

Java-8

=> Lambdas

Functional Programming

those feature that define functional programming

streams

Executor (Future)

Concurrency Collection

Style :

Traditional : Imperative

(HOW)

#exposing the steps how to perform an operation

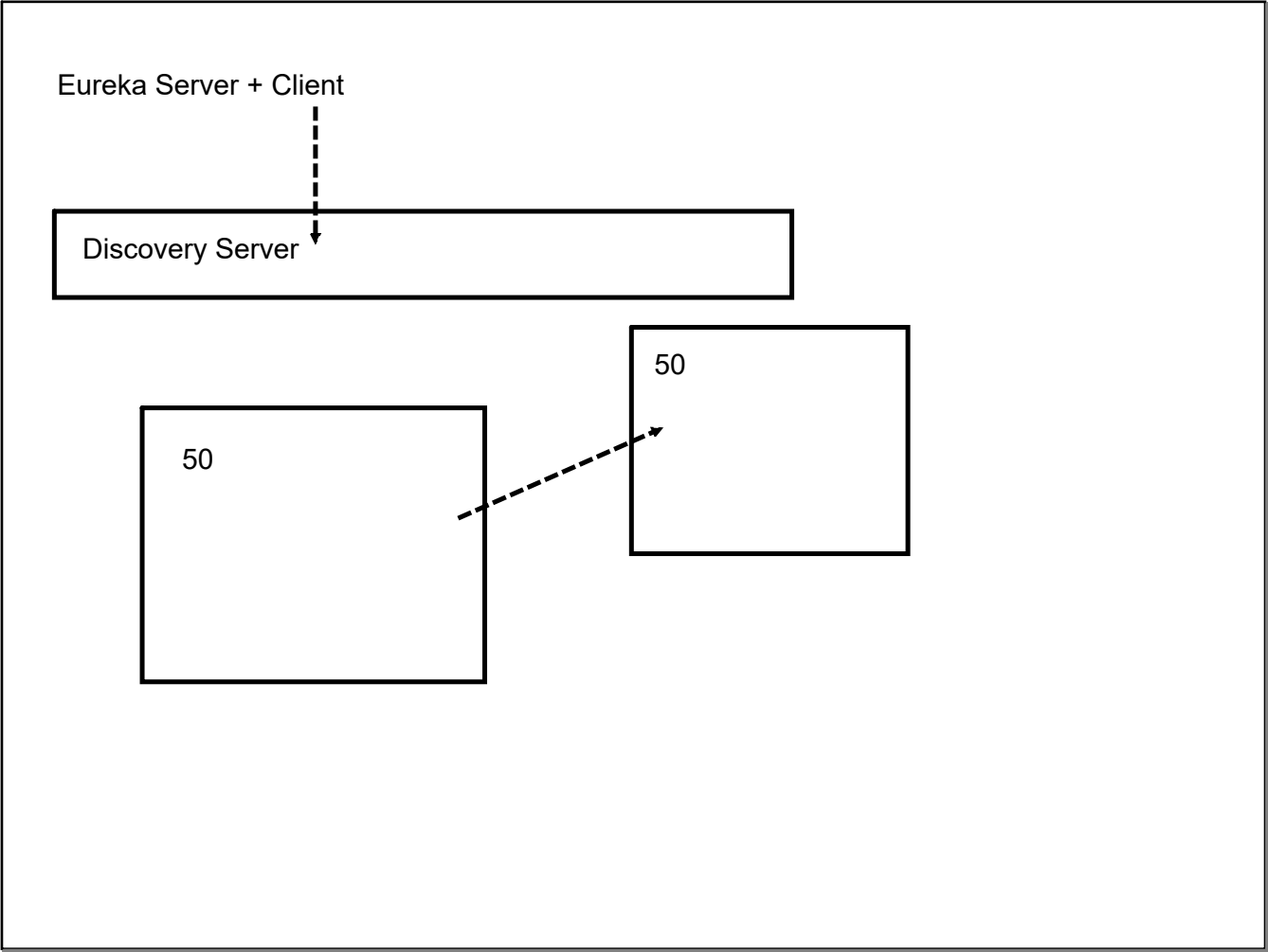
embrace object mutability (not in sync with concurrency)

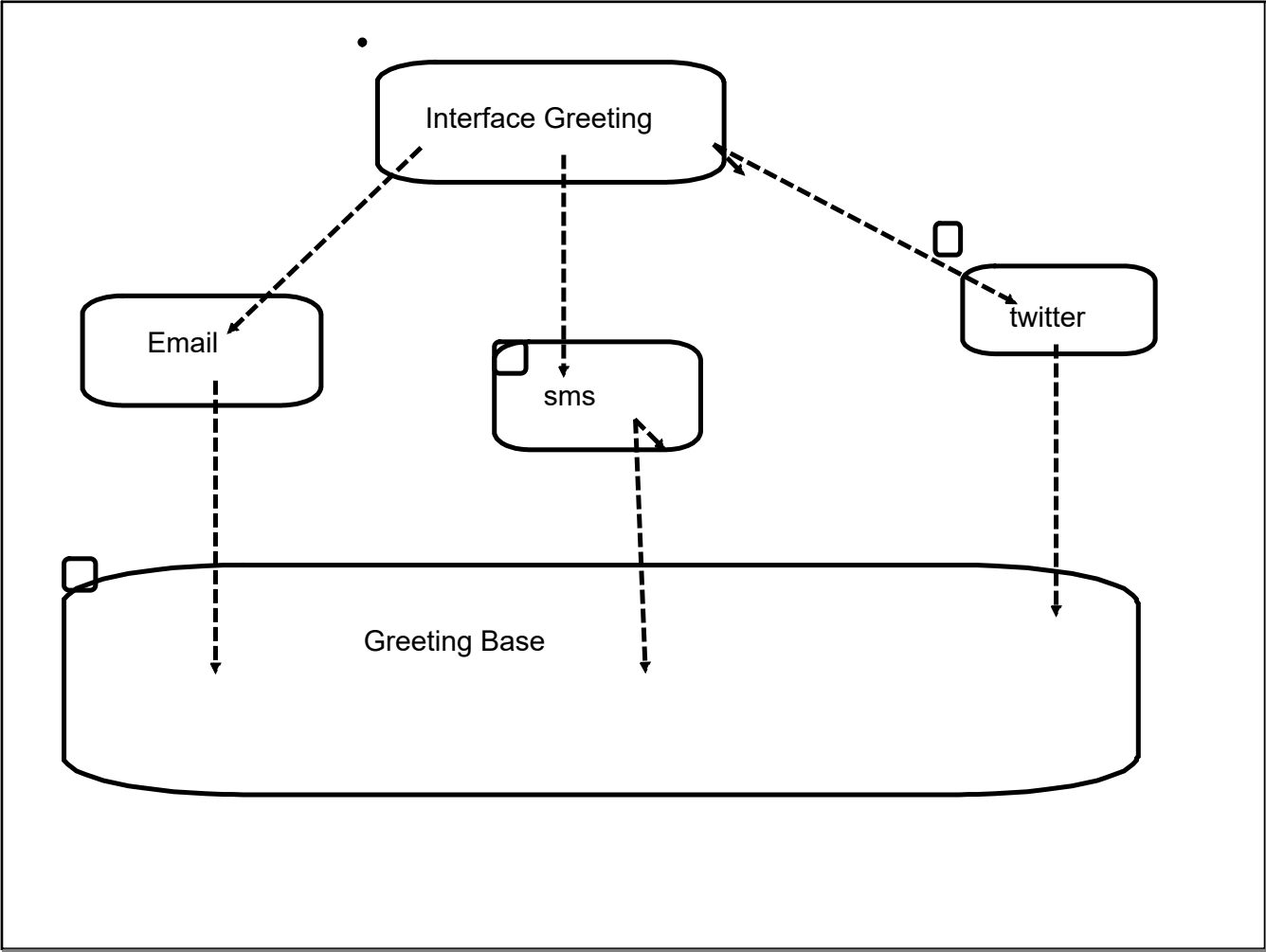
Functional : Declarative

(What) : result

immutability

Analogous SQL



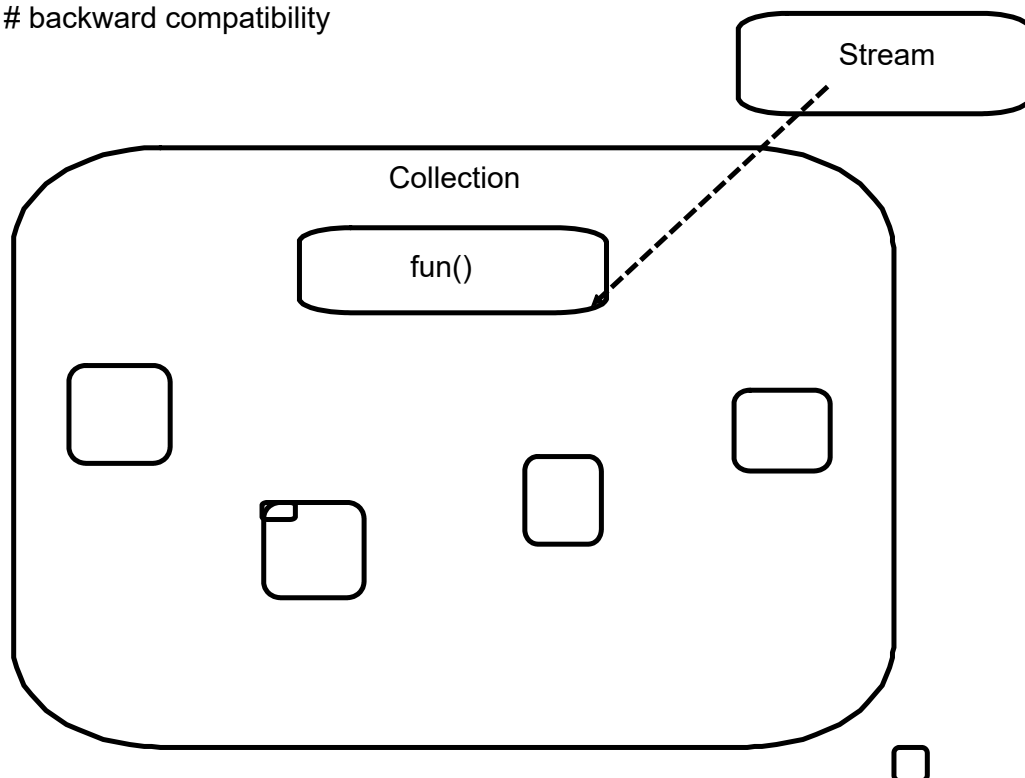


Interface :

```
# default method ( definition )
```

multiple implementations can be inheritance

backward compatibility



Escape from OOPs

independent Functions (not wrapped inside an object)

Relationship between interface and function

1. interface must have only one abstract method (any number of default/static) :

Functional Interface : Annotation `@FunctionalInterface`

2. single method signature must match with function implementation

Lambda expression

```
(<arg1>,<arg2>) -> {  
}
```

```
arg1 -> {  
}
```

```
() -> {  
}
```

```
(<arg1>) -> <return> <single instruction>
```

```
(a,b) -> <return>a+b;
```

```
(a,b) -> {  
    return a+b;  
}
```

Pre defined functional interfaces

=> Runnable

=> Comparator

Explicit Functional Interface

Consumer

void accept(<>);

DoubleConsumer() // specialized implementations on primitive

BiConsumer

void accept(<>, <>);

Predicate (test)

boolean test(<>)

Supplier

<> get()

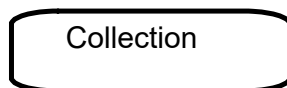
Function

<> apply(<>)

Stream :

not a data structure

immutable (Thread safe)



Stream

p / for
conveyer belt

Terminal

Stream : LAZY Processes

Stream can have 2 type of activities

intermediate activities (filter/map/flat map)

Terminal activity : terminate/close the stream

forEach()

collect()

IF terminal activity is not present : stream will not initiate

groupingBy(<return> Function(student))

(Stream of) Multiple collection
into (Stream of)single collection

return value : would become a group

Transforms

y map(x)

flatMap() : Collection into stream

map:

["", ""]

["", " ", ""]

["", ""]

flat map

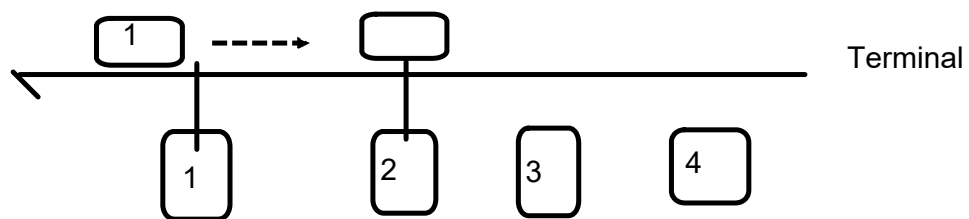
["", " ", " ", " ", " ", " ", " ", " ", " "]

return type fixed : stream of data passed as argument

Stream :

Sequential Stream

Parallel Stream



Parallel Streaming not commended if working on external mutable data (not thread safe)

Activities that are inherently complex

Binary Operator : variant Function

y Function(x) : x and y can be of different type

z BinaryOperator(x,y) : x,y,z : must be of same type

Multithreading :

interleaved (Threaded Multitasking)

1. Multiple activities waiting for I/O : that time can be used by tasks
2. Multi-core architecture of micro-processor

Base Interface :

Runnable (run)

Implementation:

Core Functionality of Multithreading (Thread)

inheriting Runnable

inheriting Thread

Traditional approach:

create, execute, manage

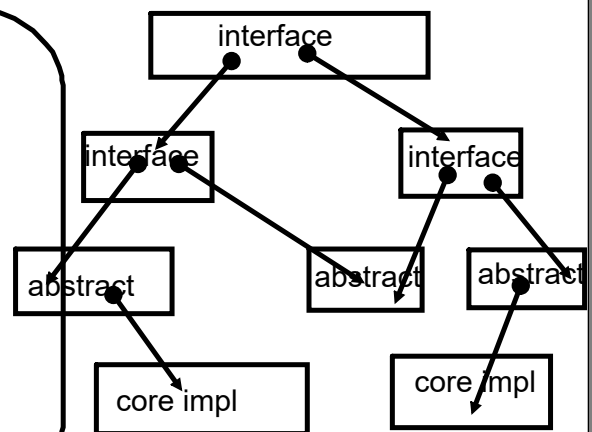
Thread per task

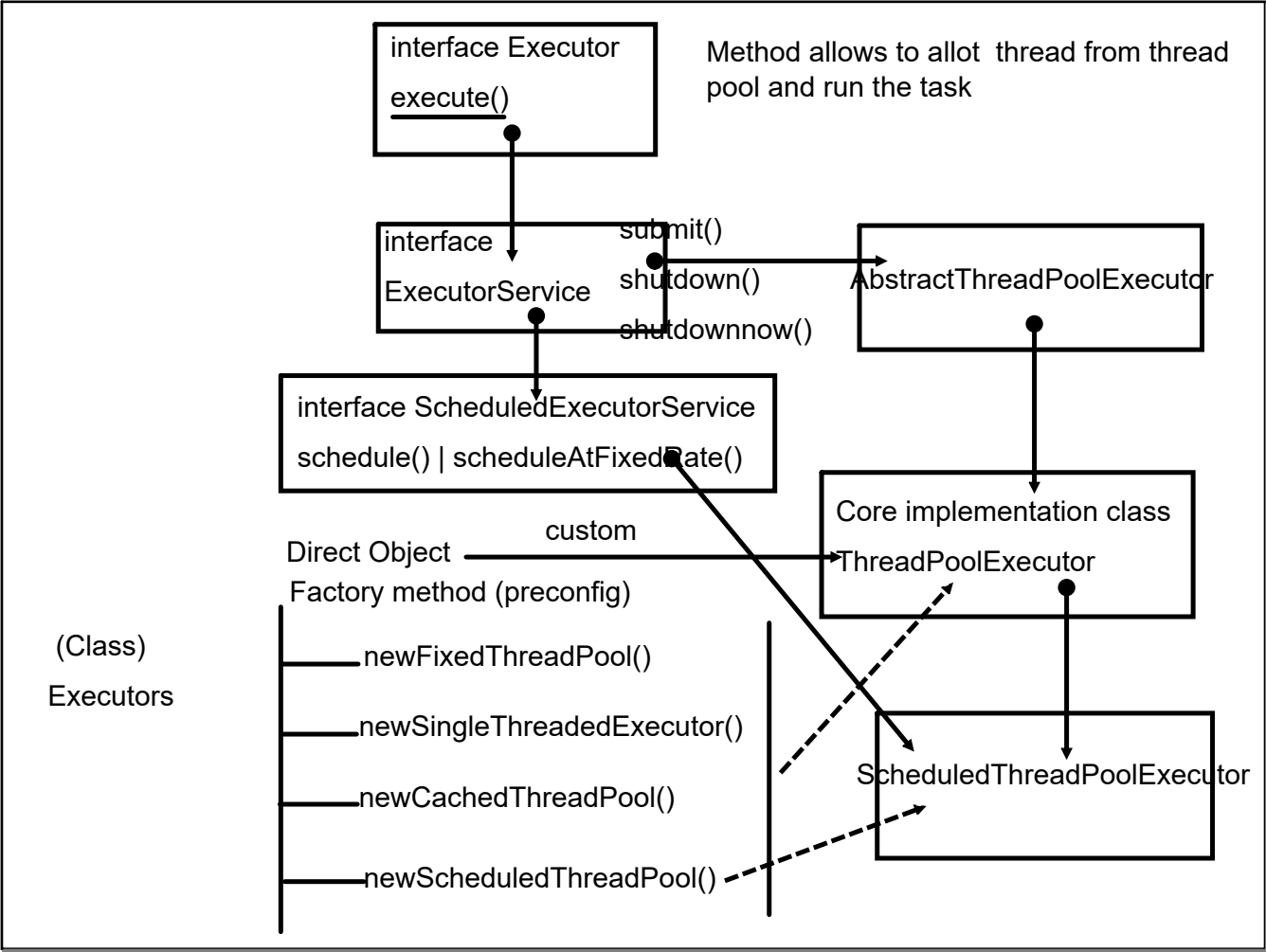
Not possible to keep a track of number of thread

Executor framework :

do all management + improve in performance

ThreadPool : create a predefined pool of thread





Need to create instance of ThreadPoolExecutor

FixedThreadPool (number of thread are predefined(extra task allotted will added to queue)

CustomThreadPoolExecutor

<corePoolSize> : number of threads to always keep even if they are idle (2)

<maxPoolSize>: max no of thread (5)

<keepAliveTime> : time to wait before idle thread gets removed/released from thread pool

<TimeUnit> :

<queue capacity>: capacity of queue

<RejectedHandler> : what to do if a task is rejected from queue

SingleThreadExecutor()

FixedThreadPool(1)

can change the thread capacity

CachedThreadPool() : Unbounded ThreadPool : Max Integer Val

if demand decreases : can tear down thread

default keep alive time : 1 min

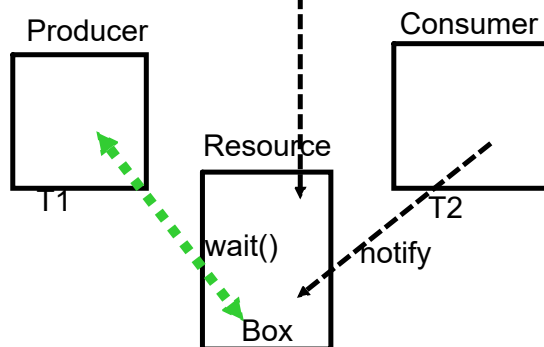
ScheduleThreadPool()

Thread returning some value

Special Interface Callable (call())

ThreadPool can work on Callable

wait/notify/notifyAll() : Object



ExecutorCompletionService

: will going to get results in order of completion of task

Future : blocking

CompletableFuture <callback : logic to follow when task is done>

Functional interfaces

Runnable

Callable

=> Supplier

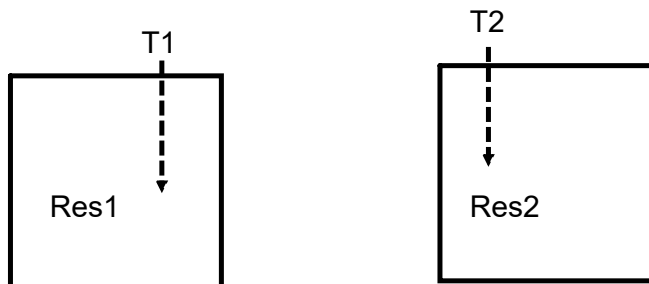
the method to associate a callback function

1. thenApply(Function); // transform
2. thenAccept(Consumer); // consuming and using

CompletableFuture by default uses the inbuilt thread pool

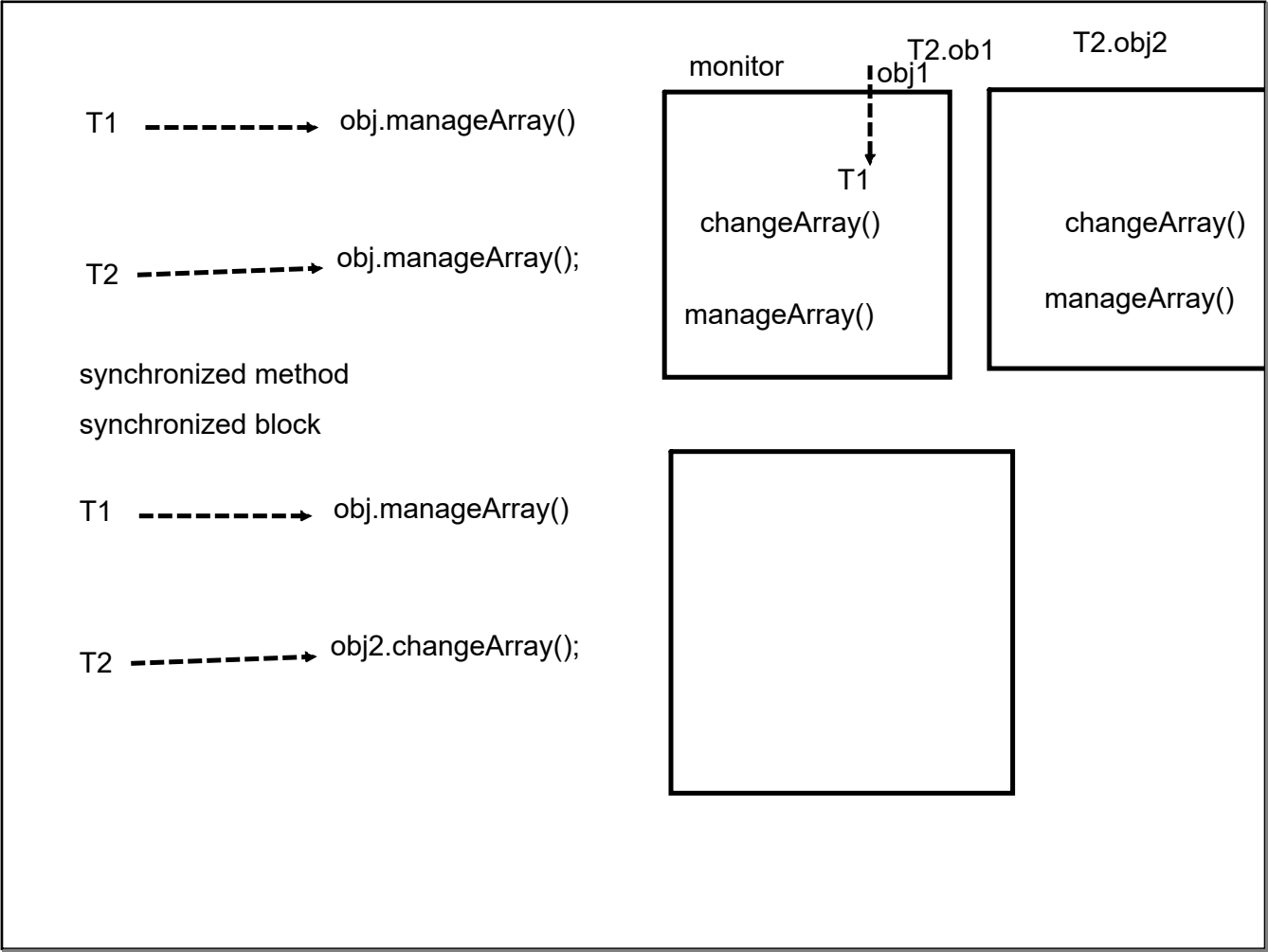
`ForkJoinPool.commonPool();`

Executor ThreadPool



Common Resource Shared among multiple threads (Thread safe)

Resolve Data inconsistency



locking :

=>wide spectrum locking : (synchronized...)

=>granular locking

java.util.concurrent.

API : Granular locking on resources

Collection API

1 .Traditional : 2

1. HashTable

2. Vector

2. To get a Thread safe variant of those class

 Collections.concurrentList();
all methods are sync

Atomic operation : single CPU instruction

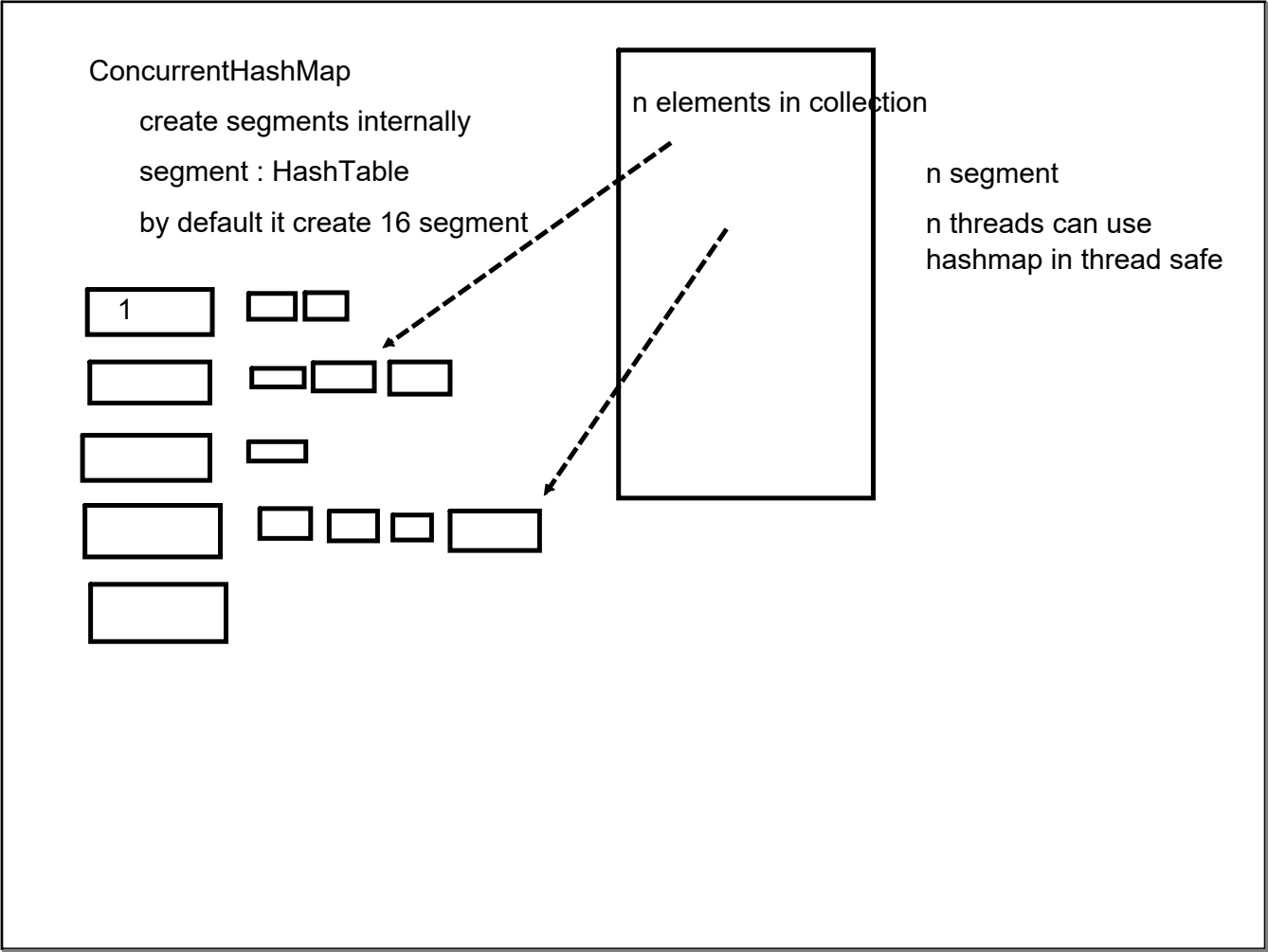
n=10; // Thread safe operations

assignment long/double are non-atomic

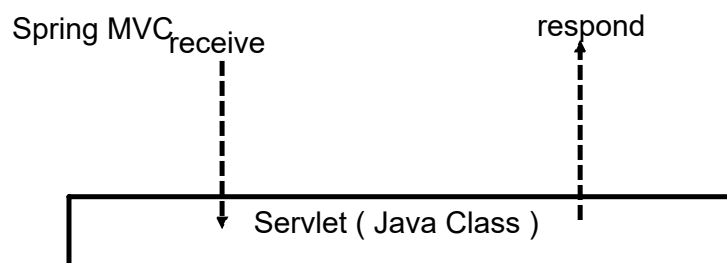
Concurrent API : Focus on granular locking

Provides Atomic Variant of type : allow to convert non-atomic activities into atomic

multiple approach for ThreadSafety along with high level of concurrency



Servlet Technology



How to define java class as Servlet

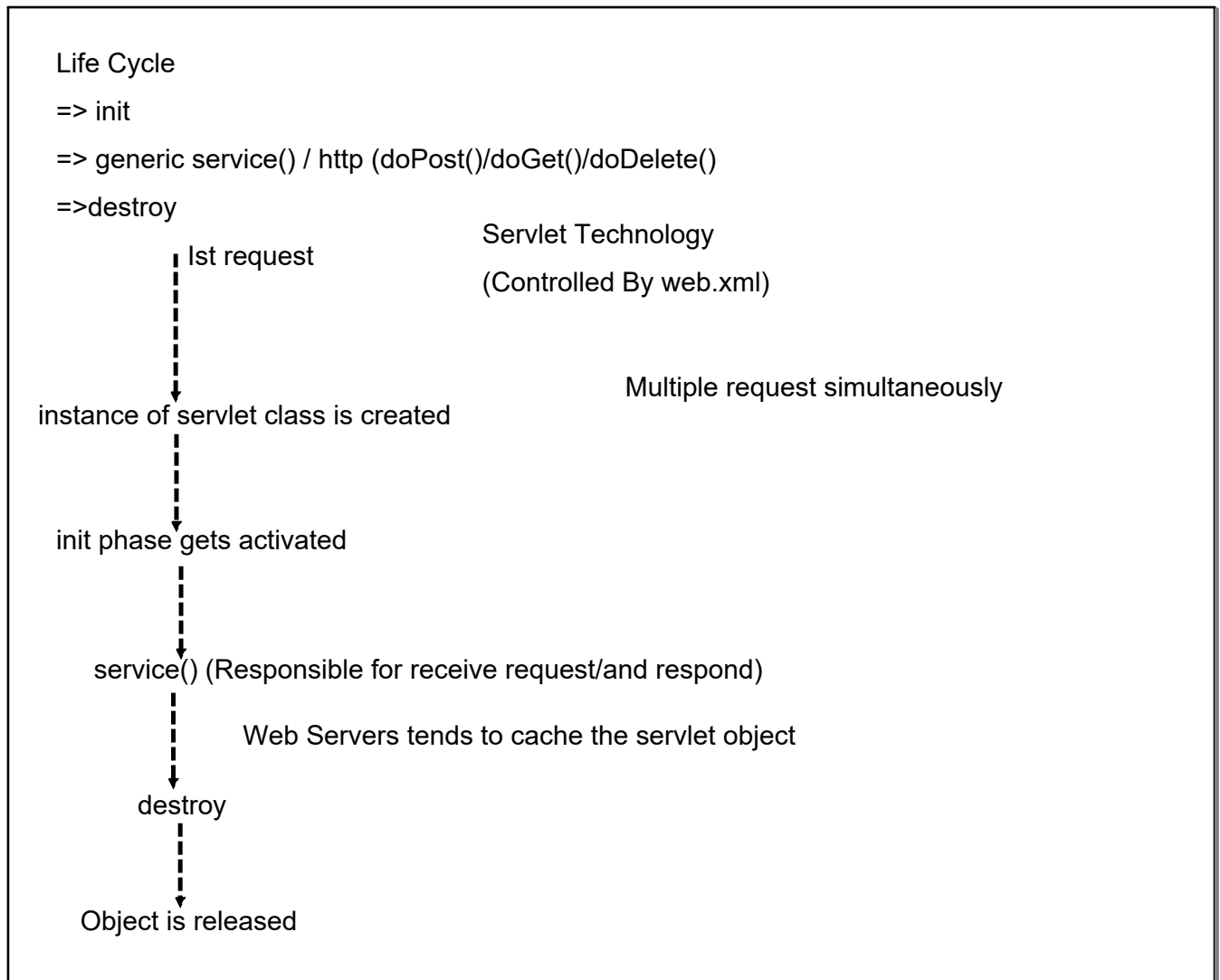
Extends

HttpServlet/GenericServlet

GenericServlet : does not classifies between various HTTP Verbs

HttpServlet : can identify

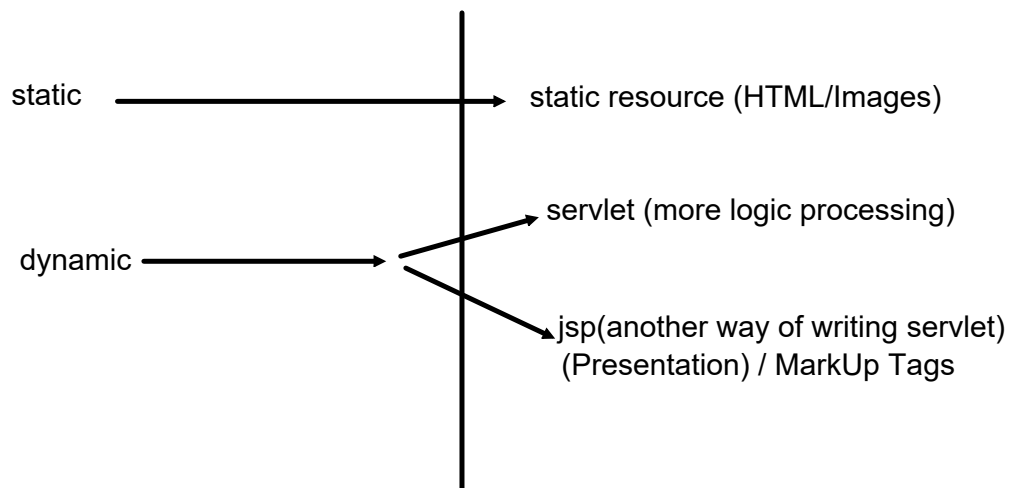
GET/POST/PUT/DELETE/PATCH



service method and variant

have an access over HttpRequest/HttpResponse object

JSP : another way of writing servlet

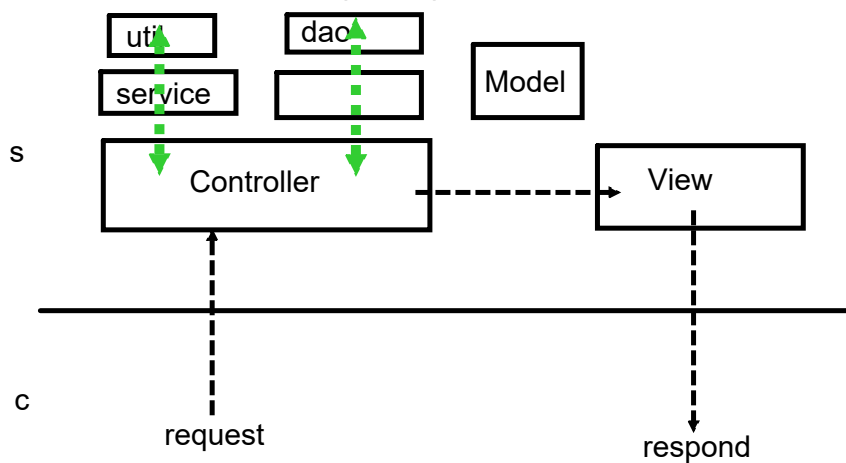


Spring uses Servlet Technology:

But provides a high level abstraction over complexities/ boilerplate req / config and enhances the separation of concerns

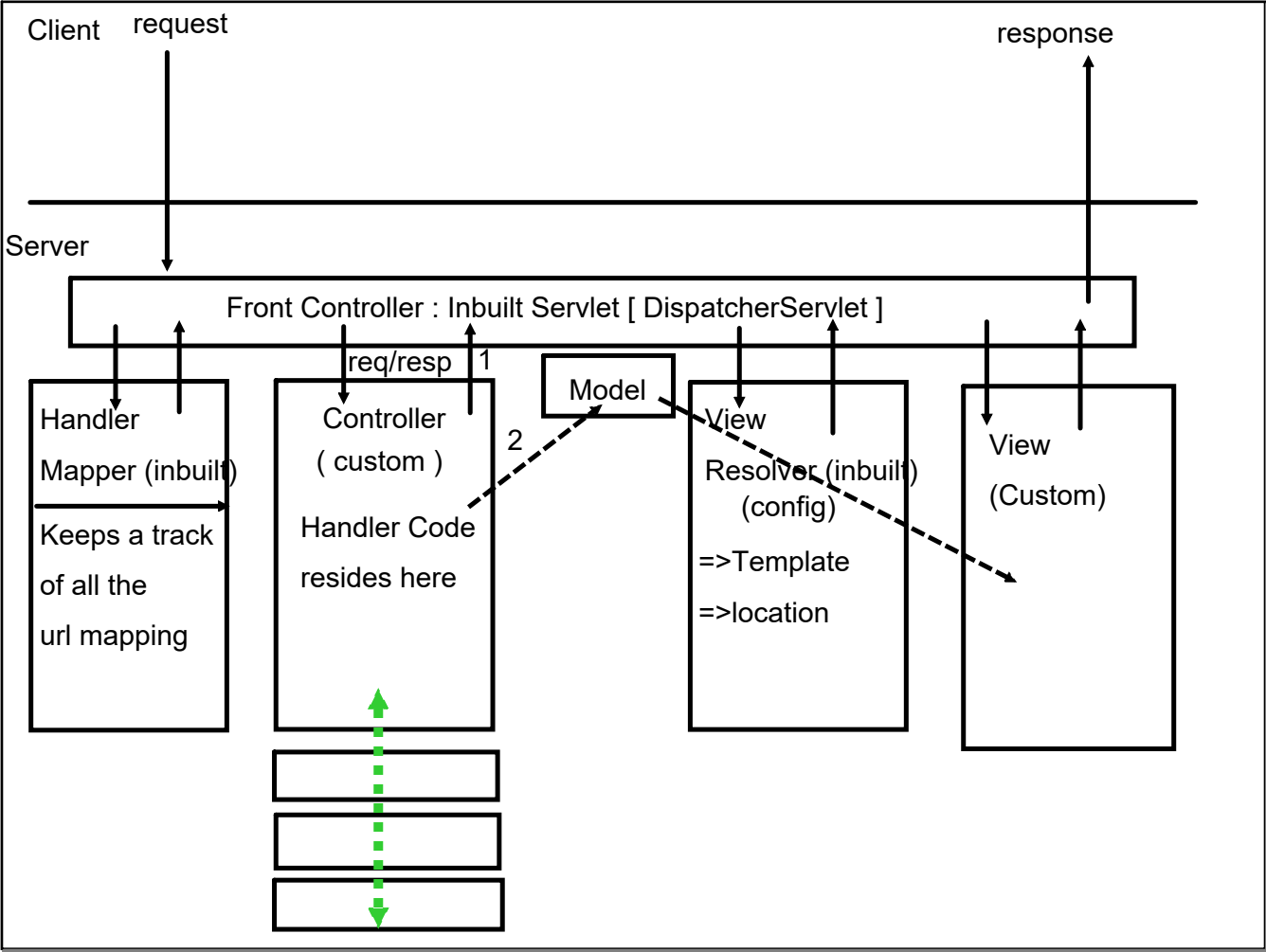
MVC architecture

Controller : to receive request / process it



Servlet

service method as task :
assign it to thread



we need to register your app resources (servlet spec)

Servlet :

need to register

registration can also be done using annotation

Register DispatcherServlet

Controller : "index"

create a complete path

Config of Spring in place

xml file

java

Need Spring config to connect with DS

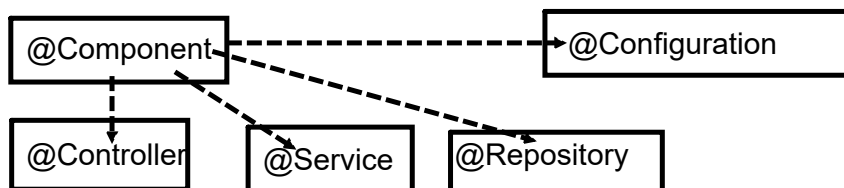
xml : <servlet-name>-servlet.xml

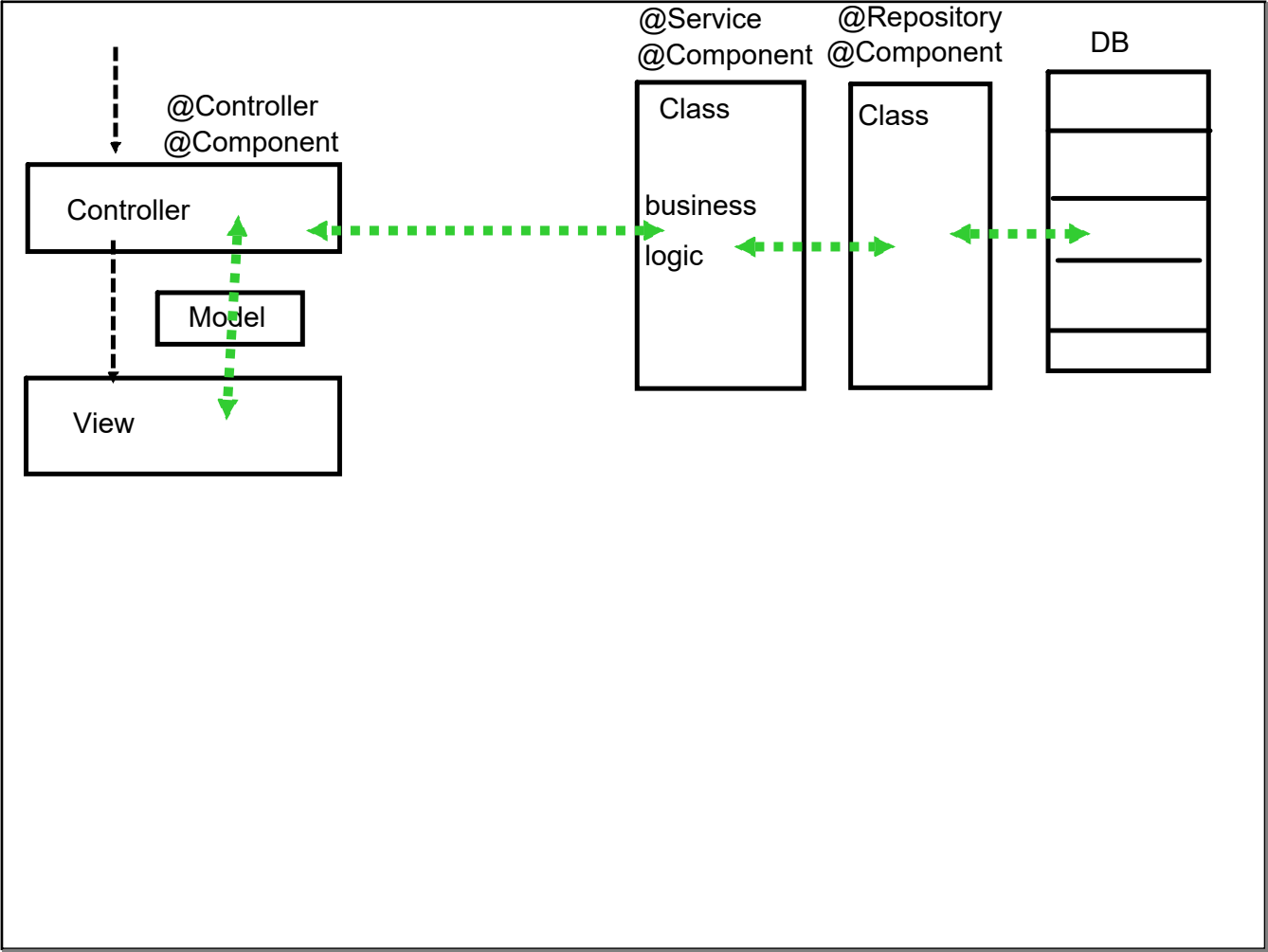
View Resolver : location + template (jsp+jstl) [extension]

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
  <property name="prefix" value="/WEB-INF/views/" />  
  <property name="suffix" value=".jsp"/>  
</bean>
```

Controller return : "index"

↓
/WEB-INF/views/index.jsp



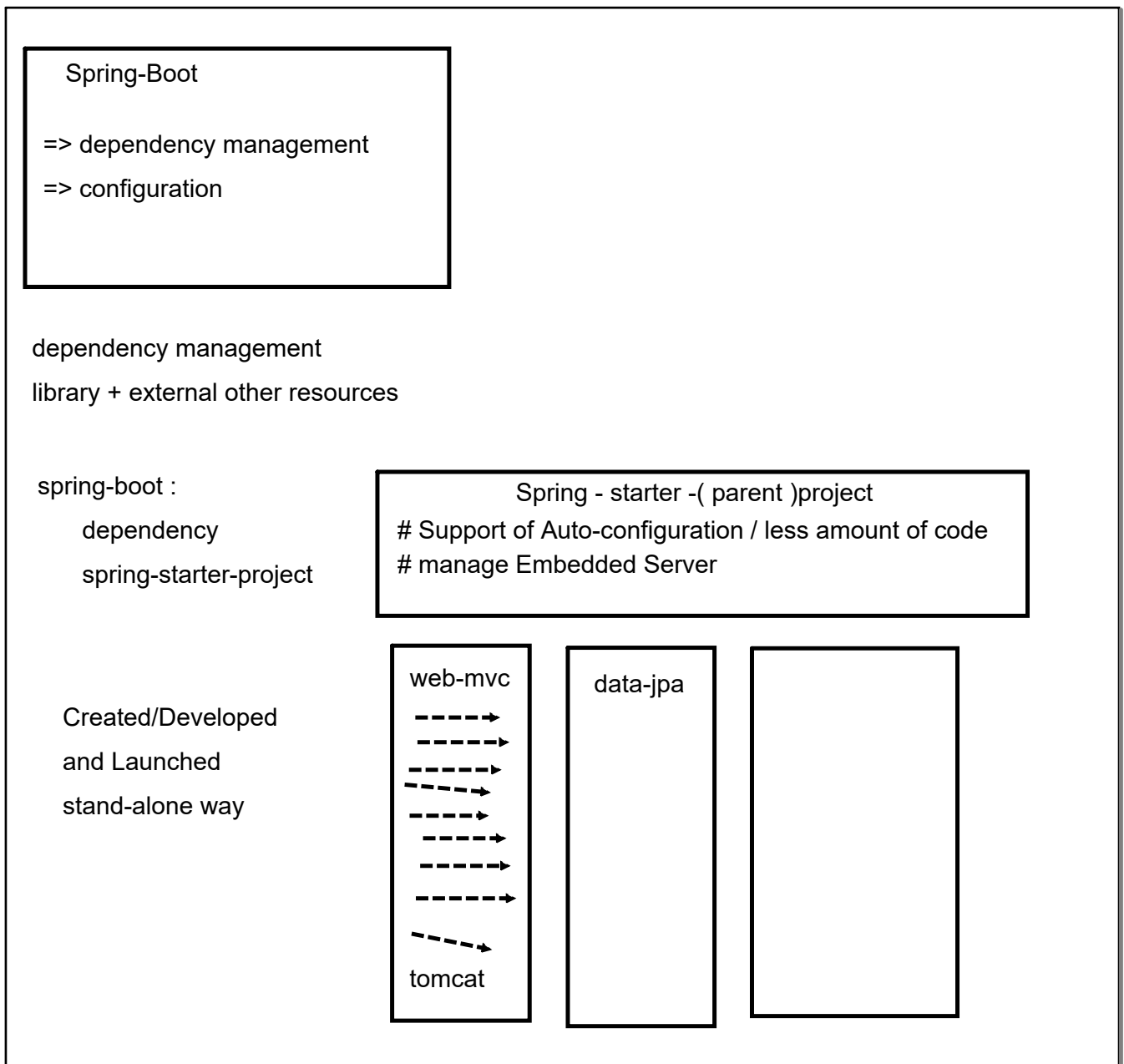


web.xml : ~ java config class

dispatcher-servlet.xml : ~ java config class

1. alternate for packaging : maven war plugin

Spring provides an inbuilt class to register DS



start.spring.io

maven cli

maven command

Configuration

- # Spring boot Annotation

- # Dependency

- # Customization : special file application.properties

 - key=value

key : predefined keys from different spring projects

- : possible values

- : custom keys/values

spring : yaml

- : heirarchy

- : application.yaml

Spring Boot Annotation

curated list of multiple annotation

EnableAutoConfiguration

tracking the dependencies

based on dependencies added:

add default config

expose the key

eg:

maven-web : Spring mvc:

DS servlet

spring-security

add default security

expose username/password

tracking the properties files

looks for custom key-values pairs

defined in config-file

cli : key-values

mvc application

controller

view

pre-configured to use thymeleaf

View pages :

View Templates

Jsp-jstl

Thymeleaf

Mustache

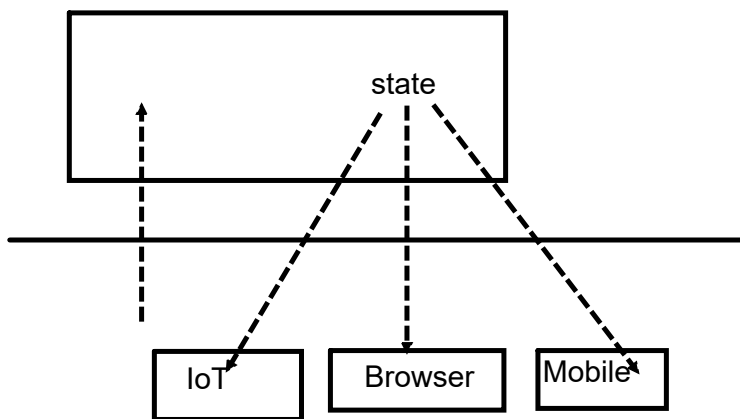
FreeMarker

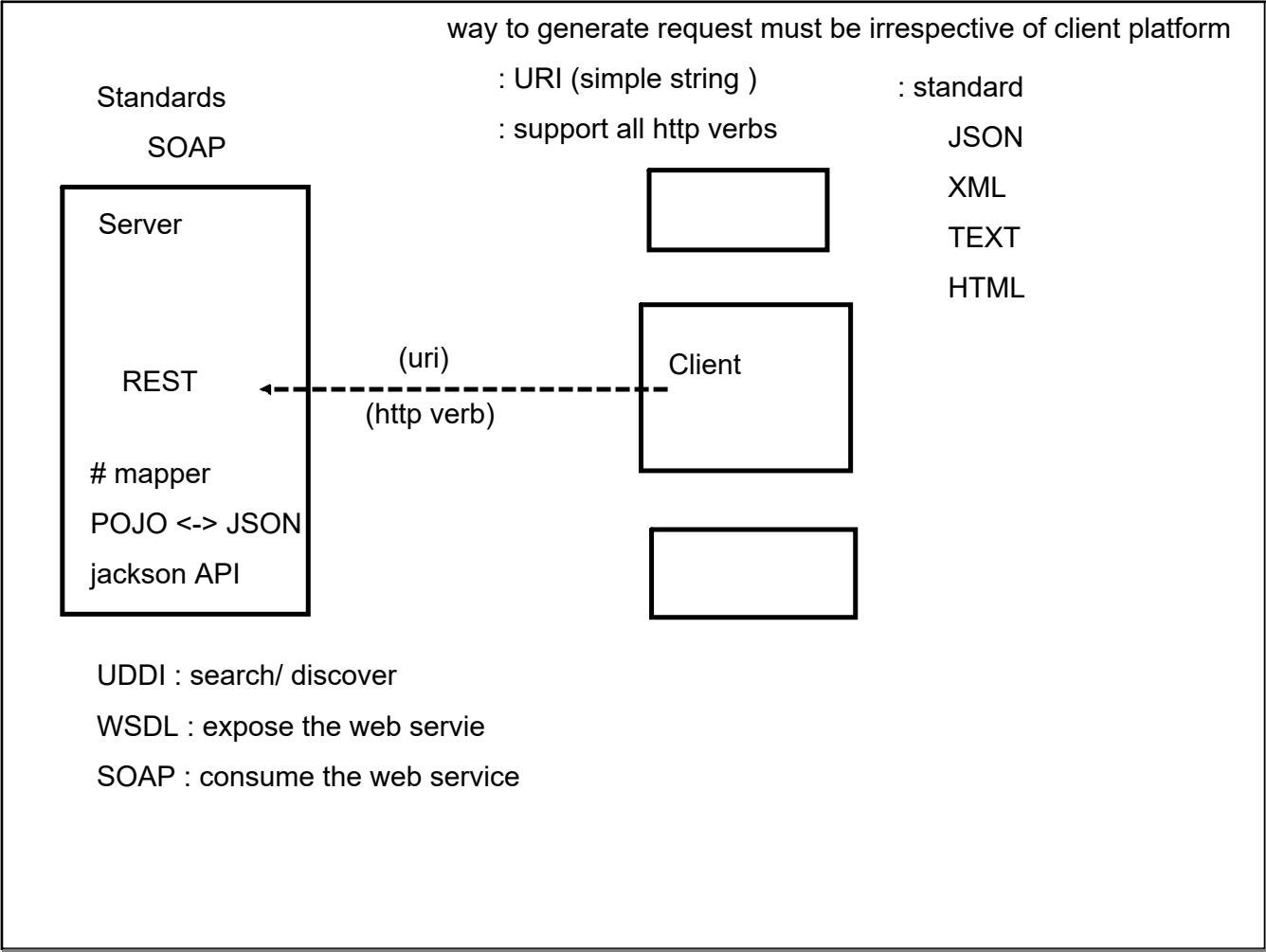
Tile

Velocity

Rest Based service

exposing server based info as Rest Service





@RestController : interconversion take care of

client intention

GET : data retrieval

POST : add new data

PUT : edition

DELETE : delete

Student /student

/getAll

Employee /employee

/getAll

@RestController : auto needs mapper in classpath (jackson)

@Controller

add mapper to classpath

produce

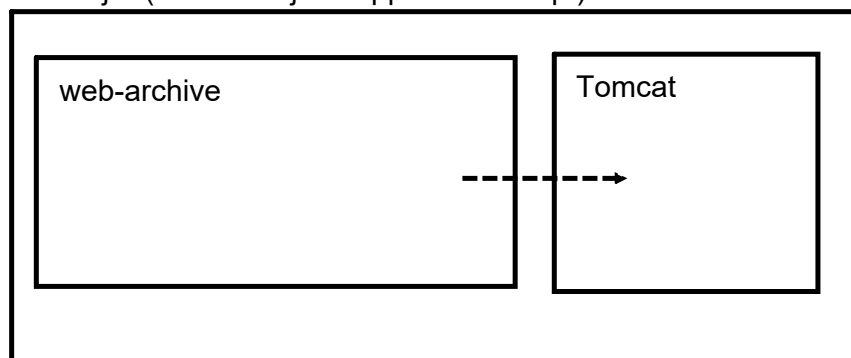
consume

@ResponseBody : content to be shared directly to client

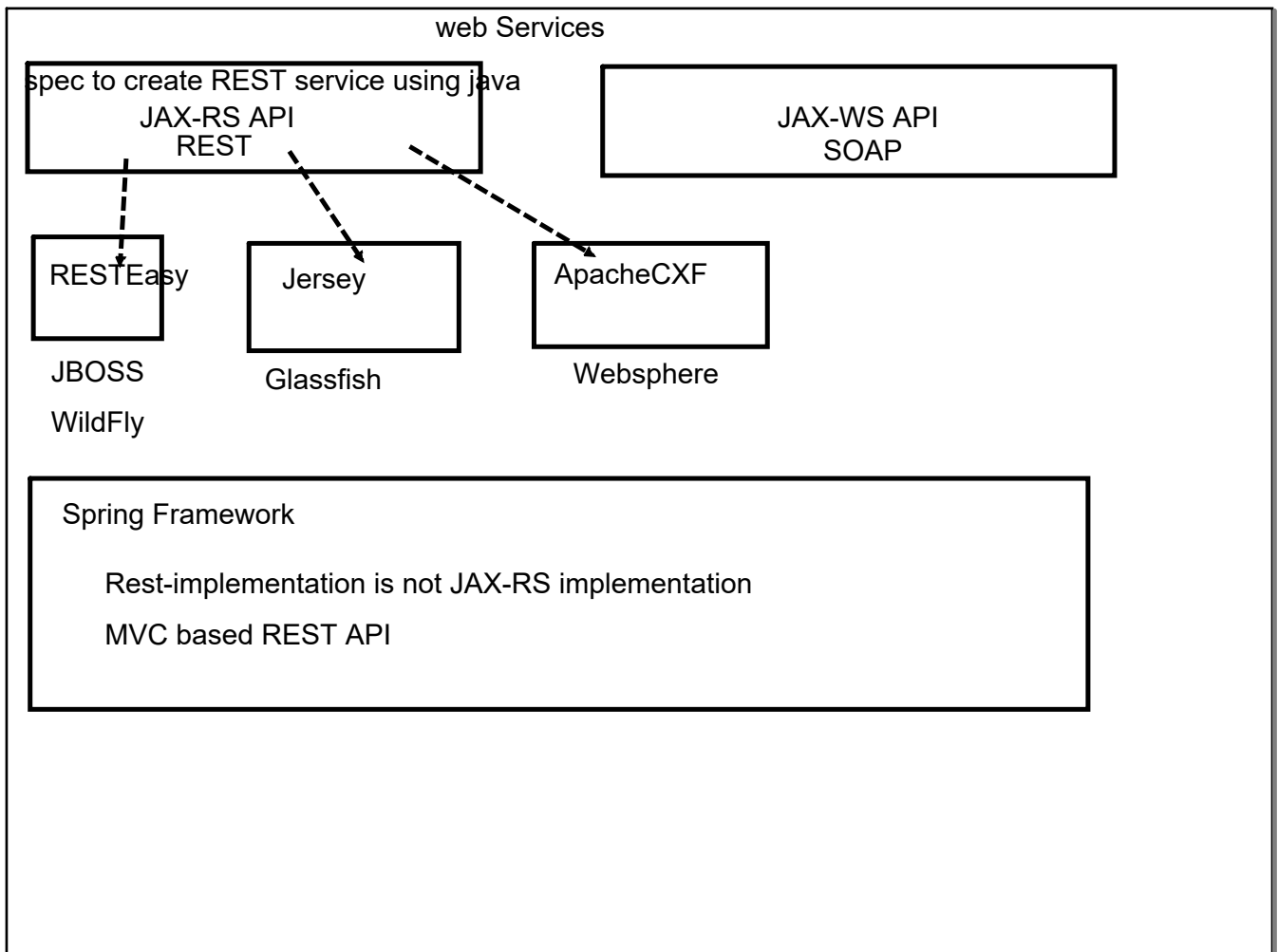
1.launch tomcat

2. deploy web-archive over tomcat

jar (additional java app build on top)



dev-tools: live reloading of server



actuator : exposes rest endpoint

Microservice architecture implements

Dividing a single large sized monolith application into
multiple smaller (independent) application

microservices : responsible to expose a particular service

- DataDriven/Rest based

- Stateless

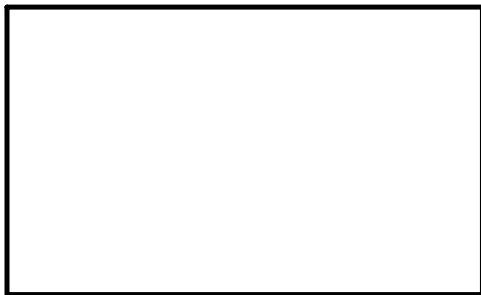
Service Oriented Architecture : SOA :

Microservice : + technology/approach/design pattern

Monolith issues

involve light wight VS for deploying service components

Deployment :



Multi-Technology service component

DB : ideally must be using independent DB

Scaling : individual service comp

Robust in implementation

Design Guideline : MS (12 factor)

Design Pattern

Lightweight : concern/runtimes/data exchanging

Reactive : highly concurrent/longer processing

Stateless: scale better

Atomic : core design principle

Externalized config : config server

Consistent : style

Resilient : eliminate bottleneck

Good Citizens : expose usage statistics

well versioned :

Design Pattern:

Decompositions :

a) business capabilities

business-oriented rather than technical

b) sub-domain (technical)

domain class (parent/God classes)

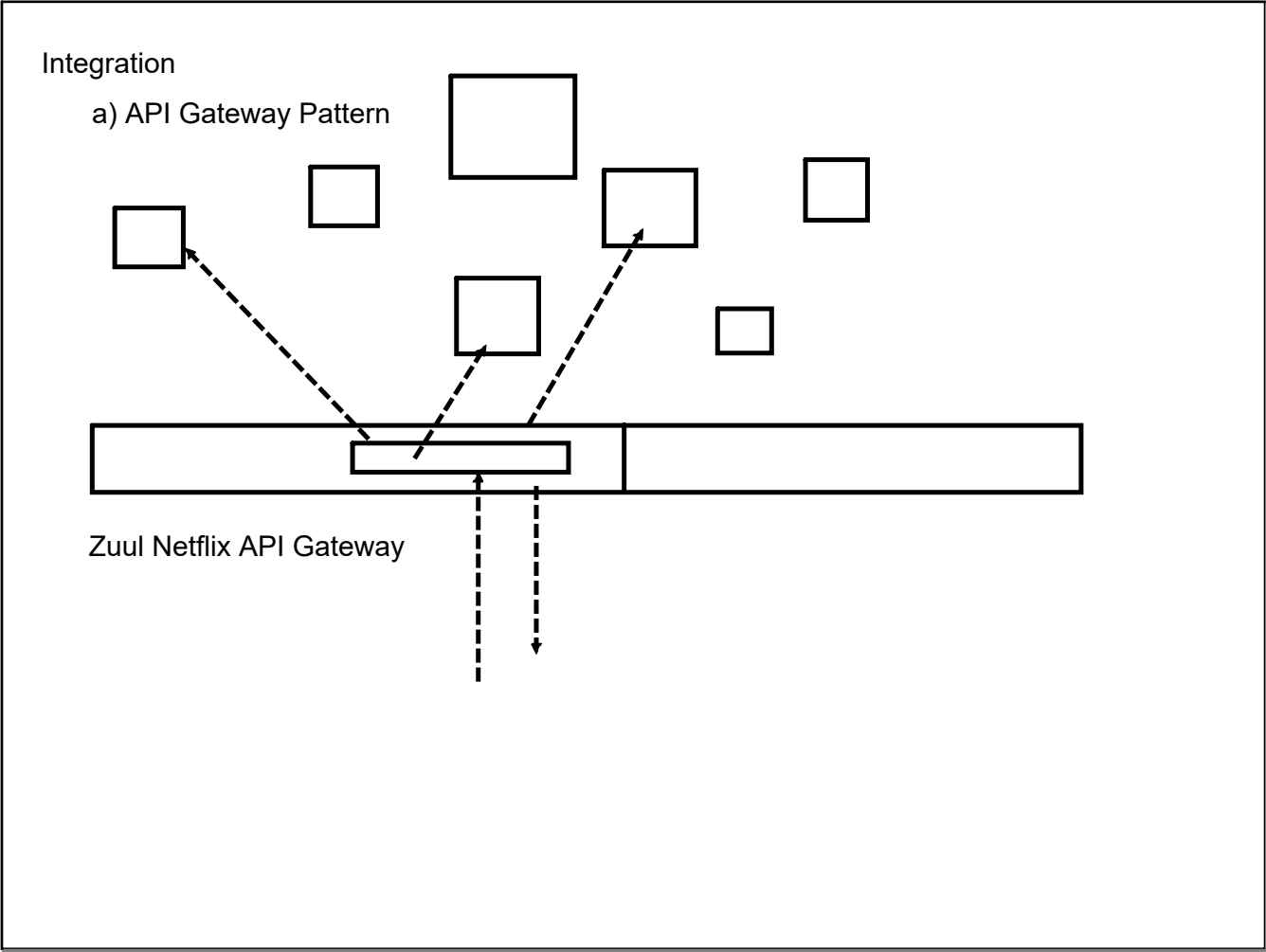
DDD : bounded context

sub-domains : BC with parent model

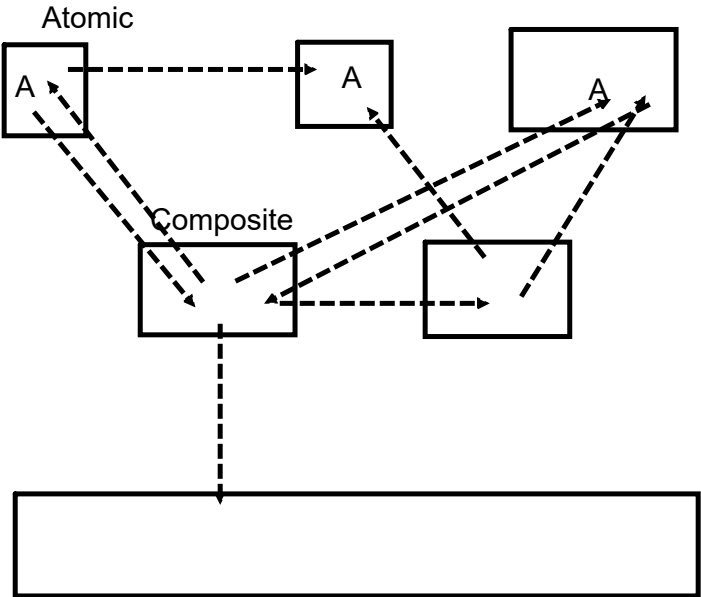
c) Strangler patterns

brownfield : converting monolith into MS

refactoring smaller req...



b) Aggregator Pattern



c) Client-side UI Composition

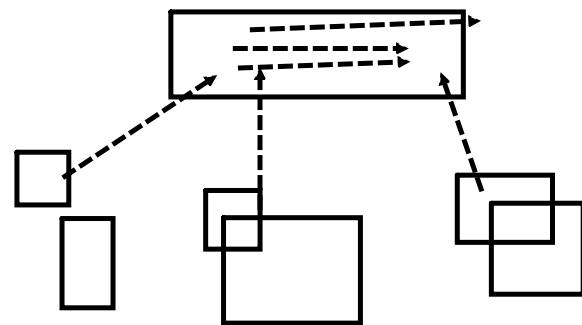
SPA : UI compositions

DataBase :

a) Ideal Pattern : Database Per Service

b) Shared Database :

2-3 MS (DDD)

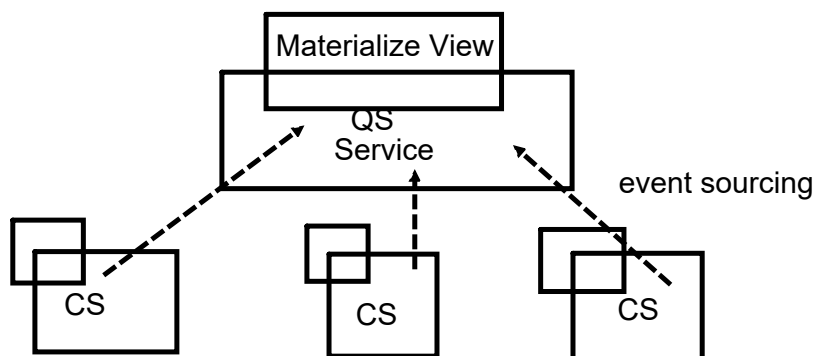


c) CQRS

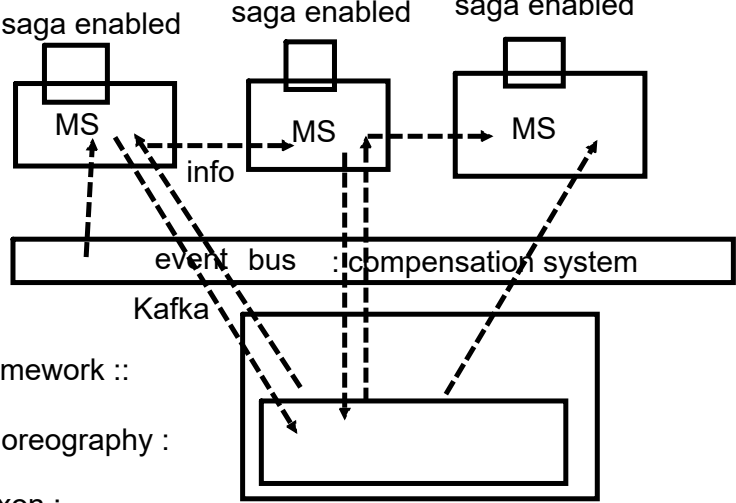
Command Query Responsibility Segregation

Command Side (create/update/delete)

Query Side (retrieve)



SAGA PATterns:



Observability PAttern

a) Log Aggregation:

Centralized Logging pattern in place

track the log on request basis,

search

analysis

triggers alert

PCF : Pivotal Cloud Foundry

AWS Cloud Watch

b) Performance based

Centralized Metric service

push/pull model

=>NewRelics

=>Prometheus

c) Distributed Tracing

system to track a request end-to-end

request id

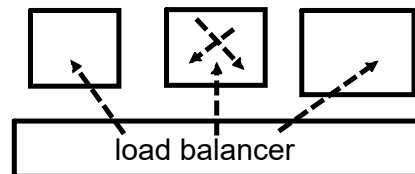
Zipkin Server

Spring Cloud Sleuth

d) Health Check

actuators /health :

Ribbon



Cross-Cutting Concerns

a) External Configuration

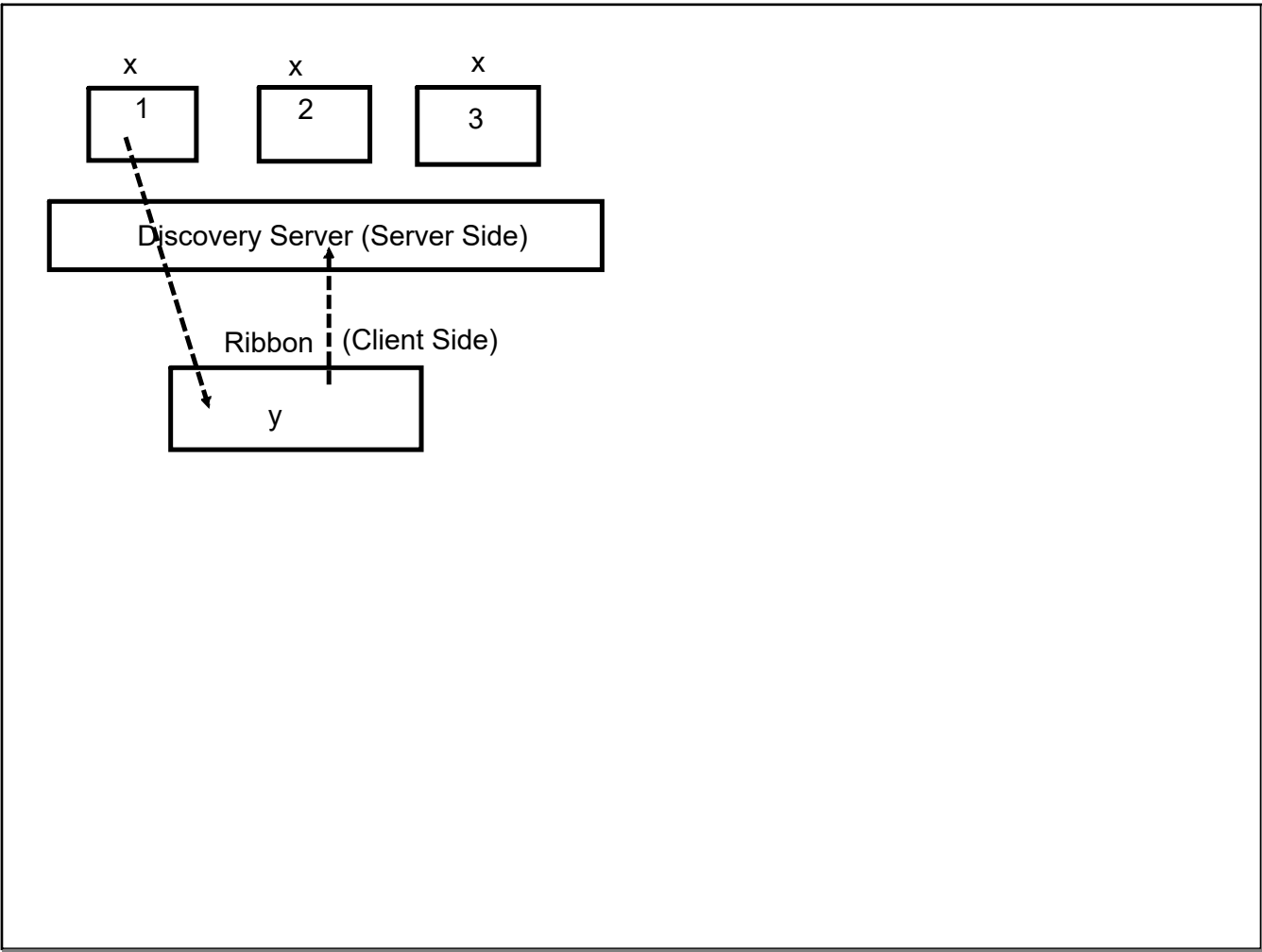
Spring Cloud Config Server

b) Service Discovery Pattern

all service shall register with registry system

Netflix Eureka Server

AWS ALB



c) Circuit Breaker Pattern

threshold

default response

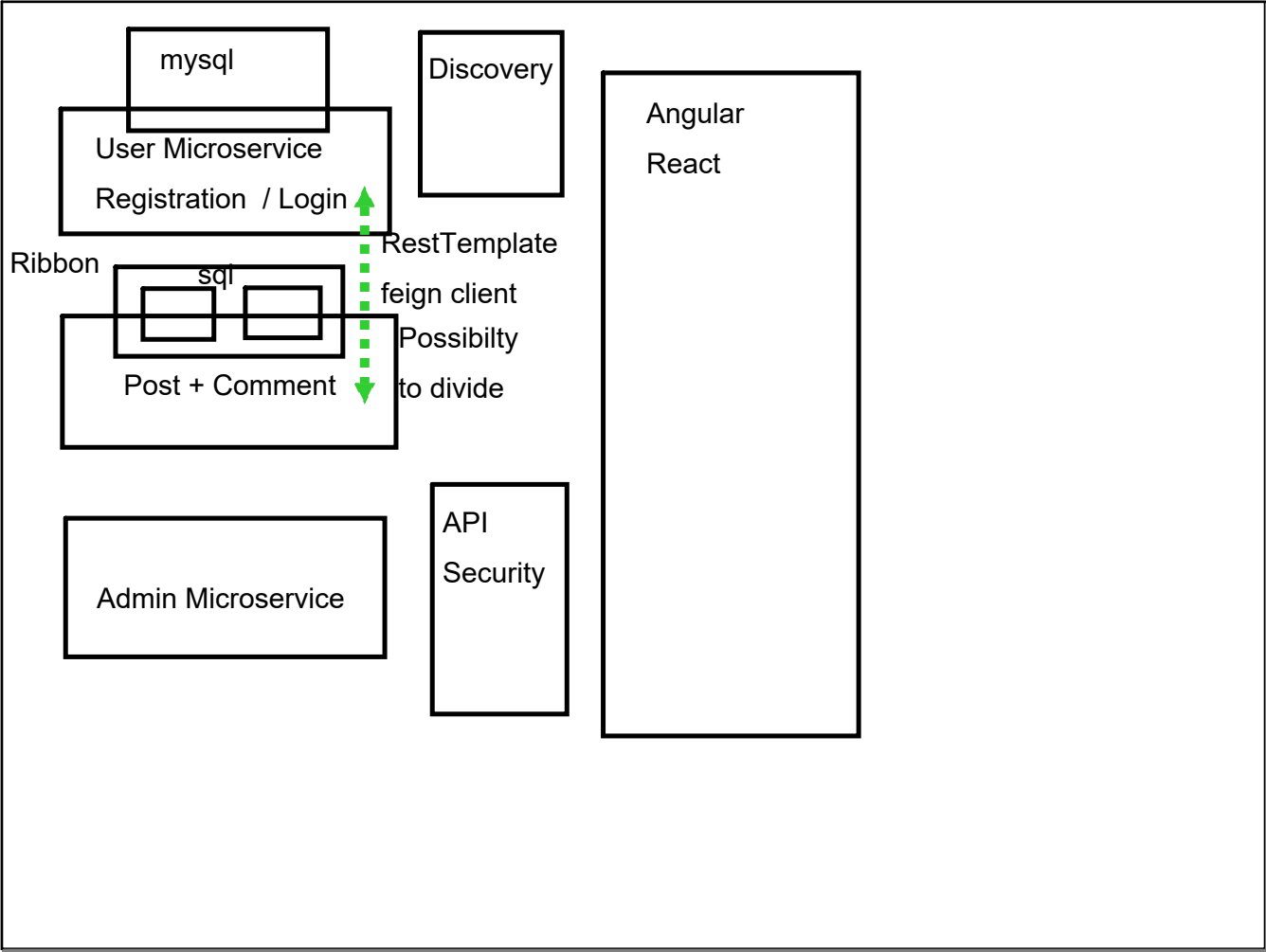
keep on trying

Netflix Hystrix

10 sec

5

fallback



Spring Cloud :

Controllers

Service

dao/repository

entity : POJO <----> DB schema

DTO : POJO
Validation

Exception

Spring Data JPA

inbuilt repository (generic) with implementation
that can be customized/extend

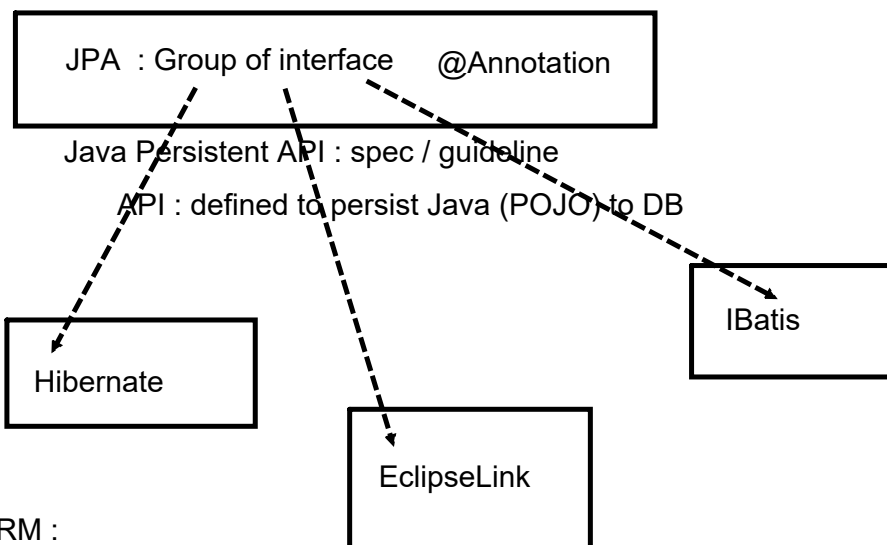
CRUD

select * from <table-name>

=>table

=>datatype of primary key

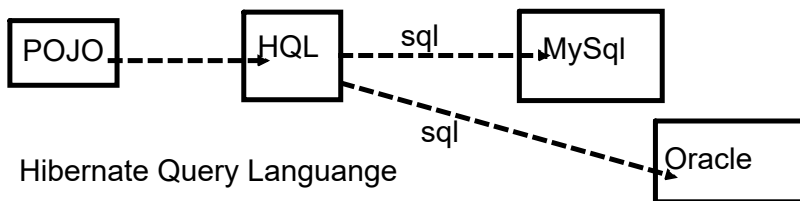
Need to configure properties file
for DB connection
: MySql-Driver : dependency



ORM :

behind the scene

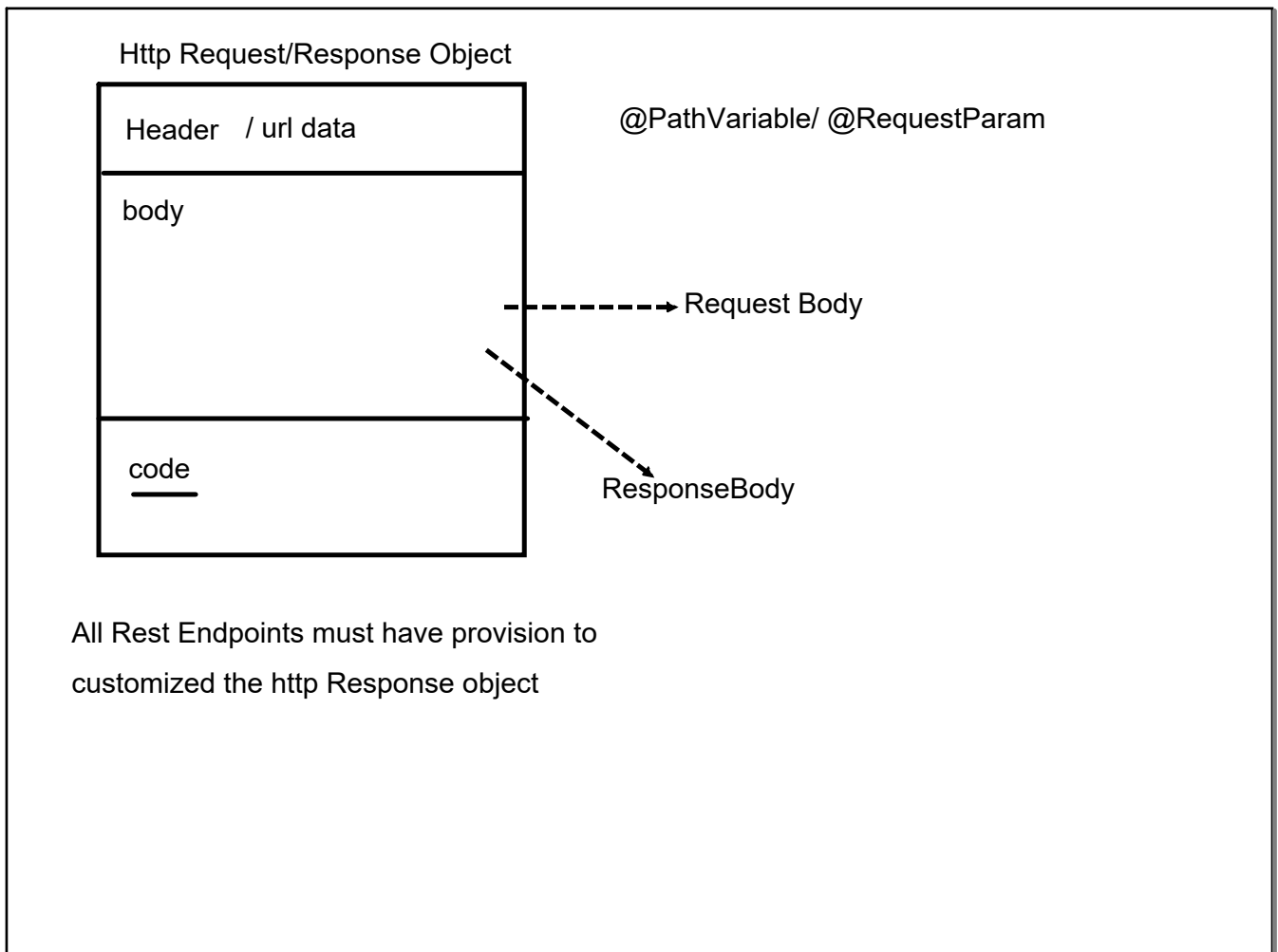
SQL queries would be created by Hibernate

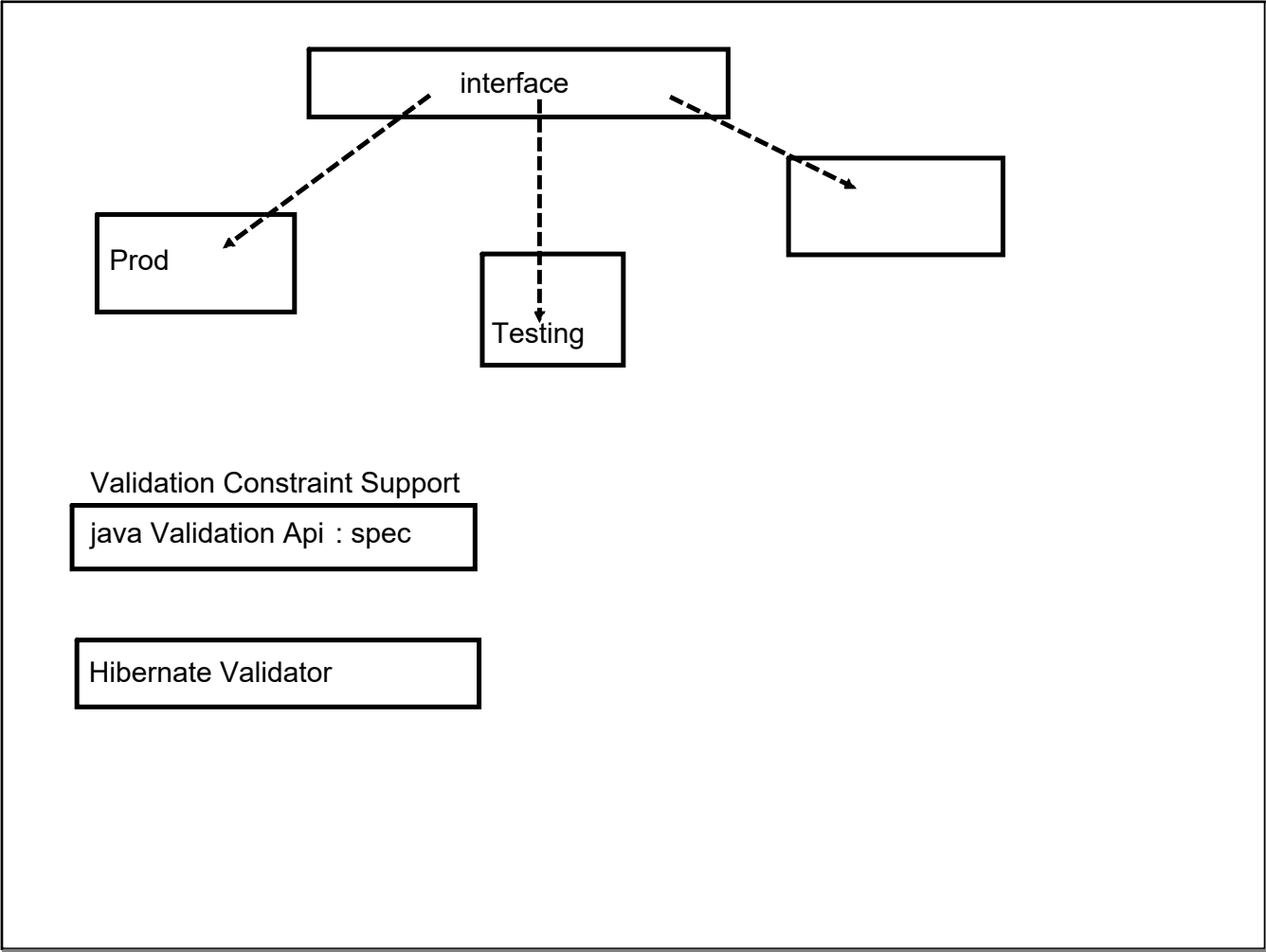


Hibernate Query Language

Need to specify the dialect

c3p0 : dependency





Client Expecting : UserDetailsDto (Success status)

Exception : UserExceptionDto (Failed status): throw an exception on client end of type mismatch

Server shall respond with appropriate status code

REst Client have provisions to check the status code

Adding a new data : instance/info about newly added data

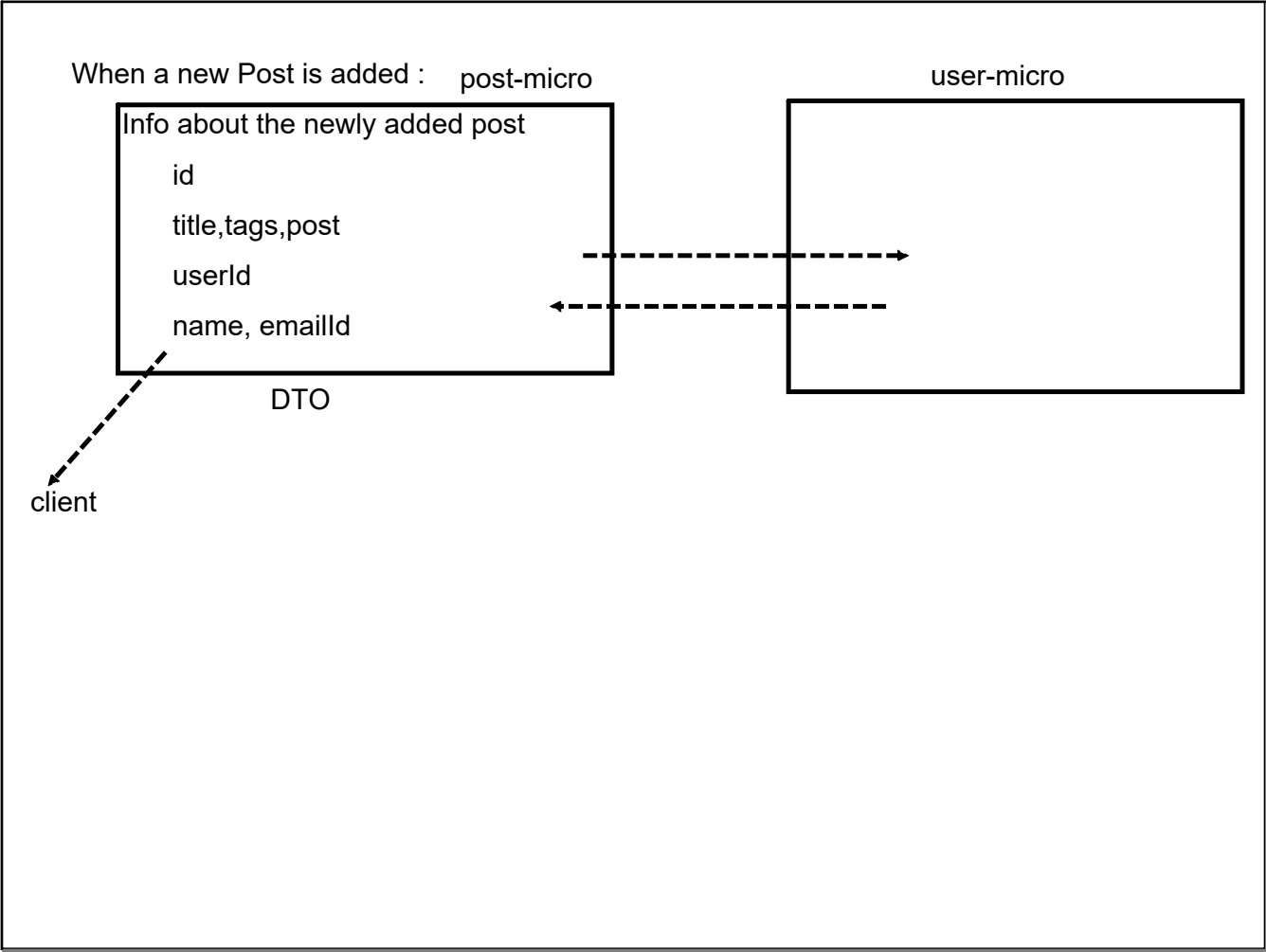
Updating the data : instance/info about update data

Deleting the record : instance/info about deleted data

DTO - entity DTO ->

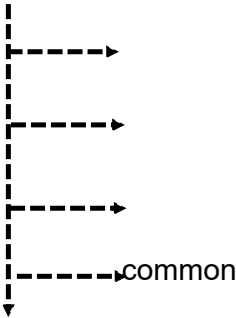
4 + 3 ---> DB

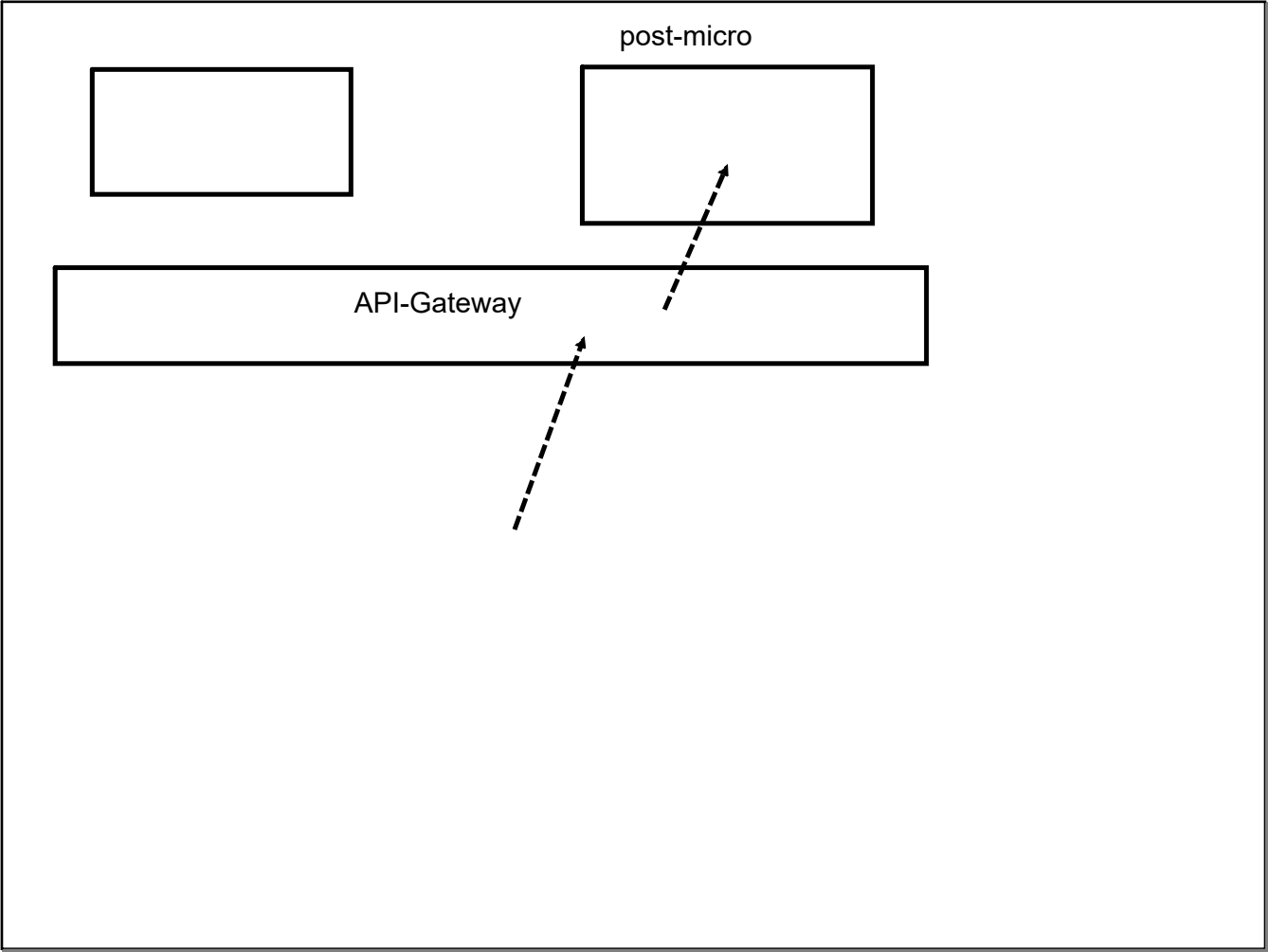
Client : 7 fld (primary

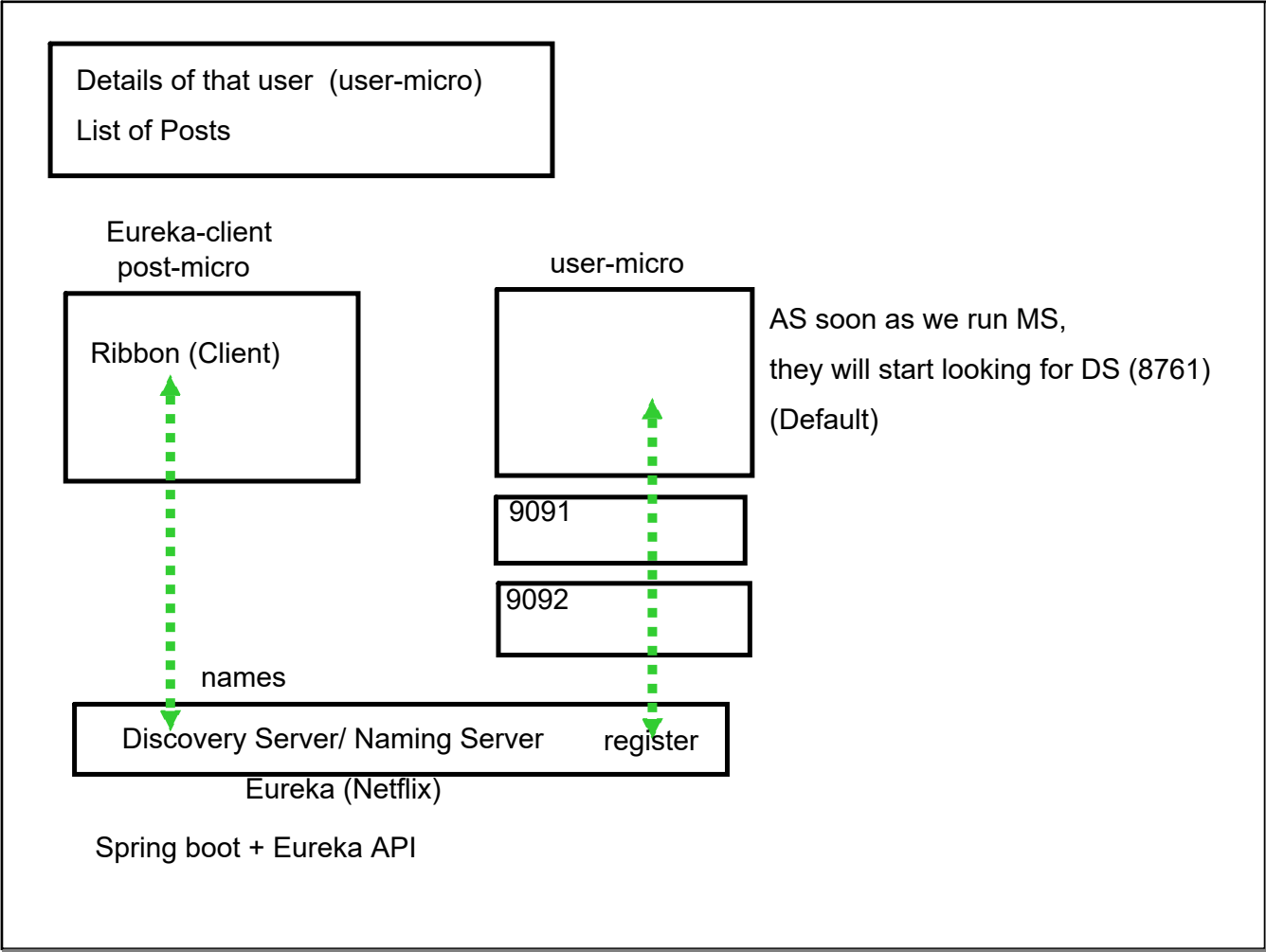




Multi-Module







Two tables

1. User credential
2. Roles

User-Credentials

table ("users")

username : String
password : String
enabled : boolean

Roles

table ("authorities")

username : String
authority : String

password : encrypted form

Spring security supports multiple encryption

eg:

Plain-Text

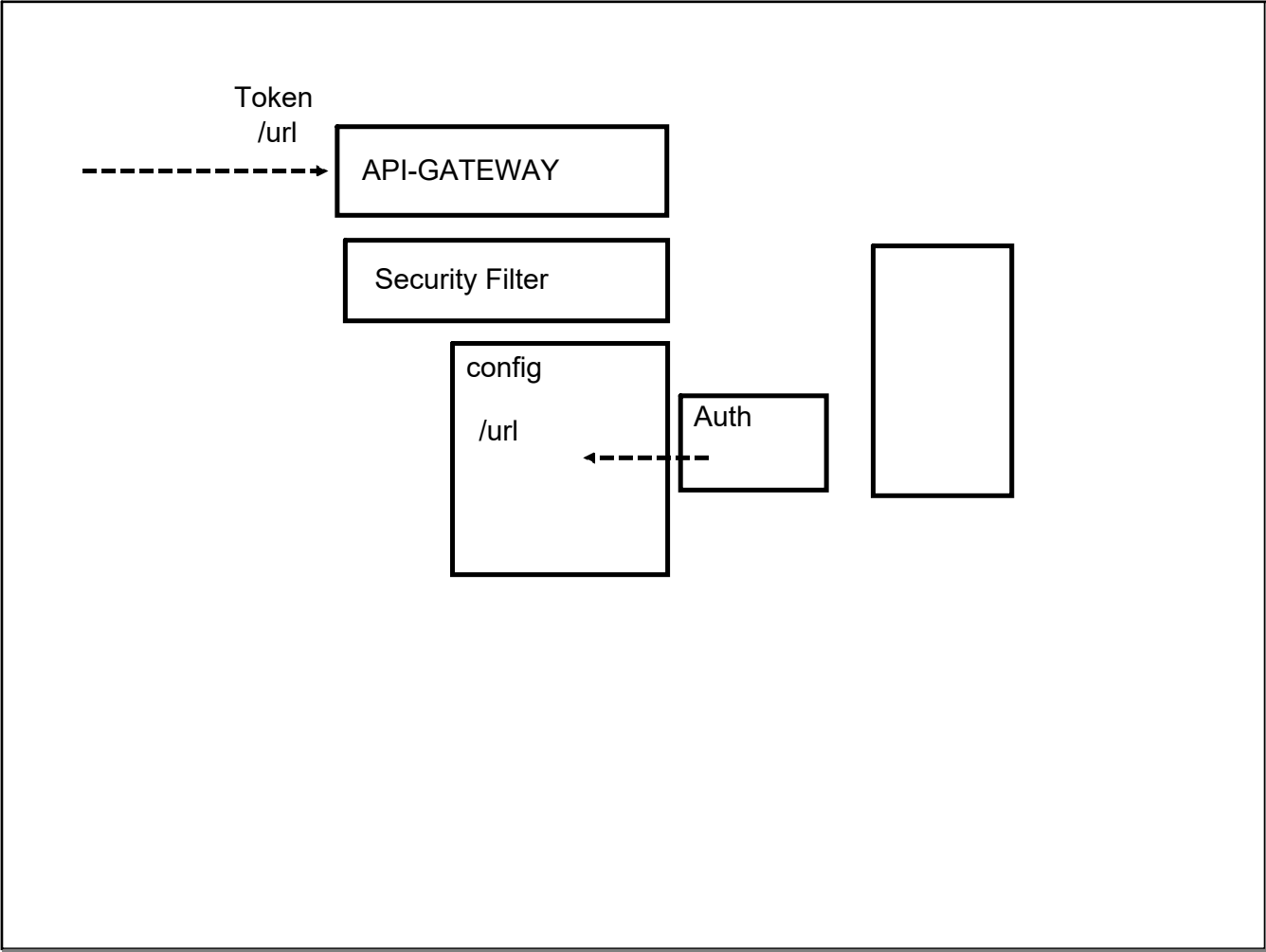
abc{noop}

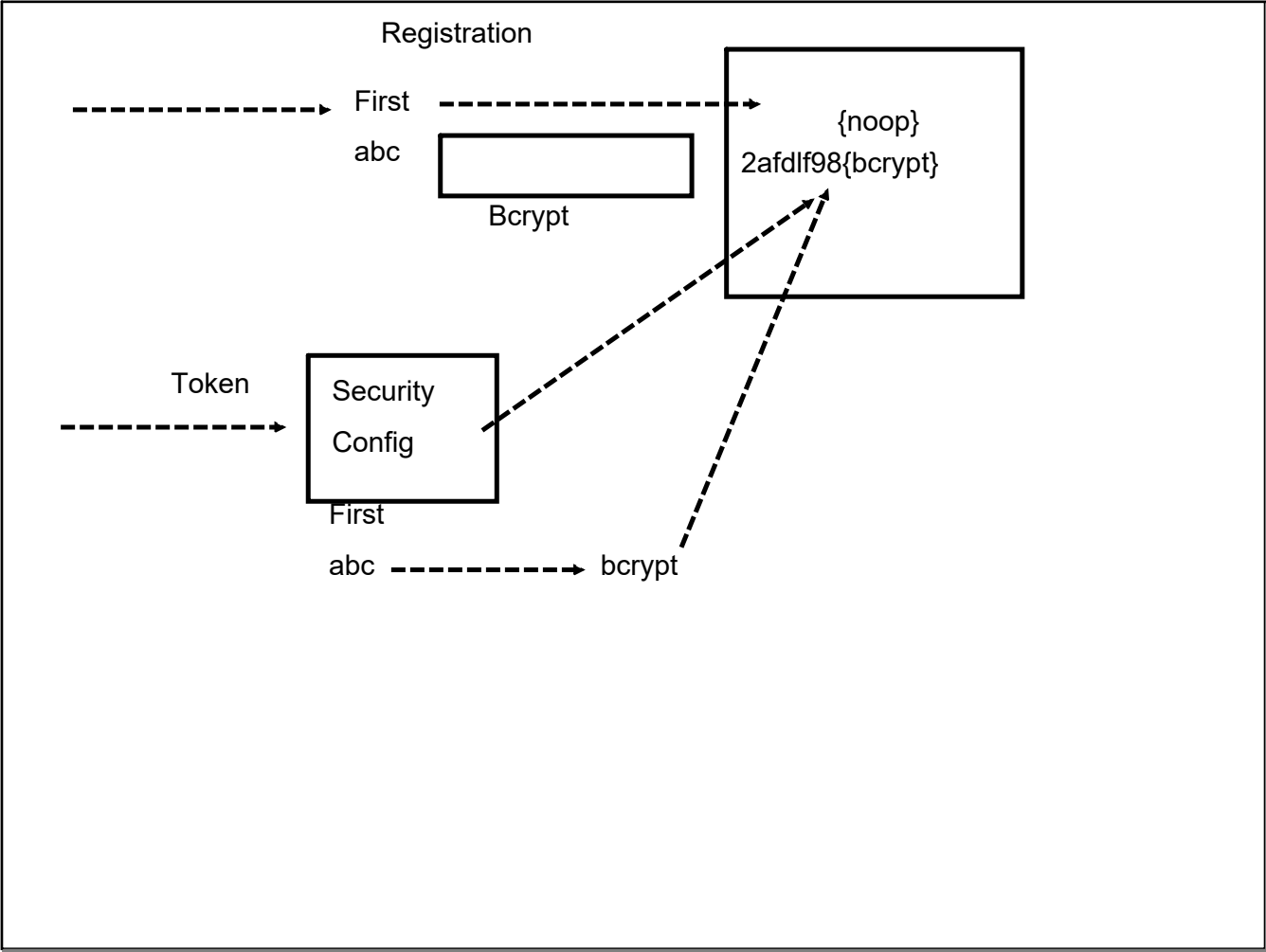
Bcrypt (one way)

{bcrypt}2afdhfldron98

Roles:

Manager ~ Role_Manager





3 core elements

HTML : Structure

CSS : Presentation

JavaScript : Behavior

HTML-5

Validations

Drag n Drop

Semantic Tags

Web Workers

Offline functionality

Geolocation

New Semantic Tag (Backward Compatible)

purpose full (specific to req)

=> container

=> attributes -- Form based extension

Smooth Rendering (outline algo)

more compatible to search algo

in sync with Assistive Tech

Standardized Error handling algo : Developers (Debug)
images/audio/videos : third party plugins : HTML5 tags + API (control)
Built-in APIs

traditional:

<p>, , <div>

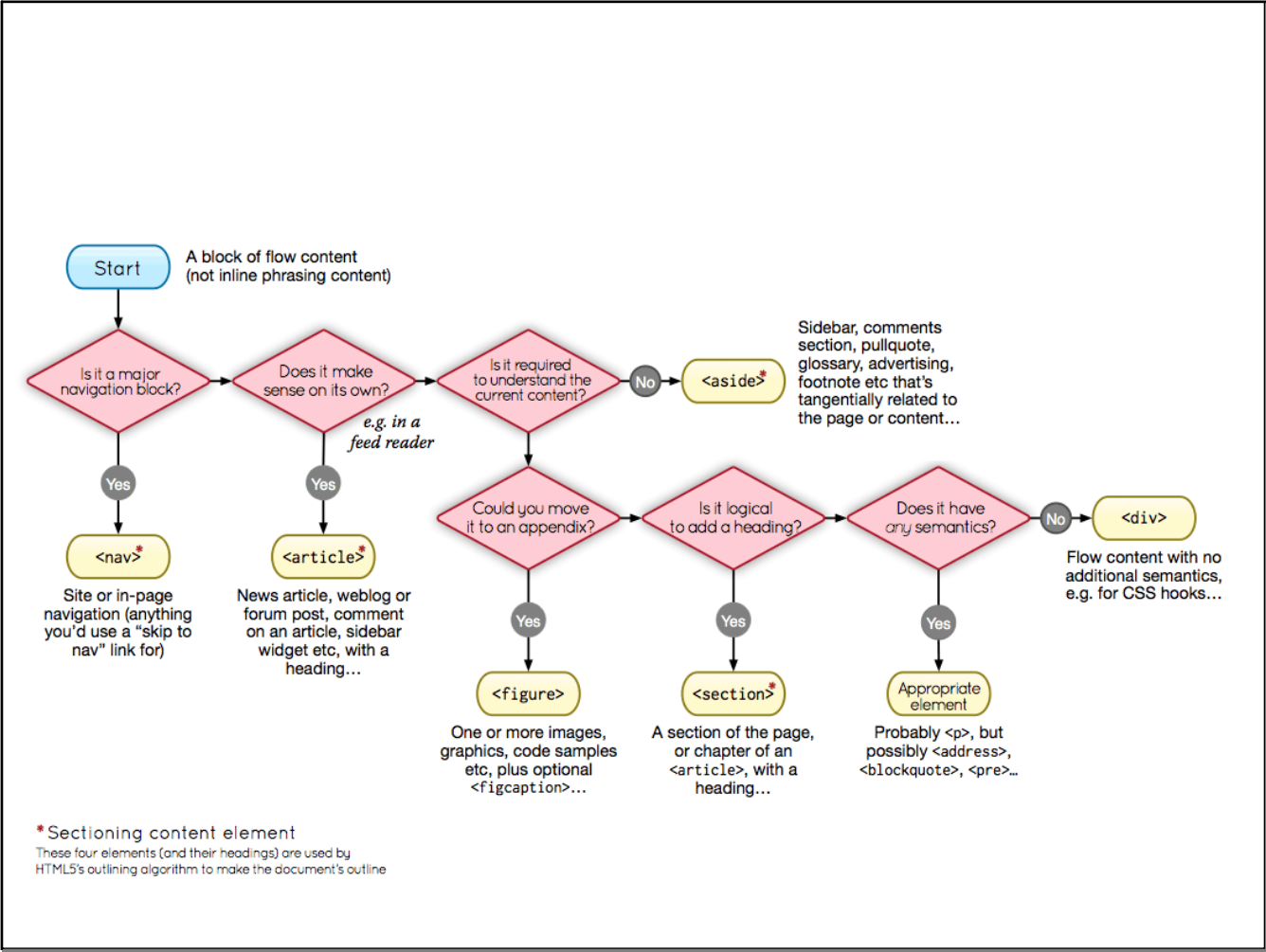
article

section

aside

header

footer



Html form :

- # specialized form inputs
- # validation : required/pattern
- # special att : custom behavior of form

<form>

S

</form>

Canvas API

DOM Tree managed by the browser

Html component(Tag) : JS - object

User Interaction : presentation : CSS

Cascade style sheet

Stylesheet : : set of rules 'presented'

Cascade : set of rules : resolve the conflict of multiple ss applied on a element

Specificity

controlling over where to apply the style

CSS rule :

CSS Selector

CSS declaration

```
selector {  
    property : value  
}
```

selector : css rule would be applied to which HTML elem

Selector

Type (most varied : wide spectrum : which type HTML element)

ID

class

eg :

p{

}

ID : very specific

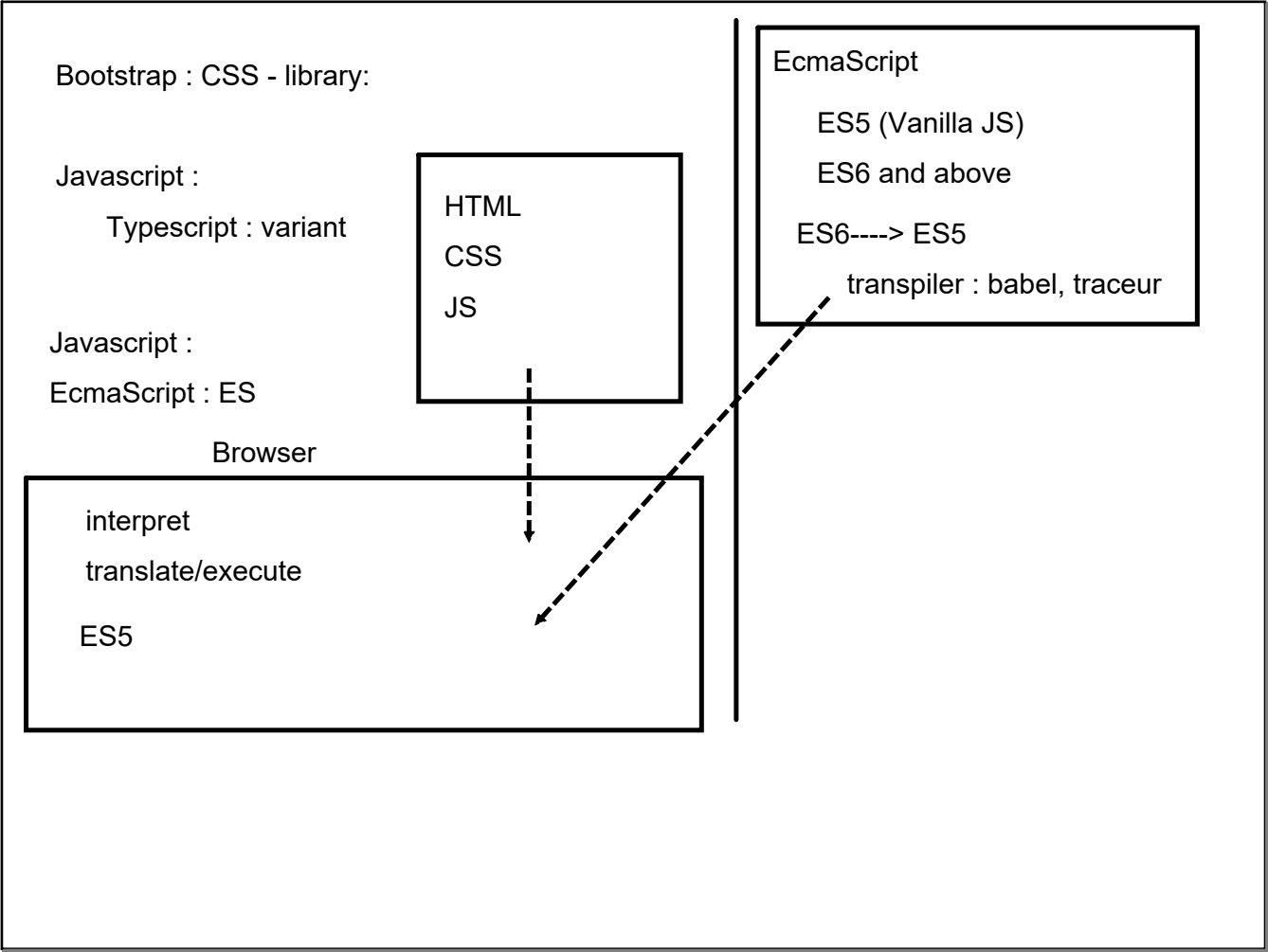
#canvastest{

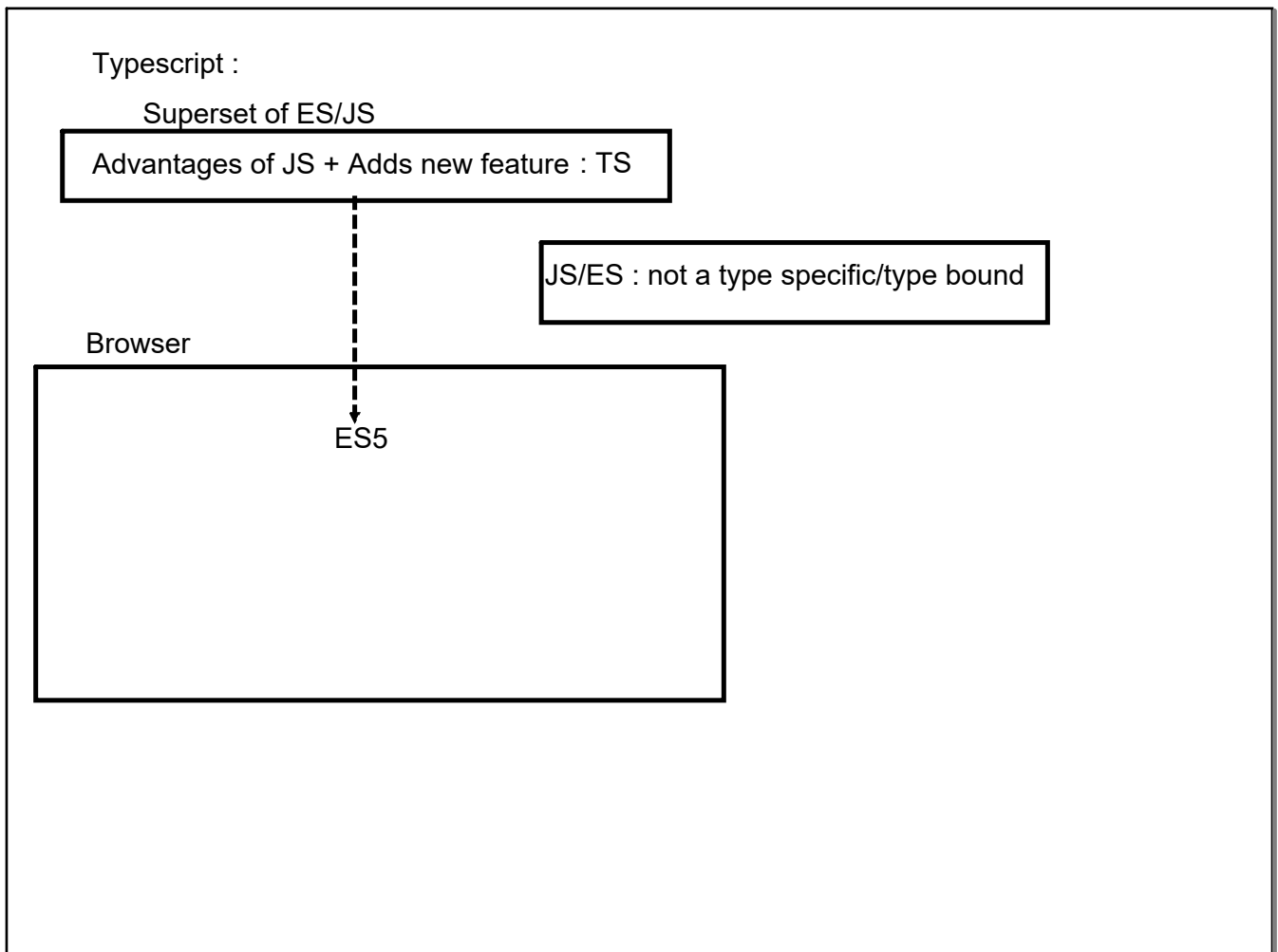
}

class

.mclass{

}





Javascript

```
function add(num1, num2){  
  // validation check  
  return num1 + num2;  
}
```

call : add(20, 30); // arithmetic addition

: add('hello', 'world'); // string concatenation

Unwanted behavior at runtime

Typescript :

Named Types...

NextGen JS features

NonJS features like Interface/Generics

Decorators (Meta-Programming)

More Config options

Transpiler : Typescript compiler

Javascript based resource, managed way

management tool :

nodejs : npm : node package manager

yarn

NodeJs : installed + system path

(npm) : cli

NodeJs : Framework that allows to use JS for server side programming:
non-blocking, asynchronous server implementation

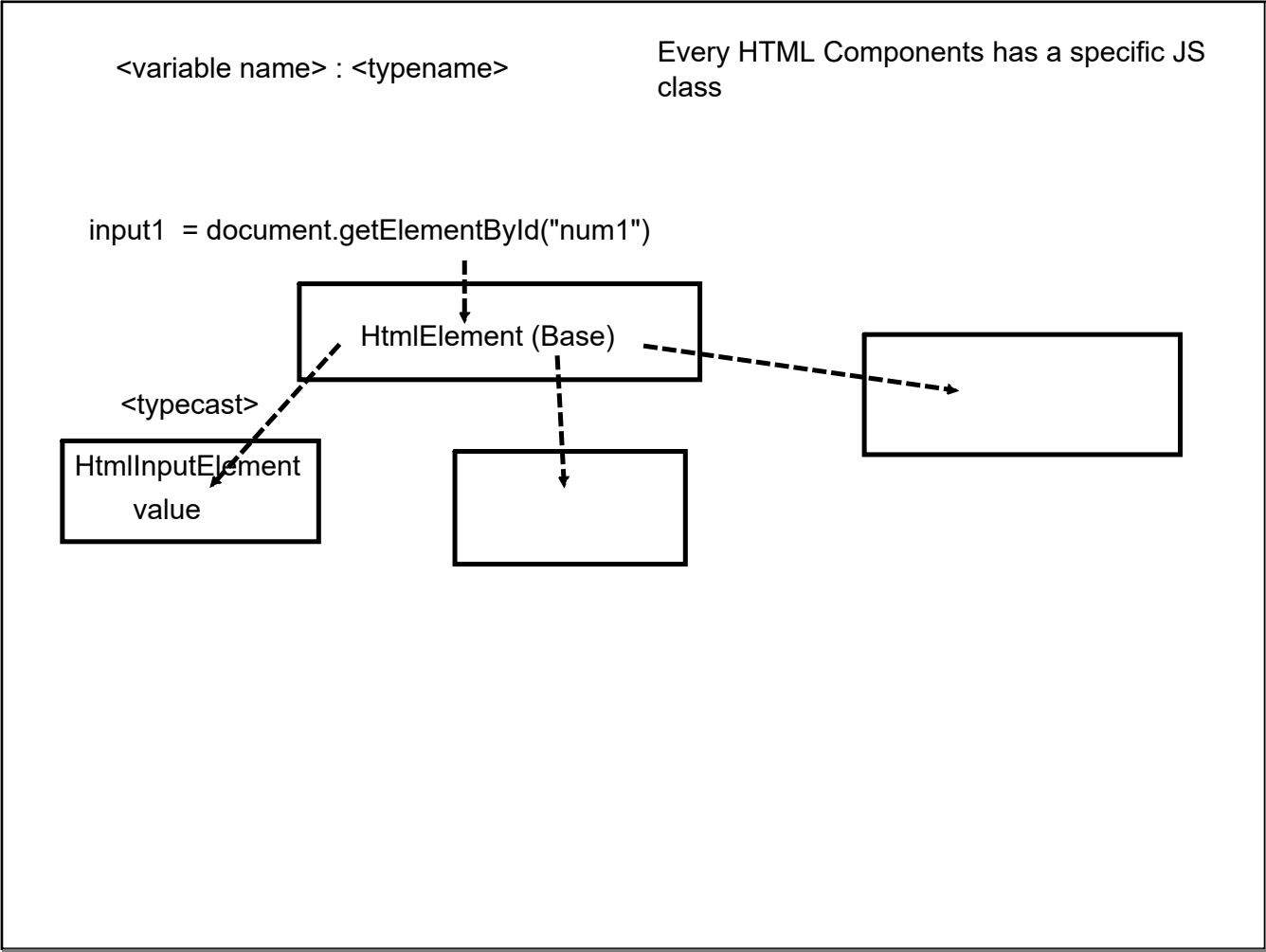
npm : is a project management tool for JS related project management

Need to install typescript compiler

>npm install -g <tool> (global installation)

> npm install -g typescript

Typescript file must have ext : .ts



var ~ ES6 : const / let

Core Types

number : integer/fractions

string : 'hello', "hello", `hello`

boolean : true,false

object : Javascript object (more type specific)Object Notation

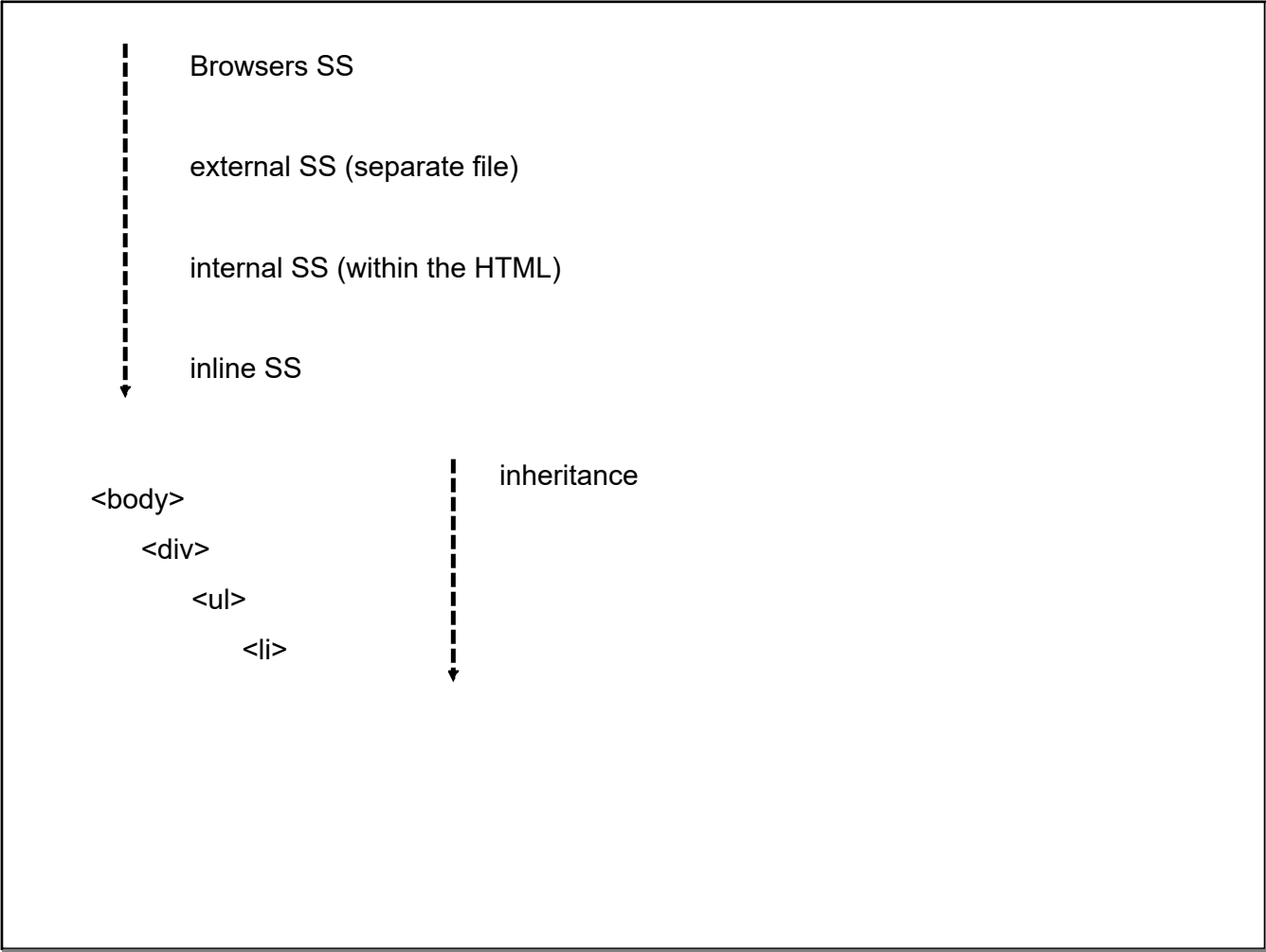
Array : JS has way to create array of heterogenous nature (TS : homogenous)

Tuple : Fixed length : Type

Union : specify multiple types

Enum : enumerated Datatype

any : default JS type

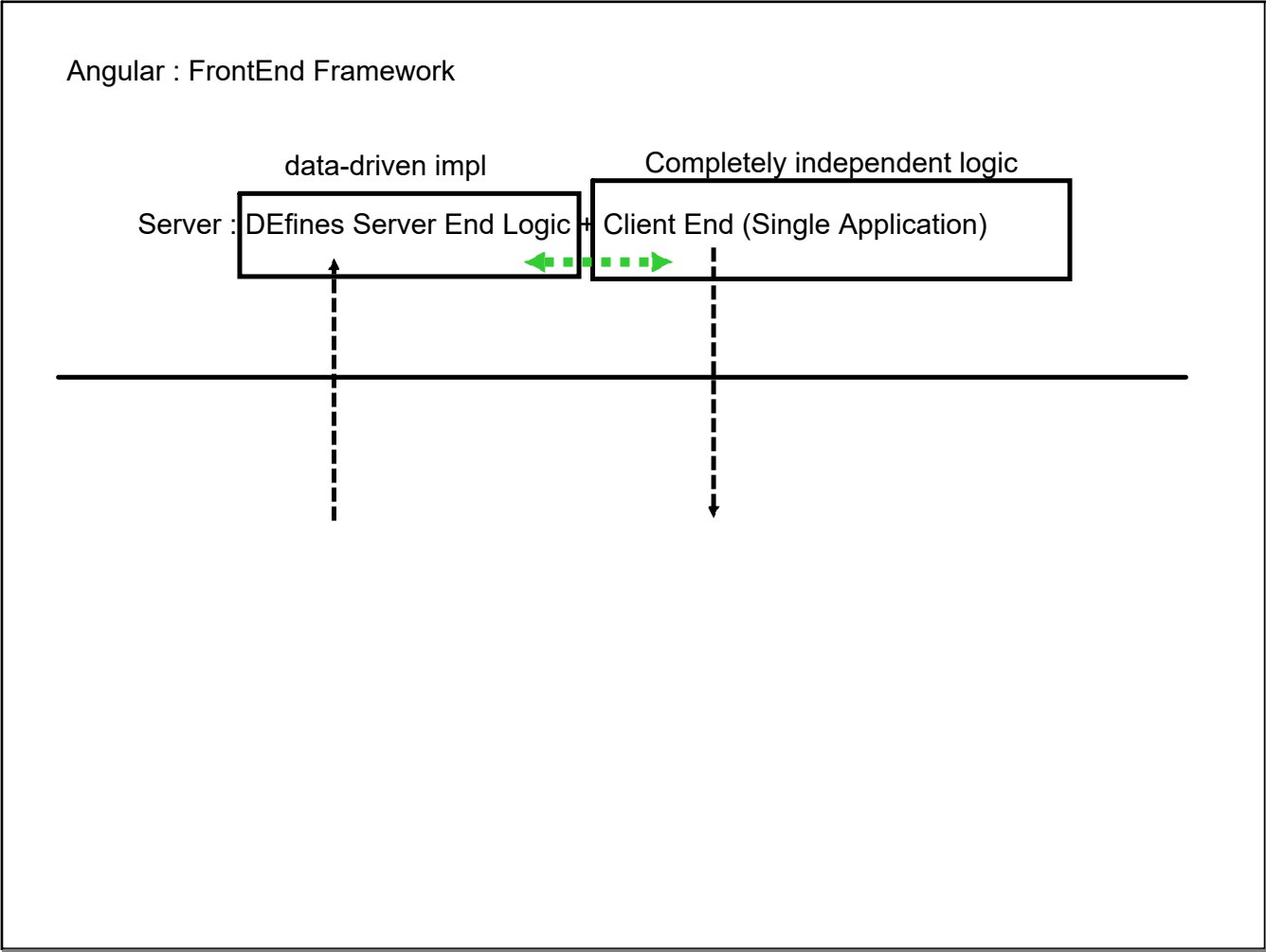


Classes : high level way :

Closures :

have global variable(memory retains across function calls) with local scope

static variables of C functions



Loose coupling of Server Side (backend logic) and Client Side (Frontend logic)

1. Server Side is reusable
2. Client Side is also reusable (flexible)
3. More independent implementation
4. Load Distribution among client machine (rendering the dynamic web-pages : JS)
5. Client End Rendering can Highly customized

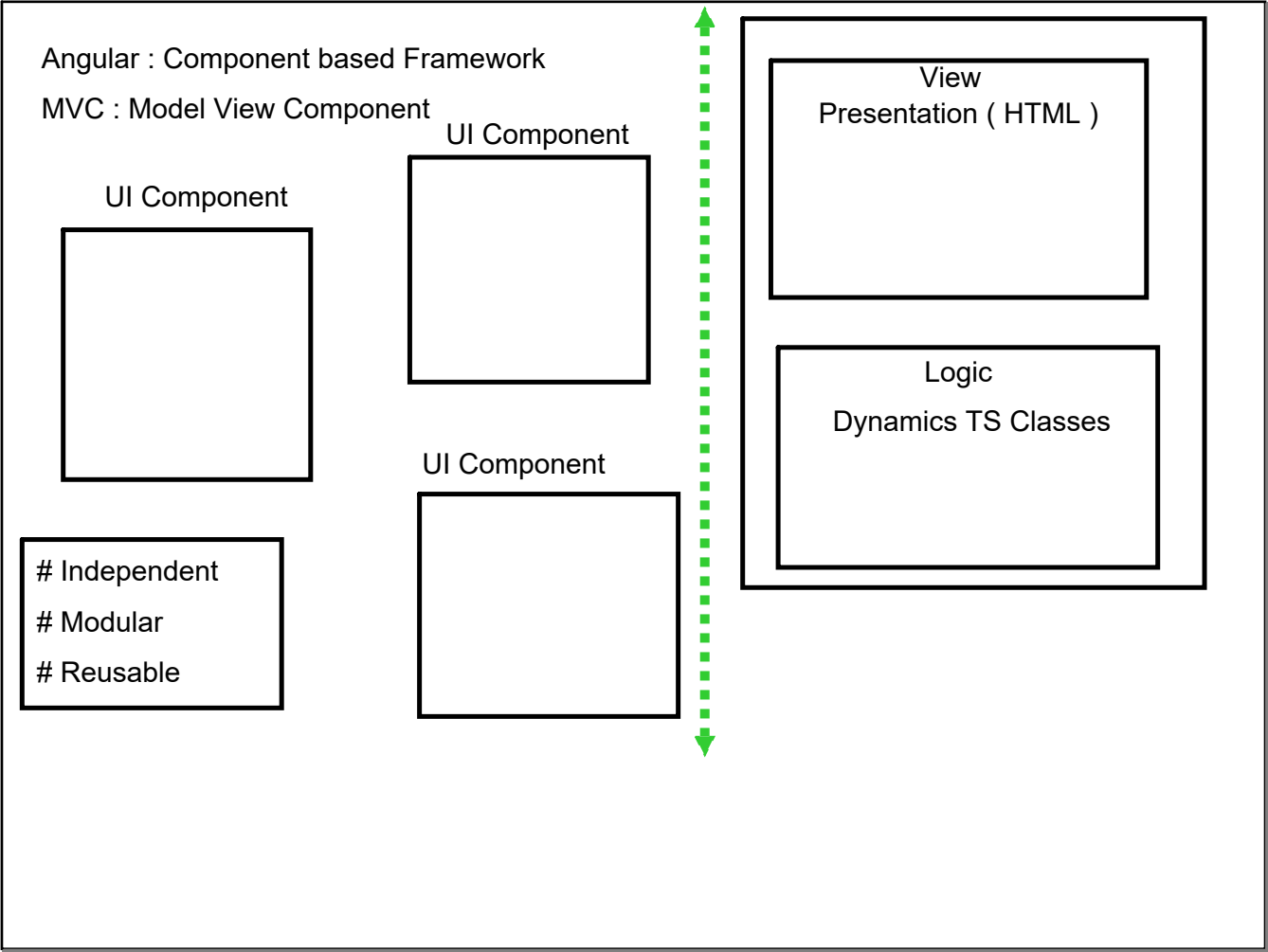
Angular Framework

Complete Framework

Base Script : TS

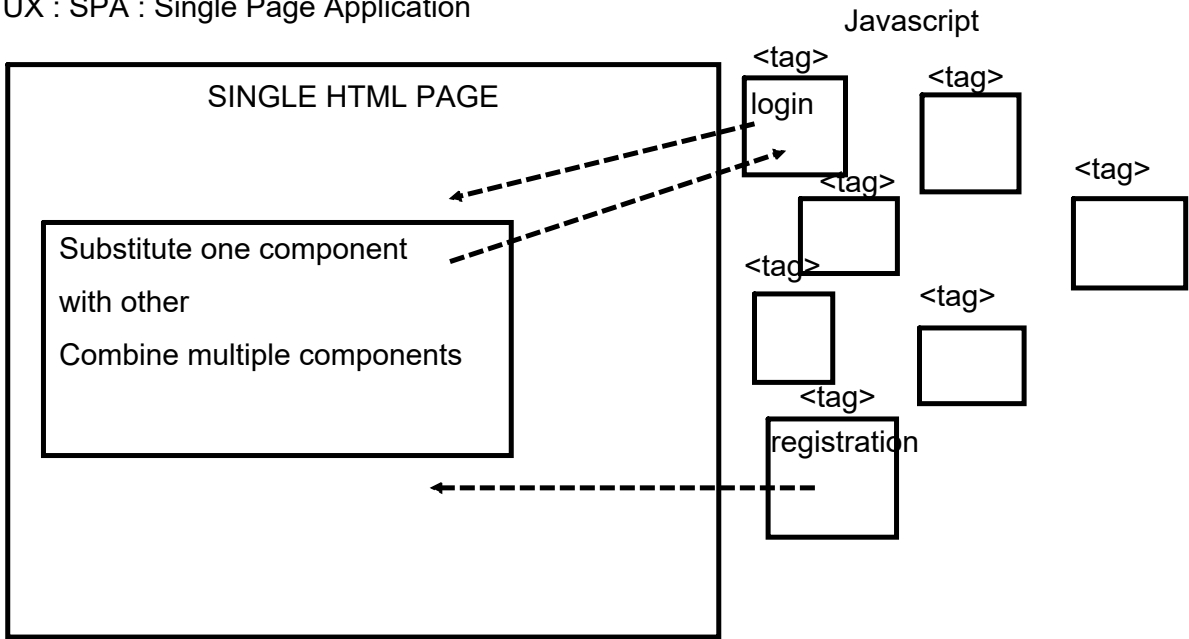
Resources : Client Side JS Community Library

npm to manage angular application



Relationship : (logic + presentation) decide the output

UX : SPA : Single Page Application



Angular/CLI Project needs to be installed

Download angular CLI/installed

(by default latest version)

> npm install -g @angular/cli

Angular CLI will expose angular specific command

> ng <option> (syntax)

> ng new <project-name>

1. Complete folder/file structure required as Angular Framework project

2. Download default Angular lib

> Add routing module (Y)

> Stylesheet : CSS(default)

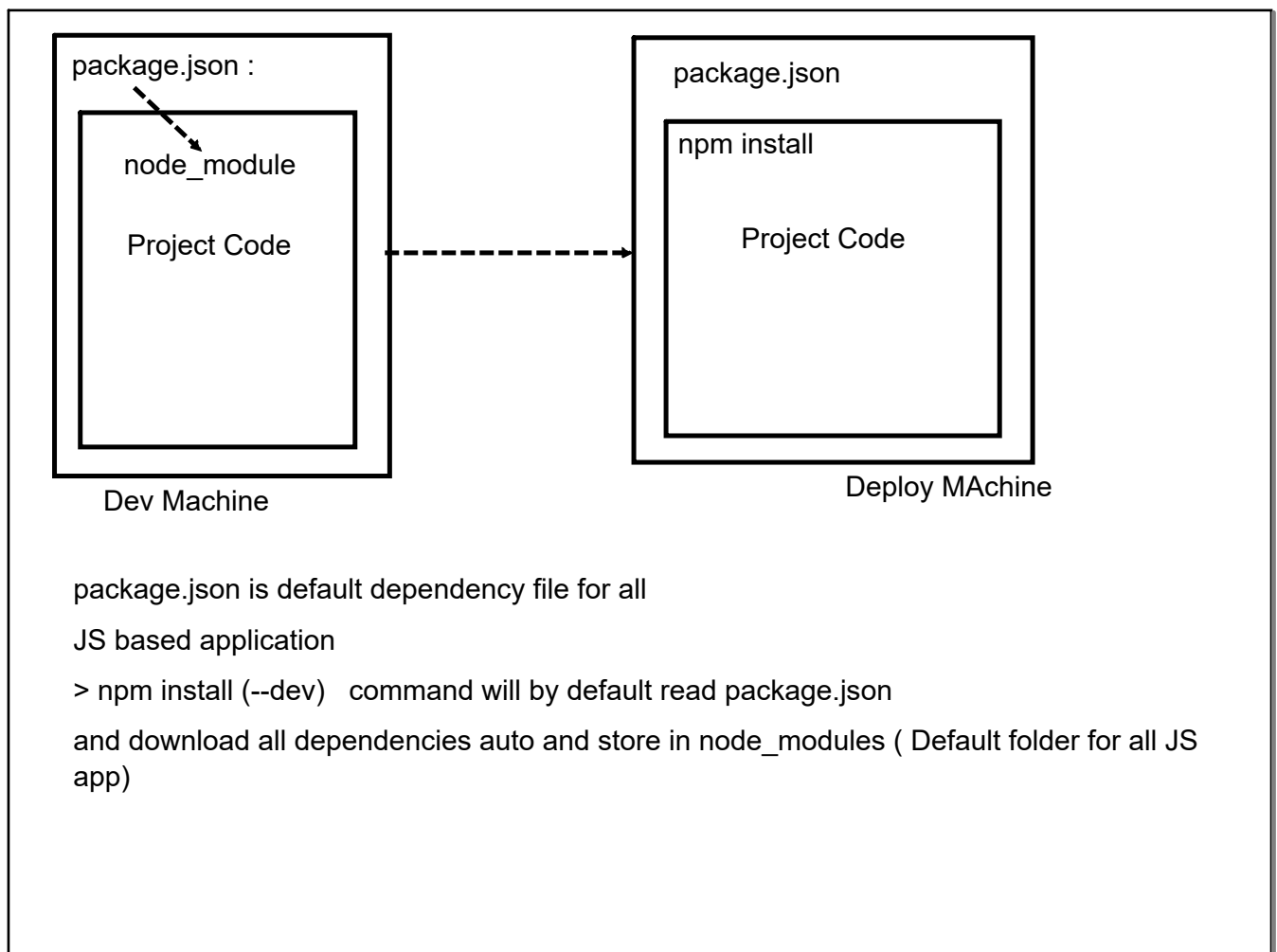
Feature Set for Unit/Integration Testing and End-To-End Testing

1. Jasmine Framework : JS Testing Framework (Write Test case unit/integration + e2e)
2. Test Runner : Unit Test (Karma)
3. Test Runnner/Framework : End-To-End Testing (Protractor)

e2e : supposed to contain test cases/config related to End-To-End Testing

node_module : All lib are stored in this folder

src : All Angular code goes here



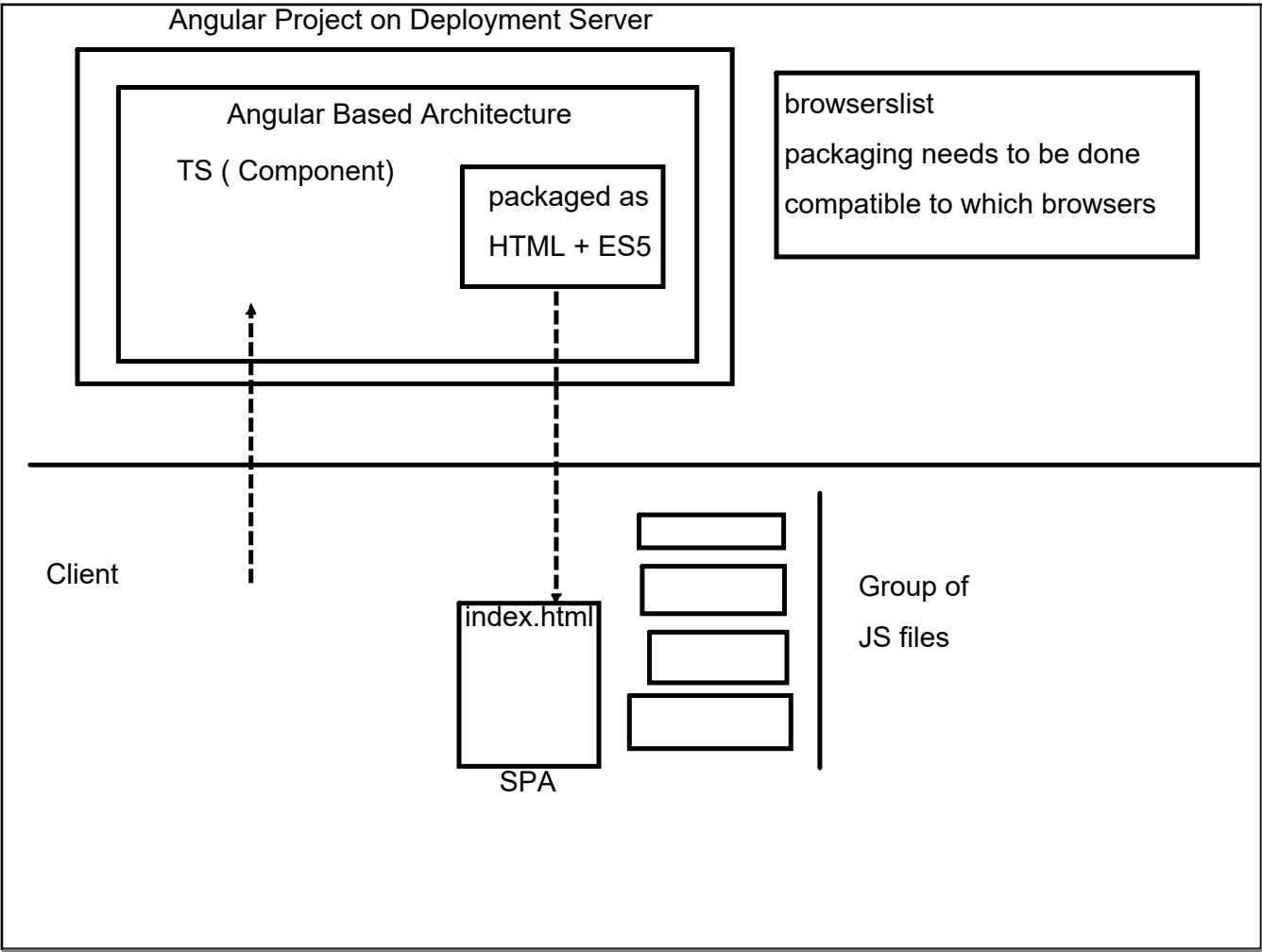
Adding a new Dependency:

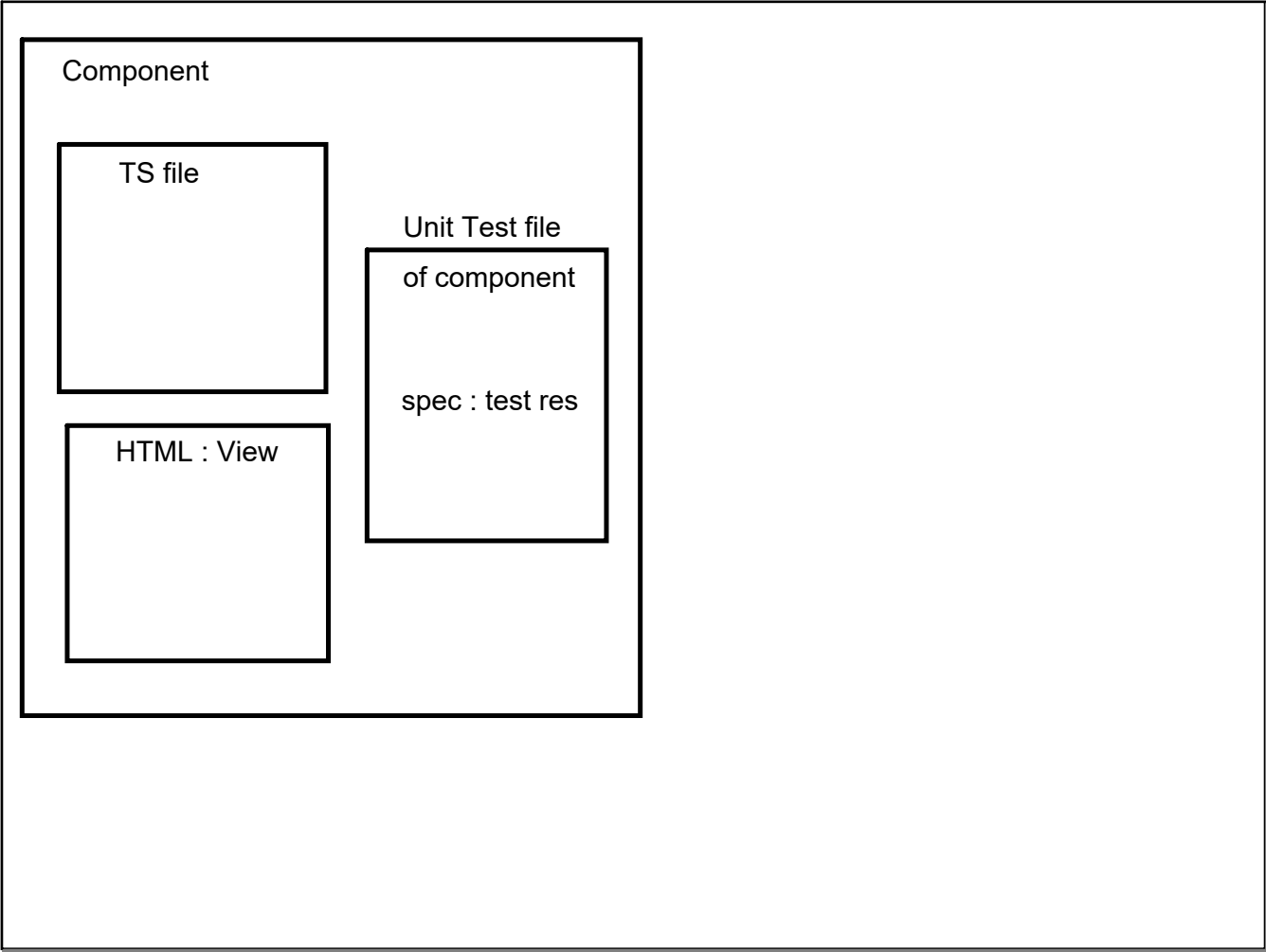
1. add an entry in package.json
2. npm install (download the dependency and add it to node_module)

1. npm install -g <lib-name> (install library globally in my system)

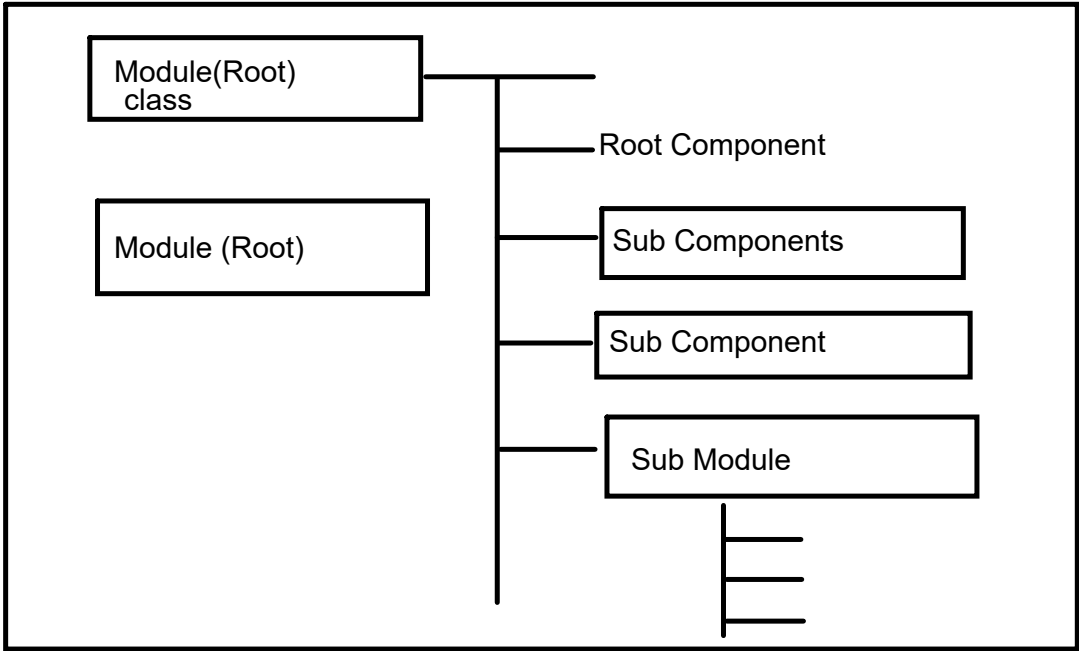
~ npm install --save --dev <lib-name>

1. add a entry in package.json(update)
2. down load dependency and save it in node_module

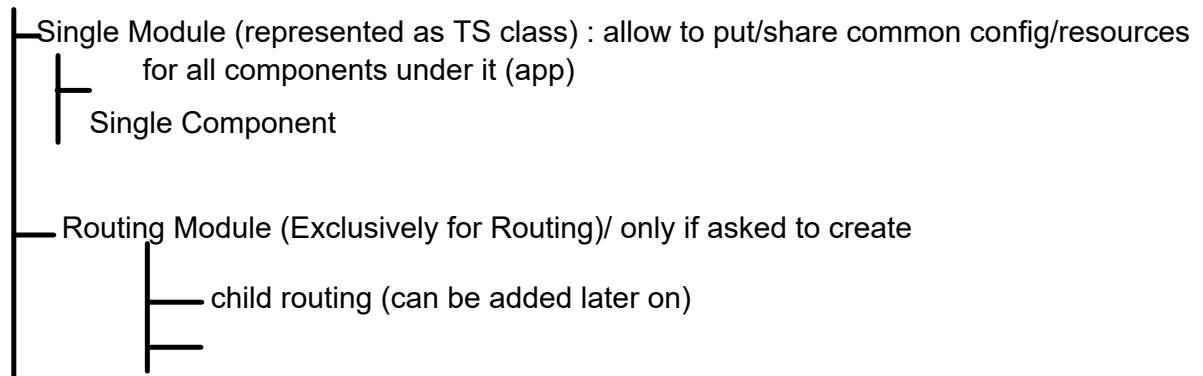




Angular Code Construct



By Default:



```
import  
import <class name> from <library>  
import {<class name1>,<class name2>} from <library>
```

Component :

TS class : supported by presentation (View)

By default :

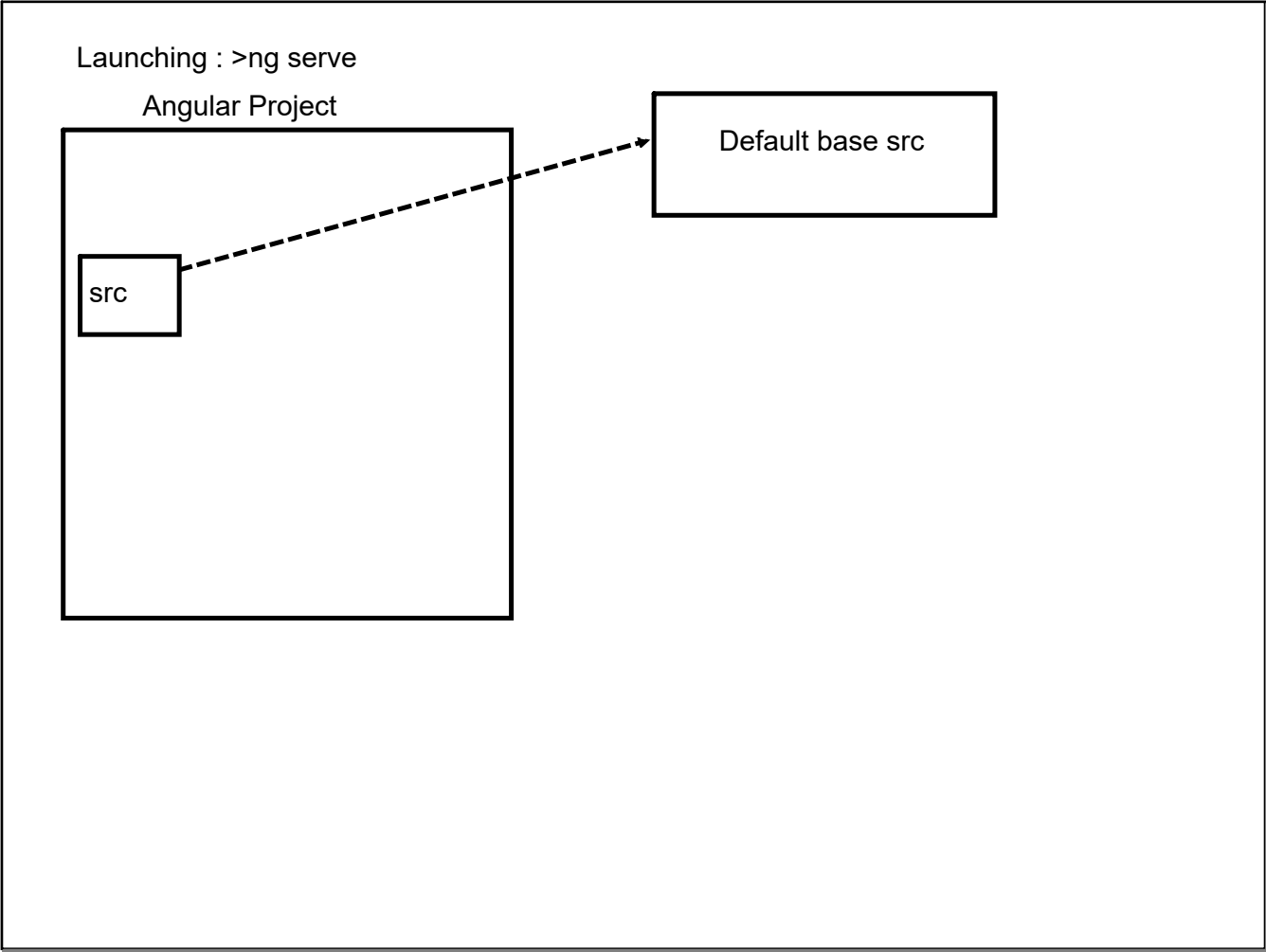
Angular : 4 files for each component

TS class (mandatory)

HTML file (View)

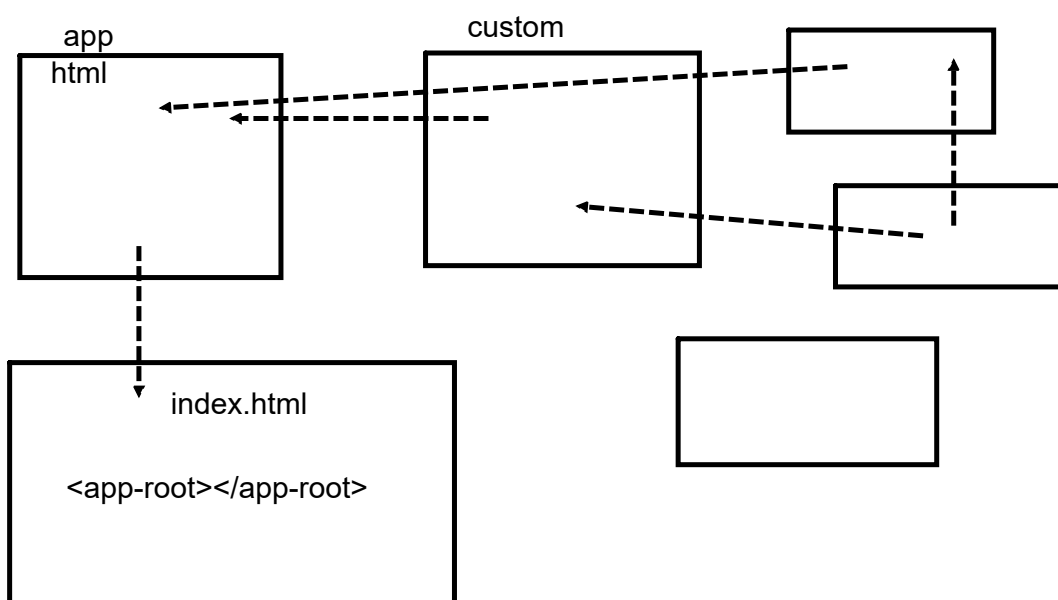
CSS (contain exclusive classes for that component)

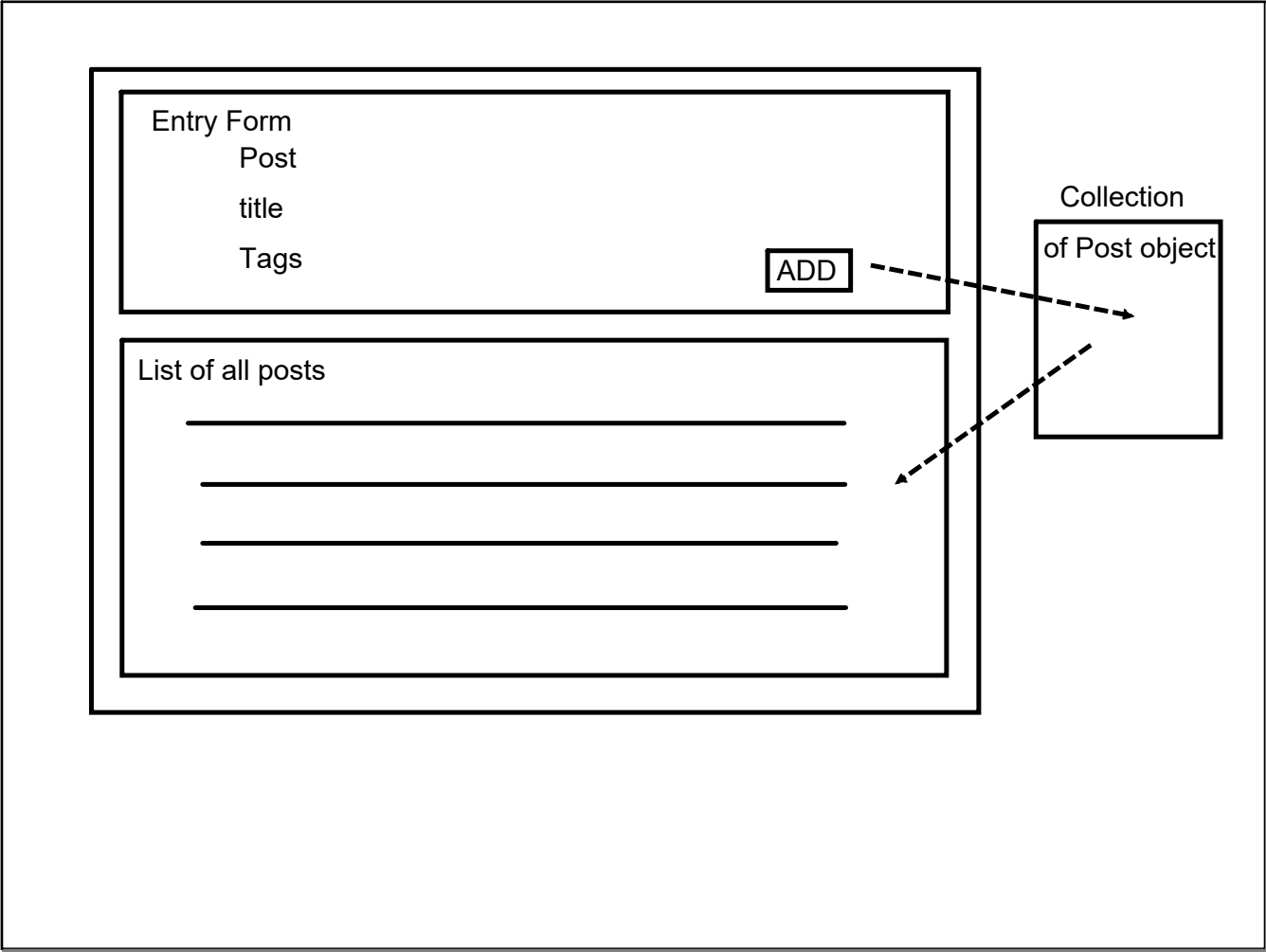
Test : unit test code for that component

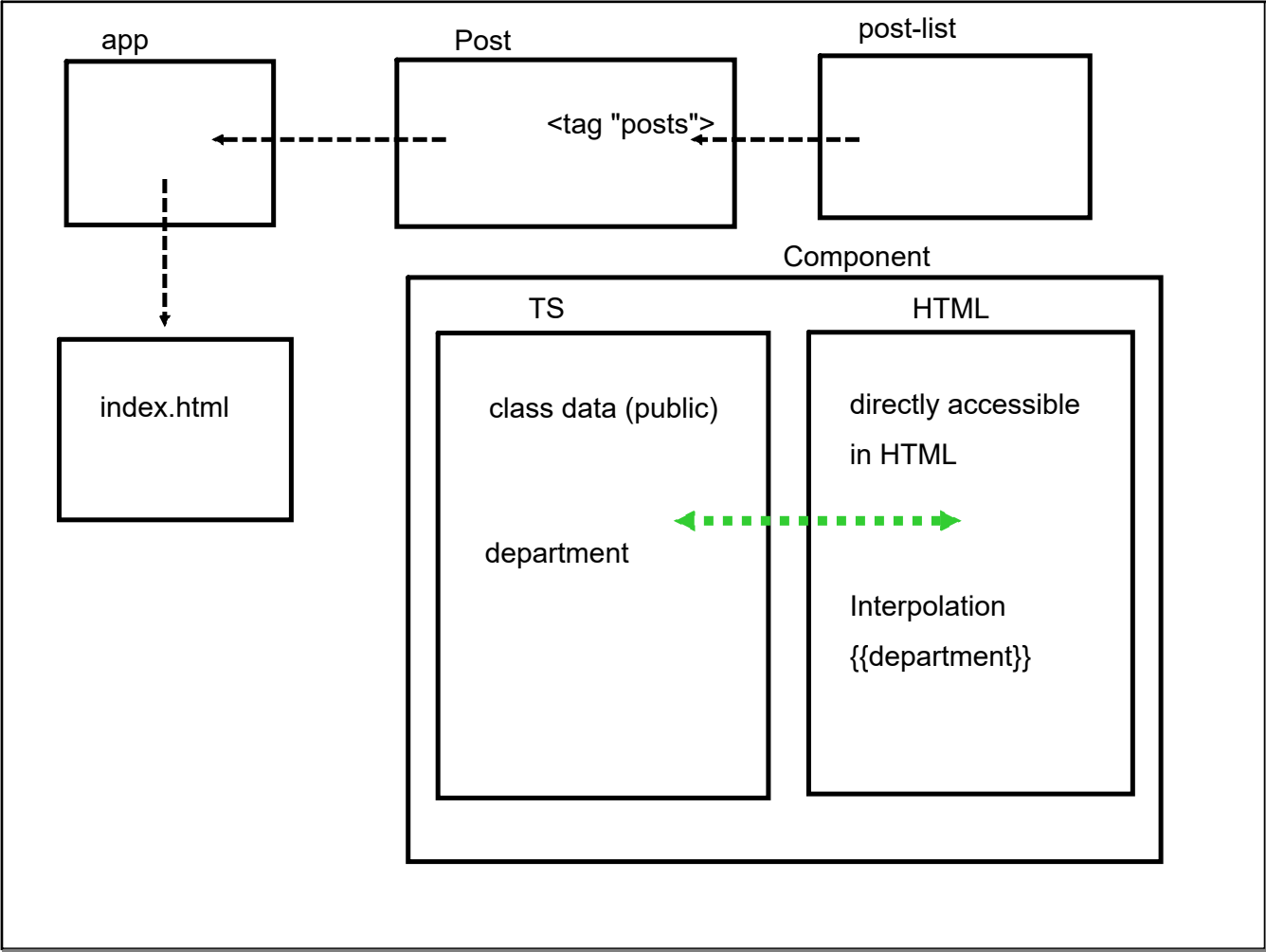


Creating a new component

> ng generate component <comp-name>







Angular : Directives (Dynamic in HTML)

HTML

Extended by directive

<new tags>

< new attributes> along with existing
HTML attributes

eg : for loop directive

Angular allows to associate a variable with HTML elements

```
var txtTitle: HTMLInputElement = document.getElementById("")! as HTMLInputElement;
```

Angular : Synthetic events : allows to call TS class methods on events

