**What is Spring boot?**

Standalone application based off of the Spring framework, that reduces the setup for applications and comes with features such as: auto-configuration, embedded servers, production features (metrics and health checks). The @Springbootapplication simplifies the configuration aspect of the application.

**Auto-configurations** – configurations created based off of the jar dependencies.

Starter Dependencies – dependency descriptors for dependency management: ie. Spring-boot-starter-web and spring-boot-starter-data-jpa

**Web Servers** – are initialized when spring boot is set up.

Application.properties & application.yml – files that can be used to override properties.

**Profiles** – segregating different parts of the application to allow it to be environment specific.

**Added Dependencies:**

Spring Web

H2 Database – DB used that will be moved to MySQL

Springboot Actuator

Spring Data JPA

Spring Boot Application

Lombok

Validation

JDK – Holds the tools, libraries, and JRE needed for running Java applications.

JVM – Runs the Byte Code

JRE – Holds the JVM Runs the Byte Code

Threads – the smallest unit of tasks. Multithreading happens when the tasks are all running at the same time.

Example – Several pieces of software all accessing the OS and HW for requestsl they need to run concurrently.

MAVEN PROJECT

POSTGRESQL

* Downloaded Postgresql
* Created Database
* Added table, properties, queries, and outputs
* ML2 Contract – Marine Contract

ETL Palantir Java -

Palantir

Some of our plugin applications were developed with Spring boot

**Spring Data JPA**

JPA – Java Persistence API – manages relational data and DBs using java, it uses ORM (Object Relational Mapping) to manage relational data without writing a lot of SQL code. It maps Java object/entities to DB tables and handles conversions.

Spring JPA has repo interfaces to interact with databases (JPA Repositor and CrudRepository)

Public interface UserRepo extends JPARepository<User, Long> {

}

**Annotations** – Are used to define how Java classes map to DB tables. Some important annotations are @Entity, @table, @id, @GenerateValue, @Column.

Entity manager is used as an integrate for managing DB operations: Create, Read, Update, Delete. CRUD capabilities.

JPQL – JPA uses JPQL as the query language that works with objects instead of tables.

Transaction Management – JPA works with the framework to manage transaction.

Caching – JPA has built in caching to improve performance by lowering the number of DB hits.

An example of JPA would be:

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

// Getters and Setters

}

@Entity – Would define the user class – (DCSA Employee) – name, region, Facility, classification level

@Id & @GenerateValue – defines the primary

**RECAP HOW THE DESIGN PATTERNS WORK TOGETHER!!**

**DESIGN PATTERNS**

**Repository Layer** – Performs the data access logic (CRUD) and abstract persistence

**Service Layer** – Contains business logic and orchestrates of repo

**Singleton Layer** – Spring beans are singletons, so theres one instance of the bean for the context. One instance of an object per application.

**Model (User.java)**

* Represents a user in the database.
* Defines the property and primary key of the object. Has the @Entity annotation
* Other properties include id, name, and email.

**Repository (UserRepository.java)**

* UserRepo interface extances the JPA and provides abstraction on built in crud methods. The is the data access layer that spring automatically implements

**Service Layer (UserService.java)**

Where service layer patterns are applied, the UserService class hands the business logic such as creating updating and getting users. It also separates the business logic from the data access layer . it interacts with the repository layer. Used **Optionals** for handling cases where values are null (getUserById).

**Controller (UserController.java)**

* Provides **REST endpoints** that interacts with the service layer. Request Handling.
* @RestController: All methods returns a domain object instead of a view.
* @RequestMapping: Handles HTTP request to /api/users enpoints
* @GetMapping: getUsers, getIDs
* @PostMapping: createUser
* @DeleteMapping: deleteUsers

TDD – Writing a test to fail prior to development, write to past, improve code without changing behavior

BDD – Behavioral Driven Design -

DDD – Domain Driven Design –

TODO:

SPRING / JPA Example

Postgres

Design Patterns

MVC Recap

PVM Use Case (2) – DCSA Employee, obtaining data

NSWDD Use Case (2) – MVC

Creating a Spring Boot application

POM.XML – Project Object Model - Configuration file that has dependencies, metadata, and settings.

Spring boot starter web – Builds REST APIs

Spring Data JPA – All JPA related libraries

H2 Database – in-memory database

1. Create Java project with Maven
2. Update POM.xml and import dependencies
3. Create main class and Invalidate caches while removing cache and logs
4. Create packages and classes

Model - User entity (user.java) Service – service layer / business logic (US.java)

Repository – repo interface (UR.java) Controller – REST API controller (UC.java)

1. Configure Database in application.properties
2. DB is configured in localhost:8080/h2-console
3. Build > Rebuild Project

Spring Bean – an object instantiated by the Spring IoC, acts the backbone of the Spring application forming components that are wired to the application. It is denoted by the annotation @Bean, configurations, and XML. They provide features like instantiation, dependency injection, and destruction.