Desis	**REQUERIMIENTO TÉCNICO**

1.- IDENTIFICACIÓN DEL REQUERIMIENTO			
Proyecto	Facturacion.cl		
Folio	2	**Versión**	1.0
Nombre	Creación de Interfaz de Cuenta		
Nº de entrega	1		

2.- DATOS DE ENTREGA	
Fecha Entrega	23-10-2024
Nombre Analista	Jeremias Campos
Observación	Implementación de una interfaz para la creación de cuentas conforme a las especificaciones y validaciones requeridas.

Modificar Archivo			
Acción	**Crear Pantalla**: creación_cuenta_screen.dart - CreacionCuentaScreen	***Nombre***	Creación de interfaz de cuenta con Flutter y Dart lib/creacion_cuenta_sreen.dart

Estructura del Formulario

- Campos incluidos: Nombre, Correo Electrónico, Dirección, Fecha de Nacimiento, Número de Teléfono, Contraseña.
- Botones: "Registrar usuario" y "Obtener Desde API".

Controladores de Texto y Validadores

- Implementar validaciones para:
  - **Nombre:**
    Crear función Validar Nombre, la cual recibirá (valor) y realizará lo siguiente:
  - **Si el valor es nulo o está vacío**:
    - **Acción**: Devolver el mensaje "El campo Nombre es obligatorio".
  - **Si el valor NO contiene solo letras y espacios**:
    - **Acción**: Devolver el mensaje "Solo se permiten letras y espacios".
  - **Si la longitud del valor es menor a 2 o mayor a 50 caracteres**:
    - **Acción**: Devolver el mensaje "El nombre debe tener entre 2 y 50 caracteres".
  - **Si el valor cumple con todas las validaciones anteriores**:
    - **Acción**: Devolver null para indicar que no hay errores de validación.

  - **Correo Electrónico:**
  - Crear función Validar Correo Electrónico, la cual recibirá valor y realizará lo siguiente:
  - **Si el valor es nulo o está vacío**:
    - **Acción**: Devolver el mensaje "El campo Correo es obligatorio".
  - **Si el valor NO tiene un formato válido de correo**:
    - **Acción**: Devolver el mensaje "Ingrese un correo válido".
  - **Si el valor cumple con todas las validaciones anteriores**:
    - **Acción**: Devolver null para indicar que no hay errores de validación.

  - **Dirección:**
  - Crear función Validar Dirección, la cual recibirá (valor) y realizará lo siguiente:
  - **Si el valor es nulo o está vacío**:
    - **Acción**: Devolver el mensaje "El campo Dirección es obligatorio".
  - **Si la longitud del valor es menor a 5 o mayor a 100 caracteres**:
    - **Acción**: Devolver el mensaje "La dirección debe tener entre 5 y 100 caracteres".
  - **Si el valor cumple con todas las validaciones anteriores**:
    - **Acción**: Devolver null para indicar que no hay errores de validación.

  - **Fecha de Nacimiento:**
  - Crear función Validar Fecha de Nacimiento, la cual recibirá valor y realizará lo siguiente:
  - **Si el valor es nulo o está vacío**:
    - **Acción**: Devolver el mensaje "El campo Fecha de Nacimiento es obligatorio".
  - **Si la fecha ingresada es una fecha futura**:
    - **Acción**: Devolver el mensaje "La fecha debe ser pasada".
  - **Si el valor cumple con todas las validaciones anteriores**:
    - **Acción**: Devolver null para indicar que no hay errores de validación.

  - **Número de Teléfono**:
  - Crear función Validar Número de Teléfono, la cual recibirá valor y realizará lo siguiente:
  - **Si el valor es nulo o está vacío**:
    - **Acción**: Devolver el mensaje "El campo Celular es obligatorio".
  - **Si el valor NO contiene solo números**:
    - **Acción**: Devolver el mensaje "Ingrese un celular válido (10-15 dígitos)".
  - **Si la longitud del valor es menor a 10 o mayor a 15 dígitos**:

- **Acción**: Devolver el mensaje "Ingrese un celular válido (10-15 dígitos)".
- **Si el valor cumple con todas las validaciones anteriores**:
  - **Acción**: Devolver null para indicar que no hay errores de validación.

- **Contraseña**:
- Crear función Validar Contraseña, la cual recibirá valor y realizará lo siguiente:
- **Si el valor es nulo o está vacío**:
  - **Acción**: Devolver el mensaje "El campo Contraseña es obligatorio".
- **Si la longitud del valor es menor a 8 caracteres**:
  - **Acción**: Devolver el mensaje "La contraseña debe tener al menos 8 caracteres".
- **Si el valor NO contiene mayúsculas, minúsculas, números y caracteres especiales**:
  - **Acción**: Devolver el mensaje "Debe incluir mayúscula, minúscula, número y carácter especial válido".
- **Si el valor cumple con todas las validaciones anteriores**:
  - **Acción**: Devolver null para indicar que no hay errores de validación.

Visualización de Validaciones

- Campos inválidos: bordes en rojo y mensajes de error debajo de cada campo.

Comportamiento de Botones

- **Registrar Usuario**: Deshabilitado hasta que todos los campos sean válidos.
- **Obtener Desde API**: Obtiene datos de https://randomuser.me/ y los completa en los campos.

IMAGEN 1

```
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'El campo Nombre es obligatorio';
          } else if (!RegExp(r'^[a-zA-Z\s]+$').hasMatch(value)) {
            return 'Solo se permiten letras y espacios';
          } else if (value.length < 2 || value.length > 50) {
            return 'El nombre debe tener entre 2 y 50 caracteres';
          }
          return null;
        },
        onSaved: (value) => nombre = value!,
      ),  // TextFormField
      SizedBox(height: 10),
      TextFormField(
        controller: _correoController,
        decoration: InputDecoration(
          labelText: 'Correo Electrónico',
          border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)),
        ),  // InputDecoration
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'El campo Correo es obligatorio';
          } else if (!RegExp(r'^[^@]+@[^@]+\.[^@]+').hasMatch(value)) {
            return 'Ingrese un correo válido';
          }
          return null;
        },
        onSaved: (value) => correo = value!,
      ),  // TextFormField
      SizedBox(height: 10),
      TextFormField(
        controller: _direccionController,
        decoration: InputDecoration(
          labelText: 'Dirección',
          border: OutlineInputBorder(borderRadius: BorderRadius.circular(10.0)),
        ),  // InputDecoration
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'El campo Dirección es obligatorio';
          } else if (value.length < 5 || value.length > 100) {
            return 'La dirección debe tener entre 5 y 100 caracteres';
```

Requerimiento ####

IMAGEN 2

```dart
Future<void> _obtenerDesdeApi() async {
  setState(() {
    _isObtenerDesdeApiButtonDisabled = true;
  });

  try {
    final response = await http.get(Uri.parse('https://randomuser.me/api/'));
    if (response.statusCode == 200) {
      final data = jsonDecode(response.body)['results'][0];
      setState(() {
        _nombreController.text = '${data['name']['first']} ${data['name']['last']}';
        _correoController.text = data['email'];
        DateTime fecha = DateTime.parse(data['dob']['date']);
        fechaNacimiento = '${fecha.day.toString().padLeft(2, '0')}/${fecha.month.toString().padLeft(2, '0')}/${fecha.year}';
        _direccionController.text = data['location']['street']['name'];
        _fechaNacimientoController.text = fechaNacimiento;

        String celularApi = data['cell'] ?? '0000000000';
        _celularController.text = celularApi.replaceAll(RegExp(r'[^0-9]'), '');

        _contrasenaController.text = data['login']['password'] ?? 'Password123';

        _isRegistrarButtonEnabled = _formKey.currentState!.validate();
      });

      _mostrarSnackBar('Datos obtenidos correctamente.');
    } else {
      print('Error en la respuesta de la API. Código: ${response.statusCode}');
      print('Respuesta: ${response.body}');
      _mostrarSnackBar('Error al obtener datos de la API.');
    }
  } catch (e) {
    print('Error durante la solicitud HTTP: $e');
    _mostrarSnackBar('Error al obtener datos de la API.');
  } finally {
    setState(() {
      _isObtenerDesdeApiButtonDisabled = false;
    });
  }
}
```

IMAGEN 3

```dart
ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: _isRegistrarButtonEnabled ? Color(0xFF1A5DD9) : Colors.grey,
    padding: EdgeInsets.symmetric(vertical: 15),
    minimumSize: Size(double.infinity, 50),
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10.0)),
  ),
  onPressed: _isRegistrarButtonEnabled
      ? () {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      _guardarUsuario();
    }
  }
      : null,
  child: Text('Registrar Usuario', style: TextStyle(fontSize: 16, color: Colors.white)),
),  // ElevatedButton
SizedBox(height: 10),
ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: _isObtenerDesdeApiButtonDisabled ? Colors.grey : Color(0xFF1A5DD9),
    padding: EdgeInsets.symmetric(vertical: 15),
    minimumSize: Size(double.infinity, 50),
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10.0)),
  ),
  onPressed: _isObtenerDesdeApiButtonDisabled ? null : _obtenerDesdeApi,
  child: Text('Obtener Desde API', style: TextStyle(fontSize: 16, color: Colors.white)),
),  // ElevatedButton
```

Requerimiento ####

Modificar Archivo			
Acción	Inicialización de la Base de Datos	***Nombre***	Base de Datos (usuarios.db)

- **Inicialización de la Base de Datos**
- La inicialización de la base de datos en la aplicación se realiza utilizando getApplicationDocumentsDirectory () para obtener la ruta del directorio donde se almacenará la base de datos. Luego, se define la ruta de la base de datos como usuarios.db y se crea la tabla usuarios si no existe. La estructura de la tabla usuarios incluye las siguientes columnas:

Estructura de la tabla usuarios:

- **id**:
  - **Tipo de dato**: INTEGER
  - **Descripción**: Clave primaria auto incrementable que identifica de manera única a cada registro en la tabla.
  - **Propiedades**: PRIMARY KEY AUTOINCREMENT
- **nombre**:
  - **Tipo de dato**: TEXT
  - **Descripción**: Almacena el nombre completo del usuario, solo permite letras y espacios.
  - **Validaciones esperadas**: Longitud entre 2 y 50 caracteres.
- **correo**:
  - **Tipo de dato**: TEXT
  - **Descripción**: Almacena el correo electrónico del usuario.
  - **Validaciones esperadas**: Debe ser un formato válido de correo electrónico (por ejemplo, usuario@dominio.com).
- **direccion**:
  - **Tipo de dato**: TEXT
  - **Descripción**: Almacena la dirección de residencia del usuario.
  - **Validaciones esperadas**: Longitud entre 5 y 100 caracteres.
- **fechaNacimiento**:
  - **Tipo de dato**: TEXT
  - **Descripción**: Almacena la fecha de nacimiento del usuario en formato DD/MM/YYYY.
  - **Validaciones esperadas**: Debe ser una fecha pasada, válida en formato DD/MM/YYYY.
- **contrasena**:
  - **Tipo de dato**: TEXT
  - **Descripción**: Almacena la contraseña del usuario de forma segura.
  - **Validaciones esperadas**: Mínimo 8 caracteres, debe incluir mayúsculas, minúsculas, números y caracteres especiales.
- **celular**:
  - **Tipo de dato**: TEXT
  - **Descripción**: Almacena el número de teléfono del usuario, permitiendo solo números.
  - **Validaciones esperadas**: Longitud entre 10 y 15 dígitos.

-

IMAGEN 1

```dart
Future<void> _initDatabase() async {
  final documentsDirectory = await getApplicationDocumentsDirectory();
  final path = p.join(documentsDirectory.path, 'usuarios.db');

  database = await openDatabase(
    path,
    version: 3,
    onCreate: (db, version) async {
      await db.execute('''
      CREATE TABLE usuarios(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nombre TEXT,
        correo TEXT,
        direccion TEXT,
        fechaNacimiento TEXT,
        contrasena TEXT,
        celular TEXT
      )
    ''');
    },
  );

  setState(() {
    _isDatabaseInitialized = true;
  });
}
```

Requerimiento ####