Introducción

Toda gran aplicación necesita su primera línea de código y Rocky surge precisamente así: como un proyecto que posibilitará llevar el gimnasio al mundo digital.

El objetivo es simple: registrar rutinas, ejercicios, repeticiones y pesos, además llevar un consolidado del progreso con base en metas propuestas por los usuarios, pero detrás de eso hay una oportunidad enorme para aprender cómo se construye una aplicación web desde cero, el cual es justamente el objetivo de este tutorial.

En este recorrido, crearemos paso a paso una versión mínima inicial de Rocky que terminará con un pequeño "Hola Mundo" que no debes subestimar, pues, aunque suene modesto, este estará conectado a una base de datos, con su entidad, su botón y su vista web.

La idea no es solo que funcione, sino que entiendas *por qué* funciona.

Lenguaje y framework seleccionados

Java

Hoy en día hay varios lenguajes nuevos, modernos y minimalistas… pero Java sigue ahí, como ese compañero que lleva años en el gimnasio: constante, fuerte y con buena técnica. Es seguro y orientado a objetos, ideal para organizar una aplicación por capas sin que todo termine en caos.

En lo que conlleva el desarrollo, Java es sinónimo de estabilidad y claridad: dos cosas que todo ingeniero agradece cuando su código decide no compilar a las 3 a.m, así que, en términos simples, Java nos da la musculatura del proyecto: clases, lógica, control y estructura.

Spring Boot

Trabajar con un lienzo blanco en Java puede sentirse como levantar pesas sin calentamiento, pero por eso usamos Spring Boot, un framework que nos permite configurar el servidor, gestionar dependencias, conectar la base de datos, entre otros aspectos, sin morir en el intento.

En pocas palabras: Java pone la fuerza bruta y Spring Boot enseña la técnica, manteniendo la aplicación limpia y organizada, separando cada responsabilidad en su capa: controlador, servicio, repositorio y entidad. Así, si algo falla, sabemos exactamente a quién culpar (de preferencia, no a los que escriben el tutorial).

Dependencias del proyecto

En este punto, nuestro querido *Rocky* empieza a ganar masa muscular… pero en forma de dependencias. Estas son las librerías que hacen que todo funcione detrás del telón.

Dependencia	Descripción
spring-boot-starter-data-jpa	El músculo encargado de la persistencia de datos. Esta dependencia permite conectar las entidades de nuestra aplicación con la base de datos usando Hibernate (ORM), simplificando la creación de consultas y evitando escribir SQL manualmente.
spring-boot-starter-security	El guardaespaldas de Rocky. Proporciona mecanismos de autenticación y autorización para proteger rutas y recursos dentro de la aplicación.
spring-boot-starter-thymeleaf	Nuestro decorador personal. Permite generar páginas HTML dinámicas en el lado del servidor mediante un motor de plantillas, ideal para integrar la lógica de Spring con una interfaz elegante.
spring-boot-starter-validation	El entrenador exigente. Valida los datos que ingresan los usuarios antes de que lleguen a la base de datos, utilizando anotaciones como @NotNull, @Email o @Size.
spring-boot-starter-web	El sistema circulatorio de Rocky. Contiene todo lo necesario para crear controladores REST, manejar peticiones HTTP y servir contenido web.
flyway-core y flyway-mysql	El doble agente del orden en la base de datos. Flyway se encarga de versionar y aplicar migraciones SQL de forma controlada, asegurando que todos los entornos tengan el mismo esquema.
thymeleaf-extras-springsecurity6	El complemento que sincroniza Thymeleaf con Spring Security, permitiendo mostrar u ocultar elementos en las vistas según los roles de usuario (por ejemplo, botones de administración).
spring-boot-devtools	El amigo que acelera los entrenamientos. Detecta cambios en el código y reinicia la aplicación automáticamente durante el desarrollo, agilizando las pruebas.
mysql-connector-j	El puente directo con nuestra base de datos MySQL. Sin él, la aplicación no podría establecer conexión ni ejecutar consultas.

lombok	El asistente invisible. Reduce el código repetitivo con anotaciones como @Getter, @Setter o @Builder, manteniendo el código limpio y legible.
spring-boot-starter-test	El equipo de prueba de Rocky, incluyendo herramientas como JUnit, Mockito y Spring Test para realizar pruebas unitarias y de integración, asegurando que los componentes del proyecto funcionen correctamente antes de desplegarlo.
spring-security-test	El sparring de seguridad. Permite simular contextos de usuarios y roles durante las pruebas, validando que las reglas de acceso y autenticación se apliquen correctamente.

Entendiendo el ORM y la conexión a la base de datos

ORM

Rocky ya tiene músculo (Java) y técnica (Spring Boot), pero le falta algo esencial: memoria. Si cada vez que levanta una pesa olvida cuántas repeticiones hizo, nunca va a progresar, así que ahí entra el ORM, que le permite recordar todo sin tener que escribir SQL a mano.

ORM (Object-Relational Mapping) es una forma elegante de decir que nuestras clases Java y la base de datos finalmente van a hablar el mismo idioma. Trabajamos con objetos (Rutina, Ejercicio, Usuario), y el ORM se encarga de guardarlos, buscarlos o actualizarlos directamente en las tablas.

JPA + Hibernate

El dúo más famoso en este gimnasio es JPA (Java Persistence API) + Hibernate. Ambos trabajan juntos, pero no son lo mismo:

JPA es la teoría, el manual del gimnasio. Define cómo deben declararse las entidades, qué significan las anotaciones (@Entity, @Id, etc.) y qué reglas debe seguir el código para comunicarse con la base de datos. En pocas palabras, JPA establece las normas: es la guía que le dice a Java cómo comportarse cuando quiere guardar algo.

Hibernate, en cambio, es quien aplica esas normas en la práctica. Es la implementación concreta de JPA que realmente hace que todo funcione: crea las tablas, ejecuta las consultas, administra las transacciones y sincroniza los datos entre la aplicación y la base.
Además, añade extras muy útiles, como el manejo de caché, paginación automática o generación de esquemas, haciendo que todo el proceso sea más eficiente.

La base de datos

Ahora que Rocky tiene su musculatura (Java) y su técnica (Spring Boot), necesitamos de un lugar en donde este pueda registrar cada repetición, cada serie y cada progreso que realize el usuario. Ha llegado la hora de construir su memoria de nuestro Rocky “la base de datos” y para ello necesitamos entender a los atletas que viven en la memoria: las entidades.

Las Entidades

Imagina que cada tabla en la base de datos es como una máquina de ejercicio especializada: hay una para usuarios, otra para rutinas, otra para ejercicios. Cada una tiene su propósito y estructura específica.

Una entidad en JPA es como el manual de instrucciones que le dice a Spring Boot:

"Oye, esta clase Java representa una tabla de la base de datos"

"Estos atributos son las columnas de la tabla"

"Aquella anotación significa que es la llave primaria"

Es la forma elegante de decir: "Este objeto sabe cómo guardarse y recuperarse de la base de datos".

Presentando al Primer Atleta: La Entidad Usuario

Rocky necesita conocer a sus usuarios, así que creamos nuestra primera entidad fundamental. Piensa en el Usuario como la ficha de inscripción de cada persona que entra al gimnasio digital.

El Equipamiento del Atleta:

```
None
@Entity
@Table(
    name = "usuario",
    uniqueConstraints = {
        @UniqueConstraint(name = "uk_usuario_email", columnNames = "email")
    }
)
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor @Builder
```

Desglose del equipamiento:

- **@Entity:** El carnet de identidad que dice "soy una entidad de base de datos"
- **@Table:** Las especificaciones de la máquina de ejercicio:
- **name** = "usuario": Esta entidad entrena en la máquina llamada "usuario"
- **uniqueConstraints:** El email debe ser único, como las huellas dactilares
- **Lombok:** El entrenador personal que genera automáticamente los movimientos básicos (getters, setters, constructores)

La Ficha Técnica del Usuario:

Cada usuario necesita sus datos básicos, y aquí definimos exactamente qué información guardaremos:

El Identificador Único

```
None
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

- **@Id:** Marca este campo como la llave primaria, el DNI digital
- **@GeneratedValue:** PostgreSQL generará automáticamente números secuenciales

El Nombre - Quié:

```
None
@NotBlank
@Column(nullable = false, length = 80)
private String nombre;
```

- **@NotBlank:** Validación que dice "este campo no puede estar vacío"
- **nullable** = false: La base de datos exige que siempre haya un nombre
- **length** = 80: Límite de caracteres, como el espacio en la ficha física

El Email - Tu Correo de Contacto

```
None
@Email
@NotBlank
@Column(nullable = false, length = 120)
private String email;
```

- **@Email:** Valida que tenga formato de email válido
- **UNIQUE:** Garantiza que no haya usuarios duplicados

La Contraseña - Tu Llave de Acceso

```
None
// Por ahora texto plano para la prueba; luego lo cambiaremos a hash
(BCrypt)
@NotBlank
@Column(nullable = false, length = 120)
private String password;
```

El Registro Temporal - Tu Fecha en la que Empezaste

```
None
@Column(nullable = false, updatable = false)
private LocalDateTime creadoEn;

@PrePersist
public void prePersist() {
    this.creadoEn = LocalDateTime.now();
}
```

Registra el momento exacto en que el usuario se une a Rocky

Archivos .yml

El gimnasio digital donde cada repetición quede registrada será MySQL, que puede ser levantado usando Docker Compose con configuraciones .yml que harán que todo funcione sin sudar configurando a mano.

1. **docker-compose.yml**

   Antes de correr la app, necesitamos montar el entorno donde Rocky va a entrenar. El siguiente archivo crea dos contenedores: uno para MySQL (la base de datos) y otro para la aplicación (Spring Boot). Ambos se comunican entre sí dentro de la misma red de Docker.

```
None
services:
  mysql:
    image: mysql:8.0
    container_name: rocky_mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASSWORD}
      TZ: ${TZ}
    ports:
      - "${LOCAL_DB_PORT}:${DB_PORT}"
    command: [
      "--default-authentication-plugin=mysql_native_password",
      "--character-set-server=utf8mb4",
      "--collation-server=utf8mb4_0900_ai_ci"
    ]
    volumes:
      - mysql_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-u", "${DB_USER}",
"-p${DB_PASSWORD}"]
      interval: 10s
      timeout: 5s
      retries: 10

  app:
    build: .
    container_name: rocky-app
    depends_on:
      mysql:
        condition: service_healthy
    ports:
      - "${HOST_PORT}:${SERVER_PORT}"
```

```
    environment:
      - SERVER_PORT=${SERVER_PORT}
      - DB_HOST=${DB_HOST}
      - DB_HOST_PORT=${DB_PORT}
      - DB_NAME=${DB_NAME}
      - DB_USER=${DB_USER}
      - DB_PASSWORD=${DB_PASSWORD}
      - TZ=${TZ}

volumes:
  mysql_data:
```

Este archivo es el encargado de armar todo el entorno de entrenamiento**.** En pocas palabras, crea dos contenedores: uno para la base de datos (MySQL) y otro para la aplicación (Spring Boot). Ambos se comunican dentro de la misma red de Docker, sin cables cruzados ni errores de conexión.

Dentro de este archivo pasan varias cosas importantes:

1. Primero, se levanta el contenedor **mysql**, que será la base de datos principal: Le damos nombre, contraseña, base inicial y zona horaria. Además, configuramos el idioma con el que hablará con la app (UTF-8 completo) al igual que un healthcheck que monitorea cuándo MySQL está listo para recibir conexiones.
2. Después, viene el contenedor **app**, que contiene nuestro proyecto Rocky. Este se construye desde el propio código usando su Dockerfile.
   Tiene una instrucción especial: no arranca hasta que la base de datos esté lista, lo cual evita esos típicos errores de conexión donde la aplicación se lanza demasiado rápido. Una vez la BD responde, Rocky se pone en marcha y se conecta usando las variables definidas en el archivo .env.
3. Finalmente, se define un volumen llamado **mysql_data**.
   De tal modo que si apagas el contenedor o reinicias tu máquina, la base de datos no se borra. Rocky no perderá su progreso (al menos no por accidente).

Cuando ejecutas el comando **docker compose up -d,** Docker levanta el gimnasio completo. MySQL se inicia, espera a estar listo y luego la aplicación entra al juego, conectándose automáticamente a la base de datos. A partir de ese momento, cada repetición, rutina o ejercicio que Rocky guarde quedará registrado ahí.

2. application-docker.yml

Cuando Rocky está dentro de su contenedor, necesita saber a dónde enviar sus datos. Ahí entra el archivo application-docker.yml, que le dice a la app cómo conectarse con

la base de datos y cómo comportarse en ese entorno. Es decir, es básicamente su hoja de entrenamiento.

```
None
spring:
  datasource:
    url:
jdbc:mysql://${DB_HOST}:${DB_HOST_PORT}/${DB_NAME}?useSSL=false&serverTimezo
ne=${TZ}&allowPublicKeyRetrieval=true
    username: ${DB_USER}
    password: ${DB_PASSWORD}
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        format_sql: true
        dialect: org.hibernate.dialect.MySQLDialect

  thymeleaf:
    cache: true  # en contenedor suele ser mejor cachear; opcional

server:
  port: ${SERVER_PORT}
  error:
    include-message: always
```

En particular en este archivo:

1. En la sección **spring.datasource**, se definen los datos de conexión con la base. Aquí, en lugar de “localhost”, usamos el nombre del servicio (DB_HOST=mysql), porque dentro del contenedor Docker los servicios se comunican por nombre, no por dirección local. Spring Boot, por su parte, usa estas variables para establecer el enlace con MySQL sin que tengas que tocar nada manualmente.
2. En **spring.jpa**, Hibernate entra en acción: Con ddl-auto: update, la aplicación se encarga de crear o actualizar las tablas automáticamente según las entidades Java. Es como tener un entrenador que ajusta la rutina cada vez que cambias tu plan.

   Además, se muestran las consultas SQL en consola para que puedas ver exactamente cómo trabaja el ORM en cada serie de ejercicios (es decir, operaciones).
3. En **spring.thymeleaf**, se mantiene la caché de vistas activada.

Dentro de Docker no estás editando HTML cada segundo, así que mantener el caché mejora el rendimiento: Menos tiempo de espera, más fuerza bruta. En server, por su parte, se define el puerto donde correrá la aplicación y se activa la opción de mostrar mensajes de error completos (porque incluso los más fuertes necesitan retroalimentación cuando algo falla).

3. **application.yml**

A veces Rocky prefiere entrenar por su cuenta, sin depender del gimnasio Docker. Para esos días, tiene una versión local del application.yml, casi igual a la anterior, pero conectada directamente a una base de datos que corre en tu propio equipo.

```
None
spring:
  datasource:
    url:
jdbc:mysql://${LOCAL_DB_HOST}:${LOCAL_DB_PORT}/${DB_NAME}?useSSL=false&serve
rTimezone=${TZ}&allowPublicKeyRetrieval=true
    username: ${DB_USER}
    password: ${DB_PASSWORD}
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        format_sql: true
        dialect: org.hibernate.dialect.MySQLDialect

  thymeleaf:
    cache: false

server:
  port: ${SERVER_PORT}
  error:
    include-message: always
```

Esta configuración es casi idéntica a la anterior, con solo un cambio importante: el host de conexión (LOCAL_DB_HOST) apunta a localhost, porque ahora MySQL está corriendo fuera del contenedor. Además, el caché de Thymeleaf se desactiva, permitiendo ver los cambios en las vistas inmediatamente, sin tener que reiniciar la aplicación. Perfecto para cuando estás en la jugada.

Ejecución final “Hola Mundo”

La conexión ya está lista gracias a los archivos docker-.yml y application.yml. Spring Boot usa esa configuración para levantar automáticamente la conexión JDBC hacia MySQL.
Sin escribir una sola línea extra, el framework ya:

1. Detecta las entidades (@Entity).
2. Crea las tablas necesarias en la base de datos.
3. Permite guardar y consultar registros usando **JPA** + **Hibernate**.

Estructura general: las capas del proyecto

Spring Boot organiza la aplicación por capas, lo que hace que todo funcione como un cuerpo bien entrenado. Cada músculo tiene su función:

Capa	**Qué hace**	**Ejemplo en Rocky**
Entity	Representa una tabla de la BD. Define los datos y reglas.	Perfil, Usuario
Repository	Es la conexión directa entre Java y MySQL. Usa JPA para guardar o leer entidades.	UsuarioRepository
Service	Contiene la lógica del negocio. Decide qué hacer antes o después de guardar datos.	UsuarioService, PerfilService
Controller	Atiende las solicitudes web. Llama al servicio y manda los datos a las vistas.	UsuarioController
View	Lo que el usuario ve (HTML con Thymeleaf).	index.html, usuarios.html

Instanciación mínima: la entidad Perfil

Para probar la conexión con la base de datos, creamos la entidad sencilla Perfil, que representa la información básica de un usuario en Rocky.

```
None
package seventh_art.rocky.entity;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Positive;
import lombok.*;

@Entity
```

```java
@Table(name = "perfil")
@Getter @Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Perfil {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    @Column(nullable = false, length = 50)
    private String nombre;

    @Column(length = 50)
    private String apellido;

    @Positive
    @Column(nullable = false)
    private int edad;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 10)
    private Sexo sexo;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 30)
    private Objetivo objetivo;

    @Positive
    @Column(nullable = false)
    private int rachaDeseada;  // En días

    @Positive
    @Column(nullable = false)
    private int rachaActual;

    @Positive
    @Column(nullable = false)
    private float estatura;  // En metros

    @Positive
    @Column(nullable = false)
    private float imc;

    @OneToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(
```

```
            name = "idUsuario",
            nullable = false,
            unique = true,
            foreignKey = @ForeignKey(name = "fk_perfil_usuario")
    )
    private Usuario idUsuario;
}
```

Con solo esa clase y la configuración previa, Hibernate genera automáticamente la tabla perfil en la base de datos, con todas sus columnas y restricciones.

Cargando datos iniciales: DataInit

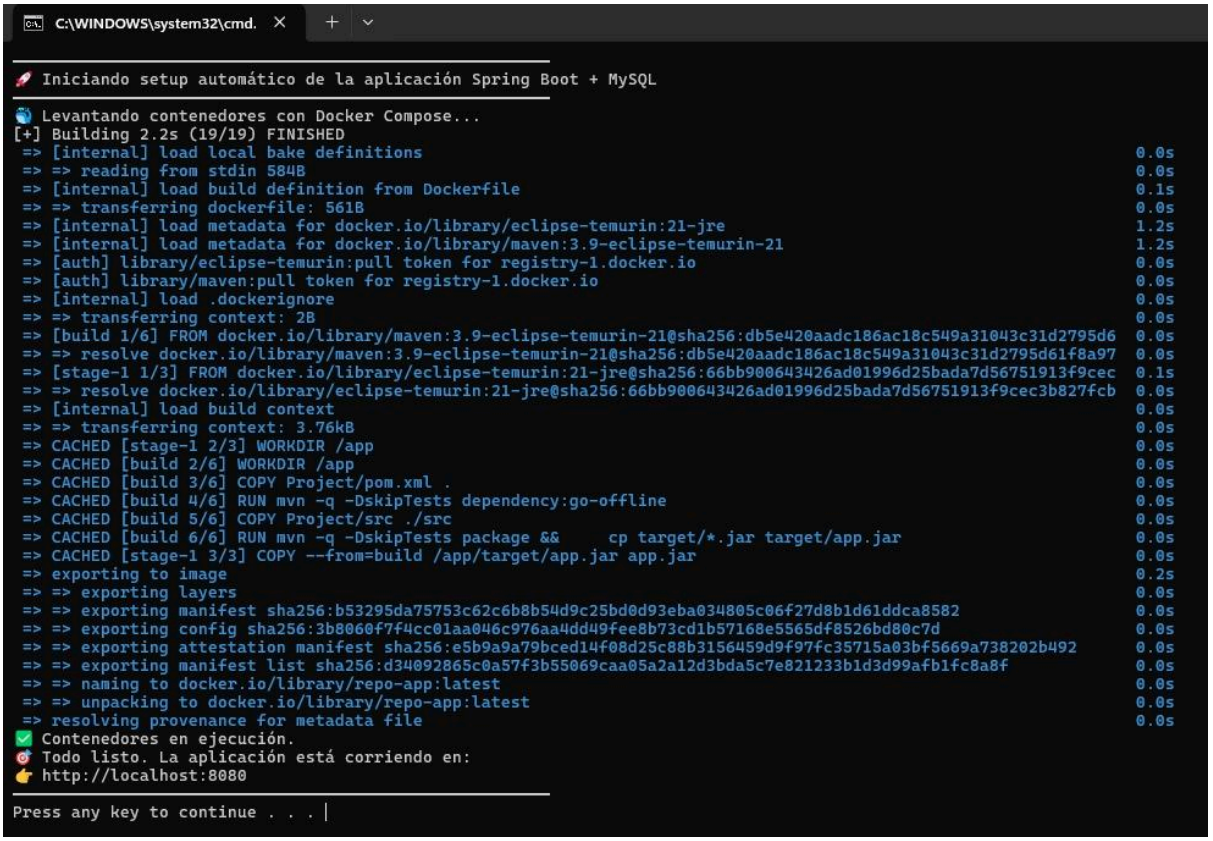
Para darle vida a la aplicación y comprobar que todo está bien conectado, se utiliza una clase DataInit con un CommandLineRunner. Este se ejecuta automáticamente cuando la aplicación arranca y sirve para crear registros iniciales en la base de datos.

```
None
@Bean
CommandLineRunner initUsuarios(UsuarioRepository repo) {
    return args -> {
        repo.deleteAll();

        repo.save(Usuario.builder()
                .nombre("Valentina")
                .email("valentina@gmail.com")
                .password("1234")
                .build());

        repo.save(Usuario.builder()
                .nombre("Nicole")
                .email("nicole@gmail.com")
                .password("5678")
                .build());

        repo.save(Usuario.builder()
                .nombre("Fernanda")
                .email("fernanda@gmail.com")
                .password("1111")
                .build());

        repo.save(Usuario.builder()
                .nombre("Alejandra")
```

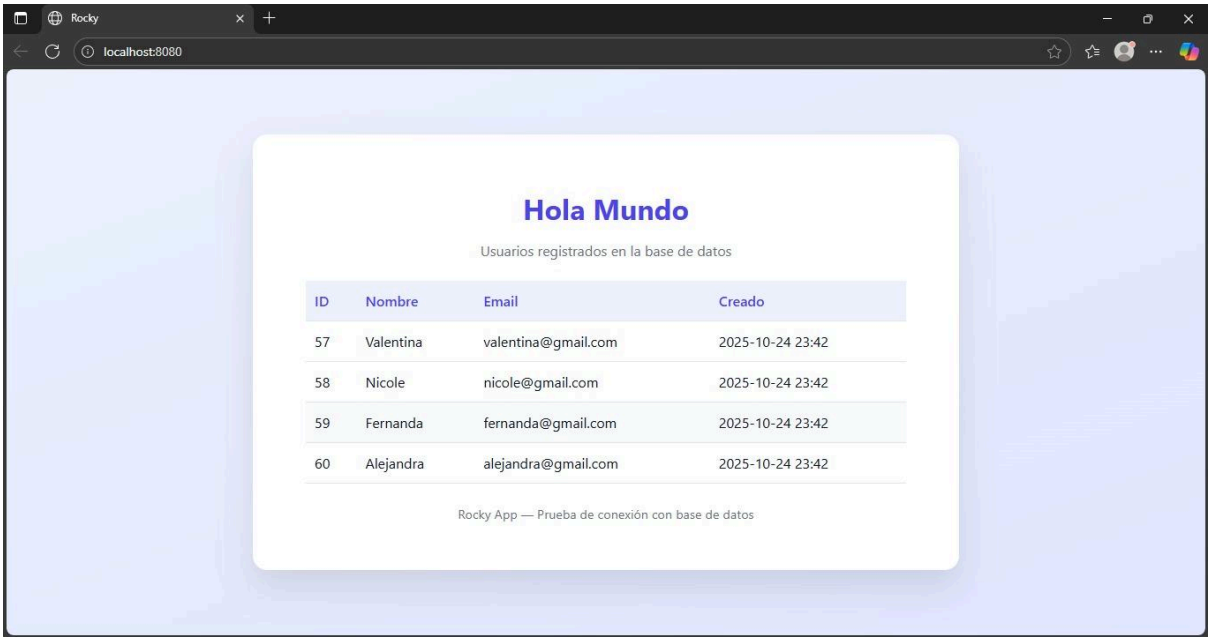
```
                .email("alejandra@gmail.com")
                .password("7777")
                .build());
        };
    }
```

Resultado

Finalmente ejecutamos la app utilizando los archivos de automatización setup.bat (Para Windows) o setup.sh (Para Linux o Mac)



Se ejecuta correctamente y ahora podemos acceder a la aplicación en la web, usando la dirección **http://localhost:8080**



Se carga la interfaz relacionada al Hola Mundo, mostrando además una lista de los usuarios almacenados en la base de datos.

¡Felicidades! Has creado una aplicación web utilizando Java, Spring Boot, Docker Compose, MySql.