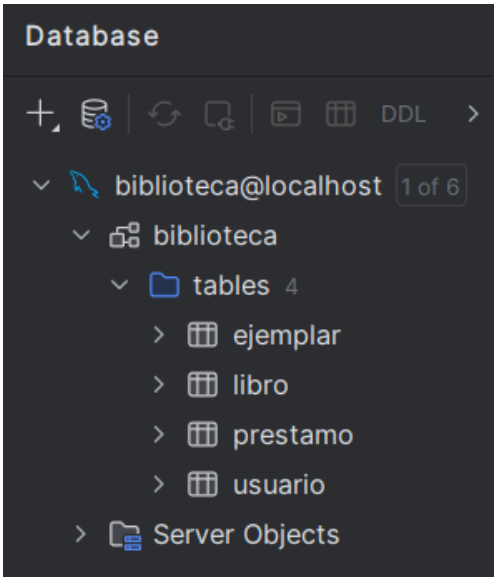
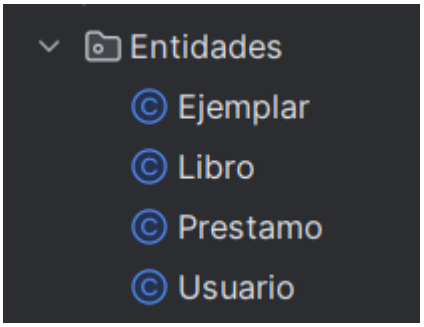
El programa funciona con persistence de jakarta, tengo una base de datos de SQL funcionando en un cliente y me conecto a ella mediante el driver del IDE



Tengo el archivo de configuración (persistence.xml) en la carpeta resources, con el url, user y contraseña. También incluye las entidades para que se haga la interacción entre el codigo y la base de datos.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="https

    <persistence-unit name="biblioteca" transaction-type="RESOURCE_LOCAL">
        <class>Entidades.Usuario</class>
        <class>Entidades.Libro</class>
        <class>Entidades.Ejemplar</class>
        <class>Entidades.Prestamo</class>

        <properties>
            <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/biblioteca?useUnicode=true&amp;characterEncod
            <property name="jakarta.persistence.jdbc.user" value="eric"/>
            <property name="jakarta.persistence.jdbc.password" value="eric"/>
            <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>

    </persistence-unit>
</persistence>
```

tengo todas las entidades creadas dentro de el paquete entidades:



Como ejemplo, asi se ve la clase usuario:

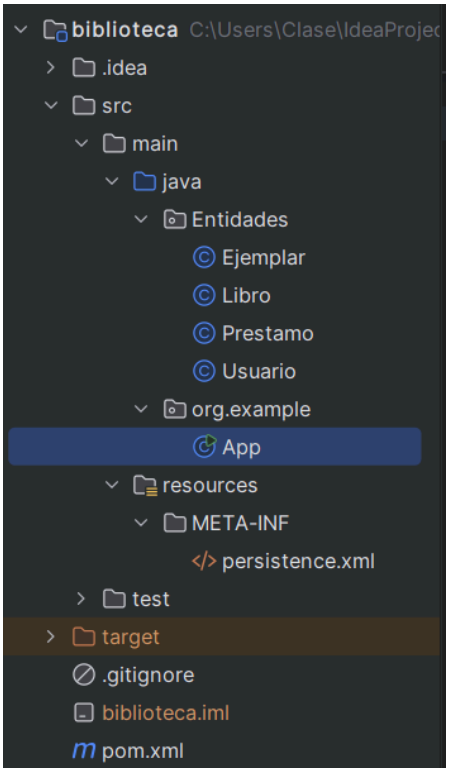
```java
import java.time.LocalDate;

@Entity  13 usages  EricVaquero
@Table(name = "usuario")
public class Usuario {

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

    @Column(nullable = false, unique = true, length = 15)  2 usages
    private String dni;

    @Column(nullable = false, length = 100)  2 usages
    private String nombre;

    @Column(nullable = false, unique = true, length = 100)  2 usages
    private String email;

    @Column(nullable = false)  2 usages
    private String password;

    @Column(nullable = false)  2 usages
    private String tipo;

    private LocalDate penalizacionHasta;  2 usages

    public int getId() { return id; }  1 usage  EricVaquero

    public String getDni() { return dni; }  no usages  EricVaquero
    public void setDni(String dni) { this.dni = dni; }  1 usage  EricVaquero
```

tiene referencias a los valores de las columnas de la base de datos, posible con la libreria de jakarta, y tiene getters y setters.

El pom tiene todas las dependencias para que maven pueda cargar las librerias:

```xml
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>7.1.7.Final</version>
  </dependency>
  <dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.2.0</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>2.0.9</version>
```

Asi queda la estructura final



Todo eso es para que sea posible la sincronización de los datos en la base de datos y que puedan ser usados como objetos en el codigo mediante persistence, la mayoria de la funcionalidad está en la clase App.

```
public class App {  EricVaquero

    private static final EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceU
    private static final EntityManager em = emf.createEntityManager();  27 usages
    private static final Scanner teclado = new Scanner(System.in);  22 usages
```

Lo primero es crear el EntityManagerFactory y el EntityManager, que permiten leer de la base de datos y preparar consultas para que se ejecuten y si hay una respuesta que la reciba.

```
public static void main(String[] args) {  EricVaquero

    login();

    if (usuarioLogueado == null) {
        System.out.println("No se pudo iniciar sesión. Saliendo...");
        cerrar();
        return;
    }

    if (usuarioLogueado.getTipo().equalsIgnoreCase( anotherString: "administrador")) {
        menuAdministrador();
    } else {
        menuUsuarioNormal();
    }

    cerrar();
}
```

Se llama a login, que te deja iniciar sesión, si el usuario es admin, te lleva a las opciones de admin, si es normal, te lleva a las opciones normales:

```
=== Iniciar sesión ===
Email: eric
Password: eric
Hibernate: select u1_0.id,u1_0.dni,u1
Bienvenido eric (administrador)

====== Menú ======
1. Registrar libro
2. Registrar ejemplar
3. Registrar usuario
4. Registrar préstamo
5. Devolver préstamo
6. Mostrar stock disponible
0. Salir
```

```
=== Iniciar sesión ===
Email: 1
Password: 1
Hibernate: select u1_0.id,u1_0.dni,
Bienvenido 1 (normal)

=== Menú Usuario ===
1. Ver mis préstamos
2. Ver ejemplares disponibles
0. Salir
Opción:
```

(No he puesto controles para email, password, nombre, etc)

Este es el metodo login:

```
private static void login() {  1 usage   EricVaquero
    System.out.println("=== Iniciar sesión ===");
    System.out.print("Email: ");
    String email = teclado.nextLine();
    System.out.print("Password: ");
    String password = teclado.nextLine();

    try {
        usuarioLogueado = em.createQuery( s: "SELECT u FROM Usuario u WHERE u.email = :email AND u.password = :pwd", Usuario.class).setParameter( s: "email", email

        System.out.println("Bienvenido " + usuarioLogueado.getNombre() + " (" + usuarioLogueado.getTipo() + ")");
    } catch (NoResultException e) {
        usuarioLogueado = null;
        System.out.println("Usuario o contraseña incorrectos");
    }
}
```

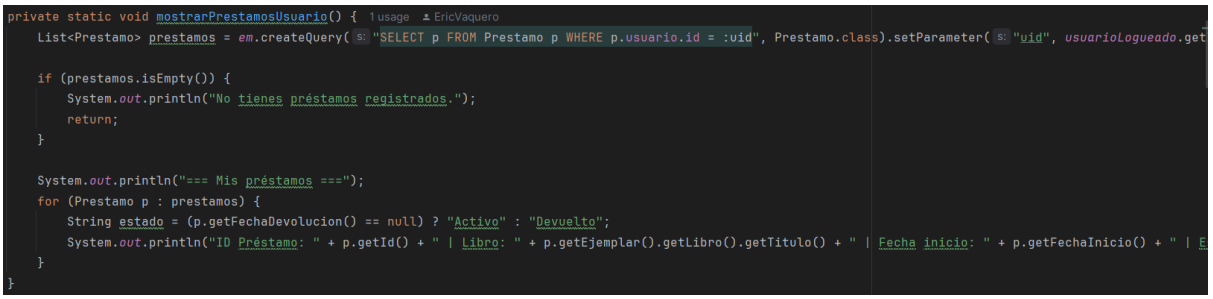
Hace una consulta que te devuelve un usuario (que contiene el valor normal o administrador) en el que corresponda el correo y contraseña introducidos.

```java
private static void menuAdministrador() {  1 usage  EricVaquero
    boolean salir = false;

    while (!salir) {
        System.out.println("\n====== Menú ======");
        System.out.println("1. Registrar libro");
        System.out.println("2. Registrar ejemplar");
        System.out.println("3. Registrar usuario");
        System.out.println("4. Registrar préstamo");
        System.out.println("5. Devolver préstamo");
        System.out.println("6. Mostrar stock disponible");
        System.out.println("0. Salir");
        System.out.print("Opción: ");

        int opcion = teclado.nextInt();
        teclado.nextLine();

        switch (opcion) {
            case 1 -> registrarLibro();
            case 2 -> registrarEjemplar();
            case 3 -> registrarUsuario();
            case 4 -> registrarPrestamo();
            case 5 -> devolverPrestamo();
            case 6 -> mostrarStockDisponible();
            case 0 -> salir = true;
            default -> System.out.println("Opción no válida");
        }
    }
}
```

Este es el menu de administrador, es tan solo un switch que llama a los metodos que hacen cada acción.

```java
private static void menuUsuarioNormal() {  1 usage   EricVaquero
    boolean salir = false;

    while (!salir) {
        System.out.println("\n=== Menú Usuario ===");
        System.out.println("1. Ver mis préstamos");
        System.out.println("2. Ver ejemplares disponibles");
        System.out.println("0. Salir");
        System.out.print("Opción: ");

        int opcion = teclado.nextInt();
        teclado.nextLine();

        switch (opcion) {
            case 1 -> mostrarPrestamosUsuario();
            case 2 -> mostrarStockDisponible();
            case 0 -> salir = true;
            default -> System.out.println("Opcion no válida");
        }
    }
}
```

Este es el menú de usuario normal, le da solo las opciones que puede acceder el usuario normal



```java
private static void mostrarPrestamosUsuario() {  1 usage   EricVaquero
    List<Prestamo> prestamos = em.createQuery( s: "SELECT p FROM Prestamo p WHERE p.usuario.id = :uid", Prestamo.class).setParameter( s: "uid", usuarioLogueado.get

    if (prestamos.isEmpty()) {
        System.out.println("No tienes préstamos registrados.");
        return;
    }

    System.out.println("=== Mis préstamos ===");
    for (Prestamo p : prestamos) {
        String estado = (p.getFechaDevolucion() == null) ? "Activo" : "Devuelto";
        System.out.println("ID Préstamo: " + p.getId() + " | Libro: " + p.getEjemplar().getLibro().getTitulo() + " | Fecha inicio: " + p.getFechaInicio() + " | E
    }
}
```

mostrarPrestamosUsuario() hace una query que selecciona todos los prestamos que tengan el id del usuario actual, si no hay te responde que no hay, si hay, te los muestra todos con un for each

```java
private static void registrarLibro() {  1 usage   EricVaque
    System.out.print("ISBN: ");
    String isbn = teclado.nextLine();
    System.out.print("Título: ");
    String titulo = teclado.nextLine();
    System.out.print("Autor: ");
    String autor = teclado.nextLine();

    Libro libro = new Libro();
    libro.setIsbn(isbn);
    libro.setTitulo(titulo);
    libro.setAutor(autor);

    em.getTransaction().begin();
    em.persist(libro);
    em.getTransaction().commit();

    System.out.println("Libro registrado: " + titulo);
}
```

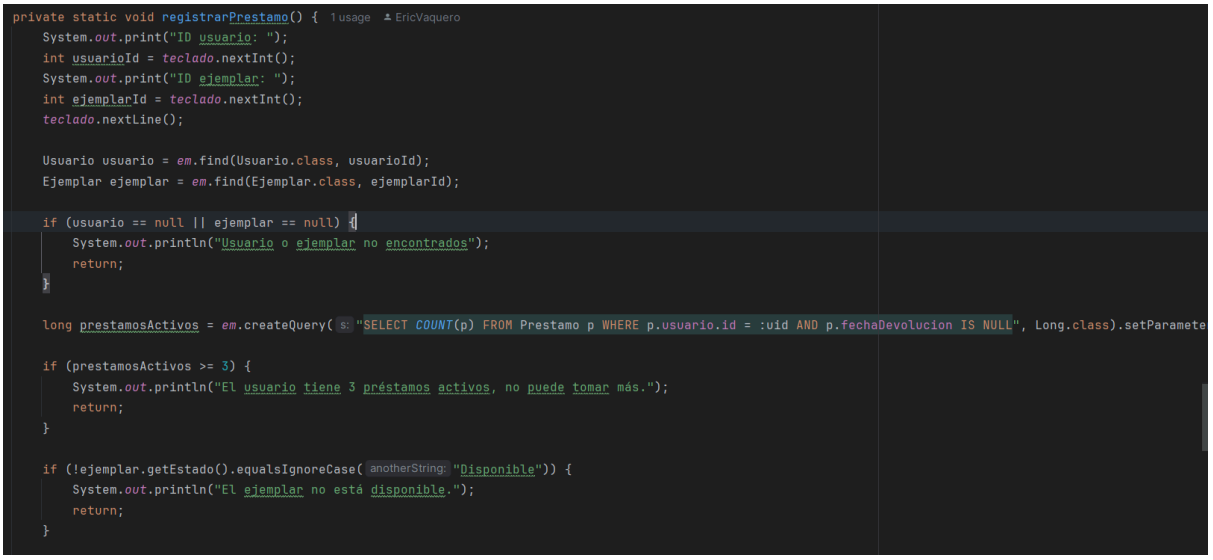
registrarLibro() crea un objeto libro con isbn titulo y autor y luego con el manager hace una transaction que mete el libro en la base de datos.

```java
private static void registrarEjemplar() {  1 usage   EricVaquero
    System.out.print("ISBN del libro: ");
    String isbn = teclado.nextLine();

    Libro libro = em.find(Libro.class, isbn);
    if (libro == null) {
        System.out.println("Libro no encontrado");
        return;
    }

    System.out.print("Estado del ejemplar (Disponible, Prestado, Dañado): ");
    String estado = teclado.nextLine();

    Ejemplar ejemplar = new Ejemplar();
    ejemplar.setLibro(libro);
    ejemplar.setEstado(estado);

    em.getTransaction().begin();
    em.persist(ejemplar);
    em.getTransaction().commit();

    System.out.println("Ejemplar registrado, ID: " + ejemplar.getId());
}
```

Primero usa el manager para buscar si el libro está registrado, luego introduce el estado del libro, crea el objeto y luego hace la transaction para meterlo en la base de datos.

```java
private static void registrarUsuario() {  1 usage   EricVaquero
    System.out.print("DNI: ");
    String dni = teclado.nextLine();
    System.out.print("Nombre: ");
    String nombre = teclado.nextLine();
    System.out.print("Email: ");
    String email = teclado.nextLine();
    System.out.print("Password: ");
    String password = teclado.nextLine();
    System.out.print("Tipo (normal/administrador): ");
    String tipo = teclado.nextLine();

    Usuario usuario = new Usuario();
    usuario.setDni(dni);
    usuario.setNombre(nombre);
    usuario.setEmail(email);
    usuario.setPassword(password);
    usuario.setTipo(tipo);

    em.getTransaction().begin();
    em.persist(usuario);
    em.getTransaction().commit();

    System.out.println("Usuario registrado: " + nombre);
}
```

registrarUsuario() te pide todos los parametros, crea el objeto y hace la transaction para meterlo en la base de datos.



```
private static void registrarPrestamo() {  1 usage   EricVaquero
    System.out.print("ID usuario: ");
    int usuarioId = teclado.nextInt();
    System.out.print("ID ejemplar: ");
    int ejemplarId = teclado.nextInt();
    teclado.nextLine();

    Usuario usuario = em.find(Usuario.class, usuarioId);
    Ejemplar ejemplar = em.find(Ejemplar.class, ejemplarId);

    if (usuario == null || ejemplar == null) {
        System.out.println("Usuario o ejemplar no encontrados");
        return;
    }

    long prestamosActivos = em.createQuery( s: "SELECT COUNT(p) FROM Prestamo p WHERE p.usuario.id = :uid AND p.fechaDevolucion IS NULL", Long.class).setParamete

    if (prestamosActivos >= 3) {
        System.out.println("El usuario tiene 3 préstamos activos, no puede tomar más.");
        return;
    }

    if (!ejemplar.getEstado().equalsIgnoreCase( anotherString: "Disponible")) {
        System.out.println("El ejemplar no está disponible.");
        return;
    }
```

registrarPrestamo pide el ID de usuario y el ID del ejemplar que quiere pedir, el manager los busca, si no los encuentra lo dice, luego se consultan los prestamos que tiene el usuario, basicamente es una query que cuenta las filas en la tabla prestamo donde aparece el id del usuario, si hay mas de tres, no puede tener mas libros pedidos. Si el estado NO es disponible, el ejemplar no esta disponible.

```
if (usuario.getPenalizacionHasta() != null && usuario.getPenalizacionHasta().isAfter(LocalDate.now())) {
    System.out.println("El usuario tiene penalización hasta: " + usuario.getPenalizacionHasta());
    return;
}

Prestamo prestamo = new Prestamo();
prestamo.setUsuario(usuario);
prestamo.setEjemplar(ejemplar);
prestamo.setFechaInicio(LocalDate.now());

ejemplar.setEstado("Prestado");

em.getTransaction().begin();
em.persist(prestamo);
em.merge(ejemplar);
em.getTransaction().commit();

System.out.println("Préstamo registrado, fecha límite: " + LocalDate.now().plusDays( daysToAdd: 15));
```

Si el usuario tiene penalización activa (checkeado con localDate) no se le permite pedir otro libro, si no, se procede a marcar el ejemplar como prestado y se registran los cambios en la base de datos con el manager.

```java
private static void devolverPrestamo() {  1 usage  EricVaquero
    System.out.print("ID préstamo: ");
    int prestamoId = teclado.nextInt();
    teclado.nextLine();

    Prestamo prestamo = em.find(Prestamo.class, prestamoId);
    if (prestamo == null) {
        System.out.println("Préstamo no encontrado");
        return;
    }

    prestamo.setFechaDevolucion(LocalDate.now());

    Ejemplar ejemplar = prestamo.getEjemplar();
    ejemplar.setEstado("Disponible");

    LocalDate limite = prestamo.getFechaInicio().plusDays( daysToAdd: 15);
    if (prestamo.getFechaDevolucion().isAfter(limite)) {
        Usuario usuario = prestamo.getUsuario();
        LocalDate penalizacionActual = usuario.getPenalizacionHasta();
        LocalDate nuevaPenalizacion = LocalDate.now().plusDays( daysToAdd: 15);
        if (penalizacionActual != null && penalizacionActual.isAfter(LocalDate.now())) {
            nuevaPenalizacion = penalizacionActual.plusDays( daysToAdd: 15);
        }
        usuario.setPenalizacionHasta(nuevaPenalizacion);
        em.merge(usuario);
        System.out.println("Usuario penalizado hasta: " + nuevaPenalizacion);
    }
```

```java
    em.getTransaction().begin();
    em.merge(prestamo);
    em.merge(ejemplar);
    em.getTransaction().commit();

    System.out.println("Préstamo devuelto");
}
```

devolverPrestamo pide el id del prestamo, pone la fecha de devolución y pone el libro en disponible, si sobrepasa el limite comparando la fecha de inicio con el dia de hoy (15 dias) se le aplica una penalización, si ya tenia una se le suman otros 15 dias. luego se actualiza toda la información con el manager en la base de datos.

```java
private static void mostrarStockDisponible() {  2 usages  EricVaquero
    List<Ejemplar> disponibles = em.createQuery( s: "SELECT e FROM Ejemplar e WHERE e.estado = 'Disponible'", Ejemplar.class).getResultList();

    System.out.println("=== Stock disponible ===");
    for (Ejemplar e : disponibles) {
        System.out.println("Libro: " + e.getLibro().getTitulo() + " - Ejemplar ID: " + e.getId());
    }
    System.out.println("Total disponibles: " + disponibles.size());
}
```

Por ultimo, mostrarStock hace una query que selecciona todos los ejemplares que tienen el estado “disponible”, luego hace un for each que los muestra y muestra total de libros disponibles.