HTTP (Hyper Text Transfer Protocol)

* Protocol la nivelul aplicatiei
* Folosit pentru a stabili legatura client – server

Servlet -> web applications

<https://www.javatpoint.com/servlet-tutorial>

Sapt 3 -> baze de date

Sapt 4 -> incepem aplicatie cu specificatii -> Spring

Sapt 5 -> Hibernate

Sapt 6 – Sapt 8 -> clean clode, cloud architecture

Baze de date, business, controller

14.03.2022 Aplicatii web:

= foloseste infrastructura internet web pt a comunica intre client – server

* Protocol TCP/IP + HTTP
* Concept de stateless si statefull
* E de tip stateless -> fiecare request e independent de alt request, raspunsul serverului e dat de starea lui actuala
* Avem un protocol stateless -> in care exista o stare
* OSI Model - <https://www.imperva.com/learn/application-security/osi-model/>

<https://www.geeksforgeeks.org/layers-of-osi-model/>

* Putem avea probleme de performanta –> clientii trb sa initieze conexiuni -> dc sunt multi client, serverul nu o sa merga bine

<https://www.geeksforgeeks.org/tcp-ip-model/>

* TCP – protocol
* UDP – protocol
* HTTP -> protocol al application layer si e folosit in aplicatiile web
* Exista doua variante: HTTP si HTTPs (secured)
* Versiunea 2 a HTTP e in format binar

**HTTP**

* Ajuta la transfer de date intre un client si server -> clientul de obicei e browserul

Ex: browser -> site

Site raspunde cu un html, cu imagini, media resources(video, youtube integrat), css (info despre font, culori, margini, clasa toogle (referinta catre css)), date javascript (script care poate fi rulat -> fiecare browser poate sa il ruleze ca pe orice program -> react, nodejs)

* Client -> proxy -> server si invers
* Load balancer -> trimite requesturi catre fiecare masina
* Server -> aplicatie care stie sa primeasca cereri http si raspunde cu html
* Proxyurile pot face: cache-ing, filter-ing, loadbalance-ing
* Intr-un mod round robin(=fiecare pe rand) -> trimite un request

HTTP -> protocol text -> requestul e un text si raspunsul e tot text

* E extensibil = avem headers - >ne ajuta sa extindem functionalitatea http-ului
* Stateless
* Sesiune = dc mergem pe un site de cum psi ne uitam la produse, add in cos si la final vrem sa cumparam - >am facut multe requesturi si cand fac plata finala, serverul trb sa stie tot ce am facut pana in mom ala, ce am add in cos => info / obiecte tinute pe session (session id e trimis cu fiecrae request)
* http -> se bazeaza pe TCP adica inainte sa trimit de la browser ctare server, browserul trb sa initieze o conexiune
* http headers -> sa nu inchida sesiunea si sa o foloseasca de mai multe ori la sesiuni consecutive
* request contine:
  + metode
  + path (resursa pe care vrem sa o accesam ex. index.html)
  + protocol
  + versiune
  + header -> nume si valoare
* raspuns:
  + status code
  + status message
  + protocol
  + versiune
  + headers: nume + valoare
  + body

HTTP Methods

GET -> aducem informatii (retrieve information)

HEAD -> similar cu GET dar nu da Body

POST -> trimitem info (ex. submit form)

PUT -> trimitem info dar face update (RESTFULL API)

DELETE -> stergem resursa

CONNECT ->

OPTIONS ->

TRACE ->

Status codes:

* 1xx -> informational
* 2xx -> successful
* 3xx -> redirection
* 4xx -> client error
* 5xx -> server error

Headers

* Folosite ca sa controlam anumite functionalitati
* General Header -> pot fi atat pe request cat si pe response si sunt si particular

Ex. putem controla cache-ul

Header de connection -> “Connection: keep-alive”

Tema:

Caching, cookies

App web

15.03.2022

HTTP -> port 80

App web dinamice request -> response dynamic

In raspunsul pe care il construieste la client sa fie -> genereaza html -> trb construit un html dynamic

<https://www.baeldung.com/intro-to-servlets>

Servlet -> clasa in Java care implementeaza ce vrem sa faca un anumit request

* Pt ca avem requesturi HTTP avem nevoie de cineva care stiesa le asculte si sa le faca accesibile codului java -> asta se intampla intr-un Servlet Container
* Servlet Coantainer -> ii dam un servlet si deleaga mai departe servletul
* Servlet lifecycle:
  + init():

Se apleleaza o singura data

* + service()

apelata dupa init(), interpreteaza requestul HTTP (GET, POST, PUT, DELETE etc.)

* + destroy()

called to take the Servlet out of service

Tomcat -> face matching intre url-ul definit pe request si url-ul pe servlet

Trb sa pornim tomcat si sa ii dam servlet-ul nostrum ca sa stie la requesturile pe care le porneste la servletul nostrum

Se face prin generearea unui fis care se numeste war.class (e ca si jar) care continue clasa compilata impreuna cu alte fis pe care o app web le poate avea

* app web se inglobeaza intr-un war –> e practic o arhiva care continue foldere si fisiere
* download tomacat
* avem clasa noastra impementata, din clasa construim un war, war ul in punem in tomcat de mana
* va trebui sa pornim startup din linia de comanda
* webapps

Maven

Servlet get si post, deploy tomcat

Tema proiect maven, construim servlet, deploy tomcat

get si post