

**TECNOLÓGICO NACIONAL DE MÉXICO**



**INSTITUTO TECNOLÓGICO DE CD. VALLES**

**INGENIERÍA EN SISTEMAS COMPUTACIONALES**

## **Desarrollo de aplicaciones móviles multiplataforma**

### **Proyecto Final**

**Integrantes:**                      **Grupo: E**

Edgardo Jahir Ortiz Ortega

Hugo Eduardo Mirón Alvizo

Alfredo Román Soto Aguilar

Víctor Hugo Stevens Sánchez

**Docente:**

Alfredo Barrón Rodríguez

**Fecha:**30 de mayo de 2025



## Índice

Introducción.....	4
Objetivos .....	5
Desarrollo .....	6
Funcionalidades.....	6
Coach.....	6
Administrador .....	6
Cliente.....	6
Pantallas.....	8
Coach.....	9
Administrador .....	12
Cliente .....	15
Código Relevante .....	18
Frontend.....	18
Backend .....	22
Conclusión .....	27

## Introducción

En el desarrollo de este proyecto, se desea crear una aplicación móvil multiplataforma funcional, que cumpla diferentes funciones que se desarrollan con el conocimiento adquirido a lo largo de la materia, entre dichas características de la aplicación se encuentra la seguridad de la aplicación, la selección de horarios, el registro y modificación de información, diferentes tipos de acceso dependiendo al usuario, etc. Se llevan a cabo dos partes de la aplicación, frontend haciendo uso de Flutter y Python para el backend, aplicando conocimiento de otras materias para el desarrollo de este.

Flutter, desarrollado por Google, es un framework de código abierto que permite crear interfaces de usuario nativas, atractivas y altamente responsivas a partir de una sola base de código. Por otro lado, el backend se ha construido utilizando Python, un lenguaje versátil y ampliamente adoptado en el desarrollo de aplicaciones web. Python, en conjunto con frameworks como Django o Flask, proporciona una estructura clara y eficiente para gestionar la lógica y el acceso a las bases de datos.

## Objetivos

- Desarrollar una aplicación móvil multiplataforma que brinde soporte a los usuarios en el manejo de su agenda de clases y horario.
- Mostrar las clases disponibles y los paquetes de horarios, dar la posibilidad de registrarse en ellas.
- Visualizar el horario del usuario y dar la capacidad de ver la disponibilidad.

# **Desarrollo**

## **Funcionalidades**

### **Coach**

- Puede iniciar/cerrar sesión.
- Acceso a clases desde el menú.
- Visualizar y editar:
  - Datos de cuenta (usuario y foto).
  - Datos personales (nombre, apellidos, teléfono, correo).
  - Datos domiciliarios (calle, número, asentamiento, código postal).
- Ver sus datos y horario offline.
- Dar de alta clases offline con sincronización posterior.

### **Administrador**

- Puede iniciar/cerrar sesión.
- Agregar, consultar, modificar y eliminar Usuarios tipo coach y administrador.
- Gestión de paquetes.
- Gestionar disponibilidad de horario.
- Asignar coach, casilla y tipo de curso a las clases.
- Registrar, consultar, modificar y eliminar cursos.
- Hacer inserciones, modificaciones, eliminaciones y consultas offline.

### **Cliente**

- Puede registrarse desde la interfaz del login.
- Puede iniciar/cerrar sesión.
- Visualizar horarios disponibles.
- Consultar información del local ("Acerca de").
- Visualizar y editar:
  - Datos de cuenta, personales y domiciliarios.
  - Métricas: estatura, peso, cardio, pulso.
- Contratar planes y seleccionar horarios (suscripción).

- Visualizar sus datos y horarios offline.

## Pantallas

### Login – Registro

- **Registro:** Registro de Usuario “Cliente” en la base de datos, solicita los datos: Nombre, apellido paterno, apellido materno, Email, teléfono, tipo de documento y documento.
- **Inicio de sesión (login):** Solicita el usuario y contraseña registrados.

The image displays two side-by-side mobile app screens. The left screen is titled 'Login' with the subtitle 'Ingresa para continuar'. It features two input fields labeled 'Usuario' and 'Contraseña', followed by a red 'Ingresar' button. Below the button, it asks '¿Tiene una cuenta?' and provides a 'Registrarse' link. The right screen is titled 'Registro' with a back arrow. It contains seven input fields labeled 'Nombre', 'Apellido paterno', 'Apellido materno', 'Email', 'Telefono', 'Tipo documento', and 'Documento'. Both screens have a dark background and red diagonal banners in the top right corner with the word 'DESAR'.

**Login**  
Ingresa para continuar

Usuario

Contraseña

Ingresar

¿Tiene una cuenta?  
Registrarse

**Registro**

Nombre

Apellido paterno

Apellido materno

Email

Telefono

Tipo documento

Documento

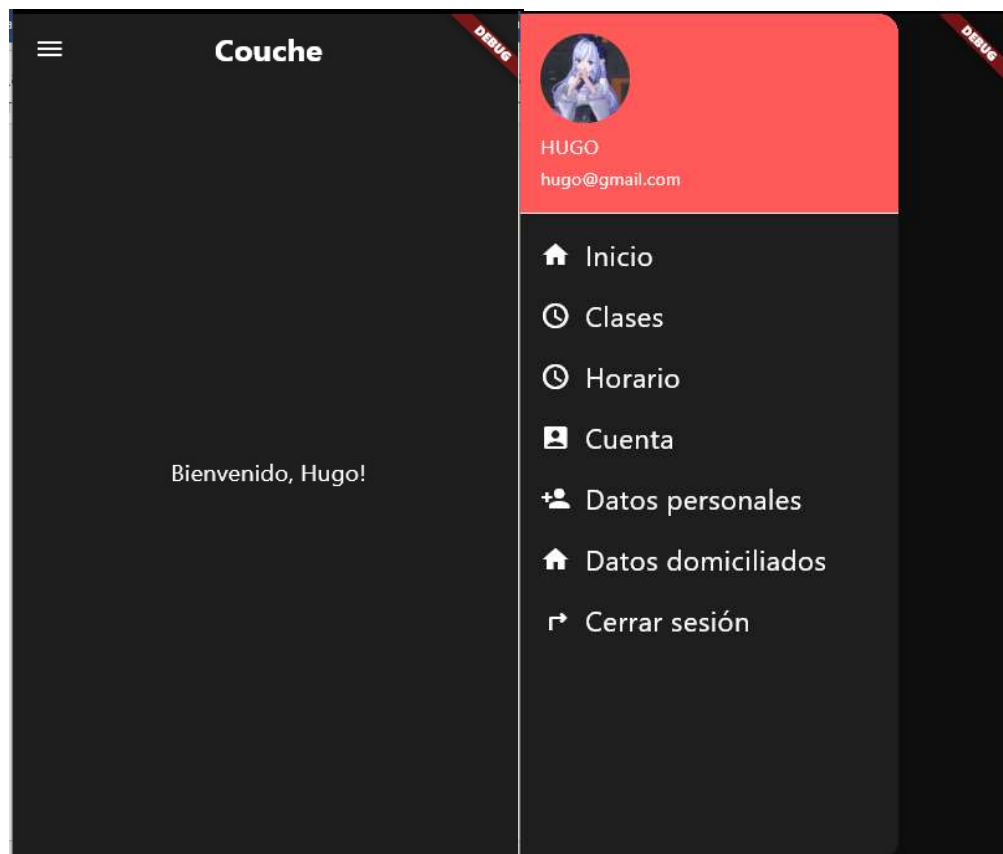
Next



## Coach

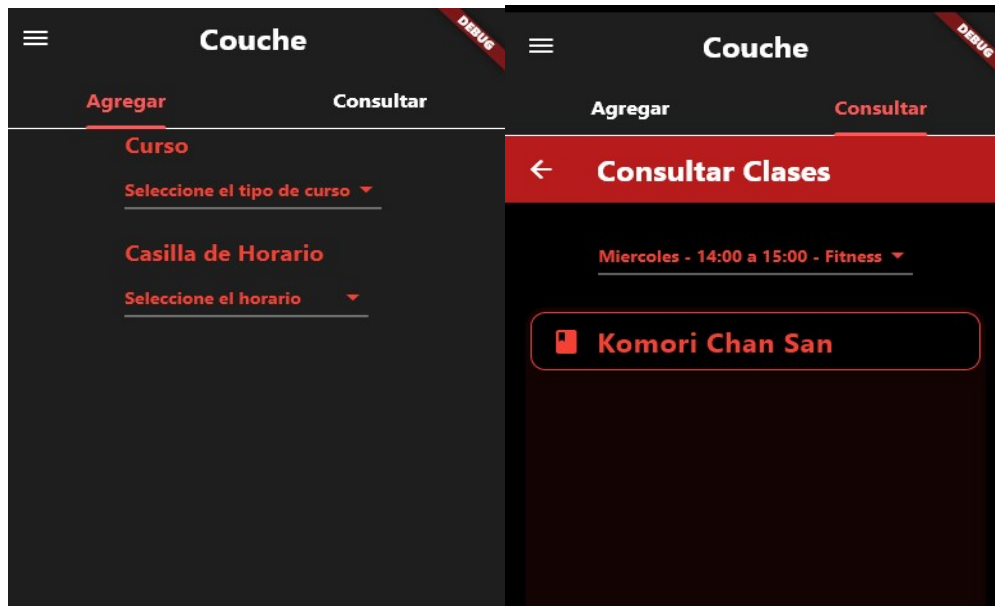
### Inicio – Menú

- **Inicio:** Pantalla principal al ingresar sesión.
- **Menú:** Surge de las barras horizontales de la parte superior izquierda de la pantalla. Muestra el nombre de usuario y correo, así como su foto de perfil. Tiene las opciones de ir a pantalla de inicio, clases, horario, cuenta, datos personales, datos domiciliados y cerrar sesión.



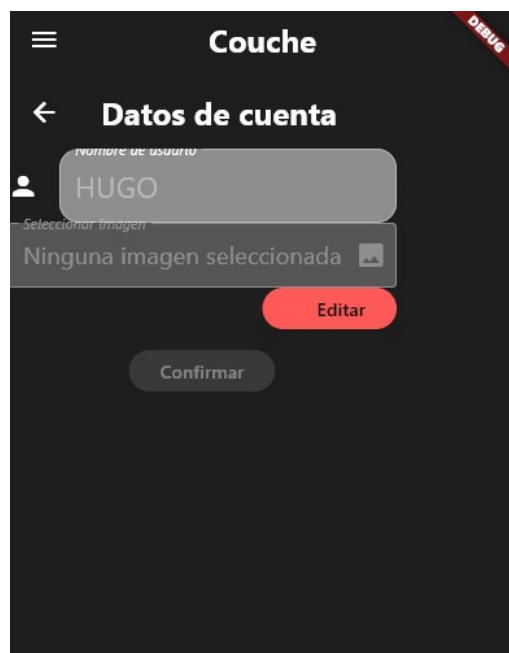
## Horario: Agregar – Consultar

- Como coach puedes agregar (por curso y horario) y consultar horarios (día, hora y tipo de curso).



## Datos de Cuenta

- Muestra los datos registrados de la cuenta: Nombre de usuario y foto de perfil (con opción de editarla).



## Datos Personales

- Apartado que muestra los datos personales registrados y que pueden ser modificados: Nombre, apellido paterno y materno, teléfono y correo electrónico.

The screenshot shows the 'Datos Personales' screen in the 'Couche' app. The header bar is dark with a white hamburger menu icon on the left, the title 'Couche' in the center, and a red 'Disponible' (Available) badge on the right. Below the header, there is a back arrow and the title 'Datos Personales' in white, followed by a lock icon. The form contains five input fields with light gray text labels and dark gray text values: 'Nombre' (Hugo), 'Apellido P.' (Hugos), 'Apellido M.' (Hugo), 'Teléfono' (4814814814), and 'Correo electrónico' (hugo@gmail.com). At the bottom right, there is a red 'Editar' (Edit) button, and at the bottom center, there is a dark gray 'Guardar' (Save) button.

## Datos Domiciliados

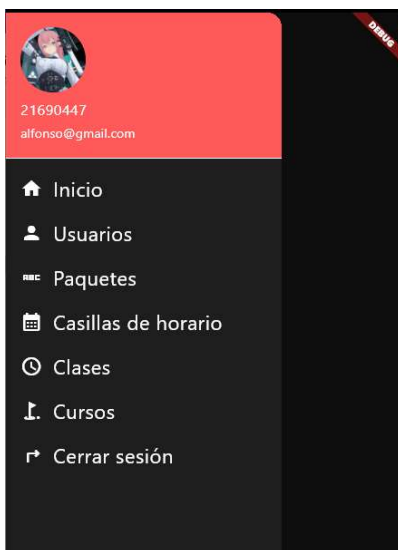
- Datos de vivienda registrados y que se pueden modificar: Calle, número de casa (interior y exterior), asentamiento y código postal.

The screenshot shows the 'Domicilio' screen in the 'Couche' app. The header bar is dark with a white hamburger menu icon on the left, the title 'Couche' in the center, and a red 'Disponible' (Available) badge on the right. Below the header, there is a back arrow and the title 'Domicilio' in white. The form contains five input fields with light gray text labels and dark gray text values: 'Calle' (Calle), 'Número ext.' (2), 'Número int.' (1), 'Asentamiento' (Asentamiento), and 'Código Postal' (11111). At the bottom right, there is a red 'Editar' (Edit) button, and at the bottom center, there is a dark gray 'Guardar' (Save) button.

## Administrador

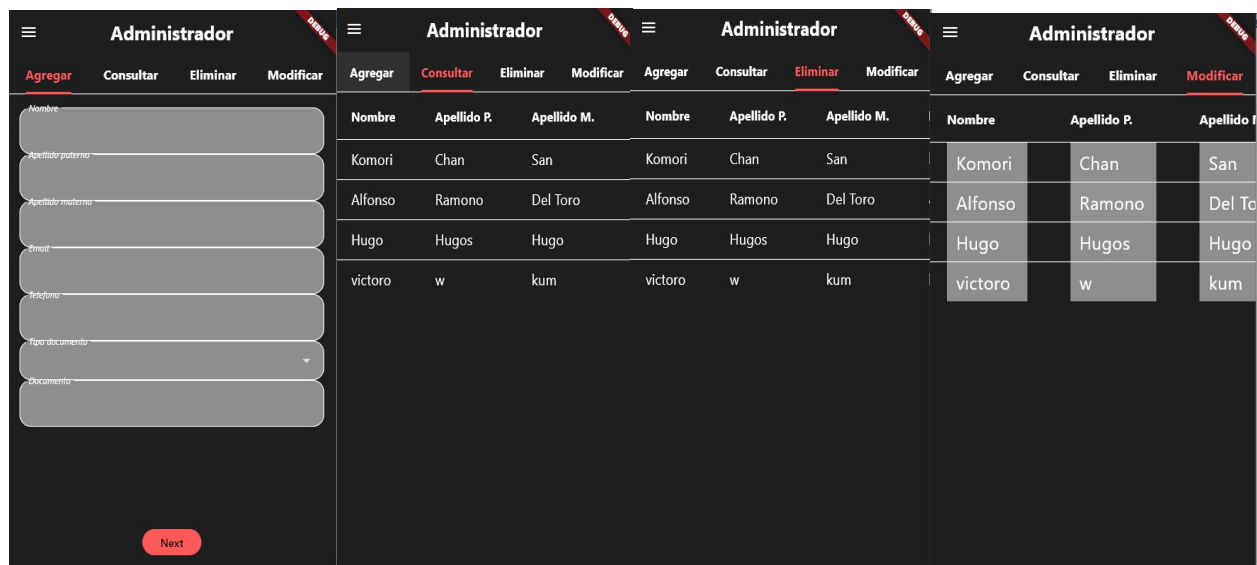
### Menú

- El Administrador cuenta con un menú lateral con las opciones para ver: Usuarios, paquetes, casillas de horario, clases, cursos y cerrar sesión.



### Usuarios

- En la opción “Usuarios” del menú, podremos agregar nuevos usuarios, consultar, eliminar y modificar datos de usuarios.



## Paquetes

- En el apartado “Paquetes” del menú, se pueden agregar paquetes de entrenamiento ingresando datos de precio, número de clases y si es flexible. Además, se pueden consultar las clases registradas, así como eliminar y modificar las ya existentes.

The image displays four sequential screenshots of a web application interface for managing training packages. Each screenshot shows a sidebar menu with a hamburger icon and the title 'Administrador'. The main content area has a top navigation bar with four tabs: 'Agregar', 'Consultar', 'Eliminar', and 'Modificar'. The 'Agregar' tab is active in the first screenshot, showing input fields for 'Precio' (499.99), 'Clases' (4), and 'Flexible' (Si), with a 'Submit' button at the bottom. The 'Consultar' tab is active in the second screenshot, showing a table with the same data. The 'Eliminar' tab is active in the third screenshot, showing the same data with a 'Selecciona una' button. The 'Modificar' tab is active in the fourth screenshot, showing the same data with a 'Selecciona una' button.

Precio	Clases	Flexible
499.99	4	Si

## Casillas de Horario

- Para decidir la disponibilidad, existe el apartado de casillas de horario, para agregar, consultar, eliminar y modificar el día de la semana, hora de inicio y fin en que se pueden llevar a cabo las clases.

The image displays four sequential screenshots of a web application interface for managing training slots. Each screenshot shows a sidebar menu with a hamburger icon and the title 'Administrador'. The main content area has a top navigation bar with four tabs: 'Agregar', 'Consultar', 'Eliminar', and 'Modificar'. The 'Agregar' tab is active in the first screenshot, showing input fields for 'Hora inicio' (14:00), 'Hora fin' (15:00), and 'Dia' (3), with a 'Submit' button at the bottom. The 'Consultar' tab is active in the second screenshot, showing a table with the same data. The 'Eliminar' tab is active in the third screenshot, showing the same data with a 'Selecciona una' button. The 'Modificar' tab is active in the fourth screenshot, showing the same data with a 'Selecciona una' button.

Hora inicio	Hora fin	Dia
14:00	15:00	3

## Clases

- En la opción “Clases”, podremos agregar, consultar, eliminar y modificar clases con los datos de coach, la casilla y el tipo de curso.

Administrador

Agregar Consultar Eliminar Modificar

Coach Casilla Descripción

Hugo Hugos 3 14:00 15:00 Fitness

Submit

Administrador

Agregar Consultar Eliminar Modificar

Coach Casilla Descripción

Hugo + Hugos 3 14:00 15:00 Fitness

Administrador

Agregar Consultar Eliminar Modificar

Coach Casilla Descripción

Hugo + Hugos 3 14:00 15:00 Fitness

Administrador

Agregar Consultar Eliminar Modificar

Coach Casilla Descripción

Hugo + Hugos 3 14:00 15:00 Fitness

## Curso

- En el apartado de curso, podemos registrar, consultar, modificar y eliminar cursos, estos cursos son registrados con datos de: nombre del curso y descripción del tipo de curso.

Administrador

Agregar Consultar Eliminar Modificar

Curso Descripción

Fitness Curso de fitness

Submit

Administrador

Agregar Consultar Eliminar Modificar

Curso Descripción

Fitness Curso de fitness

Administrador

Agregar Consultar Eliminar Modificar

Curso Descripción

Fitness Curso de fitness

Administrador

Agregar Consultar Eliminar Modificar

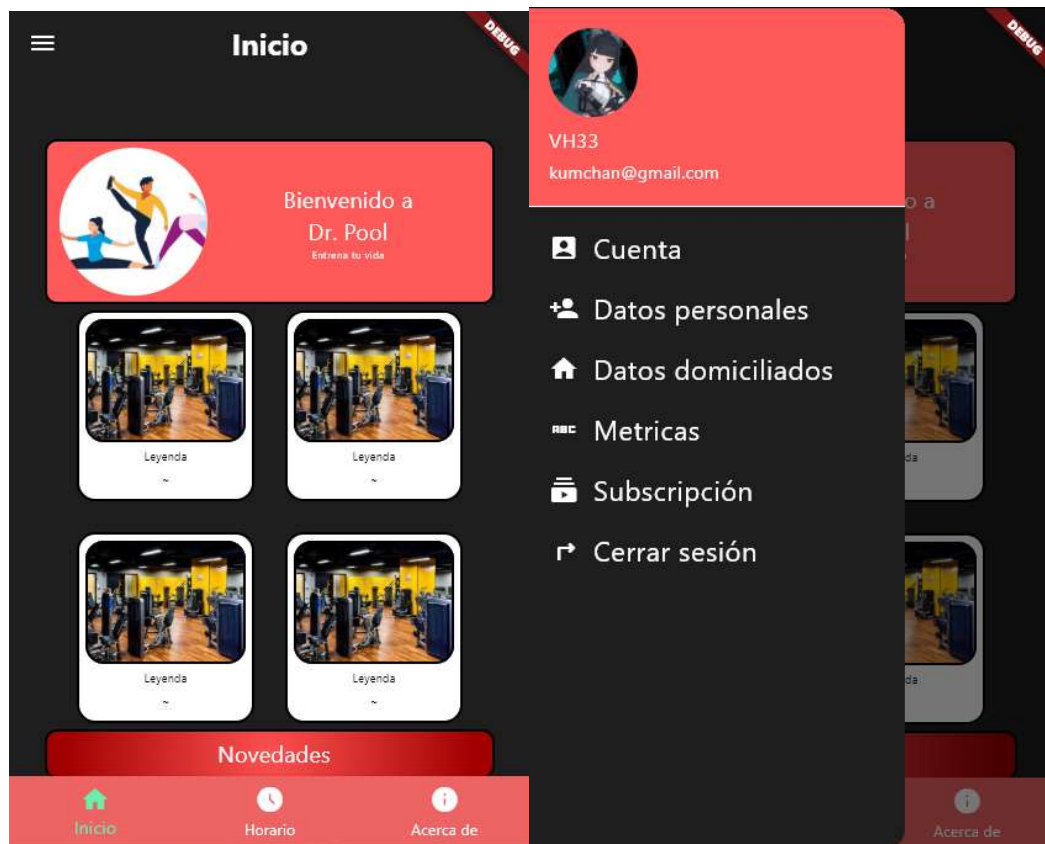
Curso Descripción

Fitness Curso de fitness

## Cliente

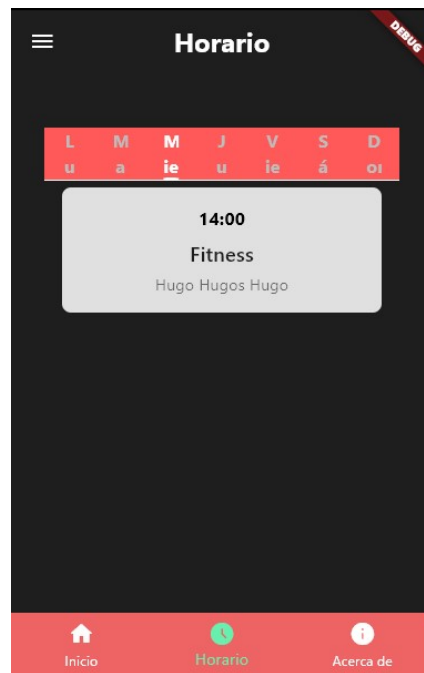
### Inicio – Menú

- **Inicio:** Es especial para ser mas llamativo para el cliente, además de que hay opciones adicionales en la parte inferior de la pantalla.
- **Menú:** El menú tiene las opciones para ver los datos de cuenta, personales y de domicilio, métricas y a que plan estas suscrito.



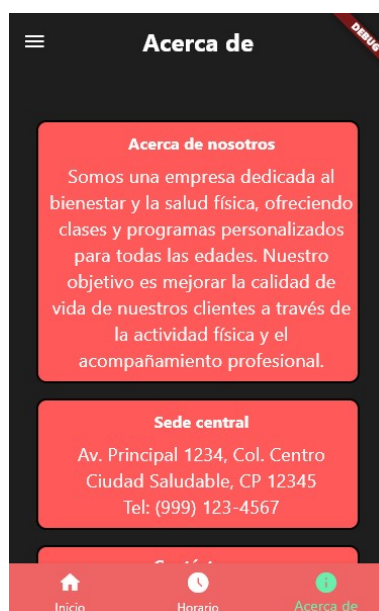
## Horario

Es una de las opciones del menú inferior y nos muestra los horarios existentes en la programación.



## Acerca de

De las opciones del menú inferior, esta muestra la información relevante del local y/o empresa.





## Métricas

- En esta opción del menú lateral se registran y modifican los datos: Estatura, peso, cardio, pulso para poder tener un mejor control en las sesiones.

← Métricas

Back Editar

Estatura (m)

Peso (kg)

Max Cardio

Max Pulso

Frecuencia Semanal

Guardar

## Suscripción

- En este apartado podemos contratar un plan de entrenamiento de nuestra preferencia junto con un horario a seleccionar.

← Suscripción

Suscripción actual

**Vigencia: 2025/5/25 - 2025/5/25**

\$499.99

**Beneficios:**

- ✓ 4 clases
- ✓ Flexible

**Historial de suscripciones**

🔄 4 clases - Flexible \$499.99  
2025/9/2 - 2025/10/2

Contratar

← Contratar suscripción

Back

Selecciona un paquete

← Contratar suscripción

Paquete 1

**Beneficios:**

- ✓ 4 clases
- ✓ Flexible

Seleccionar

← Selección tus horarios

	Lunes	Martes	Miércoles
08:00			
09:00			
10:00			
11:00			
12:00			
13:00			

Contratar

## Código Relevante

### Frontend

### Suscripción

#### Carrito.dart

##### 1.- Importaciones:

- **import 'package:flutter/material.dart';** Importa el paquete material de Flutter.
- **import 'package:proyecto/models/clase\_model.dart';** Importa la definición del modelo Clase
- **import 'package:proyecto/models/paquete\_model.dart';** Importa la definición del modelo Paquete.

**2.- Estructura de la pantalla (CarrutiPage):** Es una clase StatelessWidget que representa una vista de carrito, recibe como parámetros una liste de clases seleccionadas y un paquete opcional.

##### 3.- Body:

Si no hay clases: se muestra un texto que dice "No hay clases en el carrito."

Si hay clases:

- Se muestra la información del paquete (si está presente), incluyendo:
  - ID, precio, número de clases, si es flexible o no.
- Luego se muestra la lista de horarios seleccionados con detalles:
  - Día de la semana, hora de inicio/fin, tipo de curso.
  - Cada clase tiene un ícono de eliminar (aún no funcional).

**4.- bottomNavigationBar con botón de pago:** Solo aparece si hay clases en el carrito y el botón "Pagar" abre un diálogo de confirmación que dice "Tu suscripción ha sido registrada", luego redirige a otra pantalla: /suscripcioncliente.

5.- Usa ListView.builder: para renderizar dinámicamente las clases.

6.-Convierte el número del día (1-7) en el nombre del día (lunes a domingo).

7.-Se usan SnackBar y AlertDialog para retroalimentación visual del usuario.

```
lib > screens > cliente > pages > drawer > subscripcion > carrito.dart > CarritoPage > build
1  import 'package:flutter/material.dart';
2  import 'package:proyecto/models/clase_model.dart';
3  import 'package:proyecto/models/paquete_model.dart';
4
5  class CarritoPage extends StatelessWidget {
6    final List<Clase> clases;
7    final Paquete? paquete;
8
9    const CarritoPage({Key? key, required this.clases, this.paquete})
10      : super(key: key);
11
12    @override
13    Widget build(BuildContext context) {
14      return Scaffold(
15        appBar: AppBar(
16          title: Text('Carrito de Compras'),
17        ), // AppBar
18        body: Padding(
19          padding: const EdgeInsets.all(16.0),
20          child: clases.isEmpty
21            ? Center(child: Text('No hay clases en el carrito.'))
22            : Column(
23              crossAxisAlignment: CrossAxisAlignment.start,
24              children: [
25                if (paquete != null) ...[
26                  Text(
27                    'Paquete seleccionado:',
28                    style:
29                      TextStyle(fontWeight: FontWeight.bold, fontSize: 18),
30                  ), // Text
31                  ListTile(
32                    title: Text("Paquete ${paquete!.id}" ?? ''),
```

```

33 subtitle: Column(
34   crossAxisAlignment: CrossAxisAlignment.start,
35   children: [
36     Text("\${paquete!.precio.toString()} ?? ' '"),
37     SizedBox(height: 4),
38     Text('Beneficios:',
39       style: TextStyle(fontWeight: FontWeight.bold)), // Text
40     Row(
41       children: [
42         Icon(Icons.check, color: Colors.green, size: 20),
43         SizedBox(width: 6),
44         Text("${paquete!.clases} clases"),
45       ],
46     ), // Row
47     Row(
48       children: [
49         Icon(
50           paquete!.flexible ? Icons.check : Icons.close,
51           color: paquete!.flexible
52             ? Colors.green
53             : Colors.red,
54           size: 20), // Icon
55         SizedBox(width: 6),
56         Text("Flexible"),
57       ],
58     ), // Row
59   ],
60 ), // Column
61 ), // ListTile
62 Divider(),

```

```

63 ],
64 Text(
65   'Horarios seleccionados:',
66   style: TextStyle(fontWeight: FontWeight.bold, fontSize: 18),
67 ), // Text
68 Expanded(
69   child: ListView.builder(
70     itemCount: clases.length,
71     itemBuilder: (context, index) {
72       return Card(
73         margin: EdgeInsets.symmetric(vertical: 8),
74         child: ListTile(
75           leading: Icon(Icons.class_),
76           title: Text(
77             "${clases[index].casilla.dia == 1 ? "Lunes" : clases[index].casilla.dia == 2 ? "Martes"
78             style: TextStyle(fontWeight: FontWeight.bold),
79           ), // Text
80           trailing: IconButton(
81             icon: Icon(Icons.delete, color: Colors.red),
82             onPressed: () {
83               ScaffoldMessenger.of(context).showSnackBar(
84                 SnackBar(
85                   content: Text(
86                     'Función de eliminar no implementada'), // Text // SnackBar
87                 ),
88               );
89             }, // IconButton
90           ), // ListTile
91         ); // Card
92     },

```

```

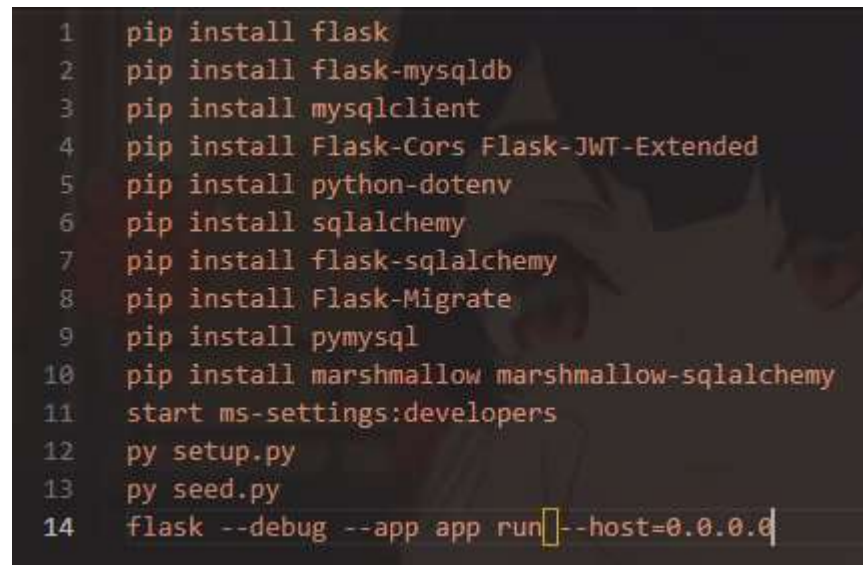
94         ), // Expanded
95     ],
96 ), // Column
97 ), // Padding
98 bottomNavigationBar: classes.isNotEmpty
99   ? Padding(
100     padding: const EdgeInsets.all(16.0),
101     child: ElevatedButton(
102       onPressed: () {
103         showDialog(
104           context: context,
105           builder: (context) => AlertDialog(
106             title: Text('¡Listo!'),
107             content: Text('Tu suscripción ha sido registrada.'),
108             actions: [
109               TextButton(
110                 onPressed: () => Navigator.pop(context),
111                 child: Text('OK'),
112               ), // TextButton
113             ],
114           ), // AlertDialog
115         ).then((_) {
116           Navigator.pushReplacementNamed(
117             context, '/suscripcioncliente');
118         });
119       },
120       child: Text('Pagar'),
121     ), // ElevatedButton
122   ) // Padding
123 : null,
124 // End of field

```

## Backend

### Configuración del backend

En la terminal tenemos que poner estos comandos para el correcto funcionamiento del backend, además de tener el XAMPP con mysql y Apache activos:



```
1  pip install flask
2  pip install flask-mysqldb
3  pip install mysqlclient
4  pip install Flask-Cors Flask-JWT-Extended
5  pip install python-dotenv
6  pip install sqlalchemy
7  pip install flask-sqlalchemy
8  pip install Flask-Migrate
9  pip install pymysql
10 pip install marshmallow marshmallow-sqlalchemy
11 start ms-settings:developers
12 py setup.py
13 py seed.py
14 flask --debug --app app run --host=0.0.0.0
```

**1.- pip install flask:**Instala Flask, un microframework para crear aplicaciones web en Python.

**2.- pip install flask-mysqldb:**Instala la extensión para usar MySQL con Flask a través del conector MySQLdb.

**3.- pip install mysqlclient:**Instala el cliente oficial de MySQL para Python (requerido por flask-mysqldb).

**4.- pip install Flask-CorsFlask-JWT-Extended:**

- **Flask-Cors:** Habilita CORS (Cross-OriginResourceSharing), útil cuando el frontend y backend están en dominios distintos.
- **Flask-JWT-Extended:** Permite la autenticación con JWT (JSON Web Tokens) en Flask.

**5.- pip install python-dotenv:**Permite cargar variables de entorno desde un archivo .env, útil para gestionar configuraciones sensibles (como contraseñas y claves API).

**6.- pip install sqlalchemy:**Instala SQLAlchemy, un ORM (ObjectRelationalMapper) para manejar bases de datos de forma más abstracta y orientada a objetos.

**7.- pip install flask-sqlalchemy:**Integra SQLAlchemy con Flask, permitiendo una forma más sencilla de definir modelos y manejar la base de datos desde la aplicación.

**8.- pip install Flask-Migrate:**Añade soporte de migraciones de base de datos con Alembic, útil para gestionar cambios en los modelos de datos de forma ordenada.

**9.- pip install pymysql:**Cliente alternativo para conectar con MySQL, compatible con SQLAlchemy.

**10.- pip install marshmallowmarshmallow-sqlalchemy:**

- **marshmallow:** Serializador/deserializador de objetos (por ejemplo, convertir modelos a JSON).
- **marshmallow-sqlalchemy:** Integración de Marshmallow con SQLAlchemy.

**11.- pip install bcrypt:** Hashing bcrypt para el lado del servidor y cliente.

## **Comandos del sistema y scripts**

**11.- startms-settings:developers:**Abre la configuración de desarrollador de Windows, necesaria a veces para ejecutar scripts o activar funciones como el modo de desarrollo.

**12.- py setup.py:**Ejecuta un script llamado setup.py con Python. Este script suele contener la inicialización de base de datos, creación de tablas o configuración inicial.

**13.- py seed.py:**Ejecuta el script seed.py con Python. Generalmente, este script puebla la base de datos con datos de prueba o iniciales (seeding).

## **Comando para ejecutar la app Flask**

**14.- flask --debug --app app run --host=0.0.0.0**

- Inicia la aplicación Flask especificando que el archivo principal es app.py (o un paquete llamado app).
- **--debug:** Activa el modo de desarrollo (recarga automática y mejores mensajes de error).



- **--host=0.0.0.0:** Hace que Flask escuche en todas las interfaces de red, útil para acceder desde otros dispositivos en la red.

## Cuenta

```
1 from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
2 from marshmallow import fields
3 from models.cuenta import Cuenta
4 import base64
5
6 class CuentaSchema(SQLAlchemyAutoSchema):
7     class Meta:
8         model = Cuenta
9         load_instance = True
10        include_fk = True
11        include_relationships = True
12
13    avatar = fields.Method("get_avatar_as_base64")
14
15    def get_avatar_as_base64(self, obj):
16        if obj.avatar:
17            if isinstance(obj.avatar, bytes):
18                return base64.b64encode(obj.avatar).decode("utf-8")
19            else:
20                try:
21                    return base64.b64encode(bytes(obj.avatar, 'utf-8')).decode("utf-8")
22                except Exception:
23                    return None
24        return None
25
```

### Importaciones:

- **SQLAlchemyAutoSchema:** clase de Marshmallow-SQLAlchemy que genera automáticamente un esquema basado en un modelo de SQLAlchemy (en este caso, Cuenta).
- **fields:** se usa para definir campos personalizados o adicionales.
- **Cuenta:** es el modelo (ORM) de base de datos importado desde el módulo models.cuenta.
- **base64:** módulo estándar de Python para codificar/decodificar datos en formato Base64 (útil para imágenes o binarios).

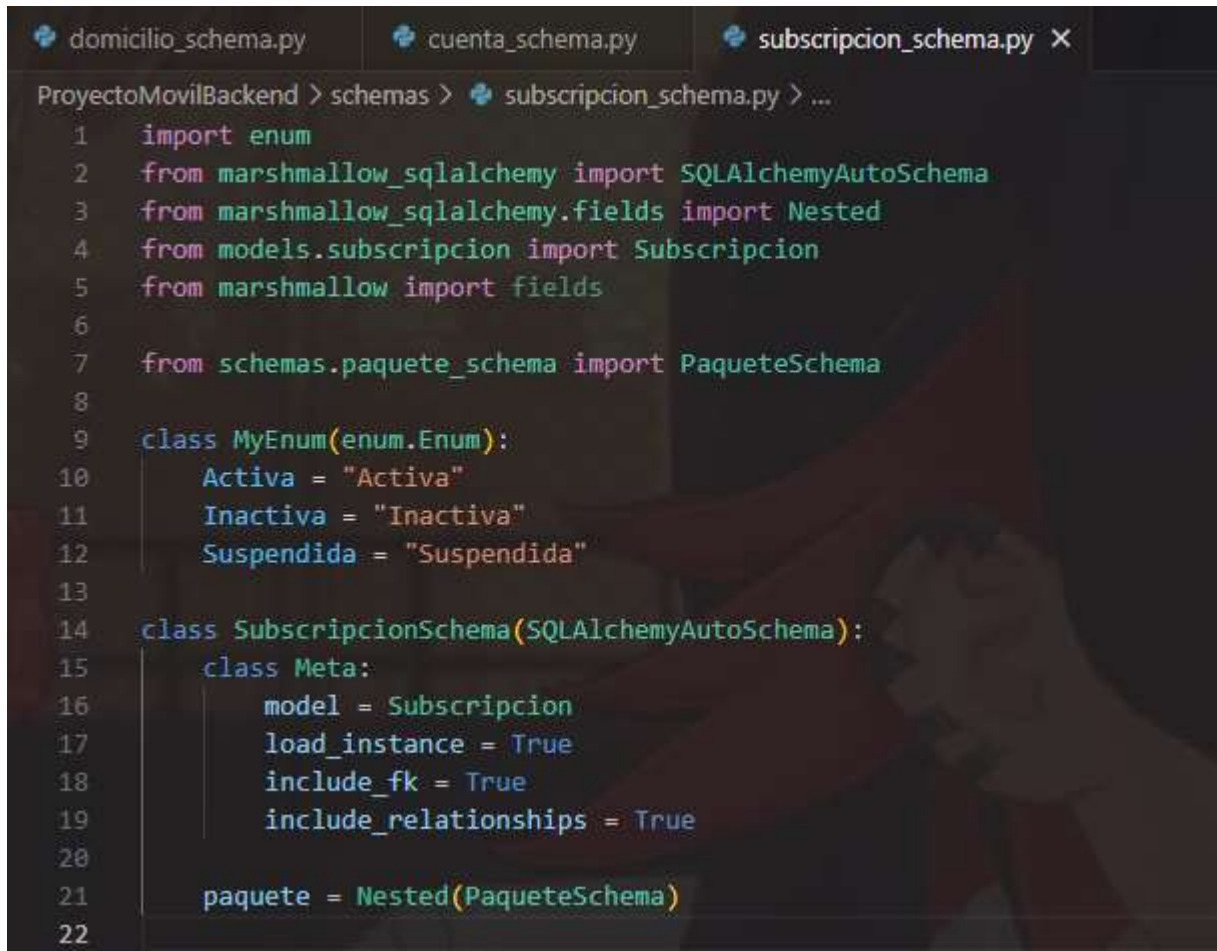
**Clase CuentaSchema:** Define un esquema de serialización/deserialización para el modelo Cuenta, para convertir los objetos de la base de datos a Json y viceversa.

**Clase Meta (parte interna):** Configura el comportamiento del esquema, es decir, indica que el esquema se basa en el modelo cuenta, el esquema carga directamente instancias de SQLAlchemy, incluye claves foráneas en la serialización, y por último incluye relaciones entre las tablas si existen.



**Método `get_avatar_as_base64`:** Este método transforma la imagen/avatar en formato binario (bytes) a una cadena codificada en Base64, para poder ser enviada como texto en respuestas JSON.

## Suscripción



```
ProyectoMovilBackend > schemas > subscription_schema.py > ...
1  import enum
2  from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
3  from marshmallow_sqlalchemy.fields import Nested
4  from models.subscription import Subscription
5  from marshmallow import fields
6
7  from schemas.paquete_schema import PaqueteSchema
8
9  class MyEnum(enum.Enum):
10     Activa = "Activa"
11     Inactiva = "Inactiva"
12     Suspendida = "Suspendida"
13
14  class SubscriptionSchema(SQLAlchemyAutoSchema):
15     class Meta:
16         model = Subscription
17         load_instance = True
18         include_fk = True
19         include_relationships = True
20
21     package = Nested(PaqueteSchema)
22
```

### Importaciones:

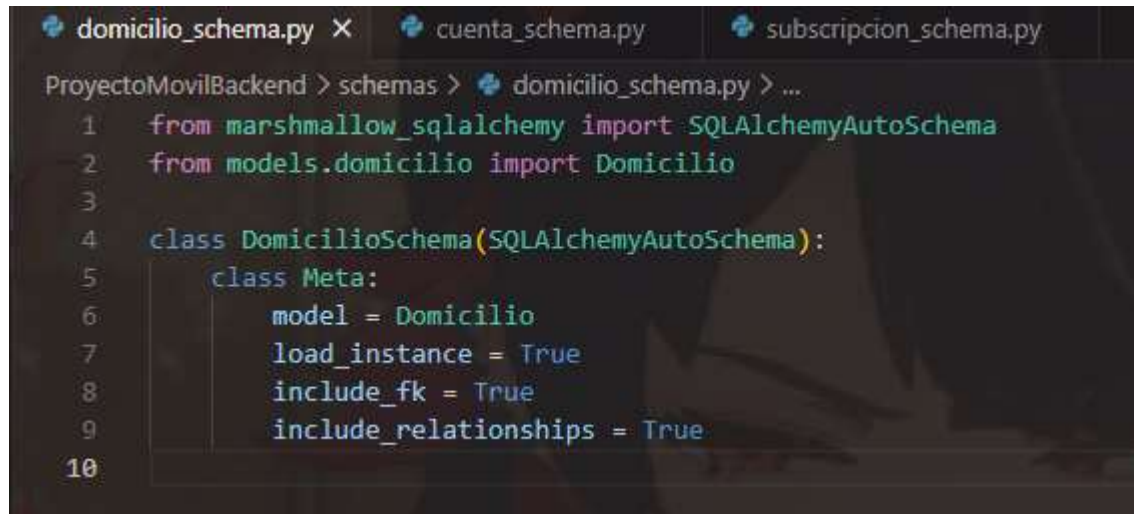
- **enum**: módulo de Python para definir **enumeraciones** (listas de valores constantes).
- **SQLAlchemyAutoSchema**: crea automáticamente un esquema Marshmallow a partir de un modelo SQLAlchemy.
- **Nested**: permite incluir otro esquema anidado (relación entre tablas).
- **Subscription**: es el modelo de base de datos al que corresponde el esquema.
- **fields**: módulo para definir campos personalizados en el esquema.
- **PaqueteSchema**: es un esquema previamente definido que representa a un modelo relacionado (probablemente la tabla Paquete).

**Enumeración:** Define una **enumeración personalizada** llamada MyEnum, que representa los posibles estados de una suscripción: Activa, inactiva y suspendida

**Clase SubscriptionSchema:** Define el esquema que representa el modelo Subscription.

**Clase Meta interna:** Configura el comportamiento del esquema.

## Domicilio



```
ProyectoMovilBackend > schemas > domicilio_schema.py > ...
1  from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
2  from models.domicilio import Domicilio
3
4  class DomicilioSchema(SQLAlchemyAutoSchema):
5      class Meta:
6          model = Domicilio
7          load_instance = True
8          include_fk = True
9          include_relationships = True
10
```

### Importaciones:

- **SQLAlchemyAutoSchema:** clase de Marshmallow que **genera automáticamente un esquema** basado en un modelo SQLAlchemy.
- **Domicilio:** es el modelo que representa una entidad de dirección o residencia (como calle, número, colonia, etc.).

**Clase DomicilioSchema:** Aquí se declara un esquema llamado DomicilioSchema, que usará el modelo Domicilio para crear reglas de serialización/deserialización.

**Clase interna Meta:** Esta clase Meta configura cómo se comporta el esquema.

## **Conclusión**

En el desarrollo de esta aplicación, se hizo uso de conocimientos adquiridos de esta y otras materias para realizar la gestión de usuarios, seguridad, la programación y personalización de diversos aspectos. El uso de flutterfacilito la creación de una interfaz fluida y compatible con múltiples plataformas desde un solo código base. Y el uso de Python, se realizo de manera fluida gracias a el conocimiento y aplicación en otras materias como estructura para el acceso a base de datos y seguridad.