JAVA Y SPRING (MYSQL)

**Introducción a SQL con MySQL: Manipule y consulta datos**

Lo que aprendimos en esta aula:

* Conocimos un poco sobre la historia de SQL como lenguaje de base de datos relacional.
* Vimos un poco sobre la historia y características de la base de datos MySQL.
* Aprendimos a instalar MySQL y Workbench.

Cap2

Lo que aprendimos en esta aula:

* A acceder y navegar por Workbench.
* A crear una base de datos por líneas de comando en SQL o por el asistente.
* A borrar una base de datos por líneas de comando en SQL o por el asistente.
* A acceder a una tabla por líneas de comando o por el asistente.

Cap3

Lo que aprendimos en esta aula:

* Los tipos de datos que componen una tabla.
* A crear una tabla, tanto por líneas de comando como por el asistente.
* A eliminar una tabla.

Cap4

Lo que aprendimos en esta aula:

* A incluir datos en una tabla, de diversas formas.
* Vimos cómo cambiar un dato ya existente en la tabla.
* Vimos cómo eliminar una fila de la tabla.
* Conocimos la importancia de las claves primarias y el cuidado que debemos tener al crearlas.
* Aprendimos a manipular campos del tipo lógicos y del tipo fecha.

Cap5

Lo que aprendimos en esta aula:

* A visualizar los datos de una tabla.
* A separar la selección de datos.
* A utilizar las condiciones mayor que y menor que en la selección de datos de la tabla.
* A filtrar los datos a través de fechas.
* A implementar filtros compuestos.

**Java y JDBC: Trabajando con una base de datos**

Cap1

Lo que aprendimos en esta aula:

* Para acceder a una base de datos necesitamos del **driver de conexión**;
  + Un driver es simplemente una librería **.jar**.
* **JDBC** significa *Java DataBase Connectivity*;
  + El JDBC define una capa de abstracción entre la aplicación y el driver de la base de datos.
  + Esta capa es compuesta de interfaces que el driver implementa.
* Para abrir una conexión con la base de datos debemos utilizar el método getConnection de la clase DriverManager;
  + El método getConnection recibe tres parámetros. Son ellos la URL de conexión JDBC, el usuario y la contraseña.

Cap2

Lo que aprendimos en esta aula:

* Para simplificar y encapsular la creación de la conexión debemos utilizar una clase ConnectionFactory;
  + Esta clase sigue el estándar de creación *Factory Method*, que encapsula la creación de un objeto.
* Podemos utilizar la interfaz java.sql.Statement para ejecutar un comando SQL en la aplicación;
  + El método execute envía el comando para la base de datos.
  + A depender del comando SQL, podemos recuperar la clave primaria o los registros buscados.

Cap3

Lo que aprendimos en esta aula:

* Cuando ejecutamos una query SQL como Statement tenemos un riesgo de seguridad llamado **SQL Injection**;
  + **SQL Injection** es el hecho de enviar comandos SQL como parámetro de las solicitudes en una aplicación.
* Para evitar el fallo por **SQL Injection** debemos utilizar la interfaz PreparedStatement;
  + Diferente del Statement, el PreparedStatement trata los parámetros del comando SQL para que caracteres y comandos especiales sean tratados como strings.
* Las bases de datos ofrecen un recurso llamado **transacción**, que junta muchas operaciones SQL como un conjunto de ejecución;
  + Si el conjunto falla no es aplicada ninguna modificación y ocurre el *rollback* de la transacción.
  + Todos los comandos del conjunto necesitan funcionar para que la transacción sea finalizada con un *commit*.
* Para garantizar el cierre de los recursos abiertos en el código, Java provee un recurso llamado *try-with-resources* para ayudarnos;
  + Para utilizar este recurso es necesario que la clase utilizada (como la Connection) implemente la interfaz Autocloseable.

Cap4

Lo que aprendimos en esta aula:

* Utilizar el ***pool de conexiones*** es una buena práctica;
* El pool de conexiones controla la cantidad de conexiones abiertas entre la aplicación y la base de datos;
  + Es normal que haya un mínimo y un máximo de conexiones.
* De la misma forma que hay, en JDBC, una interfaz para representar la conexión (java.sql.Connection), también hay una interfaz que representa el pool de conexiones (javax.sql.DataSource);
* **C3P0** es una implementación Java de un pool de conexiones.

Cap5

Lo que aprendimos en esta aula:

* Para cada tabla del modelo tenemos una clase de dominio;
  + Para la tabla de producto tenemos una clase Producto asociada.
  + Los objetos del tipo Producto representan un registro de la tabla.
* Para acceder a la tabla vamos a utilizar el estándar llamado ***Data Access Object*** (**DAO**);
  + Para cada clase de dominio hay un DAO asociado. Por ejemplo, la clase Producto posee la clase ProductoDAO.
  + Todos los métodos JDBC relacionados al producto deben estar encapsulados en ProductoDAO.
* Una aplicación es escrita en capas;
  + Las capas más conocidas son las de *view*, *controller*, *modelo* y *persistencia*, que componen el estándar MVC.
* El flujo de una requisición entre las capas es el siguiente;

view <--> controller <--> persistenciaCOPIA EL CÓDIGO

* En este curso utilizamos una aplicación con las *views* y los *controllers* ya creados y enfocamos en la capa de persistencia y modelo;
* No es una buena práctica dejar los detalles de implementación de una capa en otras que no tienen esta responsabilidad (por ejemplo la capa de *controller* lanzar una SQLException);
* Aquí estamos aprendiendo con una aplicación desktop embebida, pero hay otros tipos de aplicaciones con otros tipos de view, como **html** para aplicaciones web.

Cap 6

Lo que aprendimos en esta aula:

* Cuando tenemos una relación entre *tables* debemos tener cuidado para no crear el problema de queries N + 1;
  + N + 1 quiere decir que, para buscar los datos de una relación, es ejecutada una query y luego una más por cada relación.
  + Este tipo de problema puede generar problemas de performance en la aplicación y en la base de datos.
  + Este tipo de problema puede ser evitado utilizando *join* en la query SQL.

**Persistencia con JPA : Hibernate**

Cap 1

Para preparar el ambiente, necesitamos tener algunas cosas instaladas en la computadora, como:

**El eclipse**

* Para instalar Eclipse, clique en el link y descargue la versión actual
  + [The Eclipse Foundation](https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2022-12/R/eclipse-inst-jre-win64.exe&mirror_id=576)

**Img-eclipse**

* Luego de haberlo descargado vamos a seleccionar la versión Enterprise que permite trabajar con java EE o Jakarta (Una transición de Java EE a Jakarta EE simboliza un cambio de código comercial a código abierto no Maven Repository: com.h2database » h2 » 2.0.206 (mvnrepository.com)solo en la práctica, sino también en el nombre.)

**img-option**

* Alternativamente:
  + 1) Podemos trabajar con editores de código como intellij , visual studio code, pero estos tienen configuraciones diferentes a las que serán utilizadas en el video, sin embargo, es posible trabajar con ellos dejando como desafío usar otro editor de código.
  + [IntelliJ IDEA](https://www.jetbrains.com/idea/download/#section=windows)
  + [Visual Studio Code](https://code.visualstudio.com/Download)
* 2) En el curso vamos a utilizar la base de datos H2 dejamos como desafío el uso de otras bases de datos.
  + [Maven Repository: com.h2database](https://mvnrepository.com/artifact/com.h2database/h2/2.0.206)

Lo que aprendimos en esta aula:

En esta lección, aprendiste:

* Las desventajas de usar JDBC para acceder a bases de datos en Java;
* La historia de la creación de Hibernate y JPA;
* Cómo crear una aplicación Maven y agregar Hibernate como dependencia.

Cap2

Lo que aprendimos en esta aula:

En esta lección, aprendiste:

* Cómo configurar JPA a través del archivo de persistencia.xml;
* Cómo mapear entidades JPA;
* Cómo utilizar EntityManager para conservar entidades en la base de datos.

Lo que aprendimos en esta aula:

Cap3

En esta lección, aprendiste:

* Cómo escribir una clase DAO usando JPA;
* Cómo asignar atributos de tipo Enum en una entidad;
* Cómo mapear una relación entre entidades.

Cap4

Lo que aprendimos en esta aula:

En esta lección, aprendiste:

* Cómo funciona el ciclo de vida de las entidades JPA;
* Las transiciones de estado de una entidad cuando persisten;
* Las transiciones de estado de una entidad cuando se actualizan;
* Las transiciones de estado de una entidad cuando se quita.

Cap5

Lo que aprendimos en esta aula:

En esta lección, aprendiste:

* Cómo consultar con el método find de Entity Manager;
* Cómo realizar consultas JPQL;
* Cómo filtrar datos en consultas JPQL;
* Cómo devolver sólo una parte de una entidad en una consulta JPQL.

SPRING BOOT 3 DESARROLLAR API REST CON JAVA

CAP 1

¡Ahora está contigo! Haz el mismo procedimiento que hice en clase. Cree el proyecto inicial a través del sitio web [Spring Initializr](https://start.spring.io/), importarlo en IntelliJ y finalmente cree una clase Controller como Hello World.

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

La clase Controller de ejemplo debe tener la siguiente estructura:

@RestController

@RequestMapping("hello")

public class HelloController {

@GetMapping

public String holaMundo() {

return "Hello World Spring!";

}

}

**En esta clase, aprendiste a:**

* Crear un proyecto Spring Boot usando el sitio web Spring Initializr;
* Importar el proyecto a IntelliJ y ejecutar una aplicación Spring Boot a través de la clase que contiene el método main;
* Crear una clase Controller y mapear una URL en él usando las anotaciones @RestController y @RequestMapping;
* Realizar una solicitud de prueba en el navegador accediendo a la URL mapeada en el Controller.

CAP2

## En esta clase, aprendiste a:

* Mapear solicitudes POST en una clase Controller;
* Enviar solicitudes POST a la API usando Insomnia;
* Enviar datos a la API en formato JSON;
* Utilizar la anotación @RequestBody para recibir datos del cuerpo de la solicitud en un parámetro en el Controller;
* Use el padrón **DTO (Data Transfer Object)**, a través de Java Records, para representar los datos recibidos en una solicitud POST.

[**DISCUT**](https://app.aluracursos.com/forum/curso-spring-boot-3-desarrollar-api-rest-java/exercicio-lo-que-aprendimos/83458/novo)

CAP3

**En esta clase, aprendiste a:**

* Agregar nuevas dependencias en el proyecto;
* Asignar una entidad JPA y crear una interfaz de Repositorio para ella;
* Utilizar Flyway como herramienta de migración de proyectos;
* Realice validaciones con Bean Validation usando algunas de sus anotaciones, como @NotBlank.

CAP4

**En esta clase, aprendiste a:**

* Usar la anotación @GetMapping para mapear métodos en los Controllers que producen datos;
* Usar la interfaz Pageable de Spring para realizar consultas con paginación;
* Controlar la paginación y el ordenamiento de los datos devueltos por la API con los parámetros page, size y sort;
* Configurar el proyecto para que los comandos SQL se visualicen en la consola.

CAP5

**En esta clase, aprendiste a:**

* Mapear solicitudes PUT con la anotación @PutMapping;
* Escribir un código para actualizar la información de un registro en la base de datos;
* Mapear solicitudes DELETE con la anotación @DeleteMapping;
* Mapear parámetros dinámicos en la URL con la anotación @PathVariable;
* Implementar el concepto de exclusión lógica utilizando un atributo booleano.

**JAVA SPRING BOOT 3: APLIQUE LAS MEJORES PRACTICAS Y PROTEJA UNA API REST**

¡Ahora está contigo! Realice el mismo procedimiento que hice en clase, implementando ResponseEntity, sin embargo, para las funcionalidades del CRUD de los pacientes.

En esta clase, aprendiste a:

* Usar la clase ResponseEntity, de Spring, para personalizar los retornos de los métodos de una clase Controller;
* Modificar el código HTTP devuelto en las respuestas de la API;
* Agregar encabezados a las respuestas de la API;
* Utilice los códigos HTTP más apropiados para cada operación realizada en la API.

CAP2

En esta clase, aprendiste a:

* Crear una clase para aislar el manejo de excepciones de API, utilizando la anotación @RestControllerAdvice;
* Utilizar la anotación @ExceptionHandler, de Spring, para indicar qué excepción debe capturar un determinado método de la clase de manejo de errores;
* Manejar errores 404 (Not Found) en la clase de manejo de errores;
* Manejar errores 400 (Bad Request), para errores de validación de Bean Validation, en la clase de manejo de errores;
* Simplificar el JSON devuelto por la API en casos de error de validación de Bean Validation.

[**DISCUTIR EN EL**](https://app.aluracursos.com/forum/curso-spring-boot-3-aplique-practicas-proteja-api-rest/exercicio-lo-que-aprendimos/83823/novo)

CAP3

En esta clase, aprendiste a:

* Identificar cómo funciona el proceso de autenticación y autorización en una API Rest;
* Agregar Spring Security al proyecto;
* Cómo funciona el comportamiento padrón de Spring Security en una aplicación;
* Implementar el proceso de autenticación en la API, de forma Stateless, utilizando clases y configuraciones de Spring Security.

CAP4

¿Qué hemos aprendido?

* En esta clase, aprendiste a:
* Agregar la biblioteca Auth0 java-jwt como una dependencia del proyecto;
* Utilizar esta biblioteca para generar un token en la API;
* Inyectar una propiedad del archivo application.properties en una clase administrada por Spring, usando la anotación @Value;
* Devolver un token generado en la API cuando un usuario se autentica.

CAP5

En esta clase, aprendiste a:

* Los Filters funcionan en una solicitud;
* Implementar un Filter creando una clase que herede de la clase OncePerRequestFilter de Spring;
* Utilizar la biblioteca Auth0 java-jwt para validar los tokens recibidos en la API;
* Realizar el proceso de autenticación de la solicitud, utilizando la clase SecurityContextHolder de Spring;
* Liberar y restringir solicitudes, según la URL y el verbo del protocolo HTTP.

**7 Spring booot 3 Documentar, probar y preparar API para su implementación**

**Cap1**

Lo que aprendimos en esta aula:

* Implementar una nueva funcionalidad en el proyecto;
* Evaluar cuando es necesario crear una clase de Servicio en la aplicación;
* Crear una clase de Servicio, con el objetivo de aislar códigos de reglas de negocio, utilizando para ello la anotación @Service;
* Implementar un algoritmo para la funcionalidad de programación de consultas;
* Realizar validaciones de integridad de la información que llega a la API;
* Implemente una consulta JPQL (Lenguaje de consulta de persistencia de Java) compleja en una interfaz de repositorio, utilizando la anotación @Query.

**Cap2**

SOLID es un acrónimo que representa cinco principios de programación:

* Principio de Responsabilidad Única (Single Responsibility Principle)
* Principio Abierto-Cerrado (Open-Closed Principle)
* Principio de Sustitución de Liskov (Liskov Substitution Principle)
* Principio de Segregación de Interfaces (Interface Segregation Principle)
* Principio de Inversión de Dependencia (Dependency Inversion Principle)

Cada principio representa una buena práctica de programación que, cuando se aplica en una aplicación, facilita mucho su mantenimiento y extensión. Estos principios fueron creados por Robert Martin, conocido como Uncle Bob, en su artículo Design Principles and Design Patterns.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-spring-boot-3-api-para-su-implementacion/exercicio-para-saber-mas-principios-solid/84623/novo)

Lo que aprendimos en esta aula:

* Aislar los códigos de validación de reglas comerciales en clases separadas, utilizando la anotación @Component de Spring;
* Finalizar la implementación del algoritmo de programación de consultas;
* Usar los principios SOLID para hacer que el código de la funcionalidad de programación de citas sea más fácil de entender, evolucionar y probar.

**Cap3**

Lo que aprendimos en esta aula:

* Agregar la biblioteca SpringDoc al proyecto para que genere automáticamente la documentación de la API;
* Analizar la documentación de SpringDoc para comprender cómo configurarlo en un proyecto;
* Acceder a las direcciones que brindan la documentación de la API en formatos yaml y html;
* Utilizar la interfaz de usuario de Swagger para visualizar y probar una API Rest;
* Configurar JWT en la documentación generada por SpringDoc.

**Cap4**

Lo que aprendimos en esta aula:

* Escribir pruebas automatizadas en una aplicación con Spring Boot;
* Escribir pruebas automatizadas para una interfaz de Repositorio, siguiendo la estrategia de utilizar la misma base de datos que utiliza la aplicación;
* Sobreescribir las propiedades del archivo application.properties, creando otro archivo llamado application-test.properties que se carga solo cuando se ejecutan pruebas, usando la anotación @ActiveProfiles para eso;
* Escribir pruebas automatizadas para una clase de controlador, utilizando la clase MockMvc para simular solicitudes de API;
* Probar los escenarios de error 400 y código 200 al probar una clase de controlador.

**Cap5**

Lo que aprendimos en esta aula:

* Realizar el build de una aplicación con Spring Boot;
* Utilizar archivos de propiedades específicos para cada perfil, cambiando las propiedades que deben modificarse en cada archivo;
* Configurar información confidencial de la aplicación, como datos de acceso a la base de datos, a través de variables de entorno;
* Crear el proyecto a través de Maven;
* Ejecutar la aplicación a través de la terminal, con el comando java -jar, pasando las variables de entorno como parámetros.