10.2 HTTP, HTTPS und Webkommunikation

HTTP und HTTPS gehören zu den wichtigsten Protokollen der Anwendungsschicht.

Sie werden vor allem für Webseiten, Webanwendungen und APIs verwendet.

HTTP steht für:

```
Hypertext Transfer Protocol
```

HTTPS steht für:

```
Hypertext Transfer Protocol Secure
```

Der wichtigste Unterschied:

```
HTTP ist unverschlüsselt.
HTTPS nutzt TLS-Verschlüsselung.
```

Merksatz:

```
HTTP = Webkommunikation ohne TLS.
HTTPS = Webkommunikation mit TLS.
```

Grundidee von HTTP

HTTP ist ein Protokoll zur Übertragung von Webinhalten.

Ein Client stellt eine Anfrage.

Ein Server liefert eine Antwort.

Beispiel:

```
Browser fordert Webseite an.
Webserver liefert HTML, CSS, JavaScript, Bilder oder Daten zurück.
```

Typische Inhalte:

```
- Webseiten
- Bilder
- CSS-Dateien
- JavaScript-Dateien
- JSON-Daten
- API-Antworten
- Downloads
```

Merksatz:

```
HTTP arbeitet mit Anfrage und Antwort.
```

Client und Server bei HTTP

Bei HTTP gibt es meistens:

```
Client
Server
```

Client:

```
stellt Anfrage
```

Server:

```
verarbeitet Anfrage
und sendet Antwort
```

Beispiel:

```
Browser = Client
Webserver = Server
```

Merksatz:

```
Client fragt an,
Server antwortet.
```

Request und Response

Eine HTTP-Kommunikation besteht aus:

```
Request
Response
```

Request bedeutet:

```
Anfrage des Clients
```

Response bedeutet:

```
Antwort des Servers
```

Beispiel:

```
Request:
Browser fordert /index.html an.

Response:
Server liefert die Datei index.html zurück.
```

Merksatz:

```
Request = Anfrage.
Response = Antwort.
```

HTTP-Request

Ein HTTP-Request enthält unter anderem:

```
- Methode
- Pfad oder URL
- Header
- optionalen Body
```

Beispiel vereinfacht:

```
GET /index.html HTTP/1.1
Host: example.com
```

Bedeutung:

```
Client möchte die Datei /index.html vom Host example.com abrufen.
```

Merksatz:

```
HTTP-Request beschreibt,
was der Client vom Server möchte.
```

HTTP-Response

Eine HTTP-Response enthält unter anderem:

```
- Statuscode
- Header
- optionalen Body
```

Beispiel vereinfacht:

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>...</html>
```

Bedeutung:

Server antwortet erfolgreich mit HTML-Inhalt.

Merksatz:

HTTP-Response enthält Ergebnis und Inhalt.

URL

URL steht für:

Uniform Resource Locator

Eine URL beschreibt, wo eine Ressource erreichbar ist.

Beispiel:

https://www.example.com:443/docs/index.html?seite=1

Bestandteile:

Bestandteil	Beispiel	Bedeutung
Schema	https	Protokoll
Hostname	www.example.com	Servername
Port	443	Dienst-Port
Pfad	/docs/index.html	Ressource
Query	?seite=1	zusätzliche Parameter

Merksatz:

URL = Adresse einer Ressource.

Schema

Das Schema steht am Anfang einer URL.

Beispiele:

```
http://
https://
```

Das Schema sagt dem Client, welches Protokoll verwendet werden soll.

Beispiel:

```
http://example.com
```

nutzt HTTP.

```
https://example.com
```

nutzt HTTPS.

Merksatz:

```
Schema zeigt das verwendete Protokoll.
```

Hostname

Der Hostname ist der Name des Servers.

Beispiel:

```
www.example.com
```

Der Hostname muss meist per DNS in eine IP-Adresse aufgelöst werden.

Ablauf vereinfacht:

```
Browser kennt Namen.
DNS liefert IP-Adresse.
Browser verbindet sich zur IP-Adresse.
```

Merksatz:

```
Hostname wird über DNS zur IP-Adresse.
```

Port bei HTTP und HTTPS

HTTP nutzt typischerweise:

```
TCP 80
```

HTTPS nutzt typischerweise:

```
TCP 443
```

Wenn kein Port angegeben wird, nutzt der Client den Standardport des Schemas.

Beispiele:

```
http://example.com
→ TCP 80

https://example.com
→ TCP 443
```

Merksatz:

```
HTTP = TCP 80.
HTTPS = TCP 443.
```

Pfad

Der Pfad zeigt, welche Ressource auf dem Server angefordert wird.

Beispiele:

```
/
/index.html
/produkte
/api/users
/bilder/logo.png
```

Der Server entscheidet, welcher Inhalt zu diesem Pfad geliefert wird.

Merksatz:

```
Pfad zeigt die gewünschte Ressource auf dem Server.
```

Query-Parameter

Query-Parameter stehen nach einem Fragezeichen.

Beispiel:

```
/suche?q=netzwerk&page=2
```

Bedeutung:

```
q=netzwerk
page=2
```

Query-Parameter übergeben zusätzliche Informationen an den Server.

Typisch bei:

```
- Suchfunktionen
- Filtern
- Seitennummern
- API-Abfragen
```

Merksatz:

```
Query-Parameter geben Zusatzinformationen mit.
```

HTTP-Methoden

HTTP-Methoden beschreiben, was der Client tun möchte.

Wichtige Methoden:

Methode	Bedeutung
GET	Daten abrufen

Methode	Bedeutung
POST	Daten senden oder erstellen
PUT	Ressource vollständig ersetzen
PATCH	Ressource teilweise ändern
DELETE	Ressource löschen
HEAD	nur Kopfzeilen abrufen
OPTIONS	unterstützte Möglichkeiten abfragen

Merksatz:

```
GET liest.
POST sendet.
PUT ersetzt.
PATCH ändert teilweise.
DELETE löscht.
```

GET

GET wird verwendet, um Daten abzurufen.

Beispiele:

```
Webseite laden
Bild abrufen
API-Daten lesen
```

GET sollte normalerweise keine Daten verändern.

Beispiel:

```
GET /artikel/10
```

Bedeutung:

```
Artikel 10 abrufen.
```

Merksatz:

```
GET = Daten abrufen.
```

POST

POST wird verwendet, um Daten an den Server zu senden.

Beispiele:

```
Formular absenden
Benutzer erstellen
Datei hochladen
API-Daten senden
```

Beispiel:

```
POST /login
```

Bedeutung:

```
Login-Daten werden an den Server gesendet.
```

Merksatz:

```
POST = Daten an Server senden.
```

PUT

PUT ersetzt eine Ressource vollständig.

Beispiel:

```
PUT /users/5
```

Bedeutung:

```
Benutzer 5 wird vollständig mit den gesendeten Daten ersetzt.
```

Merksatz:

```
PUT = vollständig ersetzen.
```

PATCH

PATCH ändert eine Ressource teilweise.

Beispiel:

```
PATCH /users/5
```

Bedeutung:

```
Bei Benutzer 5 wird nur ein Teil geändert,
zum Beispiel die E-Mail-Adresse.
```

Merksatz:

```
PATCH = teilweise ändern.
```

DELETE

DELETE löscht eine Ressource.

Beispiel:

```
DELETE /users/5
```

Bedeutung:

```
Benutzer 5 löschen.
```

Merksatz:

```
DELETE = Ressource löschen.
```

HTTP-Header

HTTP-Header enthalten Zusatzinformationen zur Anfrage oder Antwort.

Beispiele:

Header	Bedeutung
Host	angefragter Hostname
User-Agent	Client-Information
Accept	gewünschte Antwortformate
Content-Type	Format des gesendeten Inhalts
Authorization	Authentifizierungsdaten
Cookie	Cookie-Daten
Set-Cookie	Server setzt Cookie
Cache-Control	Cache-Verhalten
Location	Weiterleitungsziel

Merksatz:

> Header enthalten Zusatzinformationen.

Content-Type

Content-Type beschreibt, welches Datenformat im Body enthalten ist.

Beispiele:

Content-Type	Bedeutung
text/html	HTML-Webseite
application/json	JSON-Daten
application/xml	XML-Daten
text/plain	einfacher Text
image/png	PNG-Bild
application/pdf	PDF-Dokument

Merksatz:

```
Content-Type sagt,
welches Format der Inhalt hat.
```

Accept-Header

Der Accept-Header sagt dem Server, welche Antwortformate der Client akzeptiert.

Beispiel:

```
Accept: application/json
```

Bedeutung:

```
Client möchte JSON als Antwort erhalten.
```

Merksatz:

```
Accept sagt,
was der Client empfangen möchte.
```

Authorization-Header

Der Authorization-Header wird für Authentifizierung verwendet.

Beispiele:

```
Bearer Token
Basic Authentication
API-Key-Verfahren
```

Beispiel vereinfacht:

```
Authorization: Bearer eyJ...
```

Wichtig:

```
Solche Daten müssen geschützt übertragen werden.
Deshalb sollte dafür HTTPS genutzt werden.
```

Merksatz:

```
Authorization-Header enthält Zugriffsinformationen.
```

Cookies

Cookies sind kleine Daten, die der Server im Browser speichern lassen kann.

Der Server sendet:

```
Set-Cookie
```

Der Browser sendet später:

```
Cookie
```

Typische Nutzung:

```
- Sitzung wiedererkennen
- Login-Status
- Spracheinstellung
- Warenkorb
- Tracking
```

Merksatz:

```
Cookies helfen,
Zustand über mehrere HTTP-Anfragen zu merken.
```

HTTP ist zustandslos

HTTP ist grundsätzlich zustandslos.

Das bedeutet:

```
Jede Anfrage ist zunächst unabhängig.
```

Ohne Cookies, Tokens oder andere Mechanismen wüsste der Server nicht automatisch, dass mehrere Anfragen zum gleichen Benutzer gehören.

Beispiel:

```
Anfrage 1:
Login

Anfrage 2:
Dashboard öffnen
```

Der Server braucht einen Mechanismus, um die Sitzung wiederzuerkennen.

Merksatz:

```
HTTP ist zustandslos,
Sitzungen schaffen Zusammenhang.
```

Session-Cookie

Ein Session-Cookie kann eine Session-ID enthalten.

Damit erkennt der Server:

```
Diese Anfrage gehört zu einer bestimmten Sitzung.
```

Beispiel:

```
Benutzer meldet sich an.
Server setzt Session-Cookie.
Browser sendet Cookie bei weiteren Anfragen mit.
Server erkennt Benutzer wieder.
```

Merksatz:

```
Session-Cookie verbindet mehrere HTTP-Anfragen zu einer Sitzung.
```

HTTPS

HTTPS ist HTTP über TLS.

Dabei werden HTTP-Daten verschlüsselt übertragen.

Schutz durch TLS:

- Schutz vor Mitlesen
- Schutz vor unbemerkter Veränderung
- Prüfung der Serveridentität über Zertifikat

Merksatz:

HTTPS schützt HTTP-Kommunikation mit TLS.

Ablauf bei HTTPS vereinfacht

Bei HTTPS passiert vereinfacht:

1. DNS löst den Namen auf.
2. TCP-Verbindung zu Port 443 wird aufgebaut.
3. TLS-Handshake findet statt.
4. Zertifikat wird geprüft.
5. HTTP-Anfrage wird verschlüsselt gesendet.
6. HTTP-Antwort wird verschlüsselt empfangen.

Merksatz:

Bei HTTPS kommt vor HTTP zuerst TCP und TLS.

HTTP und TLS unterscheiden

Thema	Einordnung
HTTP	Anwendungsschicht-Protokoll
HTTPS	HTTP über TLS
TLS	Verschlüsselungs- und Schutzschicht

Thema	Einordnung
TCP 443	Transportverbindung
Zertifikat	Vertrauensnachweis bei TLS

Merksatz:

```
HTTP spricht die Anwendung.
TLS schützt die Verbindung.
TCP transportiert.
```

HTTP-Versionen

Es gibt verschiedene HTTP-Versionen.

Wichtige Versionen:

Version	Kurzidee
HTTP/1.1	klassisch, weit verbreitet
HTTP/2	effizientere Übertragung über eine Verbindung
HTTP/3	nutzt QUIC über UDP

Wichtig für die Prüfung:

```
HTTP/3 nutzt QUIC über UDP.
Klassisches HTTPS nutzt häufig TCP 443.
```

Merksatz:

```
HTTP/3 nutzt QUIC über UDP.
```

HTTP/1.1

HTTP/1.1 ist eine klassische und weit verbreitete Version.

Typische Merkmale:

```
- Request/Response
- Header
```

- Host-Header wichtig für virtuelle Hosts
- mehrere Anfragen möglich
- sehr verbreitet

Merksatz:

HTTP/1.1 ist die klassische Webkommunikation.

HTTP/2

HTTP/2 verbessert die Effizienz der Webkommunikation.

Vorteile:

- mehrere Streams über eine Verbindung
- bessere Nutzung einer Verbindung
- Header-Kompression
- geringere Ladezeiten möglich

Für die Prüfung reicht meist:

HTTP/2 ist moderner und effizienter als HTTP/1.1.

Merksatz:

HTTP/2 kann mehrere Übertragungen effizienter bündeln.

HTTP/3

HTTP/3 basiert auf QUIC.

QUIC nutzt UDP.

Typisch:

UDP 443

Vorteile:

```
- schnellerer Verbindungsaufbau
- bessere Leistung bei Paketverlust
- moderne Webkommunikation
```

Prüfungsfalle:

```
HTTPS bedeutet nicht immer nur TCP 443,
bei HTTP/3 kann UDP 443 relevant sein.
```

Merksatz:

```
HTTP/3 = HTTP über QUIC über UDP.
```

Statuscodes

HTTP-Statuscodes zeigen das Ergebnis einer Anfrage.

Sie stehen in der HTTP-Antwort.

Beispiel:

```
200 OK
```

Bedeutung:

```
Anfrage erfolgreich.
```

Merksatz:

```
Statuscode zeigt,
wie der Server die Anfrage beantwortet hat.
```

Statuscode-Bereiche

Bereich	Bedeutung
1xx	Information
2xx	Erfolg

Bereich	Bedeutung
3xx	Weiterleitung
4xx	Clientfehler
5xx	Serverfehler

Merksatz:

```
2xx = Erfolg.
3xx = Weiterleitung.
4xx = Clientfehler.
5xx = Serverfehler.
```

Wichtige HTTP-Statuscodes

Statuscode	Bedeutung
200	OK
201	Created
301	dauerhaft weitergeleitet
302	vorübergehend weitergeleitet
304	nicht verändert
400	fehlerhafte Anfrage
401	nicht authentifiziert
403	verboten
404	nicht gefunden
405	Methode nicht erlaubt
429	zu viele Anfragen
500	interner Serverfehler
502	Bad Gateway
503	Dienst nicht verfügbar
504	Gateway Timeout

Merksatz:

```
404 = nicht gefunden.
500 = Serverfehler.
```

```
502 = Gateway-Problem.
503 = Dienst nicht verfügbar.
```

---

401 und 403 unterscheiden

401 bedeutet:

```
nicht authentifiziert
```

Der Benutzer ist nicht oder nicht gültig angemeldet.

403 bedeutet:

```
verboten
```

Der Benutzer ist zwar eventuell bekannt, hat aber keine Berechtigung.

Merksatz:

```
401 = wer bist du?
403 = du darfst das nicht.
```

---

404 Not Found

404 bedeutet:

```
Ressource nicht gefunden
```

Mögliche Ursachen:

```
- falscher Pfad
- Datei fehlt
- Route in Anwendung fehlt
- Link falsch
- Ressource wurde gelöscht
- Rewrite-Regel falsch
```

Merksatz:

```
404 = angeforderte Ressource existiert dort nicht.
```

---

500 Internal Server Error

500 bedeutet:

```
interner Serverfehler
```

Mögliche Ursachen:

```
- Fehler in Anwendung
- Datenbankfehler
- Programmierfehler
- falsche Konfiguration
- fehlende Rechte
- Dienstproblem
```

Merksatz:

```
500 = Server oder Anwendung hat intern ein Problem.
```

---

502 Bad Gateway

502 tritt häufig bei Proxys oder Gateways auf.

Bedeutung:

```
Ein Gateway oder Proxy hat keine gültige Antwort vom Backend erhalten.
```

Mögliche Ursachen:

```
- Backend-Dienst down
- falscher Backend-Port
- Reverse Proxy falsch konfiguriert
- Verbindung zum Backend scheitert
- Backend antwortet ungültig
```

Merksatz:

```
502 = Proxy erreicht Backend nicht sauber.
```

---

503 Service Unavailable

503 bedeutet:

```
Dienst nicht verfügbar
```

Mögliche Ursachen:

```
- Server überlastet
- Wartungsmodus
- Anwendung nicht gestartet
- Backend nicht verfügbar
- Ressourcenmangel
```

Merksatz:

```
503 = Dienst aktuell nicht verfügbar.
```

---

504 Gateway Timeout

504 bedeutet:

```
Gateway Timeout
```

Ein Proxy oder Gateway wartet zu lange auf Antwort vom Backend.

Mögliche Ursachen:

```
- Backend langsam
- Backend hängt
- Netzwerkproblem zum Backend
- Timeout zu kurz eingestellt
- Datenbankantwort dauert zu lange
```

Merksatz:

```
504 = Gateway wartet zu lange auf Backend.
```

Weiterleitungen

Weiterleitungen gehören zu den 3xx-Statuscodes.

Beispiele:

```
301
302
```

301 bedeutet:

```
dauerhaft weitergeleitet
```

302 bedeutet:

```
vorübergehend weitergeleitet
```

Der neue Ort steht häufig im Header:

```
Location
```

Merksatz:

```
3xx bedeutet:
Client soll woanders hin.
```

Cache

Caching bedeutet:

```
Inhalte werden zwischengespeichert.
```

Ziel:

```
schnellere Ladezeiten
weniger Serverlast
weniger Datenverkehr
```

Caches können existieren bei:

```
- Browser
- Proxy
- CDN
- Anwendung
- DNS
```

Merksatz:

```
Cache speichert Daten,
um spätere Anfragen schneller zu beantworten.
```

Cache-Control

Cache-Control ist ein HTTP-Header.

Er steuert, wie Inhalte zwischengespeichert werden dürfen.

Beispiele:

```
Cache-Control: no-cache
Cache-Control: no-store
Cache-Control: max-age=3600
```

Merksatz:

```
Cache-Control steuert HTTP-Caching.
```

API und HTTP

Viele APIs nutzen HTTP oder HTTPS.

Typisch:

URL beschreibt Ressource.

HTTP-Methode beschreibt Aktion.

Header enthalten Zusatzinformationen.

Body enthält Daten.

Statuscode zeigt Ergebnis.

Beispiel:

```
GET /api/users
```

Bedeutung:

Benutzer abrufen.

Merksatz:

Web-APIs nutzen HTTP als Kommunikationsgrundlage.

REST und HTTP

REST nutzt HTTP-Ideen für APIs.

Typisch:

Aktion	HTTP-Methode	Beispiel
lesen	GET	GET /users
erstellen	POST	POST /users
ersetzen	PUT	PUT /users/5
teilweise ändern	PATCH	PATCH /users/5
löschen	DELETE	DELETE /users/5

Merksatz:

REST ordnet Aktionen oft HTTP-Methoden zu.

JSON bei APIs

Viele APIs nutzen JSON als Datenformat.

Beispiel:

```
{
  "id": 5,
  "name": "Felix",
  "rolle": "admin"
}
```

Wichtig:

```
JSON gehört als Datenformat eher zu Schicht 6,
wird aber häufig in Schicht-7-APIs genutzt.
```

Merksatz:

```
APIs nutzen oft JSON für strukturierte Daten.
```

Content-Type bei APIs

Bei APIs ist Content-Type besonders wichtig.

Beispiel:

```
Content-Type: application/json
```

Bedeutung:

```
Der Body enthält JSON.
```

Wenn Content-Type falsch ist, kann der Server die Daten falsch interpretieren oder ablehnen.

Merksatz:

```
Falscher Content-Type kann API-Fehler verursachen.
```

Authentifizierung bei Webdiensten

Webdienste können unterschiedliche Authentifizierungsarten nutzen.

Beispiele:

```
- Benutzername und Passwort
- Session-Cookie
- Bearer Token
- API-Key
- Client-Zertifikat
- OAuth
- OpenID Connect
- Multi-Faktor-Authentifizierung
```

Merksatz:

```
Webdienste müssen oft Benutzer oder Clients prüfen.
```

Bearer Token

Ein Bearer Token ist ein Zugriffstoken.

Es wird häufig im Authorization-Header gesendet.

Beispiel:

```
Authorization: Bearer <token>
```

Wichtig:

```
Wer den Token besitzt,
kann ihn oft verwenden.
```

Deshalb muss er geschützt werden.

Merksatz:

```
Bearer Token wie ein Passwort schützen.
```

CORS kurz erklärt

CORS steht für:

```
Cross-Origin Resource Sharing
```

CORS regelt, ob ein Browser eine Webseite von einer Herkunft auf Ressourcen einer anderen Herkunft zugreifen lässt.

Beispiel:

```
Webseite von app.firma.de
ruft API von api.firma.de auf
```

Wenn CORS nicht passend konfiguriert ist, blockiert der Browser die Anfrage.

Merksatz:

```
CORS ist eine Browser-Sicherheitsregel für Webanfragen.
```

Same-Origin Policy

Die Same-Origin Policy ist eine wichtige Browser-Sicherheitsregel.

Sie begrenzt, wie Webseiten auf Daten anderer Herkunft zugreifen dürfen.

Herkunft besteht aus:

```
- Schema
- Host
- Port
```

Beispiel:

```
https://app.firma.de:443
```

Merksatz:

```
Gleiche Herkunft = Schema, Host und Port passen.
```

Mixed Content

Mixed Content bedeutet:

```
Eine HTTPS-Seite lädt Inhalte über HTTP nach.
```

Beispiel:

```
Hauptseite:
https://wiki.firma.de

eingebundenes Script:
http://example.com/script.js
```

Browser blockieren solche Inhalte oft oder warnen.

Merksatz:

```
HTTPS-Seiten sollten keine HTTP-Inhalte nachladen.
```

Reverse Proxy

Ein Reverse Proxy steht vor einem oder mehreren internen Diensten.

Ablauf:

```
Client
→ Reverse Proxy
→ Backend-Server
```

Typische Aufgaben:

```
- TLS beenden
- Anfragen weiterleiten
- mehrere Dienste über eine Domainstruktur bereitstellen
- Lastverteilung
- Zugriffsschutz
- Header setzen
- Weiterleitungen durchführen
```

Merksatz:

```
Reverse Proxy nimmt Anfragen an und leitet sie intern weiter.
```

Backend

Backend bezeichnet den Dienst, der hinter einem Reverse Proxy arbeitet.

Beispiel:

```
Client ruft https://wiki.firma.de auf.
Reverse Proxy nimmt Anfrage an.
Backend ist der interne Webdienst.
```

Mögliche Fehler:

```
- Backend läuft nicht
- falscher Backend-Port
- falsches Protokoll HTTP/HTTPS
- Firewall blockiert intern
- falscher Pfad
```

Merksatz:

```
Backend = eigentlicher Dienst hinter Proxy oder Anwendung.
```

Host-Header

Der Host-Header sagt dem Webserver, welcher Hostname angefragt wurde.

Beispiel:

```
Host: wiki.firma.de
```

Das ist wichtig, wenn mehrere Webseiten auf derselben IP-Adresse laufen.

Der Server entscheidet anhand des Host-Headers, welche Webseite ausgeliefert wird.

Merksatz:

```
Host-Header ermöglicht mehrere Webseiten auf einer IP.
```

Virtueller Host

Ein virtueller Host erlaubt, mehrere Webseiten oder Dienste auf einem Server zu betreiben.

Beispiel:

```
wiki.firma.de
cloud.firma.de
shop.firma.de
```

Alle können auf dieselbe IP zeigen, aber unterschiedliche Inhalte liefern.

Merksatz:

```
Virtuelle Hosts unterscheiden Webseiten über Hostnamen.
```

Fehlerbild: Webseite nicht erreichbar

Mögliche Ursachen nach Schichten:

```
Schicht 3:
IP, Gateway oder Routing falsch

Schicht 4:
TCP 80 oder 443 nicht erreichbar

Schicht 6:
TLS-Zertifikat fehlerhaft

Schicht 7:
HTTP-Fehler, Anwendung, Pfad oder Backend
```

Merksatz:

```
Webfehler systematisch nach Schichten prüfen.
```

Fehlerbild: HTTP 404

Mögliche Ursachen:

```
- falscher Pfad
- Datei existiert nicht
- Route in Anwendung fehlt
- Reverse Proxy leitet falsch weiter
- falsche Base-URL
- Anwendung erwartet anderen Pfad
```

Merksatz:

```
404 = Ressource unter diesem Pfad nicht gefunden.
```

Fehlerbild: HTTP 500

Mögliche Ursachen:

```
- Fehler in Anwendung
- Datenbank nicht erreichbar
- falsche Rechte
- Konfigurationsfehler
- Abhängigkeit fehlt
- Programmfehler
- Speicher- oder Ressourcenproblem
```

Merksatz:

```
500 = Anwendung oder Server hat intern ein Problem.
```

Fehlerbild: HTTP 502

Mögliche Ursachen:

```
- Reverse Proxy erreicht Backend nicht
- Backend-Port falsch
- Backend-Protokoll falsch
```

```
- Backend abgestürzt
- Firewall blockiert intern
- falscher Container-Name oder DNS-Name
- Timeout oder ungültige Backend-Antwort
```

Merksatz:

```
502 = Proxy-zu-Backend prüfen.
```

Fehlerbild: Login-Schleife

Eine Login-Schleife bedeutet:

```
Benutzer meldet sich an,
landet aber wieder auf Login-Seite.
```

Mögliche Ursachen:

```
- Session-Cookie wird nicht gespeichert
- Cookie-Domain falsch
- Cookie-Pfad falsch
- HTTP/HTTPS-Wechsel
- SameSite-Problem
- Server verliert Session-State
- falsche Systemzeit
- Token ungültig
```

Merksatz:

```
Login-Schleifen sind oft Cookie-, Session- oder Token-Probleme.
```

Fehlerbild: API lehnt Anfrage ab

Mögliche Ursachen:

```
- falsche HTTP-Methode
- falsche URL
```

- fehlender Authorization-Header
- Token ungültig
- falscher Content-Type
- ungültiges JSON
- fehlende Pflichtfelder
- fehlende Berechtigung
- Rate Limit erreicht

Merksatz:

API-Fehler mit Methode, URL, Header, Body und Statuscode prüfen.

Fehlersuche bei Webkommunikation

Eine sinnvolle Reihenfolge:

1. DNS-Name wird korrekt aufgelöst?
2. Ziel-IP erreichbar?
3. TCP-Port 80 oder 443 erreichbar?
4. TLS-Zertifikat gültig?
5. HTTP-Statuscode prüfen.
6. Weiterleitungen prüfen.
7. Header prüfen.
8. Cookies und Session prüfen.
9. Reverse Proxy prüfen.
10. Backend und Anwendungslogs prüfen.

Merksatz:

Bei Webfehlern erst Verbindung,
dann TLS,
dann HTTP,
dann Anwendung prüfen.

HTTP-Logs

HTTP-Logs helfen bei der Fehlersuche.

Typische Informationen:

```
- Client-IP
- Zeit
- Methode
- Pfad
- Statuscode
- Antwortgröße
- User-Agent
- Referer
- Bearbeitungszeit
```

Beispiele für wichtige Fragen:

```
Kommt die Anfrage beim Server an?
Welcher Pfad wird aufgerufen?
Welcher Statuscode wird geliefert?
Ist der Fehler nur bei bestimmten Clients?
```

Merksatz:

```
HTTP-Logs zeigen,
was der Webserver tatsächlich beantwortet.
```

Reverse-Proxy-Logs

Reverse-Proxy-Logs sind wichtig, wenn ein Proxy zwischen Client und Backend steht.

Sie zeigen oft:

```
- eingehende Anfrage
- gewähltes Backend
- Statuscode
- Upstream-Fehler
- Timeout
- TLS-Informationen
- Weiterleitungsprobleme
```

Merksatz:

```
Bei 502, 503 und 504 zuerst Proxy und Backend prüfen.
```

Sicherheit bei HTTP und HTTPS

Wichtige Sicherheitsregeln:

```
- HTTPS statt HTTP verwenden
- Zertifikate aktuell halten
- sichere Cookies setzen
- sensible Daten nicht in URLs schreiben
- Eingaben prüfen
- Authentifizierung schützen
- Rechte sauber prüfen
- unnötige Header vermeiden
- nicht benötigte Dienste abschalten
- Logs datenschutzbewusst behandeln
```

Merksatz:

```
Websicherheit betrifft Transport,
Anwendung
und Benutzerzugriff.
```

Sensible Daten nicht in URLs

Sensible Daten sollten nicht in Query-Parametern stehen.

Problem:

```
URLs können in Logs,
Browserhistorie,
Proxys
oder Referer-Headern auftauchen.
```

Schlecht:

```
/login?password=geheim
```

Besser:

```
sensible Daten im Body übertragen
und HTTPS verwenden.
```

Merksatz:

```
Passwörter und Tokens gehören nicht in URLs.
```

HTTP und Datenschutz

HTTP-Logs können personenbezogene Daten enthalten.

Beispiele:

```
- IP-Adressen
- Benutzerkennungen
- Pfade
- Suchbegriffe
- Tokens in URLs
- Zeitstempel
```

Deshalb wichtig:

```
Logs schützen,
Aufbewahrung begrenzen,
sensible Daten vermeiden.
```

Merksatz:

```
Weblogs können sensible Informationen enthalten.
```

Einordnung in das OSI-Modell

Thema	Schicht
IP-Adresse	3
TCP 80 / TCP 443	4

Thema	Schicht
TLS	6 mit Bezug zu 4 und 7
HTTP	7
HTTPS	7 über TLS
URL	7
HTTP-Header	7
HTTP-Methode	7
HTTP-Statuscode	7
Cookie	7 mit Sitzungsbezug
JSON-Format	6 / 7-Bezug
Reverse Proxy	7 mit Bezug zu 4 und 6

Merksatz:

```
HTTP ist Schicht 7.
TLS schützt darunter.
TCP transportiert darunter.
```

Typische IHK-Fragen

In AP1 und AP2 kann zum Beispiel gefragt werden:

```
- Was ist HTTP?
- Was ist HTTPS?
- Was ist der Unterschied zwischen HTTP und HTTPS?
- Welche Ports nutzen HTTP und HTTPS?
- Was ist ein HTTP-Request?
- Was ist eine HTTP-Response?
- Was ist eine URL?
- Was bedeuten GET, POST, PUT, PATCH und DELETE?
- Was ist ein HTTP-Header?
- Was bedeutet Content-Type?
- Was ist ein Cookie?
- Warum ist HTTP zustandslos?
- Was bedeuten HTTP-Statuscodes?
- Was ist der Unterschied zwischen 401 und 403?
```

- Was bedeuten 404, 500, 502, 503 und 504?
- Was ist ein Reverse Proxy?
- Warum kann eine Webseite trotz offenem Port Fehler anzeigen?

Typische Prüfungsfallen

HTTP gehört zu Schicht 7.

HTTPS gehört zu Schicht 7,
nutzt aber TLS.

HTTP nutzt typischerweise TCP 80.

HTTPS nutzt typischerweise TCP 443.

HTTP/3 nutzt QUIC über UDP.

Port gehört zu Schicht 4.

HTTP-Methode gehört zu Schicht 7.

HTTP-Statuscode gehört zu Schicht 7.

TLS-Zertifikatsfehler sind nicht einfach HTTP-Fehler.

HTTP ist grundsätzlich zustandslos.

Cookies helfen bei Sitzungen.

GET sollte Daten abrufen.

POST sendet Daten an den Server.

401 bedeutet nicht authentifiziert.

403 bedeutet verboten.

404 bedeutet nicht gefunden.

500 bedeutet interner Serverfehler.

502 deutet häufig auf Proxy-Backend-Problem.

503 bedeutet Dienst nicht verfügbar.

504 bedeutet Gateway Timeout.

Offener TCP-Port 443 heißt nicht,
dass die Webanwendung korrekt funktioniert.

Wichtige Begriffe kurz erklärt

Begriff	Kurze Erklärung
HTTP	Webprotokoll
HTTPS	HTTP über TLS
Request	Anfrage des Clients
Response	Antwort des Servers
URL	Adresse einer Ressource
Schema	Protokollteil einer URL
Hostname	Name des Servers
Pfad	angeforderte Ressource
Query	Zusatzparameter in URL
Methode	Aktion einer HTTP-Anfrage
Header	Zusatzinformationen
Body	Nutzdaten einer Anfrage oder Antwort
Content-Type	Format des Inhalts
Cookie	gespeicherte Webinformation
Session-Cookie	Cookie zur Sitzungszuordnung
Statuscode	Ergebnis einer HTTP-Anfrage
Redirect	Weiterleitung
Cache	Zwischenspeicher
API	Anwendungsschnittstelle
REST	API-Architekturstil
Bearer Token	Zugriffstoken

Begriff	Kurze Erklärung
CORS	Browser-Regel für fremde Herkunft
Reverse Proxy	vorgelagerter Webserver
Backend	interner Ziel-Dienst

IHK-sichere Kurzformulierung

HTTP ist ein Anwendungsschicht-Protokoll zur Übertragung von Webinhalten. Es arbeitet mit Requests und Responses. Ein Client fordert eine Ressource über eine URL an, der Server antwortet mit Headern, einem Statuscode und optionalem Inhalt. HTTPS ist HTTP über TLS und schützt die Übertragung durch Verschlüsselung, Integrität und Serverauthentifizierung. HTTP nutzt typischerweise TCP-Port 80, HTTPS typischerweise TCP-Port 443. HTTP-Methoden wie GET, POST, PUT, PATCH und DELETE beschreiben die gewünschte Aktion. HTTP-Statuscodes zeigen das Ergebnis einer Anfrage an. Fehler wie 404, 500, 502, 503 oder 504 helfen bei der Fehlersuche auf Anwendungsschicht.

Merksätze

HTTP = Hypertext Transfer Protocol.

HTTPS = HTTP über TLS.

HTTP gehört zu Schicht 7.

HTTPS gehört zu Schicht 7 mit TLS-Bezug.

HTTP nutzt typischerweise TCP 80.

HTTPS nutzt typischerweise TCP 443.

HTTP/3 nutzt QUIC über UDP.

Client stellt Request.

Server sendet Response.

URL zeigt Ressource.

Hostname wird per DNS aufgelöst.

GET ruft Daten ab.

POST sendet Daten.

PUT ersetzt vollständig.

PATCH ändert teilweise.

DELETE löscht.

Header enthalten Zusatzinformationen.

Content-Type beschreibt das Datenformat.

Cookie hilft bei Sitzungen.

HTTP ist zustandslos.

HTTPS schützt HTTP mit TLS.

Erst TCP,
dann TLS,
dann HTTP.

2xx = Erfolg.

3xx = Weiterleitung.

4xx = Clientfehler.

5xx = Serverfehler.

401 = nicht authentifiziert.

403 = verboten.

404 = nicht gefunden.

500 = interner Serverfehler.

502 = Bad Gateway.

503 = Dienst nicht verfügbar.

504 = Gateway Timeout.

Offener Port heißt nicht:
Webanwendung funktioniert.

Bei Webfehlern DNS,
IP,
Port,
TLS,
HTTP
und Anwendung prüfen.