

POST https://www.facebook.com/login
Content-Type: application/x-www-formurlencoded

Paload contains:

- User email
- Password
- Device metadata
- Tracking cookies

3. **DNS Resolution**

Your browser resolves:

www.facebook.com -> 157.240.x.x

This can come from:

- Cache
- Router
- ISP
- Facebook's authoritative DNS

Facebook uses **global anycast DNS** so you hit the closed data center.

4. **TCP + TLS Handshake**

You create a secure session with Facebook

Browser verifies the SSL certificate

- Issuer: DigiCert
- Validity
- Domain match

Once validated, HTTPS encryption starts.

5. **Request Hits Facebooks's Edge Infrastruture**

The request is routed into Facebook's front-end network:

- Global load balancers
- Front-end proxies (revers proxies)
- Rate limiters
- Security filters (bot detection, login fraud analysis)

Facebook immediately runs risk checks:

- Device fingerprint
- IP reputation
- Login attempt history
- Behavior pattern

6. **Application Server (Auth Service) Process the Login**

The auth service performs:

a) **Input validation**

Email format, password string integrity, etc.

b) **Account lookup**

Queries a distributed database:

*SELECT * FROM users WHERE email = ?*

c) Password verification

Facebook stored haased passwords

bcrypt(password_input) == stored_hash

d) Security checks

- Two-factor authentication?
- Suspicious login detection?
- Location mismatch?
- Device history analysis?

7. Session Token Creation

If everything passes, Facebook creates:

- Session ID
- Access token
- Refresh token

These are stored in:

- Redis
- Memcache
- Facebooks' distributed TAO system

Browser receives token via **HTTP Set-Cookie**

8. User Profile Data Fetched

Auth service fetches or instructs services to fetch:

- Feed ranking
- Friends list
- Notifications
- Messages count

All using microservices

9. Response Sent Back
HTTP/1.1 302 Found
Location: https://www.facebook.com/home
Set-Cookie: c_user=123456789; Secure; HttpOnly

Browser follows the redirect

10. Browser Renders the Feed

UI pipeline:

- DOM construction
- CSSOM
- JS execution
- React/ GraphQL hydration
- Feed appears

Final Result:
You see your Facebook homepage with personalized content.