Universidad Tecnológica Nacional - FRRo

Lenguaje de programación JAVA

Trabajo Práctico Integrador

Página web de una tienda de computación

Integrantes

- 47066 - Gorosito, Adriel - adrielgorosito14@gmail.com
- 47447 - Botali, Santiago - santiagobotali@gmail.com

Docentes

- Meca, Adrian
- Bressano, Mario

CUU: Compra de un producto

Objetivo: Comprar un producto.
Actor principal: Usuario.

Camino principal
1. El cliente ingresa a la página y se loguea con su cuenta en el sistema. El sistema valida que el cliente tiene una cuenta.
2. El cliente se decide por un producto y procede con la compra seleccionandolo. El sistema lo redirecciona para completar el pedido.
3. El cliente elige la cantidad que desea. El sistema valida que el cliente haya cargado previamente una dirección y lo redirecciona para terminar la compra.
4. El sistema muestra cantidad, subtotal, dirección, precio del envío y el total. El cliente elige un método de pago (Rapipago o Pagofácil) y confirma la compra.
5. El sistema registra la compra y actualiza el historial de compras del cliente.

Camino alternativo
1.a. <Durante> El cliente no tiene cuenta.
    1.a.1. El cliente se crea una cuenta y el sistema lo registra.
2.a. <Anterior> El cliente no se decide por ningún producto.
    2.a.1. Fin de CU.
3.a <Durante> El cliente no posee dirección
    3.a.1. El sistema se lo informa al cliente y le recomienda agregar una dirección para poder realizar la compra.
    3.a.2. Vuelve al paso 2.
5.a. <Reemplaza> El cliente no confirma o cancela la compra.
    5.a.1. Fin de CU.

CUR - Restock de un producto

Objetivo: Actualizar el stock de un producto.
Actor principal: Administrador.

Camino Básico
1. El usuario realiza la compra de un producto. **<CUU01: Compra de un producto>**.
2. El administrador emite una lista y visualiza los productos observando que ya no hay más stock de alguno. **<CUU02: Revisión de productos>**.
3. El administrador informa de la situación para poder renovar el stock y cuando lo renueven físicamente, lo actualiza en el sistema. **<CUU03: Actualización de stock>**.

Modelo de dominio

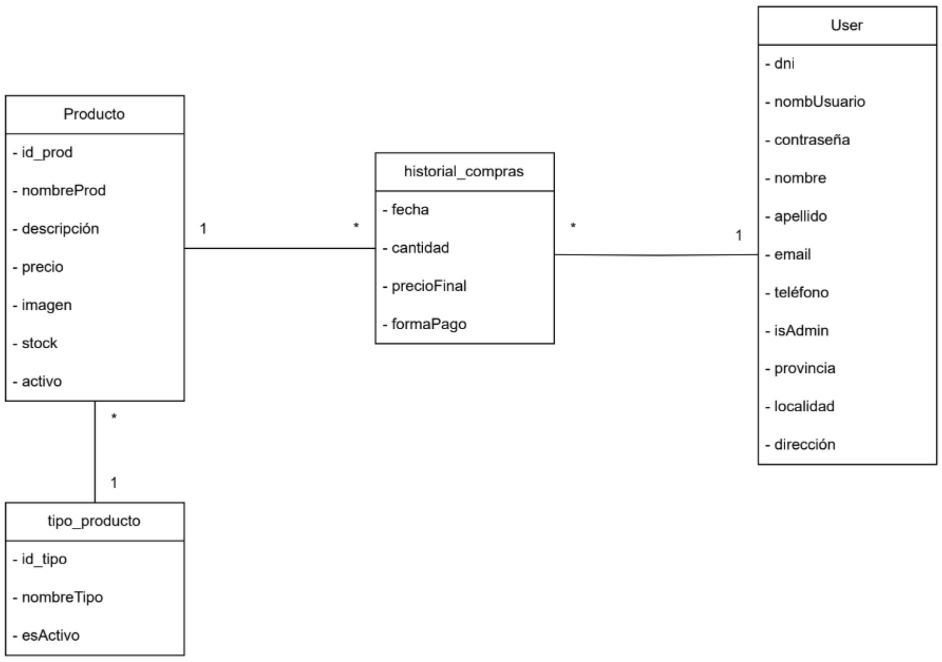
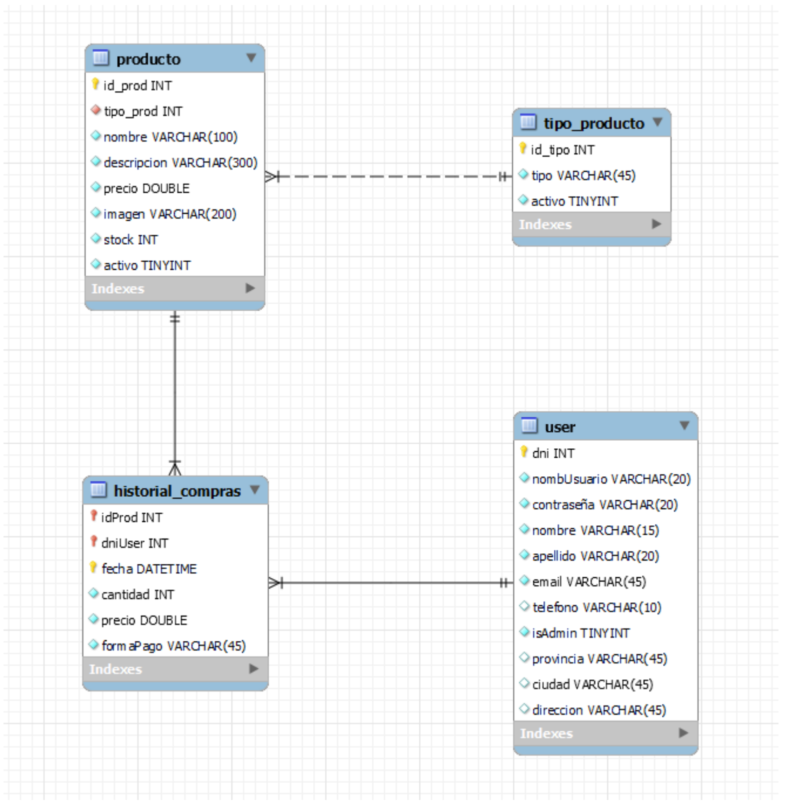
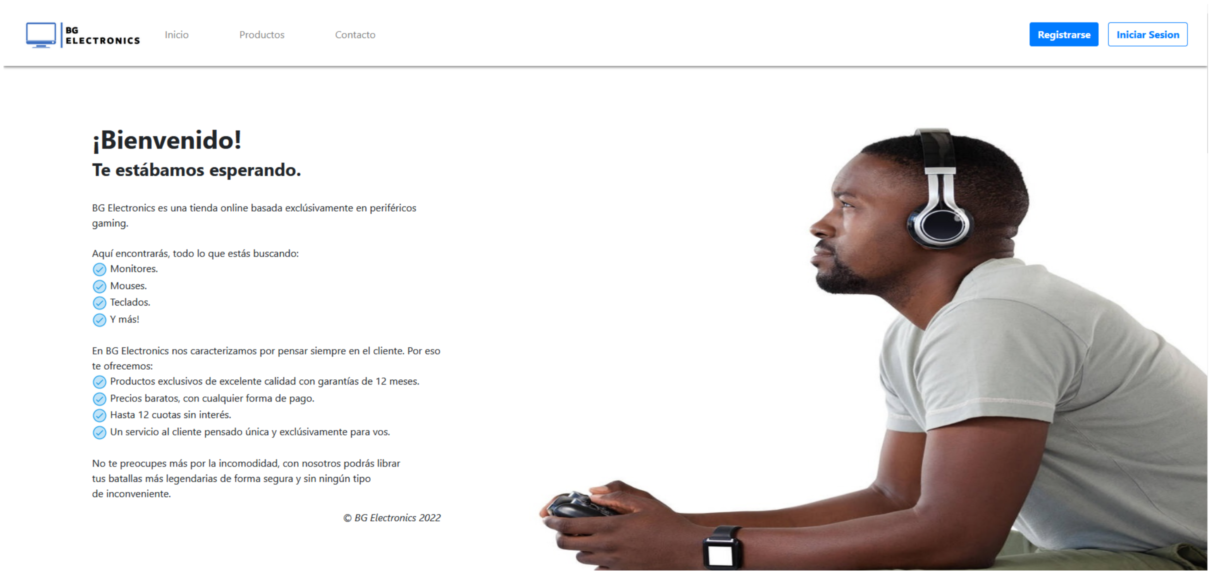


Diagrama de entidad-relación (DER)



Capturas de pantalla del CUU: Compra de un producto



¡Bienvenido!

Te estábamos esperando.

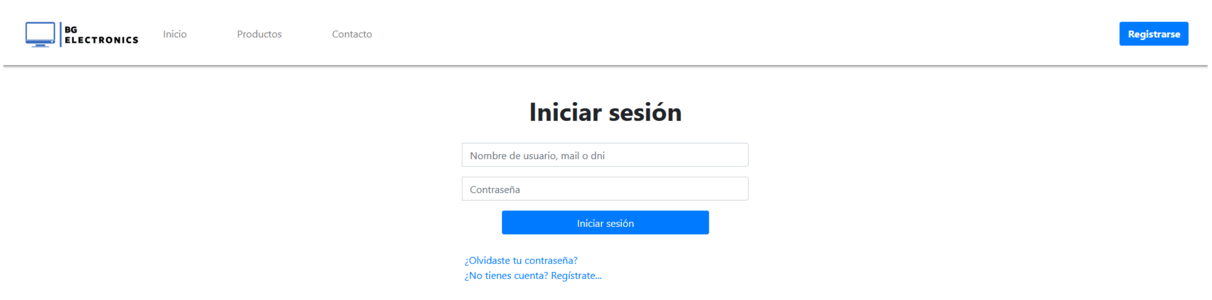
BG Electronics es una tienda online basada exclúsivamente en periféricos gaming.

Aquí encontrarás, todo lo que estás buscando:
- ✓ Monitores.
- ✓ Mouses.
- ✓ Teclados.
- ✓ Y más!

En BG Electronics nos caracterizamos por pensar siempre en el cliente. Por eso te ofrecemos:
- ✓ Productos exclusivos de excelente calidad con garantías de 12 meses.
- ✓ Precios baratos, con cualquier forma de pago.
- ✓ Hasta 12 cuotas sin interés.
- ✓ Un servicio al cliente pensado única y exclúsivamente para vos.

No te preocupes más por la incomodidad, con nosotros podrás librar tus batallas más legendarias de forma segura y sin ningún tipo de inconveniente.

© BG Electronics 2022



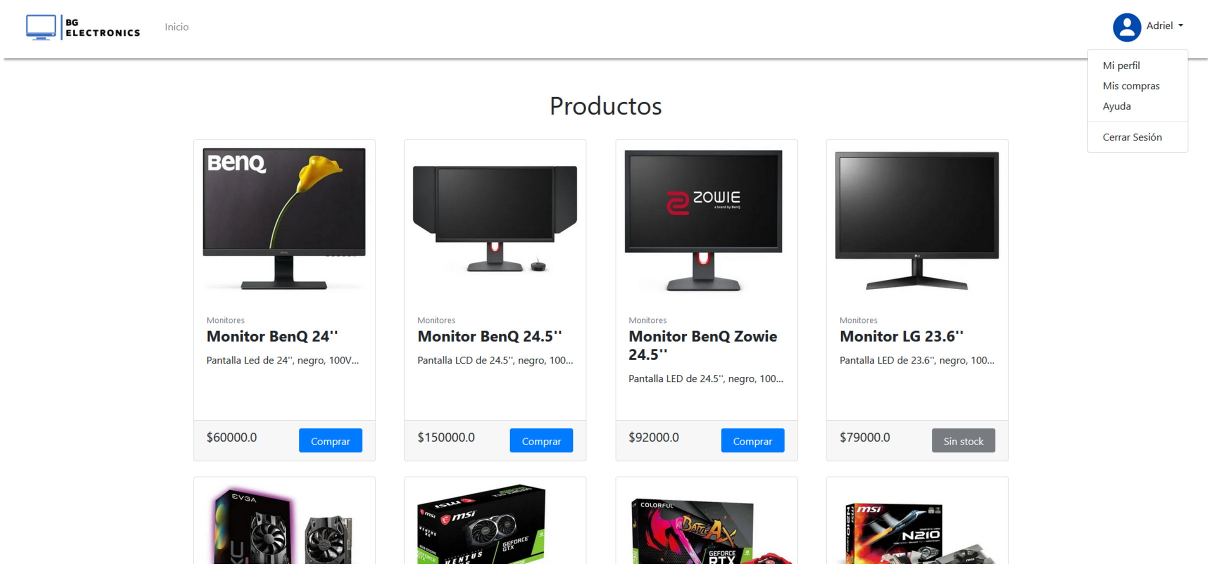
Iniciar sesión

Nombre de usuario, mail o dni

Contraseña

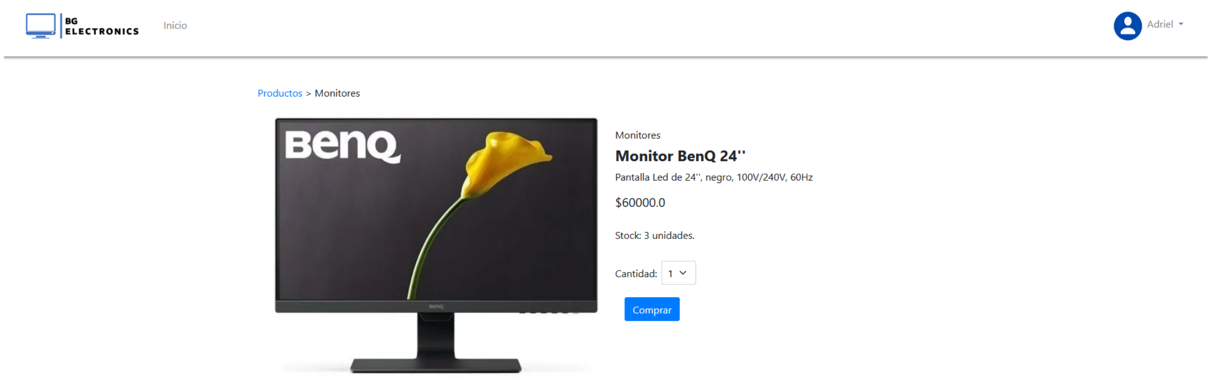
Iniciar sesión

¿Olvidaste tu contraseña?
¿No tienes cuenta? Regístrate...



Productos

Monitores	Monitores	Monitores	Monitores
Monitor BenQ 24''	**Monitor BenQ 24.5''**	**Monitor BenQ Zowie 24.5''**	**Monitor LG 23.6''**
Pantalla Led de 24'', negro, 100V...	Pantalla LCD de 24.5'', negro, 100...	Pantalla LED de 24.5'', negro, 100...	Pantalla LED de 23.6'', negro, 100...
$60000.0 Comprar	$150000.0 Comprar	$92000.0 Comprar	$79000.0 Sin stock



Productos > Monitores

Monitores

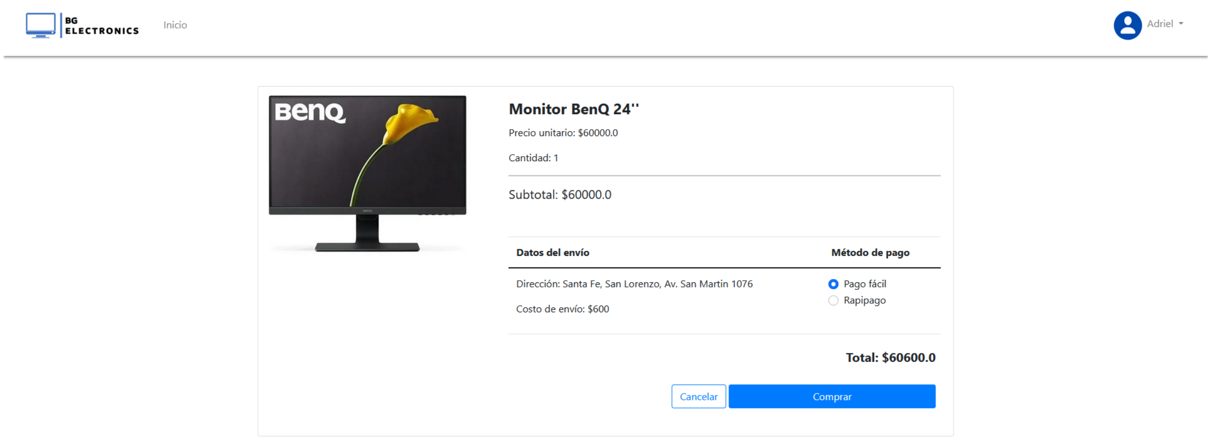
Monitor BenQ 24''

Pantalla Led de 24'', negro, 100V/240V, 60Hz

$60000.0

Stock: 3 unidades.

Cantidad: 1



Monitor BenQ 24''

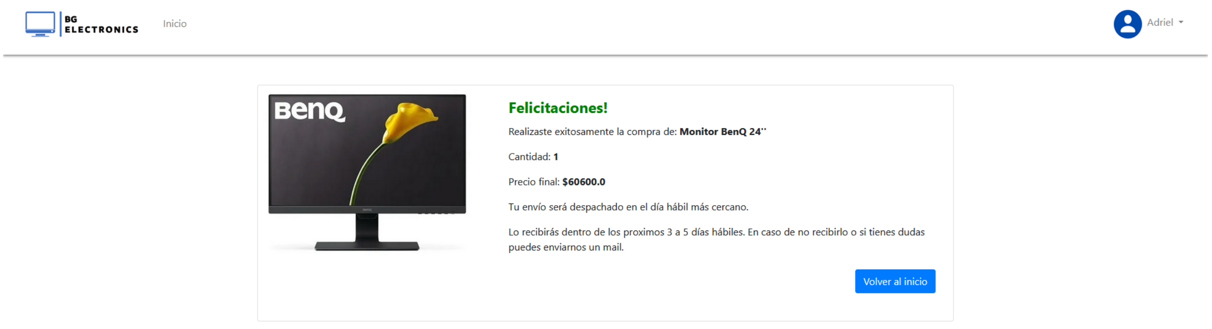
Precio unitario: $60000.0

Cantidad: 1

Subtotal: $60000.0

Datos del envío	**Método de pago**
Dirección: Santa Fe, San Lorenzo, Av. San Martin 1076	[✓ Pago fácil]
Costo de envío: $600	Rapipago

Total: $60600.0



Felicitaciones!

Realizaste exitosamente la compra de: **Monitor BenQ 24''**

Cantidad: **1**

Precio final: **$60600.0**

Tu envío será despachado en el día hábil más cercano.

Lo recibirás dentro de los proximos 3 a 5 días hábiles. En caso de no recibirlo o si tienes dudas puedes enviarnos un mail.

Código

Servlets

Servlet Login

```
public class Login extends HttpServlet {
	private static final long serialVersionUID = 1L;

	private CtrlLogin cl = new CtrlLogin();

	public Login() {
		super();
	}

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		doPost(request, response);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

		User userFound = cl.userExists(request.getParameter("userInput"),
request.getParameter("userPass"));

		if (userFound != null) {
			HttpSession sesion = request.getSession();
			sesion.setAttribute("userSession", userFound);
			sesion.setMaxInactiveInterval(30*60);

			if (userFound.isAdmin()) {
				request.getRequestDispatcher("indexAdmin.jsp").forward(requ
				est, response);
			} else {
				request.getRequestDispatcher("indexUser.jsp").forward(reques
				t, response);
			}
		} else {
			request.setAttribute("errorType", 1);
			request.getRequestDispatcher("error.jsp").forward(request, response);
		}
	}
}
```

Servlet ProductS

```
public class ProductS extends HttpServlet {
	private static final long serialVersionUID = 1L;

	public ProductS() {
		super();
	}

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		doPost(request, response);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		CtrlProduct cp = new CtrlProduct();
		Product p =
cp.getProduct(Integer.parseInt(request.getParameter("id_prod")));

		if (p.getStock() <= 0 || p == null) {
			request.setAttribute("errorType", 13);
			request.getRequestDispatcher("error.jsp").forward(request, response);
		} else {
			request.setAttribute("prod", p);
			request.getRequestDispatcher("product.jsp").forward(request,
			response);
		}
	}
}
```

Servlet BuyProduct

```
public class BuyProduct extends HttpServlet {
	private static final long serialVersionUID = 1L;

	public BuyProduct() {
		super();
	}

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		doPost(request, response);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		CtrlProduct cp = new CtrlProduct();
		Product p =
cp.getProduct(Integer.parseInt(request.getParameter("id_prod")));

		HttpSession session1 = request.getSession();
		User u1 = (User) session1.getAttribute("userSession");

		int cant = Integer.parseInt(request.getParameter("quantityInput"));

		if (p.getStock() >= cant) {
				if (u1.getState() != null && u1.getCity() != null &&
					u1.getAddress() != null) {
				request.setAttribute("prod", p);
				request.setAttribute("cant", cant);
				request.getRequestDispatcher("buy.jsp").forward(request,
				response);
			} else {
				request.setAttribute("errorType", 20);
				request.getRequestDispatcher("error.jsp").forward(request,
				response);
			}
		} else {
			request.setAttribute("errorType", 13);
			request.getRequestDispatcher("error.jsp").forward(request, response);
		}
	}
}
```

Servlet FinishBuy

```
public class FinishBuy extends HttpServlet {
	private static final long serialVersionUID = 1L;

	public FinishBuy() {
		super();
	}

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		doPost(request, response);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		HttpSession session1 = request.getSession();
		User u1 = (User) session1.getAttribute("userSession");

		CtrlProduct cp = new CtrlProduct();
		CtrlShoppingHistory csh = new CtrlShoppingHistory();

		Product p =
cp.getProduct(Integer.parseInt(request.getParameter("id_prod")));
		int cant = Integer.parseInt(request.getParameter("quantityInput"));
		String metodoPago = request.getParameter("metodoPago");

		if (p.getStock() >= cant) {
			p.setStock(cant);
			cp.updateStock(p);

			ShoppingHistory sh = new ShoppingHistory(p, u1,
				LocalDateTime.now(), cant, p.getPrice()*cant+600,
				metodoPago);
			csh.newShoppingHistory(sh);

			request.setAttribute("prod", p);
			request.setAttribute("cant", cant);
			request.setAttribute("metodo", metodoPago);
			request.getRequestDispatcher("successBuy.jsp").forward(request,
			response);
		}
	}
}
```

Controladores

CtrlLogin

```
public class CtrlLogin {
	private DataUsers du = new DataUsers();

	public User userExists(String word, String pass) {
		User u = new User();
		if (word.contains("@")) {
			u.setMail(word);
			u.setPassword(pass);
			return du.loginByMail(u);
		} else {
			u = new User(word, pass);
			if (du.loginByUsername(u) != null) {
				return du.loginByUsername(u);
			} else {
				try {
					u.setDni(Integer.parseInt(word));
					u.setPassword(pass);
					return du.loginByDni(u);
				} catch (NumberFormatException n) {
					return null;
				}
			}
		}
	}
}
```

CtrlProduct

```
public class CtrlProduct {
	private DataProducts dp = new DataProducts();

	public LinkedList<Product> getActiveProducts() {
		return dp.getActiveProducts();
	}

	public Product getProduct(int id) {
		Product p = new Product();
		p.setId_prod(id);
		return dp.searchProductById(p);
	}

	public void updateStock(Product p) {
		dp.updateStock(p);
	}
}
```

<u>CtrlShoppingHistory</u>

```
public class CtrlShoppingHistory {
	private DataShoppingHistory dsh = new DataShoppingHistory();

	public void newShoppingHistory(ShoppingHistory sh) {
		dsh.addNewShoppingHistory(sh);
	}
}
```

Data

DataUsers

```
public class DataUsers {

    public User loginByUsername(User u)  {
        PreparedStatement stmt = null;
        ResultSet rs = null;
        User userFound = null;

        try {
            stmt = DbConnector.getInstancia().getConn().prepareStatement(
                "SELECT dni, nombUsuario, nombre, apellido, email, telefono,
                isAdmin, provincia, ciudad, direccion FROM User WHERE
                nombUsuario = ? AND contraseña = ?");
            stmt.setString(1, u.getUsername());
            stmt.setString(2, u.getPassword());
            rs = stmt.executeQuery();

            if (rs != null && rs.next()) {
                userFound = new User();
                userFound.setDni(rs.getInt("dni"));
                userFound.setUsername(rs.getString("nombUsuario"));
                userFound.setName(rs.getString("nombre"));
                userFound.setSurname(rs.getString("apellido"));
                userFound.setMail(rs.getString("email"));
                userFound.setPhone(rs.getString("telefono"));
                userFound.setAdmin(rs.getBoolean("isAdmin"));
                userFound.setState(rs.getString("provincia"));
                userFound.setCity(rs.getString("ciudad"));
                userFound.setAddress(rs.getString("direccion"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null)
                    rs.close();
                if (stmt != null)
                    stmt.close();
                DbConnector.getInstancia().releaseConn();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return userFound;
    }
```

```
public User loginByMail(User u)  {
	PreparedStatement stmt = null;
	ResultSet rs = null;
	User userFound = null;

	try {
		stmt = DbConnector.getInstancia().getConn().prepareStatement(
			"SELECT dni, nombUsuario, nombre, apellido, email, telefono,
			isAdmin, provincia, ciudad, direccion FROM User WHERE email = ?
			AND contraseña = ?");
		stmt.setString(1, u.getMail());
		stmt.setString(2, u.getPassword());
		rs = stmt.executeQuery();

		if (rs != null && rs.next()) {
			userFound = new User();
			userFound.setDni(rs.getInt("dni"));
			userFound.setUsername(rs.getString("nombUsuario"));
			userFound.setName(rs.getString("nombre"));
			userFound.setSurname(rs.getString("apellido"));
			userFound.setMail(rs.getString("email"));
			userFound.setPhone(rs.getString("telefono"));
			userFound.setAdmin(rs.getBoolean("isAdmin"));
			userFound.setState(rs.getString("provincia"));
			userFound.setCity(rs.getString("ciudad"));
			userFound.setAddress(rs.getString("direccion"));
		}
	} catch (SQLException e) {
		e.printStackTrace();
	} finally {
		try {
			if (rs != null)
				rs.close();

			if (stmt != null)
				stmt.close();

			DbConnector.getInstancia().releaseConn();
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
	return userFound;
}
```

```
  public User loginByDni(User u)  {
        PreparedStatement stmt = null;
        ResultSet rs = null;
        User userFound = null;

        try {
                stmt = DbConnector.getInstancia().getConn().prepareStatement(
                        "SELECT dni, nombUsuario, nombre, apellido, email, telefono,
                        isAdmin, provincia, ciudad, direccion FROM User WHERE dni = ?
                        AND contraseña = ?");
                stmt.setInt(1, u.getDni());
                stmt.setString(2, u.getPassword());
                rs = stmt.executeQuery();

                if (rs != null && rs.next()) {
                        userFound = new User();
                        userFound.setDni(rs.getInt("dni"));
                        userFound.setUsername(rs.getString("nombUsuario"));
                        userFound.setName(rs.getString("nombre"));
                        userFound.setSurname(rs.getString("apellido"));
                        userFound.setMail(rs.getString("email"));
                        userFound.setPhone(rs.getString("telefono"));
                        userFound.setAdmin(rs.getBoolean("isAdmin"));
                        userFound.setState(rs.getString("provincia"));
                        userFound.setCity(rs.getString("ciudad"));
                        userFound.setAddress(rs.getString("direccion"));
                }
        } catch (SQLException e) {
                e.printStackTrace();
        } finally {
                try {
                        if (rs != null)
                                rs.close();

                        if (stmt != null)
                                stmt.close();

                        DbConnector.getInstancia().releaseConn();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }
        return userFound;
        }
}
```

<u>DataProduct</u>

```
public class DataProducts {

   public Product searchProductById(Product p)  {
               PreparedStatement stmt = null;
               ResultSet rs = null;
               Product p2 = null;
               try {
                       stmt = DbConnector.getInstancia().getConn().prepareStatement(
                               "SELECT * FROM Producto p INNER JOIN tipo_producto tp
                               ON p.tipo_prod = tp.id_tipo WHERE id_prod = ?");
                       stmt.setInt(1, p.getId_prod());
                       rs = stmt.executeQuery();

                       if (rs != null && rs.next()) {
                               p2 = new Product();
                               p2.setId_prod(rs.getInt("id_prod"));
                               p2.setName(rs.getString("nombre"));
                               p2.setDescription(rs.getString("descripcion"));
                               p2.setPrice(rs.getDouble("precio"));
                               p2.setImg(rs.getString("imagen"));
                               p2.setStock(rs.getInt("stock"));
                               p2.setActive(rs.getBoolean("activo"));

                               ProductType pt = new ProductType();
                               pt.setId(rs.getInt("id_tipo"));
                               pt.setName(rs.getString("tipo"));
                               pt.setActive(rs.getBoolean("activo"));
                               p2.setType(pt);
                       }
               } catch (SQLException e) {
                       e.printStackTrace();
               } finally {
                       try {
                               if (rs != null) rs.close();
                               if (stmt != null) stmt.close();
                               DbConnector.getInstancia().releaseConn();
                       } catch (SQLException e) {
                               e.printStackTrace();
                       }
               }
               return p2;
        }
```

```
public void updateStock(Product p)  {
        PreparedStatement pstmt = null;
        PreparedStatement pstmt2 = null;
        ResultSet rs = null;

        int cantComprada = p.getStock();

        try {
            DbConnector.getInstancia().getConn().setAutoCommit(false);

            pstmt = DbConnector.getInstancia().getConn().prepareStatement(
                     "SELECT stock FROM Producto WHERE id_prod = ?");
            pstmt.setInt(1, p.getId_prod());
            rs = pstmt.executeQuery();

            if (rs != null && rs.next()) {
                p.setStock(rs.getInt("stock"));

                if (p.getStock() >= cantComprada) {
                    pstmt2 = DbConnector.getInstancia().getConn().
                             prepareStatement("UPDATE Producto SET
                             stock = ? WHERE id_prod = ?");
                    pstmt2.setInt(1, p.getStock() - cantComprada);
                    pstmt2.setInt(2, p.getId_prod());
                    pstmt2.executeUpdate();
                } else {
                    throw new SQLException("No hay suficiente stock
                    disponible para realizar la compra.");
                }
            }

            DbConnector.getInstancia().getConn().commit();
            DbConnector.getInstancia().getConn().setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();

            try {
                if (DbConnector.getInstancia().getConn() != null)
                    DbConnector.getInstancia().getConn().rollback();
            } catch (SQLException e2) {
                e2.printStackTrace();
            }
        } finally {
            try {
                if (pstmt != null)
                    pstmt.close();

                if (pstmt2 != null)
```

```
                                        pstmt2.close();

                                if (rs != null) {
                                        rs.close();
                                }

                        DbConnector.getInstancia().releaseConn();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }
}

public LinkedList<Product> getActiveProducts() {
                PreparedStatement stmt = null;
                ResultSet rs = null;
                LinkedList<Product> allProducts = new LinkedList<>();

                try {
                        stmt = DbConnector.getInstancia().getConn().prepareStatement(
                                "SELECT * FROM Producto p INNER JOIN Tipo_producto tp
                                ON p.tipo_prod = tp.id_tipo WHERE p.activo = 1");
                        rs = stmt.executeQuery();

                        if (rs != null) {
                                while (rs.next()) {
                                        Product p = new Product();
                                        p.setId_prod(rs.getInt("id_prod"));
                                        p.setName(rs.getString("nombre"));
                                        p.setDescription(rs.getString("descripcion"));
                                        p.setPrice(rs.getDouble("precio"));
                                        p.setImg(rs.getString("imagen"));
                                        p.setStock(rs.getInt("stock"));
                                        p.setActive(rs.getBoolean("activo"));

                                        ProductType pt = new ProductType();
                                        pt.setId(rs.getInt("id_tipo"));
                                        pt.setName(rs.getString("tipo"));
                                        pt.setActive(rs.getBoolean("activo"));
                                        p.setType(pt);

                                        allProducts.add(p);
                                }
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                } finally {
                        try {
```

```
                    if (rs != null)
                        rs.close();

                    if (stmt != null)
                        stmt.close();

                    DbConnector.getInstancia().releaseConn();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            return allProducts;
        }
  }
```

<u>DataShoppingHistory</u>

```
public class DataShoppingHistory {

        public void addNewShoppingHistory(ShoppingHistory sh) {
            PreparedStatement stmt = null;

            try {
                stmt = DbConnector.getInstancia().getConn().prepareStatement(
                     "INSERT INTO Historial_compras VALUES (?, ?, ?, ?, ?, ?)");

                stmt.setInt(1, sh.getProd().getId_prod());
                stmt.setInt(2, sh.getU().getDni());
                stmt.setTimestamp(3, Timestamp.valueOf(sh.getFecha()));
                stmt.setInt(4, sh.getCantidad());
                stmt.setDouble(5, sh.getPrecio());
                stmt.setString(6, sh.getFormaPago());

                stmt.executeUpdate();
            } catch (SQLException e) {
        e.printStackTrace();
            } finally {
        try {
          if (stmt != null)
            stmt.close();
          DbConnector.getInstancia().releaseConn();
        } catch (SQLException e) {
            e.printStackTrace();
        }
            }

        }
}
```

Entidades

Product

```
public class Product {
	private int id_prod;
	private String name;
	private String description;
	private Double price;
	private String img;
	private int stock;
	private boolean isActive;
	private ProductType type;

	public int getId_prod() {
		return id_prod;
	}
	public void setId_prod(int id_prod) {
		this.id_prod = id_prod;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getDescription() {
		return description;
	}
	public void setDescription(String description) {
		this.description = description;
	}
	public Double getPrice() {
		return price;
	}
	public void setPrice(Double price) {
		this.price = price;
	}
	public String getImg() {
		return img;
	}
	public void setImg(String img) {
		this.img = img;
	}
	public int getStock() {
		return stock;
	}
	public void setStock(int stock) {
		this.stock = stock;
	}
	public boolean isActive() {
		return isActive;
	}
	public void setActive(boolean isActive) {
		this.isActive = isActive;
	}

	public ProductType getType() {
		return type;
	}
	public void setType(ProductType type) {
		this.type = type;
	}
}
```

<u>ProductType</u>

```
public class ProductType {
	private int id;
	private String name;
	private boolean isActive;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public boolean isActive() {
		return isActive;
	}
	public void setActive(boolean isActive) {
		this.isActive = isActive;
	}
}
```

```
}
```

User

```
public class User {
	private int dni;
	private String username;
	private String password;
	private String name;
	private String surname;
	private String mail;
	private String phone;
	private boolean isAdmin;
	private String state;
	private String city;
	private String address;

	public User(String username, String password) {
		this.setUsername(username);
		this.setPassword(password);
	}

	public User() {
	}

	public int getDni() {
		return dni;
	}
	public void setDni(int dni) {
		this.dni = dni;
	}
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getSurname() {
		return surname;
	}
	public void setSurname(String surname) {
		this.surname = surname;
	}
	public String getMail() {
		return mail;
	}
	public void setMail(String mail) {
		this.mail = mail;
	}
	public String getPhone() {
		return phone;
	}
	public void setPhone(String phone) {
		this.phone = phone;
	}
	public boolean isAdmin() {
		return isAdmin;
	}
	public void setAdmin(boolean isAdmin) {
		this.isAdmin = isAdmin;
	}
	public String getState() {
		return state;
	}
	public void setState(String state) {
		this.state = state;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}
}
```

ShoppingHistory

```
public class ShoppingHistory {
	private Product prod;
	private User u;
	private LocalDateTime fecha;
	private int cantidad;
	private double precio;
	private String formaPago;

	public ShoppingHistory(Product prod, User u, LocalDateTime now, int cant,
				Double price, String metodoPago) {
		this.setProd(prod);
		this.setU(u);
		this.setFecha(now);
		this.setCantidad(cant);
		this.setPrecio(price);
		this.setFormaPago(metodoPago);
	}

	public ShoppingHistory() {}

	public Product getProd() {
		return prod;
	}
	public void setProd(Product prod) {
		this.prod = prod;
	}
	public User getU() {
		return u;
	}
	public void setU(User u) {
		this.u = u;
	}
	public LocalDateTime getFecha() {
		return fecha;
	}
	public void setFecha(LocalDateTime fecha) {
		this.fecha = fecha;
	}
	public int getCantidad() {
		return cantidad;
	}
	public void setCantidad(int cantidad) {
		this.cantidad = cantidad;
	}
	public double getPrecio() {
		return precio;
	}
	public void setPrecio(double precio) {
		this.precio = precio;
	}
	public String getFormaPago() {
		return formaPago;
	}
	public void setFormaPago(String formaPago) {
		this.formaPago = formaPago;
	}
}
```

Repositorio de Github

https://github.com/adrielgorosito/javaTP