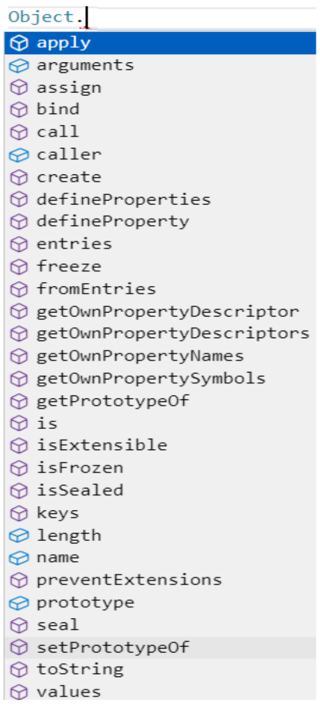
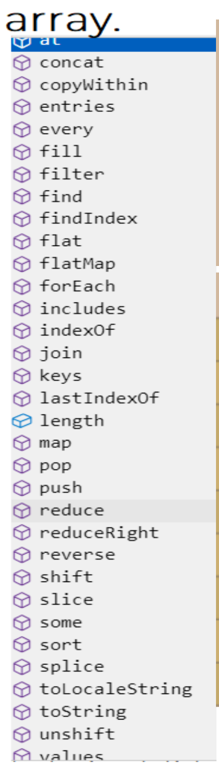
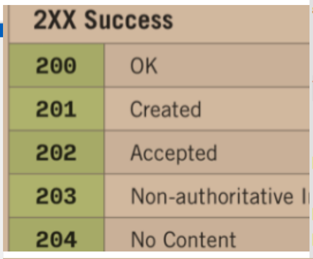


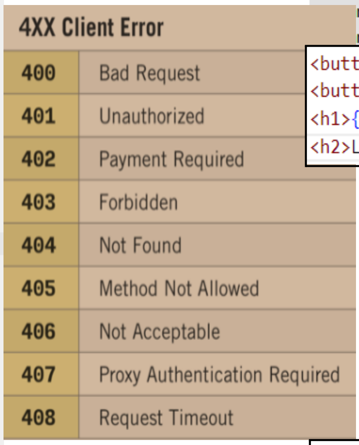
```
Object.
apply
arguments
assign
bind
call
caller
create
defineProperties
defineProperty
entries
freeze
fromEntries
getOwnPropertyDescriptor
getOwnPropertyDescriptors
getOwnPropertyNames
getOwnPropertySymbols
getPrototypeOf
is
isExtensible
isFrozen
isSealed
keys
length
name
preventExtensions
prototype
seal
setPrototypeOf
toString
values
```



```
array.
at
concat
copyWithin
entries
every
fill
filter
find
findIndex
flat
flatMap
forEach
includes
indexOf
join
keys
lastIndexOf
length
map
pop
push
reduce
reduceRight
reverse
shift
slice
some
sort
splice
toLocaleString
toString
unshift
values
```



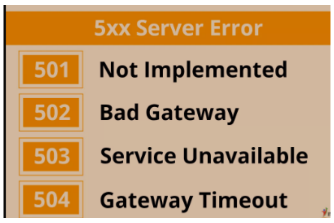
2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative I
204	No Content



4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

```
## Same site: (eTLD+1)
- https://example.co.uk -> eTLD+1 = example.co.uk
- https://abc.xyz.example.co.uk -> eTLD+1 = example.co.uk
They have different origins, so they don't share cookies
- http//:localhost:8080 -> localhost: Same Site. port:8080: Different origins
- http//localhost:8081 -> localhost: Same Site. port8081: Different origins
Hostname defines sites
Port and domains defines origins
Different origins protect against cross fire scripting by not sharing cookies
Different origins, same site:
    mple.to.uk
    mple.co.uk
```

```
<button onClick={onStart}>Start (click again to stop)</button>
<button onClick={onLap} disabled={!isRunning}>Lap</button>
<h1>{counter ? counter : 0}</h1>
<h2>Laps: {laps}</h2>
```



5xx Server Error	
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

```
let timeout: NodeJS.Timeout
export function _2021_T2(){
```

```
const [laps, setLaps] = React.useState([])

React.useEffect(() => {
    return () => {
        clearTimeout(timeout)
        console.log("destructor called")
    }
}, [])
```

```
React.useEffect(() => {
    console.log("counter updated")
    function count(){
        setCounter(counter+1)
    }
    if(counter!=undefined) timeout = setTimeout(count, 1000)
}, [counter])
```

```
const options = {
    method: method,
    credentials: 'same-origin',
    body: (body || method!="GET" || method!="DELETE") ? JSON.stringify(body) : null,
    headers: { "Content-Type": "application/json" , "Accept" : "application/json"}
}
```

```
signal: signal
```

```
return fetch(path, options).then(async rsp => {
    if(!rsp.ok){     rsp.json().then(message => {
            alert(`Error: ${rsp.statusText}. ${message.error}`)
            window.location=`/error?type=${response.statusText}`
            return null
        })
    }
    try {const obj = await rsp.json()
        console.log(`Fetch JSON response -> ${JSON.stringify(obj)}`)
        return obj
    } catch (e) {console.log("Error parsing to json -> " + e); return null}
}).catch(e => {console.log("Request error -> "+e); return null
})
```

```
function onChangeTextArea(e: React.ChangeEvent<HTMLTextAreaElement>){
    const maybeANumber = new Number(e.target.value).valueOf()
    let daNumber = maybeANumber
    if(isNaN(maybeANumber)) daNumber = Number.POSITIVE_INFINITY
```

```
const root = ReactDOM.createRoot(document.getElementById('root'))
```

Forbidden means: "you are not authorized regardless of authentication"

Authorization error means:I see the user you are (have token, but you dont have permission to do this

```
signal.addEventListener("abort", () => {
    console.log("aborted!")
})
```

```
<>URI = {uri}</>
<h2>Is running {counter==undefined ? <>no</> : <>yes</>}</h2>
<h2>Is fetching {isFetching ? <>yes</> : <>no</>}</h2>
<h2>Counter = {counter}. Period ms = {periodMS}</h2>
<h1>Status code: {statusCode}</h1>
<h1>Time taken for response: {timeTaken}</h1>
<h1>{!wasRequestTimedOut ? content.slice(0, content.indexOf("
{error ? <h1>{error}</h1> : <></>}
```

```
setTime(prevTime => prevTime + 1);
```

```
function onButtonClick(){
    console.log("uri=", uri.current)
    async function perFormFetch(){
        setIsFetching(true)
        const data = await doFetch(uri.current)
        setTextArea(data)
        setIsFetching(false)
    }
    perFormFetch()
}
```

```
    console.log('Now aborting');
    controller.abort()
    setController(new AbortController())
}
```

```
function updateUri(e: React.ChangeEvent<HTMLInputElement>)
    if(isFetching) abortFetching()
    uri.current = e.target.value
```

```
const [controller, setController] = React.useState(new AbortController())
const signal = controller.signal
```

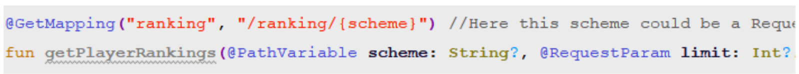
```
tion renderThis(exam: JSX.Element
root.render(exam)
```

```
let keys = 0
return urlsStatus.map(urlStatus => {
    return (
        <div key={keys++} >
            <h2>{urlStatus.url}</h2>
            <button onClick={() => get(urlStatus.url)}>Fetch</button>
            <p>Is fetching? {urlStatus.isFetching ? <>...</> : <></>}</p>
            <p>Contents: {urlStatus.content==undefined ? <>none yet</> : <>{urlStatus.content}</>}</p>
            <br/>
        </div>
    )
})
```

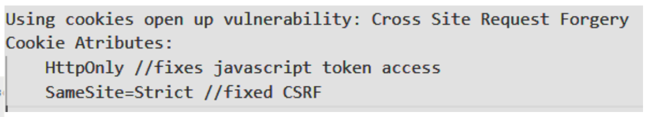
```
setLaps([...laps, `${counter}s at ${new Date().toLocaleTimeString()}`])
```

```
React.createElement(
  type,
  [props],
  [...children]
)
```

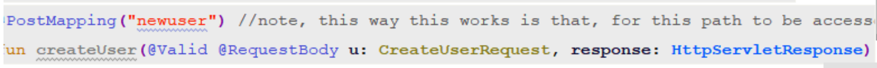
```
<>URI:</><input type="text" placeholder="uri" ref={uri} onChange={(e) => updateUri(e)} ></input>
<button disabled={isFetching} onClick={onButtonClick}>GET</button>
<br/>
<textarea cols={30} rows={10} value={textArea} readOnly>{}</textarea>
```



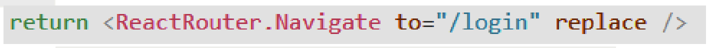
```
@GetMapping("ranking", "/ranking/{scheme}") //Here this scheme could be a Reque
fun getPlayerRankings(@PathVariable scheme: String?, @RequestParam limit: Int?
```



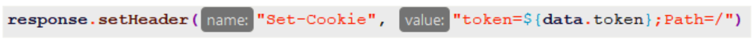
```
Using cookies open up vulnerability: Cross Site Request Forgery
Cookie Atributes:
    HttpOnly //fixes javascript token access
    SameSite=Strict //fixed CSRF
```



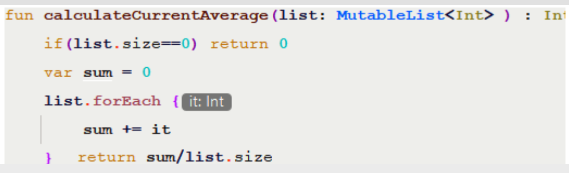
```
PostMapping("newuser") //note, this way this works is that, for this path to be access
un createUser(@Valid @RequestBody u: CreateUserRequest, response: HttpServletResponse)
```



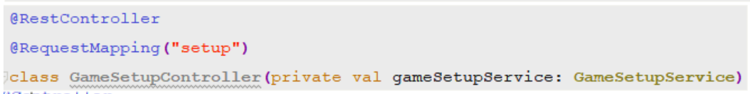
```
return <ReactRouter.Navigate to="/login" replace />
```



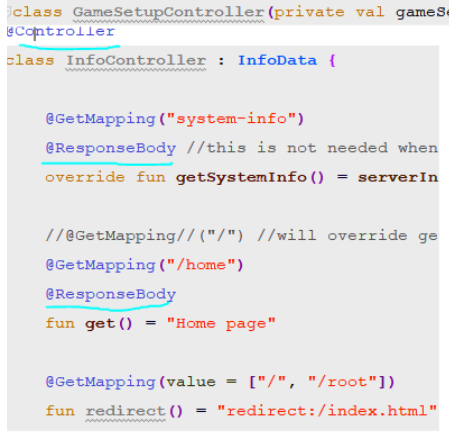
```
response.setHeader( name: "Set-Cookie",  value: "token=${data.token};Path=/")
```



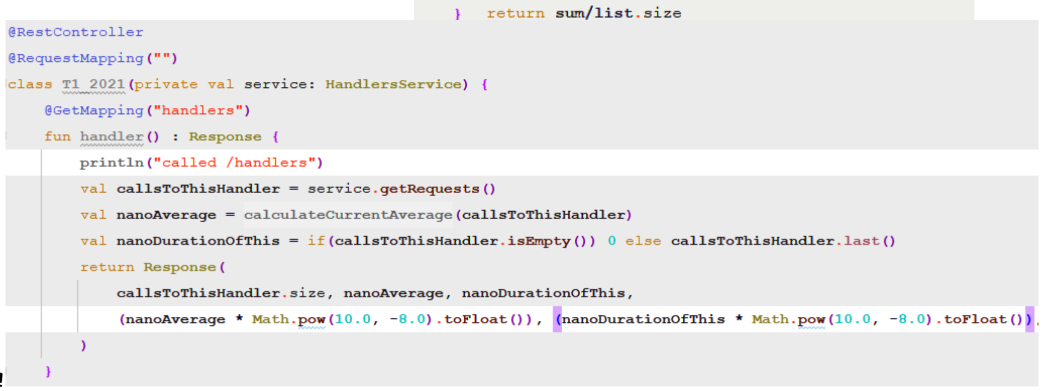
```
fun calculateCurrentAverage(list: MutableList<Int> ) : In
    if(list.size==0) return 0
    var sum = 0
    list.forEach { it: Int
        sum += it
    }  return sum/list.size
```



```
@RestController
@RequestMapping("setup")
class GameSetupController(private val gameSetupService: GameSetupService)
```

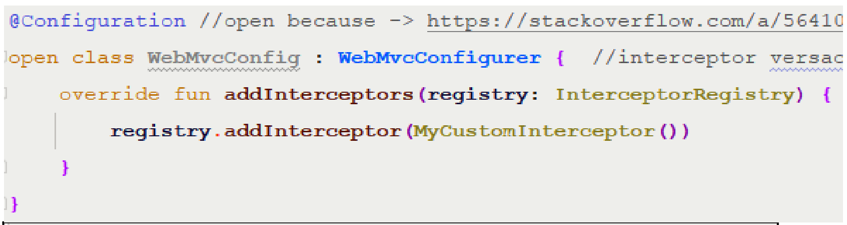


```
@Controller
class InfoController : InfoData {

    @GetMapping("system-info")
    @ResponseBody //this is not needed when
    override fun getSystemInfo() = serverIn

    //@GetMapping//("/") //will override ge
    @GetMapping("/home")
    @ResponseBody
    fun get() = "Home page"

    @GetMapping(value = ["/", "/root"])
    fun redirect() = "redirect:/index.html"
```

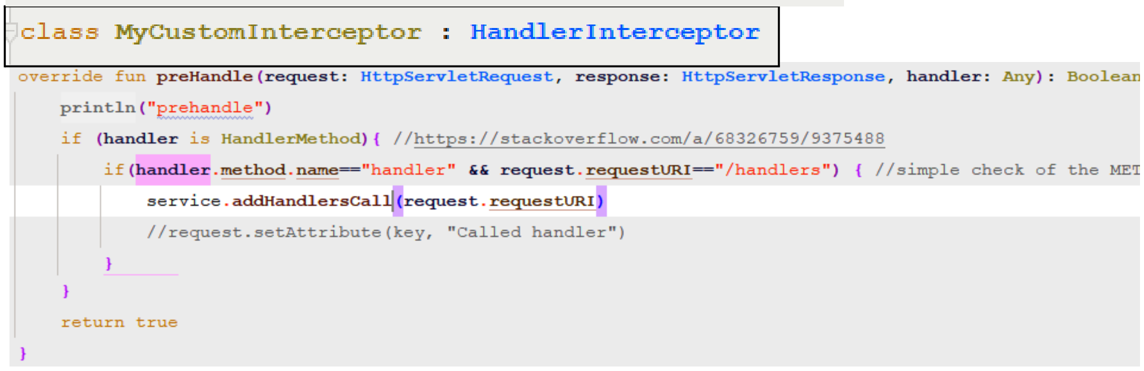


```
@RestController
@RequestMapping("")
class T1_2021(private val service: HandlersService) {
    @GetMapping("handlers")
    fun handler() : Response {
        println("called /handlers")
        val callsToThisHandler = service.getRequests()
        val nanoAverage = calculateCurrentAverage(callsToThisHandler)
        val nanoDurationOfThis = if(callsToThisHandler.isEmpty()) 0 else callsToThisHandler.last()
        return Response(
            callsToThisHandler.size, nanoAverage, nanoDurationOfThis,
            (nanoAverage * Math.pow(10.0, -8.0).toFloat()), (nanoDurationOfThis * Math.pow(10.0, -8.0).toFloat()),
        )
    }
```

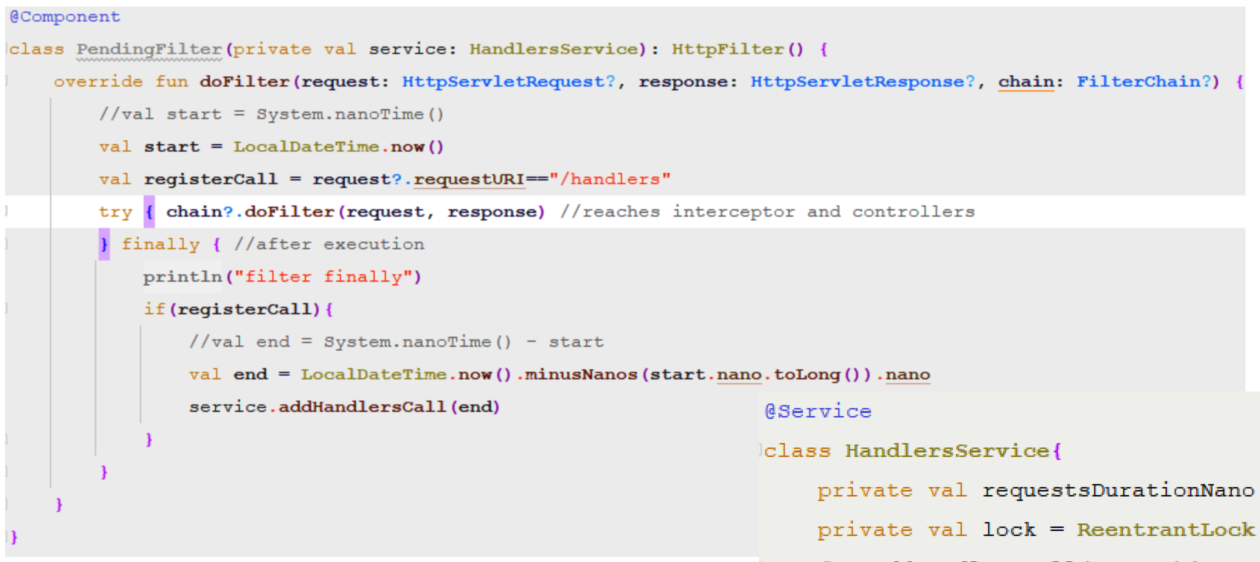
EX5 (preferivelmente usar doFilter!



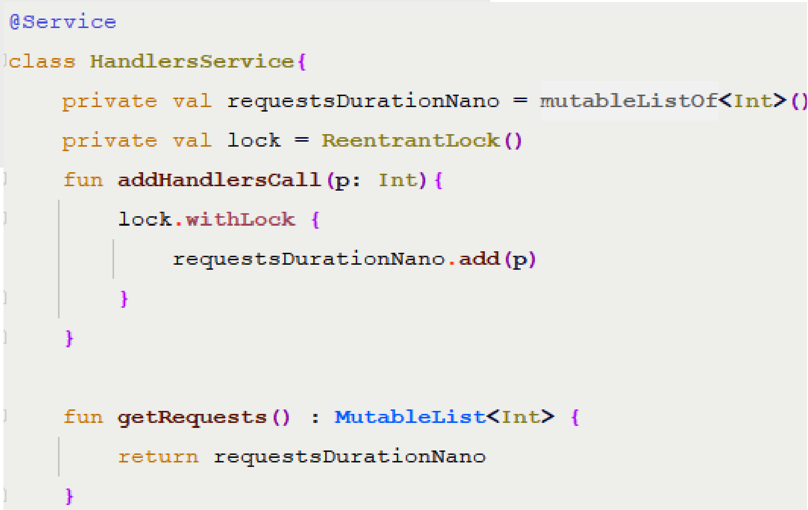
```
@Configuration //open because -> https://stackoverflow.com/a/56410
open class WebMvcConfig : WebMvcConfigurer {  //interceptor versac
    override fun addInterceptors(registry: InterceptorRegistry) {
        registry.addInterceptor(MyCustomInterceptor())
    }
}
```



```
class MyCustomInterceptor : HandlerInterceptor
override fun preHandle(request: HttpServletRequest, response: HttpServletResponse, handler: Any): Boolean
    println("prehandle")
    if (handler is HandlerMethod){ //https://stackoverflow.com/a/68326759/9375488
        if(handler.method.name=="handler" && request.requestURI=="/handlers") { //simple check of the MET
            service.addHandlersCall(request.requestURI)
            //request.setAttribute(key, "Called handler")
        }
    }
    return true
}
```



```
@Component
class PendingFilter(private val service: HandlersService): HttpFilter() {
    override fun doFilter(request: HttpServletRequest?, response: HttpServletResponse?, chain: FilterChain?) {
        //val start = System.nanoTime()
        val start = LocalDateTime.now()
        val registerCall = request?.requestURI=="/handlers"
        try { chain?.doFilter(request, response) //reaches interceptor and controllers
        } finally { //after execution
            println("filter finally")
            if(registerCall){
                //val end = System.nanoTime() - start
                val end = LocalDateTime.now().minusNanos(start.nano.toLong()).nano
                service.addHandlersCall(end)
            }
        }
    }
}
```



```
@Service
class HandlersService{
    private val requestsDurationNano = mutableListOf<Int>()
    private val lock = ReentrantLock()
    fun addHandlersCall(p: Int){
        lock.withLock {
            requestsDurationNano.add(p)
        }
    }

    fun getRequests() : MutableList<Int> {
        return requestsDurationNano
    }
```