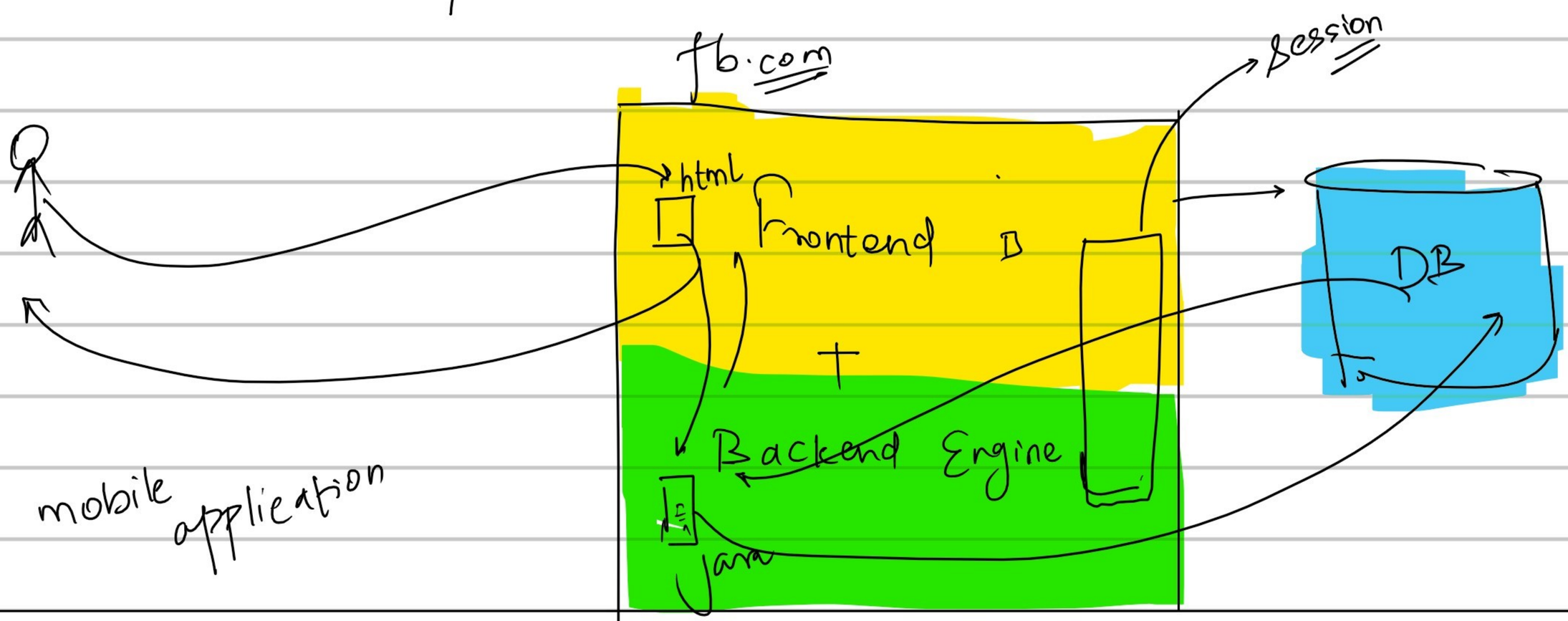
Full Stack

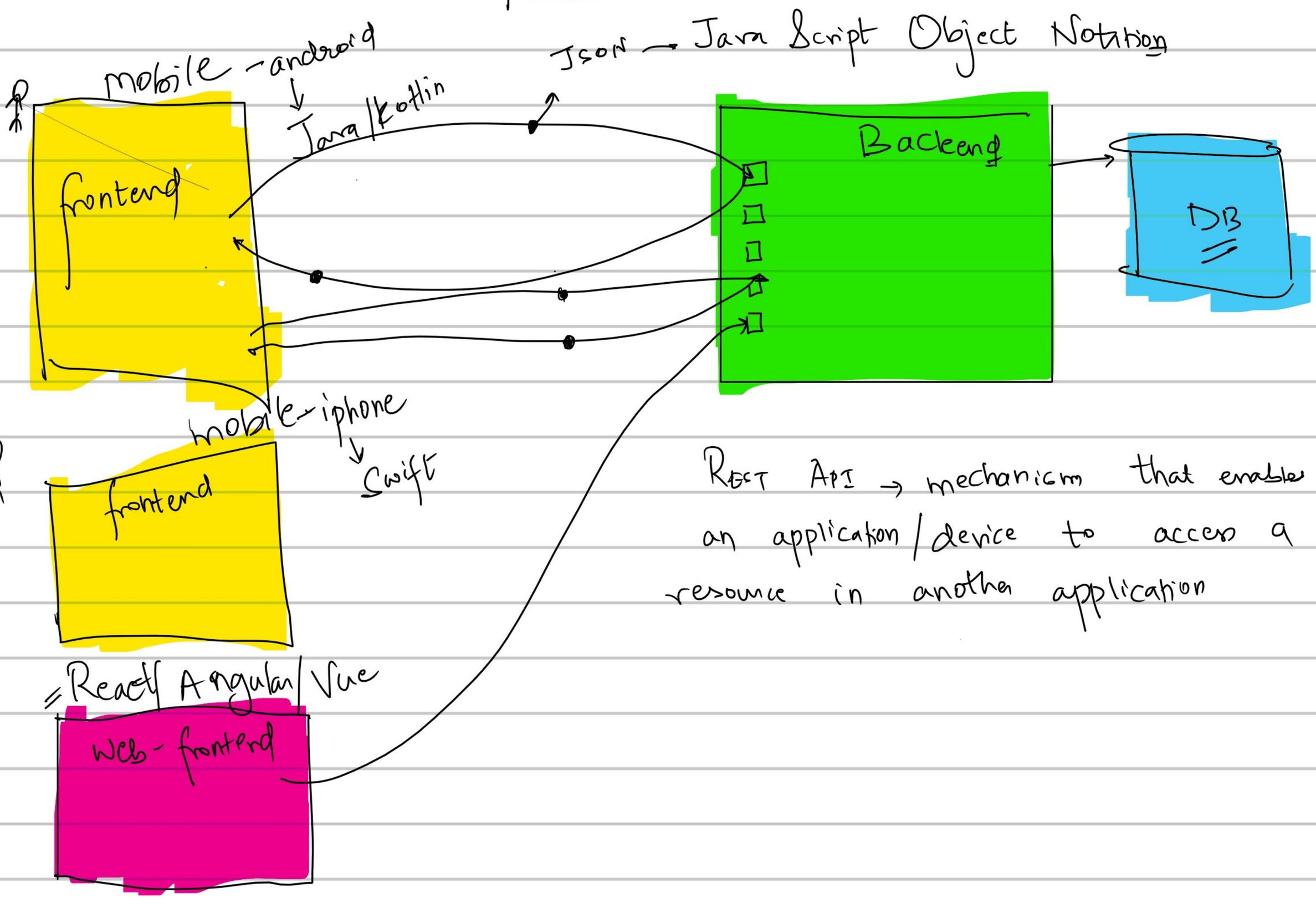
facebook

web

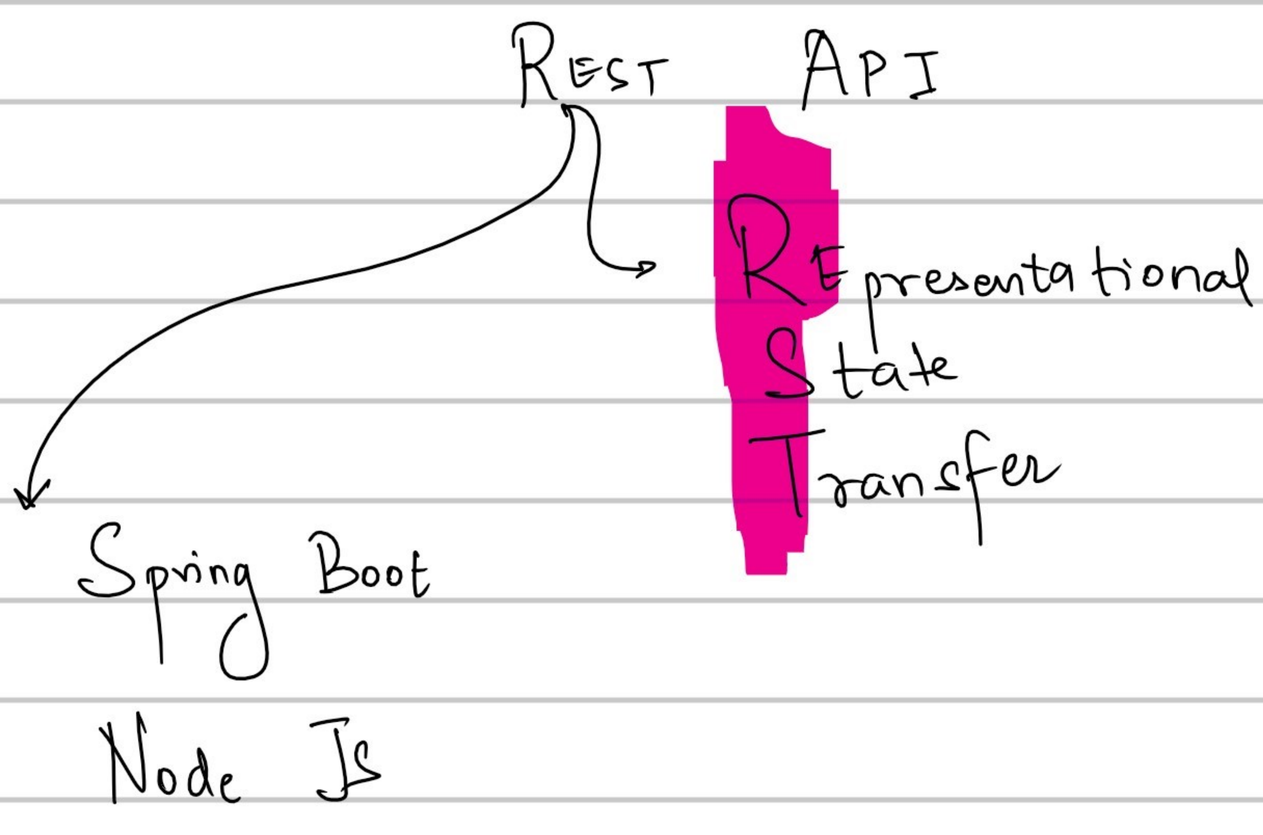
android

apple





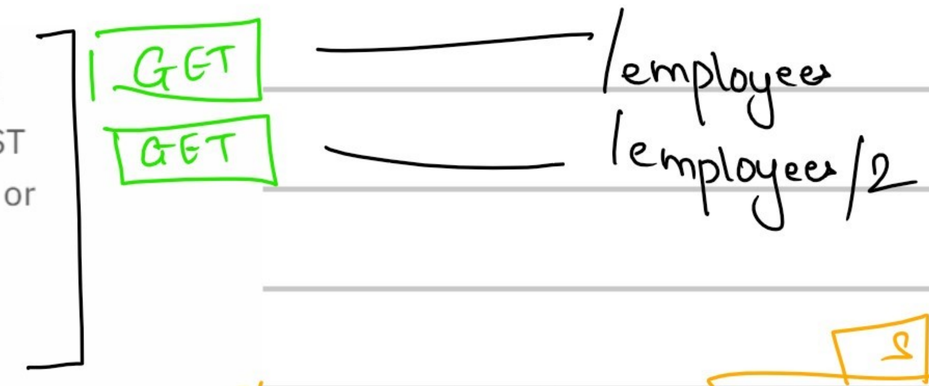
REST API → mechanism that enables an application / device to access a resource in another application



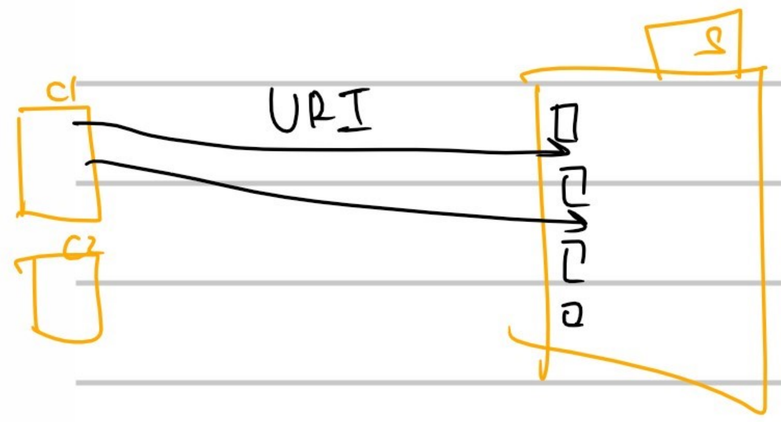
Django

Guiding Principles of REST API

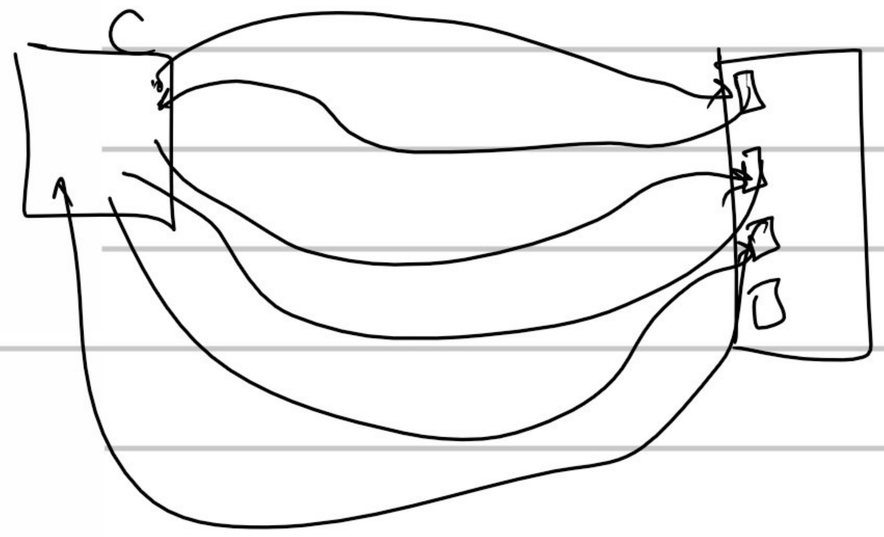
1. **Uniform interface**. All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn't be too large but should contain every piece of information that the client might need.



2. **Client-server decoupling**. In REST API design, client and server applications must be completely independent of each other. The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.



3. **Statelessness**. REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren't allowed to store any data related to a client request.



4. **Cacheability**. When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side.

5. **Layered system architecture**. In REST APIs, the calls and responses go through different layers. As a rule of thumb, don't assume that the client and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

6. **Code on demand (optional)**. REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets). In these cases, the code should only run on-demand.

GET → fetching the resource

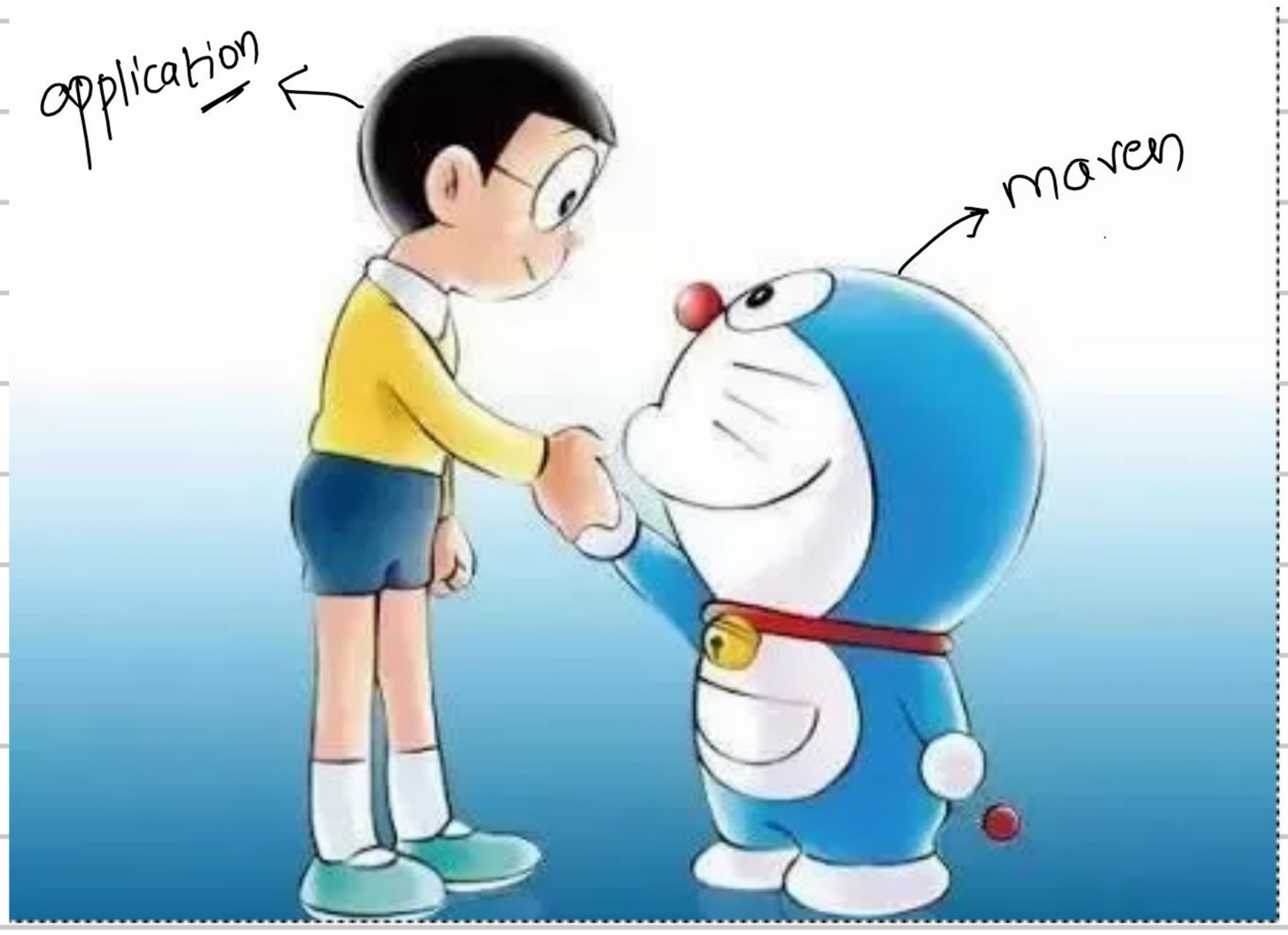
POST → creating a new resource

PUT → updating an existing resource

DELETE → deleting an existing resource

HTTP VERBS	customers
GET	https://api.xcb.com/customers
GET	https://api.xcb.com/customers/25
POST	https://api.xcb.com/customers → body
PUT	https://api.xcb.com/customers/26 body
DELETE	https://api.xcb.com/customers/25

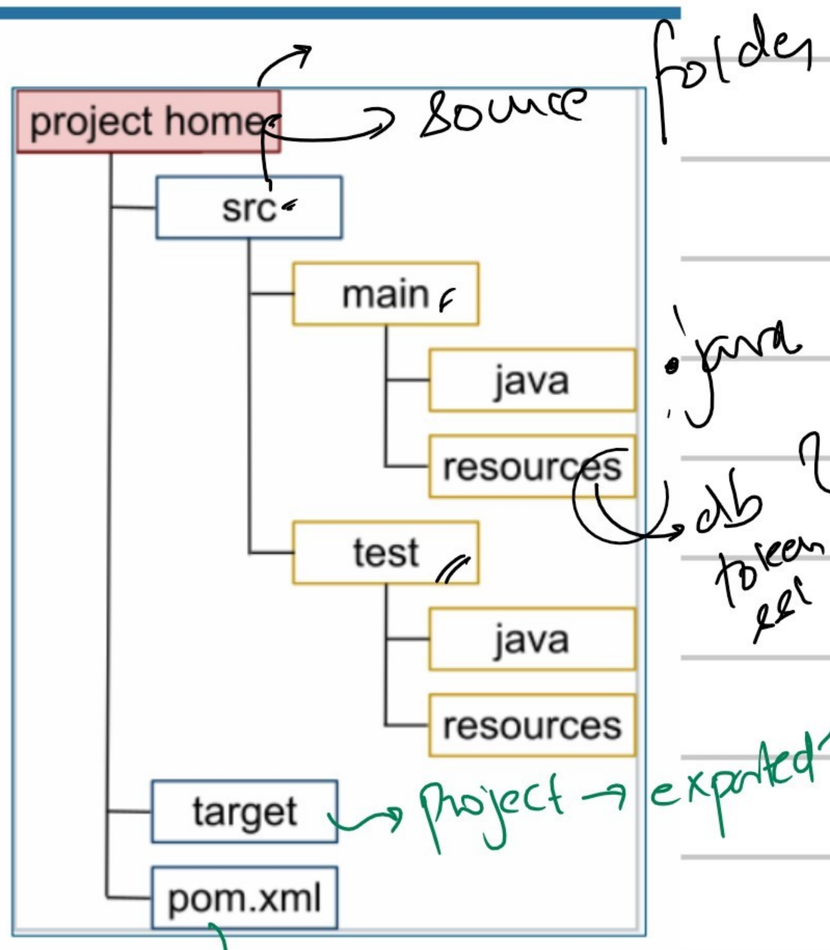
Spring Boot → Maven based application



Directory Structure in depth

aXess ACADEMY

Directory name	Purpose
project home	Contains the pom.xml and all subdirectories.
src/main/java	Contains the deliverable Java source code for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing Java sourcecode (JUnit or TestNG test cases, for example) for the project.
src/test/resources	Contains resources necessary for testing.

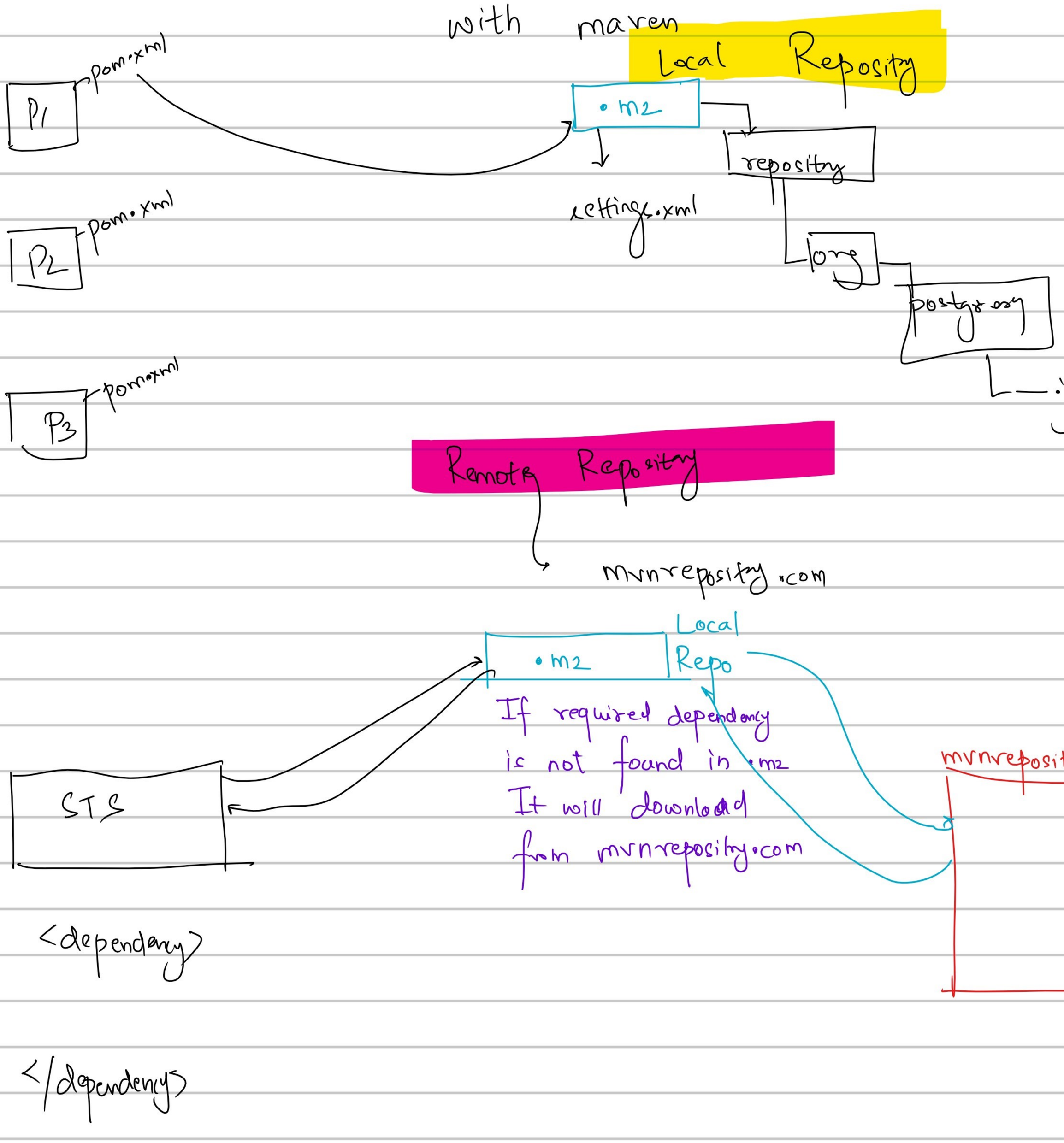


project object model
↳ for bringing in dependencies & managing information

pom.xml

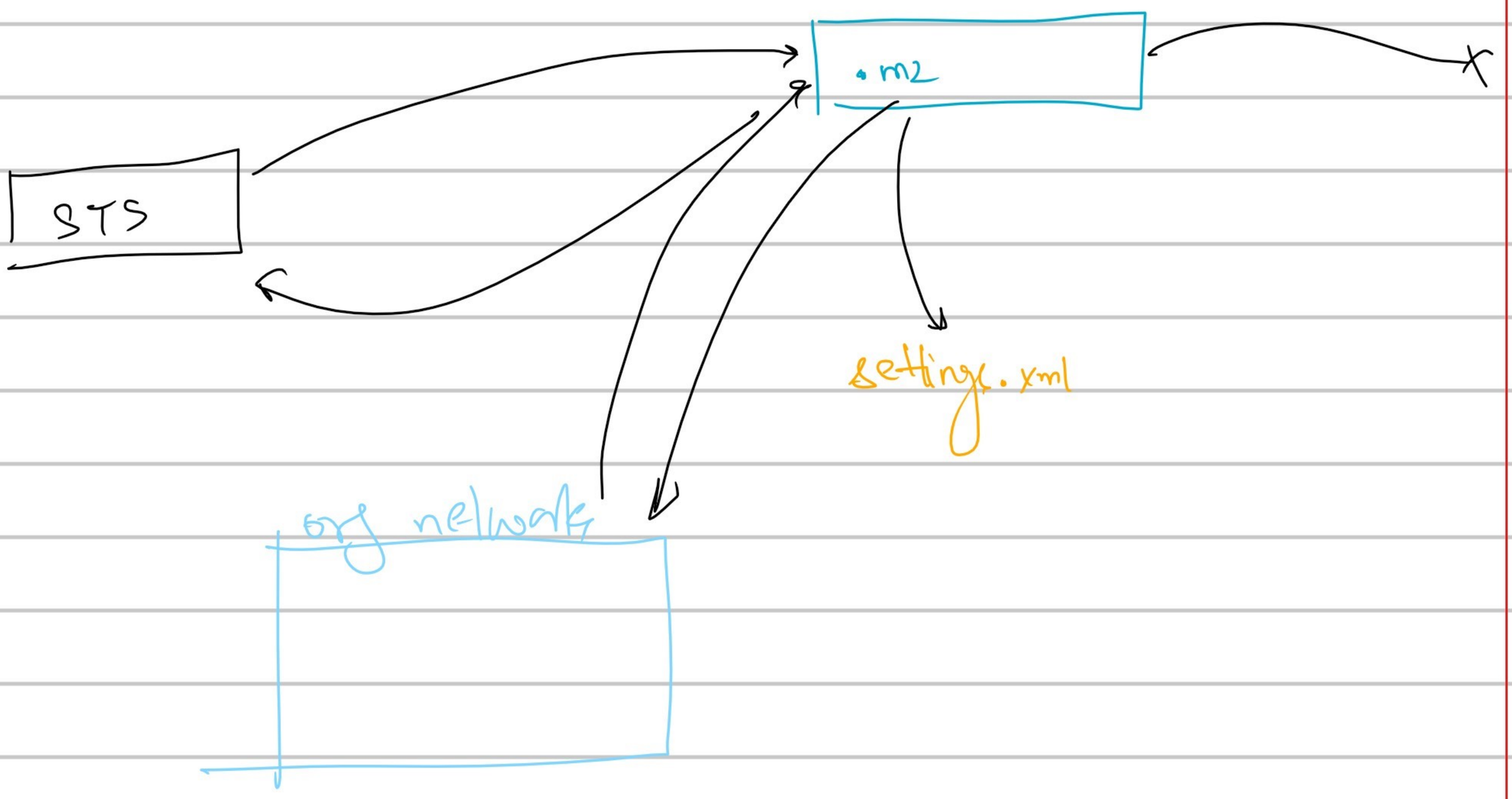
Without maven

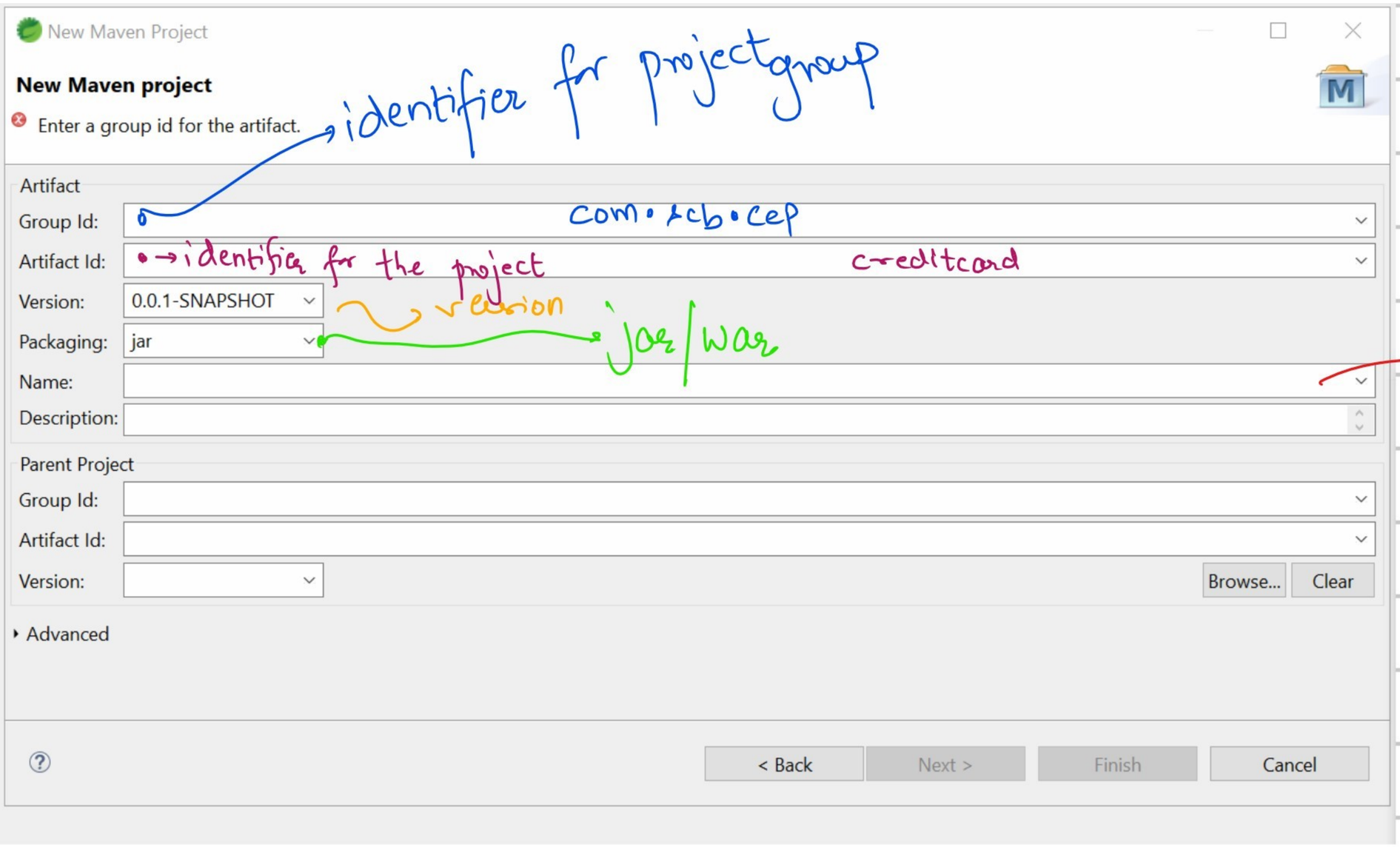
with maven



<dependency>

</dependency>

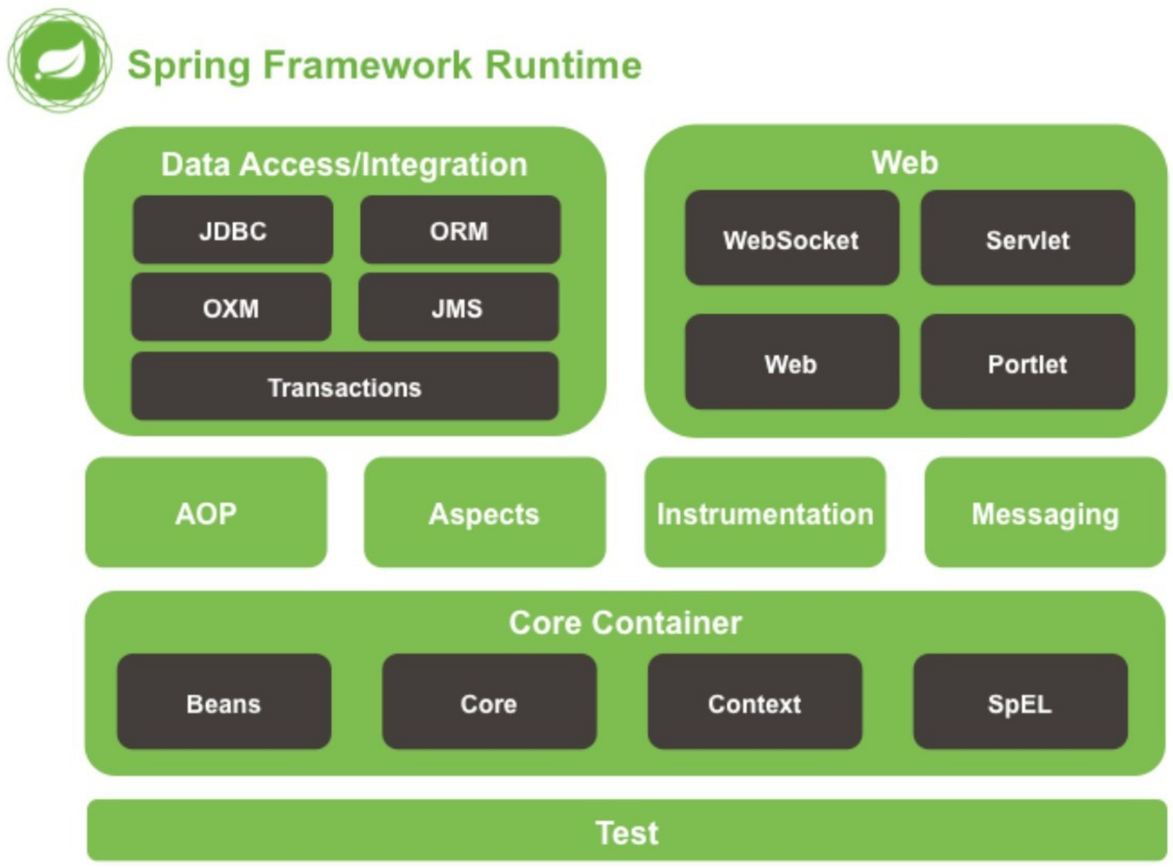

.m2
STS
mvnrepo
settings.xml
org.network


New Maven Project
New Maven project
Enter a group id for the artifact.
identifier for projectgroup
Artifact
Group Id:
com.icbc.cep
Artifact Id:
creditcard
Version:
0.0.1-SNAPSHOT
Packaging:
jar
Name:
Description:
Parent Project
Group Id:
Artifact Id:
Version:
Browse...
Clear
Advanced
< Back
Next >
Finish
Cancel
identifier for the project
version
jar/war
name of artifact

Spring Boot

→ SpringBoot is successor of Springframework.

→ SpringBoot works on top of Springframework



Springframework is known as framework of frameworks

lot of configuration was required

MVC

SpringBoot

makes it easy to create standalone, production grade Spring based application; that you can "Just Run"

* FEATURES OF SPRING BOOT *

① Auto-Configuation

② STAND ALONE

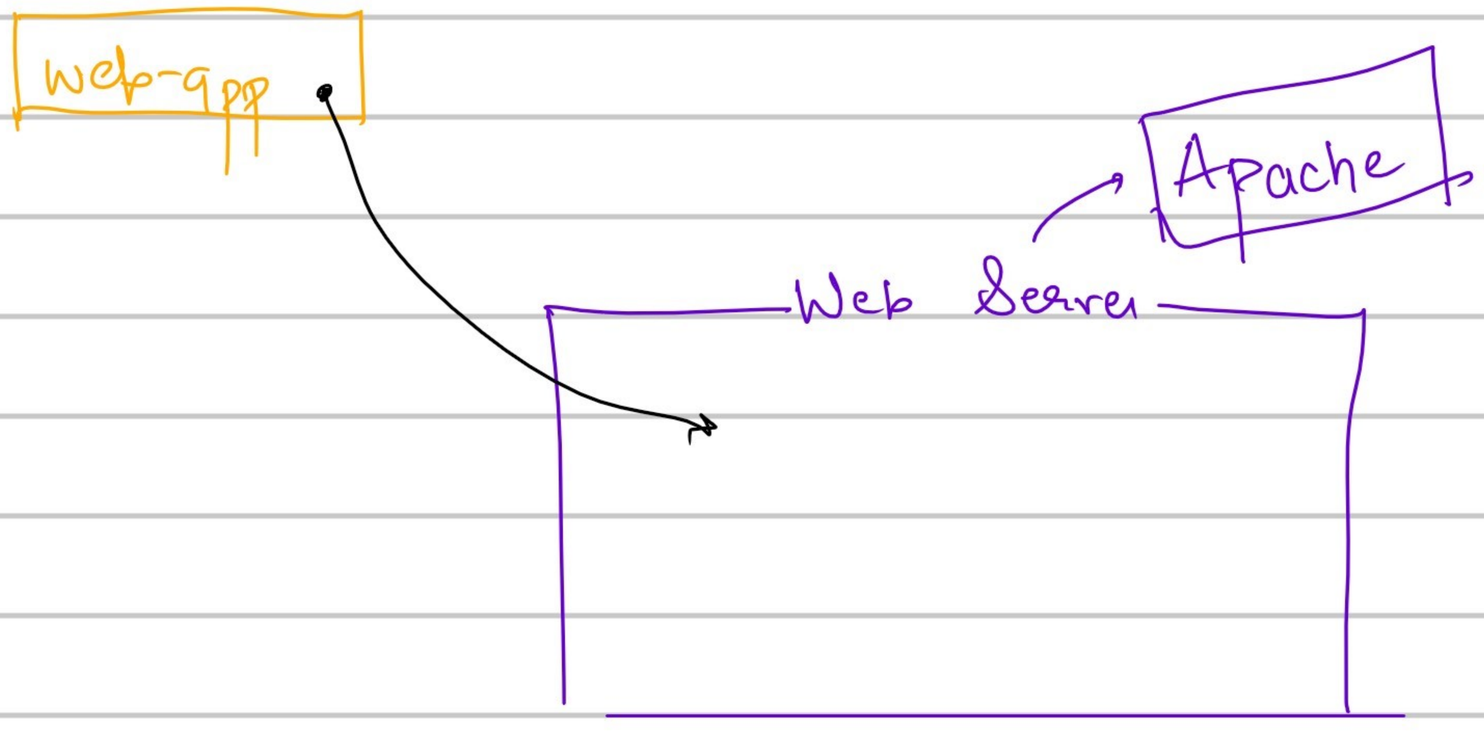
③ OPINIONATED

① Auto Configuration

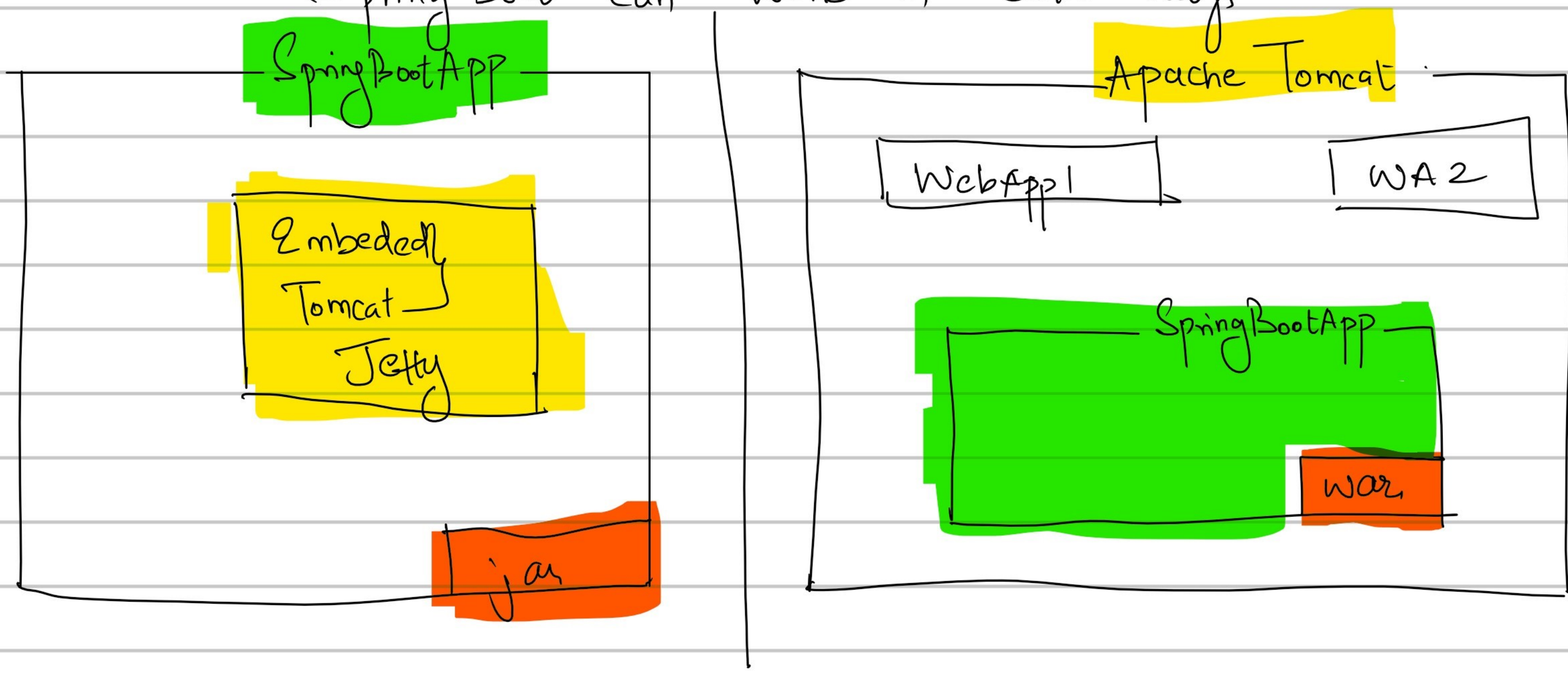
→ using Annotations @ we can configure the whole

→ Even with auto-configuration; Spring Boot allows for customization

② Stand-Alone



Spring Boot can work in both ways



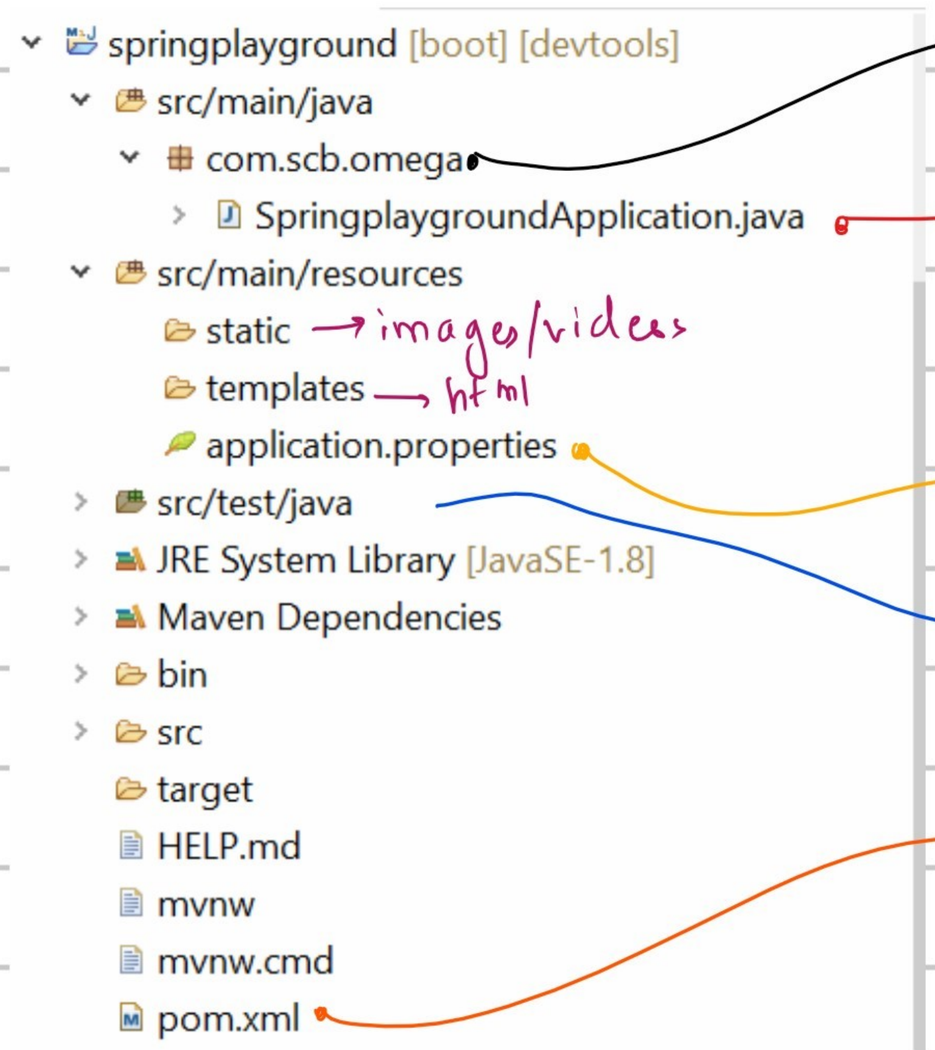
③ Opinionated

Spring Boot decides what defaults to use for configuration.

It also decides which packages are to be installed as a dependency.

Food Cart
↓
French Fries
+
Ketchup

Structure of a Spring Boot App



All packages & java file goes in com.scb.cep

Java file → main() → Runs the application
@SpringBootApplication

Properties are set
server.port
db-connect

Test cases goes here

where we add configurations & dependencies

@SpringBootApplication

```
1  package com.scb.cep;
2
3  import org.springframework.boot.SpringApplication;
5
6  @SpringBootApplication
7  public class SpringPlaygroundApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(SpringPlaygroundApplication.class, args);
11     }
12
13 }
14
```

@EnableAutoConfiguration
@ComponentScan
@Configuration

① Starts the Tomcat Server

② Runs the application

@ComponentScan

Com.scb.cep → base package

↳ @ComponentScan will scan in this base package

↳ Scan for Spring Component

Layered Architecture

com.scb.cep

com.scb.cep.controllers

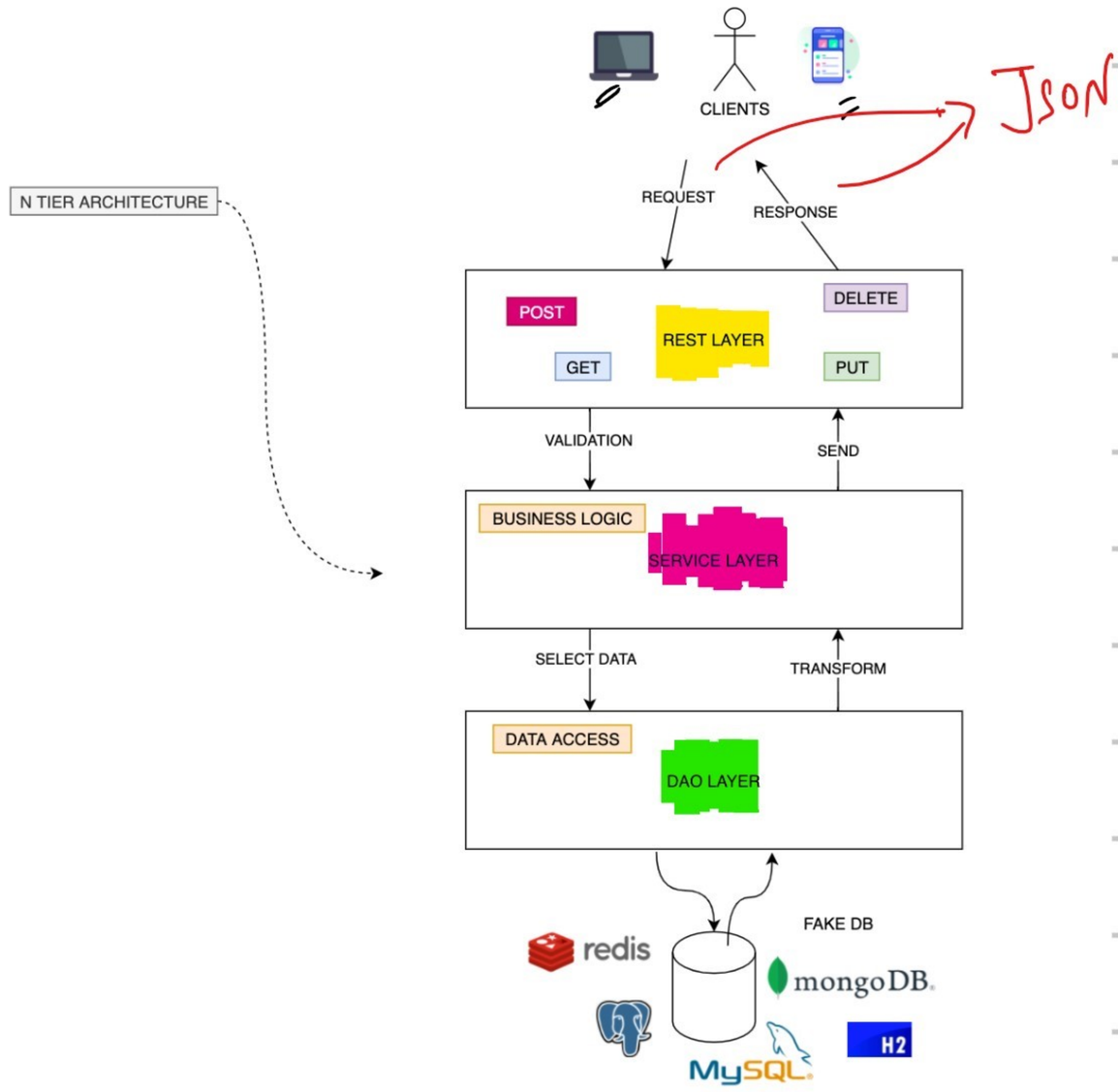
com.scb.cep.service

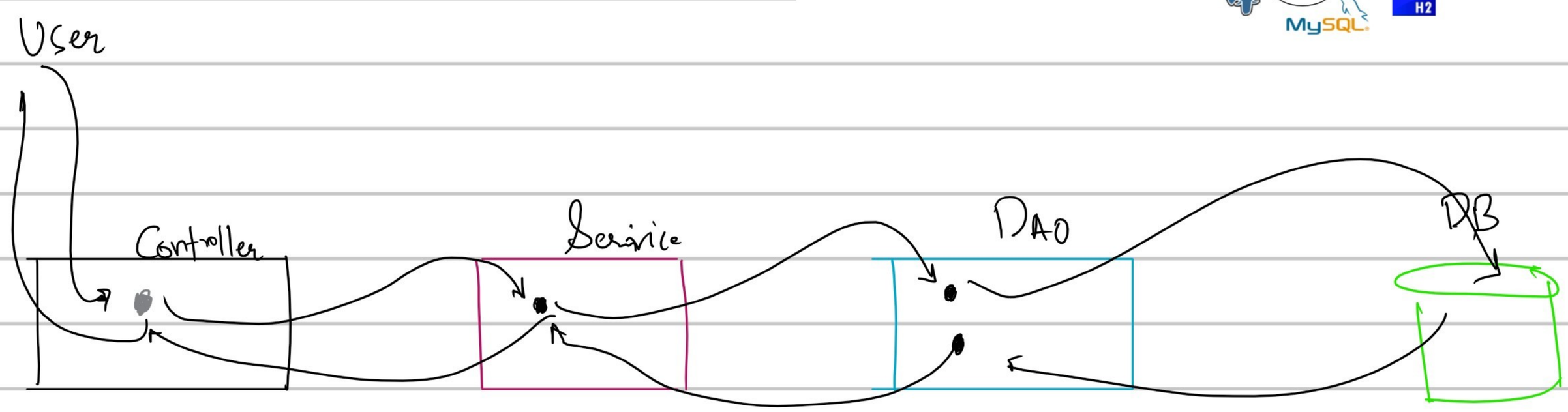
com.scb.cep.service.impl

com.scb.cep.dao

com.scb.cep.entities

com.scb.cep.exceptions





@Component → Specifies that the class is a Spring Boot Component

@Controller

@Service

@Repository

org.springframework.stereotype

Annotation Type Component

```
@Target(value=TYPE)
 @Retention(value=RUNTIME)
 @Documented
 @Indexed
public @interface Component
```

Indicates that an annotated class is a "component". Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

Other class-level annotations may be considered as identifying a component as well, typically a special kind of component: e.g. the `@Repository` annotation or AspectJ's `@Aspect` annotation.

Since:

2.5

```
@RestController
public class HelloController {
  @RequestMapping(path="/hello", method=RequestMethod.GET)
  public String sayHello(){
      return "Helo CR";
  }
```

@RequestMapping

@GetMapping

@PostMapping

@PutMapping

@PatchMapping

@DeleteMapping

* How to fit a REST API in Java Class *

① One Class One API

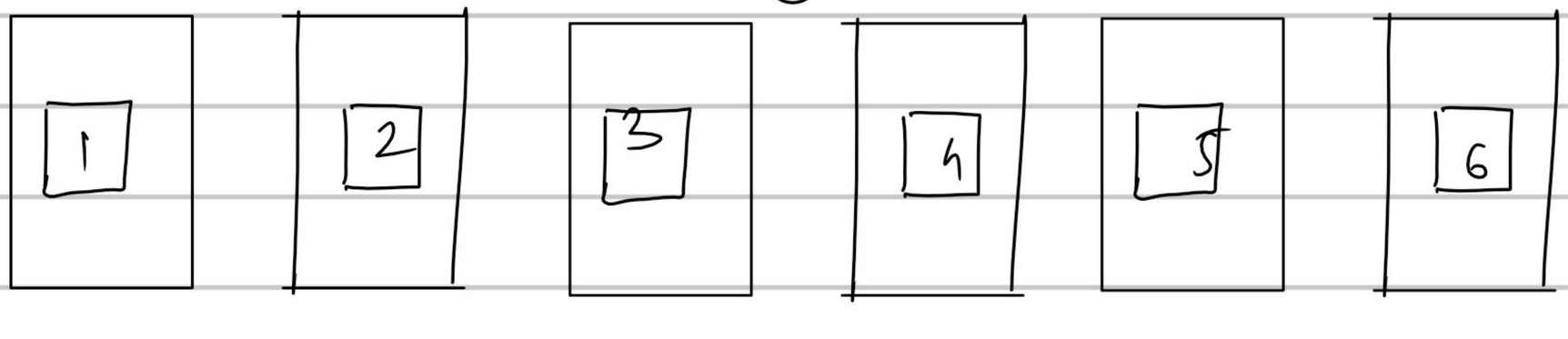
② One Class All API

③ One Class Some API

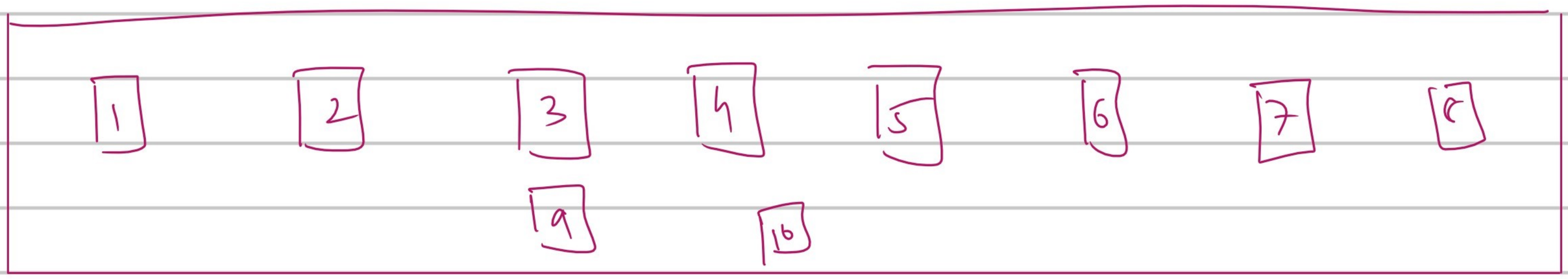
Customer
- /customers → GET
- /customers/25 → GET
- /customes → POST
- /customes/26 → PUT
- /customes/25 → DELETE

order
- /orders → GET
- /orders/1001 → GET
- /orders → POST
- /orders/1002 → PUT
- /orders/1001 → DELETE

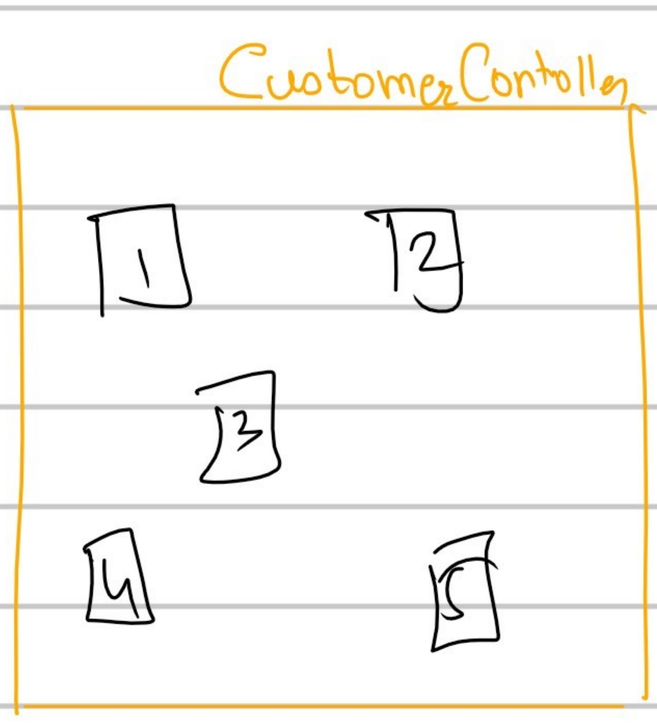
①

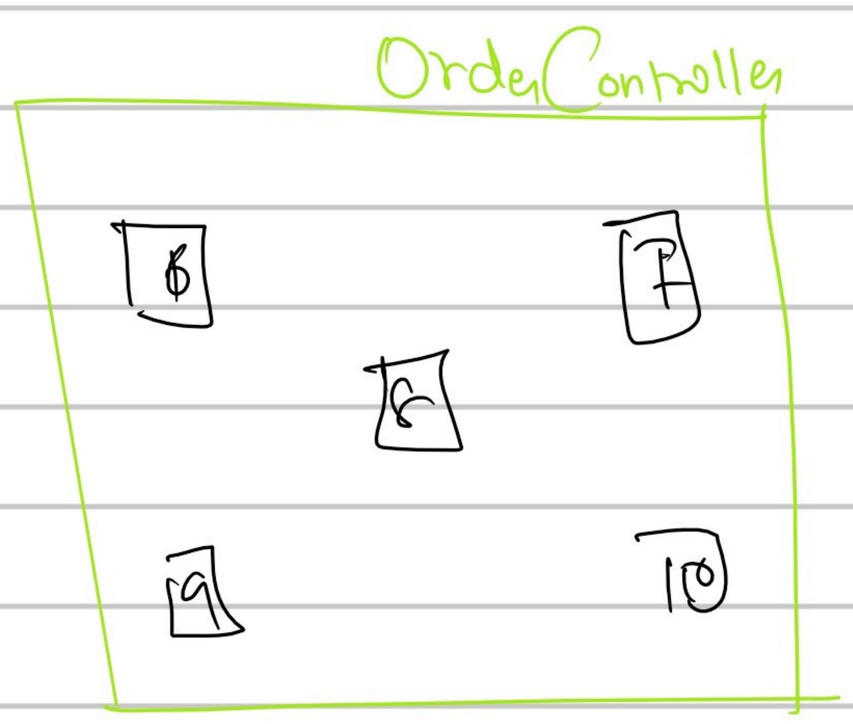


②



③

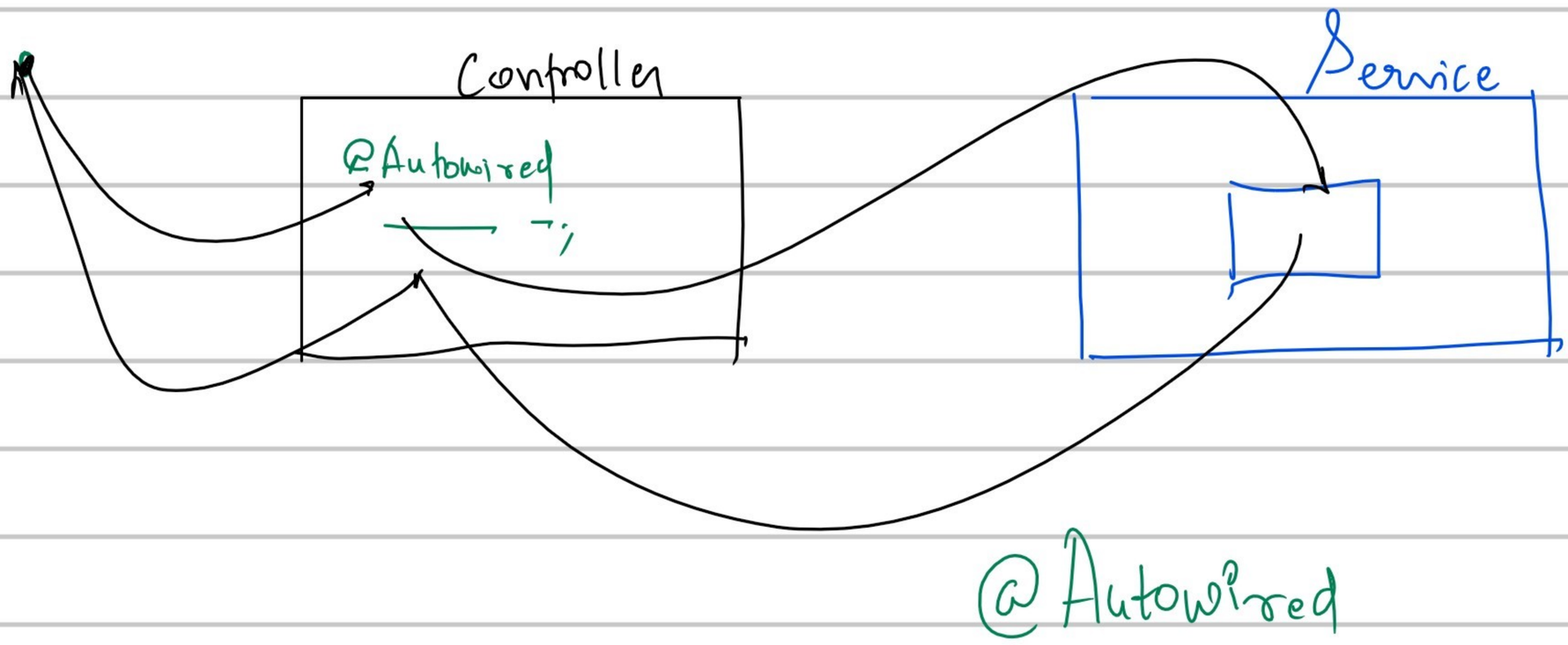




* PathVariable & * Request Parameters

http://localhost:9099/user/5 → Path Variable

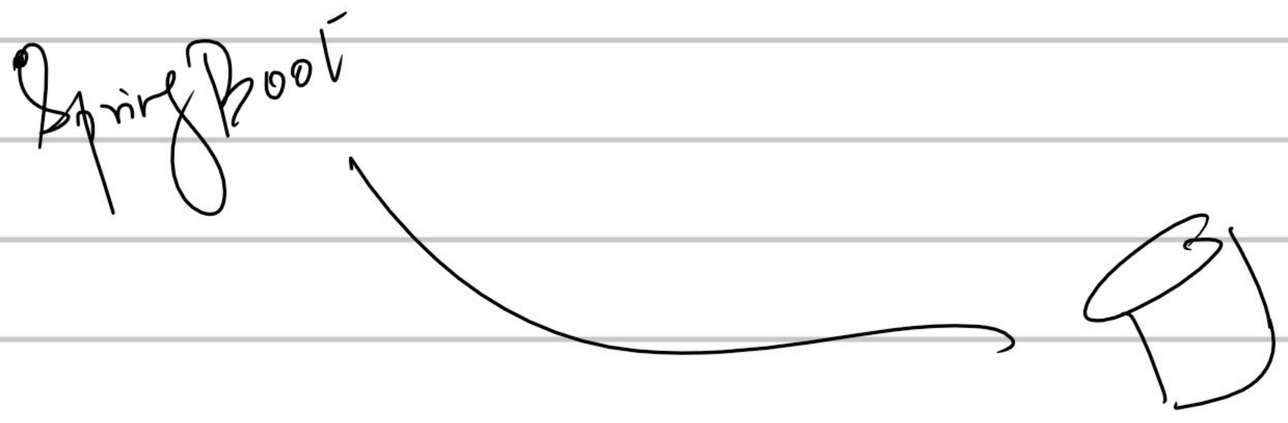
http://localhost:9099/user?id=5 → Request Parameter

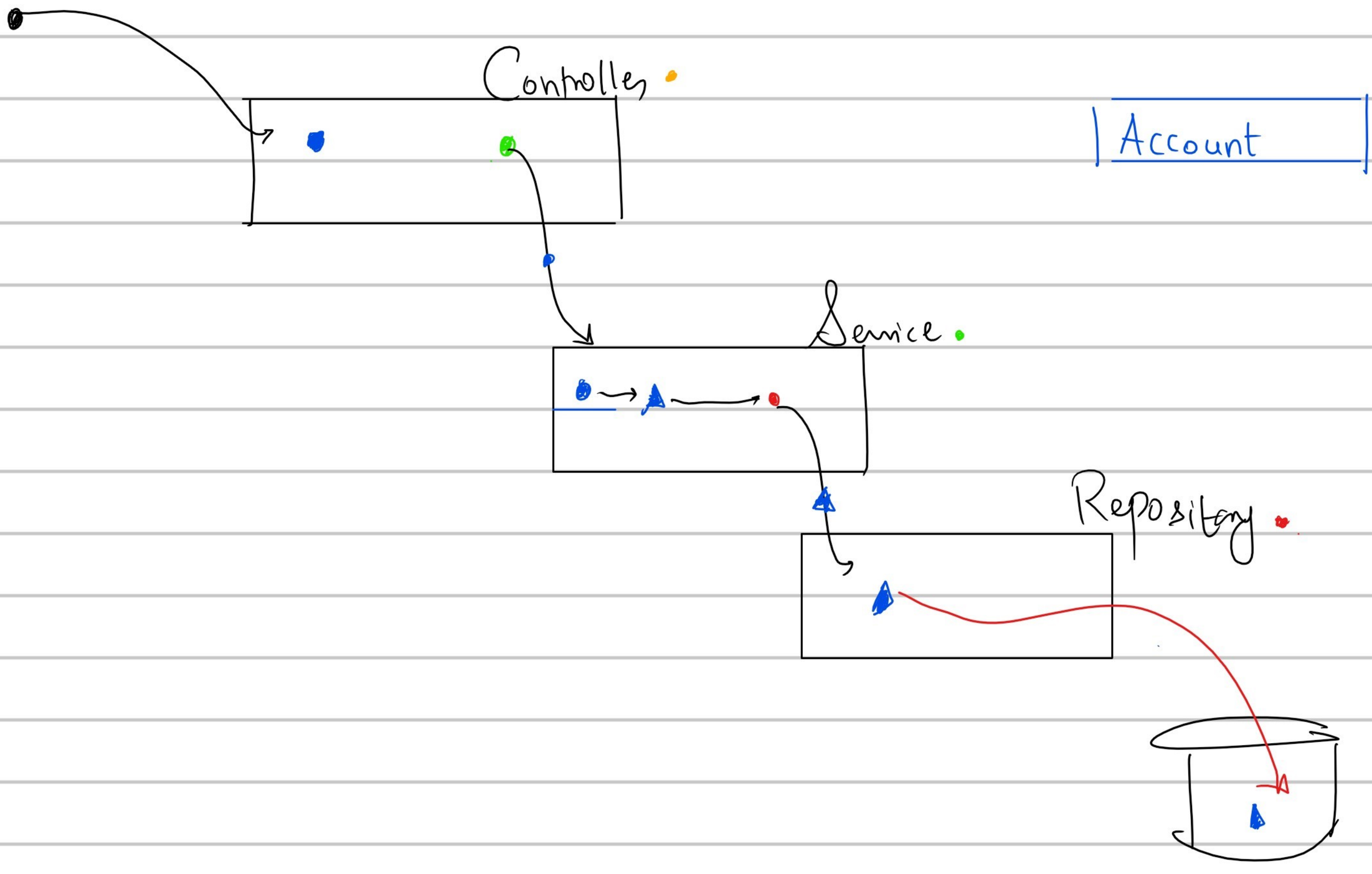


* Spring Data JPA *

↓

Java Persistence API.





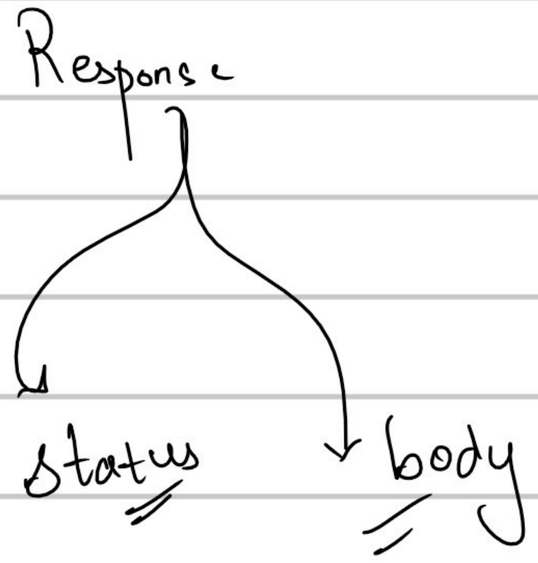
HTTP Status Codes

When a browser request a service from a web service, a response code will be given.
These are the list of HTTP Status code that might be returned

1XX Information	
100	Continue
101	Switching Protocols
102	Processing
103	Early Hints
2XX Success	
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
205	Reset Content
206	Partial Content
207	Multi-Status (WebDAV)
208	Already Reported (WebDAV)
226	IM Used (HTTP Delta Encoding)
3XX Redirection	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
306	Unused
307	Temporary Redirect
308	Permanent Redirect
4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable

Compiled by Ivan Tay.

4XX Client (Continue)	
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	URI Too Large
415	Unsupported Media Type
416	Range Not Satisfiable
417	Exception Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity (WebDAV)
423	Locked (WebDAV)
424	Failed Dependency (WebDAV)
425	Too Early
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
451	Unavailable for Legal Reasons
499	Client Closed Request
5XX Server Error Responses	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
507	Insufficient Storage (WebDAV)
508	Loop Detected (WebDAV)
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error



Java Script

* let vs const
* var
* objects
* Arrow functions
* map, filter
* Object Destructuring
* Spread Operator
* Class - Inheritance
* Named & Default Exports

* Arrow functions *

- cleaner way of creating traditional JS function
- arrow function can be assigned to a variable or can be passed as an argument to a higher order function

* Syntax

```
let someVar = (args1, args2, ...., argN) => {
                statement(s)
}
```

String Interpolation in JS

```
value = 10
`Value is ${value} and Square is ${square}`
"Value is " + value + " Square is " + square
```

Object Destructuring

Named & Default Exports

person.js

```
export default class Person {

}

export function m1(){

}

export function m2(){

}
```

trainer.js

```
import Person, {m1, m2} from 'person.js'

class Trainer extends Person {

}
```

runner.js

Named → import {....} from '——'

Default → import ... from '——'

```
import React, {useState} from 'react';
```

Default

Named

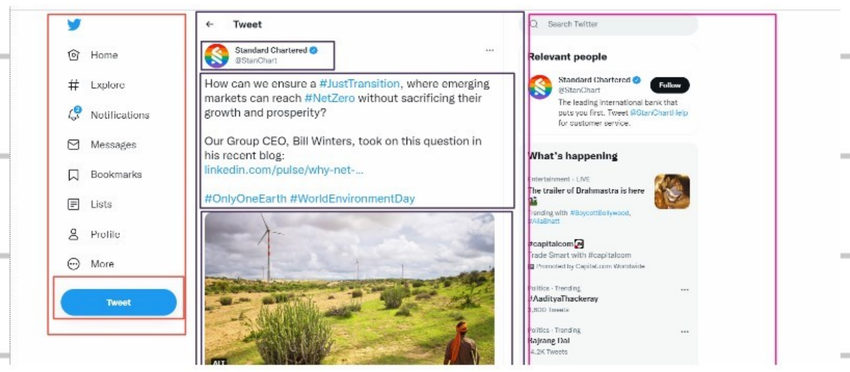
React → UI library

React was developed at facebook.

→ JavaScript based front-end

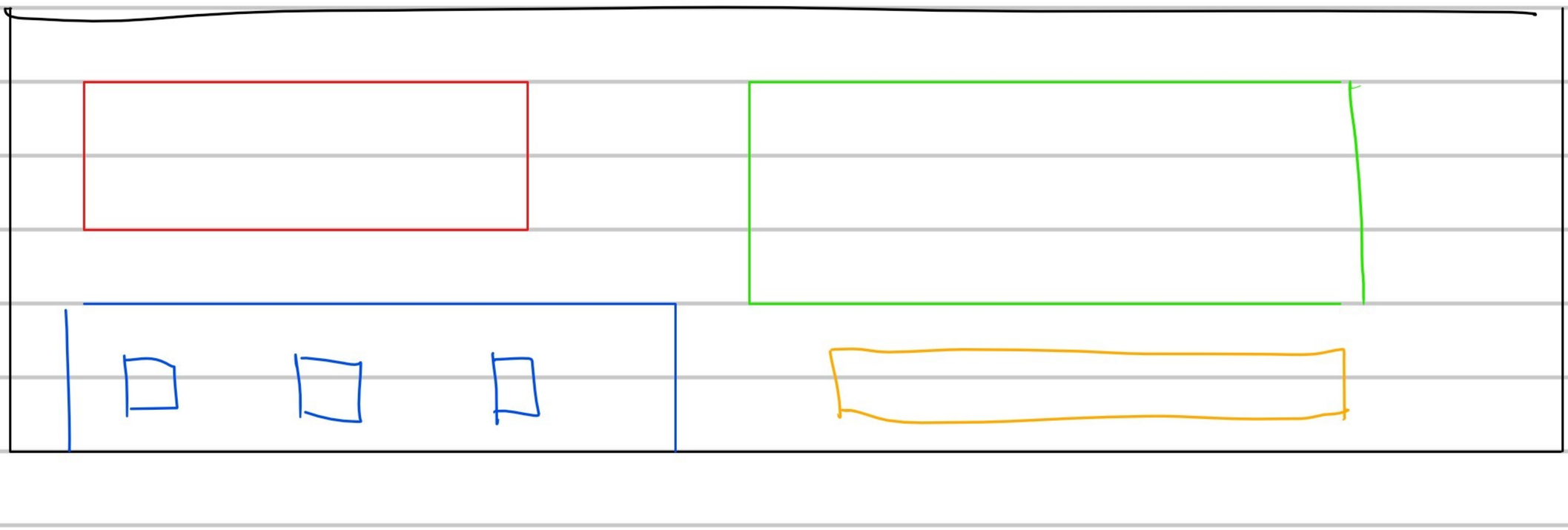
→ UI library.

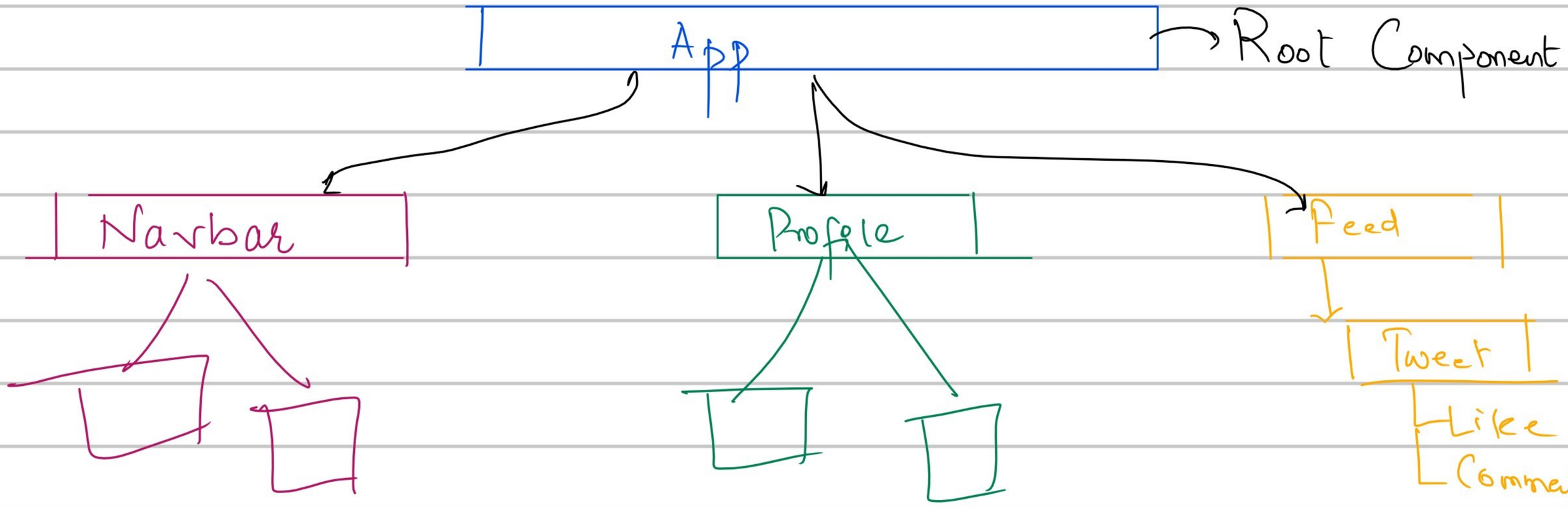
React is made up of Components!!



Component

piece of UI; who has it's own STATE & LIFECYCLE





React uses JSX as it base language

JavaScript XML

→ syntax extension to JavaScript.

```
const element = <h1> Hello &CB </h1>
```

element → JS

<h1> Hello &CB </h1> → HTML

```
let name = 'Zartab'

const el = <h1> Hello {name} </h1>
```

{name} → React Component

Setting up React Environment

```
npm i -g create-react-app
```

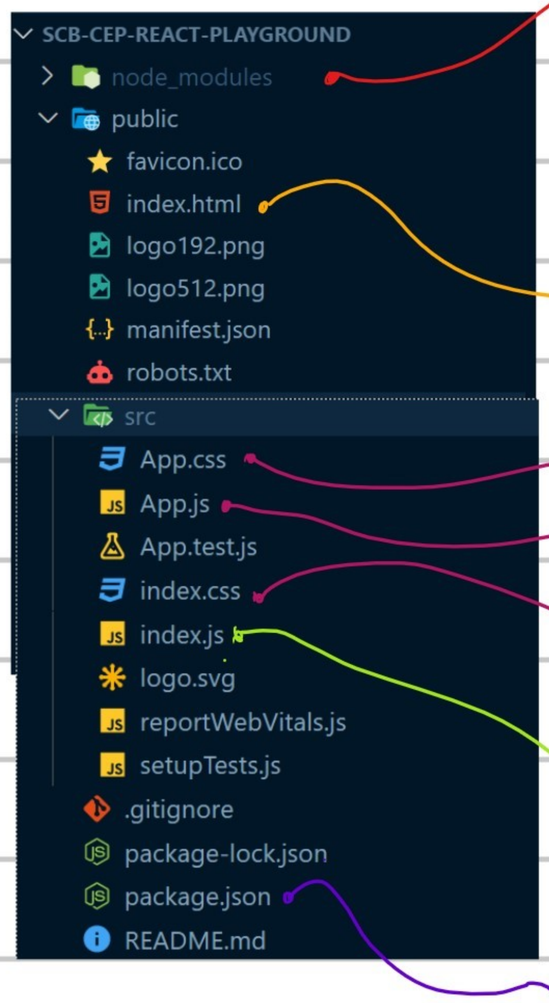
To create a New React App

```
npx create-react-app name-of-app
```

To start React App

```
cd name-of-app

npm start
```

Structure of a React Application



Contains all the required & imported library Packages.

renders the data given by index.js

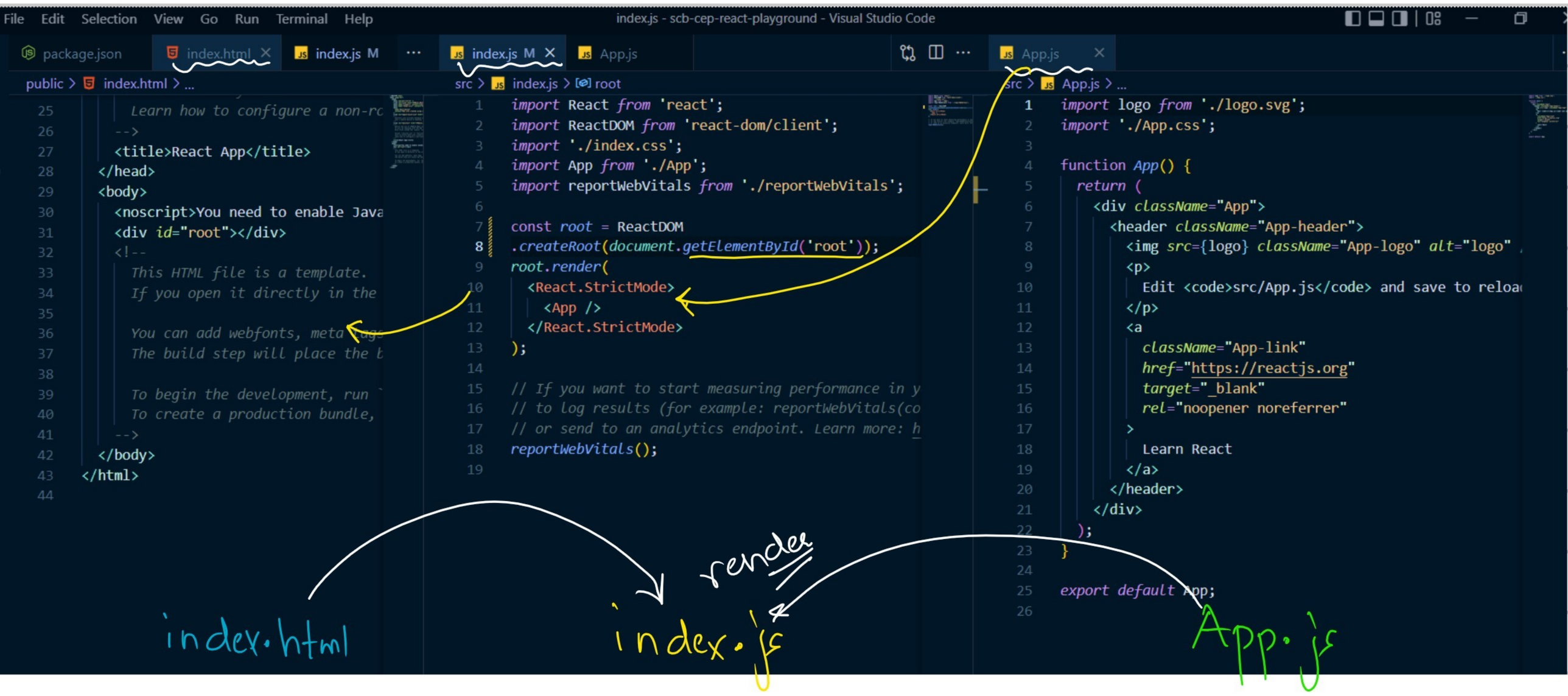
styles App Component.

Root React Component.

styles index.html

main js file which loads Root Component

Configuration file



ProductComponent → App

<ProductComponent />

* React Components *

- **Stateful Component**
  - can have it's own state
- **Stateless Component**
  - cannot have state [hooks]

```
class Tweet {                    → React Component
    state = {                    → State goes here
    }
    render() {                   → UI Implementation
        return React Element     → return React Element
    }
}
```

```
1   import React, { Component } from "react";
2
3   export class HelloComponent extends Component {
4
5       render() {
6           return (
7               <h2>Hello Component</h2>
8           )
9       }
10  }
```

JSX → always returns a single parent

```
<div>
    <h1> —</h1>
    <img> —/img>
</div>
```

```
<h1> — </h1>
<img> — <h1>
```

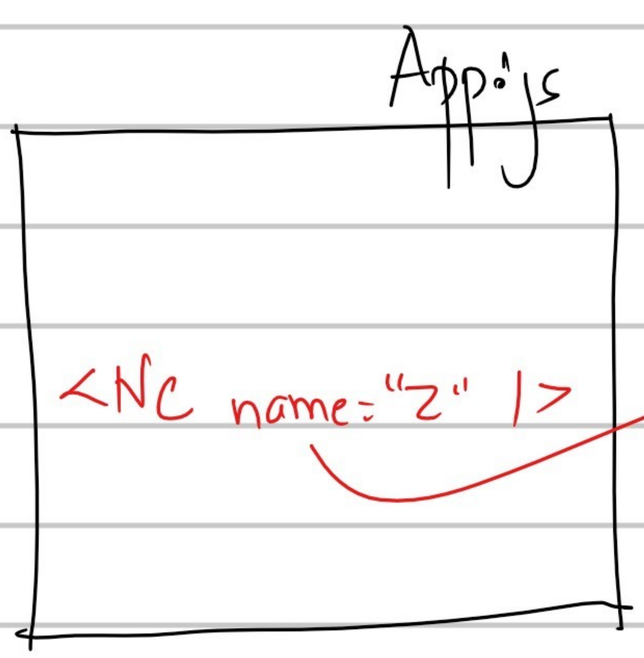
* Props *

→ HTML components have attributes

attribute

→ React Components have props

<NameComponent name="Zartab" />



props → equivalent to attributes

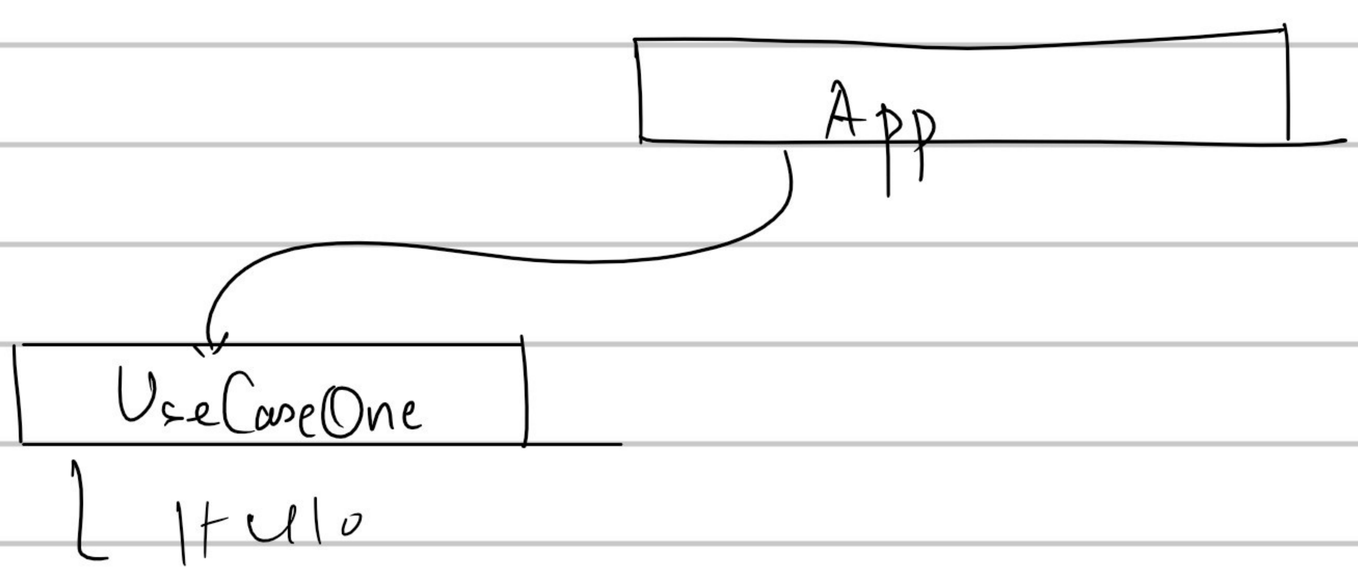
there can any number of props & they can any name.

App.js

<NameComponent
name="Zartab">

① Class Component → Stateful
→ available in class component as an object → props

② Function Component → Stateless
→ available as an argument

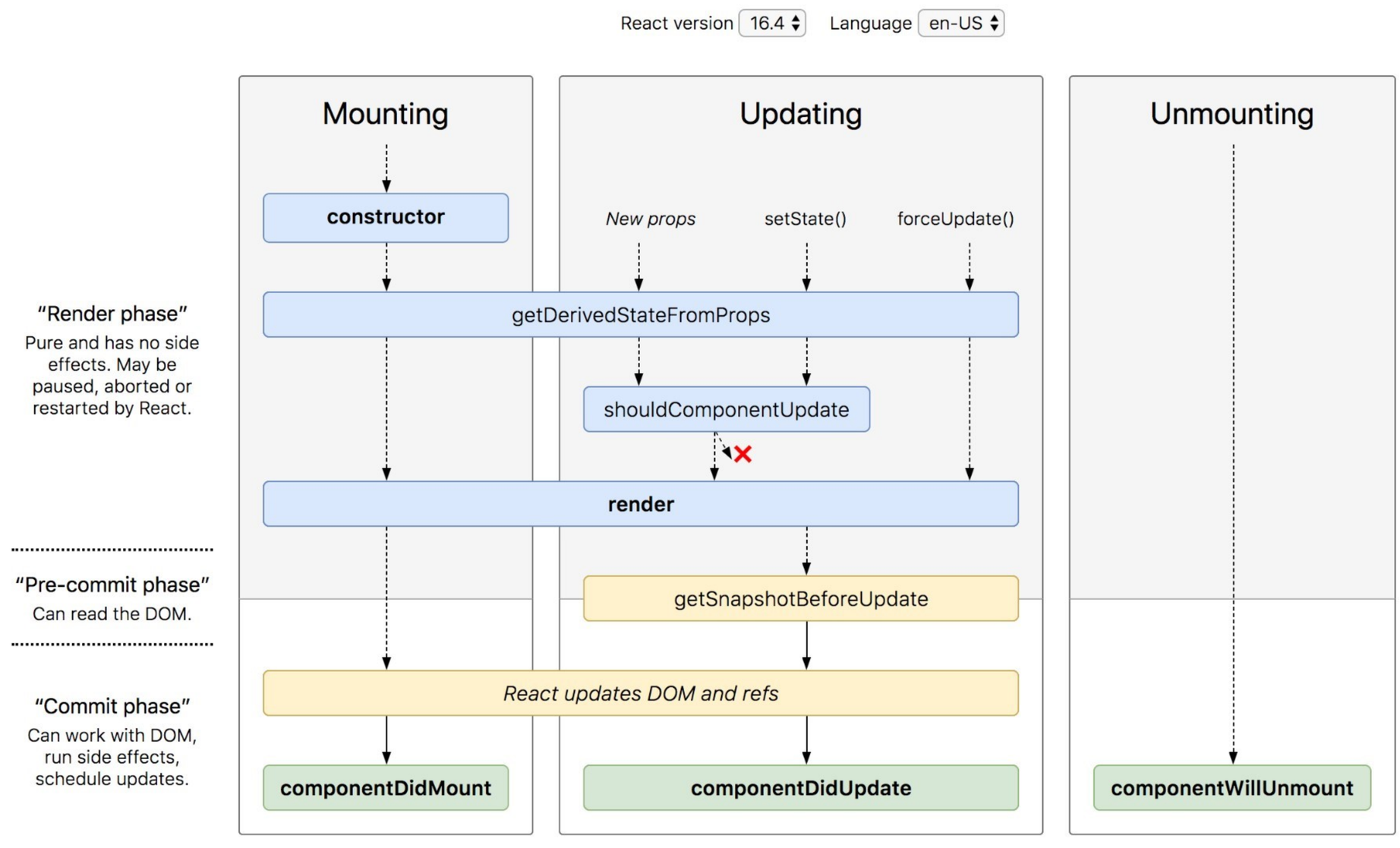


* State *

State is a plain JS object used by React to represent information about a component's current situation.

```
class Tweet {                    → React Component
  state = {
                                 → State goes here
  }
  render() {                     → UI Implementation
                                 → return React Element
  }
}
```

State is like props; but it is private & fully controlled by Component.



Hooks & State

→ Special type of functions which allows us to break class components into smaller functional Component.

---

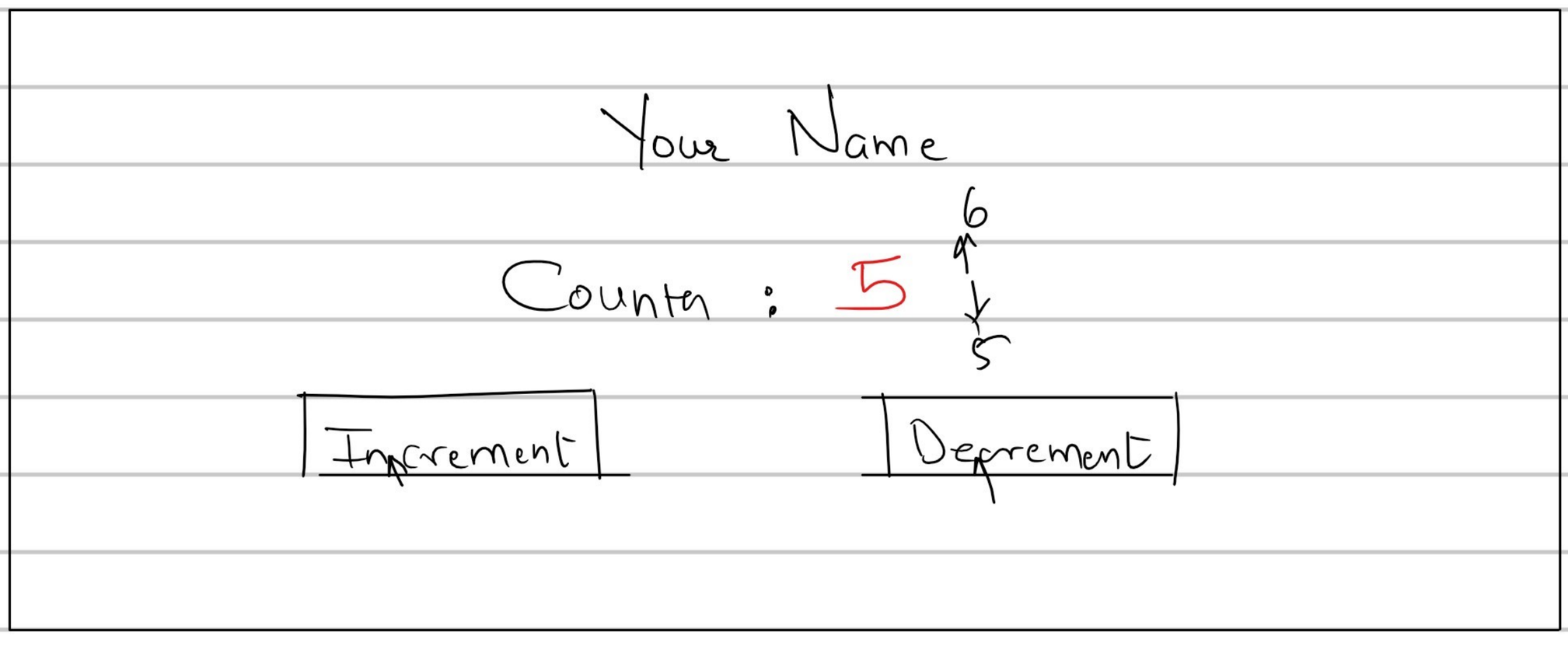
useState → hook

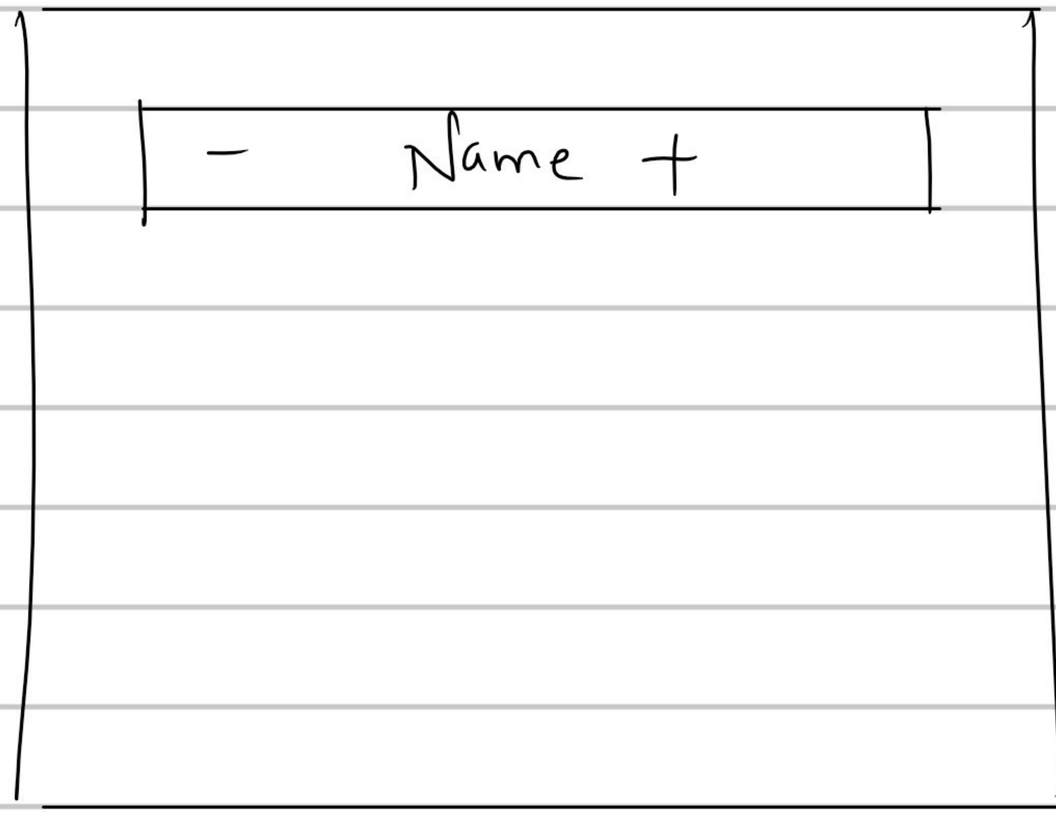
```
const [variable, setVariable] = useState(defaultValue)
```

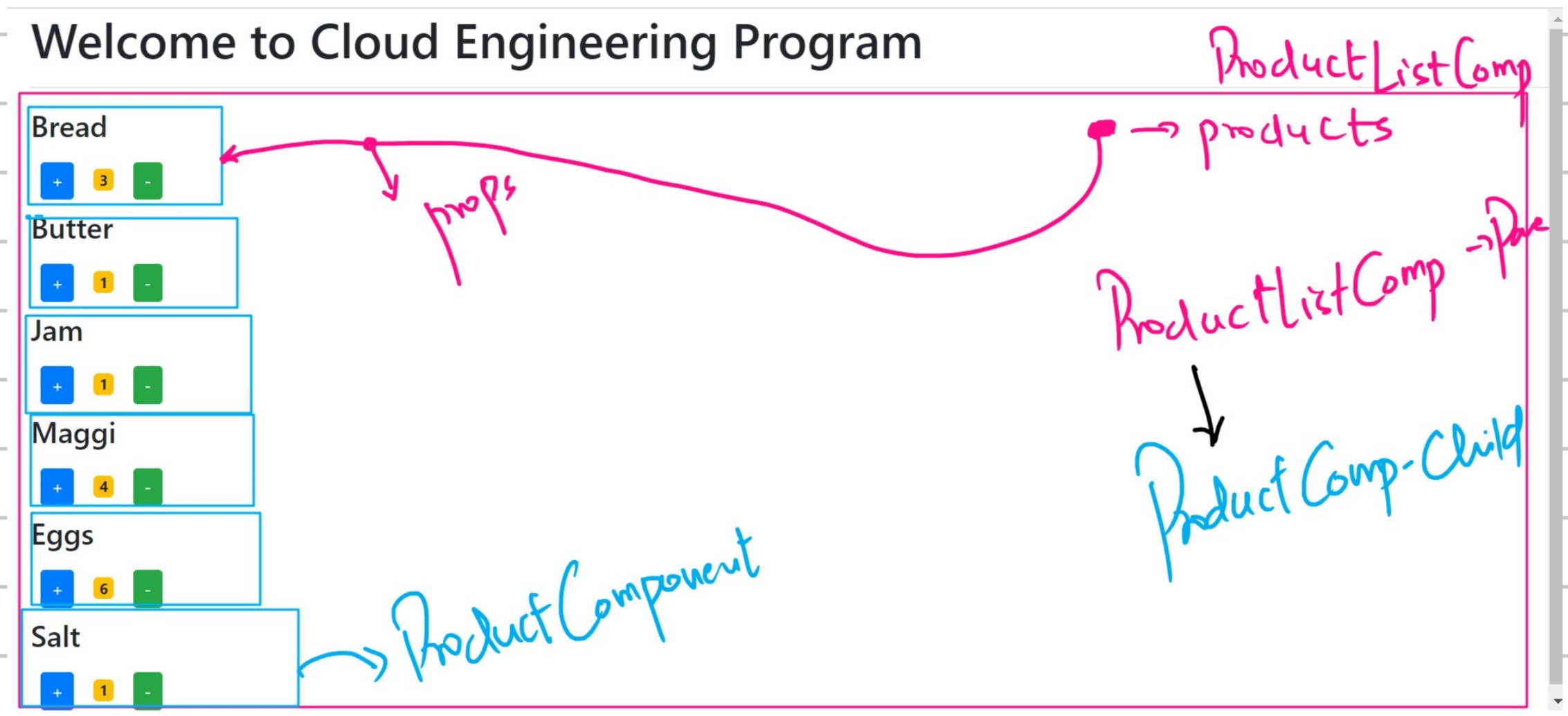
- variable → state
- setVariable → function to update state
- useState → hook for creating state in function component
- defaultValue → default value for state variable

```
const [productCount, setProductCount] = useState(0);
```

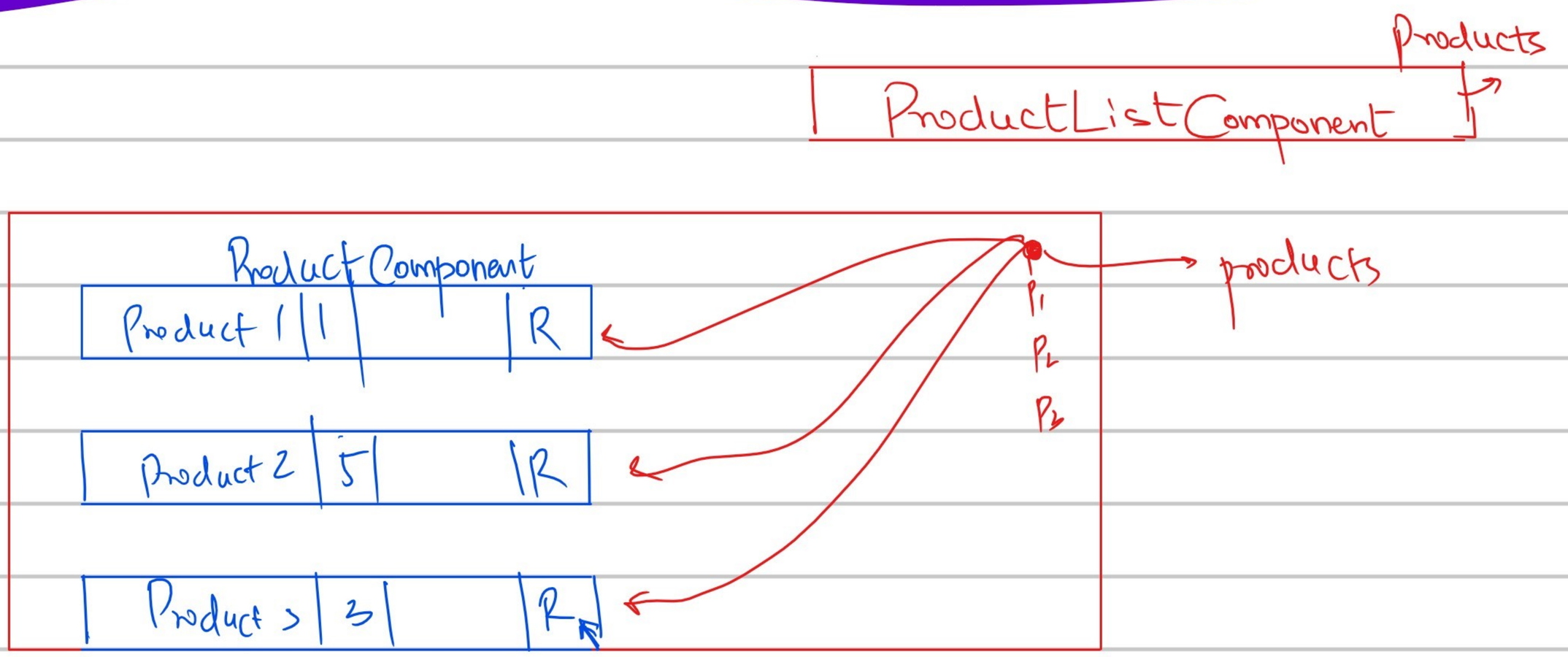
```
const [age, setAge] = useState(0);

const [fruit, setFruit] = useState('Apple');

const [colors, setColors] = useState([])

const [address, setAddress] = useState({})
useState({city:'xyz'})
```




- Name +


Welcome to Cloud Engineering Program
Bread
+ 3 -
Butter
+ 1 -
Jam
+ 1 -
Maggi
+ 4 -
Eggs
+ 6 -
Salt
+ 1 -
ProductComponent
props
ProductListComp
→ products
ProductListComp → Pro
ProductComp-Child

State of a component can only be modified by The component to which it belongs!!



1. **ProductListComponent** is responsible for updating the state
   - removeFromCart()
2. **ProductListComponent** will pass the function (removeFromCart) to **ProductComponent** → via props
3. **ProductComponent** will invoke the function passed by **ProductListComponent**

* React Routing *

React App → SPA → Single Page Application

<BrowserRouter>

holds all the browsing history

</BrowserRouter>

* Route *

<Route> → component of react-router-dom

→ renders a particular component when URL matches

* Switch *

renders first available Component

* Link *

→ declarative, accessible navigation around project

```
<a href="/products">Products</a>
```

→ `<Link to="/products">Products</Link>`

* Route props *

react_devtools_backend.js:4082

{sortBy: 'newest', match: {…}, location: {…}, history: {…}, staticContext: undefined}

history:
- action: "PUSH"
- block: ƒ block(prompt)
- createHref: ƒ createHref(location)
- go: ƒ go(n)
- goBack: ƒ goBack()
- goForward: ƒ goForward()
- length: 50
- listen: ƒ listen(listener)
- location: {pathname: '/products', search: '', hash: '', state: undefine
- push: ƒ push(path, state)
- replace: ƒ replace(path, state)
- [[Prototype]]: Object

location:
- hash: ""
- key: "tyxqlk"
- pathname: "/products"
- search: ""
- state: undefined
- [[Prototype]]: Object

match:
- isExact: true
- params: {}
- path: "/products"
- url: "/products"
- [[Prototype]]: Object

sortBy: "newest"

staticContext: undefined

[[Prototype]]: Object

navigate

your current location

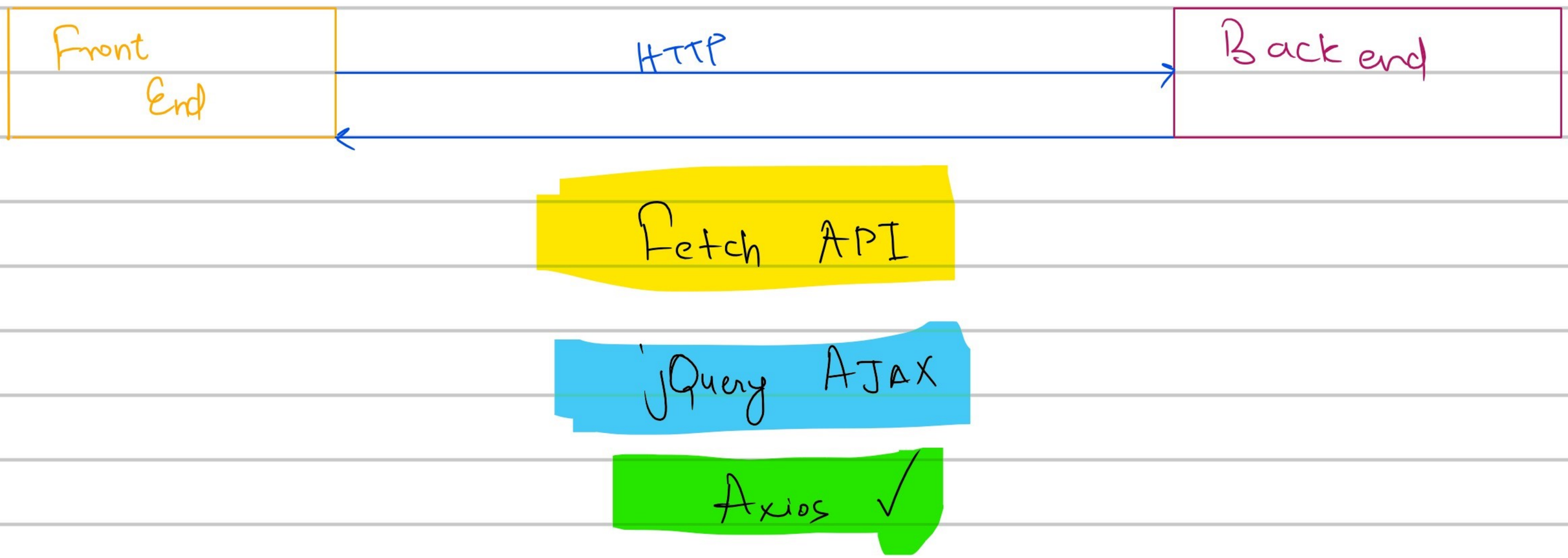
route parameters

* useEffect Hook *

↳ lets you perform side effects in functional components

Class	Function
componentDidMount()	
componentDidUpdate()	useEffect(function)
componentWillUnmount()	

HTTP Calls



* Promises *

→ holds eventual result of an asynchronous operation.

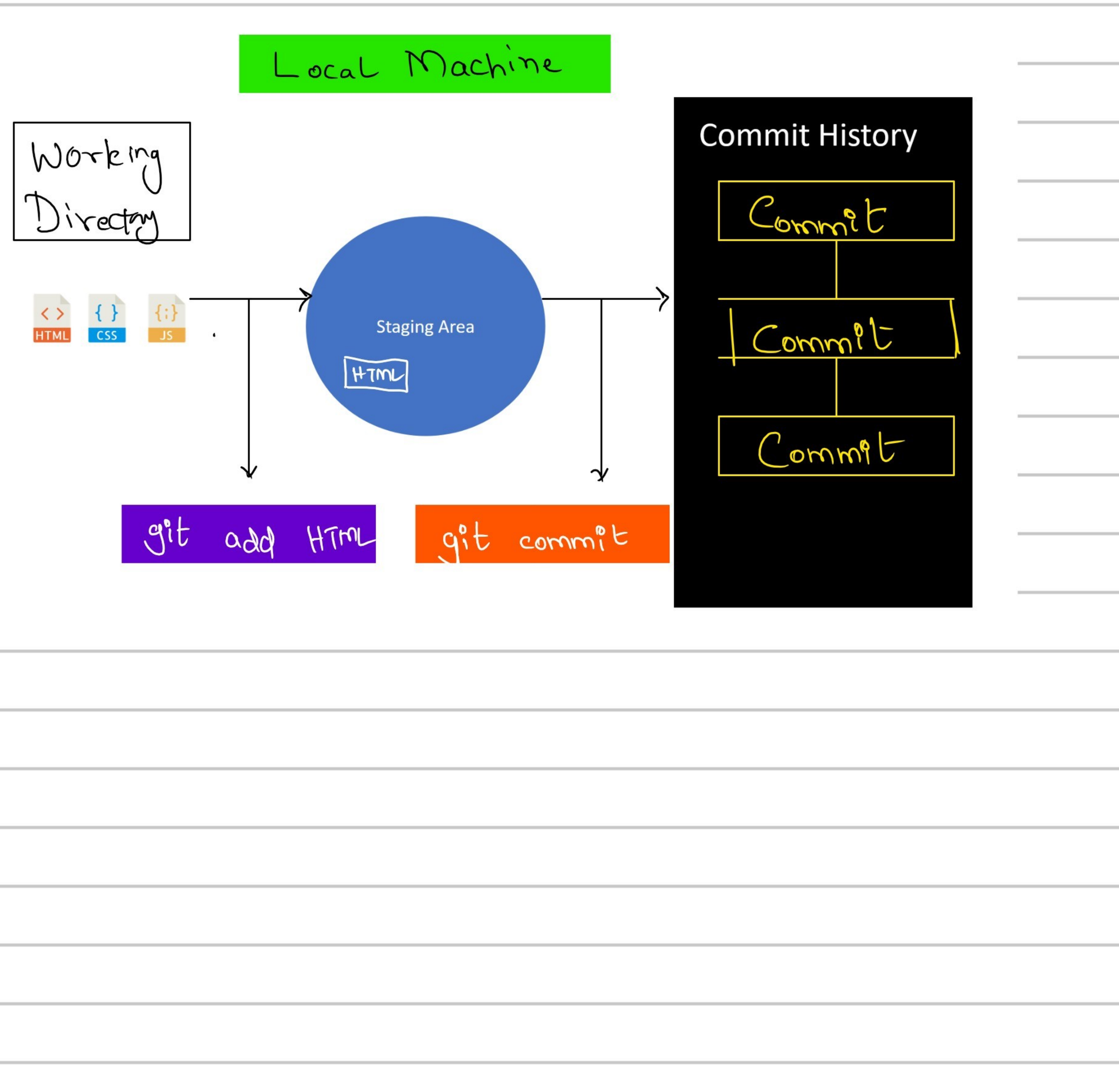
Promise can in 3 states

Pending --- async operation → Fulfilled
                           → Rejected

Git

= Source Code Management

How Git works



Local Machine

Working Directory

HTML CSS JS

Staging Area

HTML

git add HTML

git commit

Commit History

Commit

Commit

Commit

Remote Server

Github

Bitbucket

AWS Code Commit

git push

Tom
Singapore

git push

GitHub

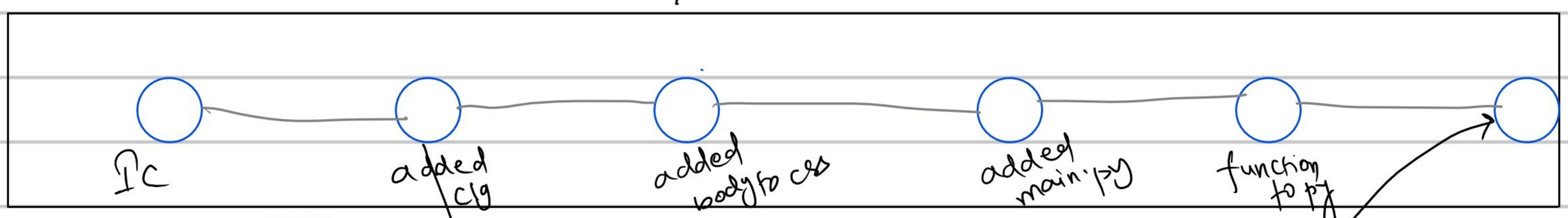
Gitlab

Elizabeth
UK

git pull

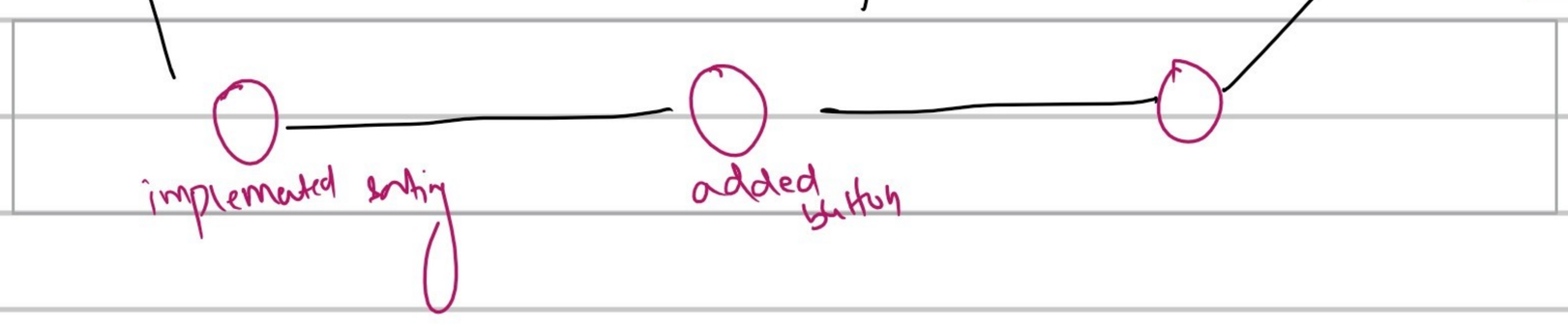
Branching

master/main



Creating a branch

feature-a Branch



merging