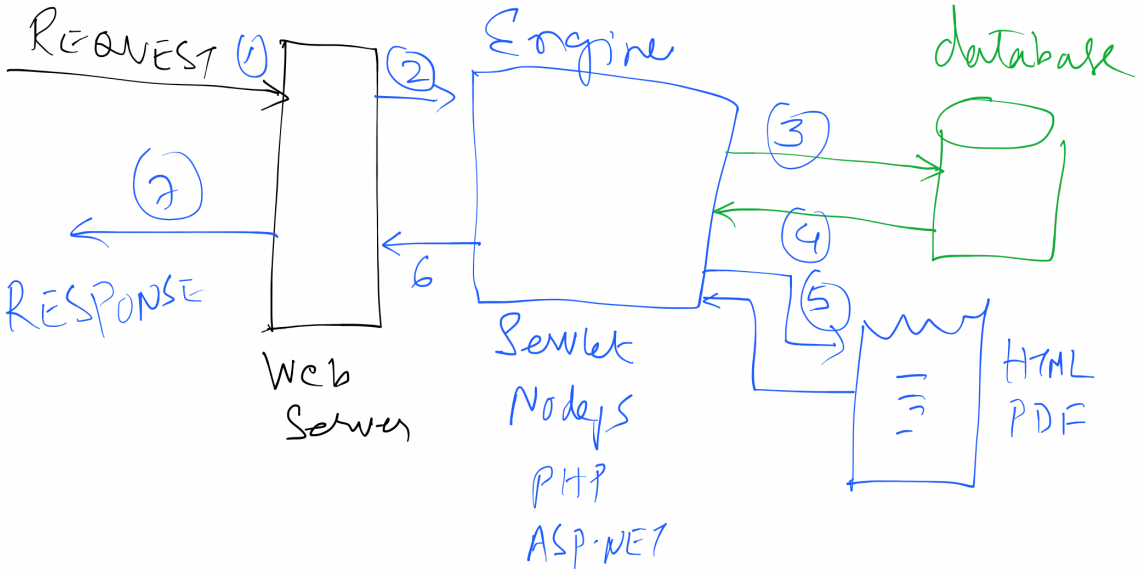
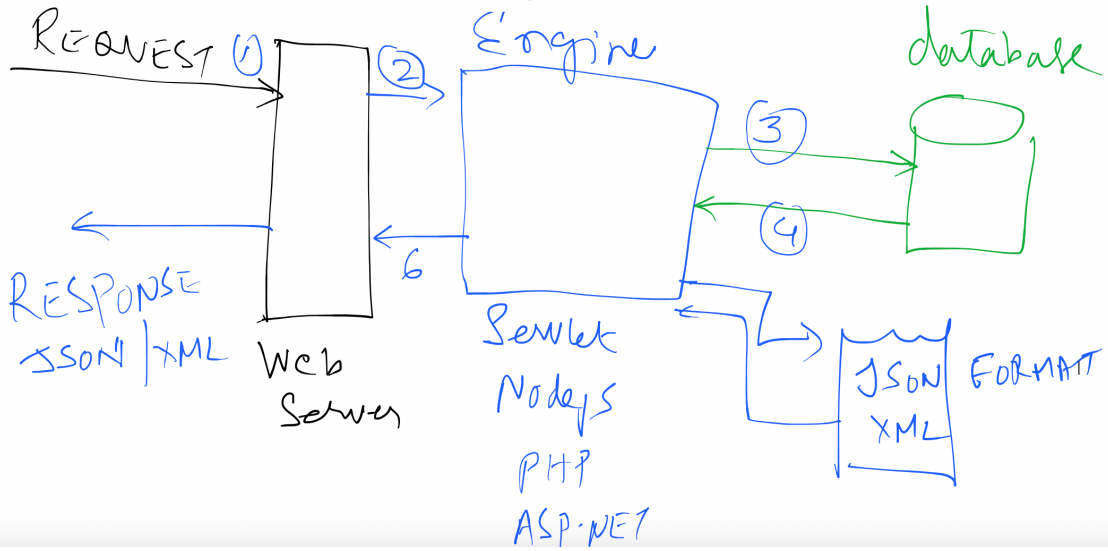
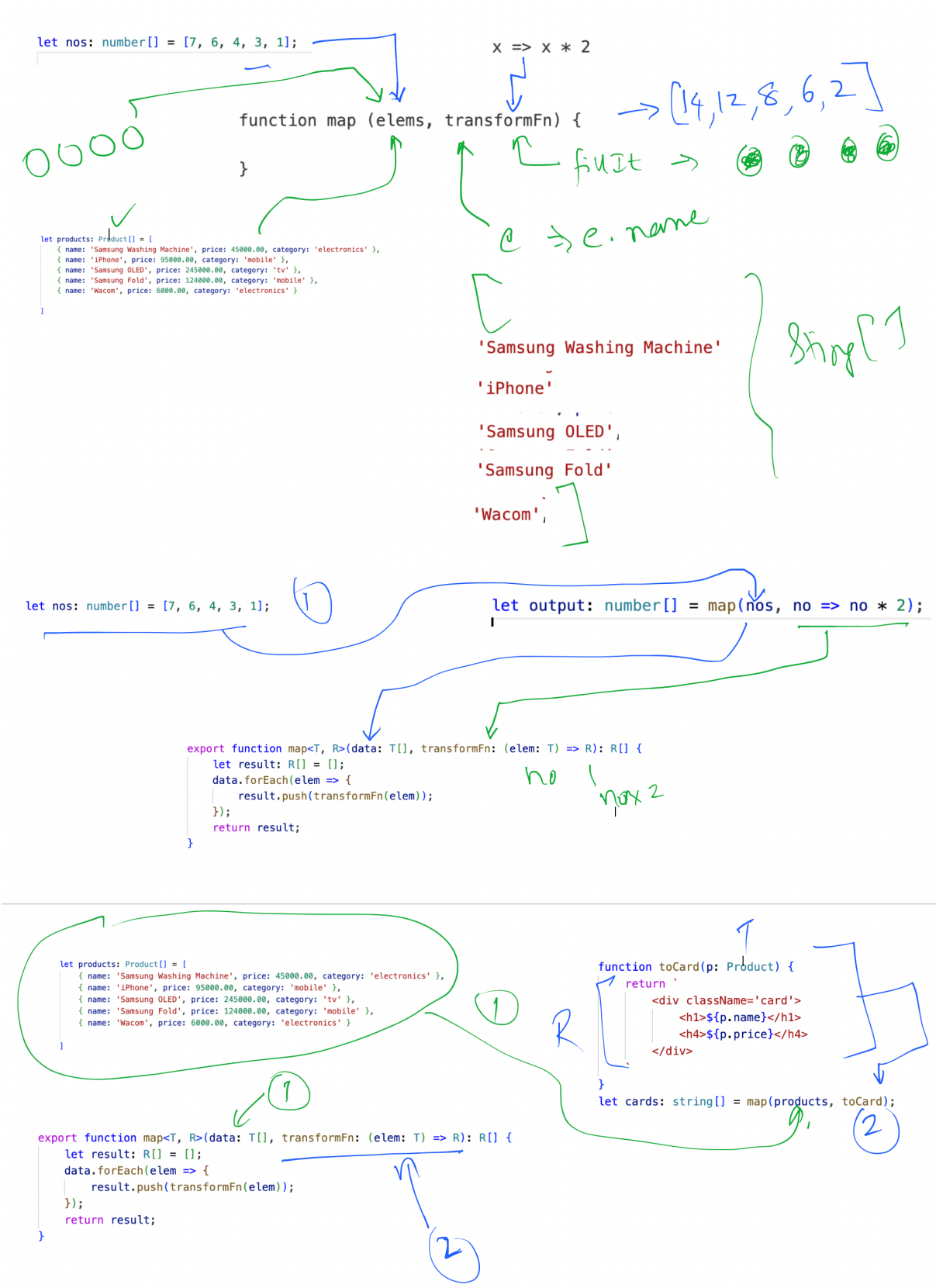
Server Side Rendering



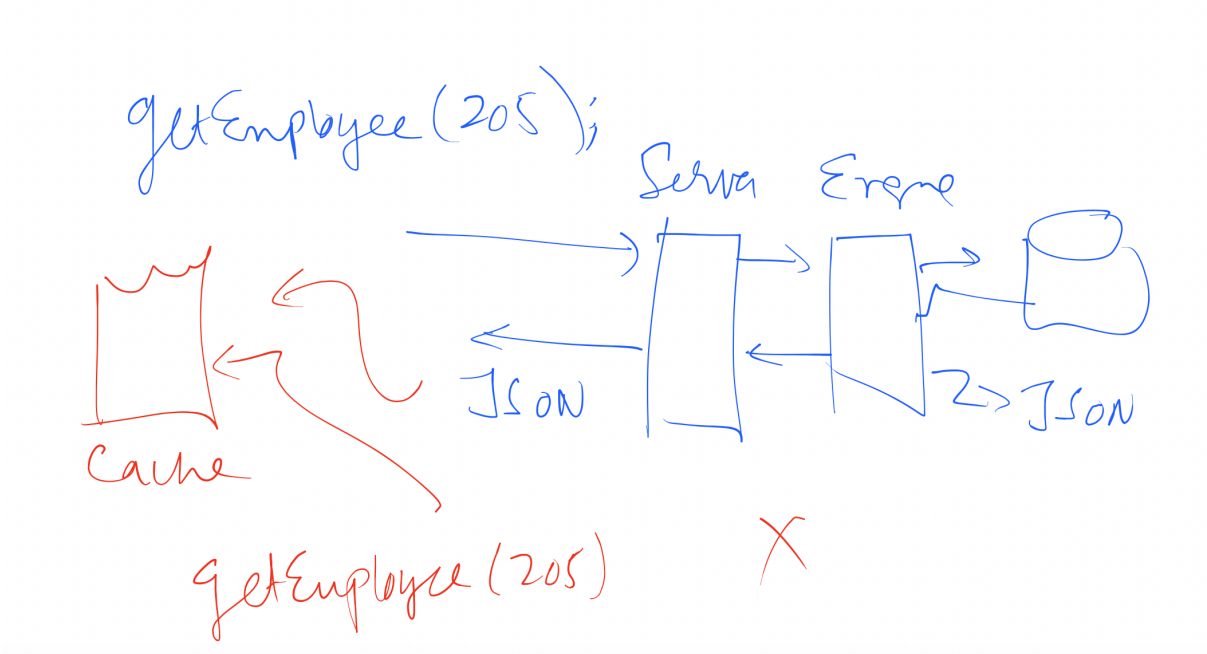
Client Side Rendering




let nos: number[] = [7, 6, 4, 3, 1];
function map (elems, transformFn) {
}
x => x * 2
[14,12,8,6,2]
fuit
e => e.name
'Samsung Washing Machine'
'iPhone'
'Samsung OLED'
'Samsung Fold'
'Wacom'
string[]
let products: Product[] = [
{ name: 'Samsung Washing Machine', price: 45000.00, category: 'electronics' },
{ name: 'iPhone', price: 95000.00, category: 'mobile' },
{ name: 'Samsung OLED', price: 245000.00, category: 'tv' },
{ name: 'Samsung Fold', price: 124000.00, category: 'mobile' },
{ name: 'Wacom', price: 6000.00, category: 'electronics' }
]
let nos: number[] = [7, 6, 4, 3, 1];
let output: number[] = map(nos, no => no * 2);
export function map<T, R>(data: T[], transformFn: (elem: T) => R): R[] {
let result: R[] = [];
data.forEach(elem => {
result.push(transformFn(elem));
});
return result;
}
no
no * 2
function toCard(p: Product) {
return `

<h1>${p.name}</h1>
<h4>${p.price}</h4>

`
}
let cards: string[] = map(products, toCard);

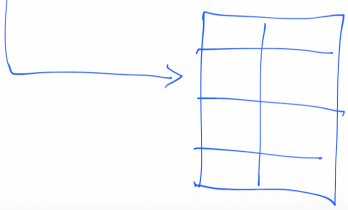



```ts
export default function Memo(target: any, methodName: string, descriptor: PropertyDescriptor) {
    let fn:Function = descriptor.value; // actual method code --> fibanocci

    let cache:Map<String, any> = new Map();

    descriptor.value = function(...args:number[]) {
        let key = args.join("-");
        if(!cache.has(key)) {
            const result = fn.apply(this, args);
            cache.set(key, result);
        }
        return cache.get(key);
    }
}
```

```ts
@Memo
fibanocci(no: number): number {
    return (no == 0 || no == 1) ? no : this.fibanocci(no - 1) + this.fibanocci(no - 2);
}
```



```ts
export default function Memo(target: any, methodName: string, descriptor: PropertyDescriptor) {
    let fn:Function = descriptor.value; // actual method code --> fibanocci

    let cache:Map<String, any> = new Map();

    descriptor.value = function(...args:number[]) {
        let key = args.join("-");
        if(!cache.has(key)) {
            const result = fn.apply(this, args);
            cache.set(key, result);
        }
        return cache.get(key);
    }
}
```

SOA of PROXY

```ts
@Memo
fibanocci(no: number): number {
    return (no == 0 || no == 1) ? no : this.fibanocci(n
}
```