Guía creación del backend y frontend: paso a paso

Stefanny Nikoll Hidalgo Urrea

Análisis y desarrollo de software

Ficha. 2694679

Servicio Nacional de Aprendizaje

Neiva, Huila

2024

Tabla de contenido.

1. Resumen.
2. Service Security.
   1. Planteamiento del problema.
   2. XAMPP.
   3. Spring Initializr.
      1. Empaquetamiento.
      2. Dependencias.
   4. Base de datos.
   5. Backend.
      1. Configuración de propiedades.
      2. Archivo “pom.xml”.
      3. Crear paquetes y sus archivos.
         1. Paquete “Entity”.
         2. Paquete “IRepository”.
         3. Paquete “IService”.
         4. Paquete “Service”.
         5. Paquete “DTO”.
         6. Paquete “Controller”.
   6. Frontend.
      1. Crear carpetas y sus archivos.
         1. Carpeta “HTML”.
         2. Carpeta “JS”.
         3. Carpeta “ASSETS”.
3. Location.
   1. Planteamiento del problema.
   2. XAMPP.
   3. Spring Initializr.
      1. Empaquetamiento.
      2. Dependencias.
   4. Base de datos.
   5. Backend.
      1. Configuración de propiedades
      2. Archivo “pom.xml”.
      3. Crear paquetes y sus archivos.
         1. Paquete “Entity”.
         2. Paqueta “IRepository”.
         3. Paquete “IService”.
         4. Paquete “DTO”.
         5. Paquete “Controller”.
   6. Frontend.
      1. Crear carpetas y sus archivos.
         1. Carpeta “HTML”.
         2. Carpeta “JS”.
         3. Carpeta “ASSETS”.
4. Shopping car.
   1. Planteamiento del problema.
   2. XAMPP.
   3. Spring Initializr.
      1. Empaquetamiento.
      2. Dependencias.
   4. Base de datos.
   5. Backend.
      1. Configuración de propiedades.
      2. Archivo “pom.xml”.
      3. Crear paquetes y sus archivos.
         1. Paquete “Entity”.
         2. Paquete “IRespository”.
         3. Paquete “IService”.
         4. Paquete “Service”.
         5. Paquete “DTO”.
         6. Paquete “Controller”.
   6. Frontend.
      1. Crear carpetas y sus archivos.
         1. Carpeta “HTML”.
         2. Carpeta “JS”.
         3. Carpeta “ASSESTS”.
5. Despedida.

Contenido.

1. Resumen.

En la actividad de hoy se me pidieron 3 actividades, las cuales consistían en crear el backend, usando el framework “Spring Boot”; y el frontend de los siguientes temas: Service security, location y shopping car. Y explicar en un documento Word el procedimiento para aquellos que no estén familiarizados con su construcción o elaboración.

1. Service Security, Location y Shopping Car.
   1. Planteamiento del problema.

Service Security: Este tema se basa en la importancia y necesidad de los sistemas y servicios de seguridad de los administradores y recién agregados usuarios con sus datos personales, por ejemplo: darles a ciertos usuarios un rol y estos roles dependiendo de su capacidad sean limitados a tener acceso a ciertas páginas o vistas para mantener el orden a los usuarios en una página o aplicación.

Entidades y atributos:

Person:

- id

- firstName

- lastName

- email

- phone

- dateOfBirth

- gender

- address

User:

- id

- username

- password

- personId (foreign key referencing Person)

Role:

- id

- name

- description

Module:

- id

- name

- description

- route

View:

- id

- name

- description

- route

Estas últimas tablas son llamadas tablas “pivote” que se crean temporalmente al tener tablas o entidades relacionadas de manera N:N o muchos a muchos, ya que esta relación daría redundancia de datos, se crea una pivote para almacenar la relación de las tablas N:N

UserRole:

- id

- userId (foreign key referencing User)

- roleId (foreign key referencing Role)

ModuleRole:

- id

- moduleId (foreign key referencing Module)

- roleId (foreign key referencing Role)

ModuleView:

- id

- moduleId (foreign key referencing Module)

- viewId (foreign key referencing View)

Location: Este tema se basa en la ubicación precisa de cada lugar, estado, país, etc. A nivel mundial. Ejemplo: El estado de Sao Paolo, está en el país de Brasil y este país es del continente Américano.

Entidades y atributos:

Continente

- id

- code

- name

- description

Pais

- id

- code

- name

- description

- continente\_id (foreign key referencing Continente)

Estado

- id

- code

- name

- description

- pais\_id (foreign key referencing Pais)

Municipio

- id

- code

- name

- description

- estado\_id (foreign key referencing Estado)

Localidad

- id

- code

- name

- description

- municipio\_id (foreign key referencing Municipio)

Barrio

- id

- code

- name

- description

- localidad\_id (foreign key referencing Localidad)

Shopping Car: Este tema abarca las entidades básicas y simples de un minimercado a la hora de comprar y el “recibo” o “factura” cuando se efectúa la compra. Ejemplo: El cliente compra un producto y en la factura muestra la fecha de compra, el total, que producto compró y quién lo compró.

Entidades y atributos:

Cliente

- id

- nombre

- correo

- teléfono

- dirección

Producto

- id

- código

- nombre

- precio

- cantidad

Factura

- id

- código

- fecha

- valor\_total

- cliente\_id (foreign key referencing Cliente)

DetalleFactura

- id

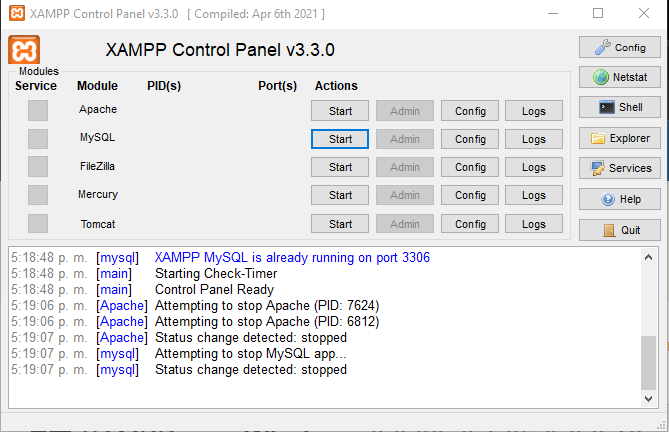
- cantidad

- producto\_id (foreign key referencing Producto)

- factura\_id (foreign key referencing Factura)

* 1. XAMPP.

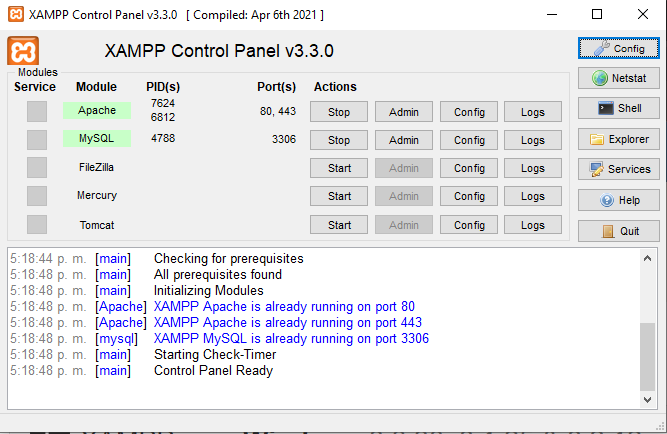
Lo mejor para empezar en este proceso, es activar el XAMPP, una aplicación que inicia los servidores y puertos de MySQL, Apache, etc. Eso evitaría problemas a la hora de crear una base de datos o que el archivo a punto de crear no se conecte a esta.



Si no posee la aplicación XAMPP, ingrese al link: <https://www.apachefriends.org/es/download.html>. Para descargarlo, solo debe descargarlo según su sistema operativo y la versión que desee.

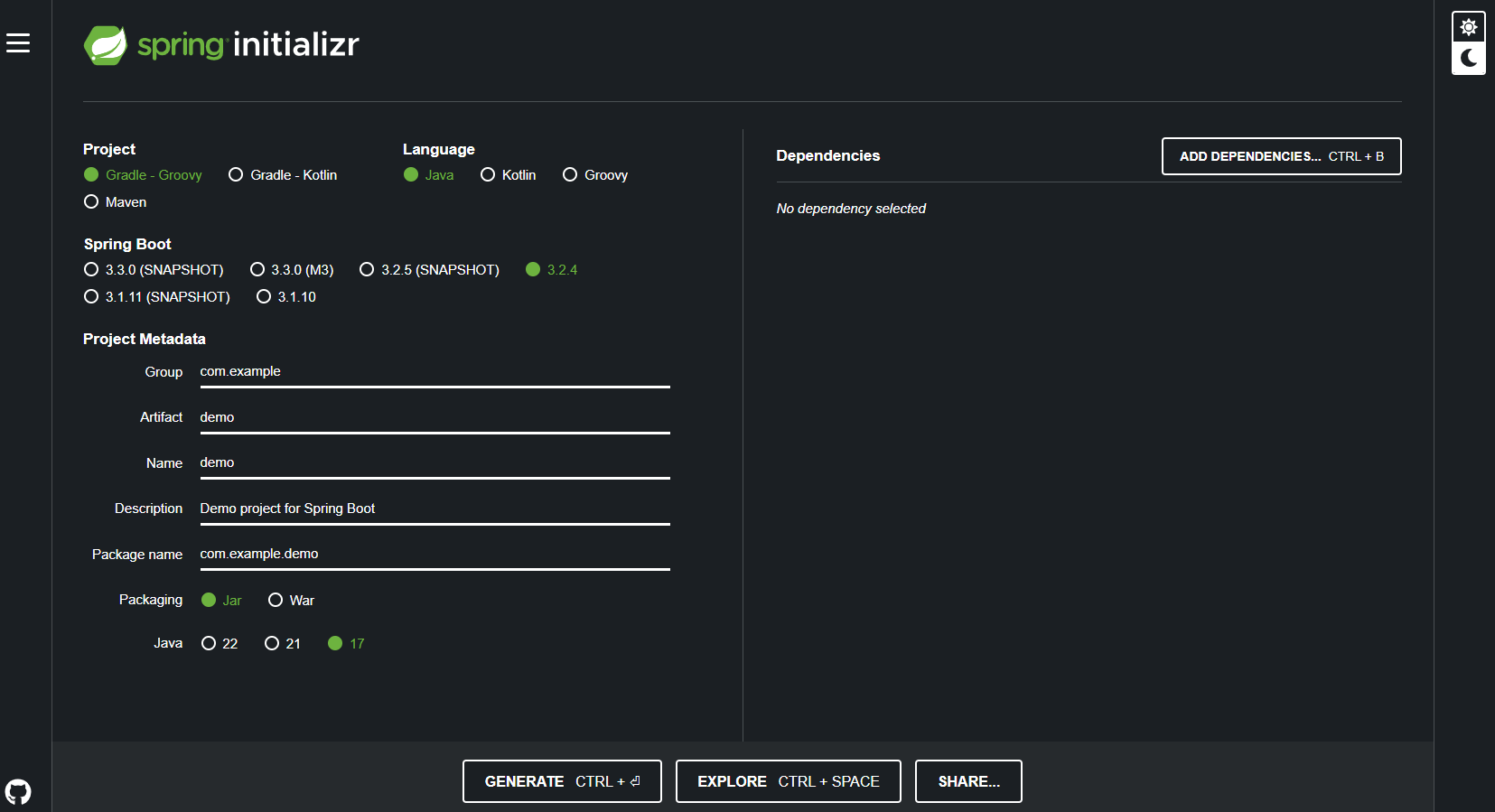
Una vez instalado, deberá abrir la aplicación y activar los 2 primeros servidores.

Ejemplo:



* 1. Spring Initializr.

Existen muchas formas de crear el archivo “spring” pero se explicará el método más sencillo a través de internet, para ello, ingrese al link: [https://start.spring.io](https://start.spring.io/). Y esta será la página principal:



* + 1. Empaquetamiento.

Para crear el proyecto debe seguir la siguiente sentencia paso a paso:

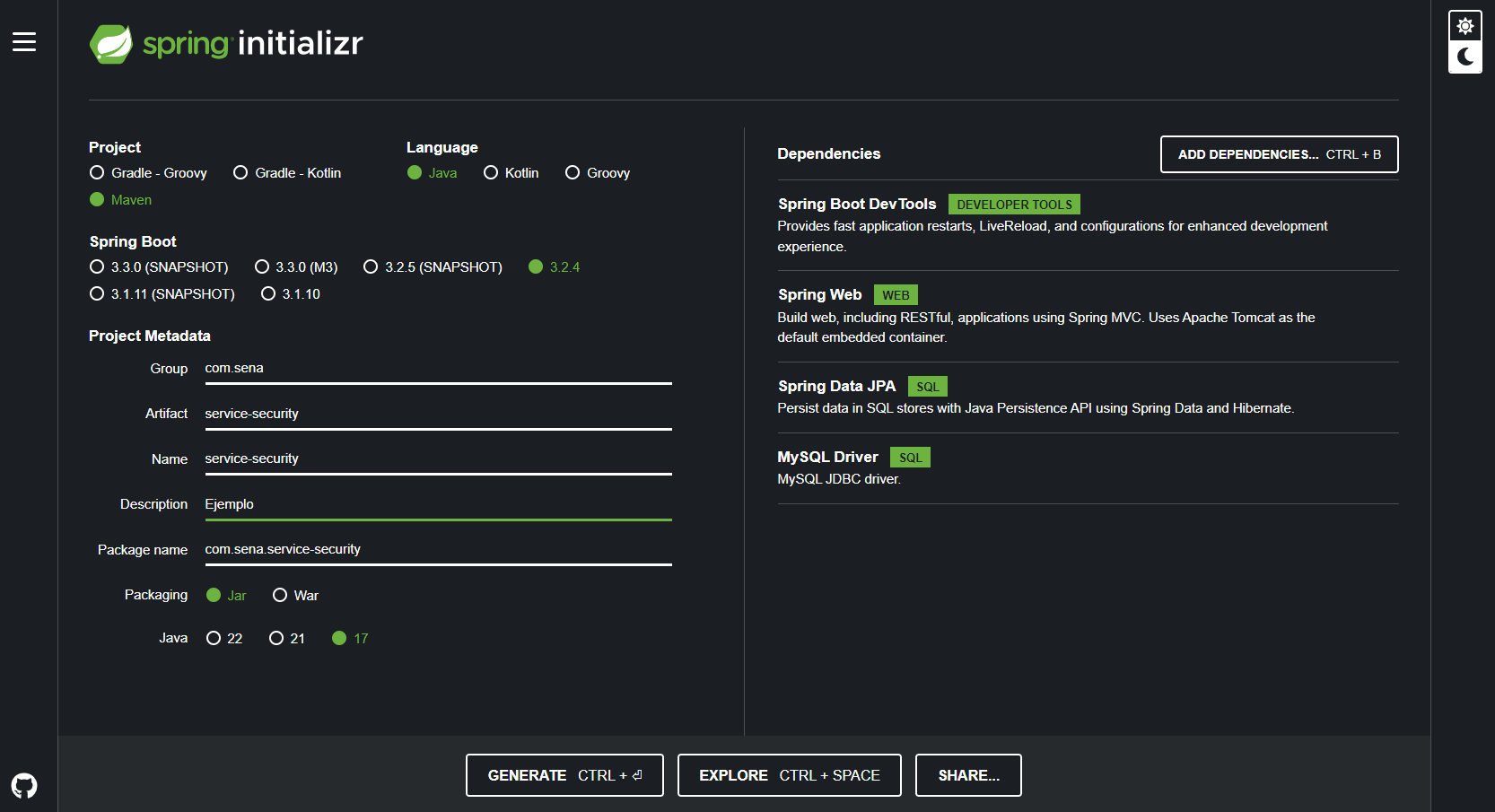
* + - 1. Elegir el tipo de proyecto como “Maven”.
      2. Elegir el tipo de lenguaje como “Java”.
      3. Elegir la versión de Spring Boot más reciente pero que esta no se encentre en desarrollo o prueba, como “3.2.4”.
      4. Para los datos del proyecto:
         1. En Group, en este caso se llenará con “sena” pero se puede elegir un empaquetamiento que se desee. Ejemplo: com.sena.
         2. Artifact y Name comparten el mismo titulo, en este caso se usará “service-security”, “location” o “shopping-car” (preferiblemente usar “-“ para separar los espacios vacíos)
         3. Description, es opcional, pero ayuda a que el equipo de desarrolladores tenga claro de que trata el proyecto.
         4. Package Name quedaría: *com.sena.service-security, com.sena.location,* o *com.sena.shopping-car.*
         5. El tipo de empaquetamiento (Packaging) sea de tipo “Jar”.
         6. Usar la versión de java del pc que se esté utilizando o ciertas funciones dejarán de hacerlo, la versión 17 es la más predeterminada.

***\*\* Observación: para comprobar la versión ingresar al “cmd” del equipo (se puede encontrar en el buscador de aplicaciones) y escribir “java -version” y en caso de que la versión no coincida con las opciones corregirlo en el archivo “pom.xls”, esto se explicará más adelante. \*\****

* + 1. Dependencias.

En las dependencias se requerirán de 4: Spring Boot Dev Tools, Spring Web, Spring Data JPA y MySQL Driver.

Con estas configuraciones quedaría así:



Dando “click” en “Generate” se descargará de manera automática nuestro proyecto.

* 1. Base de datos.

Lo mejor es crear la base de datos para que al ejecutar el proyecto tenga donde conectarse, pero al ejecutar este es capaz de crear las tablas con sus atributos de manera automática así que en la aplicación sql que se use, usando el puerto localhost:3306, escribir la sentencia sql: *create database nameBD.* Y en nuestro caso se llamará *service\_security, location* o *shopping car* (sql tampoco lee los espacios vacíos ni las mayúsculas, así que se usa “\_” para separar los componentes).

* 1. Backend.
     1. Configuración de propiedades.

Para configurarlo, se encuentra con el nombre *aplication.properties* en la carpeta *main/resource* del proyecto .jar. Para configurarlo se usan los componentes.

**spring.application.name=** (nombre de la aplicación)

**server.servlet.context-path=**/name-aplication (usar el swagger-ui, requiere el nombre de la aplicación)

**server.port =** 9000 (número para el puerto a usar)

**spring.jpa.hibernate.ddl-auto =** update (actualiza la base de datos cada vez que se ejecuta, créate-drop, hace que se cree y borre cada vez que se usa)

**spring.datasource.url=** jdbc:mysql://localhost:3306/nameBD (ejemplo: nombre de la base de datos “service\_security”)

**spring.datasource.username =** root

**spring.datasource.password =** (si se tiene una contraseña en el local o la conexión a una BD)

* + 1. Archivo “pom.xml”.

En este archivo se puede encontrar la información que posee el proyecto como la versión del spring, la versión del java que usa el proyecto y sus dependencias, con la opción de actualizar de manera manual, en estas para acceder a la página de swagger-ui, se necesita de:

**<dependency>**

**<groupId>org.springdoc</groupId>**

**<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>**

**<version>2.0.4</version>**

**</dependency>**

* + 1. Crear paquetes y sus archivos.

En este caso cada paquete ya que cuenta con varias entidades, algunos con la capacidad de hacer lo mismo como guardar, actualizar, eliminar y mostrar, se uso un archivo base para cada paquete para evitar la redundancia de datos.

Todos los archivos en los paquetes deben ser de tipo “*.java*”.

* + - 1. Paquete “Entity”.

El archivo base de Entity debe ser de clase: *abstract.*

ABaseEntity:

@MappedSuperclass

public abstract class ABaseEntity {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "estado", nullable = false)

private Boolean estado;

//Encapsulamiento “Getter” y “Setter”.

Estos son un ejemplo de los atributos que todas las entidades poseen.

* + - * + El @MappedSupperclass hace que la clase *ABaseEntity* y sus atributos se mapeen por todas las entidades que hereden esos atributos bases.
        + El @Id junto al @GenerateValue(…) crean la llave primaria de cada entidad en la BD con auto incrementación.
        + El @Column crea las columnas con sus nombre y su tipo null o not null en la BD.

nameEntity: *ejemplo, Peron.java*

@Entity

@Table(name = "nombre\_de\_la\_tabla")

public class nameEntity extends ABaseEntity{

@Column(name = "name\_atributo", length = 50, nullable = true)

private String nameAtributo;

//Encapsulamiento “Getter” y “Setter”.

* El @Entity define que la case es una entidad y el @Table(name = “nombre de la tabla y/o entidad”) crea la tabla en la BD esta debe tener el mismo nombre que la entidad.
* En el @Column se le da un nombre para la BD y la definición del atributo debe tener el mismo nombre con la diferencia de que no debe incluir “- “ni “\_” y en caso de espacios vacíos dejarlo como uno solo separando por una MAYÚSCULA.
  + - 1. Paquete “IRepository”.

Los archivos de IRepository deben ser de tipo clase: *interface.*

IBaseRepository:

@Repository

public interface IBaseRepository<T extends ABaseEntity, ID> extends JpaRepository<T, Long> {

}

INameRepository: *ejemplo, ICarsRepository*

@Repository

public interface INameRepository extends IBaseRepository<nameEntity, Long> {

}

* + - 1. Paquete “IService”.

Los archivos de IService deben ser de tipo clase: *interface.*

IBaseService:

public interface IBaseService<T extends ABaseEntity> {

List<T> all() throws Exception;

List<T> findByStateTrue() throws Exception;

Optional<T> findById(Long id) throws Exception;

T save(T entity) throws Exception;

void update(Long id, T entity) throws Exception;

void delete(Long id) throws Exception;

}

En el IService básicamente le digo lo que va a hacer pero no como hacerlo, estas acciones a usar son sacadas del IRepository.

INameService: *ejemplo, IContinenteService.*

public interface INameService extends IBaseService<nameEntity> {

}

* + - 1. Paquete “Service”.

ABaseService:



El @MappedSuperclass hace que se mapee por las entidades que hereden la base.

El @Override sobrescribe las funciones de save, update, delete, findById y all, y les dice que hacer además de agregar datos como la fecha en que se creen, actualicen y eliminen de manera lógica.

nameEntityService: *ejemplo, FacturaService*

@Service

public class nameEntityService extends ABaseService<nameEntity> implements INameEntityService{

@Override

protected IBaseRepository<nameEntity, Long> getRepository() {

return repository;

}

@Autowired

private INameEntityRepository repository;

}

Los archivos Service de las entidades deben tener el @Service, para definir que es un servicio como tal.

El @Override sobrescribe el IRepository

* + - 1. Paquete “DTO”.

El DTO es un paquete y entidad opcional pero en términos sencillos es decirle una función específica con atributos específicos a la entidad específica que realice una acción, pero este también posee un archivo base o genérico y también un ApiResponseDto para comunicarse con la API.

ApiResponseDto:

public class ApiResponseDto<T> {

private Boolean status;

private T data;

private String message;

public ApiResponseDto() {

}

public ApiResponseDto(String message, T data, Boolean status) {

this.message = message;

this.data = data;

this.status = status;

}

public Boolean getStatus() {

return status;

}

public void setStatus(Boolean status) {

this.status = status;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

}

Debe ser de clase: *interface*

IGenericDto:

public interface IGenericDto {

Long getId();

Boolean getEstado();

}

Para los DTO de cada unidad es la misma estructura, de clase *interface* con el nombre INameEntityDto y los atributos que se vayan a usar para la acción especifica

* + - 1. Paquete “Controller”.

ABaseController:



El Controller es el encargado de hacer las acciones en la API teniendo:

Get, para ver los datos.

Post, para guardar los datos.

Put, para actualizar los datos.

Delete, para eliminar de física o lógica los datos.

nameEntityController: *ejemplo, PaisController*

@CrossOrigin(origins = "\*")

@RestController

@RequestMapping("v1/api/name-entity")

public class NameEntityController extends ABaseController<NameEntity, INameEntityService>{

protected NameEntityController(INameEntityService service) {

super(service, "NameEntity");

}

}

El @RequestMapping da la url de la API de las entidades

* 1. Frontend.
     1. Crear carpetas y sus archivos.
        1. Carpeta “HTML”.

Se crea una carpeta preferiblemente para mantener el orden entre los archivos, cabe recalcar que cada archivo debe tener el minmo nombre que la entidad del Backend y debe terminar con “.html” (*ejemplo: Pais.html*) y recalcar que los archivos html es diseño básico de las páginas web.

Después de crear el archivo, el documento debe llevar:

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

</body>

</html>

El cual es la estructura base de todo html ya que los documentos html dependen de estilos y demás vinculaciones para tener un diseño apto y agradable, los links que se uso para cada proyecto son:

Para un estilo de “registro de datos” ya vistos por los instructores en anteriores clases, se utilizó un Modal, el Modal es una caja que se carga en la misma página la cual puede mostrar mensajes, formularios o tablas dependiendo del diseño que se desee, un Modal base sería:

Y el botón para activar o mostrar el Modal sería: Dentro de los Modals o en el “body” del Modal, iría el contenido a mostrar, ya sea formulario, tabla, etc. Y los proyectos, en el cuerpo del Modal irían las siguientes etiquetas:

<form>

                <fieldset>

                    <legend>Título</legend>

                    <div *class*="form-group row">

                        <label *class*="col-sm-2 col-form label">Name</label>

                    <div *class*="col-sm-10">

                <input *type*="hidden" *name*="id" *id*="id" *value*="">

            <input *type*="text" *class*="form-control" *name*="name" *id*="name" *value*="" *placeholder*="Name">

                        </div>

                    </div>

                    <div *class*="form-group row">

                        <label *class*="col-sm-2 col-form-label">Department</label>

                        <div *class*="col-sm-10">

                            <select *class*="form-select" *id*="department">

                                <option *selected*>--Seleccione mano--</option>

*<!-- Aquí puedes cargar las ciudades desde el backend -->*

                            </select>

                        </div>

                    </div>

                    <div *class*="col-sm-10">

                        <br>

                        <select *class*="form-select" *id*="state">

                            <option *selected*>--Seleccione--</option>

                            <option *value*="1">Activo</option>

                            <option *value*="0">Inactivo</option>

                        </select>

                    </div>

</fieldset>

</form>

La etiqueta <form></form> es el cuerpo del formulario.

La etiqueta <legend></legend> actúa como título de los formularios o de páginas dependiendo del estilo y diseño que se le quiera dar.

Las etiquetas <div></div> actúan como cajas o contenedores para diferentes etiquetas o vistas que se le dan a las páginas.

La etiqueta <label></label> actúa como cierto subtitulo para ocasiones de formularios en los espacios a rellenar.

La etiqueta <input></input> en este caso es quien hará la vinculación en la configuración *name = “nameAtributo”* y *id = “nameAtributo”* a los atributos encapsulados que se crearon en el backend, para evitar errores deben ser los mismo que se usaron para cada entidad; y con el *type = “ ”* se darán los tipos de formularios a rellenar según los atributos del backend esten definidos (*ejemplo: text = String*).

Hay 2 <select></select> o más en algunas entidades, 1 que siempre aparecerá se trata de el atributo base “estado” o “state” el cual es de tipo Boolean o lógico el cual dira si el dato está activo o inactivo, el 2 se trata sobre una relación foránea o las foreign key de algunas entidades, y el select implementando una función en Javascript que veremos mas adelante, lo que hace es mostrarme y sobreponer los valores existentes y ya agregados de la tabla o entidad de la cual depende (*ejemplo: User, depende de Person, por lo tanto aparecerán los datos de Person al momento de registrar un User*).

* + - 1. Carpeta “JS”.
* Familiarizarse con HTML y CSS:

Antes de comenzar con JavaScript, es fundamental tener una base sólida en HTML y CSS. HTML define la estructura de la página web, mientras que CSS le da estilo y diseño.

* Entender el funcionamiento de JavaScript:

JavaScript se ejecuta en el navegador del usuario, permitiéndote modificar el contenido de la página web en tiempo real. Puedes añadir interactividad, animaciones, juegos y mucho más.

* Insertar un script JavaScript en tu HTML:

Para empezar a trabajar con JavaScript, necesitas crear un archivo con extensión .js y luego incluirlo en tu archivo HTML. Puedes hacerlo utilizando la etiqueta <script>.

* Escribir código JavaScript:

Dentro del archivo .js, puedes escribir código JavaScript que se ejecutará cuando se cargue la página web. Un ejemplo sencillo sería mostrar un mensaje en una alerta.

* Probar y depurar el código:

Es importante probar tu código JavaScript para asegurarte de que funciona correctamente. Puedes utilizar las herramientas de desarrollo integradas en tu navegador para ver errores y corregirlos.

* + - 1. Carpeta “ASSETS”.

son elementos digitales que se utilizan en el desarrollo web para dar vida a una página web. Estos pueden incluir imágenes, vídeos, iconos, fuentes y mucho más.

* Definir los assets necesarios: El primer paso es identificar qué tipo de assets necesitas para tu proyecto web. Algunas preguntas que te puedes hacer son:
* ¿Qué imágenes o vídeos necesito?
* ¿Necesito iconos o ilustraciones?
* ¿Qué tipo de fuente quiero usar?
* Elegir las herramientas adecuadas:

Existen diversas herramientas para crear y editar assets. Algunas opciones populares son:

* Imágenes: Photoshop, GIMP, Canva
* Vídeos: Adobe Premiere Pro, Final Cut Pro, iMovie
* Iconos: Font Awesome, Material Design Icons, Ionicons
* Fuentes: Google Fonts, Adobe Fonts, Font Squirrel
* Crear o adquirir los assets:

Dependiendo de tus habilidades y presupuesto, puedes crear los assets tú mismo o adquirirlos de terceros.

Creación de assets:

* Imágenes: Puedes tomar fotos, dibujar o usar herramientas como Photoshop o GIMP para crear imágenes.
* Vídeos: Puedes grabar vídeos con tu cámara o teléfono, o editar vídeos existentes.
* Iconos: Puedes usar herramientas online como Font Awesome o descargar packs de iconos gratuitos.
* Fuentes: Puedes descargar fuentes gratuitas de Google Fonts o comprar fuentes premium de Adobe Fonts.
* Almacenar y organizar los assets: Es importante almacenar y organizar los assets de forma eficiente para que puedas encontrarlos fácilmente cuando los necesites.
* Usar los assets en tu proyecto web: Una vez que tengas tus assets optimizados y organizados, puedes empezar a usarlos en tu proyecto web. Puedes hacerlo insertando imágenes, vídeos, iconos y fuentes en tu código HTML o CSS.
  + - 1. Despedida.
* Puntos clave de la creación de un backend: Elección del lenguaje de programación: Hay diversos lenguajes de programación para el backend, como Python, Java, Ruby y PHP. La elección dependerá del tipo de proyecto y las preferencias del desarrollador.
* Diseño de la arquitectura: Es importante definir la arquitectura del backend, que incluye la organización de los archivos, las bases de datos y las APIs.
* Implementación de la lógica del negocio: El backend debe implementar la lógica del negocio, que define cómo funciona la aplicación.
* Seguridad: Es fundamental garantizar la seguridad del backend para proteger los datos de los usuarios.
* Puntos clave de la creación de un frontend:
* Elección del framework: Existen frameworks populares para el frontend como React, Angular y Vue.js. La elección dependerá del tipo de proyecto y las preferencias del desarrollador.
* Diseño de la interfaz de usuario: El frontend debe tener una interfaz de usuario atractiva y fácil de usar.
* Implementación de la interacción del usuario: El frontend debe implementar la interacción del usuario, como la navegación, la validación de formularios y la gestión de eventos.
* Optimización del rendimiento: Es importante optimizar el rendimiento del frontend para que la carga de la página sea rápida.

**ENLACE DEL VIDEO:** https://drive.google.com/file/d/14xyIp9Rwk0gdQ\_JV-qfPNm55W8mZQBD9/view?usp=sharing