ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

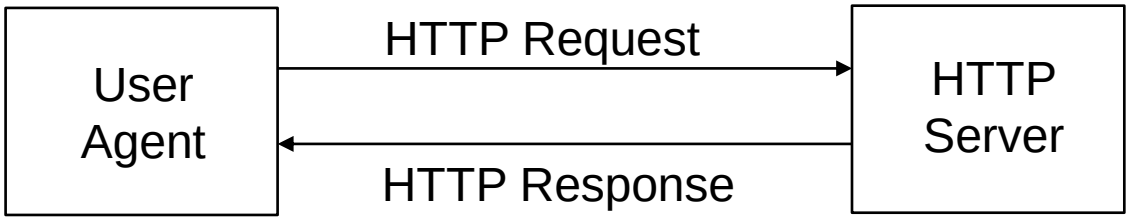
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

HTTP Protocol

Αθ. Ανδρούτσος

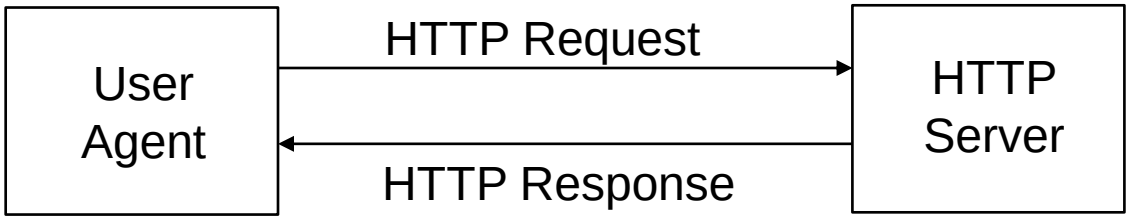
Το πρωτόκολλο HTTP (1)

- Το πρωτόκολλο HTTP (Hypertext Transfer Protocol) είναι ο βασικός μηχανισμός μεταφοράς πληροφοριών της υπηρεσίας World Wide Web. Η αρχιτεκτονική είναι client-server
- Ο User Agent (Client) αποστέλλει ένα αίτημα (HTTP Request Packet) με τη χρήση του πρωτοκόλλου TCP στον Web Server (HTTP Server).
- Ο Web Server αποστέλλει πίσω ένα HTTP response Packet



Το πρωτόκολλο HTTP (2)

- Ο User Agent μπορεί να είναι είτε ένας Web Browser ή ένα πρόγραμμα
- Ο προορισμός ενός HTTP Request καλείται πόρος (resource). Ένα resource μπορεί να αντιστοιχεί σε ένα **φυσικό αρχείο** ή να είναι **μία υπηρεσία** που παρέχεται μέσω του Web Server. **Κάθε resource γίνεται identify από ένα URI (Uniform Resource Identifier)**



Το πρωτόκολλο HTTP (3)

- Τα URLs (Uniform Resource Locators) είναι ένας τύπος URI
- Μέσω των URIs ο User Agent επικοινωνεί με τον HTTP Server και στη συνέχεια ο Server αποστέλλει πίσω τα δεδομένα που ζητήθηκαν που μπορεί να είναι ένα αρχείο όπως HTML σελίδα (καθώς και CSS/JS αρχεία, εικόνες, κλπ. static data) ή δυναμικά δεδομένα που μορφοποιούνται ως JSON strings

URIs

- Τα URIs ορίζονται στο RFC 2396 (Request For Comments). Παραδείγματα:
- https://www.aueb.gr
- https://www.aueb.gr/docs/successstory
- https://www.aueb.gr/search?stream=web&item=css

URIs Syntax

- Έστω το URL https://www.aueb.gr:80/path/to/myfile.html? key1=value1&key2=value2
  - https:// *το πρωτόκολλο*
  - www.aueb.gr *domain name*
  - 80 *port*
  - path/to/myfile.html - φυσικό αρχείο ή συνήθως ένα abstract path (χωρίς κατάληξη) που το χειρίζεται ο server και το αντιστοιχεί σε μία υπηρεσία
  - ?key1=value1&key2=value2 *query parameters* Διαχωρίζονται με το &. Πρόκειται για παραμέτρους που στέλνει ο client στον server για filtering των δεδομένων που θα στείλει πίσω ο Server

Εκδόσεις HTTP

- 1989 – HTTP/0.9
- 1996 – HTTP/1.0 (RFC 1945 both 0.9 & 1.0 versions) - ένα connection per resource request
- 1997 – HTTP/1.1 (RFC 2616) - ένα connection per multiple resource requests
- 2015 – HTTP/2 (RFC 7540) - ένα connection per server domain. Επίσης, ενώ πιο πριν μόνο ο client έκανε initiate ένα connection, τώρα μπορεί και ο Server να στείλει, ασύγχρονα, πακέτα (push capability)

HTTP/3

- 2022 – HTTP/3 (RFC 9114). Αντί για TCP, χρησιμοποιεί UDP και το Google QUIC, που κάνει το ίδιο congestion control και packet resend (μιας και το UDP είναι best-effort protocol και δεν κάνει congestion control και επαναμεταδόσεις πακέτων όπως το TCP)

HTTP & SSL

- Το HTTP protocol λειτουργεί μέχρι την έκδοση HTTP/1 over TCP connection ή πλέον κυρίως over TLS δηλ. **HTTPS στο port 443** – Transport Layer Security, παλαιότερα SSL – Secure Socket Layer
- Το HTTP/3 λειτουργεί over QUIC και over UDP (και όχι over TCP)

HTTP/0.9

- Η αρχική έκδοση του HTTP/0.9 ήταν πολύ απλή
- Το request packet περιείχε μόνο την εντολή GET και το αρχείο που ζητείται (αφού πρώτα είχε γίνει σύνδεση στον απομακρυσμένο server). Η μόνο μορφή αρχείου που υποστήριζε ήταν html
- Το response περιείχε μόνο ένα html αρχείο

request

```
GET /mypage.html
```

response

```
<html>
  A very simple HTML page
</html>
```

HTTP/1.0

- Η έκδοση 0.9 δεν περιείχε σημαντικά χαρακτηριστικά όπως:
  - Το **status code** το οποίο προστέθηκε στην 1η γραμμή του response header ώστε ο browser να λαμβάνει γνώση για το success ή το failure (π.χ. 200 OK)
  - Το **Content-Type** στο response header που περιείχε τον τύπο του επιστρεφόμενου αρχείου ώστε να μην υποστηρίζονται μόνο HTML αρχεία (π.χ. text/html)

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

```
HTTP/1.0 200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
A page with an image
  <IMG SRC="/myimage.gif">
</HTML>
```

Επόμενα requests

- Στη συνέχεια και μετά το fetch του html αρχείου μπορεί να στέλνονται και άλλα requests για τυχόν resources που χρειάζεται το HTML αρχείο, π.χ. images

```
GET /myimage.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

```
HTTP/1.0 200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

HTTP/1.1 - Request


```
GET /en-US/docs/Glossary/CORS-safelisted_request_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/CORS-safelisted_request_header
```

- Η βασική αλλαγή στο HTTP/1.1 ήταν η δυνατότητα να **διατηρείται το connection** και όχι να γίνεται ένα connection για κάθε request
- Το **Content negotiation** περιλαμβάνει language, content και encoding
- Σημαντική αλλαγή ήταν και το **Host** ώστε να μπορεί το ίδιο IP να έχει πολλούς virtual hosts (ένας Web Server με πολλά domains)
- Ο **Referer** είναι το URL από όπου προέρχεται το αίτημα

HTTP/1.1 - Response

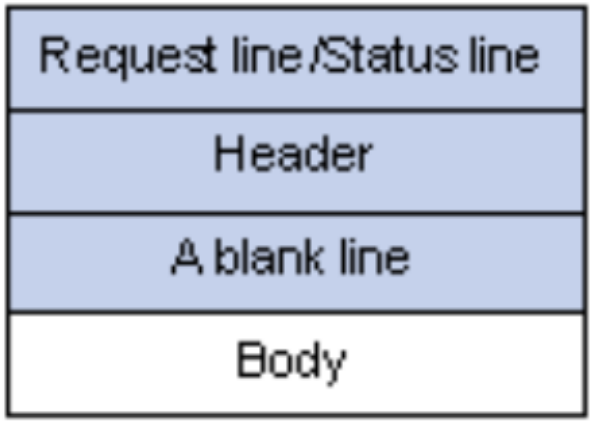

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding

(content)
```

- Στο response του server το **Connection: keep-alive** αφορά τη διατήρηση του connection
- Το **Etag** είναι signature (hash) του resource ώστε να ξανακατέβει από τον server μόνο αν αλλάξει (cache-control σε συνδυασμό με το If-None-Match: “ETag” στο επόμενο request)
- Το **Keep-Alive** αφορά το TCP connection (5 sec, max 1000 requests στο ίδιο connection)
- Το **Transfer-Encoding** αφορά την αποστολή των data σε chunks (στο ίδιο όμως response)
- Το **Vary** αφορά την αποστολή διαφορετικού περιεχομένου ανάλογα με τον χρήστη που έχει κάνει login

HTTP Packet

- Γενικά, το πρωτόκολλο HTTP είναι ένα πρωτόκολλο του επιπέδου 7 και χρησιμοποιεί τη δομή των πακέτων που έχουμε δει στα TCP/IP δίκτυα



- Το HTTP Packet αποτελείται από τα δεδομένα του χρήστη (payload στο body) και από πάνω το HTTP Header (η 1η γραμμή είναι το request line ή το response status line, αντίστοιχα για client ή server).
- Η δομή των HTTP πακέτων που στέλνει ο χρήστης και των πακέτων που αποστέλλει ο server έχουν ίδια δομή αλλά διαφέρουν ως προς τα πεδία μέσα στον Header

Λειτουργίες HTTP (1)

- Το πρωτόκολλο HTTP ορίζει μεθόδους (methods) για την εκτέλεση πράξεων στους πόρους του δικτύου. Οι πιο σημαντικές είναι:
  - **GET, POST, PUT, DELETE , PATCH, HEAD, TRACE, OPTIONS**
- Οι παραπάνω μέθοδοι αναφέρονται ως η 1η λέξη στο Request Line του HTTP Packet
- Για παράδειγμα
  - POST /path/to/resource HTTP/1.1
  - GET /path/index.html HTTP/3
  - DELETE /path/to/resource HTTP/1.0
  - PUT /path/to/resource HTTP/2

Λειτουργίες HTTP (2)

- **GET**: η μέθοδος get αιτείται από τον HTTP Server ενός πόρου, όπως για παράδειγμα μια σελίδα .html, μια εικόνα, ένα αρχείο ή μία υπηρεσία. Μπορούμε να περάσουμε και παραμέτρους στο URL της GET με ?key1=value1&key2=value
- Για παράδειγμα η εντολή: GET /myDir/aueb.jpg HTTP/1.1 αιτείται της εικόνας aueb.jpg που βρίσκεται στο ευρετήριο myDir.
- **HEAD**: Αιτείται της επικεφαλίδας ενός πόρου. Οπότε δεν λαμβάνεται το body

Λειτουργίες HTTP (3)

- **POST**: Αποστέλλει δεδομένα του χρήστη, για παράδειγμα τα πεδία μιας φόρμας, στον εξυπηρετητή. Τα δεδομένα δεν είναι στο URL όπως στην GET αλλά στο body του request packet.
- **PUT**: "Ανεβάζει" (uploads), στον εξυπηρετητή πόρους
- **DELETE**: Διαγράφει πόρους στον εξυπηρετητή πόρους
- **TRACE:** Η μέθοδος HTTP TRACE έχει σχεδιαστεί για διαγνωστικούς σκοπούς. Εάν ενεργοποιηθεί, ο Web Server θα κάνει echo το μήνυμα που ελήφθη

Λειτουργίες HTTP (4)

- Το HTTP δεν γνωρίζει τα στοιχεία των χρηστών που εξυπηρετεί. Δεν διατηρεί δεδομένα (state) για την κατάσταση της σύνδεσης, οπότε λέμε πως **το HTTP είναι stateless**. Επομένως, το HTTP δεν είναι προσανατολισμένο σε συνδέσεις, αλλά εξυπηρετεί μεμονωμένα requests
- Το γεγονός αυτό δυσχεραίνει την ανάπτυξη εφαρμογών, όπου χρειάζεσαι να είναι γνωστή η κατάσταση μιας σύνδεσης, όπως όταν έχει **login**
- Ο πιο κοινός τρόπος για την επίλυση αυτού του προβλήματος είναι η χρήση μικρών αρχείων που περιέχουν πληροφορίες κατάστασης του χρήστη και της σύνδεσης και είναι γνωστά ως **cookies ή η χρήση JSON Web Tokens**

Δομή HTTP Request Packet

Method	URL	Version		
Header field Name		:	Value	
Header field Name		:	Value	
Header field Name		:	Value	
Δεδομένα (Data)				

← Request Line

← Headers

← Data (Payload)

Request Line / Header Lines

- Το **Request-Line** όπως αναφέραμε περιέχει τρείς τιμές: **Method URI HTTP-Version\r\n**
- Π.χ. POST / HTTP/1.1
- Μετά το Request Line έχουμε μία ακολουθία (possible zero) από HTTP Header Lines
- Κάθε Header-Line περιέχει ένα key value pair της μορφής: **attribute : value**

Header Lines

- Τα **Request Header-Lines** περιέχουν τις παρακάτω κατηγορίες πληροφοριών
  - **General headers**, π.χ. *Connection*: keep-alive (παραμένει ανοικτό το TCP connection μεταξύ διαφορετικών requests, default στο HTTP/1.1)
  - **Request Headers**. Π.χ. *User-agent*: Chrome/97.0, *Accept*: text/html, κλπ. Το HTTP 1.1 απαιτεί ένα host header π.χ. *Host*: localhost:8080

Παράδειγμα Request Line & Header-Lines


```
POST /form/insertstudent.html HTTP/2.0
Host: codingfactory.aueb.gr
User-Agent: Mozilla/4.0
Accept: text/html, */*
Content-type: multipart/form-data; boundary="auto_bound"
--auto_bound
 Content-Disposition:  form-data;  name="id"

5
--auto bound
Content-Disposition: form-data; name="lastname"

Wonderland
--auto bound
Content-Disposition: form-data; name="firstname"

Alice
--auto bound
```

Content-type boundary

- Το boundary στο multipart content-type είναι ένα string για να ξεχωρίζουμε τα μέρη του data μιας και είναι multipart δηλαδή τα data περιέχουν πολλά αρχεία ή πεδία διαφορετικού τύπου και μεγέθους
- Όλα τα boundaries ξεκινάνε με δύο hyphens (--). Το τελευταίο boundary τελειώνει με two hyphens (--)

Παράδειγμα Post Request

- POST /than/cfservlet HTTP/1.1
- Host: www.aueb.gr
- User-Agent: Chrome
- Accept: */*
- Content-Type: application/x-www-form-urlencoded
- Content-Length: 35
- <Blank Line>
- lastname=androutsos&firstname=thanassis&id=1

(Στην τελευταία παραπάνω γραμμή είναι το body του HTTP Packet)

MIME Types (1)

- Τόσο το **Content-Type** όσο και το **Accept** μπορεί ως τιμές να περιέχει MIME Types (Multi-purpose Internet Mail Extensions) δηλαδή τον τύπο των δεδομένων
- Η σύνταξη είναι στη μορφή **Type/Subtype**
- Η γενική κατηγορία περιεχομένου μπορεί να είναι: ***text***, ***image***, ***video***, ***application***, κλπ.

MIME Types (2)

- Το subtype είναι ο πιο ειδικός τύπος data, π.χ. για text μπορεί να είναι ***plain, css, html, JavaScript, csv***, κλπ.
- Μπορεί να υπάρχει προαιρετικά και μία παράμετρος της μορφής type/subtype; parameter=value
- Π.χ. **text/plain; charset=UTF-8**

MIME Types (3)

- Στο Mime Type/Subtype, το Type μπορεί να είναι discrete ή multipart
- Discrete
  - application/pdf, application/zip
  - text/html
- Multipart
  - multipart/form-data, που χρησιμοποιείται σε POST methods για HTML Forms

MIME Types (4)

- Τα MIME types & subtypes είναι case sensitive
- Ο γενικός τύπος των MIME Types είναι:
- type "/" [tree"."] subtype ["+" suffix]
- Π.χ. application/vnd.api+json είναι το content-type όταν ο client δέχεται json strings από κάποιο vendor API

Δομή HTTP Response Packet

HTTP-Version	Status-Code	Message	← Status Line
Header field Name	:	Value	← Headers
Header field Name	:	Value	
Header field Name	:	Value	
Δεδομένα (Data)			← Data (Payload)

Το *Status Code* είναι 3-digit number (για non-humans). Το Message είναι text με το ίδιο νόημα με το status-code αλλά για humans. Παράδειγμα Status Line: HTTP/1.1 200 OK

Παραδείγματα Status Line

- HTTP/1.1 200 OK (Response includes a response body)
- HTTP/1.1 201 CREATED
- HTTP/1.1 204 No Content (No response body, όπως σε PUT, DELETE Requests)
- HTTP/1.1 301 Moved Permanently
- HTTP/1.1 400 Bad Request
- HTTP/1.1 500 Internal Server Error

Status Codes

- Τα HTTP Status Codes είναι τιμές που επιστρέφουν από τον Server στον client ως επιστρεφόμενες τιμές επιτυχούς ή όχι ολοκλήρωσης του request
- Τεκμηριώνονται στο RFC 9110
- Ομαδοποιούνται σε πέντε κλάσεις, 100-199 , 200-299 , 300-399 , 400-499 , 500-599

Κλάσεις Status Codes

- Ομαδοποιούνται σε πέντε κλάσεις:
  - 100-199. Informational responses
  - 200-299. Successful responses, π.χ. 200 OK, 201 Created, 204 No Content
  - 300-399. Redirection Messages, π.χ. 301 Moved Permanently
  - 400-499. Client Error Responses, π.χ. 400 Bad Request, 401 Unauthorized, 404 Not Found
  - 500-599. Server Error Responses, π.χ. 500 Internal Server Error

Response Header


```
HTTP/2 200 OK
Access-Control-Allow-Origin: *
Date: Mon, 10 Feb 2024 16:06:00 GMT
Set-Cookie: mykey=myvalue; expires=Mon, 20-Jul-2023 16:06:00 GMT;
Server: Apache Tomcat/9.0.48
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Content-Length: 1234

<!DOCTYPE html>
<html>
<head>
   <title>Example Page</title>
</head>
<body>
   <h1>Hello, World!</h1>
   <p>This is an example HTML page served by Apache Tomcat.</p>
</body>
</html>
```

CORS (1)

- Το Cross-Origin Resource Sharing (CORS) είναι μία πολιτική ασφάλειας που υλοποιείται από τους servers **ώστε να ορίζεται ποιος client μπορεί να καλεί υπηρεσίες του server**
- **Origin**: Είναι μία τριπλέτα: scheme/host/port που μοναδικοποιεί ένα μέρος ενός URL, π.χ. το **http://www.aueb.gr:8080** είναι ένα origin
- Οι browsers (η JavaScript) **επιτρέπουν by default μόνο αυτό που λέμε same-origin content**, δηλαδή μόνο από το ίδιο domain μπορούν να γίνονται requests
- Αυτό μπορεί να αλλάξει από τον server, ο οποίος να επιτρέπει requests από οποιοδήποτε origin με το **Access-Control-Allow-Origin** response header

CORS (2)

- Αν για παράδειγμα ένας browser κατεβάσει μία σελίδα από το https://aueb.gr και αυτό το response δεν επιστρέψει κάτι σχετικό με Access-Control-Allow-Origin, τότε υπονοείται πολιτική same-origin
- Αν κάποιο AJAX JavaScript από άλλο domain κάνει κλήση προς τον aueb.gr Web Server και την εφαρμογή μας θα απαγορευτεί και θα επιστρέψει error

HTTP Session Management (1)

- Η έννοια του ‘Session‘ υπάρχει ήδη από το πρωτόκολλο TCP το οποίο είναι προσανατολισμένο σε **συνδέσεις απ’ άκρη σ’ άκρη (connection-oriented communication)**
- Στην ορολογία του TCP τα sessions ονομάζονται virtual paths, ενώ το κάθε endpoint χρησιμοποιεί ένα συνδυασμό IP Address και port ώστε να μοναδικοποιεί το virtual path σε κάθε endpoint

HTTP Session Management (2)

- Connection-oriented communications μπορούν να υλοποιηθούν εκτός από το transport layers και σε άλλα layers όπως στο session layer του ISO/OSI, με το SIP (Session Initiation Protocol) για voice over IP και στο application layer, όπως με τα HTTP Sessions
- Επίσης, connection-oriented είναι και το QUIC (HTTP/3) που τρέχει over UDP

HTTP Session Management (3)

- Το πρωτόκολλο HTTP είναι connectionless και όχι προσανατολισμένο σε σύνδεση. Για κάθε HTTP packet που φτάνει στον Server, ο Server δεν αποθηκεύει πληροφορίες (state) σύνδεσης
- Υπάρχουν ωστόσο εφαρμογές βασισμένες στο HTTP, όπου θα πρέπει ο Server να γνωρίζει το session στο οποίο ανήκει ένα request από τον client ώστε να μπορεί να προσφέρει προσωποποιημένες υπηρεσίες, όπως καλάθι αγορών και γενικά υπηρεσίες που έχουν τη λογική του login

Session management με Cookies

- Η βασική ιδέα είναι ότι όταν ένας χρήστης **γίνεται authenticate μέσω των user credentials όπως username και password**, ο **Server σε περίπτωση επιτυχούς authentication να στέλνει πίσω στον browser/client ένα μικρό κομμάτι δεδομένων (cookie)**
- Στη συνέχεια η επικοινωνία του client με τον Server γίνεται **μέσω της αποστολής του cookie από τον client στον server σε κάθε επικοινωνία που ανήκει στο ίδιο session**

REST API & JWT Authentication

- Στην αρχιτεκτονική REST δεν έχουμε sessions. Η επικοινωνία είναι stateless. Η βασική ιδέα είναι ότι όταν ένας χρήστης **γίνεται authenticate μέσω των user credentials όπως username και password**, ο Server σε περίπτωση επιτυχούς authentication**, στέλνει πίσω στον browser/client ένα μικρό κομμάτι δεδομένων, που ονομάζεται access token**. Μία ειδική μορφή access token σε μορφή JSON, είναι τα JWTs (**JSON Web Tokens**)
- Τα JWTs είναι αυτόνομα authentication tokens και δεν χρειάζεται ο Server να κρατάει κάποιο state για να τα επιβεβαιώσει
- Στη συνέχεια η επικοινωνία του client με REST Endpoint του Server γίνεται μέσω της αποστολής του JWT από τον client στον server σε κάθε επικοινωνία