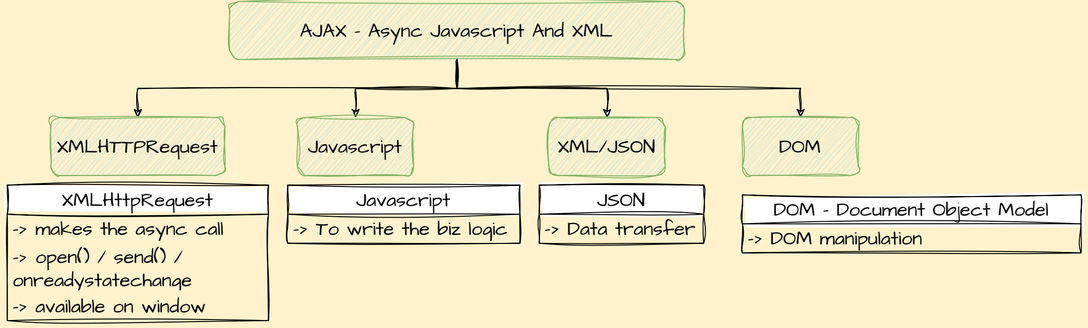
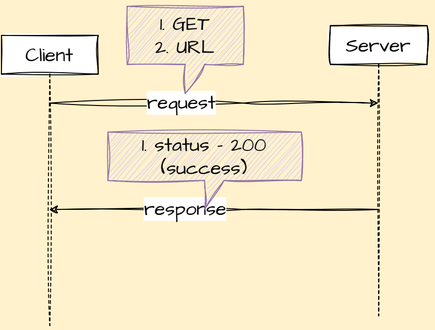
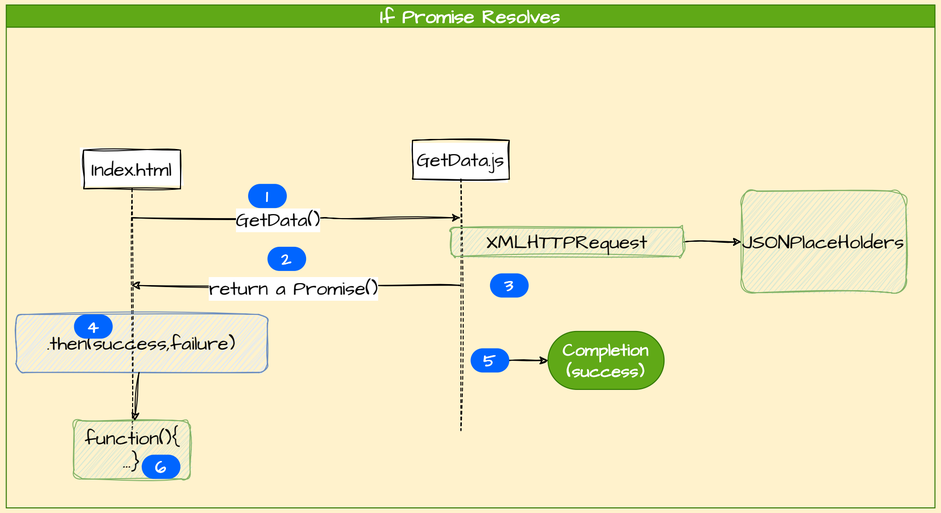
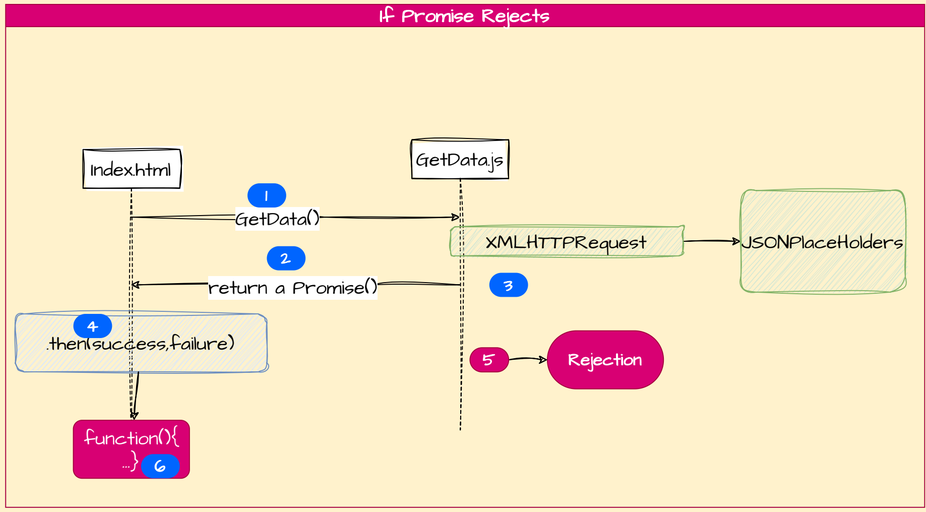
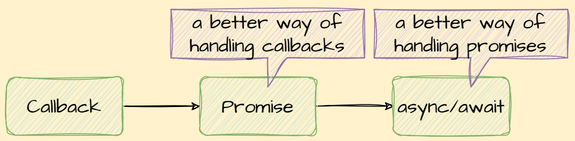

AJAX - Async Javascript And XML
XMLHTTPRequest
Javascript
XML/JSON
DOM
XMLHttpRequest
-> makes the async call
-> open() / send() / onreadystatechange
-> available on window
Javascript
-> To write the biz logic
JSON
-> Data transfer
DOM - Document Object Model
-> DOM manipulation

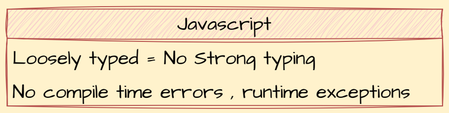

Client
1. GET
2. URL
Server
request
1. status - 200
(success)
response

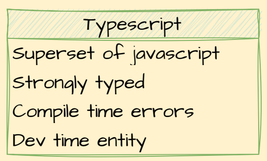

If Promise Resolves
Index.html
GetData.js
1
GetData()
XMLHTTPRequest
JSONPlaceHolders
2
return a Promise()
3
4
.then(success,failure)
5
Completion
(success)
function(){
...}
6

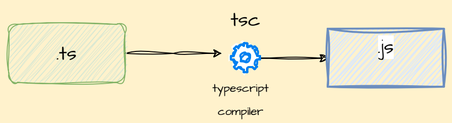

If Promise Rejects
Index.html
GetData.js
1
GetData()
XMLHTTPRequest
JSONPlaceHolders
2
return a Promise()
3
4
.then(success,failure)
5
Rejection
function(){
...}
6


You
assigns some job + call me back
Manager


a better way of handling callbacks
a better way of handling promises
Callback
Promise
async/await


Javascript
Loosely typed = No Strong typing
No compile time errors , runtime exceptions


Typescript
Superset of javascript
Strongly typed
Compile time errors
Dev time entity


.ts
tsc
typescript
compiler
.js

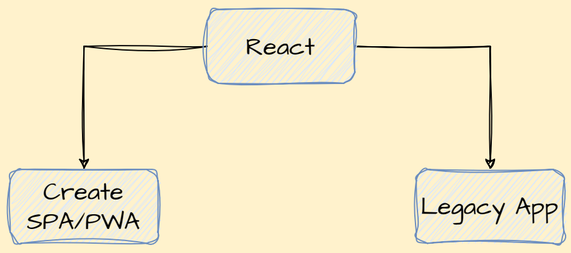
React

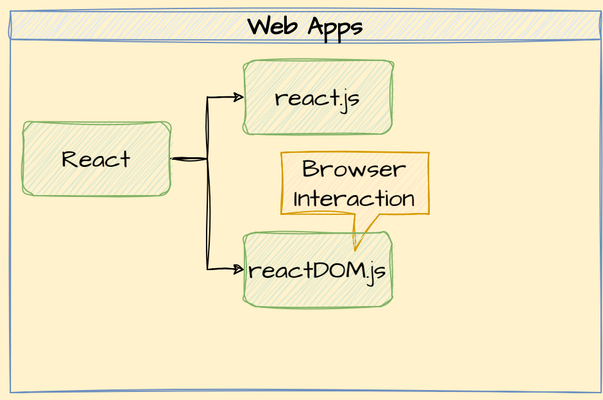
-> A Javascript Library

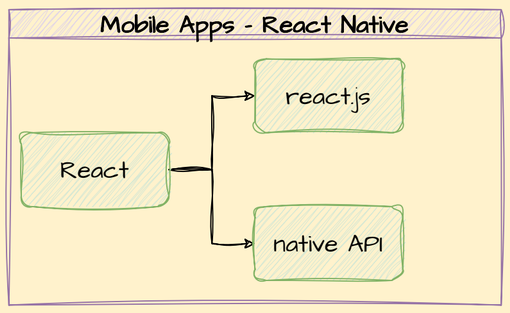
-> Fast Rendering UI

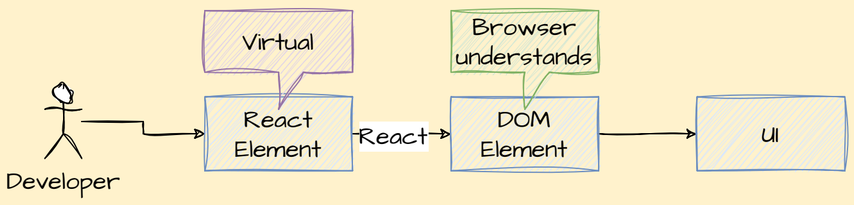
-> Declarative

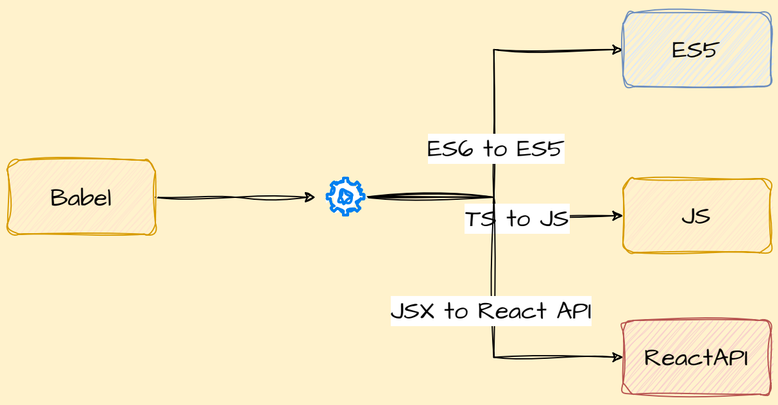
-> Components











Component - Collection of Elements

-> Resuable (UI + Biz Logic)

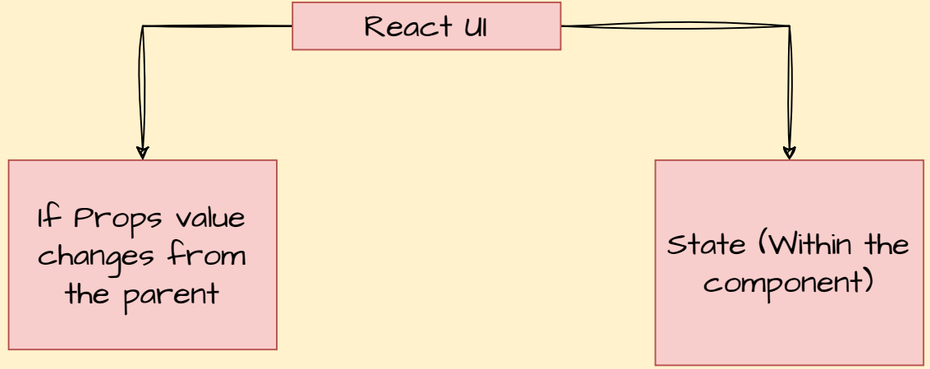
-> Modular

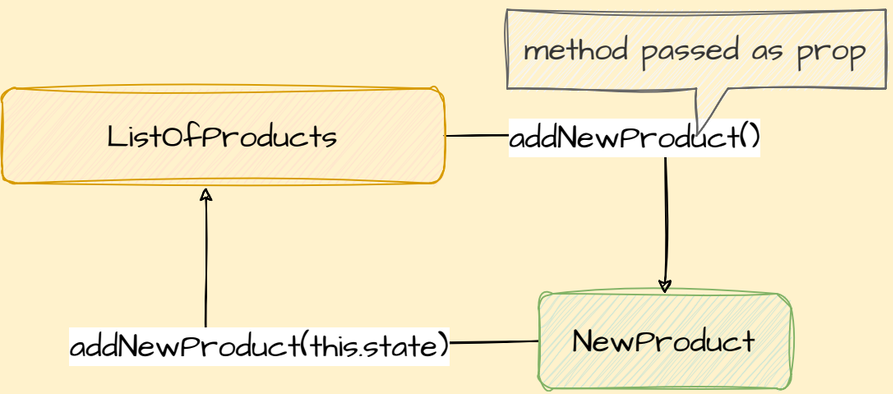
-> Maintenable

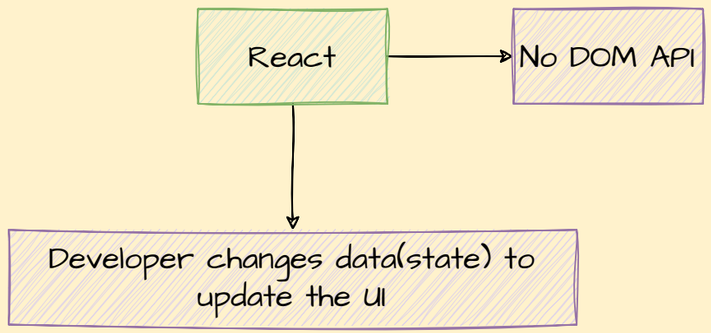
-> Testable

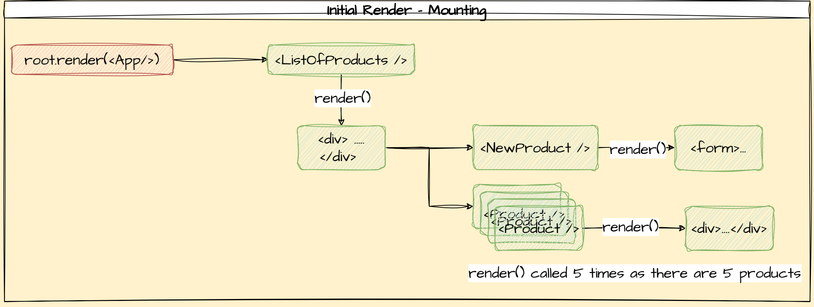
-> Configurable

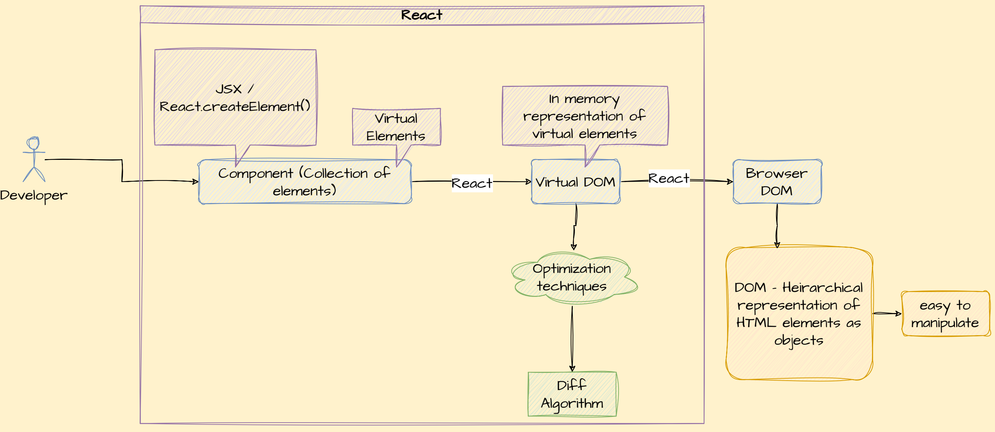
-> Composable

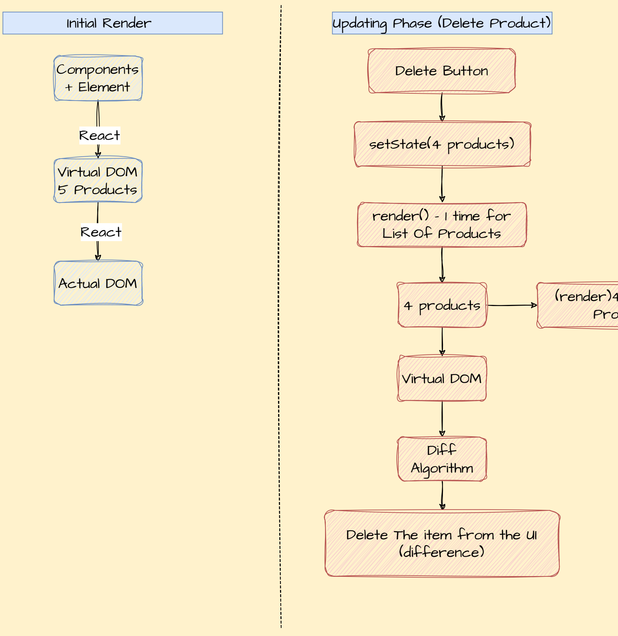

React UI
If Props value changes from the parent
State (Within the component)
State
-> private for a instance component
-> when state changes UI changes
-> setState() [class based component]

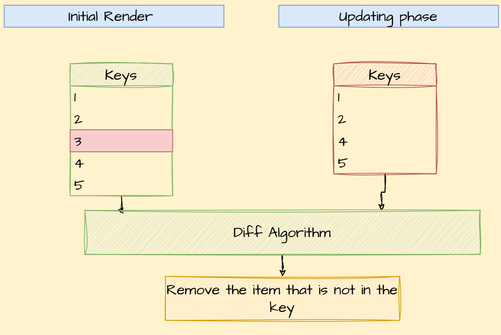

method passed as prop
ListOfProducts
addNewProduct()
addNewProduct(this.state)
NewProduct

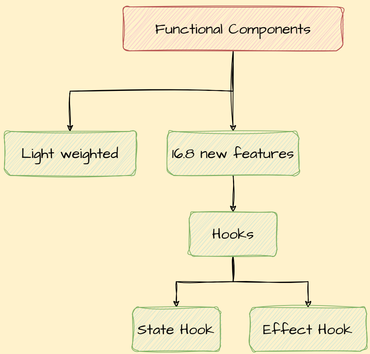

React
No DOM API
Developer changes data(state) to update the UI

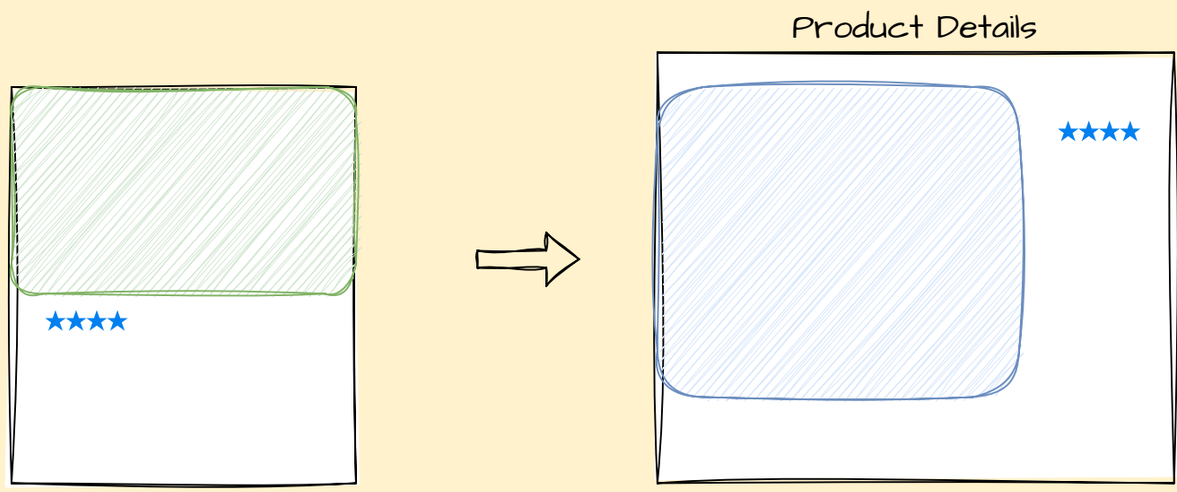

Initial Render - Mounting
root.render(<App/>)
<ListOfProducts />
render()
 .....
<NewProduct />
render()
<form>...
<Product />
<Product />
<Product />
render()
...
render() called 5 times as there are 5 products


React
JSX / React.createElement()
Virtual Elements
In memory representation of virtual elements
Developer
Component (Collection of elements)
React
Virtual DOM
React
Browser DOM
Optimization techniques
DOM - Heirarchical representation of HTML elements as objects
easy to manipulate
Diff Algorithm

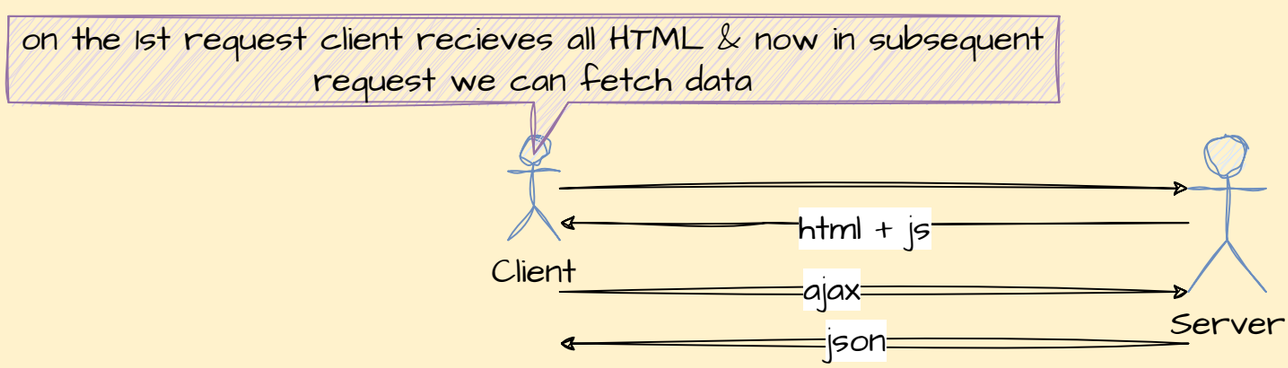

Initial Render
Components + Element
React
Virtual DOM 5 Products
React
Actual DOM
Updating Phase (Delete Product)
Delete Button
setState(4 products)
render() - 1 time for List Of Products
4 products
(render)4 times for Product
Virtual DOM
Diff Algorithm
Delete The item from the UI (difference)

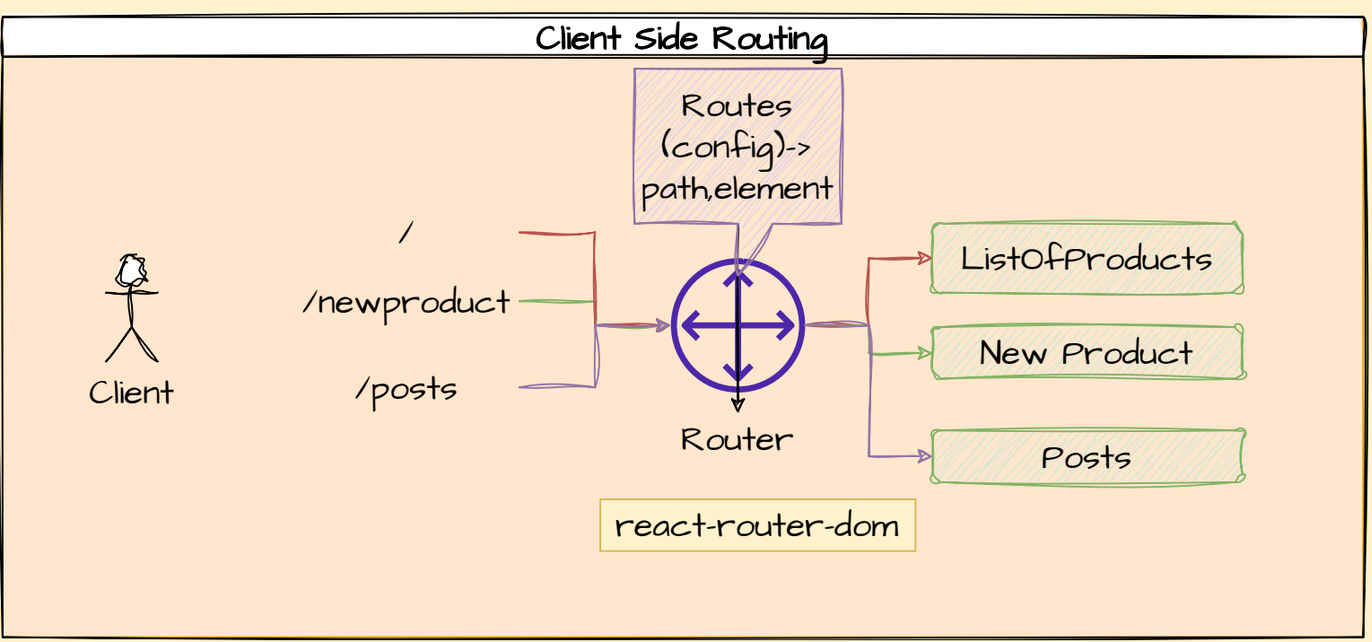

Initial Render
Updating phase
Keys
1
2
3
4
5
Keys
1
2
4
5
Diff Algorithm
Remove the item that is not in the key


Functional Components
Light weighted
16.8 new features
Hooks
State Hook
Effect Hook



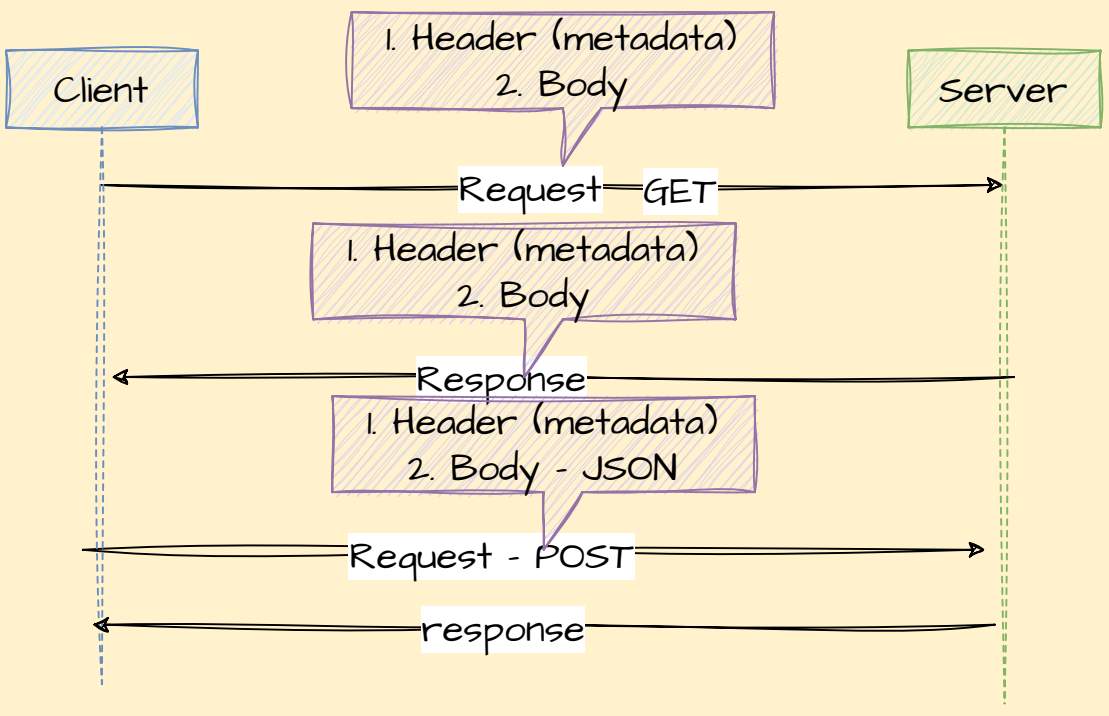
Single Page Application

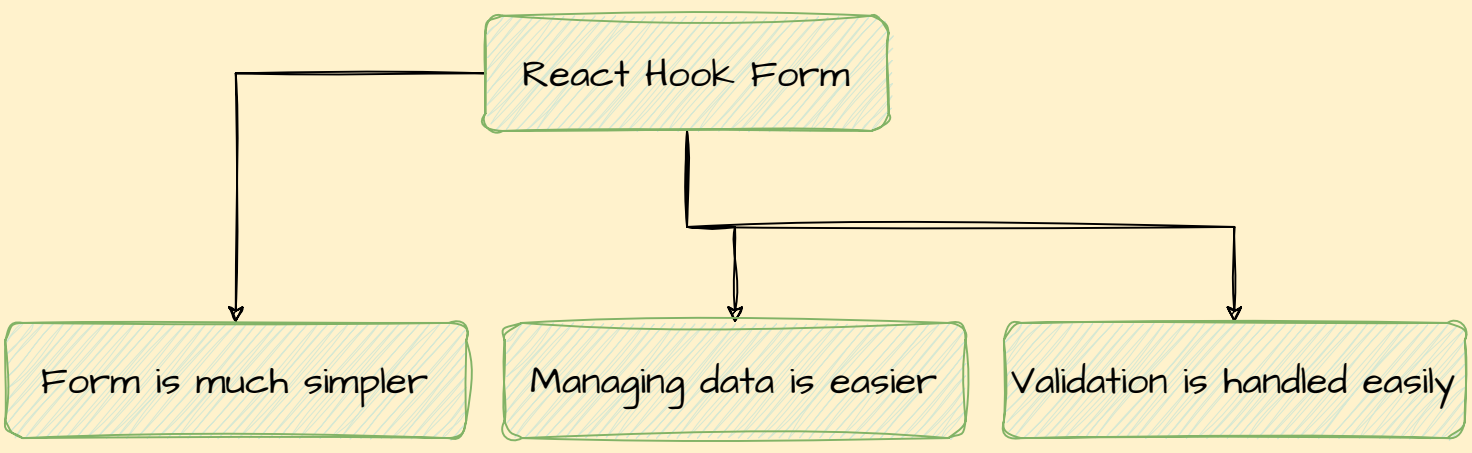




HTTP → Hypertext Transfer Protocol

HTML → Hypertext Markup Language





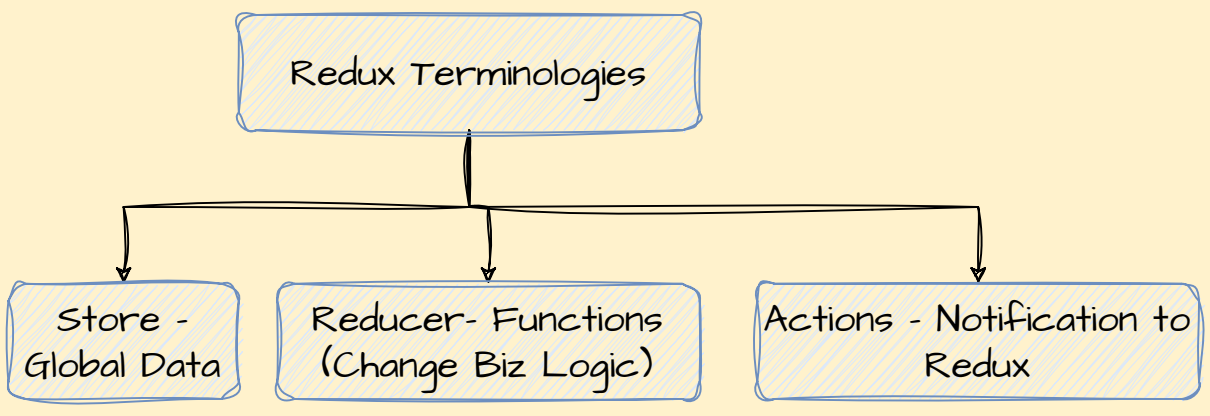
The Problems

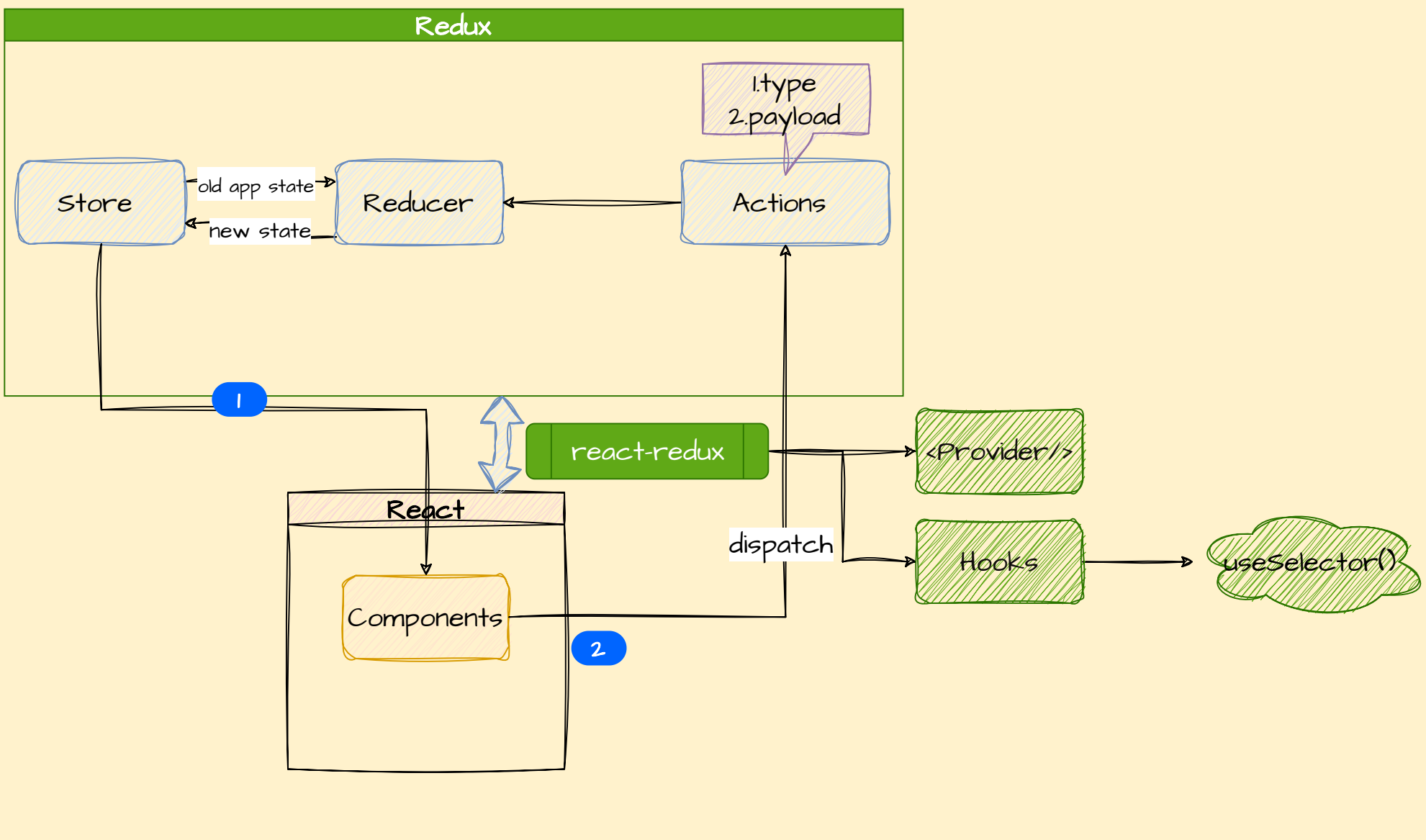
1. Can't share data between Product & Product Details (Posts & Post Details)

2. Delete a product

3. No global data

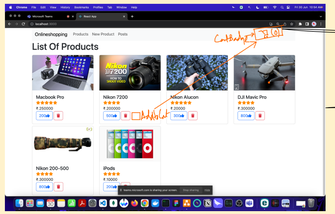
→ Redux



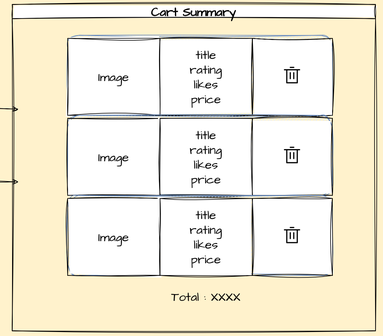


CartBadge

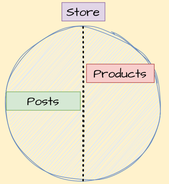
CartItems

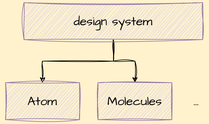


Cart Summary



Action creators → Functions that return an action

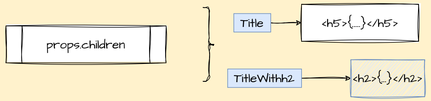




Atomic Principle / Atomic Pattern

Atomic design is a methodology composed of five distinct stages working together to create interface design systems in a more deliberate and hierarchical manner. The five stages of atomic design are:

1. Atoms
2. Molecules
3. Organisms
4. Templates
5. Pages
6.



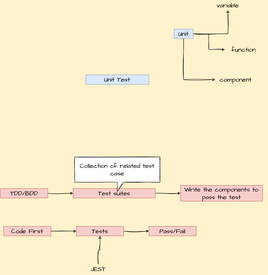
ProductDetails

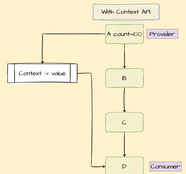
```
import { FC } from "react";


export const ProductDetails: FC = () => {
    return <>
        <Title>
    <h2>{theProduct.title}</h2>
        </Title>
        </>;
    };
```

Title

```
import { FC } from "react";


type TitlePr
  title: st
};
export const Title: FC<TitleProps> = (props: TitleProps) => {
    return <>{props.children}</>;
};
```

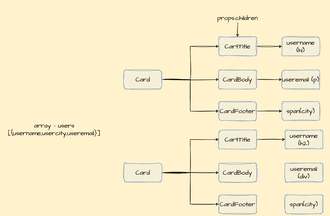
Intrinsic Property

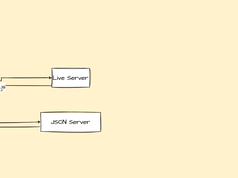

Unit Test
Collection of related test case
TDD/BDD
Test suites
Write the components to pass the test
Code First
Tests
Pass/Fail
JEST
Unit
variable
Function
component

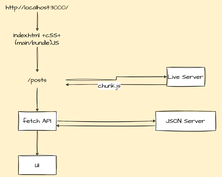

With Context API
A count=100
Provider
Context -> value
B
C
D
Consumer

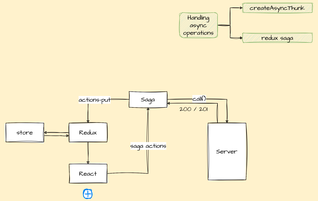

Without Context API
A count=100
B
C
D

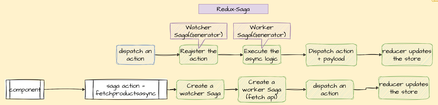

Context API (4.0)
way to share props from parent to any of its descendent

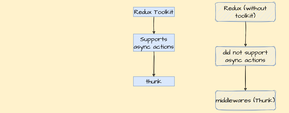

array - users
[{username,userrcity,useremail}]
Card
CardTitle
CardBody
CardFooter
props.children
username (u1)
useremail (u1)
spanCity)
username (u2)
useremail (u2)
spanCity)

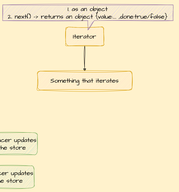

ListOfMyProducts
Posts
NewProduct
100 components
Larger the file larger will be the time to load it from server
bundle.js
chunk.js (4 For posts)
Code Splitting/Lazy Loading

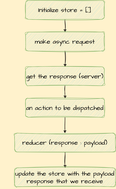

http://localhost:3000/
index.html +CSS+ (main/bundle).JS
/posts
Live Server
chunk.js
fetch API
JSON Server
UI


Handling async operations
createAsyncThunk
redux saga
store
Redux
React
actions-put
Saga
call
100 / 201
saga actions
Server

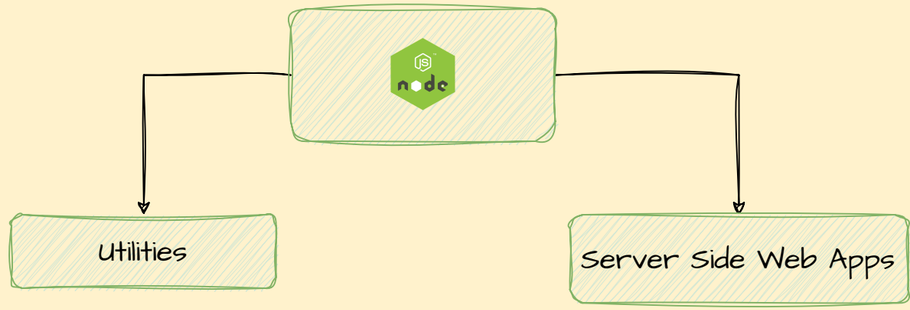

Redux-Saga
Watcher Saga(Generator)
Worker Saga(Generator)
dispatch an action
Register the action
Execute the async logic
Dispatch action + payload
reducer updates the store
component
saga action = fetchProductAsync
Create a watcher Saga
Create a worker Saga (fetch api)
dispatch an action
reducer updates the store

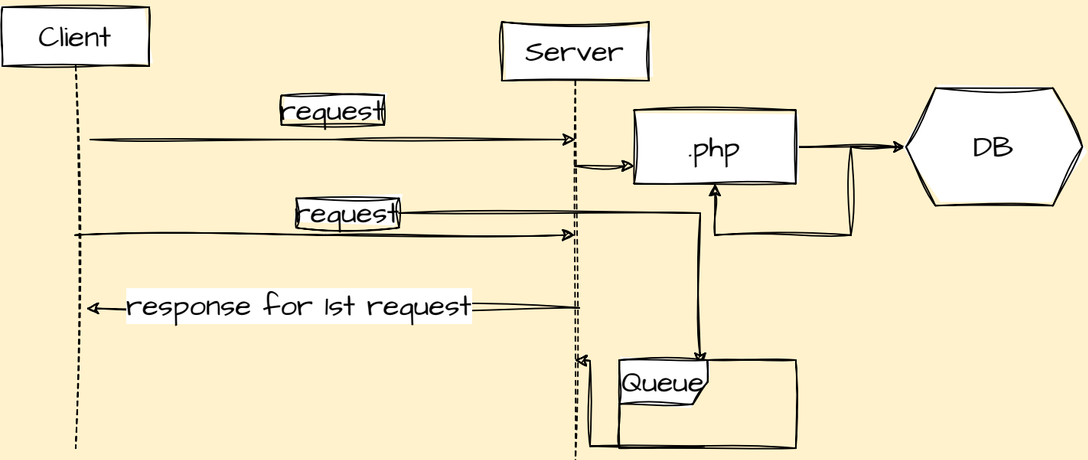

Redux Toolkit
Supports async actions
thunk
Redux (without toolkit)
did not support async actions
middlewares (thunk)

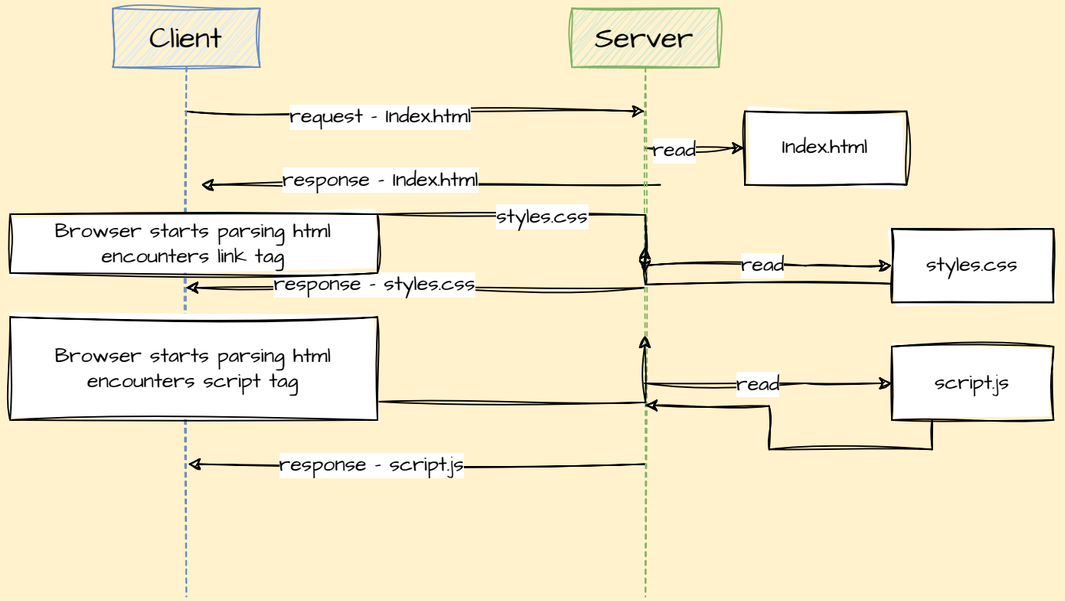

1. as an object
2. next() -> returns an object {value:... done:true/false}
Iterator
Something that iterates

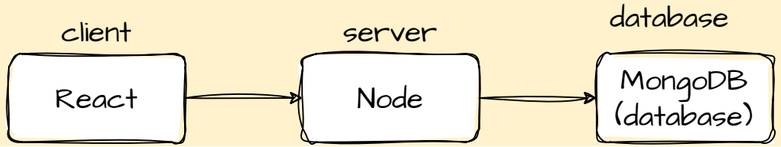

initialize store = []
make async request
get the response (server)
an action to be dispatched
reducer (response - payload)
update the store with the payload (response that we receive

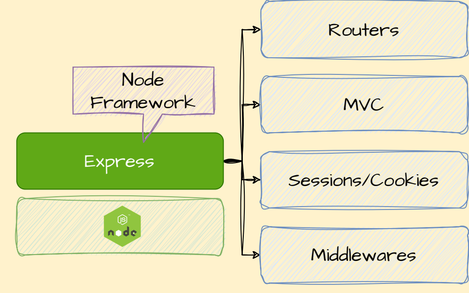
A Javascript Runtime
Built on top of Chrome's V8 engine

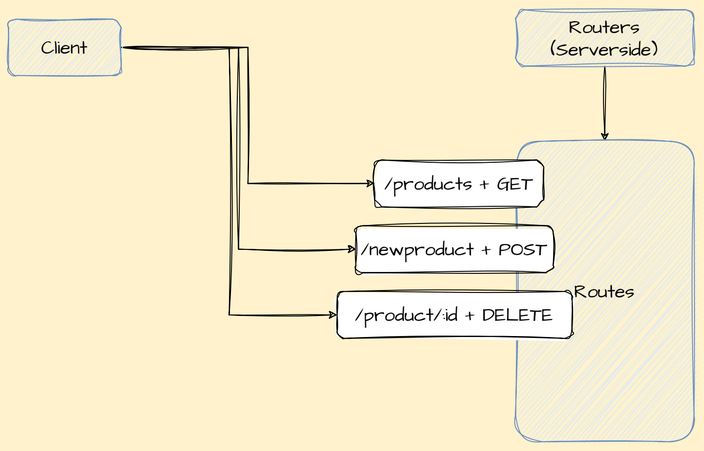


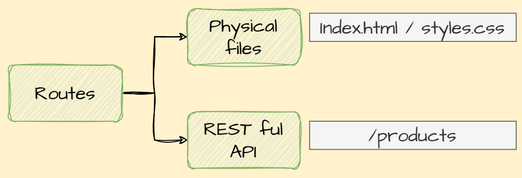


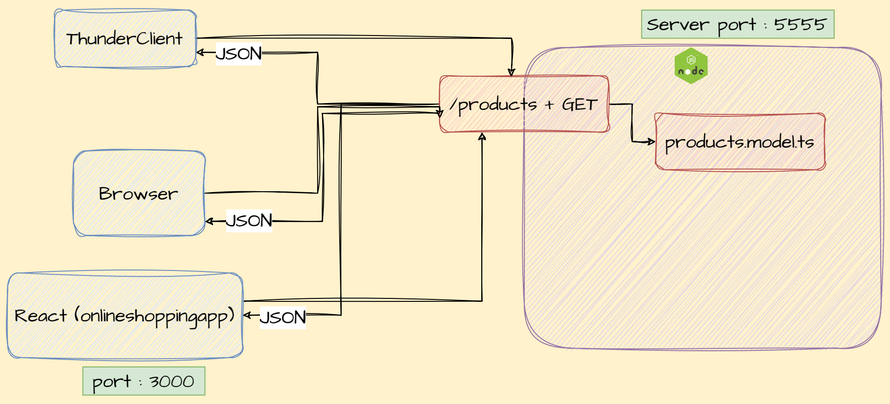


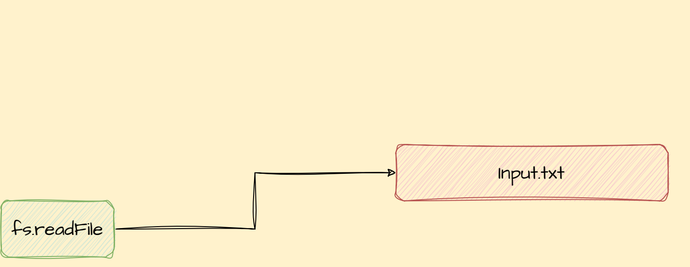


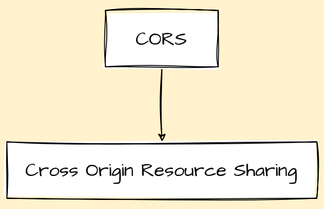

Node Framework
Express
Routers
MVC
Sessions/Cookies
Middlewares

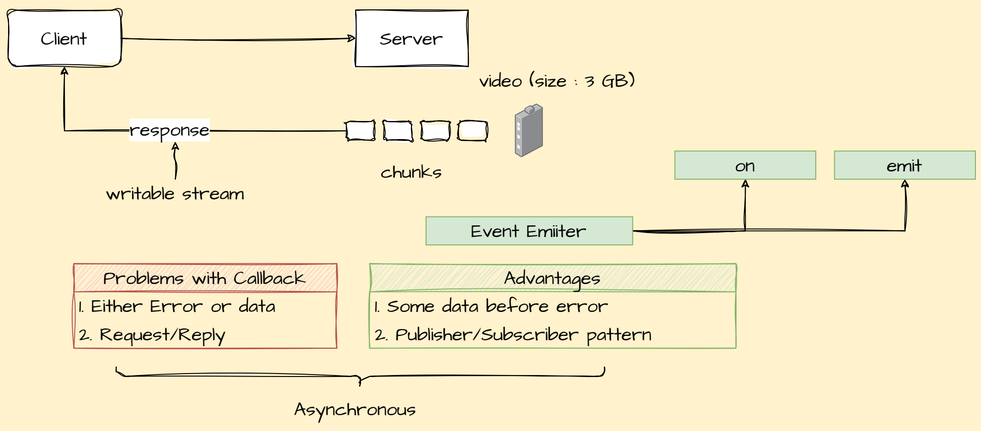

Client
Routers (Serverside)
/products + GET
/newproduct + POST
/product/:id + DELETE
Routes

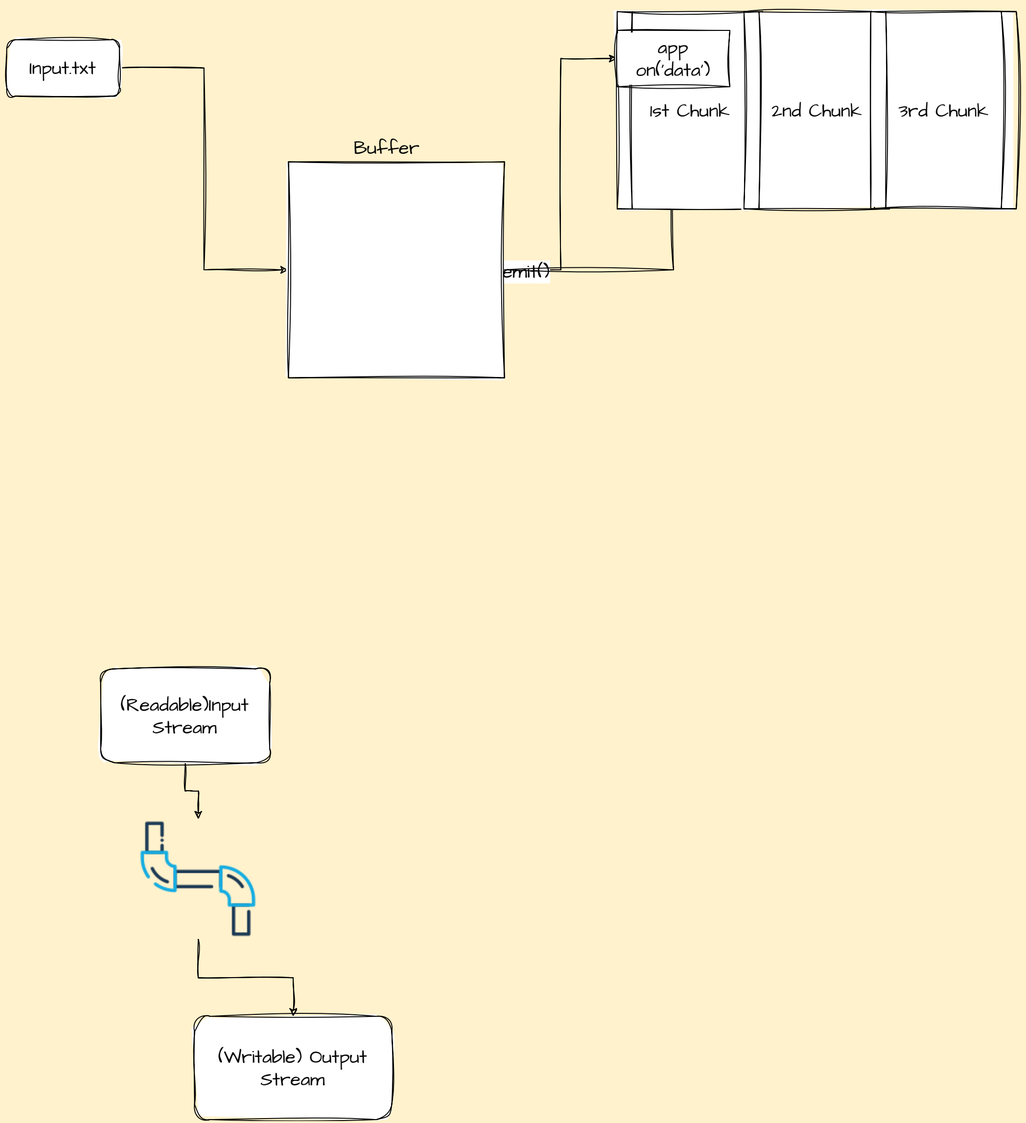

Routes
Physical files
Index.html / styles.css
REST ful API
/products


ThunderClient
JSON
Server port : 5555
/products + GET
products.model.ts
Browser
JSON
React (onlineshoppingapp)
JSON
port : 3000

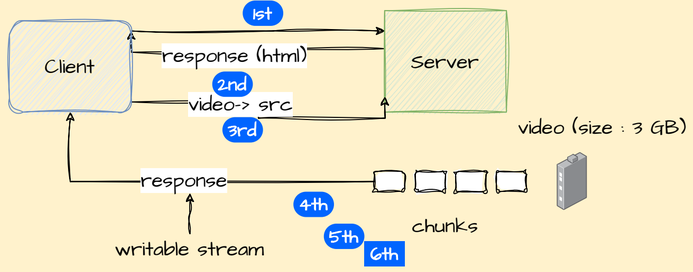

fs.readFile
Input.txt


CORS
Cross Origin Resource Sharing

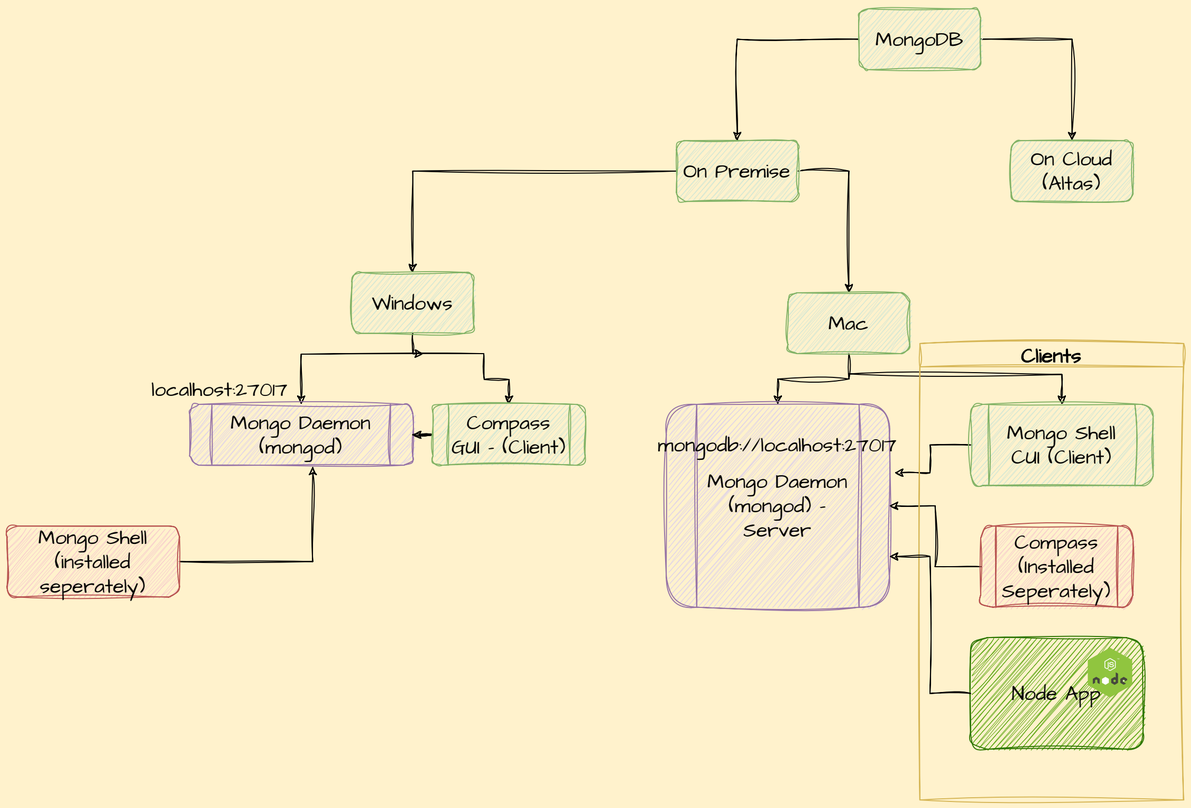

Client
Server
video (size : 3 GB)
response
chunks
writable stream
on
emit
Event Emiiter
Problems with Callback
1. Either Error or data
2. Request/Reply
Advantages
1. Some data before error
2. Publisher/Subscriber pattern
Asynchronous

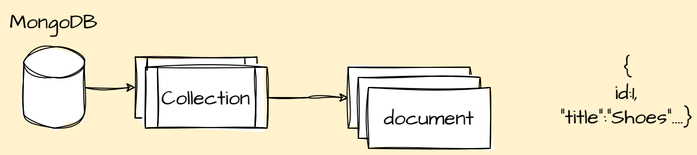

Input.txt
Buffer
app
on('data')
1st Chunk
2nd Chunk
3rd Chunk
emit()

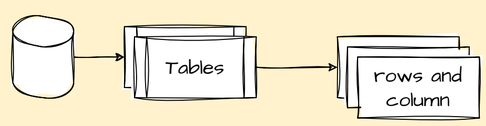
(Readable)Input Stream
(Writable) Output Stream

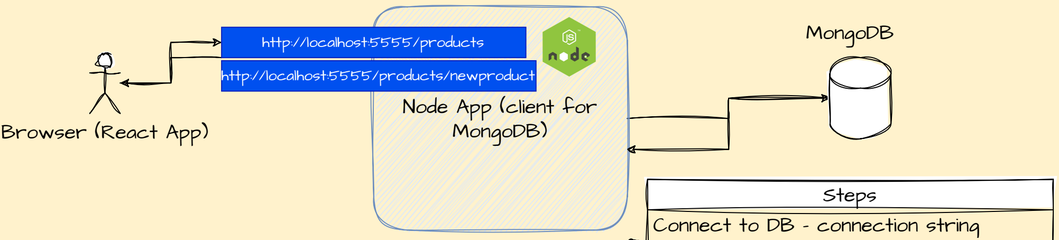


MongoDB
A No SQL Database
Stores data in the form of a document (json)
Faster



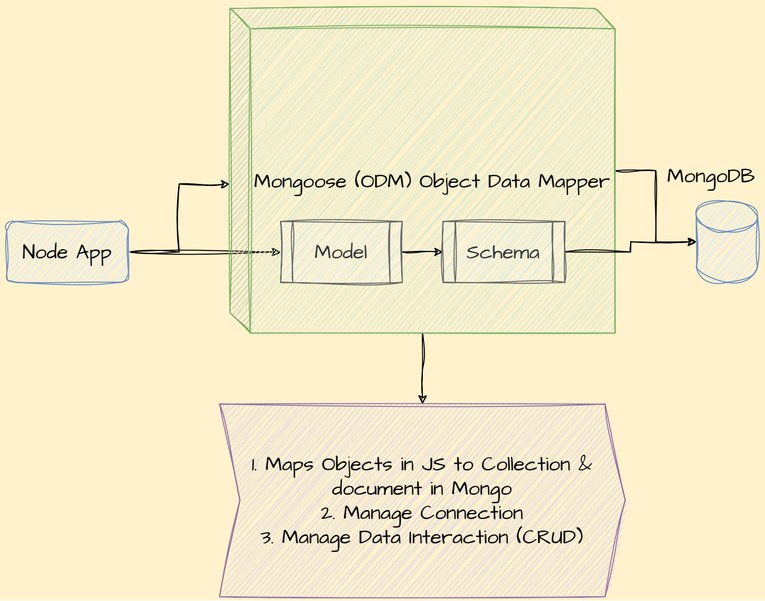


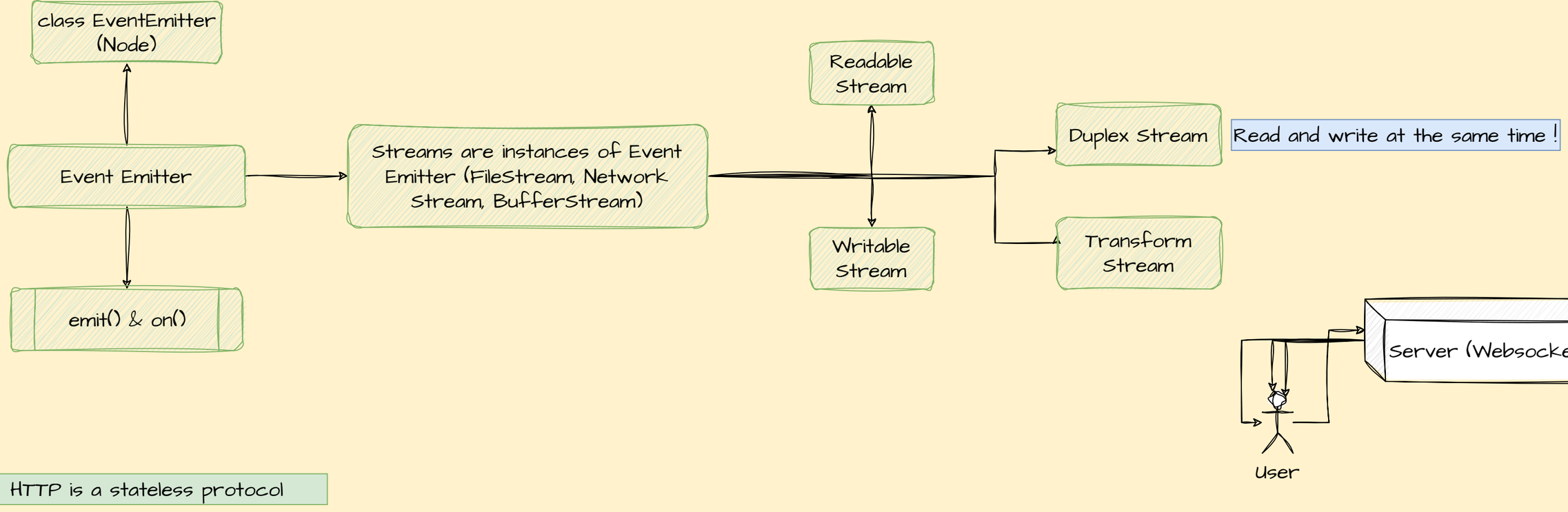


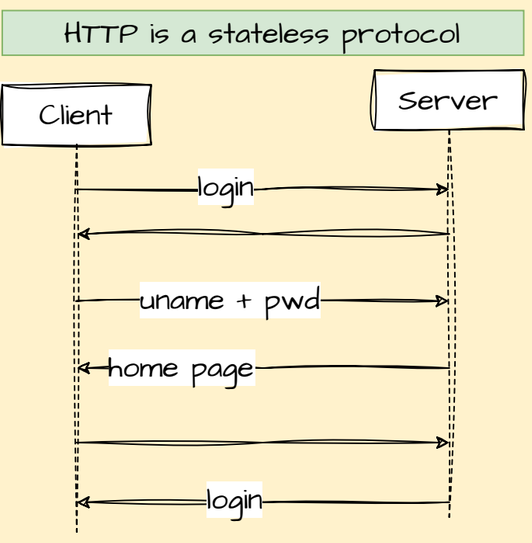


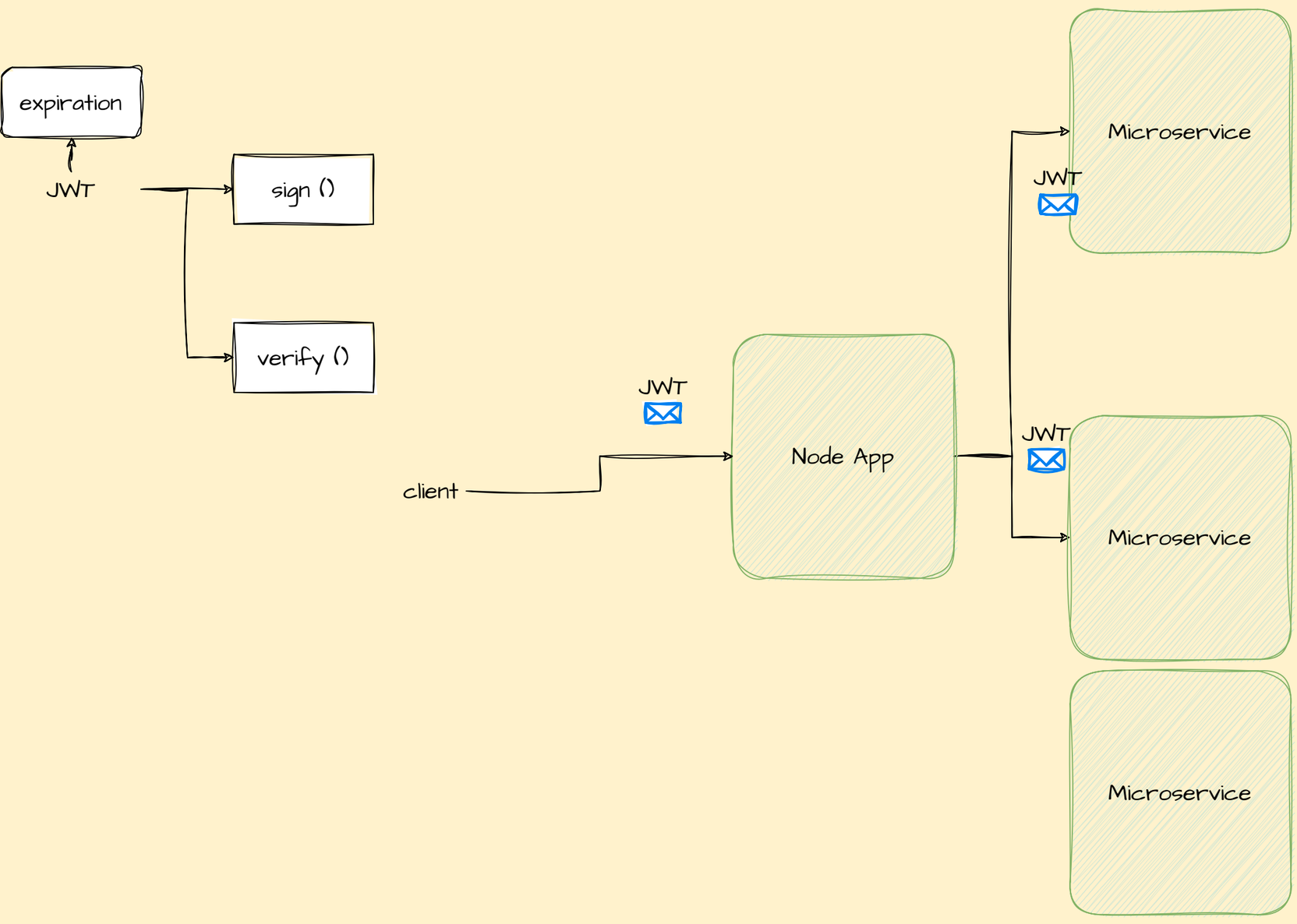
Steps
Connect to DB - connection string
GET data
Insert data
Delete data

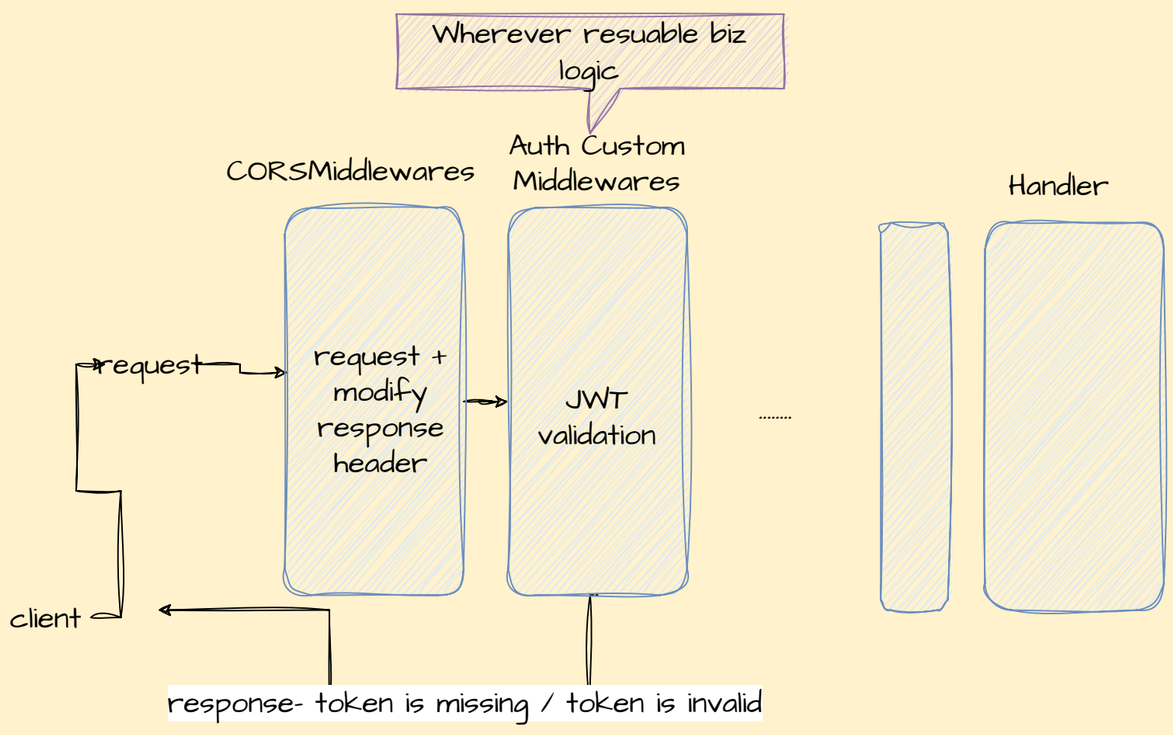
mongodb

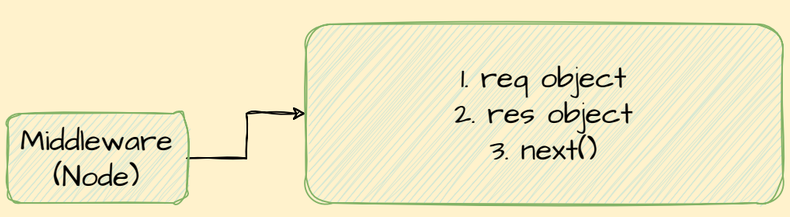












Nested Routes

```
<Route path="/" element={...} />
```

```
<Route path="/user" element={...} >
<Route path="profile" element={...}/>
<Route path="info" element={...}/>
</Route>
```

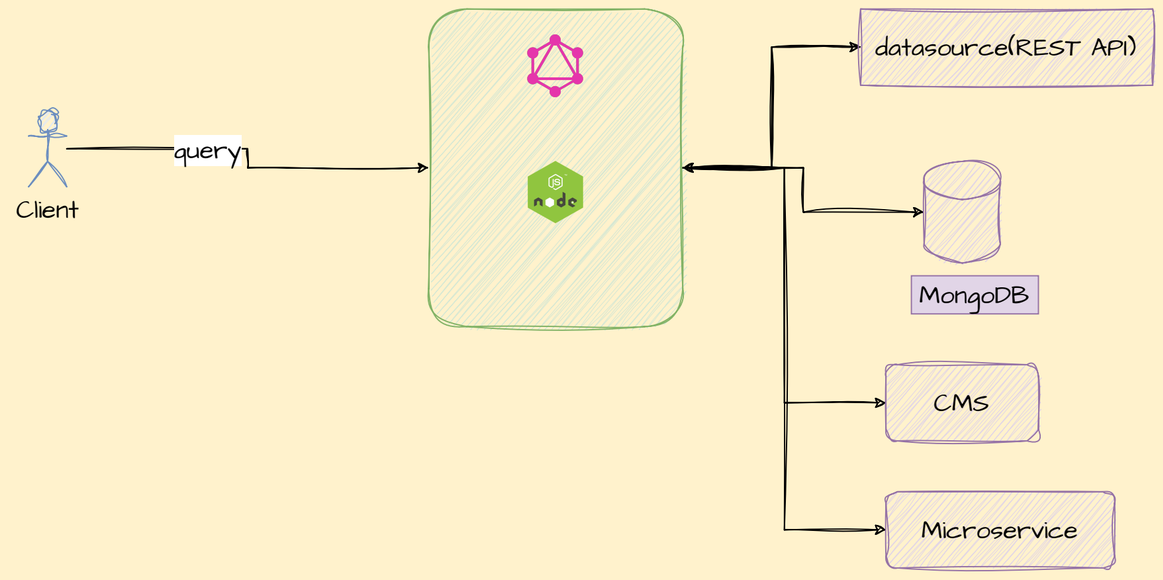
Custom Hook

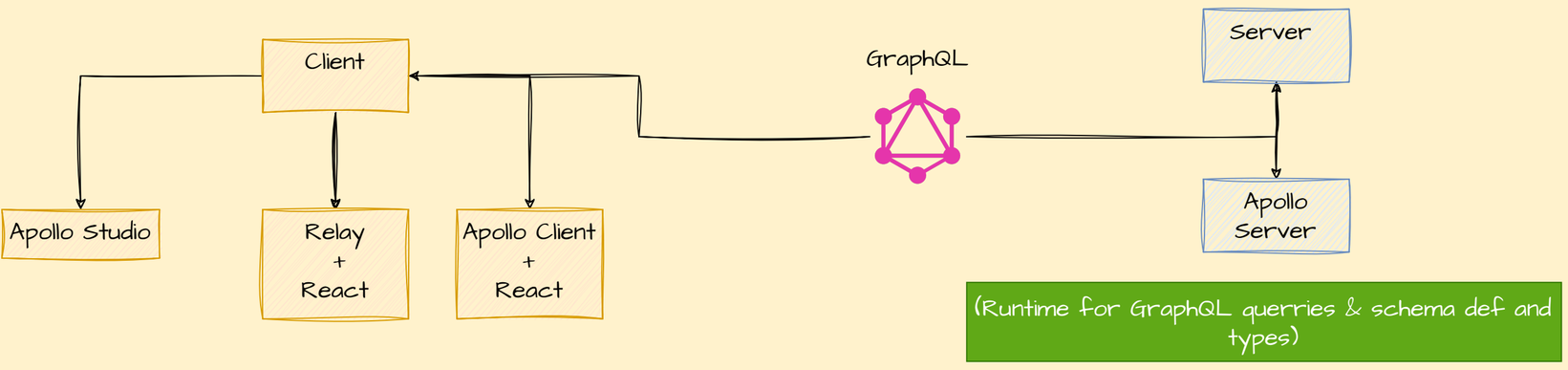
Higher Order Component

REST API
Fixed Endpoints
The structure of data (endpoint returns) is fixed
Cannot query selective data
Multiple requests for related data

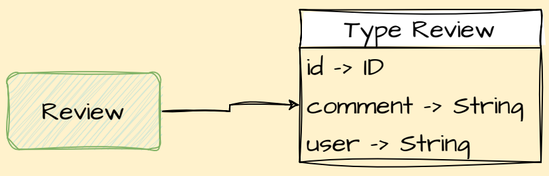
vs

GraphQL
flexible querries
The structure of data (query returns) is not fixed
Can query a selective data
Related data can be fetched in the same query





GraphQL Terminologies
Schema
Type Definitions
Type
Resolvers
Query (client | Root root server)
Mutations



To Do
Define the Type Review
Define Review Data [3 Reviews]
1. reviews -> all reviews 2. review(id:ID) -> single review