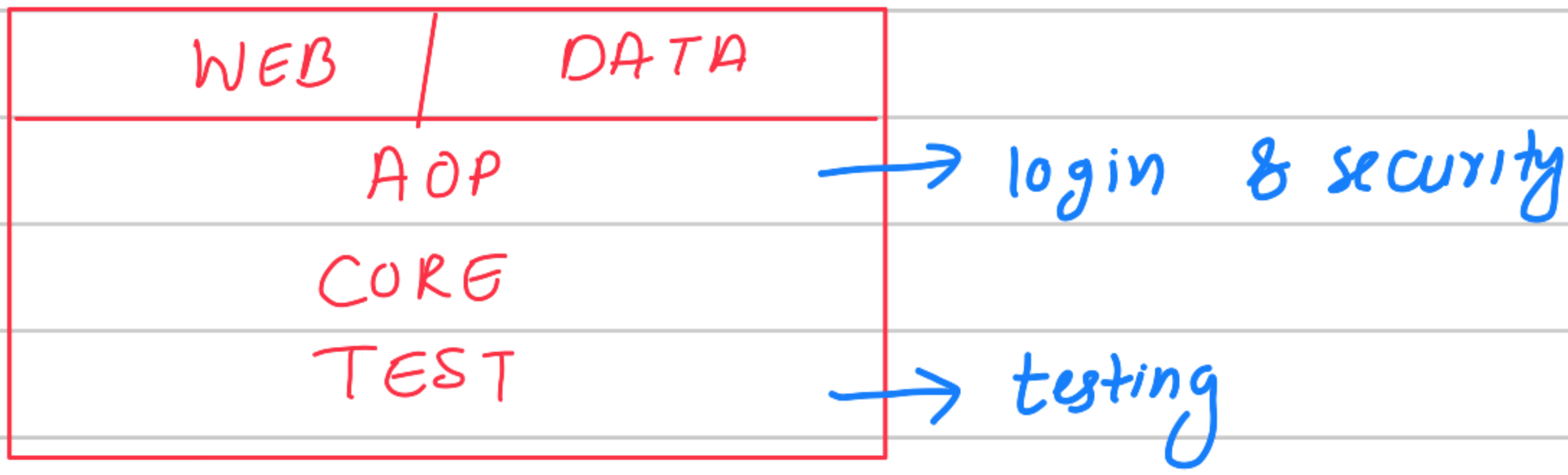
SPRING

↳ A popular framework for building Java applications.



SPRING BOOT → saves up the time & is used & comes up with

which JDK version?

SPRING BOOT

① Auto configuration
② Embedded Servers
③ External config.

H2 Database →

* relational database management system written in Java.
* fast, open-source, JDBC API.
* Embedded & Server modes; in memory databases
* Small footprint.

CRUD Operations

public interface DepartmentRep extends
CrudRepository < Department, Long > { }

entity id

JPA → Java Persistence API

JPA contains APIs for basic CRUD operations, pagination and sorting.
By using Jpa Repository we dont need to write DDL/ DML queries instead we can use XML.

Controller
→ Department Controller Class

Entity
→ Department Class.

Repository
→ Department Repository Interface.

Service
→ Department Service
→ Department Service Impl

Demo Project Application.

application.properties

server.port = 9090

spring.datasource.url

spring.datasource.driverClassName

spring.datasource.username

spring.datasource.password

spring.jpa.show-sql = true

spring.jpa.properties.hibernate.dialect=

spring.jpa.hibernate.ddl-auto = update

spring.h2.console.enabled

.

Type (Individual or Company)
false true.

Status active or inactive.

Role User or ADMIN

Spring ⟶ Springboot

↳ a small module.

Spring Boot makes it easy to create - stand alone production grade Spring based Application-

↳ Enterprise Application Development.

↳ No ORM

↳ Rapid development

* Convention over configuration

* Without Spring boot:

→ implement a rest api which provides the output.

↳ hello spring boot.

→ we need to add every dependency manually with its version.

→ we need to add server also to deploy our application.

Why we used Springboot

↓ Request

(first Controller) DispatcherServlet class ← we have to make

by mapping using invoke particular (controller) -

Hello Controller

Hi Controller

→ we need to configure a lot of stuff!

---

src /main /java

com. Bhawish Project

↳ HelloController.java

@RestController

↓

public class HelloController {

@GetMapping("/home")

public String home() {

return "Hello Spring Boot";

3

3

Version Configuration also automatic.

* Creating Project & opening

* Run Method

* Web Application Context → run methods
* Application Context → run method
* Configurable Application context

→ Container.

|

Objects

@ Component

↳ put this above a class and it will create an object for you and also push into the container by using spring container.

@ Bean → used to create an object manually

* Jar vs Fat Jar

↳ collection of many .class file.

* compiled → bytecode → .class file

Fat Jar

Jar

↳ collection of .class files

↳ compiled → byte code → .clay file

no main manifest attribute in spring boot Project_0.0.1

↳ It has no other files.

↳ It won't run

Fat Jar

↳ All files are include classes, pom, application.properties

↳ It can run.

↳ I send this to my friend and my friend can run it.

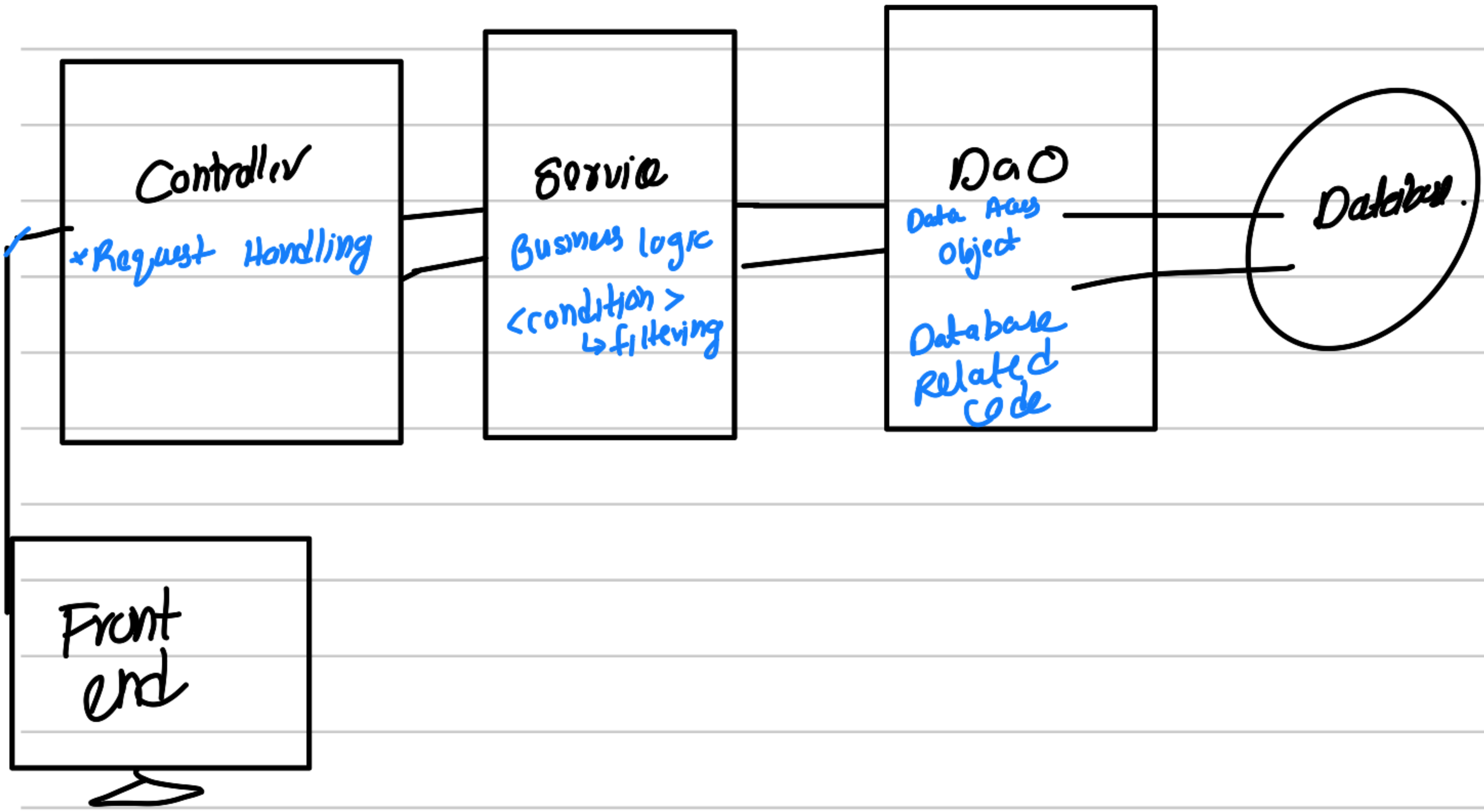
Application.properties is application.yml →

Spring Boot Starter Parent :-

↳ Version Management

Layered Archi tecture :-

GET POST PUT DELETE



DoA.java

public class Dao {

* List<IplTeam> iplTeam = Arrays.asList
(new IplTeam("MI", 5, "Ambani"),
new IplTeam("CSK", 2, "ABC", "R"),
new IplTeam("KKR", 2, "SRK", "C"))

} ← Basic example of our database.

public List<IplTeam> getIplTeams() {

return iplTeam;

}

}

Service.java

```
public class Service {
    @Autowired
    Dao dao;

    public void getIplTeams() {
        List<Ipl team> iplTeams = dao.getIplTeams();
        return iplTeams;
    }
}
```

Controller.java

```
@RestController
public class Controller {

    @Autowired
    Service service;

    List<Ipl team> public void iplTeams();

    List<Ipl team> iplTeams =
        service.getIplTeams();
        return iplTeams;
}
```

Searching /get particular data

CRUD Operations

Dependencies → Spring web
→ MySQL Driver
→ Spring Data JPA
→ Spring Boot Dev Tools.

Syntax →

@ Component
Generic annotation applicable for any class -
→ Base for all Spring Stereo type Annotations:

Specialization :-
* @ Service → Indicates that an annotated class has business logic.

@ Controller

@ Repository → Indicates that an annotated class is used to retrieve and/or manipulate data in a database.

Spring Annotations (most important)

* @Configuration

Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions.

↳ This class has methods that will return beans.

@ Configuration
public class AppConfig {

@ Bean → method produces a bean to be managed..
public MyService myservice() {
return new MyServiceImpl();
}

@ Bean
public UserRepository userrepository() {
return new UserImpl();
}

* @ Component
↳ Class.

Best thing

@ Spring Boot Application

↳ * SpringBoot Configuration
↳ * Enable Auto Configuration
↳ * Component Scan.

① JDBC (Java Database Connectivity):-
we write all the SQL but gets messy

② Spring JDBC
still SQL but less code.

③ JPA (Java Persistence API)
No SQL writing required mostly.
Entity class defined.
object oriented approach.

④ Spring Data JPA

Takes JPA to next level
↳ You don't even need to implement the DAO layer.

public interface TodoRepository extends JpaRepository<Todo, Long> { }

todoRepository.findAll();
todoRepository.deleteById(id);

JPA to Spring Data JPA.

@Repository
public class PersonJpaRepository {

@PersistenceContext
EntityManager entityManager;

public Person findById(int id) {
return entityManager.find(Person.class, id);
}
}

↓

public interface PersonRepo extends
JpaRepository<Todo, Integer> {}

Hibernate vs JPA
↳ defines the specification (API)
→ define entities
→ map attributes
→ entities. manage.

Request Methods for REST API

GET	Retrieve details of a resource
POST	Create a new resource.
PUT	Update an existing resource.
PATCH	Update part of a resource
DECETE	Delete.

APIS backend logic ⟶

```
const [userCreating, setUserCreating] =
useState<boolean>(false);
```

* userCreating ← checks whether a user is being created.
  ↓
  false.

```
<AlertDialog open={userCreating}
  false → Dialog is closed
  true → Dialog is open
```

```
open = {user creating}
onOpenChange={(open) => {
        if (!open) setUserCreating(true);
}}
```

Spring Boot Backend Connection to NextJs

① User submits a form on frontend:

```
fetch('/api/users', {
    method: 'POST',
    body: JSON.stringify(userForm),
    headers: {'Content-Type':
              'application/json'},
})
```

① userController (API Layer)

* exposes REST endpoints (/api/users)
* Handles HTTP requests.
* Passes data to service layer.

@PostMapping
public ResponseEntity<User> createUser
(@ResponseBody User user) {
    User created = userService.create(user);
    return ResponseEntity.ok(created);
}

Frontend → POST /api/users → UserController
→ UserService → UserRepository → DataBase.

Common Spring Concepts

@ Rest Controller	Rest API
@ Service	Business logic Component
@ Repository	Handles Database operations
@ Entity	JPA - mapped class for Database table
@ Post Mapping	Handles HTTP POST requests
@ Request Body	Maps JSON payload to Java object.
@ Jpa Repository	Interface with CRUD functions

CRUD Basic understanding :-

Todo REST API :-

* Retrieve Todos :-

@GetMapping("/users/{username}/todos")

CRUD Basic understanding :-

Todo REST API :-

* Retrieve Todos :-

@GetMapping("/users/{username}/todos")

Todo public getTodos(username){

Todo a = TodoService.getTodo(username);

return a;

}

↓

Spring Security →

In a system

we have

* Resources → API, Web App, Database
* Identities → identities need to access & perform actions on top of resources.

Key questions :- How to identify users?

↓

Authentication → User id & password.

→ Biometrics

Authorization

User A can only read data.

User B can read & update.

Understanding Important Security Principles :- IN COMPUTER NETWORKS

① Trust Nothing

↳ Validate every request.

↳ Validate all incoming data.

Input validation on Client side

② Assign Least Priviledges

↳ Start design of system with security requirements.

↳ Clear user roles & acces.

↳ Minimum possible priviledges at all levels.

No full permissions!

③ Have a complete mediation

↳ everyone pass through one gate.

Check every single time.

④ Have Defense In Depth

↳ Multiple levels of security

Frontend + backend checks.

⑤ Economy of Mechanism

↳ Simple security architecture

token based DB verified

⑥ Openness of Design

Use peer reviewed security algos.

Spring Provides Security :-

Filter Chain

Authentication Manager

Authentication Providers.

Request → Spring Security → Dispatcher servlet → Controllers.

Spring Security executes a series of filters

filters are providing :-

① Authentication : Is it a valid user?

② Authorization: Does the user have right access?

③ Cross Origin Resource Sharing (CORS Filter)

④ Cross Site Request Forgery (CSRF)

Cross - Site Request Forgery

* You logged on ipv4's portal and you at the same time visited some malicious website -
* That malicious website executes a bank transfer without your knowledge using Cookie - A.

```
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
        public void addCorsMapping("/**")
            .allowedOrigins(...)
    }
}
```

Encoding vs Hashing vs Encryption

Encoding :- Transform data — one form to another

* no Keys
* reversible
* Not used for security.
* Compression, Streaming.

Hashing :- Convert data into a Hash

* One way process
* Not reversible
* validate integrity of data
* bcrypt, scrypt.

Encryption : Encoding data using a key.
RSA.

PasswordEncoder: interface for performing one way transformation of a password.

JWT :-

* Basic Authentication
  - ↳ No expiration Time
  - ↳ No User Details.
  - ↳ Easily Decoded.

pg 185.

File Upload Component.

Multipart file is a type of file transfer method commonly used in web applications for the file uploads.

* built in support.
* POST Api
* Files can be processed on server
* Annotation-based configuration

Application.properties →

enabling multipart uploads

* spring.servlet.multipart.enabled= true .

max file size

* spring.servlet.multipart.max-file-size = 300MB

@RestController

@RequestParam("file")
MultipartFile file

@PostMapping("/upload-file")
public ResponseEntity<String> uploadfile(

return ResponseEntity.ok("Working");

3

Production :- Virtual Machine

AWS
EC2

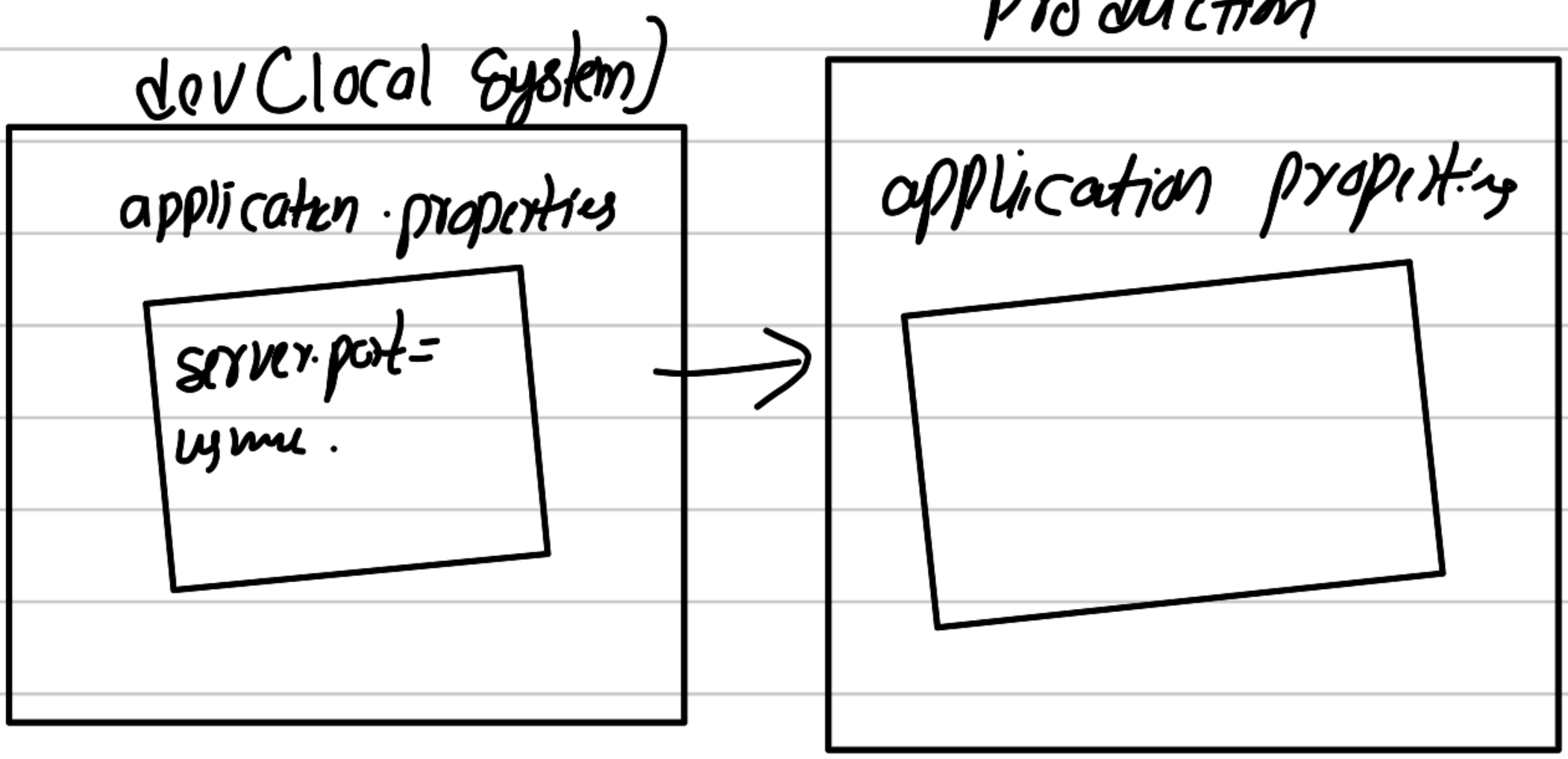
↙ simple

ubuntu
>> terminal.

↳ need to manually configure.

AWS Elastic Beanstalk automatically handles the deployment.

web server
mysql sql
configure automatically



We can store it locally for now under some local directory & later on we can

We should def make a new database table called attachments - we can store filename, path and other fields. This will enable fast access & so it's a standard practise.

Implementation of Mapstruct

* DTOs → Data Transfer OBJECTS.

* Entities → Backend

Never use entities to transfer data between different layers such as Controllers, services & repositories.

Use entity to save data in the database.

We need a mapping for transfering entity to Dtos.

MAPSTRUCT → reduces manual work. This a automatic mapper.

What does spring.jpa.hibernate.ddl-auto = update do ?

* create : Hibernate first drops existing tables and then create new tables.

* update: The object model created based on mappings (annotations/XML) is compared with existing schema & hibernate updates the schema according to the diff. It never deletes the existing tables or columns even if they are no longer the application.

* create-drop :- similar to create, with the addition that hibernate will drop the database after all operations are completed : typicaly for test.

* Validate: Only checks whether the tables or columns exist - It throws an exception otherwise.

Flow goes

→ Controller auto wire service
→ Servie auto wire repositoy
Repositoy interacts with your database.

Service Layer

```
public Employee saveEmployee(EmployeeDTO employeeDTO) {
    //code to convert dto to employee
    to entity

    EmployeeRepo.save( );
}
```

Controller

```
@PostMapping("/add")
public ResponseEntity<Employee> addEmployee
    (@RequestBody EmployeeDTO employeeDTO) {

return new ResponseEntity<>(employeeServiceImpl.saveEmployee(
        EmployeeDTO), HTTP.(REMED))
```

Front end TO
CONTROLLER TO
SERVICE TO
REPOSITORY TO
DATABASE

enums in Java:-

```
public enum Currency ( ) {
    HKD;
    //methods
    public void getString() {
        return ("HKD");
    }
}
```

S

The practise of using constructor injection instead of field injection in Spring.

→ Improves testability
→ Improves Immutability
→ Improves Readability
→ No Reflection ; resolved instantiation _

```
private final SomeDependency someDependency;

public AdminServiceImpl (SomeDependency
                         someDependency ) {
    this.someDependency = someDependency;
}
```

Dynamic Queries with Specifications

Specifications

hasId
containsName
containsDescription
hasOwnerName

→ Query

hasId and
contains Name and
contains Description

make a specification class
↳ make static functions that
return specification
lambda function.

example

```
public class UserSpec {

    public static Specification<User> hasId
            (String providedId) {

        return (root, query, criteriaBuilder) ->

            criteriaBuilder.equal(root.get
                ("id"), providedId);
    }
```

HTTP Request Body

{

"name" : "Sword"

"description" : "Hogwarts"

3

KEYS VALUES

"name" "Sword"

"description" "Hogwarts"

In Services →

public Page <User> findByCriteria(Map<String, String>
searchCriteria,
Pageable pageable) {

Specification <Artifact> spec =
Specification.where((spec: null));

if(StringUtils.hasLength(searchCriteria.
get("id"))) {
spec = spec.and(UserSpec.hasId(
searchCriteria.get("id")))

x — x END OF BASIC.