HTTP and Web Architecture

Servlet is a java class that manage HTTP request and responses.

HTTP methods (verbs) → Used to tell the server actually what kind of action we want to take.

1. GET: you just want to read the information.
2. POST: You are sending new data to be processed.
3. PUT: Replacing something that already exists
4. DELETE: You are removing something.

Job of Servlet:

1. Listen for request
2. check for which method was used.
3. Run specific java code for that method.
4. Sends a response back to you.

Post contains data in form of packages, meaning data is hidden inside the body of the request.

Idempotency

if doing a action changes nothing it is then

Idempotent → Turning off a already off button.
But if something does end up changing it is non idempotent. → Increasing the volume.

GET → Idempotent POST → non-Idempotent

PUT / PATCH → ,,

↓ updates the whole record

↳ changes the specific targetted data only. like: changing Age.

Understanding Status Codes (ANSWER)

Everytime servlet finishes its job it sends back a 3 digit response.

200 → OK (data found)

201 → created (User Saved)

204 → the request was sent successfully but there is no response to return
Eg: DELETE

300 → Multiple choices.

400: Bad request is found when user either leaves the feild blank or sends in data of diff. data type. Like integer implace of string.

401: Unauthorized, user enters wrong credentials.

403: forbidden, User lacks permissions to

access the specific resource

404 → Nof found (URL is wrong)

409 : conflict happens when the request is completely formatted but it clashes with what's already on the server.

Eg → Signing up with an email that has already been signed up with.

500 → Internal Server Error

502 → Bad Gateway (Recieved an invalid response from upstream server.

503 → Service Unavailable, due to overloading or Maintainence

Request & Response headers

Headers are the metadata of the request.

↳ data about the data

In servlet these are key value pairs –

(A) Request headers (The client's "context")

These tell the servlet here is what I'm sending and here's what I can handle.

- content-Type : Tells the servlet how to parse the body

  Eg → application/json means the body is a JSON

U object.

- Authorization: The security badge
  Eg → Bearer (JWT-Token)

  Servlet filter reads this error, validates token, and decides if the user is allowed to enter.

- Accept: The "language" preference
  Eg → text/html (Browser wants webpage)
  application/json (Mobile app wants raw data)

(B) Response Headers (The server's "Instructions")

These tell the browser: here's how you should treat what I just sent you.

- content-Type: Tells the browser how to render the data
- Location: Used for redirects
- Cache-Control: The "Expiration date".

→ HTTP is stateless, it doesn't remember anything after giving a response, then how does Servlet remember who you are, every time you log-in? We use headers to carry who we are and what we are sending.

Exact process: ① You log in (1st Time) → ② Servel creates a Session Id for you. ↓ ③ after servlet ④ Browser ⑤ when you ⑥ Servletsees

the I'D. checks it's folders and says "It's you again". ← open website again, browsers puts Id back into cookie header ← auto-saves that Id ← respons back, it sends back your id in a special header called set-cookie

This whole process comes under JWT i.e.

JSON Web Token.

Authorization: Bearer <eyth...>

REST API principle & constraints

↓

Representable State Transfer are basically the rules

if your servlet follows these rules, it's RESTful

Principle	Meaning
Stateless	No "Memory" on server.
Uniform Interface	you use standart methods (GET, POST) and resources have unique ID's (URLs)
Client-Server	The user client and data (Server) is diff
Cacheble	Resource should define if they can be cached to save time.

Cross origin Resource Sharing (CORS)

When building API's the hurdle faced by dev's is if my server is running on api.myapp.com but frontend website is

running on www.myapp.com. Then by default browser block the API from talking to API because they are on diff. origins.

To handle these we need to add specific headers to your response to tell the browser "It's okay I trust this website".

i.e.

Access-Control-Allow-Origin: https://www.my....

But before browser actually uses this it sends an pre-flight Request using OPTIONS method

what this request does is it sends a post request to API and asks is it cool to send the JSON body to you from that website? If the API accepts it and you have 200/201 OK then you're good to send the actual request.

→ Servlet is controlled by servlet container.

The container:

1. Creates the servlet
2. Calls it's methods
3. destroys it

Servlet life cycle (3 main methods)

① init() → Sets things up

- Called once
- When servlet is created
- Used for initialization

Example uses – . database connection setup
. loading configs

② service()

- Called for every request.
- This is where work happens

→ We usually don't override service() directly
Instead we do: • doGet()
• doPost() etc

③ destroy() → does the clean up

- Called once
- When server shuts down or servlet is removed

Example: Closing DB connections

lifecycle flow

Servlet loaded
↓
init() once
↓
service() → doGet() / doPost() many

↓

destroy() once

HTTP Servlet & Request handeling

Why HTTP Servlet?

because HTTP has methods —

- GET
- POST
- PUT
- DELETE

HTTPServlet gives you readymade hooks for them.

Mapping HTTP methods

HTTP method	Servlet Method
① GET	doGet()
② POST	doPost()
③ PUT	doPut()
④ DELETE	doDelete()

forward vs Redirect

① RequestDispatcher.forward()

↳ Internal transfer inside server

- URL in browser does not change.
- Same request object
- Faster

Used when - • Internal flow
• MVC-style chaining

Eg - Client → Servlet A → Servlet B

(2) sendRedirect()

↳ Tells browser to go somewhere else

- Browser URL changes
- New request
- Slower

Eg - Client → Servlet A → Client → Servlet B

Session Management

HTTP is stateless

So we need ways to remember users.

(1) HTTP Session (Most used)

- Stored on server
- Session ID stored in cookie
- Automatic expiry

```
HttpSession session = request.getSession();
session.setAttribute("user", "Pankaj");
```

Used for : • login
• user state

(2) Cookies

- Stored on client
- Small size
- less secure

```
Cookie c = new Cookie("theme", "dark");
response.addCookie(c);
```

Used for - • preferences

- packing

Servlet context vs ServletConfig

• Shared by all servlets	• Config for one servlet
• Used for App-wide data	• initialization parameters

Filters → It is a gatekuper before servlet

- Intercept requests
- Can block or modify it
- pass it forward

Use cases -
- Authentication
- logging
- CORS
- Imput validation

filter chain

Request
↓
Filter 1
↓
Filter 2
↓
Servlet
↓
Response (Reverse Order)

Authentication using filter

① Check session

check session

② If user logged in → Allow
③ Else redirect to login

filter

Concurrency

How do servlet handles multiple users?

One Servlet → multiple threads

Meaning a servlet may be one but it contains multiple threads

Therefore when a request is recieved, container assigns a single thread from thread pool and handles the request.

Need of Spring MVC (Model view controller)

MVC solves –

- Annotations
- Cleaner controllers
- Dependency Injection
- Better Scalability