**PROYECTO DE AULA**

**DISEÑO DE ROPA Y PERSONALIZACION DE PRENDAS DE VESTIR**

**BRAHIAN GARCES MARTINEZ**

**JUAN JOSE MONTOYA**

**SAMUEL CASTAÑO**

**ESTRUCTURA DE DATOS**

**CHRISTIAN DELANY**

**INSTITUCION UNIVERSITARIA SALAZAR Y HERRERA**

**20 DE NOVIEMBRE DE 2024**

**1. Descripción del MVP:**

El MVP de la plataforma de diseño de ropa personalizada permite a los usuarios registrarse, iniciar sesión y diseñar prendas. Ahora incluye la selección de tipos de prendas, colores, tejidos, detalles opcionales y la capacidad de ajustar la talla de las prendas. Los usuarios también pueden ver una vista previa de sus diseños.

**2. Tecnologías Seleccionadas y Diseño:**

Lenguaje de Programación: C#

Framework: .NET Core (versión más reciente)

Repositorio de Código: GitHub

**3. Modelo de Datos:**

Usuario: Contiene propiedades como correo, contraseña y una lista enlazada de prendas diseñadas.

Prenda: Contiene propiedades como tipo, color, tejido, detalles y una nueva propiedad talla (por defecto "Estándar", pero ajustable por el usuario).

**4. Controladores:**

Controlar usuario: Maneja el registro de usuarios e inicio de sesión.

Controlador deprendas: Permite a los usuarios diseñar prendas, ajustar la talla, obtener una vista previa de la prenda.

**5. Vistas:**

Formularios gráficos para:

Registro de Usuario

Inicio de Sesión

Diseño de Prendas

Ajuste de Talla

Vista Previa de la Prenda

**6. Arquitectura General**

El sistema sigue una arquitectura modular, orientada a objetos:

Clases de Datos: Nodo, Prenda, y Usuario definen las entidades básicas.

Estructuras de Datos: lista enlazadas prendas y Lista Enlazada Usuarios utilizan listas enlazadas para manejar prendas y usuarios dinámicamente.

Arboles binarios y recursión: Implementa un árbol binario de búsqueda (ABB) en la clase Usuario además, Este método es recursivo ya que se encarga de encontrar la posición correcta para un nuevo valor en el árbol binario de búsqueda.

Controladores: controlar usuario y controlador de prendas encapsulan la lógica.

Interfaz de Usuario: Se utiliza una un interfaz gráfica para acceder a las funcionalidades principales como diseñar, ajustar talla, ver vista previa, y compartir el diseño.

**7.Problema y Enfoque de Solución**

Problema: La aplicación requiere gestionar un sistema de registro, inicio de sesión de usuarios y la creación de diseños personalizados de prendas de ropa, Los usuarios deben poder registrarse, iniciar sesión, y, tras autenticarse, diseñar prendas con opciones específicas (tipo, color, tejido, detalles).

Solución Propuesta: Se desarrolla un sistema con clases que representan tanto los usuarios como las prendas. La gestión de usuarios se implementa mediante una lista enlazada, que permite registrar e iniciar sesión de manera eficiente. Una vez autenticado, el usuario puede diseñar prendas, utilizando los atributos específicos de cada prenda y almacenándolos en una lista de diseño de prendas asociada al usuario.

**8. Descripción de Estructuras de Datos**

**Lista Enlazada Usuarios**: La elección de una lista enlazada es adecuada para almacenar usuarios porque el número de usuarios no está predeterminado y se espera que las operaciones principales sean adiciones y búsquedas secuenciales. Cada usuario se representa como un nodo de la lista.

* **Nodo<Usuario>**: Un nodo en la lista enlazada contiene un objeto Usuario y un puntero al siguiente nodo.
* **Agregar Usuario Rec**: Método recursivo que agrega un usuario al final de la lista enlazada, asegurando que la lista crezca sin problemas.
* **Buscar Usuario Rec**: Método recursivo para buscar un usuario en la lista por correo electrónico. Esto permite validar si el usuario ya existe antes de registrar uno nuevo.

**Usuario**: Cada usuario está representado con un correo y una contraseña. La clase controlar usuario se encarga de las operaciones de registro y de inicio de sesión, verificando si el usuario existe y si las credenciales son correctas.

**Controlador deprendas** y **Prenda**: Estas clases se encargan del diseño de prendas. La clase Prenda representa una prenda específica con sus atributos (tipo, color, tejido, detalles).

* **Controlador de prendas** almacena y gestiona las prendas diseñadas por cada usuario autenticado, lo que permite asociar las prendas con el usuario correcto.

**9.DIAGRAMA DE CLASE**

[**https://lucid.app/lucidchart/129f6866-bec0-41fd-91db-eac01033880e/edit?viewport\_loc=-260%2C1384%2C4440%2C1980%2CHWEp-vi-RSFO&invitationId=inv\_2ce14d63-f492-4066-ba2b-bcab46327a00**](https://lucid.app/lucidchart/129f6866-bec0-41fd-91db-eac01033880e/edit?viewport_loc=-260%2C1384%2C4440%2C1980%2CHWEp-vi-RSFO&invitationId=inv_2ce14d63-f492-4066-ba2b-bcab46327a00)

**10.** **ORDEN DE MAGNITUD BIG-O**

**Clases con sus métodos y notaciones Big-O:**

**1. Clase listaenlzprendas**

**Constructor (listaenlzprendas):**

**Inicializa la lista con cabeza = null.**

**Complejidad: O(1).**

**AgregarPrenda(Prenda):**

**Llama recursivamente a AgregarPrendaRec, recorriendo la lista hasta el final para agregar la nueva prenda.**

**Complejidad: O(n).**

**AgregarPrendaRec(Nodo<Prenda>, Prenda):**

**Este método recorre recursivamente la lista enlazada.**

**Complejidad: O(n).**

**ObtenerPrendas():**

**Llama a ObtenerPrendasRec para recorrer la lista y agregar cada nodo a una lista de objetos Prenda.**

**Complejidad: O(n).**

**ObtenerPrendasRec(Nodo<Prenda>, List<Prenda>):**

**Similar al anterior, recorre todos los nodos para obtener sus valores.**

**Complejidad: O(n).**

**---**

**2. Clase ListaEnlazadaUsuarios**

**Constructor (ListaEnlazadaUsuarios):**

**Inicializa con cabeza = null.**

**Complejidad: O(1).**

**AgregarUsuario(Usuario):**

**Llama recursivamente a AgregarUsuarioRec, recorriendo la lista para agregar un usuario al final.**

**Complejidad: O(m).**

**AgregarUsuarioRec(Nodo<Usuario>, Usuario):**

**Similar al anterior, recorre recursivamente la lista enlazada.**

**Complejidad: O(m).**

**BuscarUsuario(string correo):**

**Llama a BuscarUsuarioRec para buscar el correo en toda la lista.**

**Complejidad: O(m).**

**BuscarUsuarioRec(Nodo<Usuario>, string correo):**

**Recorre los nodos recursivamente para comparar el correo buscado.**

**Complejidad: O(m).**

**3. Clase controlarusuario**

**Constructor (controlarusuario):**

**Inicializa una instancia de ListaEnlazadaUsuarios.**

**Complejidad: O(1).**

**RegistrarUsuario(string correo, string contraseña):**

**Usa BuscarUsuario para verificar si el correo existe, y si no, lo agrega.**

**Complejidad: O(m).**

**IniciarSesion(string correo, string contraseña):**

**Llama a BuscarUsuario y compara la contraseña, por lo que depende de la búsqueda.**

**Complejidad: O(m).**

**4. Clase Nodo<T>**

**Constructor (Nodo<T>):**

**Crea un nodo con el valor dado y establece siguiente = null.**

**Complejidad: O(1).**

**5. Clase Prenda**

**Constructor (Prenda):**

**Inicializa las propiedades de la prenda.**

**Complejidad: O(1).**

**6. Clase Usuario**

**Constructor (Usuario):**

**Inicializa las propiedades del usuario y una lista enlazada de prendas diseñadas.**

**Complejidad: O(1).**

**7. Clase Form1 (Eventos de la interfaz gráfica)**

**btnRegistrar\_Click:**

**Llama a RegistrarUsuario para registrar un nuevo usuario.**

**Complejidad: O(m).**

**btnIniciarSesion\_Click:**

**Llama a IniciarSesion para autenticar al usuario.**

**Complejidad: O(m).**

**btnDiseñarPrenda\_Click:**

**Si el usuario está autenticado, llama a métodos relacionados con las prendas (probablemente AgregarPrenda), por lo que dependerá de la lista enlazada de prendas.**

**Complejidad: O(n).**

| **OPERACION** | **COMPLEJIDAD** |
| --- | --- |
| **Inicializar listas/ nodos** | **O(1)** |
| **Agregar prenda** | **O(n)** |
| **Obtener todas las prendas** | **O(n)** |
| **Agregar usuario** | **O(m)** |
| **Buscar Usuario** | **O(m)** |
| **Registrar usuario** | **O(m)** |
| **Iniciar sesión** | **O(m)** |