

Sistema de Gestión de Créditos

Documentación Técnica

Desarrollado por: Irvin Segura Ortiz

Fecha: 10 de Marzo de 2025

Para: DELTA DATA

Índice

Contenido

Índice	2
1. Introducción	3
2. Tecnologías Utilizadas	3
2.1. Backend	3
2.2. Frontend	3
3. Instalación y Configuración	4
3.1. Requisitos Previos	4
3.2. Pasos para la Instalación	4
4. Arquitectura del Sistema	6
5. Funcionalidades del Sistema	7
6. Seguridad del Sistema	10
Diagrama de Clase	11
Diagrama de Casos de Uso	12
Diagrama de Objetos	12
Diagrama de Actividad	13
Diagrama de Secuencia	14

1. Introducción

El Sistema de Gestión de Créditos es una aplicación web diseñada para facilitar la administración de créditos otorgados a clientes. Esta plataforma permite a los usuarios registrar, visualizar, editar y eliminar créditos de manera eficiente. Adicionalmente, cuenta con herramientas de análisis visual para proporcionar estadísticas detalladas sobre los datos almacenados.

El sistema está desarrollado utilizando tecnologías modernas como Flask para la gestión del backend, SQLite como base de datos, y Bootstrap junto con Chart.js para la creación de una interfaz de usuario dinámica y atractiva.

Este documento tiene como objetivo describir en detalle la arquitectura, funcionalidades y procedimientos de instalación del sistema, así como proponer futuras mejoras para su optimización.

2. Tecnologías Utilizadas

2.1. Backend

- Flask: Framework ligero de Python que permite desarrollar aplicaciones web de manera sencilla y flexible.
- SQLAlchemy: Herramienta ORM (Object Relational Mapper) que facilita la interacción con bases de datos relacionales.
- SQLite: Base de datos ligera y auto-contenida, ideal para aplicaciones de tamaño pequeño a mediano.

2.2. Frontend

- HTML, CSS y JavaScript: Lenguajes esenciales para el desarrollo de interfaces web.
- Bootstrap: Framework CSS que permite diseñar interfaces responsivas y atractivas con menor esfuerzo.

- SweetAlert2: Biblioteca de JavaScript utilizada para mostrar alertas y cuadros de diálogo interactivos.
- Chart.js: Librería que permite la generación de gráficos estadísticos de manera sencilla y personalizable.

Link del Repo: <https://github.com/IrvinSegura/Registros-de-Creditos>

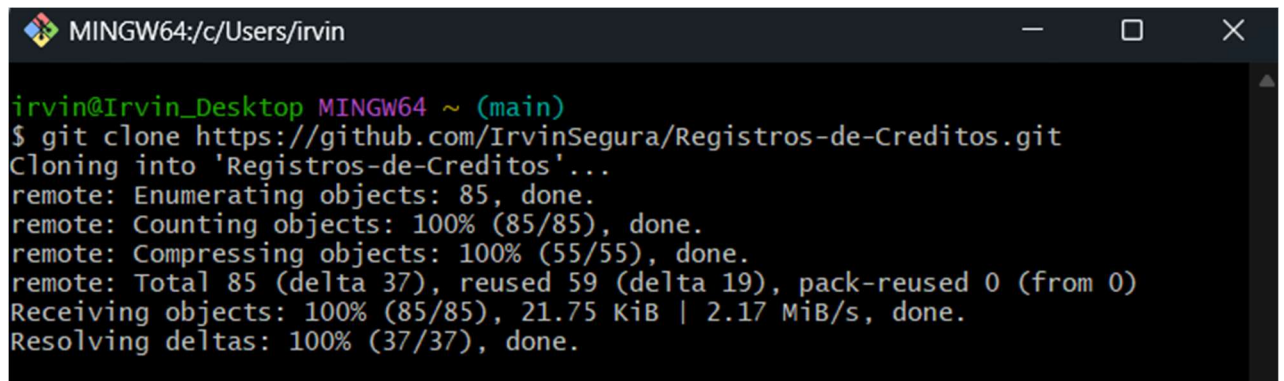
3. Instalación y Configuración

3.1. Requisitos Previos

- Tener instalado Python 3.x.
- Disponer de Git para clonar el repositorio.
- Instalar un entorno virtual (opcional pero recomendado).

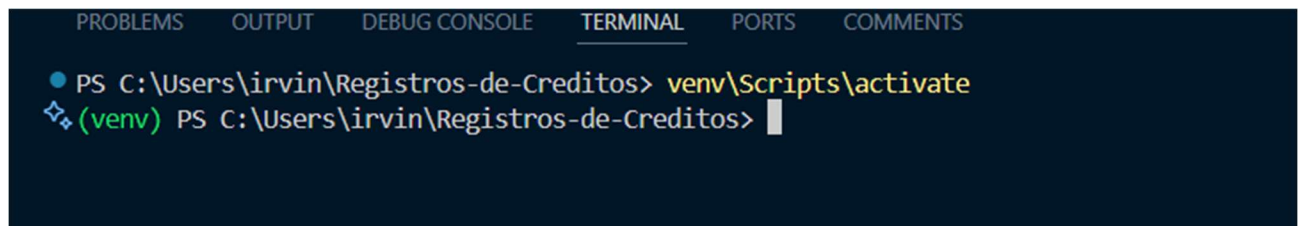
3.2. Pasos para la Instalación

- Clonar el repositorio:



```
irvin@Irvin_Desktop MINGW64 ~ (main)
$ git clone https://github.com/IrvinSegura/Registros-de-Creditos.git
Cloning into 'Registros-de-Creditos'...
remote: Enumerating objects: 85, done.
remote: Counting objects: 100% (85/85), done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 85 (delta 37), reused 59 (delta 19), pack-reused 0 (from 0)
Receiving objects: 100% (85/85), 21.75 KiB | 2.17 MiB/s, done.
Resolving deltas: 100% (37/37), done.
```

- Crear y activar un entorno virtual:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
● PS C:\Users\irvin\Registros-de-Creditos> venv\Scripts\activate
❖ (venv) PS C:\Users\irvin\Registros-de-Creditos> |
```

- Instalar las dependencias:

```
(venv) PS C:\Users\irvin\Registros-de-Creditos> pip install flask flask_sqlalchemy flask_cors
Collecting flask
  Using cached flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting flask_sqlalchemy
  Using cached flask_sqlalchemy-3.1.1-py3-none-any.whl.metadata (3.4 kB)
```

- Ejecutar la aplicación:

```
(venv) PS C:\Users\irvin\Registros-de-Creditos> python routes.py
* Serving Flask app 'routes'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 912-591-288
127.0.0.1 - - [09/Mar/2025 14:27:57] "GET / HTTP/1.1" 200 -
```

- Acceder a la aplicación en el navegador:

<http://127.0.0.1:5000/>

Registro de Créditos

RegistroCréditos RegistradosEstadísticas

Formulario de Registro

Cliente:

Fecha de Otorgamiento:

Tasa de Interés:

Plazo (meses):

Monto:

Registrar

4. Arquitectura del Sistema

El sistema sigue el patrón de diseño Modelo-Vista-Controlador (MVC), donde:

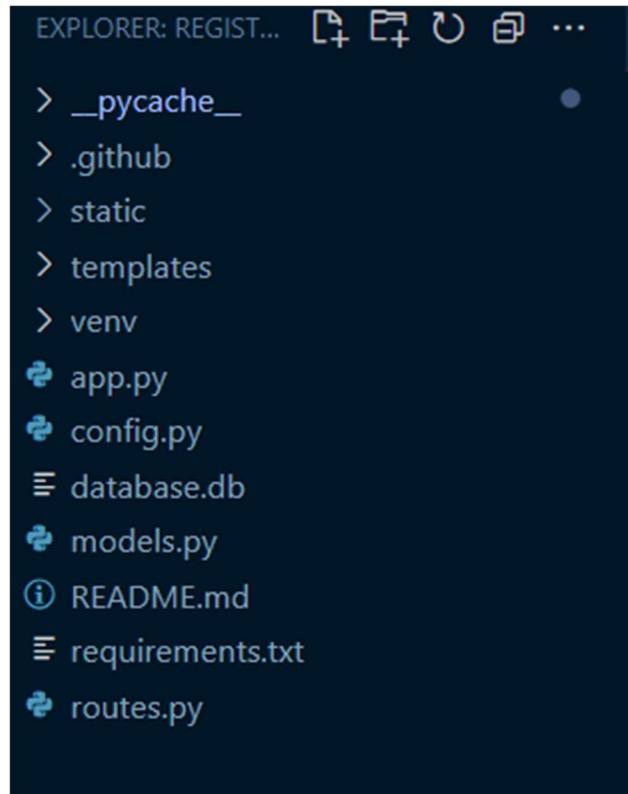
- El modelo (Model) gestiona la base de datos y la lógica de negocio.

```
1 from flask_sqlalchemy import SQLAlchemy
2
3 db = SQLAlchemy()
4
5 class Credito(db.Model):
6     id = db.Column(db.Integer, primary_key=True, autoincrement=True)
7     cliente = db.Column(db.String(100), nullable=False)
8     monto = db.Column(db.Float, nullable=False)
9     tasa_interes = db.Column(db.Float, nullable=False)
10    plazo = db.Column(db.Integer, nullable=False)
11    fecha_otorgamiento = db.Column(db.String(10), nullable=False)
12
```

- La vista (View) se encarga de la presentación y la interfaz de usuario.

```
1 <!-- Register Form -->
2 <div class="tab-pane fade show active" id="registro" role="tabpanel">
3   <div class="card shadow-sm mb-3" style="max-width: 70%; margin: auto;">
4     <div class="card-header text-center p-2">Formulario de Registro</div>
5     <div class="card-body p-3">
6       <form id="formulario" class="needs-validation" novalidate>
7         <div class="row mb-2">
8           <div class="col-6">
9             <label for="cliente" class="form-label">Cliente:</label>
10            <input type="text" id="cliente" class="form-control form-control-sm required">
11          </div>
12          <div class="col-6">
13            <label for="fecha_otorgamiento" class="form-label">Fecha de Otorgamiento:</label>
14            <input type="date" id="fecha_otorgamiento" class="form-control form-control-sm required">
15          </div>
16        </div>
17        <div class="row mb-2">
18          <div class="col-4">
19            <label for="tasa_interes" class="form-label">Tasa de Interés:</label>
20            <input type="number" step="0.01" id="tasa_interes" class="form-control form-control-sm required">
21          </div>
22          <div class="col-4">
23            <label for="plazo" class="form-label">Plazo (meses):</label>
24            <input type="number" id="plazo" class="form-control form-control-sm required inputmode="numeric" pattern="\d+>
25          </div>
26          <div class="col-4">
27            <label for="monto" class="form-label">Monto:</label>
28            <input type="number" id="monto" class="form-control form-control-sm required inputmode="numeric" pattern="\d+>
29          </div>
30        </div><br>
31        <div class="text-center">
32          <button type="submit" class="btn btn-primary btn-sm">Registrar</button>
33        </div>
34      </form>
35    </div>
36  </div>
37 </div>
38 <!-- End Register Form -->
```

- El controlador (Controller) gestiona las solicitudes del usuario y la lógica de flujo de datos.



5. Funcionalidades del Sistema

- Registrar un Crédito: Permite al usuario ingresar los datos de un nuevo crédito.

Formulario de Registro

Cliente:

Fecha de Otorgamiento:

Tasa de Interés:

Plazo (meses):

Monto:

Registrar

- Listar Créditos: Muestra todos los créditos registrados en una tabla.

Registro

Créditos Registrados

Estadísticas

Créditos Registrados						
Cliente	Monto	Tasa de Interés	Plazo (meses)	Fecha de Otorgamiento	Acciones	
Tito	5000	100%	360	2025-08-03	<div><div></div> Editar</div>	<div><div></div> Eliminar</div>
Irvin	10000	100%	360	2026-05-01	<div><div></div> Editar</div>	<div><div></div> Eliminar</div>
Jose	20000	100%	360	2025-05-02	<div><div></div> Editar</div>	<div><div></div> Eliminar</div>

- Editar un Crédito: Permite modificar los datos de un crédito existente.

Editar Crédito

Cliente

Tito

Monto

5000

Tasa de Interés (%)

100

Plazo (meses)

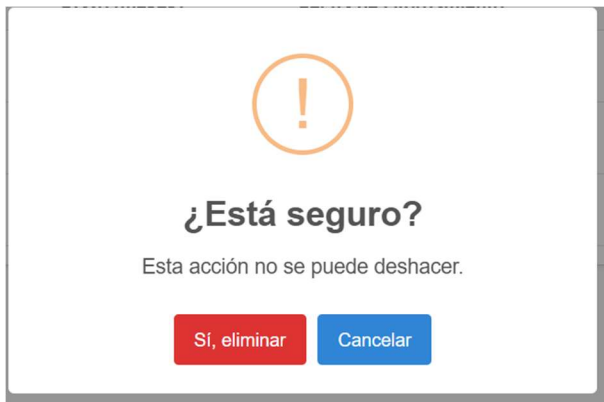
360

Fecha de Otorgamiento

08/03/2025

Guardar Cambios

- **Eliminar un Crédito:** Permite eliminar un crédito de la base de datos.

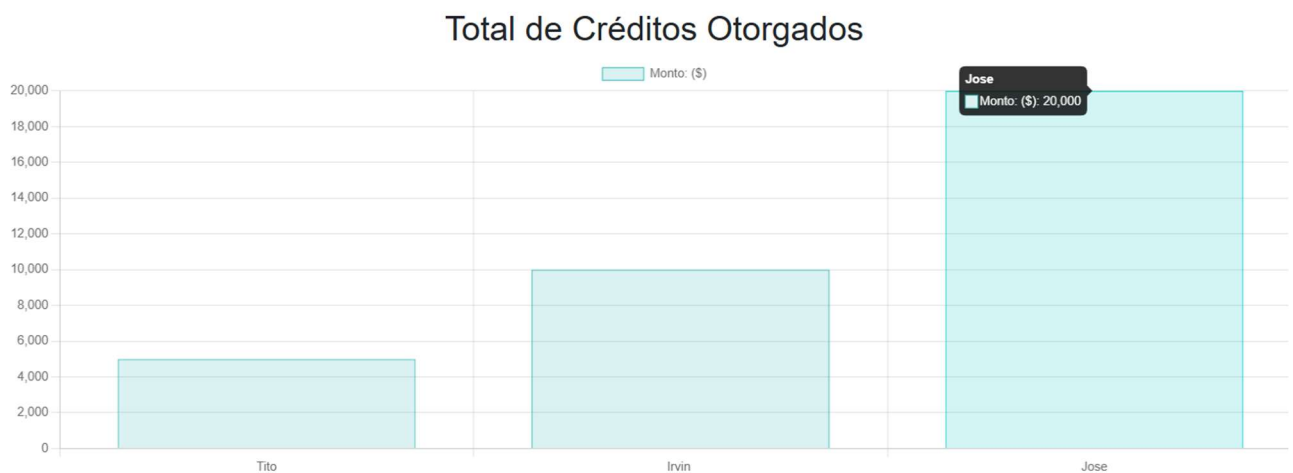
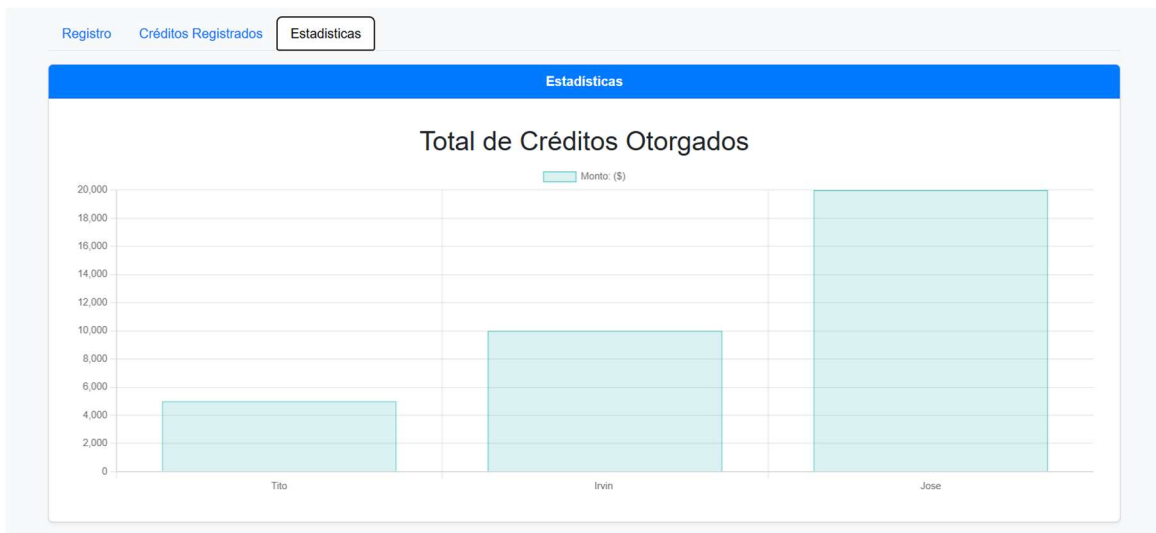


Eliminado

El crédito ha sido eliminado correctamente.

OK

- **Visualización de Estadísticas:** Genera gráficos estadísticos utilizando Chart.js.



6. Seguridad del Sistema

- Validación de datos para evitar inyecciones SQL.

```
1 // DOM elements for the diferents
2 document.addEventListener("DOMContentLoaded", function () {
3   const form = document.getElementById("formulario");
4   const formEditor = document.getElementById("formEditorCredito");
5   const inputs = form.querySelectorAll("input");
6   const inputsEditor = formEditor.querySelectorAll("input");
7   const submitButtonEditor = formEditor.querySelector("button[type='submit']");
8   const initialValuesEditor = {};
9
10  // Event listener for the inputs
11  inputs.forEach(input => input.addEventListener("input", () => validarCampo(input)));
12  inputsEditor.forEach(input => {
13    initialValuesEditor[input.id] = input.value.trim();
14    input.addEventListener("input", () => validarCampo(input));
15  });
16
17  // Event listener for the form submit
18  form.addEventListener("submit", async function (e) {
19    e.preventDefault();
20    if (!inputs.every(validarCampo)) return;
21    await enviarDatos("POST", "/creditos", obtenerDatosFormulario());
22    actualizarGrafico();
23  });
24
25  // Event listener for the button edit
26  formEditor.addEventListener("input", () => {
27    submitButtonEditor.disabled = !inputsEditor.some(input => input.value.trim() !== initialValuesEditor[input.id]);
28  });
29
30  // Event send data to the endpoint
31  formEditor.addEventListener("submit", async function (e) {
32    e.preventDefault();
33    if (!inputsEditor.every(validarCampo)) return;
34    let id = document.getElementById("edit-id").value;
35    await enviarDatos("PUT", "/creditos/${id}", obtenerDatosFormulario(true));
36    actualizarGrafico();
37  });
38
39  cargarCreditos();
40  cargarCreditosParaGrafica();
41 });
```

```
1 / Validations in the inputs
2 const validaciones = {
3   "cliente": value => /^[A-Z][a-zA-Zs]+$/.test(value) || "El nombre debe empezar con mayúscula y no contener números.",
4   "edit-cliente": value => validaciones["cliente"](value),
5   "fecha_otorgamiento": value => {
6     let fecha = new Date(value), hoy = new Date();
7     hoy.setHours(0, 0, 0, 0);
8     return !isNaN(fecha.getTime()) && fecha >= hoy || "Ingrese una fecha válida y no anterior a hoy.";
9   },
10  "edit-fecha_otorgamiento": value => validaciones["fecha_otorgamiento"](value),
11  "tasa_interes": value => {
12    let tasa = parseFloat(value);
13    return !isNaN(tasa) && tasa >= 1 && tasa <= 100 || "La tasa debe ser un número entre 1 y 100.";
14  },
15  "edit-tasa_interes": value => validaciones["tasa_interes"](value),
16  "plazo": value => {
17    let plazo = parseInt(value);
18    return !isNaN(plazo) && plazo > 0 && plazo <= 360 || "El plazo debe ser mayor a 0 y menor o igual a 360.";
19  },
20  "edit-plazo": value => validaciones["plazo"](value),
21  "monto": value => {
22    let monto = parseInt(value);
23    return !isNaN(monto) && monto > 0 || "El monto debe ser un número mayor a 0.";
24  },
25  "edit-monto": value => validaciones["monto"](value)
26 };
27
28 // Function to validate the inputs
29 function validarCampo(input) {
30   let errorSpan = obtenerErrorSpan(input);
31   let resultado = validaciones[input.id](input.value.trim());
32   errorSpan.textContent = typeof resultado === "string" ? resultado : "";
33   return typeof resultado !== "string";
34 }
35
36 // Function to get the error span
37 function obtenerErrorSpan(input) {
38   let errorSpan = input.nextElementSibling;
39   if (!errorSpan || !errorSpan.classList.contains("error-message")) {
40     errorSpan = document.createElement("span");
41     errorSpan.classList.add("error-message");
42     errorSpan.style.color = "red";
43     input.parentNode.appendChild(errorSpan);
44   }
45   return errorSpan;
46 }
```

Diagrama de Clase

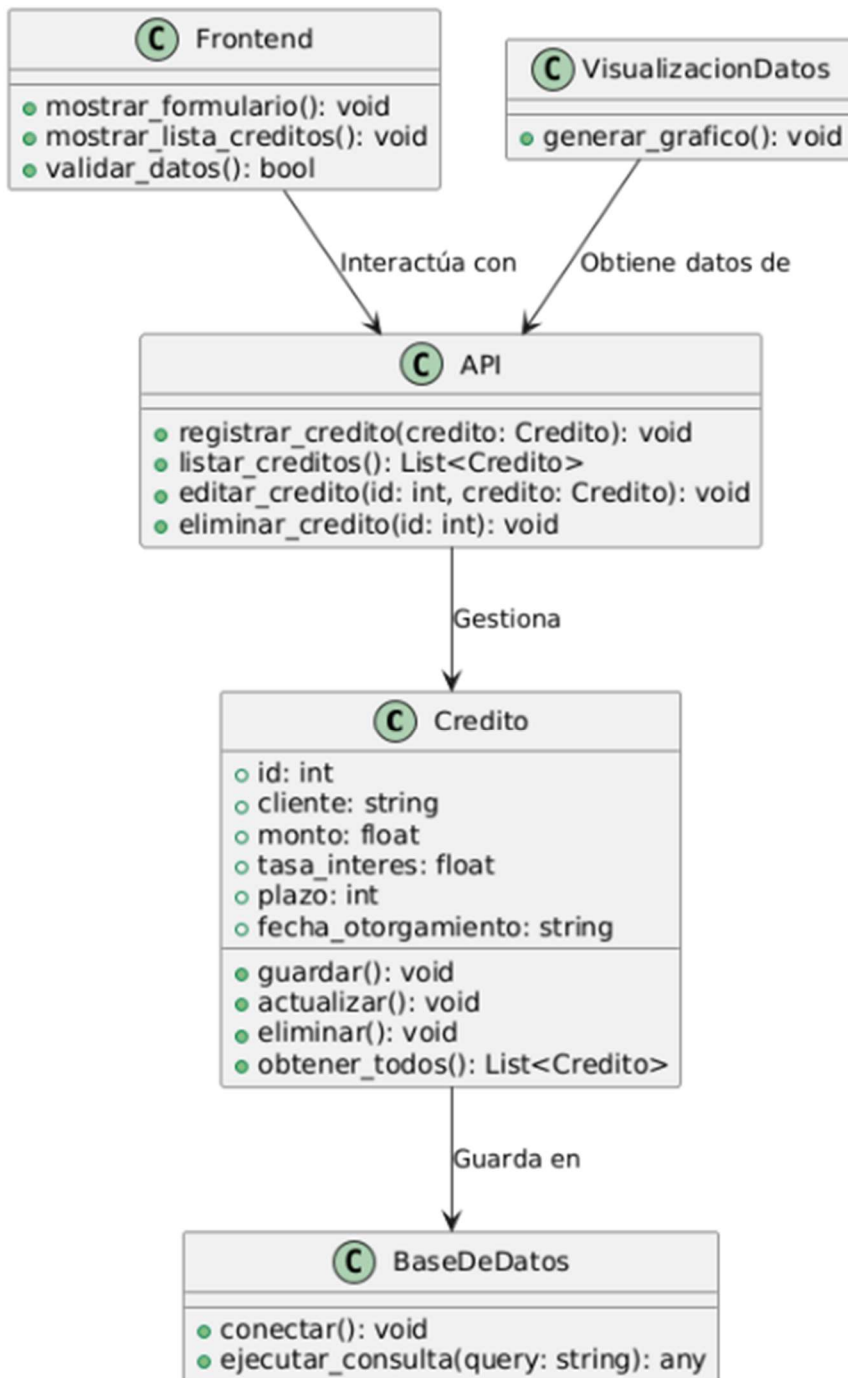


Diagrama de Casos de Uso

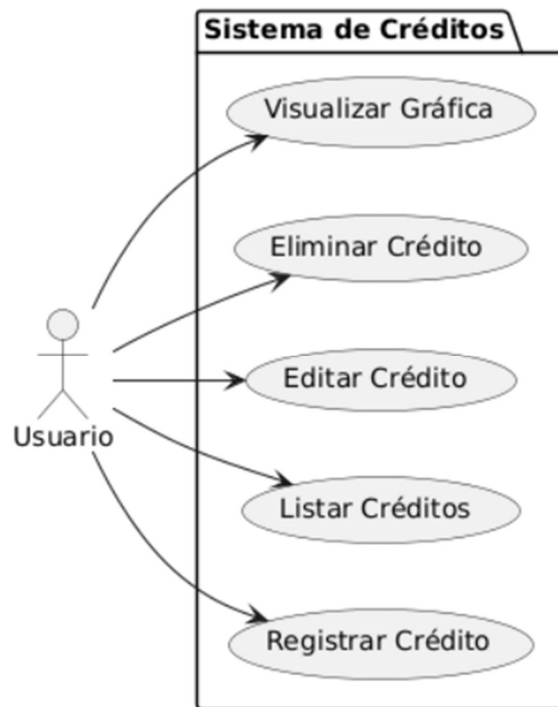


Diagrama de Objetos

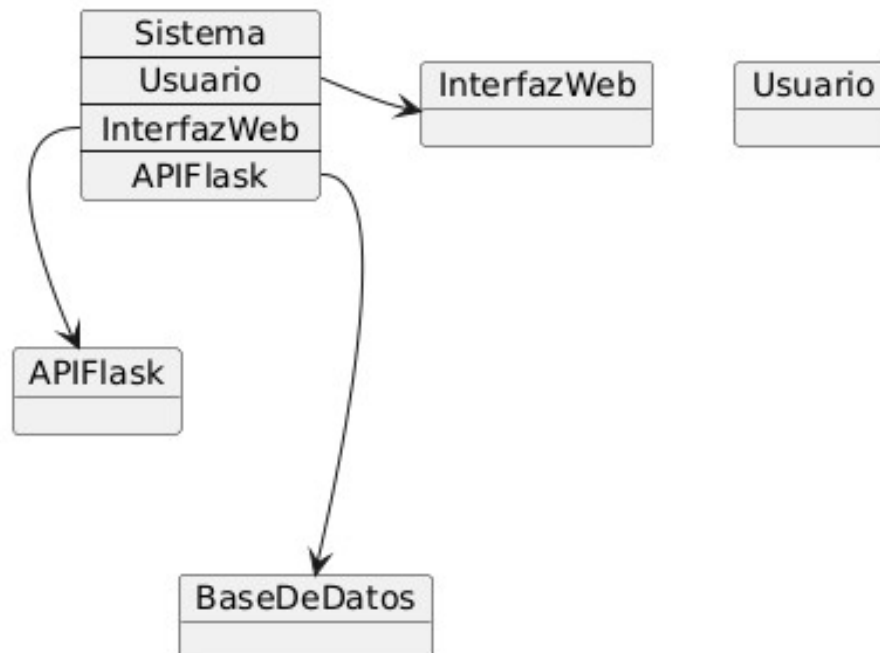


Diagrama de Actividad

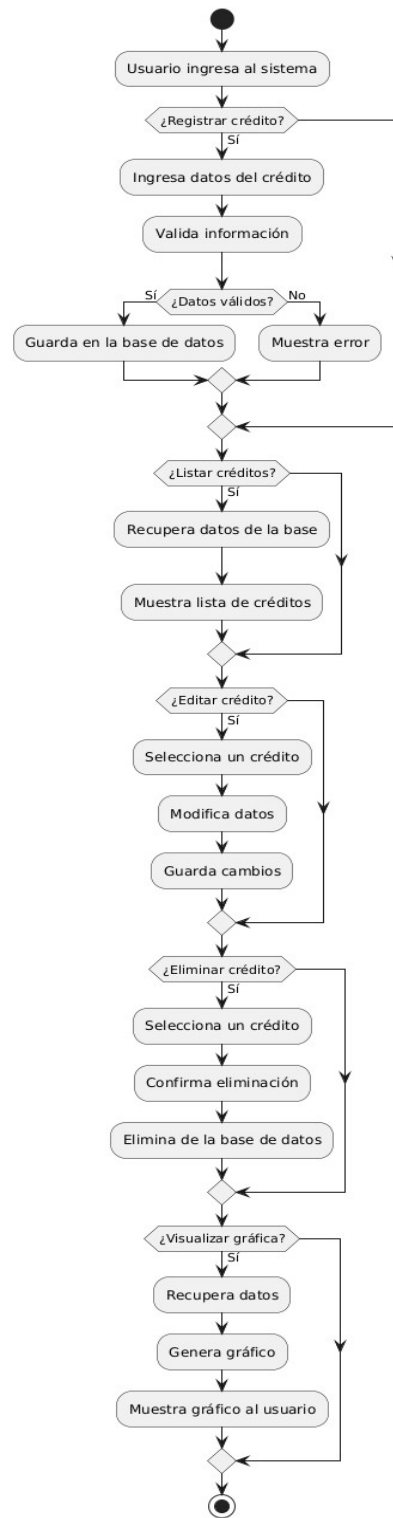


Diagrama de Secuencia

