**Books** => Purpose of life is to have a purpose - https://rickwarren.org/purpose-driven-life

=> Head First Servlets by Bert Bates, Kathy Sierra, et.at

=> How to program -by Paul Deitel and Harvey Deitel

50 Algorithms, Hackerrank - Badges, Certifications, PhD

**Sir vision (opportunity):** [**JavaFreshers**](https://javafreshers.com/) **(outside the office hours (part-time) [if interested])**

**1.vision for freshers**

**2.software development - ((demo video) if accepted then code )=> will be added to the library**

**3.College projects (innovation)**

**4.US college assignments**

**==>>{diff b/w landline and mobile: wired , wireless,**

**no other features only calls, all are available**

**fixed , movable, no storage: phn dic, storage}**

Types of servers and their configurations:

Virtualization (how software is converted into an image)

Y JRE called sandbox or Blackbox

Limitations of Monolith:

1) difficult to integrate

2) scalability, scaling is of two types-horizontal, vertical

3) not easy to upgrade

4) SOA- Service Oriented Architecture

Monolith ex: SAP company- ERP (Enterprise Resource Planning software)

-> Interoperability: communication b/w devices or softwares

-> SOA principles are realised and implemented using web services

-> webservices types- soap services and rest services

**SOA:**

java; cmd:

set classpath=%CLASSPATH%;add the path---> to access the files in different folder or drive

javac -d . A.java-------> to create a package using javav cmd

d: to change the drive

tree\f---> shows heirarchy in cmd

decompiler--> javap (for all packages) java.lang.String (gets all classes and methods in the package)

**Maven project:**

add spring-core, spring-context, asm, cglib, spring-bean dependencies

logging using java.util.logging package learn instead of System.out.println

To build a maven project in cmd: mvn archetype:generate (command)

ctrl+space bar--> to add default constructor

shift+alt+s---> to add getters,setters,toString,parameteralised constructor etc.... or go from source by right click

**Spring:**

Spring supports two dependencies: constructor and setter (setter injection or property injection)

---->Spring is non-invasive

Factory means (object creation)--> abstraction, static methods

Spring best practice---> it supports interface object implementation (object into interface)

==>>BeanFactory and ApplicationContext are the two containers of spring

Logger logger=Logger.getLogger("filename"); (in place if sysout) from the package java.util.logging.Logger;

Default bean scope is singleton => [Bean Scopes :: Spring Framework](https://docs.spring.io/spring-framework/reference/core/beans/factory-scopes.html)

idref instead of ref finds using id of the bean

by default beans are lazily loaded so if don't want that then lazy-init as false

for bean factory beans are lazily loaded, and in application context singleton beans are eagerly loaded

xml is verbose

Annotations concept was introduced in the version 2.5 in spring

DAO-Data Acess Object

JWT-JSON Web token

RMI-Remote Method Invocation

POJO-Plain Old Java Object

SOAP-Simple Object Acess Protocol

REST-Representational State Transfer

->Rest uses jackson to convert the files into JSON

Bios-Basic Input Output System it will boot strap the system

mvc-Model View Controller

ORM-Object Relational Mapping

{{Servlet API and JSPs

Servlet Library

Bean Life Cycles

Synchronization vs Synchronous communication(one sends other is blocked)

-how to mappings {one to one, one to many, many to one}, relationships and associations

Garbage collection architecture in jvm

}}

**Annotations learned:**

@Component

@Bean

@Primary

@Qualifier

@PostConstruct--}

@PreDestroy-----} JSA

@Value

@Configuration - Java configuration

@Service

@ComponentScan - Scanning the packages

@Autowired

@PropertySource

@Override

@Repository

@Controller - MVC Controller classes

@Lazy alongwith @Autowired

@DependsOn ("bean on which this bean depends to be created before this bean is created")

@Scope("prototype")

@Lookup indicated a method as lookup method. It is best used for injecting a prototype-scoped bean into a singleton bean.

@Profile - Dev Test Prod

@Import({JpaConfi.class, SchedulerConfig,class})

@ImportResource({"spring-context.xml"})

@RestController = Controller+ResposeBody- <body>, http headers other than GET you will have body (pay load)

we can send or write into renspose body using ResponseEntity class

controller -> commit or render views using view technologies(HTML or JSP or Tiles or Velocity......)

@GetMapping - handler resource for HTTP GET requests

input through get method: 1.query string (?name=value) is sent via requested url

2.main url followed by a forward slash(/)

@PathVariable to receive this input

@Data---------------}

@NoArgsConstructor--} from Lombok

@AllArgsConstructor-}

@Entity-}

@Id-----}from persistence

@RequestBody

-@GeneratedValue - to have the system itself to provide the primary key

@PostMapping - handler resource for POST requests

@PutMapping - handler for PUT requests

@DeleteMapping - handling for delete

@RequestMapping -

@SpringBootApplication (automatically generated when spring application is created) - annotation given to main class=> it will scan only those components/classes which are in that package or in which the spring boot application is there

ResponseEntity - 4constructors 1.arg is status, 2.body and status, 3.header and status, 4.body, header and status

Spring-Configuration files+(Autoconfiguration+Staters+Actuators+Devtools+Cloud tools+CLI(Command Line Interface))=Spring Boot

Spring boot= Ten years of Spring experience -DRY(Do not repeat yourself)

Spring boot= Agile Development+dependencies starters+inbuilt tomcat server+In memory database-H2db

Spring boot <-> Microservices need -> RESTful (These 3 pillers are supporting the enterprising applications)

Swagger - API will be well documented, it tells user what is needed to fill in that place or what is the value needed to fill

Client-> end user-> Browser(user-agent, proxy)-> application & Server communication through http headers- request and response headers.

Most common headers- content\_type, expires, connection, age, cache\_control

-> Autowired

-> Autowire

-> AutoConfiguration

mvn spring-boot:run--}to run from terminal

/mvnw spring-boot:run}

Callback methods are also known as hooks

Maven supports 3 builds-Jar, War, pom

Spring supports only Jar and War

Maven scopes: compile,runtime,system,import,test

Model->Repository->Service->Controller

@RestControllerAdvice

@ExceptionHandler(value=NullPointerException.class)

@ResponseStatus(HttpStatus.----)

@Profile("dev")

Profile (It is all about application-properties.profile=dev) 3 types: Development, Test and Production

application-dev.properties

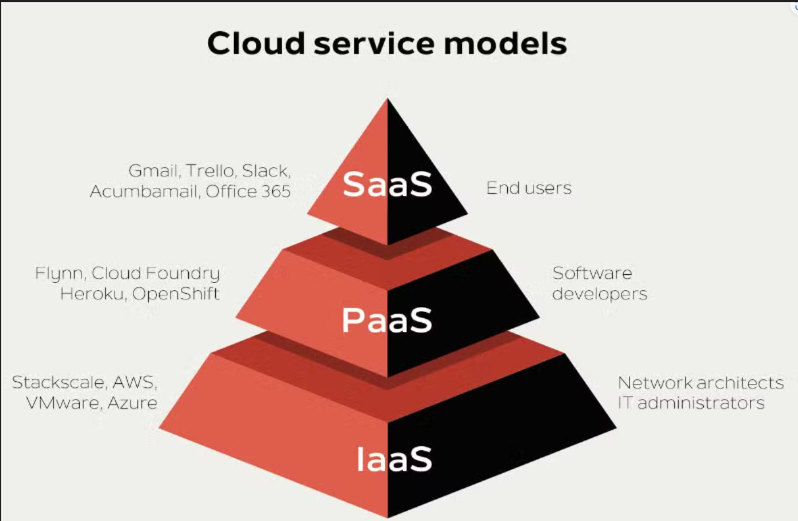
application-test.properties

application-prod.properties

**Microservices: :>**

Service Discovery: register all the services and discover them (eureka)

API Gateway is also known as Edge Service



Cloud service models

Cloud Gateway ports:

@EnableFeignClients

@EnableConfigServer

@Value

@EnableEurekaServer

----------------------------------

**Circuit Breaker intro:**

@CircuitBreaker

->Three stages in circuit breaker: Closed, opne, half open

Gateway load balancing:

->Sleuth: [Application name, traceId, sparId, export]

Application name: the name we set in the propertien file, and can be used to aggregate logs from multiple instances of the same application.

TraceId: id thats assigned to a single request, job, or action. Something like each unique user initiated web request will have its own traceId.

SpanId: Tracks a unit of work. think of a request that consists of multiple steps. Each step could have its own spanId and be tracked individually. By default, any application flow will start with the samw TraceId and SpanId.

Export: This property is a boolean that indicates whether or not this log was exported to an agrregator like Zipkin. Zipkin is beyond the scope of this article, but plays an important role in analyzing logs created by Sleuth.

@EnableWebSecurity

@EnableMethodSecurity

@PreAuthorize

**JENKINS:** It is an open source automation server. From planning, coding, building, testing, releasing, deploy, operate, monitoring everything is happed=ned by jenkins now it also has dockers integerated.

It also provides feedbacks.

Jenkins-pipeline- Script based- Groovy- and UI based

pushing your sourcce code to repo-> run quality gates(clean code)-> Testing (all types of testing all types of reports)-> pushing -> deploy/install-> release

**Docker**=> java-> jar file -> images-> container

Docker steps: 0. create a new folder and switch

1. create a Dockerfile without extensions

2. define the bins/libs

3. build an image with Dockerfile

4. run the image file to instantiate the container - runtime instance of an image which is based on Dockerfile specifications.

**Docker commands:** website::>> https://www.hostingercom/tutorials/docker-cheat-sheet

docker version

docker images -> provides the details of all the images in the docker

docker ps -> displays all containers short display

docker ps -a or --all -> lists all containers long display

docker pull nginx:1.23

docker run ngins:1.23 -> blocks terminal

docker run -d or --detach ngnix:1.23

docker logs <container id>

docker stop <container id>

docker run -ti --rm <container-id> bash

docker exec -it [container id] bash -> to enter a running container

docker run -d --publish 9000:80

docker run -d --publish 9000:80 nginx:1.23

(or)

docker run -d -p 9000:80 nginx:1.23

docker run --name web-app -d -p 9000:80 nginx:1.23

docker build -t node-app:1.0 .

docker tag hello

docker push anilboppuri/dockerhub:hello-app

**KUBERNETES:**

commands:

minikube start --driver=docker

kubectl apply -f nginx-dep.yaml (or) kubectl create -f nginx.yml

kubectl get deploymentsn / pods

kubectl apply -f ngnix-ser.yaml

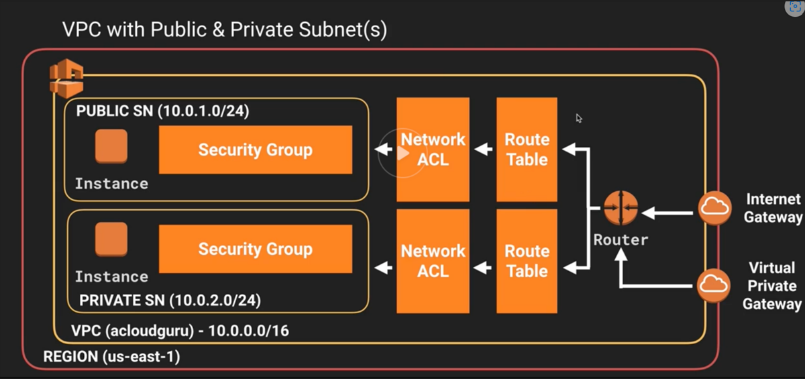
kubectl port-forward service/nginx-svc 7080:80 then next go to browser or curl localhost:7080

kubectl get pods

kubectl get svc <springboot name>

kubectl exec -- bash

JMS- Java Message Service:-> [Java Message Service (JMS) (oracle.com)](https://www.oracle.com/java/technologies/java-message-service.html)



VPC with public and private subnet:

**AWS (Amazon Web Services)**

-> EC2 compute service: creating instance, security groups, keypair, inbound ports, SSH, putty on windows, VPC, subnets, connecting through RDS, start, stop, terminate ec2.

-> IAM: Users, Groups, rules, Policies

-> S3: Bucket, how to create bucket, upload files, giving public acess to files, ACL script difference between DB and S3

-> RDS: Launch RDS-MYSQL server, connect with MYSQL workbench on local machine.

-> Integration-EC2 with RDS service

=> **Steps to integrate RDS to EC2 service:**

1. create instance, create database in RDS

2. connect to RDS in the actions and networking at the instance running

3. then check if the connection is established in security groups

4. then connect the ec2 console and perform operations.

5. by using the command "mysql -h <url or endpoint> -u <username> -p"

6. enter password and start performing operations.

-> **Elastic Beanstalk**-> creating 2 roles one for service and 2nd for ec2 instance

-> **Loose Coupling**: Abstraction- Interfaces- Messaging systems helps to build loosely coupled applications.

Loose Coupling is a design pattern in SE that involves reducing the interdependencies between the components in a system.

It makes components independent, and systems flexible and easier to maintain.

->**Serialization:** Marker Interface-an interface which has no methods

->**Kafka configuration:**

1. specify the path of kafka bin in the environmental variables path

2. then in log.dir in zookeeper file in config folder in kafka specify the kafka bin path

3. then in log.dir under log basics line in server file in config folder specify the kafka-logs folder path (kafka-logs folder create a new one)

**HTML5, CSS and JavaScript**

->DNS Resolutions- hostnames to ip addresses

->TCP/TLS Handshakes

->Fetch webpage from server

->Browser parsers and renders the HTML

->Browser handles post-load user interactions with the page

CUI-GUI-> AWT, Swing-create content, how to handle events/actions and produce or process actions/events

applets=client side java programs on web, servlets=(rest runs on servlets)they run on server side

Html: https://developer.mozilla.org/en-US/docs/Web/HTML/Element

Semantic tags

Difference between Website vs WebApp

html-> navigation, JavaScript-> interactive, dynamic-> serverside

Process that happens behind:

**CSS**: propagate downwards

styles at three levels: element level- inline

page level- effect elements in a single page

external style sheet- easy to change globally to all html files, maintenance easy

will have least effect

selectors: syntax-<selector>{}

Universal selector: \* {}

Element/Type selector body,h1,ol,li etc..

Group selector div, span

Grouping selectors div p or div h2

ID selectors

Class selectors .classname

Attribute selectors

Pseudo selectors

Pseudo nth Child selectors

//Combining selectors

Pseudo element ::

Descendant selector p h3

Child selector >

Adjacent selector +

General sibling selector ~

Complex selector – compound

**ANGULAR:**

Day1:

1.Installayion and issues

2.Architecture + history

3.How angular works

4.Before sending first response Angular bundles all scripts in index.html

5.Important building blocks of Angular

{component, view, template, bindings(interpolation, event, property, two-way binding), directives(\*ngIf,\*ngFor, ngSwitchCase), services, http, Module, decorators(@)}

=>One directive can only be in one tag, can't place if and for or any two directives in a same tag.

6.Elements of a component; 1.selector 2.html file 3.css file 4.a class 5.test files

Day2:

1.Switch case, property binding innerHtml, ngClass

#<name>how to

2.---------

3.Define model classes

4.Define service classes

5.How to inject service classes and perform filters on the component view page

Day3:

1.Pipes 40mins

2.Bootstrap in angular

3.Forms

4.---------

Day5:

Promise vs Observable:

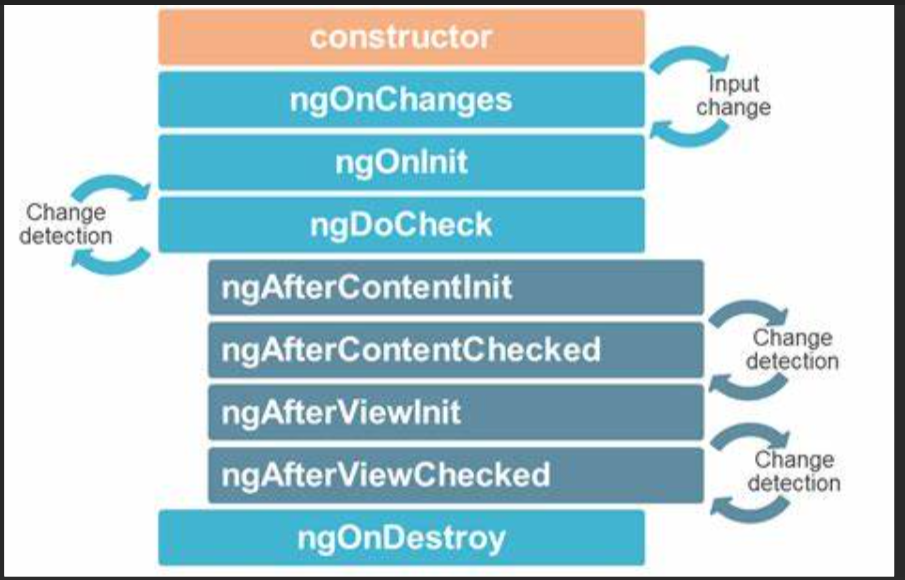
1. Promise- Produces in one large chunk 1. Observable- Produces in streams i.e. small chunks

2. Promise- Produces even if there is no consumer 2. Observable- Produces only when there is a subscriber

3. Promise- Communicates synchronously 3. Observable communicates in asynchronous mode

4. Promise- Blocks the client during communication 4. Observable- doesn't block client and needs a callback to invoke when it completes

Angular life cycle hooks:



1.constructor is not a life cycle method, but it is the 1st method that is called when an angular is started

2.then when constructor is called then ngOnChanges is fired. It is fired whenever a change happens.

3.ngDoCheck is also fired on every change detection.