UNIVERSIDAD CATÓLICA DE TEMUCO | DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
FACULTAD DE INGENIERÍA

ClaveForte

Avances realizados estas dos semanas

Curso: Desarrollo de Aplicaciones Empresariales
Integrantes: Ignacio De Celis y Oscar Barra
Fecha: 15/06/2025

1. Resumen del proyecto

Descripción funcional

ClaveForte es una aplicación web creada para que las personas puedan guardar y compartir sus credenciales de forma segura y ordenada. El sistema utiliza un método de autenticación basado en un correo electrónico único y una clave principal para el inicio de sesión. Para ver o compartir credenciales, se solicitará una clave secundaria, lo que añade una capa extra de seguridad. Además, todas las contraseñas almacenadas en la plataforma serán encriptadas, asegurando que la información sensible permanezca protegida frente a accesos no autorizados.

Rol y alcance en un entorno empresarial y educativo

En empresas y centros educativos es común que los usuarios necesiten acceder a varios servicios digitales como videollamadas o plataformas de gestión educativa . En estos contextos, ClaveForte ayuda a mantener las credenciales organizadas y disponibles en un solo lugar, reduciendo la necesidad de escribirlas en libretas o buscarlas constantemente. Esto no solo ahorrará tiempo, sino que también evita el cambio frecuente de las contraseñas y facilitará el acceso controlado a cuentas compartidas entre compañeros o grupos de trabajo.

Ventajas adicionales y seguridad

ClaveForte permite compartir credenciales de forma segura, sin comprometer la privacidad de la información. La clave secundaria garantiza que solo el dueño de la cuenta pueda autorizar la visualización o el envío de accesos a otros usuarios. Gracias al uso de contraseñas encriptadas y un sistema de autenticación en dos niveles, la plataforma promueve una gestión responsable de los datos de acceso, adaptándose a las necesidades reales de entornos colaborativos.

2. Meta de avance propuesta al inicio del periodo

Durante estas semanas de desarrollo, se priorizó la implementación del Prototipo Mínimo Navegable (PMN) y del Producto Mínimo Viable (PMV) del proyecto. En cuanto al sistema de autenticación, se conectaron los formularios de registro e inicio de sesión con la base de datos, permitiendo simular estos procesos de forma realista. Además, se diseñó el módulo de gestión de credenciales, donde el usuario puede revisar, agregar y compartir sus credenciales. Y cabe recalcar que todas estas acciones quedan registradas en la base de datos.

3. Descripción técnica del avance logrado

3.1. Visión general de la arquitectura

El sistema desarrollado sigue una arquitectura **cliente-servido**. El back-end fue implementado en Flask (Python), encargado de la lógica del sistema, manejo de rutas, validación de datos y conexión con la base de datos SQLite3. El front-end está construido con HTML, Tailwind CSS y JavaScript, lo que permite una interfaz simple, moderna y responsiva. La aplicación se despliega en la plataforma **Render**, utilizando **GitHub** como sistema de control de versiones y fuente de integración continua. Para mantener el servicio en línea se implementó **UptimerBot**.

3.2. Módulos desarrollados y su función

a) Autenticación de usuarios:

- **Inicio de sesión:** El usuario ingresa un correo único y una contraseña principal. Los datos son validados contra la base de datos. En caso de éxito, se genera un token de autenticación mediante **Flask.session,** lo que permite controlar el acceso a las demás funcionalidades del sistema.

- **Registro de usuarios:** Permite crear nuevas cuentas de usuario. El formulario valida los campos requeridos y, si son correctos, almacena la información en la base de datos. Luego, el usuario es redirigido automáticamente a la interfaz de inicio de sesión.

b) Página de inicio:

- Ofrece una interfaz de bienvenida al usuario autenticado. Desde aquí, se puede acceder a las secciones principales del sistema mediante mensajes de sugerencia que orientan la navegación: revisar credenciales o compartirlas con otros usuarios.

c) Gestión de credenciales:

- **Visualización y registro:** Los usuarios pueden revisar todas las credenciales que les pertenecen o a las que tienen acceso compartido. Estas se obtienen directamente desde la base de datos. También pueden agregar nuevas credenciales a su cuenta mediante un formulario validado en el servidor.

- **Compartir credenciales:** Permite compartir de forma controlada las credenciales propias. Para ello, el usuario debe ingresar el correo del destinatario, seleccionar la credencial y validar la acción ingresando su contraseña secundaria. Esta funcionalidad

fue diseñada pensando en entornos donde múltiples usuarios deben acceder a cuentas comunes (como en grupos de trabajo o clases universitarias).

3.3. Clasificación por tipo de desarrollo

- **Back-end (Flask / Python):**
    - Manejo de rutas y vistas.
    - Conexión y operaciones con SQLite3.
    - Validación de credenciales.
    - Creación de sesiones (tokens con Flask.session).
    - Carga de variables de entorno (dotenv).
    - Redirecciones (Flask.redirect).

- **Front-end (HTML / Tailwind CSS / JS):**
    - Interfaces para registro, inicio de sesión, credenciales y home.
    - Validaciones básicas en formularios.
    - Control de visibilidad para las credenciales (mostrar/ocultar).

3.4. Tecnologías utilizadas

- **Flask (Python):** Framework principal utilizado para el desarrollo del servidor, definición de rutas, gestión de sesiones, validación de datos y renderizado de vistas.
- **SQLite3:** Base de datos relacional local, utilizada por su simplicidad y fácil integración con proyectos pequeños a medianos.
- **Flask.session:** Módulo de Flask utilizado para generar y mantener tokens de autenticación mediante cookies.
- **Flask.render_template:** Función utilizada para cargar y renderizar páginas HTML dinámicamente desde el servidor.
- **Flask.redirect:** Utilizada para redirigir al usuario entre páginas según el estado de la sesión o los resultados de validaciones.
- **dotenv:** Permite cargar variables de entorno desde un archivo .env, protegiendo datos sensibles como claves o configuraciones.
- **HTML:** Lenguaje de marcado base para la estructura de las páginas web.
- **Tailwind CSS:** Framework de diseño basado en utilidades, utilizado para aplicar estilos modernos y consistentes de forma eficiente.
- **JavaScript:** Usado para mejorar la interacción del usuario con los formularios, validar datos en el cliente y controlar la visualización de credenciales (mostrar/ocultar).
- **Render:** Plataforma utilizada para el despliegue automático del proyecto web desde GitHub.
- **GitHub:** Repositorio remoto que permite el control de versiones y es el origen que Render utiliza para generar el entorno de producción.

- **UptimerBot:** Herramienta de monitoreo que envía solicitudes periódicas al servidor desplegado en Render, evitando que entre en modo de suspensión.

3.5. Estructura general del sistema

La organización del sistema sigue una estructura modular y clara:

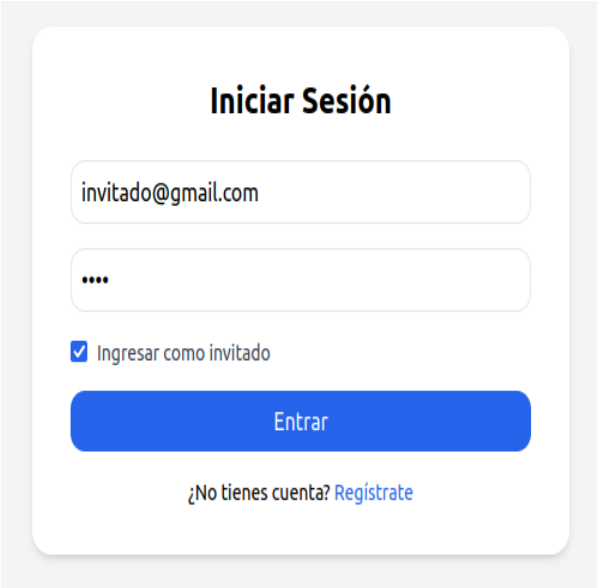
- **Rutas principales (Flask):**
  - **/:** redirige al login o home según sesión activa.
  - **/login:** formulario de inicio de sesión.
  - **/register:** formulario de registro de usuario.
  - **/home:** panel de inicio para usuarios autenticados.
  - **/crendentials:** muestra credenciales del usuario.
  - **/share_credential:** formulario para compartir credenciales.

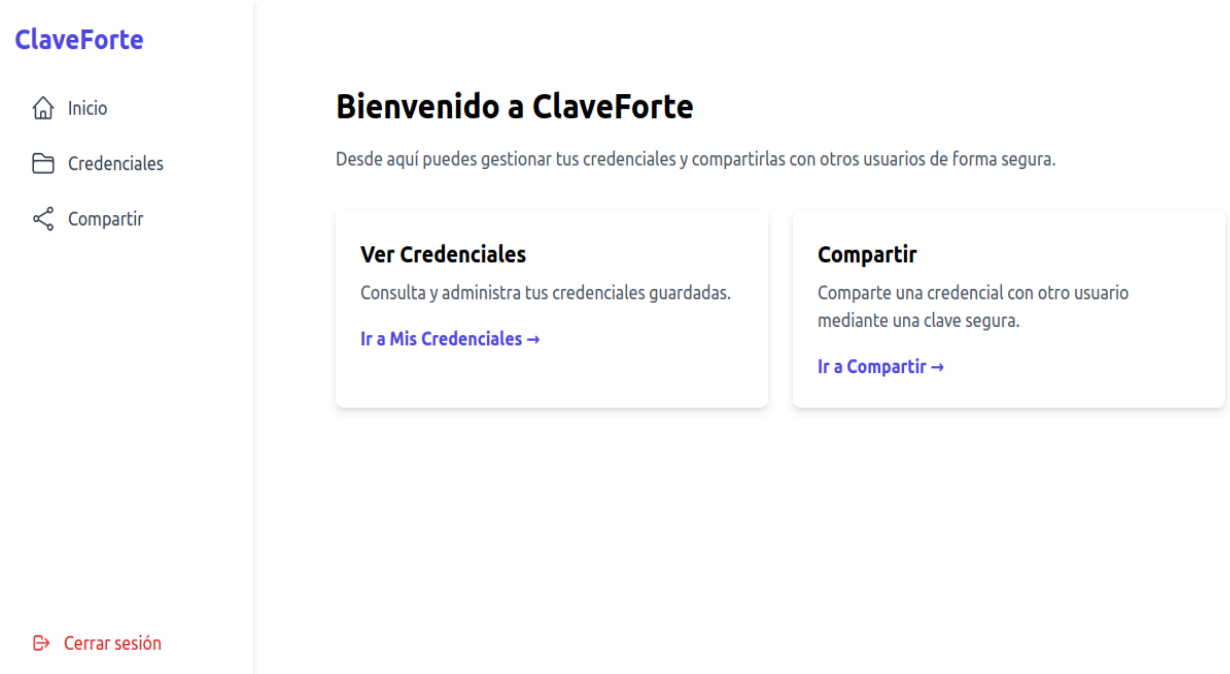
- **Archivos y carpetas relevantes:**
  - **/templates/:** contiene las páginas HTML.
  - **app.py:** lógica principal del back-end (rutas, conexión, validaciones).
  - **/instance/ClaveForte.db:** base de datos SQLite con tablas para usuarios, credenciales y permisos.

3.6. Consideraciones actuales

En esta etapa inicial, las contraseñas de los usuarios y credenciales se están almacenando como texto plano. Esta decisión se tomó para facilitar las pruebas y priorizar las funcionalidades básicas. Sin embargo, se tiene contemplado aplicar **encriptación robusta** mediante la librería cryptography en la siguiente etapa de desarrollo, como parte de las buenas prácticas de seguridad.

4. Capturas de pantalla y/o evidencia funcional





ClaveForte

Inicio

Credenciales

Compartir

Cerrar sesión

Mis Credenciales

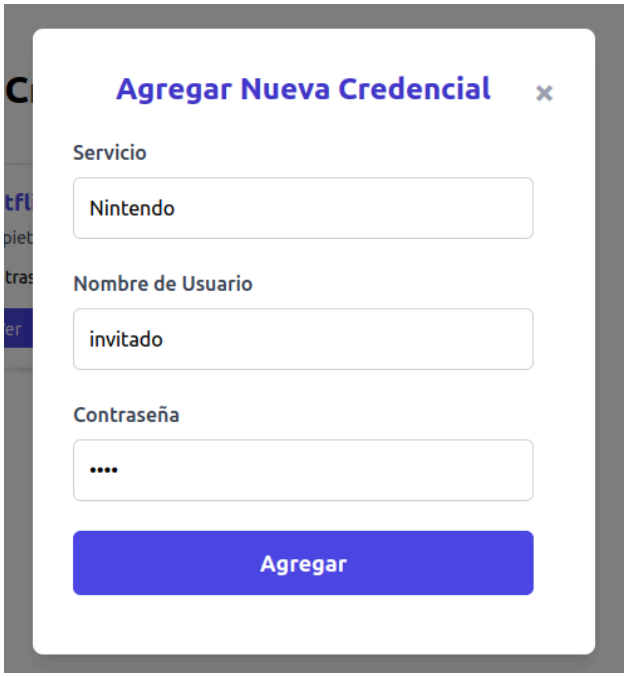
Agregar nueva credencial

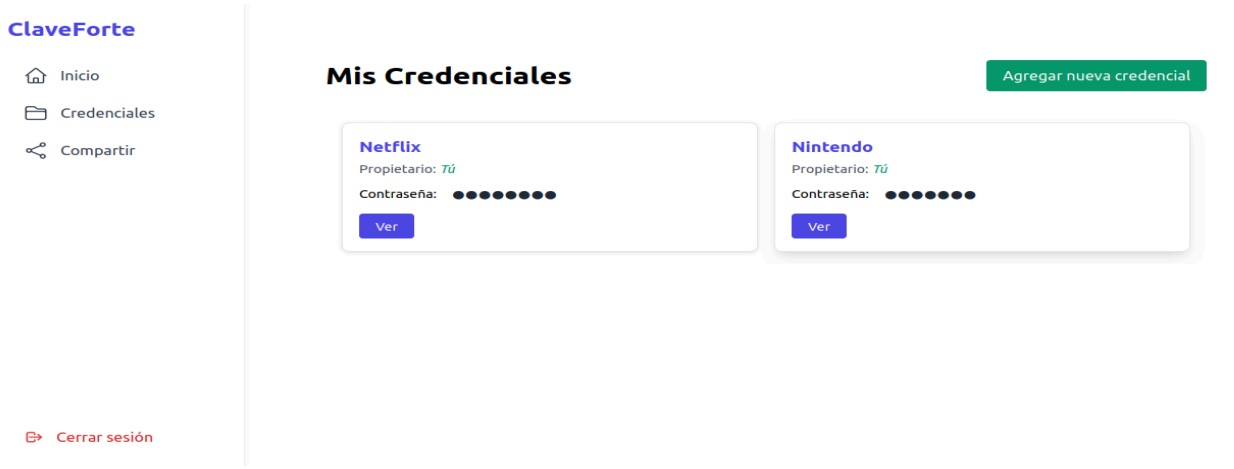
Netflix

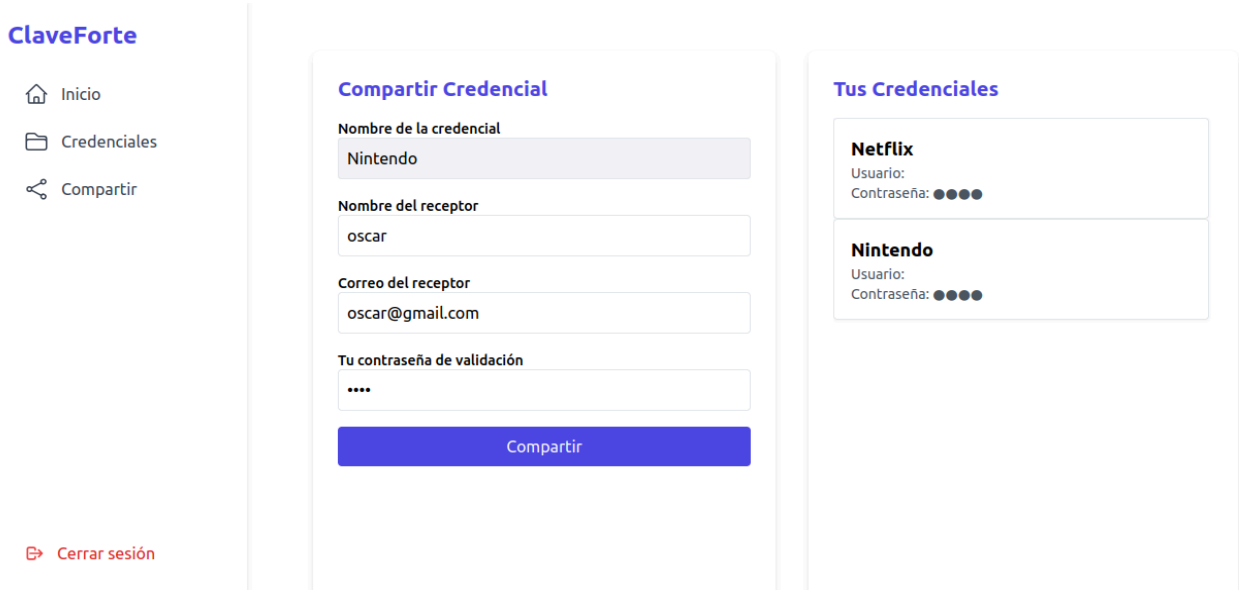
Propietario: *Tú*

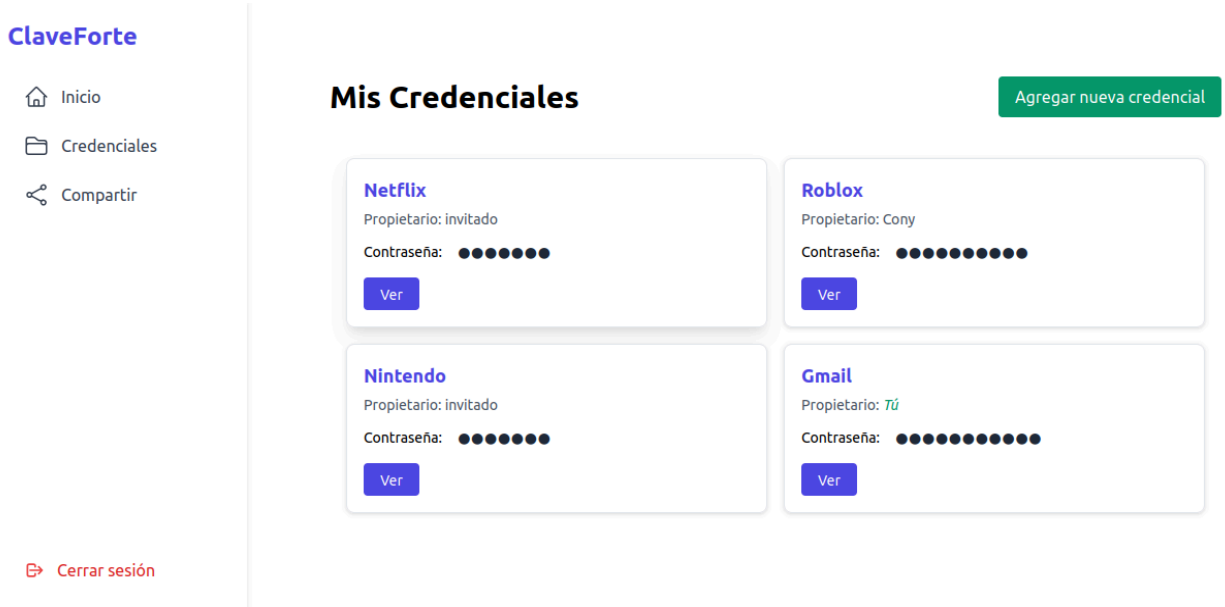
Contraseña: ●●●●●●●●●●●

Ver




ClaveForte
Inicio
Credenciales
Compartir
Cerrar sesión
Mis Credenciales
Agregar nueva credencial
Netflix
Propietario: Tú
Contraseña: ●●●●●●●●
Ver
Nintendo
Propietario: Tú
Contraseña: ●●●●●●●
Ver


ClaveForte
Inicio
Credenciales
Compartir
Cerrar sesión
Compartir Credencial
Nombre de la credencial
Nintendo
Nombre del receptor
oscar
Correo del receptor
oscar@gmail.com
Tu contraseña de validación
••••
Compartir
Tus Credenciales
Netflix
Usuario:
Contraseña: ●●●●
Nintendo
Usuario:
Contraseña: ●●●●


ClaveForte
Inicio
Credenciales
Compartir
Cerrar sesión
Mis Credenciales
Agregar nueva credencial
Netflix
Propietario: invitado
Contraseña: ●●●●●●●
Ver
Roblox
Propietario: Cony
Contraseña: ●●●●●●●●●●
Ver
Nintendo
Propietario: invitado
Contraseña: ●●●●●●●
Ver
Gmail
Propietario: Tú
Contraseña: ●●●●●●●●●●●
Ver

5. Código fuente comentado

5.1 Ruta: /login

- **GET:**
  Muestra la página del formulario de login (login.html).
- **POST:**
  Cuando el usuario envía el formulario con su correo y contraseña:
  1. **Obtiene** los datos enviados (email y password) desde el formulario.
  2. **Abre conexión** con la base de datos SQLite ClaveForte.db.
  3. **Consulta** en la tabla Users para buscar un usuario con el correo ingresado.
  4. **Si encuentra el usuario:**
     - Compara la contraseña enviada con la almacenada.
     - Si coinciden, **crea una sesión** guardando el id y nombre del usuario, usando cookies para mantener la sesión activa.
     - Luego, redirige a la página principal (home).
     - Si la contraseña no coincide, muestra un mensaje de error "Contraseña incorrecta".
  5. **Si no encuentra el usuario:**
     - Muestra un mensaje de error "Correo no registrado".
  6. **Finalmente**, si es GET o hubo error, renderiza el formulario de login para que el usuario pueda intentarlo.

```
@app.route('/login', methods=['GET','POST'])
def login():
  if request.method == 'POST':
    correo    = request.form['email']
    contraseña = request.form['password']

    conexion = sqlite3.connect('instance/ClaveForte.db')
    cursor = conexion.cursor()
    cursor.execute("SELECT id_usr, usr_name, usr_pass FROM Users WHERE    usr_mail = ?", (correo,))
    usuario = cursor.fetchone()
    conexion.close()

    if usuario:
      id_usr, usr_name, usr_pass = usuario
      if usr_pass == contraseña:
        session['usuario_id'] = id_usr
        session['usuario_nombre'] = usr_name
        return redirect(url_for('home'))
      else:
        flash("Contraseña incorrecta")
    else:
      flash("Correo no registrado")
  return render_template('/auth/login.html')
```

5.2 Ruta: /signup

- **GET:**
  Muestra el formulario de registro (signup.html) para que el usuario pueda registrarse.
- **POST:**
  Cuando el usuario envía el formulario:
  1. **Obtiene** los datos ingresados: nombre de usuario (username), correo (email), contraseña (password), y rol (rol).
  2. Usa la misma contraseña también como secret_pass (este es temporalmente mientras se probaban las funcionalidades).
  3. Registra la **fecha y hora actual** como last_login (aunque el usuario aún no ha iniciado sesión).
  4. **Conecta** con la base de datos ClaveForte.db y prepara una consulta para **insertar** los datos en la tabla Users.
  5. **Guarda** los datos con commit() y **cierra** la conexión.
  6. Redirige automáticamente al usuario a la ruta de login (/login) después de registrarse.

```
@app.route('/signup', methods=['GET','POST'])
def signup():
  if request.method == 'POST':
    username = request.form['username']
    email    = request.form['email']
    password = request.form['password']
    secret   = request.form['password']
    created  = str(datetime.now())
    rol      = request.form['rol']

    conexion = sqlite3.connect('instance/ClaveForte.db')
    cursor = conexion.cursor()

    cursor.execute("""
      INSERT INTO Users (usr_name, usr_mail, usr_pass, secret_pass, last_login, id_rol)
      VALUES (?, ?, ?, ?, ?, ?)
    """, (username, email, password, secret, created , rol))

    conexion.commit()
    conexion.close()

    return redirect(url_for('login'))
  return render_template('/auth/signup.html')
```

5.3 Ruta: /add_credential

1. Se recupera el ID del usuario desde la sesión:
2. Se recuperan los datos del formulario:
3. Se crea una lista vacía para posibles usuarios con acceso compartido a la credencial:
4. Se obtiene el nombre del usuario desde la sesión:
5. Se abre la conexión con la base de datos y se crea un cursor:
6. Se inserta la nueva credencial en la tabla Credentials:
7. Se guardan los cambios y se cierra la conexión:
8. Se redirige al usuario a la ruta /credentials:

```
@app.route('/add_credential', methods=['GET','POST'])
def agregar_credencial():
    id_usr = session.get('usuario_id')
    service_name  = request.form['servicio']
    service_pass  = request.form['contrasena']
    users_allows  = json.dumps([])
    name_owner    = session.get('usuario_nombre')

    conn = sqlite3.connect('instance/ClaveForte.db')
    cur = conn.cursor()
    cur.execute("""
      INSERT INTO Credentials (service_name, service_pass, users_allows, name_owner, id_usr)
      VALUES (?, ?, ?, ?, ?)
    """, (service_name, service_pass, users_allows,name_owner, id_usr))
    conn.commit()
    conn.close()
    return redirect('/credentials')
```

6. Dificultades encontradas

Oscar: Uno de los principales desafíos fue implementar una estructura basada en componentes dentro de Flask, considerando que esta funcionalidad no es nativa del framework. Esto implicó aprender y aplicar el mecanismo de herencia de plantillas para lograr una organización modular. Si bien la búsqueda inicial requirió tiempo, la implementación en sí resultó sencilla una vez comprendido el enfoque, por lo que no representó una gran complejidad técnica.

Ignacio : Durante la construcción del módulo de credenciales, el principal desafío fue asegurar que cada elemento estuviera correctamente conectado y ubicado dentro de su sección correspondiente. Esto exigió una revisión constante del flujo de datos y la estructura visual. La solución fue reorganizar el contenido y validar continuamente el funcionamiento del sistema para mantener la coherencia y evitar errores en etapas posteriores del desarrollo.

7. Autoevaluación del cumplimiento

Oscar: Tomando en consideración los objetivos planteados y el tiempo destinado para su desarrollo, considero que tuvimos un buen desempeño, ya que se cumplieron todos los puntos acordados. Si bien el trabajo se concentró principalmente durante el fin de semana, es importante reconocer que, como estudiantes en la etapa final del semestre, enfrentamos constantemente evaluaciones y entregas durante la semana, lo cual condiciona nuestra distribución del tiempo. A esto se sumaron algunas dificultades externas, como problemas climáticos. A pesar de ello, logramos presentar un avance funcional dentro del plazo establecido, demostrando compromiso y responsabilidad. En función de estos aspectos, considero que nuestro trabajo merece una calificación de **6.5**.

Ignacio: Considerando el tiempo disponible y el trabajo realizado, creo que se logró un resultado acorde a lo esperado. Además, se avanzó significativamente durante estas primeras semanas, lo cual nos permitirá enfocar las etapas futuras en la corrección de errores y en la mejora de ciertos aspectos, en lugar de solo completar funcionalidades pendientes. En ese sentido, considero que se hizo un uso adecuado de las herramientas disponibles y que el ritmo de trabajo fue oportuno para el momento del semestre en el que nos encontramos. Por lo tanto, estimo que una calificación de **6.0** refleja de forma justa nuestro desempeño, reconociendo lo logrado, pero también entendiendo que siempre es posible mejorar.

8. Plan de acción para las próximas dos semanas

Acciones específicas para avanzar hacia la finalización del proyecto:

- Encriptar las contraseñas de usuarios.
- Agregar protección a las rutas.
- Eliminar archivos sensibles de GitHub (.env).
- Implementar una diferenciación entre roles (Usuarios, Administradores).
- Trabajar en las interfaces finales.
- Agregar el módulo de perfil de usuario.
- Implementar formulario para que el usuario pueda crear su contraseña secundaria.
- Terminar de implementar la lógica de funcionamiento prometida.

Asignación de tareas si trabajan en grupo

Oscar:

- Encriptar las contraseñas de usuarios.
- Agregar protección a las rutas.
- Trabajar en las interfaces finales.
- Implementar formulario para que el usuario pueda crear su contraseña secundaria.
- Terminar de implementar la lógica de funcionamiento prometida.

Ignacio:

- Eliminar archivos sensibles de GitHub (.env).
- Implementar una diferenciación entre roles (Usuarios, Administradores).
- Trabajar en las interfaces finales.
- Agregar el módulo de perfil de usuario.
- Terminar de implementar la lógica de funcionamiento prometida.

9. Enlaces de interés

- GitHub: https://github.com/oscarbarra/DAE_Project2.git

- Render: https://claveforte-uct-2025.onrender.com