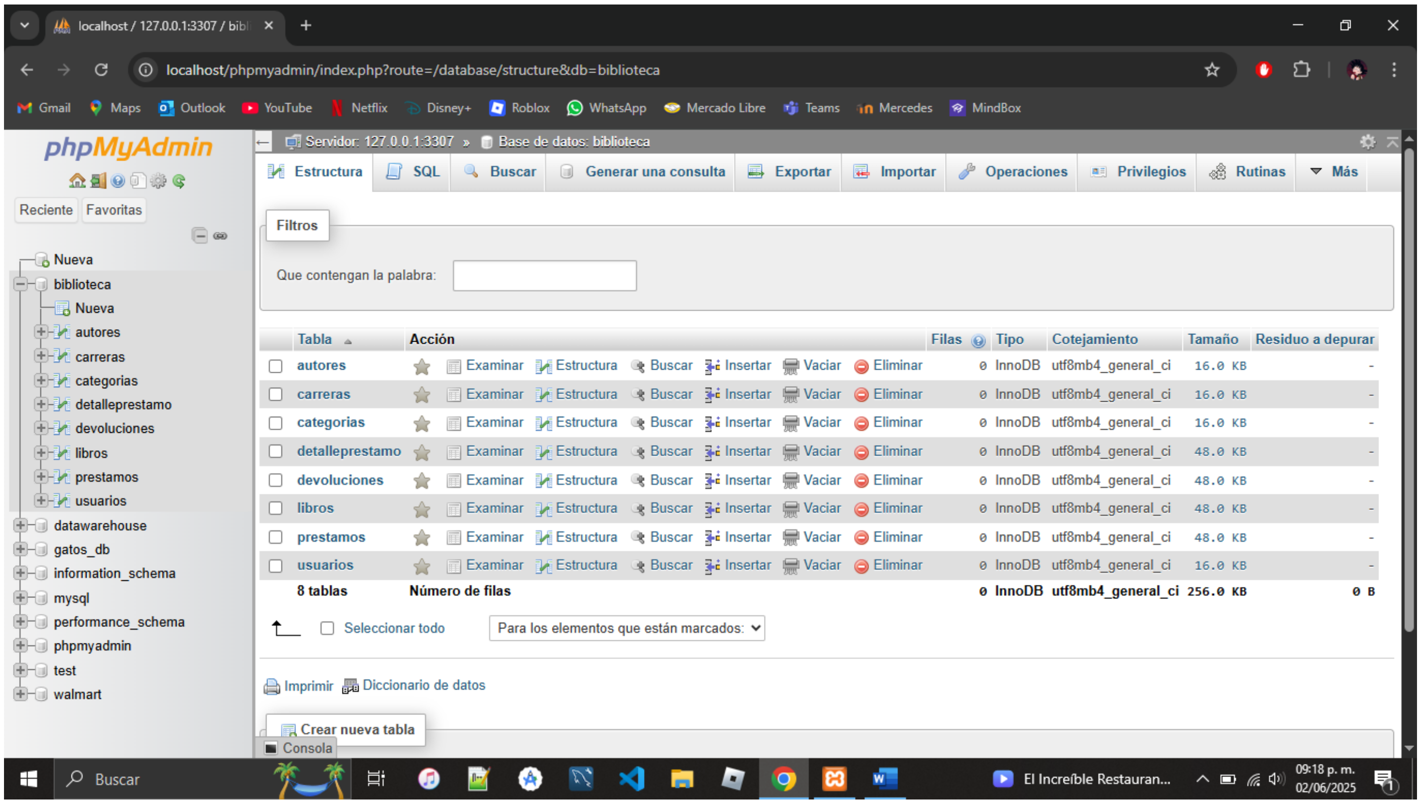
Base de datos “Biblioteca” creada en phpmyadmin desde XAMPP.

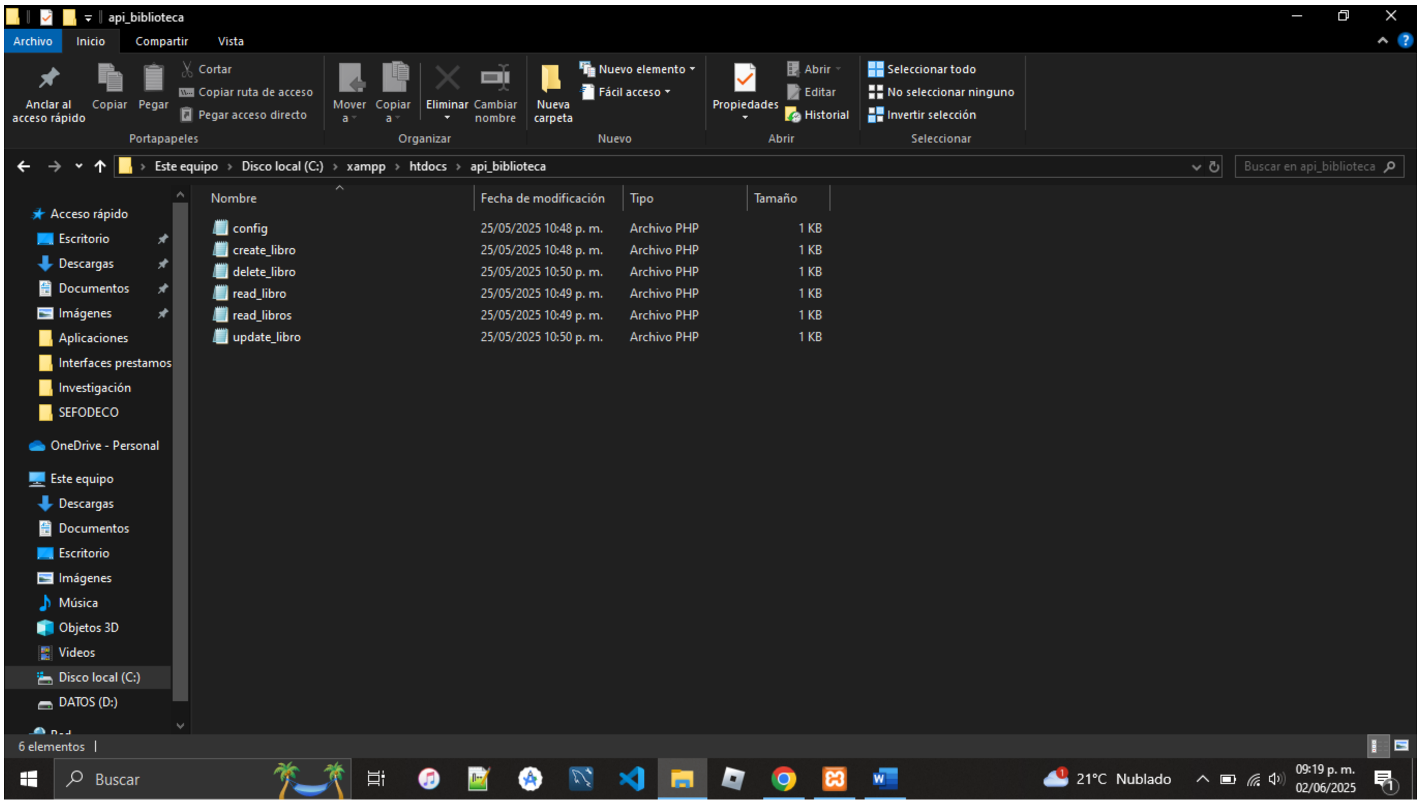


localhost/phpmyadmin/index.php?route=/database/structure&db=biblioteca

Servidor: 127.0.0.1:3307 » Base de datos: biblioteca

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
autores	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	16.0 KB	-
carreras	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	16.0 KB	-
categorias	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	16.0 KB	-
detalleprestamo	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	48.0 KB	-
devoluciones	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	48.0 KB	-
libros	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	48.0 KB	-
prestamos	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	48.0 KB	-
usuarios	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_general_ci	16.0 KB	-
8 tablas	Número de filas	0	InnoDB	utf8mb4_general_ci	256.0 KB	0 B

Archivos .php para crear la conexión con la base de datos y llamar las funciones.



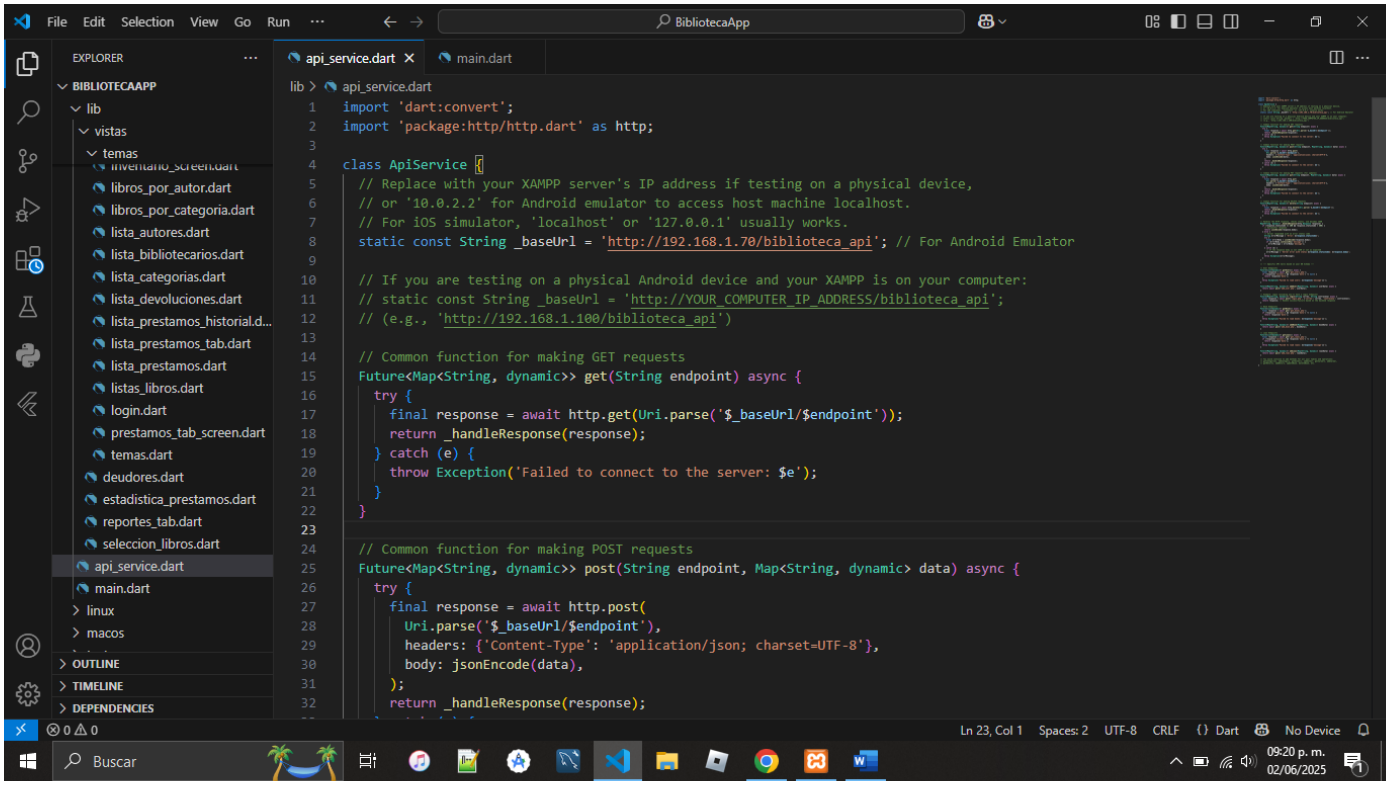
Este equipo › Disco local (C:) › xampp › htdocs › api_biblioteca

Nombre	Fecha de modificación	Tipo	Tamaño
config	25/05/2025 10:48 p. m.	Archivo PHP	1 KB
create_libro	25/05/2025 10:48 p. m.	Archivo PHP	1 KB
delete_libro	25/05/2025 10:50 p. m.	Archivo PHP	1 KB
read_libro	25/05/2025 10:49 p. m.	Archivo PHP	1 KB
read_libros	25/05/2025 10:49 p. m.	Archivo PHP	1 KB
update_libro	25/05/2025 10:50 p. m.	Archivo PHP	1 KB

6 elementos

Evidencias – Kevin Naim Diaz Memije

ApiService creada para hacer la conexión entre el programa y la base de datos en XAMPP.



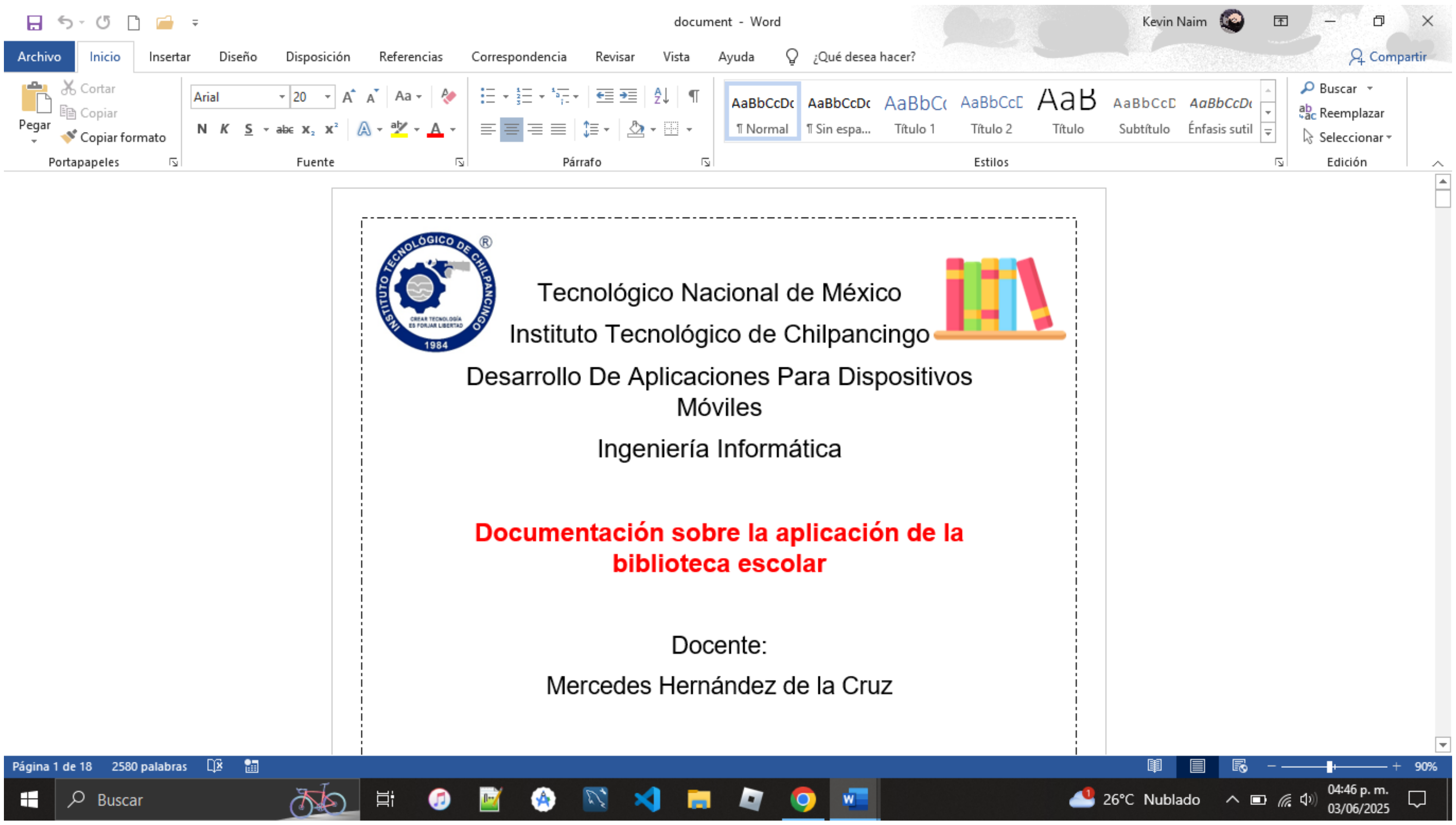
```dart
import 'dart:convert';
import 'package:http/http.dart' as http;

class ApiService {
  // Replace with your XAMPP server's IP address if testing on a physical device,
  // or '10.0.2.2' for Android emulator to access host machine localhost.
  // For iOS simulator, 'localhost' or '127.0.0.1' usually works.
  static const String _baseUrl = 'http://192.168.1.70/biblioteca_api'; // For Android Emulator

  // If you are testing on a physical Android device and your XAMPP is on your computer:
  // static const String _baseUrl = 'http://YOUR_COMPUTER_IP_ADDRESS/biblioteca_api';
  // (e.g., 'http://192.168.1.100/biblioteca_api')

  // Common function for making GET requests
  Future<Map<String, dynamic>> get(String endpoint) async {
    try {
      final response = await http.get(Uri.parse('$_baseUrl/$endpoint'));
      return _handleResponse(response);
    } catch (e) {
      throw Exception('Failed to connect to the server: $e');
    }
  }

  // Common function for making POST requests
  Future<Map<String, dynamic>> post(String endpoint, Map<String, dynamic> data) async {
    try {
      final response = await http.post(
        Uri.parse('$_baseUrl/$endpoint'),
        headers: {'Content-Type': 'application/json; charset=UTF-8'},
        body: jsonEncode(data),
      );
      return _handleResponse(response);
```

Código en cuestión:

```dart
import 'dart:convert';

import 'package:http/http.dart' as http;


class ApiService {

  // Replace with your XAMPP server's IP address if testing on a physical device,

  // or '10.0.2.2' for Android emulator to access host machine localhost.

  // For iOS simulator, 'localhost' or '127.0.0.1' usually works.

  static const String _baseUrl = 'http://192.168.1.70/biblioteca_api'; // For
Android Emulator


  // If you are testing on a physical Android device and your XAMPP is on your
computer:

  // static const String _baseUrl =
'http://YOUR_COMPUTER_IP_ADDRESS/biblioteca_api';
```

```
  // (e.g., 'http://192.168.1.100/biblioteca_api')


  // Common function for making GET requests

  Future<Map<String, dynamic>> get(String endpoint) async {

    try {

      final response = await http.get(Uri.parse('$_baseUrl/$endpoint'));

      return _handleResponse(response);

    } catch (e) {

      throw Exception('Failed to connect to the server: $e');

    }

  }


  // Common function for making POST requests

  Future<Map<String, dynamic>> post(String endpoint, Map<String, dynamic> data)
async {

    try {

      final response = await http.post(

        Uri.parse('$_baseUrl/$endpoint'),

        headers: {'Content-Type': 'application/json; charset=UTF-8'},

        body: jsonEncode(data),

      );

      return _handleResponse(response);

    } catch (e) {

      throw Exception('Failed to connect to the server: $e');

    }

  }


  // Common function for making PUT requests (for updates)

  Future<Map<String, dynamic>> put(String endpoint, Map<String, dynamic> data)
async {

    try {

      final response = await http.put(

        Uri.parse('$_baseUrl/$endpoint'),

        headers: {'Content-Type': 'application/json; charset=UTF-8'},
        body: jsonEncode(data),
      );
      return _handleResponse(response);
    } catch (e) {
      throw Exception('Failed to connect to the server: $e');
    }
  }

  // Common function for making DELETE requests
  Future<Map<String, dynamic>> delete(String endpoint) async {
    try {
      final response = await http.delete(Uri.parse('$_baseUrl/$endpoint'));
      return _handleResponse(response);
    } catch (e) {
      throw Exception('Failed to connect to the server: $e');
    }
  }

  // Handles the HTTP response, checks status, and decodes JSON
  Map<String, dynamic> _handleResponse(http.Response response) {
    if (response.statusCode >= 200 && response.statusCode < 300) {
      // Successful response
      return jsonDecode(response.body);
    } else {
      // Server responded with an error status code
      String errorMessage = 'Error: ${response.statusCode}';
      try {
        final errorBody = jsonDecode(response.body);
        if (errorBody.containsKey('message')) {
          errorMessage = errorBody['message'];
        }
      } catch (e) {
        // If the response body is not JSON or not as expected
        errorMessage = 'Server error with status ${response.statusCode}: 
${response.body}';
      }
      throw Exception(errorMessage);
    }
  }

  // --- Specific API Calls based on your DB Schema ---

  // User Endpoints
  Future<List<dynamic>> getUsers() async {
    final response = await get('get_users.php');
    if (response['success'] && response['data'] != null) {
      return response['data'];
    }
    throw Exception('Failed to load users: ${response['message']}');
  }

  Future<Map<String, dynamic>> addUser(Map<String, dynamic> userData) async {
    return await post('add_user.php', userData);
  }

  // Example: Login (assuming you'll have a login endpoint)
  Future<Map<String, dynamic>> login(String correo, String contrasena) async {
    final response = await post('login.php', {'correo': correo, 'contrasena': 
contrasena});
    return response; // Handle success/failure based on the backend response
  }

  // Book Endpoints
  Future<List<dynamic>> getBooks() async {
    final response = await get('get_books.php');
    if (response['success'] && response['data'] != null) {
      return response['data'];
    }
    throw Exception('Failed to load books: ${response['message']}');
  }

  Future<Map<String, dynamic>> addBook(Map<String, dynamic> bookData) async {
    return await post('add_book.php', bookData);
  }

  // Loan Endpoints
  Future<List<dynamic>> getLoans() async {
    final response = await get('get_loans.php');
    if (response['success'] && response['data'] != null) {
      return response['data'];
    }
    throw Exception('Failed to load loans: ${response['message']}');
  }

  Future<Map<String, dynamic>> addLoan(Map<String, dynamic> loanData) async {
    return await post('add_loan.php', loanData);
  }

  // You would continue to add methods for all your tables and operations:
  // getCategories, addCategory, getAuthors, addAuthor, getCareers, addCareer,
  // getReturns, addReturn, updateBook, deleteBook, etc.
}
```

Creación de la documentación de la aplicación:



Tecnológico Nacional de México

Instituto Tecnológico de Chilpancingo

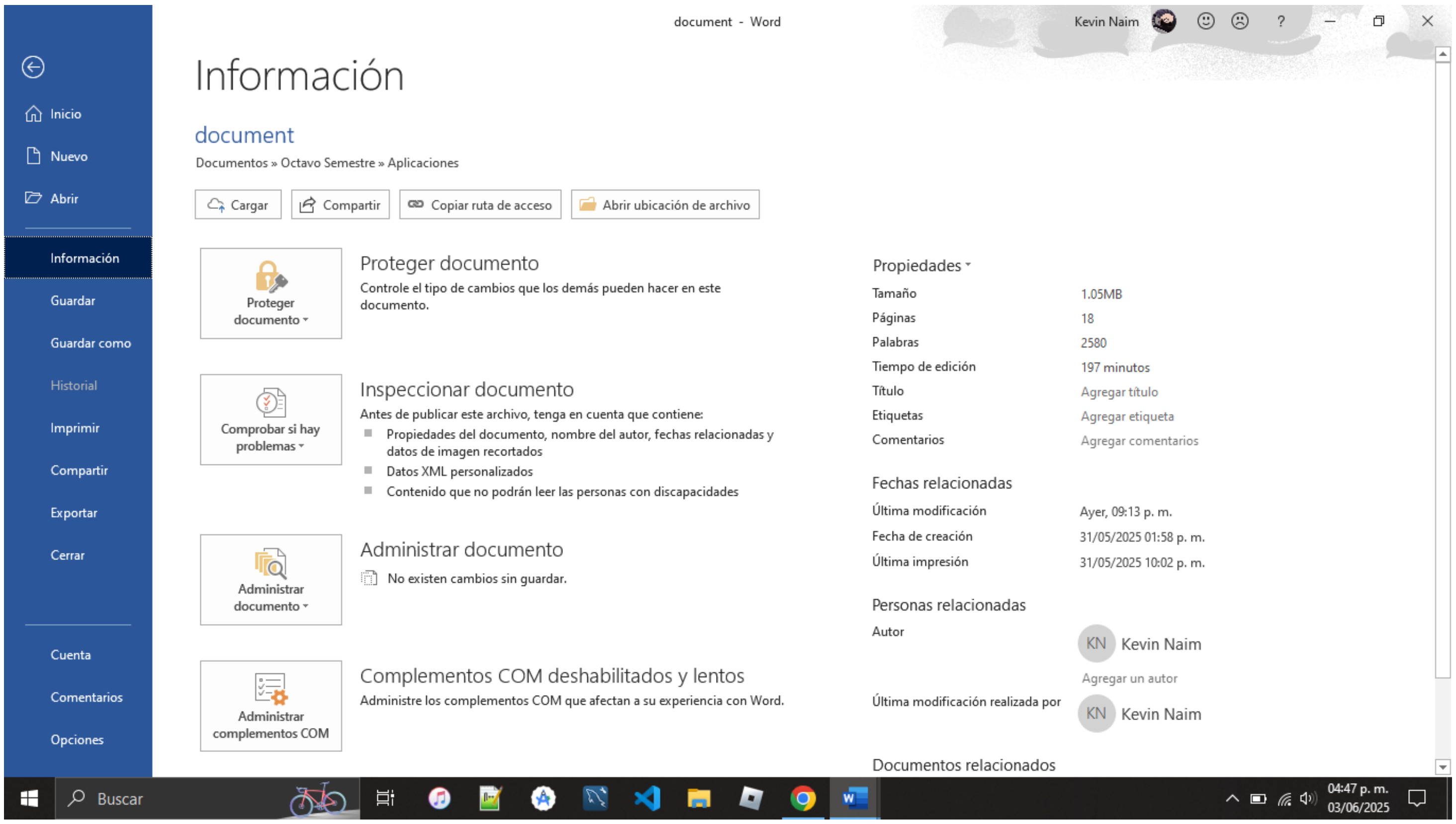
Desarrollo De Aplicaciones Para Dispositivos Móviles

Ingeniería Informática

Documentación sobre la aplicación de la biblioteca escolar

Docente:

Mercedes Hernández de la Cruz



Información

document

Documentos » Octavo Semestre » Aplicaciones

Proteger documento

Controle el tipo de cambios que los demás pueden hacer en este documento.

Inspeccionar documento

Antes de publicar este archivo, tenga en cuenta que contiene:

- Propiedades del documento, nombre del autor, fechas relacionadas y datos de imagen recortados
- Datos XML personalizados
- Contenido que no podrán leer las personas con discapacidades

Administrar documento

No existen cambios sin guardar.

Complementos COM deshabilitados y lentos

Administre los complementos COM que afectan a su experiencia con Word.

Propiedades

Tamaño	1.05MB
Páginas	18
Palabras	2580
Tiempo de edición	197 minutos
Título	Agregar título
Etiquetas	Agregar etiqueta
Comentarios	Agregar comentarios

Fechas relacionadas

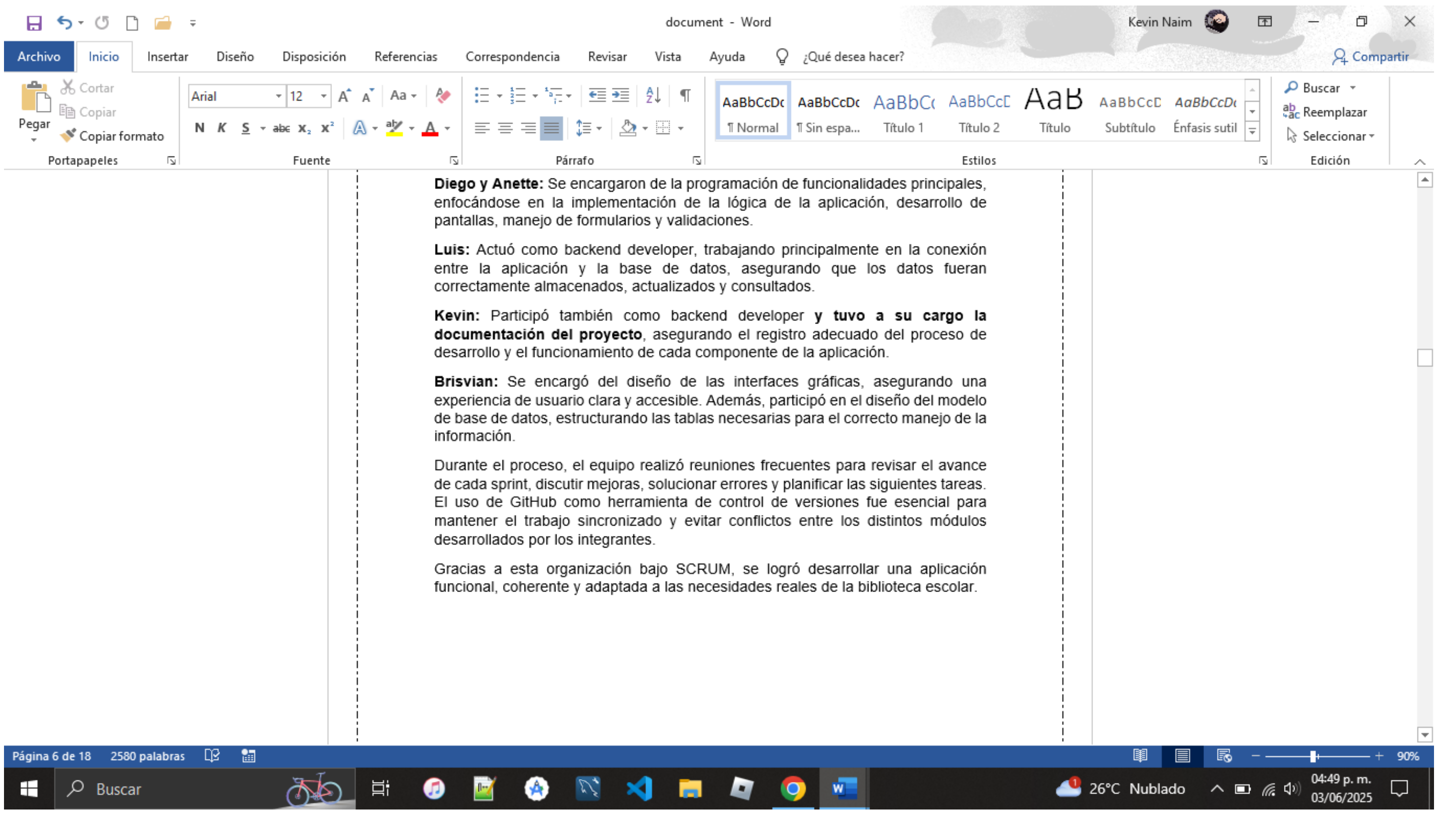
Última modificación	Ayer, 09:13 p. m.
Fecha de creación	31/05/2025 01:58 p. m.
Última impresión	31/05/2025 10:02 p. m.

Personas relacionadas

Autor	KN Kevin Naim
	Agregar un autor
Última modificación realizada por	KN Kevin Naim

Documentos relacionados

Evidencias – Kevin Naim Diaz Memije



Diego y Anette: Se encargaron de la programación de funcionalidades principales, enfocándose en la implementación de la lógica de la aplicación, desarrollo de pantallas, manejo de formularios y validaciones.

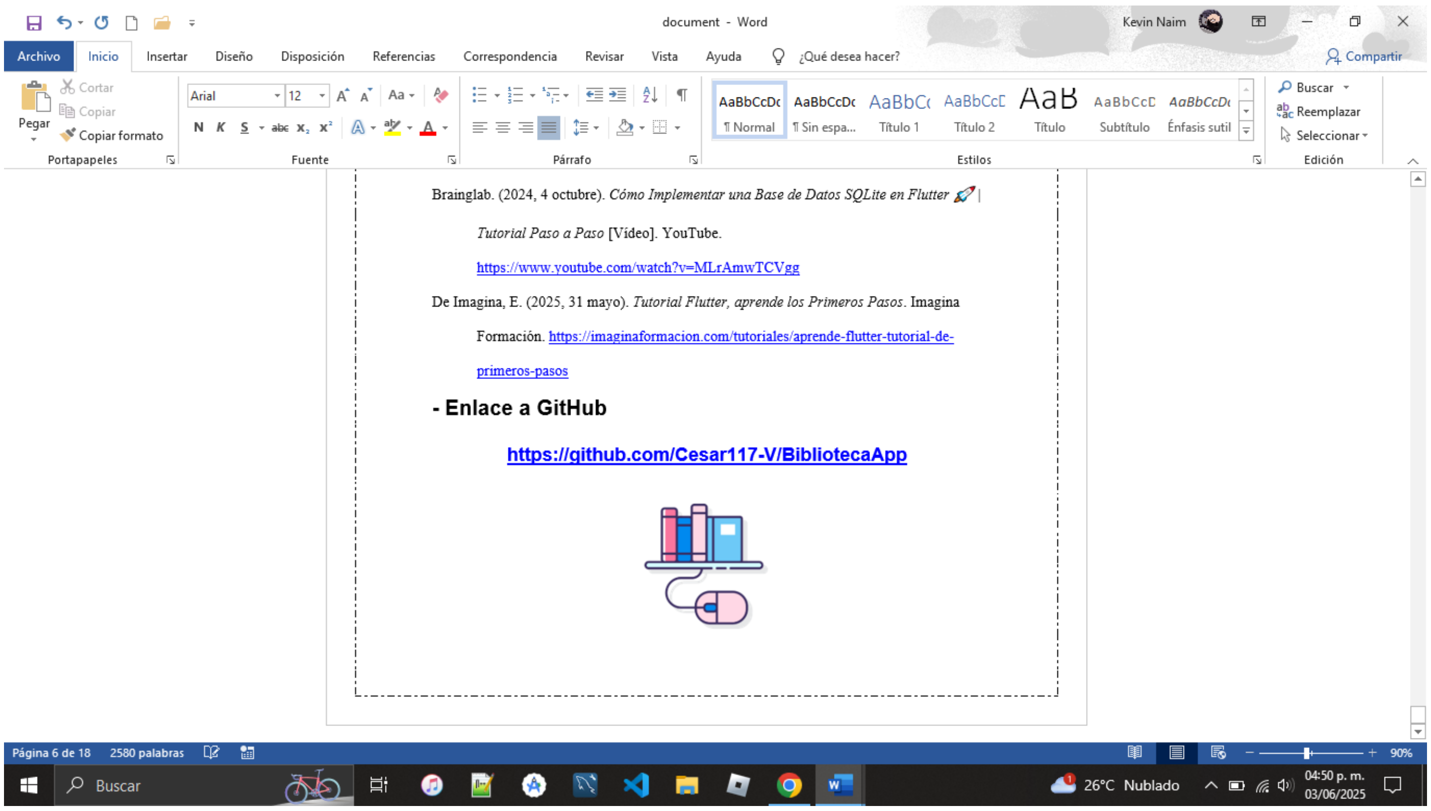
Luis: Actuó como backend developer, trabajando principalmente en la conexión entre la aplicación y la base de datos, asegurando que los datos fueran correctamente almacenados, actualizados y consultados.

Kevin: Participó también como backend developer **y tuvo a su cargo la documentación del proyecto**, asegurando el registro adecuado del proceso de desarrollo y el funcionamiento de cada componente de la aplicación.

Brisvian: Se encargó del diseño de las interfaces gráficas, asegurando una experiencia de usuario clara y accesible. Además, participó en el diseño del modelo de base de datos, estructurando las tablas necesarias para el correcto manejo de la información.

Durante el proceso, el equipo realizó reuniones frecuentes para revisar el avance de cada sprint, discutir mejoras, solucionar errores y planificar las siguientes tareas. El uso de GitHub como herramienta de control de versiones fue esencial para mantener el trabajo sincronizado y evitar conflictos entre los distintos módulos desarrollados por los integrantes.

Gracias a esta organización bajo SCRUM, se logró desarrollar una aplicación funcional, coherente y adaptada a las necesidades reales de la biblioteca escolar.



Brainglab. (2024, 4 octubre). *Cómo Implementar una Base de Datos SQLite en Flutter* 🚀 | *Tutorial Paso a Paso* [Video]. YouTube. https://www.youtube.com/watch?v=MLrAmwTCVgg

De Imagina, E. (2025, 31 mayo). *Tutorial Flutter, aprende los Primeros Pasos*. Imagina Formación. https://imaginaformacion.com/tutoriales/aprende-flutter-tutorial-de-primeros-pasos

- Enlace a GitHub

https://github.com/Cesar117-V/BibliotecaApp

Evidencias – Kevin Naim Diaz Memije