

Spring Framework:

Framework for building Java Applications

Simpler and lightweight alt to J2EE

Early version of J2EE complex object management

Multiple deployment descriptors

Multiple interface

Poor Performance

Rod Johnson

Lightweight Object management tool : Object Factory/Bean Factory/Application Context

Spring :

=> highly Modular

=> Independent to be used : loose coupling among modules

=> Java POJOs : Lightweight dev process

Relationship among resources of Spring application

1. IoC : Inversion of control
2. DI : Dependency Injection
3. AOP : Aspect Oriented Programming (proxy)

Modules

Core Container : Bean Factory

Web Layer : MVC Framework

Data Access Module : JDBC/ORM/Transaction

Infra : AOP/Messaging

Testing : JUnit/Mocking

Spring Projects:

Spring Data

Spring Cloud

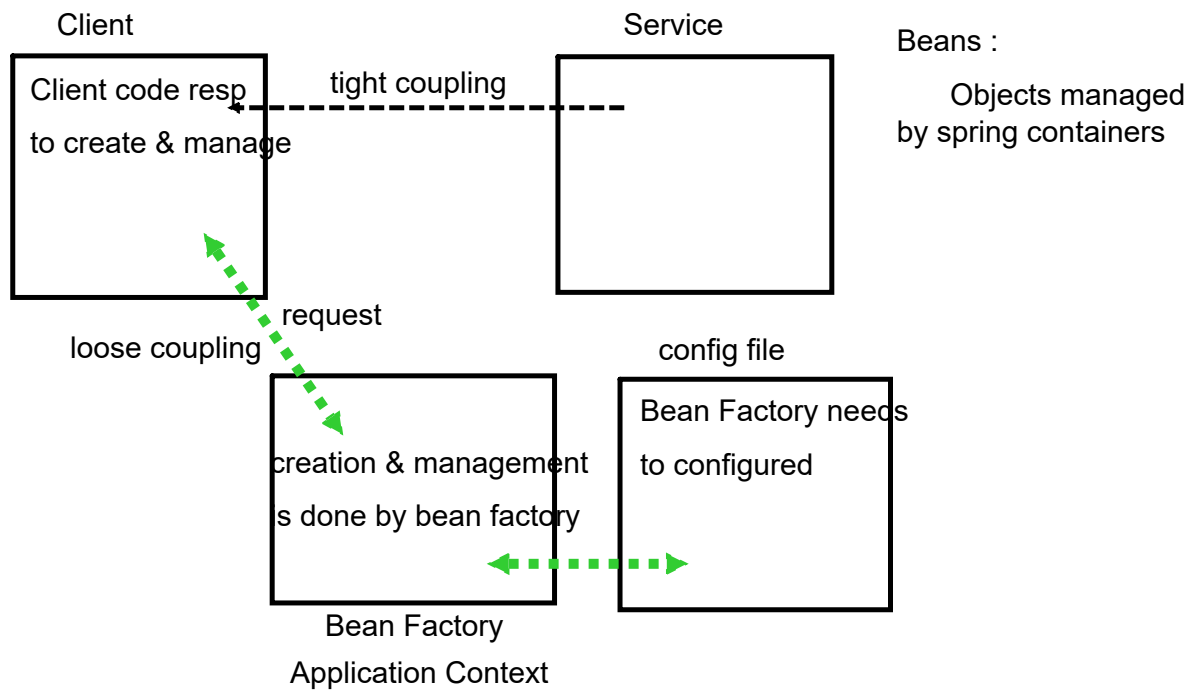
Spring security

WebFlux

1. IoC : Inversion of Control

Outsourcing the Creation and Management of Object :

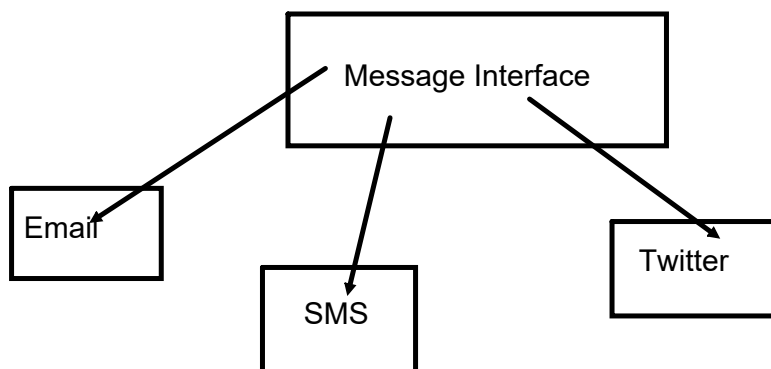
Object Factory/Bean Factory



Spring Jar files : <https://repo.spring.io/release/org/springframework/spring/>

Configuration of Bean Factory

1. XML based config (legacy)
2. Annotations
3. Pure Java Code



```
<bean id="mbservice" class="com.wf.training.spring.service.EmailService">
```

key: by which bean is exposed

class: whose bean we want to create

DI : Dependency Injection

Creation of complex object might have dependency on other object :

injecting those dependency with the help of Spring Container

XML based config:

1. Constructor based
2. Setter based



literal values should not be integrated in config file

==> property file to keep the literal values

KEY - VALUE PAIRS

Referred by SpEL

Scope of Beans

Default scope : Singleton

Only one instance which shared among all calls

Scope possibility :

singleton

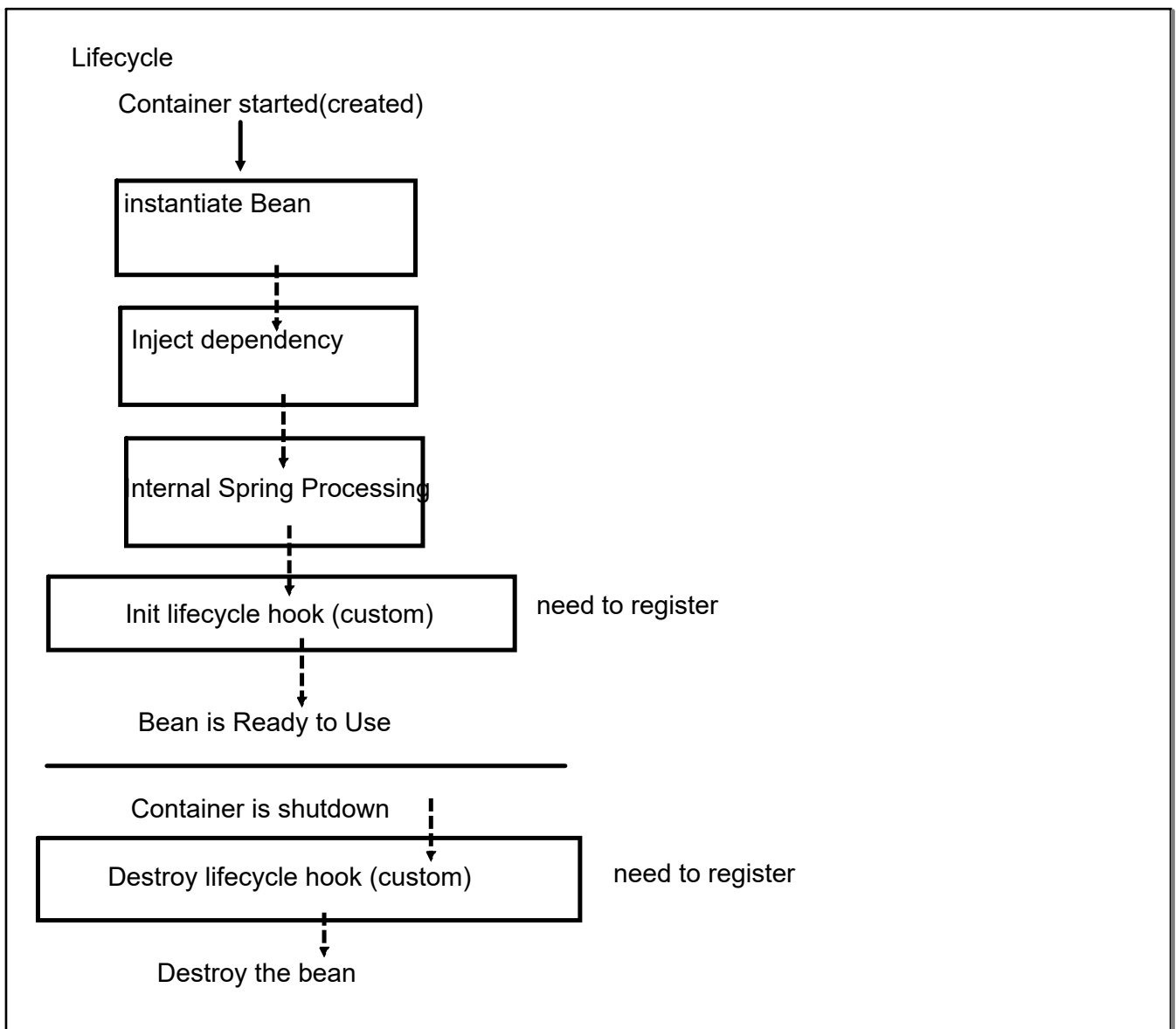
prototype : a new bean would be created

Web Context :

request

session

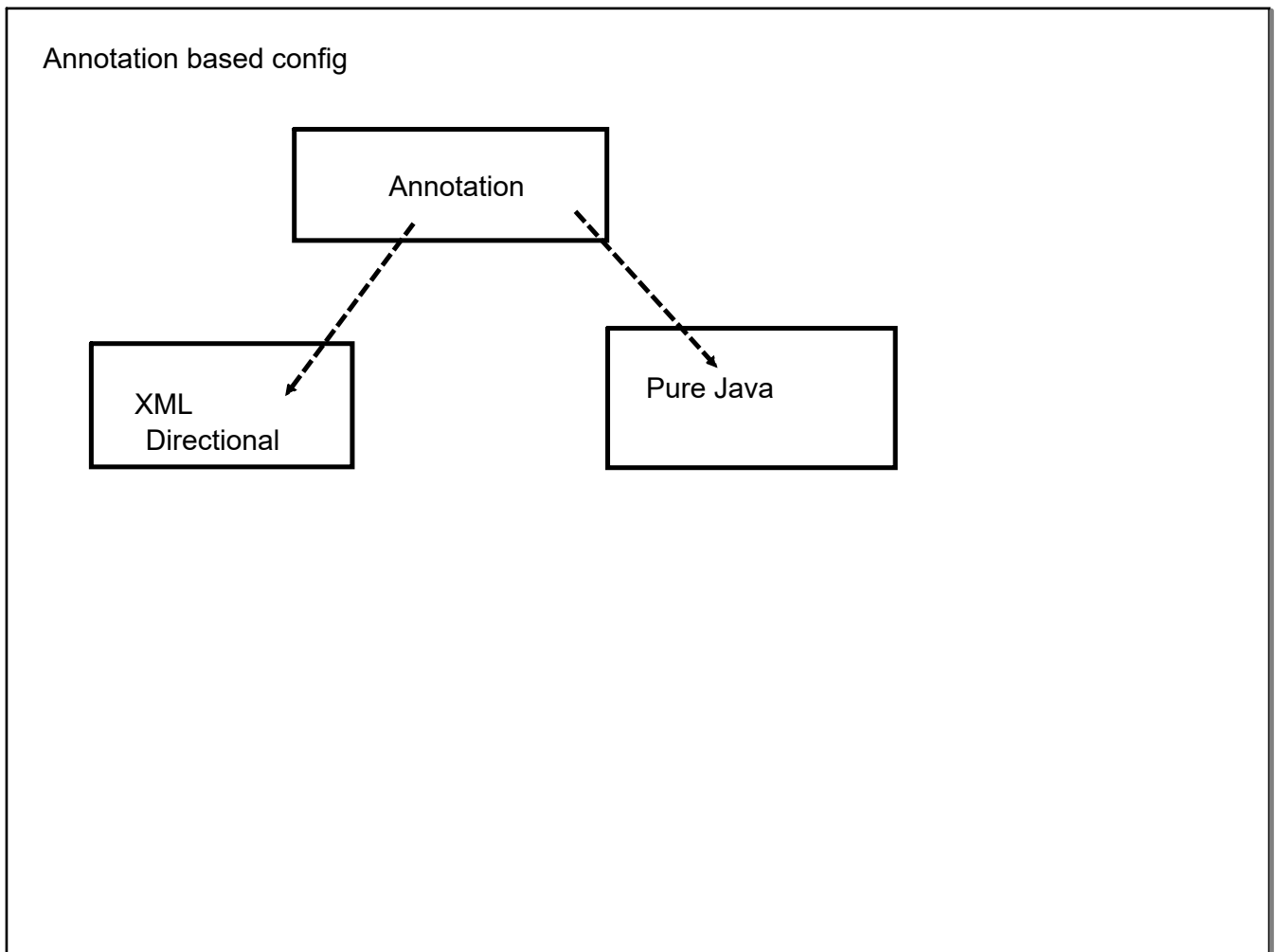
global-session



Life cycle hook method

1. any name
2. any access modifier
3. not static
4. they may return values but can't capture
5. No parameter

Prototype : Spring container does not maintain lifecycle



@Component : informing spring to create and manage bean of that class

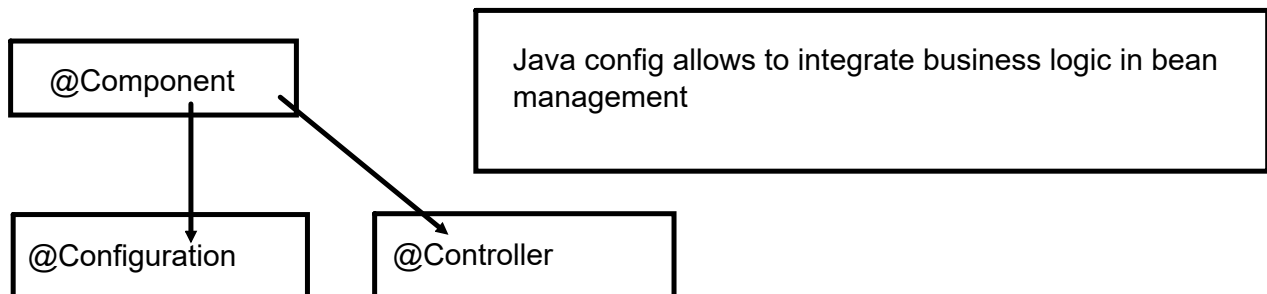
Every bean must be exposed by an id (Class name (with first char small) is default id)

Annotation based DI

1. Constructor
2. Setter
3. Field

Java Based Configuration

XML file is replaced by a Java Config



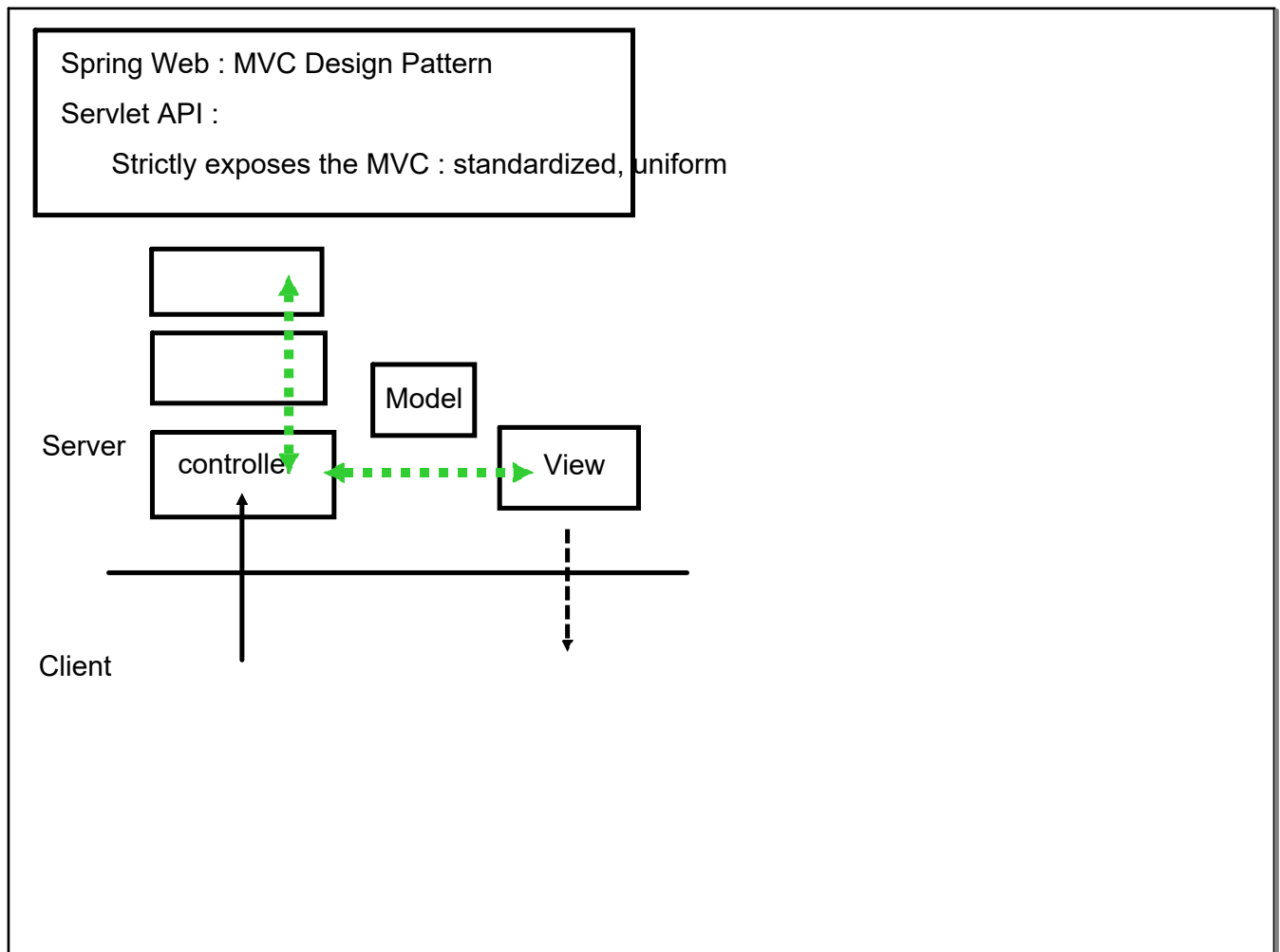
Developing Web Based Application using Spring Framework

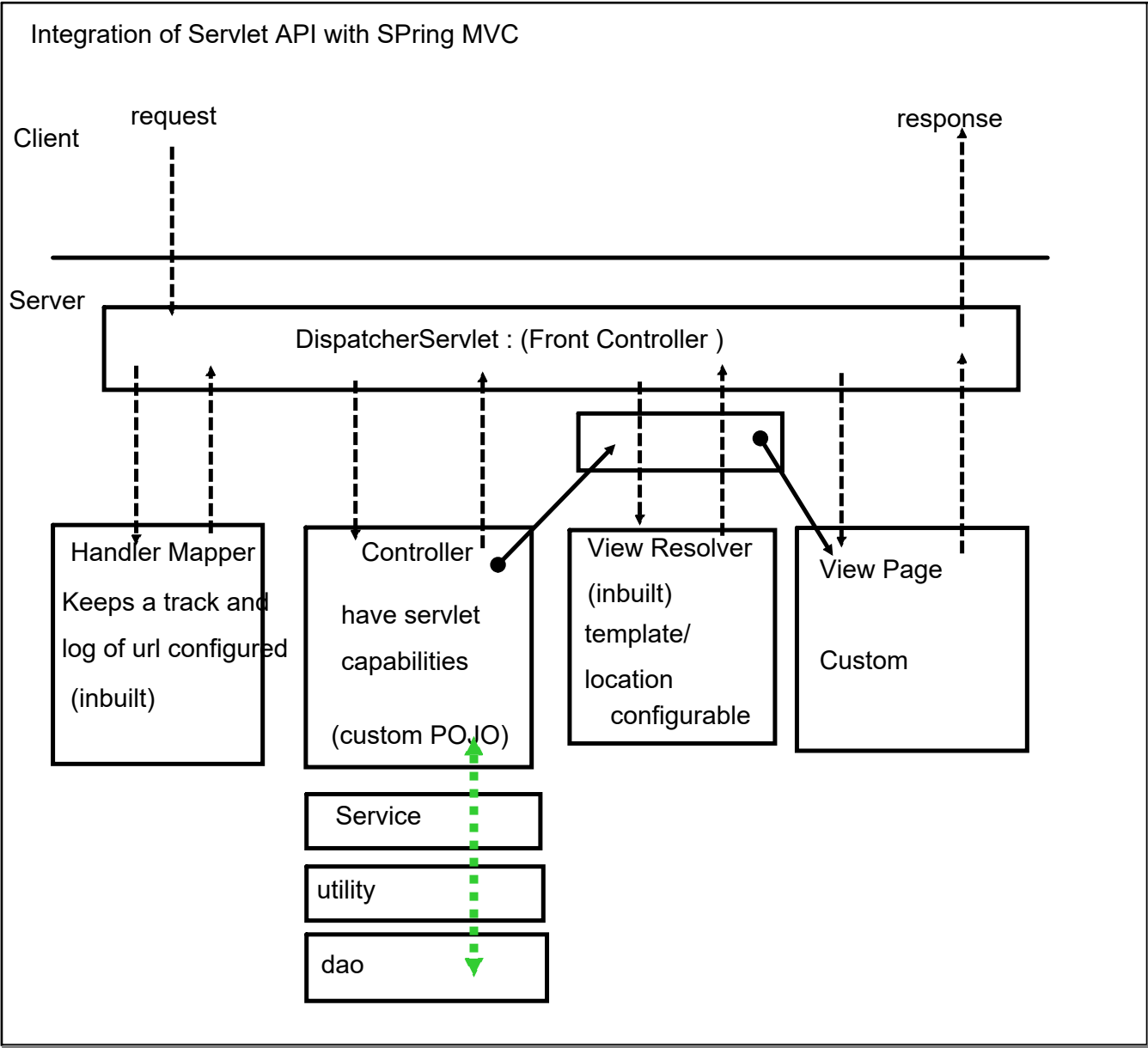
spring web-mvc
spring-core

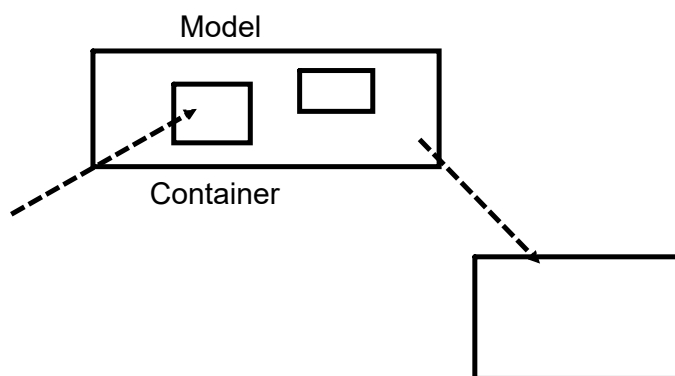
Spring also uses Servlet-API (Servlet Spec)

spring container

highly abstraction : POJOs







Spring MVC:

Multiple View Templates :

Traditional Default : JSP + JSTL (lib added for jsp+jstl)

Thymeleaf

FreeMarker

Tiles

Velocity

Mustache

Two configurations

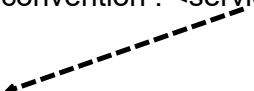
1. Servlet API based : Need register and configure the Dispatcher Servlet
2. Spring config for that servlet, helpers for servlet

xml file needs to tightly bind with servlet

naming convention : <servlet-name>-servlet.xml

eg:

dispatcher-servlet.xml



View Resolver : Creating and exposing a bean of View Resolver

Controller revert back : Name of view page

eg: "my-view"

/WEB-INF/views/my-view.jsp

URL Mapping must be unique across the application

Class level URL Mapping can be done

Multiple Urls mapped to same method

Default mapping

Fallback

@RequestMapping() maps all http verbs by default (GET/POST/PUT/DELETE)

Spring Web MVC project configured using java servlet

XML files would be replaced by Java classes

↑ web.xml : Java class

↓ dispatcher-servlet.xml : Java class

a config file for
that servlet

web.xml ~

Register the DS : inherit a spring framework class

Other config to be done by overriding the abstract method

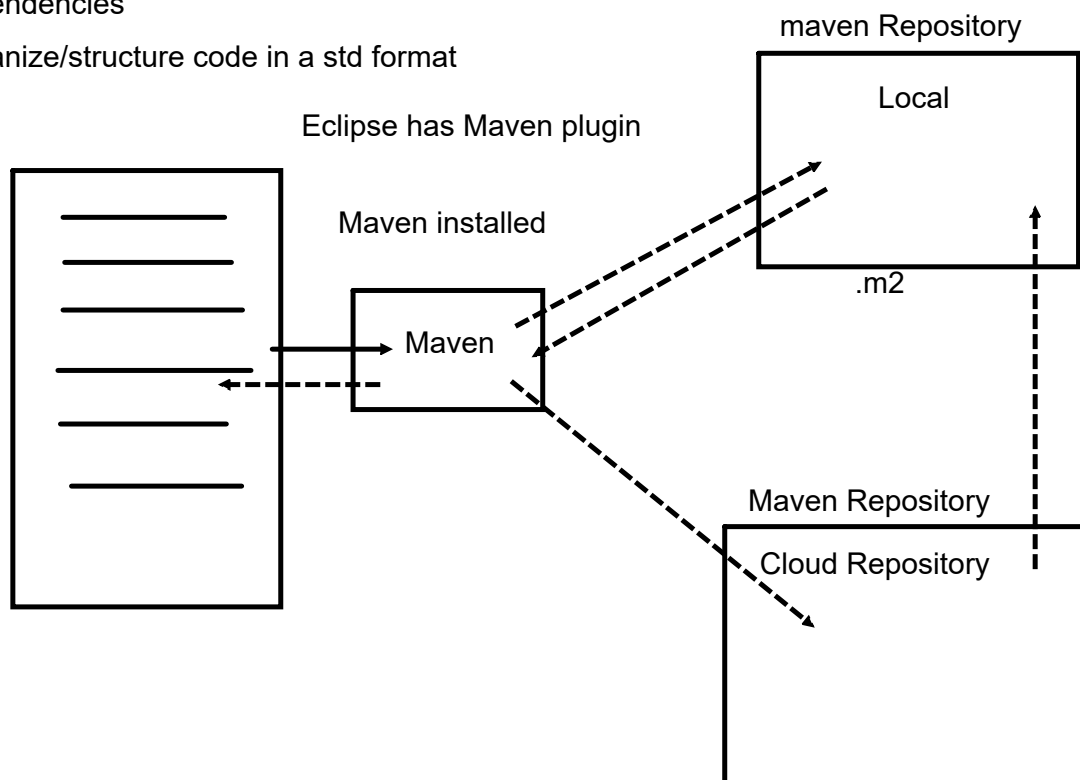
Traditional way : jar file management

Tools Project Management Tools

Ant, Gradle, Maven

1. dependencies

2. Organize/structure code in a std format



Each Maven Project is identified by GAV coordinates

GROUP : organisation (reverse domain)

Artifact ID : project name

Version : version of project

Dependency:

1. spring webmvc
2. servlets
3. jsp

plugin :

maven war plugin

maven webapp archetype project
changed java 1.5 to 1.8
added the Server Runtime Support
Added maven dependency
deleted the web.xml
add maven war plugin

Forms in Spring

Spring provides library that exposes tags for form handling

Spring Form tags :

It can be mapped with Model Object

Inherent Security : CSRF attack

Validation

Client Side : Javascript

Server Side :

Java Validation API: Validate the java object

Just a spec (interfaces) but no implementation is there

Most Popular implementation : Hibernate validators (not ORM)

Native JAVa Validation API :

Implementation Annotation not recommended : prevent vendor locking

Added dependency hibernate validator
Decorated the model flds with validation annotation
Need to tell spring to validate and get the result
need to check the result and decide
show the message

Model classes shall not use
the primitive type
wrapper types

data -----> String

1. eg _____First_____

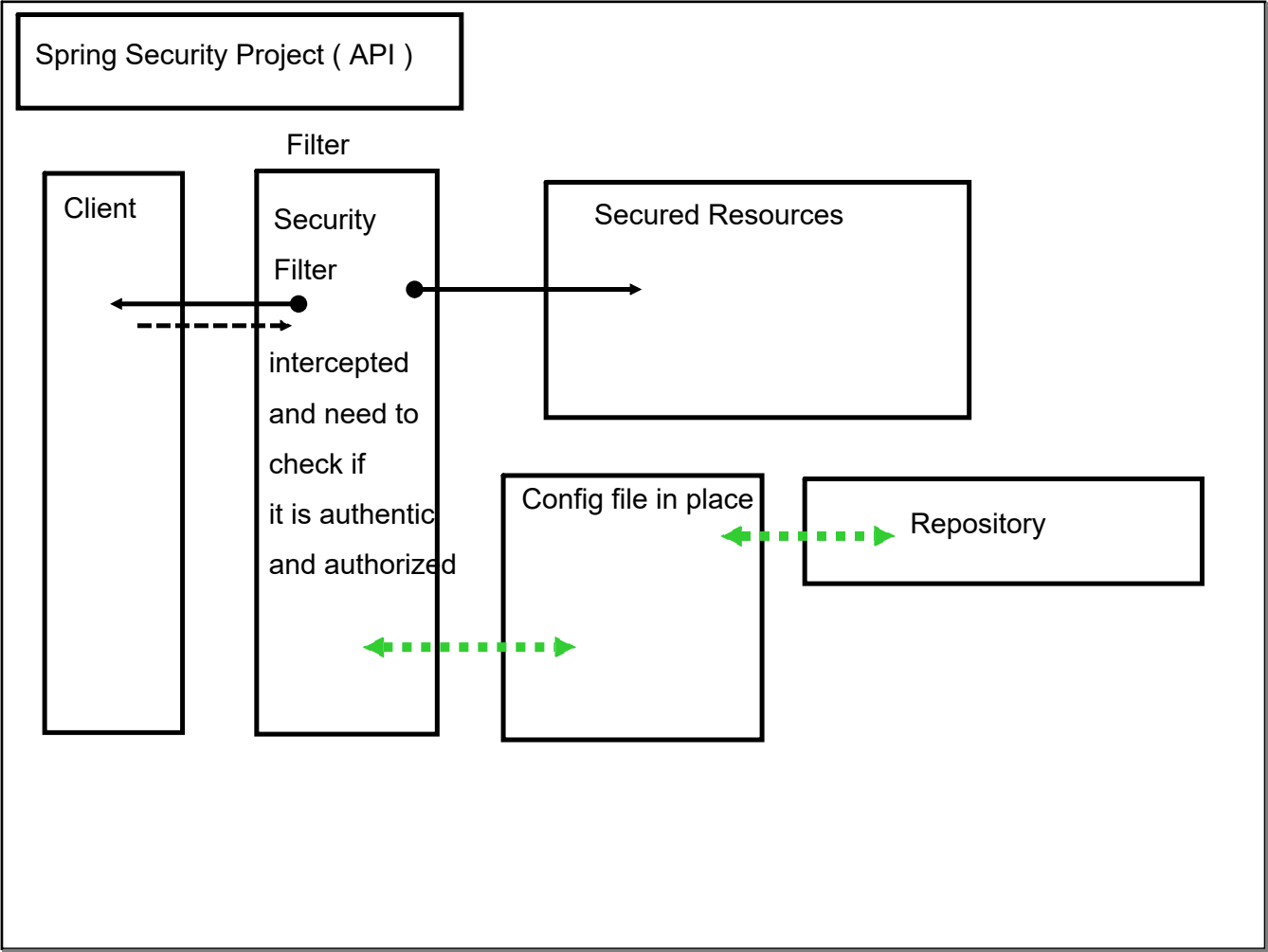
2. Empty string : NULL

proxy method

Custom Validation Annotation

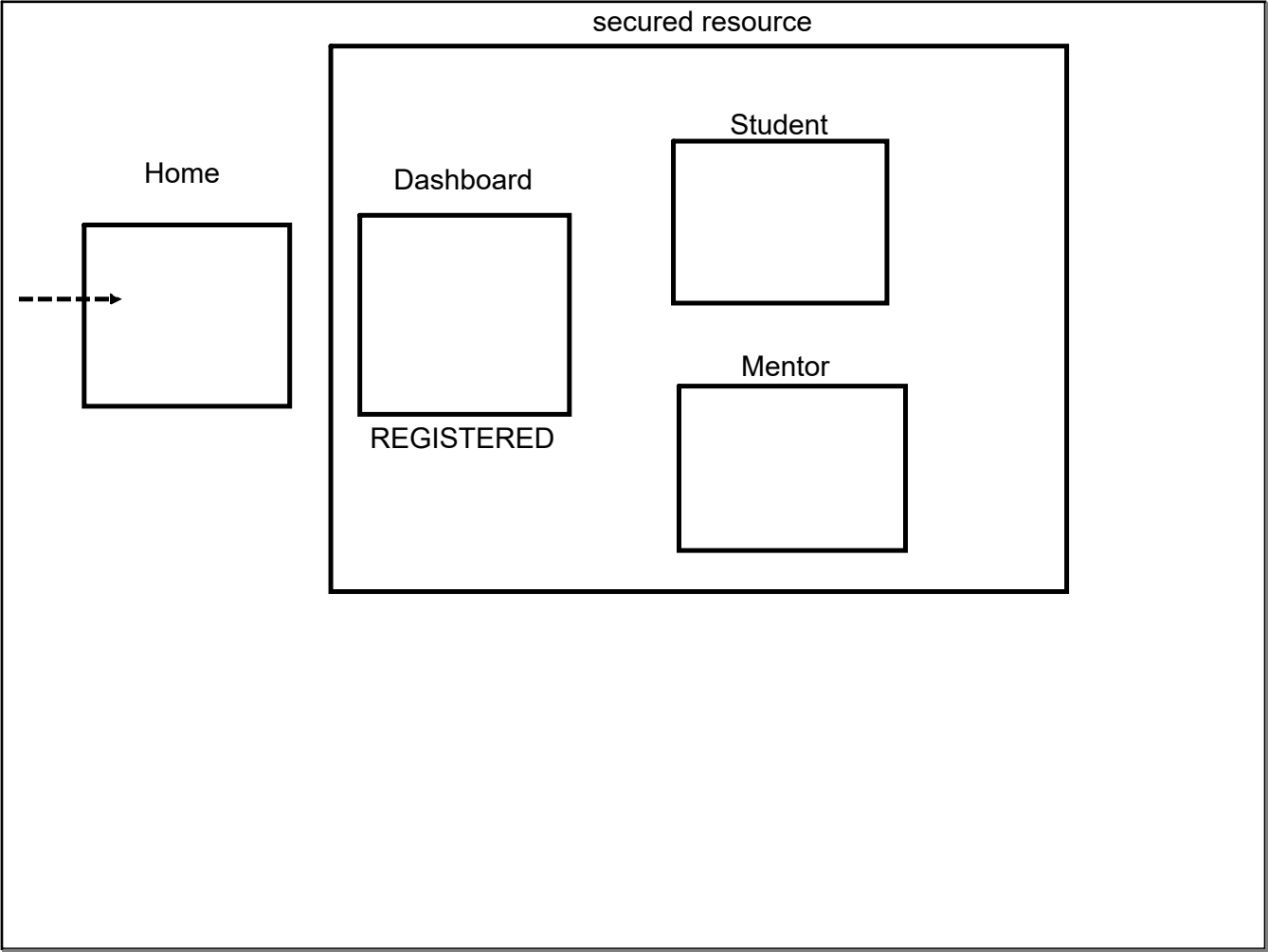
Create an annotation

interface



Confirmation of all these are through config file

1. if request is not of secured resource, direct access
2. is for secured resource : filter exposes a login form (inbuilt login form provided by spring security API)
3. Credential submission is confirmed by config file, reverted back to filter



Dependency :

- spring-security-web (filter)
- spring-security-config (add config)
- spring-security-taglibs

1. Initialize the security Filter

Authentication can take place in three different ways

1. httpbasic : instruct the browser to generate a login dialog-box (not-recommended)
2. formlogin : (default) default login form / customization
3. basicauth: Token based (Data-driven) REST APIs (Stateless)

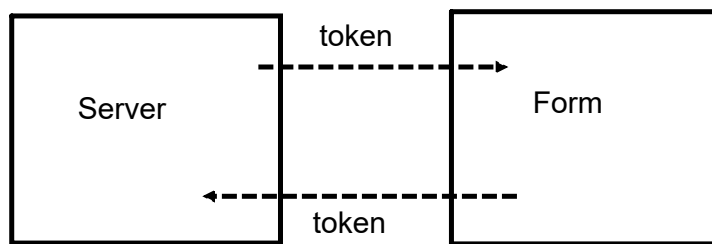
in case of bad credentials /custom-login?error

Spring form : CSRF attack

spring form : token system

+

spring security

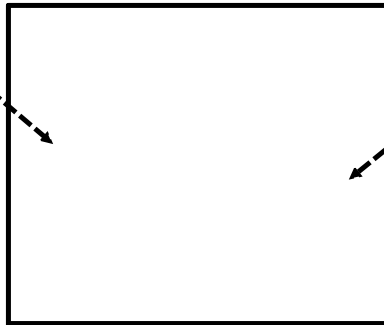


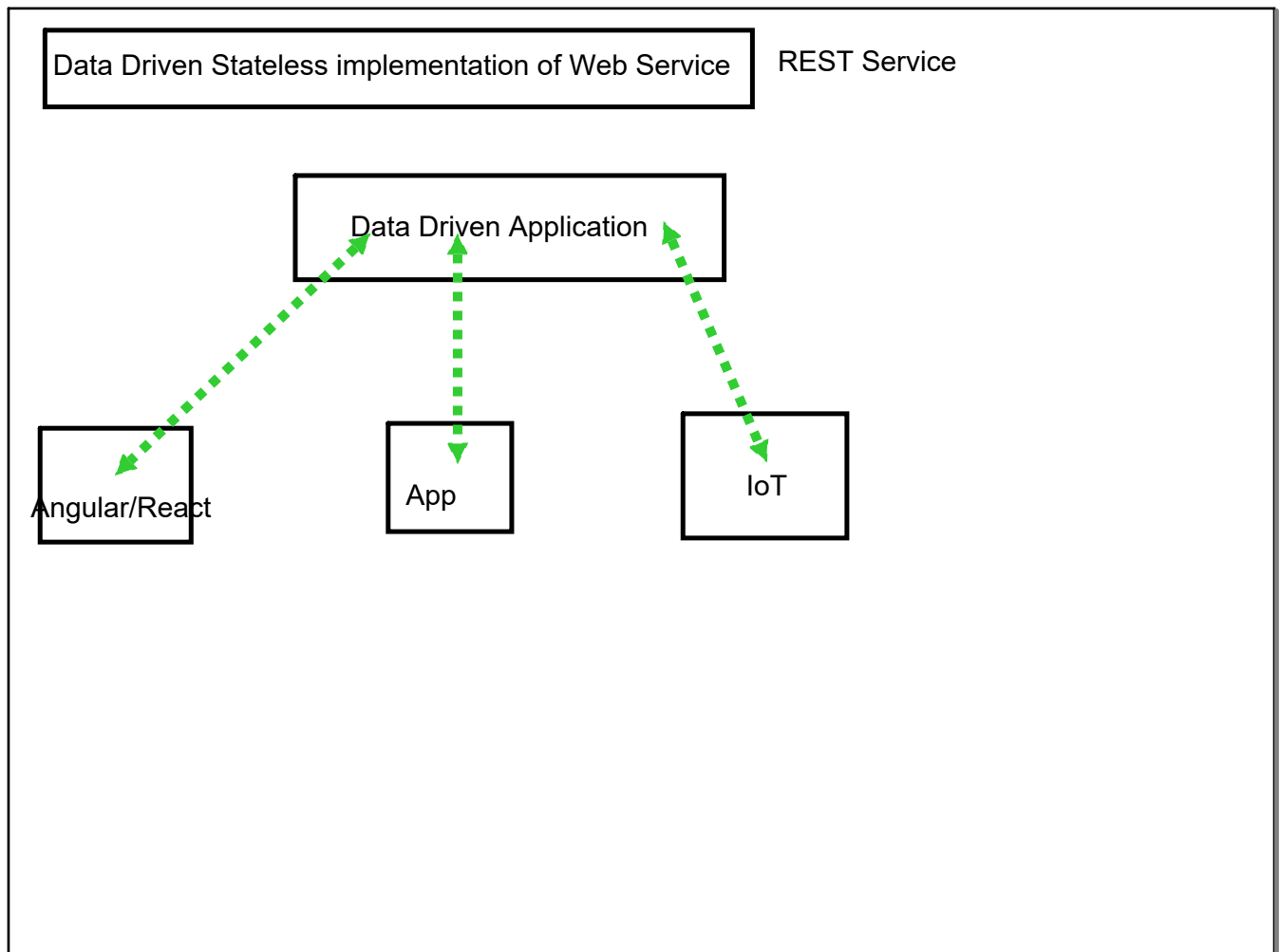
logout : form created , with a submit button (logout)

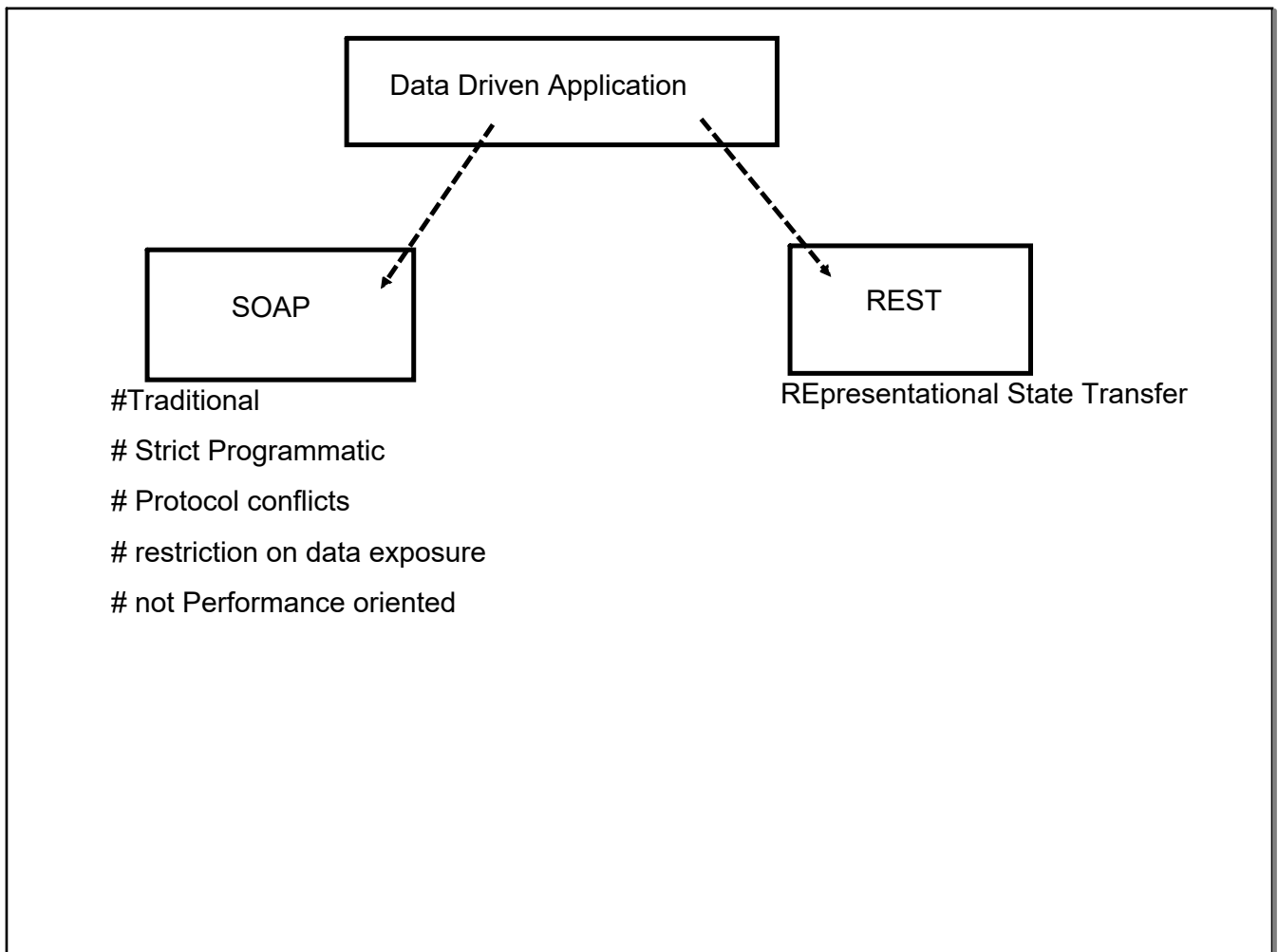
Principal : an object initialized with logged in user details

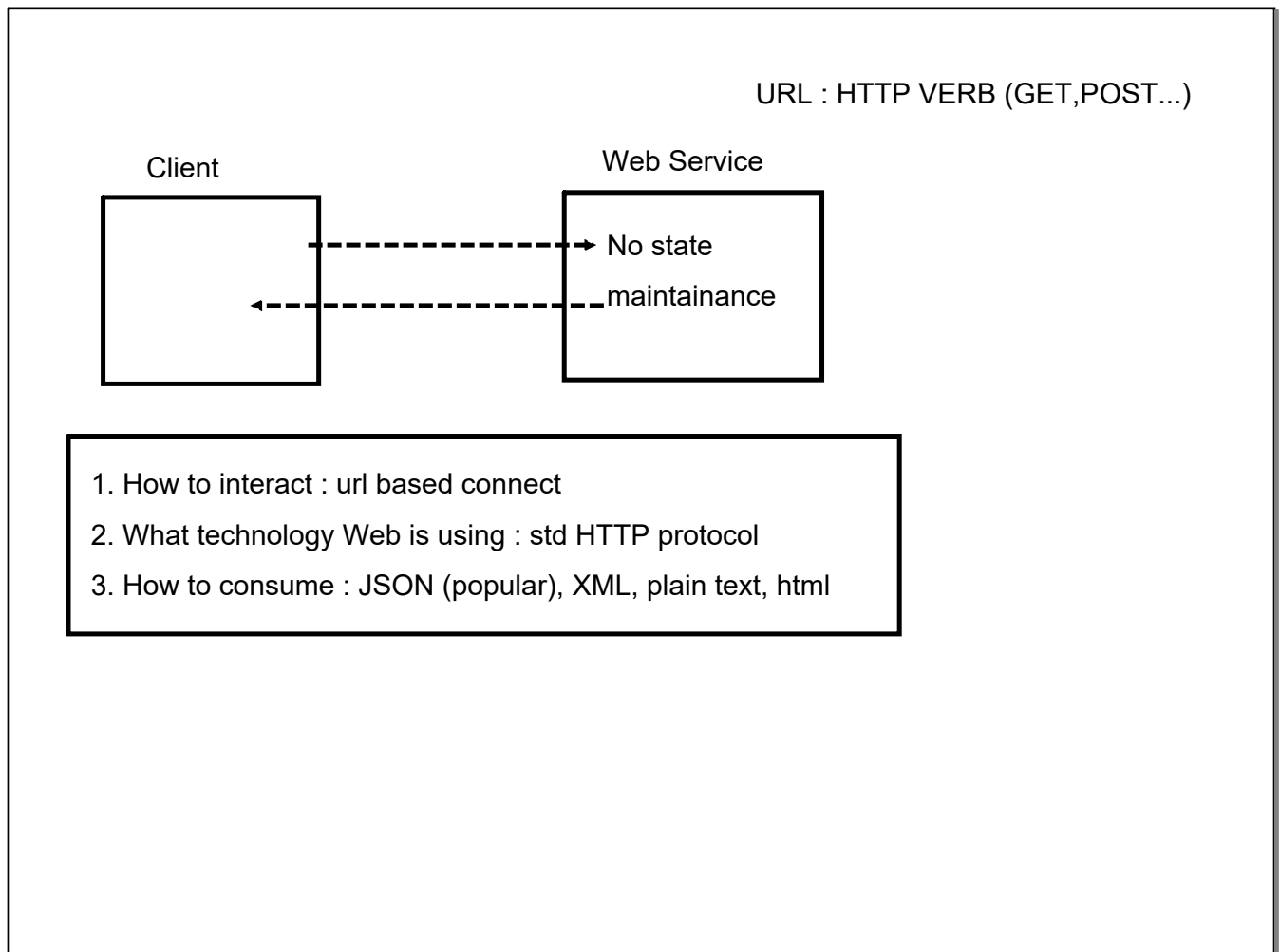
Mentor

Student









Spring : Allows to create REST Controllers

JSON <-----> POJO

mapping

Library : Jackson-databind project

@RestController : Auto consume the mapping api:

Mapping uses getter/setter method

Request

Body

Spring Boot

Framework/Tool : to allow to create(boot),initialize and develop a spring application quickly eliminating a lot of boiler plate code

Similar in performance

Advantages

1. Dependency management
2. Configuration
 - a. More abstract annotation (highly specialized,curated)
 - b. Auto Configuration : default set of config is provided based on dependency
 - c. Custom config : outsource and place the config detail in text file (properties)
3. Tends to create a standalone/self sufficient spring application
 - NO Specific IDE
 - Integrates MAven
 - Web Server

Dependency Management

starter project (projects)

spring-boot-starter-parent project

Autoconfiguration, property file, create a boiler-plate

starter-spring-mvc
project

curated list of lots
of dependency

Database

by default all starter projects
will have ver same as parent

Online portal : spring initializr

spring initializr plugin :

mvnw : linux/mac

mvnw.cmd : win

batch file : mvn cli command

Live Reloading of server : devtools

Application Maintenance/Devops : Actuators

=> Application Architecture : Multi-layered architecture

Controller

Service

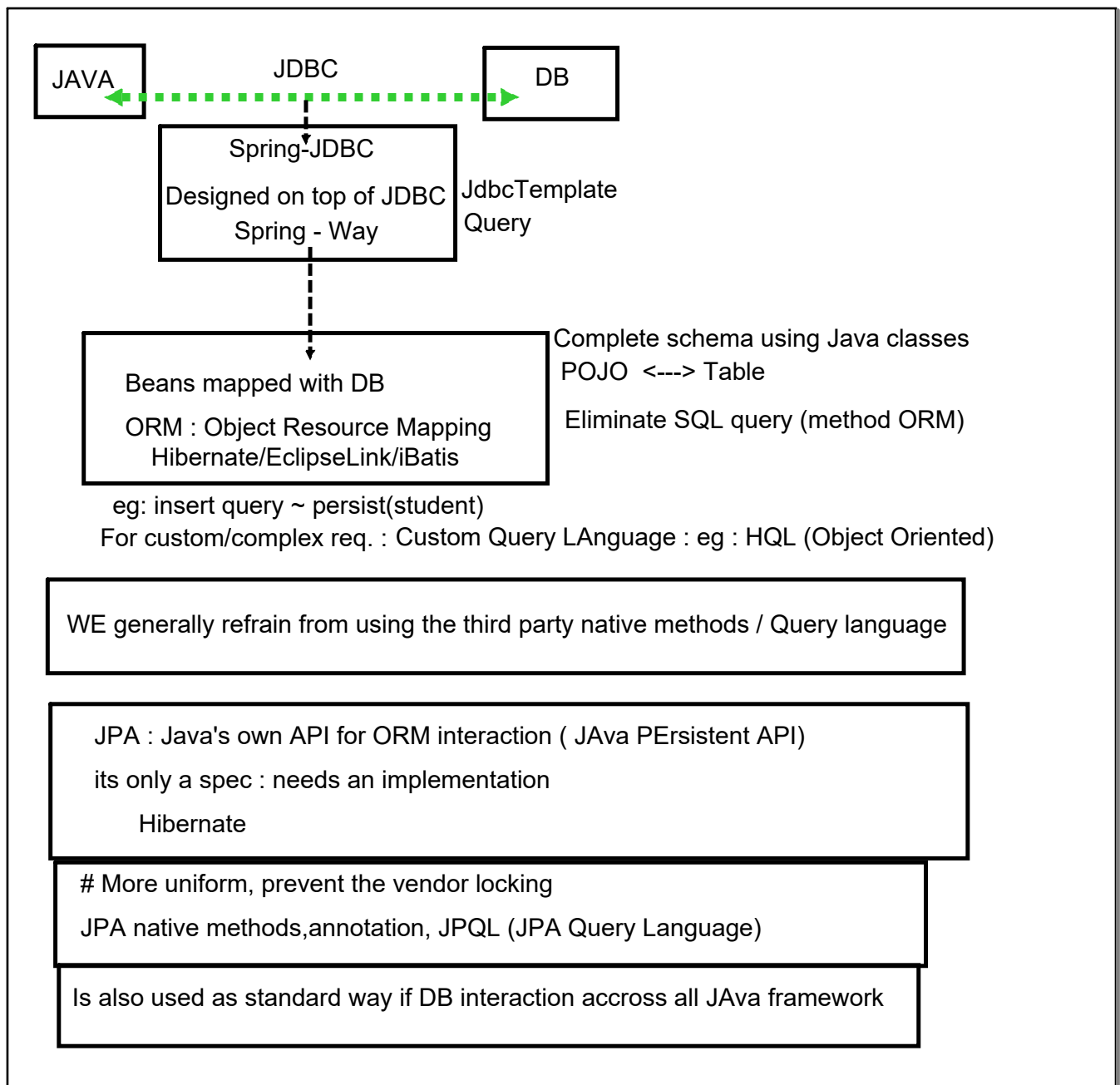
DAO

Exception

Entity

Dto/model

=> Persistence : backend DB (mysql) ~ H2 (Embedded)



Specialized project (module) to ORM based DB interaction using JPA spec
spring-data

More abstraction has been added

spring-data : SQL/NO-SQL

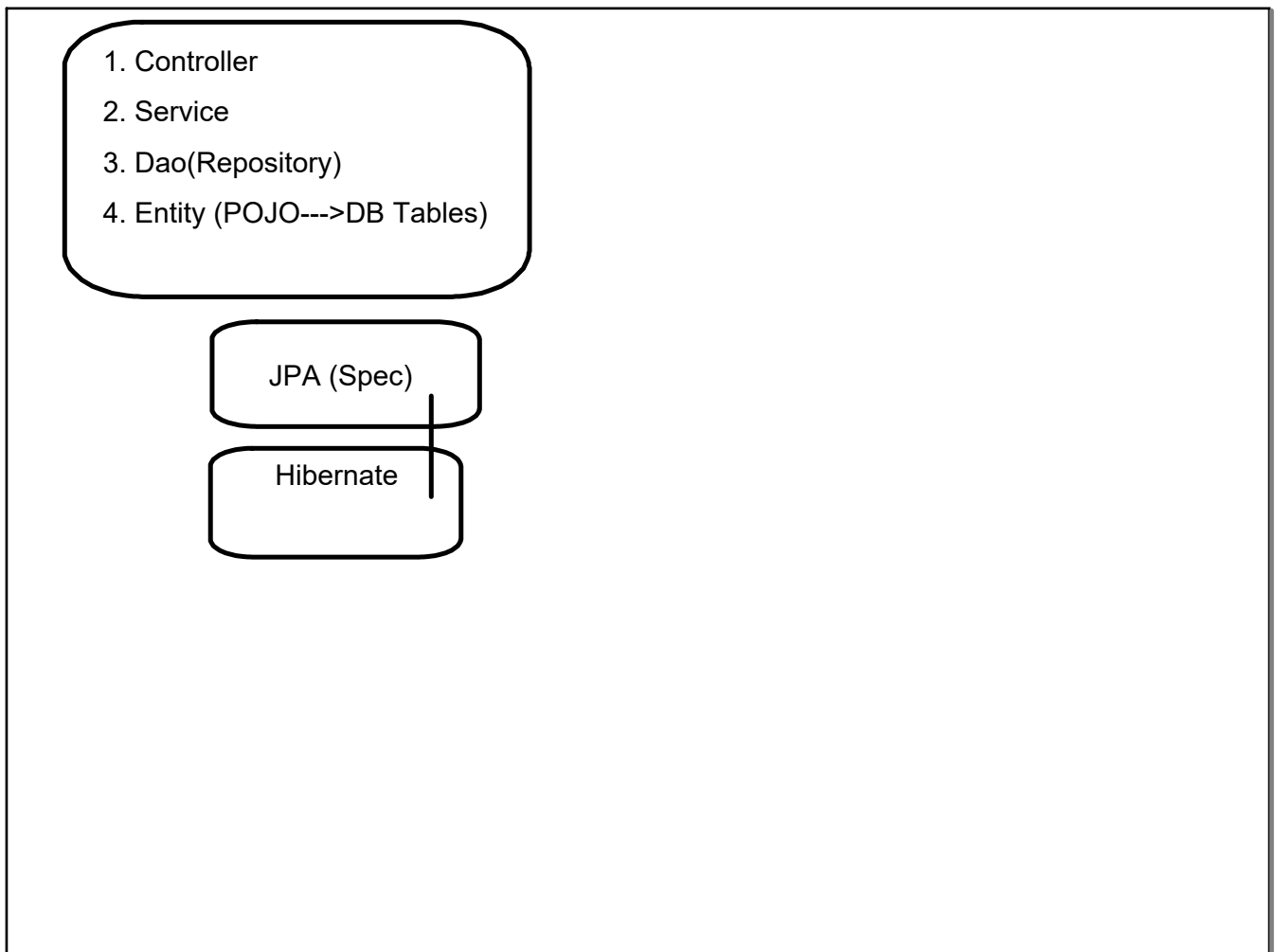
getter/setter

Default Constructor

All Arg Constructors()

ToString()

need to have lombok plugin
configured in IDE



For DB Interaction:

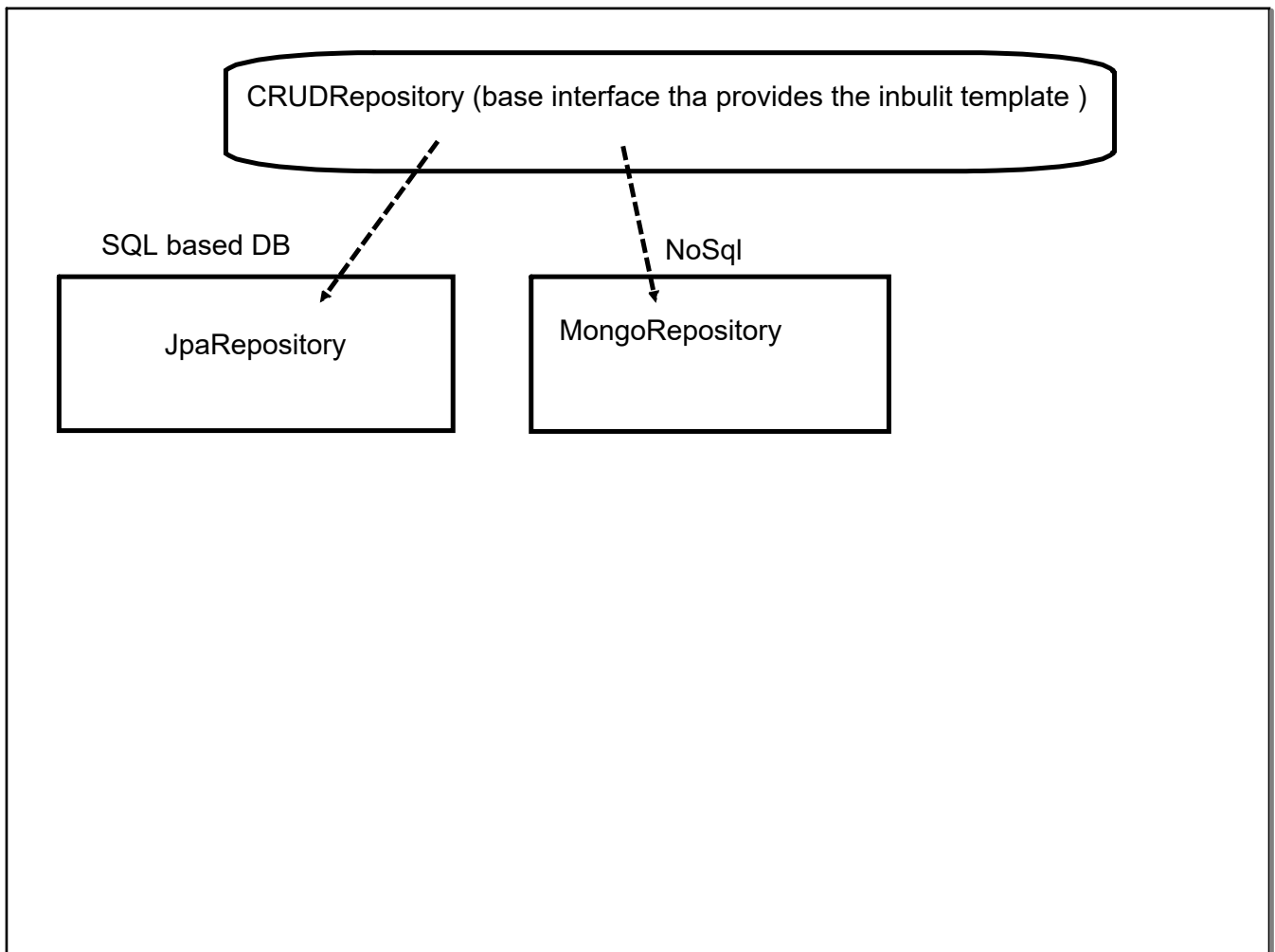
student : fetch all : select * from student;

product : fetch all : select * from product

<entity> : fetch all : select * from <entity>

All crud functionalities same template

Pass on the Entity info , it will come back with all std CRUD functionalities



Expose REST Endpoints for Basic CRUD Functionalities

Best Practices:

fetch all records : /getallrecords
fetch a single record based on id : /getbyid
add a new record : /addstudent
edit an existing record : /editstudent
delete an existing record : /deletestudent

1. Rest Endpoint are generally exposed on the basis of entity
2. plural form of entity : eg student~students
3. stick to a common end-point but we'll change the http verb

Eg:

fetch all records : /students (GET)
fetch a single record based on id : /students/{id} (GET)
add a new record : /students (POST) | data as a part Request Body
edit an existing record : /students (PUT) | data as a part Request Body
delete an existing record : /students/{id} (DELETE)

