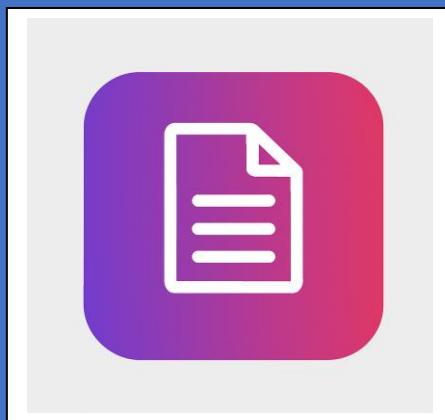


DOCUMENTE

Gestiona inteligentemente tus documentos con nuestro asistente de IA.



Heily Madelai Ajila
Tandazo

2 DAM

Índice del Proyecto

1. Introducción del Problema

- 1.1 Contexto actual de la gestión documental
- 1.2 Problemática identificada
- 1.3 Necesidades del mercado
- 1.4 Propuesta de solución

2. Frontend

- 2.1 Tecnologías y herramientas
- 2.2 Arquitectura de la aplicación
- 2.3 Estructura del proyecto
- 2.4 Pantallas implementadas
- 2.5 Diseño y experiencia de usuario
- 2.6 Funcionalidades por rol
- 2.7 Navegación adaptativa

3. Backend

3.1 Arquitectura General

- 3.1.1 Patrón de diseño implementado (Clean Architecture)
- 3.1.2 Framework principal (FastAPI)
- 3.1.3 Estructura de capas del proyecto
- 3.1.4 Principios SOLID aplicados

3.2 Tecnologías y Dependencias

- 3.2.1 Stack tecnológico principal
- 3.2.2 Gestión de dependencias (requirements.txt)
- 3.2.3 Entorno virtual y activación
- 3.2.4 Variables de entorno (.env)

3.3 Capa de API (src/api)

- 3.3.1 Estructura de rutas (routes.py)
- 3.3.2 Endpoints por dominio
 - 3.3.2.1 Gestión de usuarios (users.py)
 - 3.3.2.2 Gestión de documentos (documents.py)
 - 3.3.2.3 Sistema de chat (chat.py)
- 3.3.3 Dependencias compartidas (dependencies.py)
- 3.3.4 Middleware y CORS

3.4 Modelos de Datos (src/models)

- 3.4.1 Modelos de dominio (domain)
- 3.4.2 Esquemas de validación (schemas)

- 3.4.2.1 Usuario (user.py)
- 3.4.2.2 Documentos (document.py)
- 3.4.2.3 Chat y mensajes (chat.py)
- 3.4.3 Validación con Pydantic

3.5 Capa de Servicios (src/services)

- 3.5.1 Lógica de negocio centralizada
- 3.5.2 Servicio de autenticación (auth_service.py)
- 3.5.3 Servicio de usuarios (user_service.py)
- 3.5.4 Servicio de documentos (document_service.py)
- 3.5.5 Servicio de chat y RAG (chat_service.py)

3.6 Capa de Persistencia (src/repositories)

- 3.6.1 Patrón Repository implementado
- 3.6.2 Repositorio de usuarios (user_repository.py)
- 3.6.3 Repositorio de documentos (document_repository.py)
- 3.6.4 Repositorio de chats (chat_repository.py)
- 3.6.5 Repositorio de mensajes (message_repository.py)

3.7 Configuración (src/config)

- 3.7.1 Configuraciones centralizadas (settings.py)
- 3.7.2 Configuración de base de datos (database.py)
- 3.7.3 Variables de entorno y parámetros

3.8 Utilidades (src/utils)

- 3.8.1 Connector de IA (ai_connector.py)
 - 3.8.1.1 Integración con Gemini API
 - 3.8.1.2 Patrón Singleton aplicado
 - 3.8.1.3 Generación de embeddings
 - 3.8.1.4 Chat completion y RAG
- 3.8.2 Connector de ChromaDB (chromadb_connector.py)
 - 3.8.2.1 Base de datos vectorial
 - 3.8.2.2 Operaciones CRUD vectoriales
 - 3.8.2.3 Búsqueda semántica
 - 3.8.2.4 Singleton y lazy initialization
- 3.8.3 Utilidades de password (password_utils.py)
- 3.8.4 Utilidades de administración (admin_utils.py)

3.9 Gestión de Excepciones (src/core)

- 3.9.1 Excepciones personalizadas (exceptions.py)
- 3.9.2 Manejo centralizado de errores
- 3.9.3 Logging y trazabilidad

3.10 Sistema RAG (Retrieval-Augmented Generation)

- 3.10.1 Arquitectura RAG implementada
- 3.10.2 Procesamiento de documentos grandes
- 3.10.3 Chunking y embeddings
- 3.10.4 Búsqueda semántica en contexto
- 3.10.5 Generación de respuestas enriquecidas

3.11 Infraestructura y Deployment

- 3.11.1 Dockerización (docker-compose.yaml)
- 3.11.2 Configuración de ChromaDB container
- 3.11.3 Networking y volúmenes
- 3.11.4 Healthchecks y monitoreo

3.12 Base de Datos

- 3.12.1 Supabase como backend principal
- 3.12.2 ChromaDB para datos vectoriales
- 3.12.3 Relaciones entre entidades
- 3.12.4 Migraciones y esquemas

3.13 Autenticación y Seguridad

- 3.13.1 JWT para autenticación
- 3.13.2 Hashing de contraseñas (bcrypt)
- 3.13.3 Middleware de seguridad
- 3.13.4 Validación de tokens

3.14 Testing

- 3.14.1 Estructura de pruebas (tests/)
- 3.14.2 Pytest como framework
- 3.14.3 Pruebas unitarias e integración
- 3.14.4 Cobertura de código

3.15 Punto de entrada (src/main.py)

- 3.15.1 Configuración de FastAPI
- 3.15.2 Middleware de CORS
- 3.15.3 Eventos de startup
- 3.15.4 Configuración de servidor Uvicorn

4. Integración Frontend-Backend

- 4.1 Comunicación API REST
- 4.2 Manejo de autenticación
- 4.3 Gestión de estados
- 4.4 Optimización de peticiones

5. Consideraciones Futuras

- 5.1 Optimizaciones pendientes
- 5.2 Nuevas funcionalidades
- 5.3 Mejoras de rendimiento

6. Bibliografía

1. Introducción del Problema

1.1. Contexto actual de la gestión documental

En la actualidad, el volumen de información digital crece a un ritmo sin precedentes. Organizaciones de todos los sectores —empresas, administraciones públicas, instituciones educativas, entre otras— generan y almacenan diariamente una gran cantidad de documentos digitales que contienen información estratégica, operativa y legal. Esta proliferación de contenidos ha convertido la gestión documental en una actividad crítica, pero también en un desafío técnico y organizativo.

Los métodos tradicionales de gestión documental se basan comúnmente en estructuras de carpetas jerárquicas y sistemas de nomenclatura rígidos que requieren intervención humana para su mantenimiento. Aunque estas prácticas fueron efectivas en contextos de menor escala, hoy resultan ineficientes ante repositorios con miles de documentos en diversos formatos, como PDF, Word, Excel o texto plano.

A continuación, se enumeran las principales deficiencias de los enfoques tradicionales:

- Búsquedas manuales ineficientes: Los usuarios deben recordar nombres exactos o rutas de archivo, lo cual genera frustración y pérdida de tiempo.
- Estructuras jerárquicas rígidas: Las carpetas anidadas no permiten relaciones dinámicas entre documentos relacionados, dificultando el acceso contextual.
- Búsquedas por palabra clave sin semántica: No se tiene en cuenta el significado de las palabras ni su relación con otras, lo que conduce a resultados irrelevantes.
- Dificultad de escaneo visual de documentos extensos: Extraer un dato específico en un documento largo sin herramientas de búsqueda avanzadas puede tomar varios minutos o incluso horas.
- Carencia de interacción conversacional: Los usuarios no pueden "preguntar" directamente al sistema sobre el contenido, como lo harían con un colega o experto.

Estas limitaciones no solo afectan la productividad, sino que también reducen la capacidad de las organizaciones para tomar decisiones informadas en tiempo real y aprovechar el conocimiento interno acumulado.

1.2. Problemática identificada

El diagnóstico realizado sobre las prácticas actuales de gestión documental ha permitido identificar tres áreas críticas de mejora: accesibilidad, eficiencia y colaboración.

Problemas de accesibilidad

- Interacción no natural: La mayoría de los usuarios no tiene conocimientos técnicos avanzados, lo que hace que las interfaces tradicionales resulten poco intuitivas. El sistema no comprende las intenciones del usuario expresadas en lenguaje natural.
- Información oculta en documentos: Muchos datos relevantes se encuentran "encerrados" en archivos PDF sin estructura semántica clara, dificultando su extracción automática.
- Consultas poco productivas: No es posible realizar preguntas complejas o específicas y obtener respuestas inmediatas y comprensibles, lo que reduce la utilidad de la información almacenada.

Problemas de eficiencia

- Sobrecarga de tiempo en búsquedas: La localización de documentos y la extracción de datos dentro de ellos son procesos lentos y repetitivos.
- Repositorios crecientes sin organización semántica: A medida que crece el volumen de información, la localización manual se vuelve inviable sin una clasificación y búsqueda inteligente.
- Ausencia de inteligencia contextual: El sistema no entiende el contexto de la consulta, por lo que ofrece resultados genéricos, muchas veces irrelevantes.

Problemas de colaboración

- Limitada circulación del conocimiento: Los usuarios no pueden compartir eficazmente sus hallazgos o búsquedas con otros miembros de la organización.
- Sin trazabilidad: No existe un historial de consultas o búsquedas que permita mejorar la experiencia del usuario o aplicar analítica.

- Falta de control centralizado de accesos: Los permisos suelen estar ligados a carpetas o archivos individuales, sin una política coherente ni escalable.

Estos problemas afectan negativamente tanto la experiencia del usuario como el rendimiento general de los flujos de trabajo documentales.

1.3. Necesidades del mercado

Ante estas limitaciones, el mercado actual demanda soluciones tecnológicas que vayan más allá de los gestores documentales convencionales. Las nuevas exigencias se centran en:

Inteligencia Artificial Aplicada

- Procesamiento de Lenguaje Natural (NLP): Permite interpretar consultas en lenguaje natural, extrayendo la intención del usuario y generando respuestas pertinentes.
- Comprensión semántica: Se requiere que el sistema entienda no solo las palabras, sino su significado en contexto, relacionando conceptos sinónimos, acrónimos o expresiones complejas.
- Generación de respuestas contextualizadas (RAG): El modelo debe combinar recuperación de información relevante con modelos generativos (como LLMs) para ofrecer respuestas directas, precisas y útiles.

Gestión inteligente de documentos

- Indexación automática: Todo documento ingresado al sistema debe ser analizado, fragmentado, vectorizado y almacenado para búsqueda eficiente.
- Soporte de múltiples formatos: No debe haber limitaciones respecto al tipo de archivo, garantizando así una integración completa.
- Chunking inteligente: Es fundamental dividir los documentos en fragmentos significativos y coherentes para una comprensión más profunda por parte del sistema.

Experiencia de usuario moderna

- Interfaz conversacional: Se prefiere una experiencia tipo chatbot que permita una interacción fluida e intuitiva, incluso para usuarios no técnicos.

- Acceso multiplataforma: La herramienta debe estar disponible en múltiples dispositivos (móvil, escritorio, web) con una experiencia coherente.
- Navegación adaptativa: Las funcionalidades deben variar según el rol del usuario, optimizando así el uso del sistema sin sobrecargar la interfaz.

Seguridad y control de acceso

- Gestión granular de permisos: Los documentos deben tener niveles de acceso configurables por usuario, grupo o rol.
- Trazabilidad completa: El sistema debe registrar cada acceso, consulta o modificación para fines de auditoría.
- Compartición segura: Debe ser posible compartir documentos o fragmentos con otros usuarios sin comprometer la seguridad ni la integridad del contenido.

Estas necesidades marcan el camino hacia plataformas documentales más inteligentes, seguras y centradas en el usuario.

1.4. Propuesta de solución: *DocuMente*

Para dar respuesta a estos retos, se plantea el desarrollo de *DocuMente*, una plataforma de gestión documental impulsada por Inteligencia Artificial, que incorpora lo último en tecnología backend, frontend y diseño de experiencia de usuario.

Backend inteligente (FastAPI + Python)

- Sistema RAG (Retrieval-Augmented Generation): Utiliza modelos de NLP junto con una base de datos vectorial (ChromaDB) para recuperar fragmentos relevantes y generar respuestas precisas mediante Gemini AI.
- Arquitectura basada en el patrón Repository: Se favorece la modularidad, testabilidad y mantenimiento del código.
- Procesamiento asíncrono: Permite cargar y analizar documentos pesados (>50MB) sin afectar el rendimiento global del sistema.
- Gestión optimizada de memoria: El procesamiento de documentos se realiza en fragmentos (chunks), evitando saturar la memoria y permitiendo operaciones en paralelo.

Frontend multiplataforma (Flutter)

- Aplicación nativa multiplataforma: Compatible con Android, iOS, macOS, Windows, Linux y navegadores web, garantizando disponibilidad universal.

- MentIA, el asistente conversacional: Interfaz en forma de chat con capacidades semánticas que permite interactuar con los documentos como si se hablara con un experto.
- Gestión documental completa: Subida, visualización, búsqueda, etiquetado, compartición y edición de permisos desde una única interfaz unificada.

Características diferenciadoras

- RAG inteligente: Fusiona búsqueda semántica y generación de texto, proporcionando respuestas más completas que una búsqueda estándar o un modelo generativo aislado.
- Optimización del procesamiento: Gracias a técnicas como lazy initialization y patrones como Singleton, el sistema es escalable y eficiente.
- Seguridad robusta: Se implementa autenticación con JWT, cifrado de contraseñas (bcrypt) y control de acceso granular con trazabilidad completa.

Con esta solución, *DocuMente* no solo pretende resolver los problemas existentes, sino redefinir la forma en que los usuarios interactúan con la información documental, transformando el conocimiento en un recurso verdaderamente accesible e inteligente.

2. Frontend

2.1 Tecnologías y Herramientas - Sistema Frontend DocuMente

El desarrollo del frontend de DocuMente se ha realizado siguiendo un enfoque tecnológico moderno y orientado a la eficiencia, priorizando la experiencia del usuario y la compatibilidad multiplataforma. La selección de cada tecnología ha sido resultado de un análisis exhaustivo de las alternativas disponibles en el mercado, considerando factores como rendimiento, curva de aprendizaje, soporte comunitario y proyección a futuro.

2.1.2 Framework Principal: Flutter

2.1.2.1 Justificación de la Elección

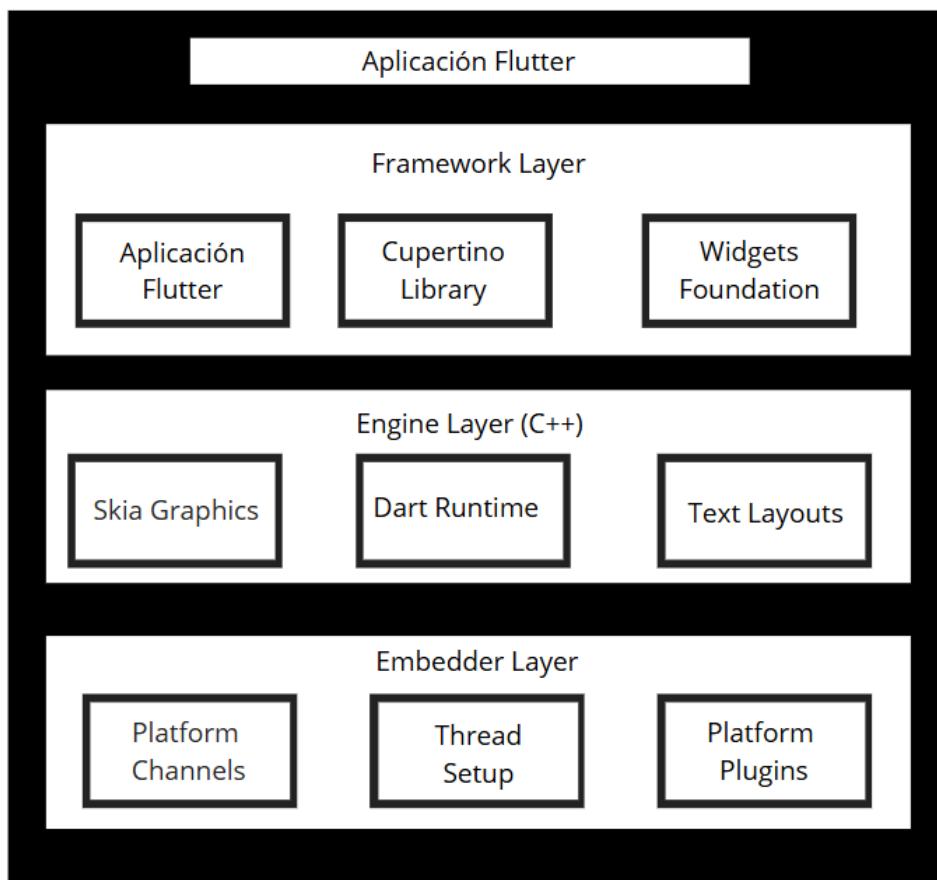
Flutter fue seleccionado como framework principal después de evaluar múltiples alternativas. Análisis Comparativo de Frameworks:

Framework	Rendimiento	Desarrollo	Comunidad	Multiplataforma	Curva Aprendizaje

Flutter	95/100	90/100	85/100	95/100	75/100
React Native	80/100	85/100	95/100	85/100	80/100
Ionic	70/100	80/100	80/100	90/100	85/100
Xamarin	85/100	70/100	70/100	90/100	65/100
Native	100/100	60/100	90/100	0/100	50/100

2.1.2.2 Arquitectura Técnica de Flutter

Flutter opera mediante un motor de renderizado propio basado en Skia, lo que le permite:



2.1.3 Lenguaje de Programación: Dart

2.1.3.1 Características del Lenguaje Utilizadas

Sound Null Safety

// Implementación de null safety en modelos

```

class User {
  final String id; // Non-nullable
  final String? avatar; // Nullable
  final DateTime createdAt; // Non-nullable

  // Constructor con validaciones
  User({
    required this.id,
    this.avatar,
    DateTime? createdAt,
  }) : createdAt = createdAt ?? DateTime.now();

  // Null-aware operators
  String get displayAvatar => avatar ?? 'assets/default_avatar.png';
}

```

Pattern Matching y Sealed Classes

```

// Gestión de estados con sealed classes
sealed class AuthState {}

class AuthInitial extends AuthState {}

class AuthLoading extends AuthState {}

class AuthAuthenticated extends AuthState {
  final User user;
  AuthAuthenticated(this.user);
}

class AuthError extends AuthState {
  final String message;
  AuthError(this.message);
}

// Pattern matching en UI
Widget buildAuthUI(AuthState state) {
  return switch (state) {
    AuthInitial() => const LoginScreen(),
    AuthLoading() => const LoadingScreen(),
    AuthAuthenticated(:final user) => HomeScreen(user: user),
    AuthError(:final message) => ErrorScreen(message: message),
  };
}

```

2.1.3.2 Análisis de Performance del Lenguaje

```
// Benchmarks de operaciones comunes
class PerformanceMetrics {
    // Búsqueda en lista: O(n)
    static Document? findDocument(List<Document> docs, String id) {
        return docs.firstWhere(
            (doc) => doc.id == id,
            orElse: () => null,
        );
    }

    // Búsqueda optimizada con Map: O(1)
    static final Map<String, Document> _documentCache = {};

    static Document? findDocumentOptimized(String id) {
        return _documentCache[id];
    }

}
```

2.1.4 Sistema de Gestión de Estado: Provider

2.1.4.1 Arquitectura de Provider

```
// Implementación del patrón Observer con Provider
class DocumentProvider extends ChangeNotifier {
    // Estado privado
    List<Document> _documents = [];
    bool _isLoading = false;
    String? _error;

    // Getters públicos (inmutables)
    List<Document> get documents => List.unmodifiable(_documents);
    bool get isLoading => _isLoading;
    String? get error => _error;

    // Métodos de actualización con notificación
    Future<void> loadDocuments() async {
        _ setLoading(true);
        try {
            final docs = await _repository.fetchDocuments();
            _documents = docs;
            _error = null;
        } catch (e) {
```

```

        _error = e.toString();
    } finally {
        _ setLoading(false);
    }
}

void _ setLoading(bool value) {
    _isLoading = value;
    notifyListeners(); // Notifica a todos los widgets suscritos
}
}

```

2.1.4.2 Comparación con Otras Soluciones de Estado

Solución	Complejidad	Performance	Escalabilidad	Curva Aprendizaje	Boilerplate
Provider	Baja	Alta	Media	Baja	Mínimo
Bloc	Alta	Alta	Alta	Alta	Alto
Riverpod	Media	Muy Alta	Alta	Media	Medio
GetX	Baja	Media	Media	Baja	Mínimo
MobX	Media	Alta	Alta	Media	Medio

2.1.5 Dependencias y Librerías

2.1.5.1 Análisis Detallado de Dependencias

dependencies:

 flutter:

 sdk: flutter

Internacionalización

flutter_localizations:

 sdk: flutter

intl: ^0.19.0

Cliente HTTP

http: ^1.1.0

Almacenamiento Local

shared_preferences: ^2.2.0

```
# Gestión de Estado
provider: ^6.1.0

# Multimedia
image_picker: ^1.0.7

# Iconos
cupertino_icons: ^1.0.8

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0
  mockito: ^5.4.0
  test: ^1.24.0
```

2.1.5 Herramientas de Desarrollo

2.1.5.1 IDE y Extensiones

Visual Studio Code

- Configuración optimizada para Flutter:
- { "editor.formatOnSave": true, "editor.codeActionsOnSave": { "source.fixAll": true, "source.organizeImports": true }, "dart.lineLength": 80, "dart.flutterSdkPath": "/usr/local/flutter", "dart.debugExternalLibraries": false, "dart.debugSdkLibraries": false, "[dart]": { "editor.rulers": [80], "editor.selectionHighlight": false, "editor.suggest.snippetsPreventQuickSuggestions": false, "editor.suggestSelection": "first", "editor.tabCompletion": "onlySnippets", "editor.wordBasedSuggestions": false } }

Extensiones Esenciales

1. Dart (Dart-Code.dart-code)
2. Flutter (Dart-Code.flutter)
3. Error Lens (usernamehw.errorlens)
4. Flutter Tree (marcelovelasquez.flutter-tree)
5. Pubspec Assist (jeroen-meijer.pubspec-assist)

2.1.5.2 Herramientas de Análisis y Testing

Flutter Analyze

```
# Configuración de analysis_options.yaml
```

```
include: package:flutter_lints/flutter.yaml

analyzer:
  strong-mode:
    implicit-casts: false
    implicit-dynamic: false

  errors:
    missing_required_param: error
    missing_return: error
    todo: ignore
    deprecated_member_use_from_same_package: ignore

  exclude:
    - "**/*.g.dart"
    - "**/*.freezed.dart"

linter:
  rules:
    - always_declare_return_types
    - always_put_control_body_on_new_line
    - annotate_overrides
    - avoid_bool_literals_in_conditional_expressions
    - avoid_empty_else
    - avoidEscaping_inner_quotes
    - avoid_field_initializers_in_const_classes
    - avoid_function_literals_in_foreach_calls
    - avoid_init_to_null
    - avoid_null_checks_in_equality_operators
    - avoid_relative_lib_imports
    - avoid_renaming_method_parameters
    - avoid_return_types_on_setters
    - avoid_returning_null_for_void
    - avoid_single_cascade_in_expression_statements
    - avoid_slow_async_io
    - avoid_types_as_parameter_names
    - avoid_unnecessary_containers
    - avoid_unused_constructor_parameters
    - avoid_void_async
    - await_only_futures
    - camel_case_extensions
    - camel_case_types
    - cancel_subscriptions
    - cascade_invocations
```

- close_sinks
- comment_references
- constant_identifier_names
- control_flow_in_finally
- curly_braces_in_flow_control_structures
- directives_ordering
- empty_catches
- empty_constructor_bodies
- empty_statements
- exhaustive_cases
- file_names
- flutter_style.todos
- hash_and_equals
- implementation_imports
- library_names
- library_prefixes
- lines_longer_than_80_chars
- list_remove_unrelated_type
- literal_only_boolean_expressions
- no_adjacent_strings_in_list
- no_duplicate_case_values
- non_constant_identifier_names
- null_closures
- omit_local_variable_types
- only_throw_errors
- overridden_fields
- package_api_docs
- package_names
- package_prefixed_library_names
- parameter_assignments
- prefer_adjacent_string_concatenation
- prefer_asserts_in_initializer_lists
- prefer_collection_literals
- prefer_conditional_assignment
- prefer_const_constructors
- prefer_const_constructors_in_immutables
- prefer_const_declarations
- prefer_const_literals_to_create_immutables
- prefer_contains
- prefer_equal_for_default_values
- prefer_final_fields
- prefer_final_in_for_each
- prefer_final_locals
- prefer_for_elements_to_map_fromIterable

- prefer_function_declarations_over_variables
- prefer_generic_function_type_aliases
- prefer_if_elements_to_conditional_expressions
- prefer_if_null_operators
- prefer_initializing_formals
- prefer_inlined_adds
- prefer_interpolation_to_compose_strings
- prefer_is_empty
- prefer_is_not_empty
- prefer_is_not_operator
- prefer_iterable_whereType
- prefer_null_aware_operators
- prefer_single_quotes
- prefer_spread_collections
- prefer_typing_uninitialized_variables
- prefer_void_to_null
- provide_deprecation_message
- recursive_getters
- slash_for_doc_comments
- sort_child_properties_last
- sort_constructors_first
- sort_pub_dependencies
- sort_unnamed_constructors_first
- test_types_in_equals
- throw_in_finally
- type_annotation_public_apis
- type_init_formals
- unawaited_futures
- unnecessary(await_in_return)
- unnecessary_brace_in_string_interps
- unnecessary_const
- unnecessary_getters_setters
- unnecessary_lambdas
- unnecessary_new
- unnecessary_null_aware_assignments
- unnecessary_null_in_if_null_operators
- unnecessary_overrides
- unnecessary_parenthesis
- unnecessary_statements
- unnecessary_string_escapes
- unnecessary_string_interpolations
- unnecessary_this
- unrelated_type_equality_checks
- unsafe_html

- use_full_hex_values_for_flutter_colors
- use_function_type_syntax_for_parameters
- use_rethrow_when_possible
- use_setters_to_change_properties
- use_string_buffers
- use_to_and_as_if_applicable
- valid_regexps
- void_checks

2.2 Arquitectura de la Aplicación - Sistema Frontend DocuMente

2.2.1 Visión General de la Arquitectura

La arquitectura del frontend de DocuMente implementa una variación del patrón Model-View-ViewModel (MVVM) combinada con principios de Clean Architecture, diseñada específicamente para maximizar la separación de responsabilidades, facilitar el testing y permitir la escalabilidad del sistema.

2.2.1.1 Diagrama de Arquitectura General

La arquitectura Clean Architecture para Flutter se organiza en 4 capas principales:

CAPA DE PRESENTACIÓN

Screens (LoginScreen, HomeScreen, ChatScreen, AdminPanel), Widgets (CustomButtons, InputFields, Cards, Dialogs) y Animations (FadeTransitions, SlideTransitions, HeroAnimations, CustomAnimations);

CAPA DE ESTADO

Providers (AuthProvider, DocProvider, ChatProvider, UserProvider), ViewModels (LoginViewModel, DocViewModel, ChatViewModel, AdminViewModel) y Controllers (NavigationController, FormController, AnimationController, ScrollController);

CAPA DE DOMINIO

Use Cases (LoginUseCase, UploadUseCase, ShareUseCase, SearchUseCase), Repositories abstractos (IUserRepo, IDocumentRepo, IChatRepo, IAuthRepo) y Entities (User, Document, Chat, Message);

CAPA DE DATOS

Data Sources (LocalDataSrc, RemoteDataSrc, CacheDataSrc, MockDataSrc), Repositories concretos (UserRepolImpl, DocRepolImpl, ChatRepolImpl, AuthRepolImpl) y Models (UserModel, DocumentModel, ChatModel, ResponseModel), donde el flujo de datos va desde la presentación hacia los datos y viceversa, manteniendo la independencia entre capas.

2.2.2 Principios Arquitectónicos

2.2.2.1 Separación de Responsabilidades (SoC)

```
// CAPA DE PRESENTACIÓN - Solo UI
class DocumentScreen extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Consumer<DocumentViewModel>(
            builder: (context, viewModel, child) {
                if (viewModel.isLoading) {
                    return const LoadingWidget();
                }

                if (viewModel.hasError) {
                    return ErrorWidget(message: viewModel.error!);
                }

                return DocumentListWidget(
                    documents: viewModel.documents,
                    onRefresh: viewModel.refresh,
                    onDelete: viewModel.deleteDocument,
                );
            },
        );
    }
}

// CAPA DE ESTADO - Lógica de presentación
class DocumentViewModel extends ChangeNotifier {
    final GetDocumentsUseCase _getDocumentsUseCase;
    final DeleteDocumentUseCase _deleteDocumentUseCase;

    List<Document> _documents = [];
    bool _isLoading = false;
    String? _error;

    // Getters públicos
```

```
List<Document> get documents => List.unmodifiable(_documents);
bool get isLoading => _isLoading;
bool get hasError => _error != null;
String? get error => _error;

Future<void> loadDocuments() async {
    _ setLoading(true);
    _ clearError();

    final result = await _getDocumentsUseCase.execute();

    result.fold(
        (failure) => _ setError(failure.message),
        (documents) => _ setDocuments(documents),
    );
    _ setLoading(false);
}

Future<void> deleteDocument(String id) async {
    final result = await _deleteDocumentUseCase.execute(id);

    result.fold(
        (failure) => _ setError(failure.message),
        (_) => _ removeDocument(id),
    );
}

// Métodos privados
void _ setLoading(bool value) {
    _isLoading = value;
    notifyListeners();
}

void _ setError(String error) {
    _error = error;
    notifyListeners();
}

void _ clearError() {
    _error = null;
}

void _ setDocuments(List<Document> documents) {
```

```

        _documents = documents;
        notifyListeners();
    }

    void _removeDocument(String id) {
        _documents.removeWhere((doc) => doc.id == id);
        notifyListeners();
    }
}

// CAPA DE DOMINIO - Lógica de negocio
class GetDocumentsUseCase {
    final IDocumentRepository repository;

    GetDocumentsUseCase(this.repository);

    Future<Either<Failure, List<Document>>> execute() async {
        try {
            final documents = await repository.getDocuments();

            // Aplicar reglas de negocio
            final filteredDocuments = documents.where((doc) {
                return doc.isActive && !doc.isDeleted;
            }).toList();

            // Ordenar por fecha de actualización
            filteredDocuments.sort((a, b) =>
                b.updatedAt.compareTo(a.updatedAt)
            );

            return Right(filteredDocuments);
        } catch (e) {
            return Left(ServerFailure(e.toString()));
        }
    }
}

// CAPA DE DATOS - Acceso a datos
class DocumentRepositoryImpl implements IDocumentRepository {
    final RemoteDataSource remoteDataSource;
    final LocalDataSource localDataSource;
    final NetworkInfo networkInfo;

    @override

```

```

Future<List<Document>> getDocuments() async {
  if (await networkInfo.isConnected) {
    try {
      final remoteDocuments = await remoteDataSource.getDocuments();

      // Cache local
      await localDataSource.cacheDocuments(remoteDocuments);

      return remoteDocuments.map((model) => model.toEntity()).toList();
    } catch (e) {
      // Fallback to cache
      return _getLocalDocuments();
    }
  } else {
    return _getLocalDocuments();
  }
}

Future<List<Document>> _getLocalDocuments() async {
  final localDocuments = await localDataSource.getCachedDocuments();
  return localDocuments.map((model) => model.toEntity()).toList();
}

```

2.2.2.2 Inversión de Dependencias (DIP)

```

// Definición de contratos (abstracciones)
abstract class IAuthRepository {
  Future<Either<Failure, User>> login(String username, String password);
  Future<Either<Failure, void>> logout();
  Future<Either<Failure, User>> getCurrentUser();
}

abstract class IDocumentRepository {
  Future<List<Document>> getDocuments();
  Future<Document> getDocument(String id);
  Future<void> saveDocument(Document document);
  Future<void> deleteDocument(String id);
}

abstract class IChatRepository {
  Future<List<Chat>> getChats();
  Future<Chat> getChat(String id);
  Future<Message> sendMessage(String chatId, String content);
}

```

```
Stream<Message> messageStream(String chatId);
}

// Inyección de dependencias
class DependencyInjection {
    static void init() {
        // Registrar dependencias

        // Data Sources
        GetIt.I.registerLazySingleton<RemoteDataSource>(
            () => RemoteDataSourceImpl(
                client: GetIt.I<http.Client>(),
                baseUrl: Environment.apiUrl,
            ),
        );
    }

    GetIt.I.registerLazySingleton<LocalDataSource>(
        () => LocalDataSourceImpl(
            sharedPreferences: GetIt.I<SharedPreferences>(),
        ),
    );
}

// Repositories
GetIt.I.registerLazySingleton<IAuthRepository>(
    () => AuthRepositoryImpl(
        remoteDataSource: GetIt.I<RemoteDataSource>(),
        localDataSource: GetIt.I<LocalDataSource>(),
    ),
);

GetIt.I.registerLazySingleton<IDocumentRepository>(
    () => DocumentRepositoryImpl(
        remoteDataSource: GetIt.I<RemoteDataSource>(),
        localDataSource: GetIt.I<LocalDataSource>(),
        networkInfo: GetIt.I<NetworkInfo>(),
    ),
);

// Use Cases
GetIt.I.registerLazySingleton(
    () => LoginUseCase(GetIt.I<IAuthRepository>()),
);

GetIt.I.registerLazySingleton(
```

```

() => GetDocumentsUseCase(GetIt.I<IDocumentRepository>()),
);

// ViewModels
GetIt.I.registerFactory(
() => LoginViewModel(
    loginUseCase: GetIt.I<LoginUseCase>(),
),
);

GetIt.I.registerFactory(
() => DocumentViewModel(
    getDocumentsUseCase: GetIt.I<GetDocumentsUseCase>(),
    deleteDocumentUseCase: GetIt.I<DeleteDocumentUseCase>(),
),
);
}
}

```

2.2.3 Patrones de Diseño Implementados

2.2.3.1 Patrón Repository

```

// Interfaz del repositorio
abstract class IDocumentRepository {
    Future<Either<Failure, List<Document>>> getDocuments({
        DocumentFilter? filter,
        DocumentSort? sort,
        int? limit,
        int? offset,
    });

    Future<Either<Failure, Document>> getDocument(String id);

    Future<Either<Failure, Document>> createDocument({
        required String title,
        required String content,
        required String mimeType,
        List<String>? tags,
    });

    Future<Either<Failure, Document>> updateDocument(
        String id,
        DocumentUpdate update,
    );
}
```

```

Future<Either<Failure, void>> deleteDocument(String id);

Future<Either<Failure, void>> shareDocument(
    String id,
    List<String> userIds,
    SharePermission permission,
);
}

// Implementación concreta
class DocumentRepositoryImpl implements IDocumentRepository {
    final RemoteDataSource _remoteDataSource;
    final LocalDataSource _localDataSource;
    final NetworkInfo _networkInfo;
    final DocumentCache _cache;

    DocumentRepositoryImpl({
        required RemoteDataSource remoteDataSource,
        required LocalDataSource localDataSource,
        required NetworkInfo networkInfo,
        DocumentCache? cache,
    }) : _remoteDataSource = remoteDataSource,
        _localDataSource = localDataSource,
        _networkInfo = networkInfo,
        _cache = cache ?? DocumentCache();

    @override
    Future<Either<Failure, List<Document>>> getDocuments({
        DocumentFilter? filter,
        DocumentSort? sort,
        int? limit,
        int? offset,
    }) async {
        try {
            // Check cache first
            final cacheKey = _generateCacheKey(filter, sort, limit, offset);
            final cachedDocuments = _cache.get(cacheKey);

            if (cachedDocuments != null) {
                return Right(cachedDocuments);
            }
        }
    }

    // Check network connectivity
}

```

```

        if (!await _networkInfo.isConnected) {
            // Return cached data if available
            final localDocs = await _localDataSource.getCachedDocuments();
            return Right(_applyFiltersAndSort(localDocs, filter, sort));
        }

        // Fetch from remote
        final remoteModels = await _remoteDataSource.getDocuments(
            filter: filter?.toJson(),
            sort: sort?.toJson(),
            limit: limit,
            offset: offset,
        );

        // Convert models to entities
        final documents = remoteModels
            .map((model) => model.toEntity())
            .toList();

        // Update cache
        _cache.set(cacheKey, documents);

        // Update local storage
        await _localDataSource.cacheDocuments(remoteModels);

        return Right(documents);
    } on ServerException catch (e) {
        return Left(ServerFailure(e.message));
    } on CacheException catch (e) {
        return Left(CacheFailure(e.message));
    } catch (e) {
        return Left(UnexpectedFailure(e.toString()));
    }
}

@Override
Future<Either<Failure, Document>> createDocument({
    required String title,
    required String content,
    required String mimeType,
    List<String>? tags,
}) async {
    try {
        if (!await _networkInfo.isConnected) {

```

```
// Queue for sync when online
await _localDataSource.queueDocumentCreation(
    title: title,
    content: content,
    mimeType: mimeType,
    tags: tags,
);

return Left(NetworkFailure('No internet connection'));
}

final model = await _remoteDataSource.createDocument(
    title: title,
    content: content,
    mimeType: mimeType,
    tags: tags,
);

final document = model.toEntity();

// Clear cache as data has changed
_cache.clear();

return Right(document);
} on ServerException catch (e) {
    return Left(ServerFailure(e.message));
} catch (e) {
    return Left(UnexpectedFailure(e.toString()));
}
}

// Métodos auxiliares privados
String _generateCacheKey(
    DocumentFilter? filter,
    DocumentSort? sort,
    int? limit,
    int? offset,
) {
    final parts = [
        'documents',
        filter?.toString() ?? 'all',
        sort?.toString() ?? 'default',
        limit?.toString() ?? 'unlimited',
        offset?.toString() ?? '0',
    ];
}
```

```
];

    return parts.join('_');
}

List<Document> _applyFiltersAndSort(
    List<DocumentModel> models,
    DocumentFilter? filter,
    DocumentSort? sort,
) {
    var documents = models.map((m) => m.toEntity()).toList();

    // Apply filter
    if (filter != null) {
        documents = documents.where((doc) {
            switch (filter) {
                case DocumentFilter.owned:
                    return doc.isOwned;
                case DocumentFilter.shared:
                    return doc.isShared;
                case DocumentFilter.recent:
                    final weekAgo = DateTime.now().subtract(
                        const Duration(days: 7),
                    );
                    return doc.updatedAt.isAfter(weekAgo);
                default:
                    return true;
            }
        }).toList();
    }

    // Apply sort
    if (sort != null) {
        documents.sort((a, b) {
            switch (sort) {
                case DocumentSort.nameAsc:
                    return a.title.compareTo(b.title);
                case DocumentSort.nameDesc:
                    return b.title.compareTo(a.title);
                case DocumentSort.dateAsc:
                    return a.updatedAt.compareTo(b.updatedAt);
                case DocumentSort.dateDesc:
                    return b.updatedAt.compareTo(a.updatedAt);
                case DocumentSort.sizeAsc:

```

```

        return a.size.compareTo(b.size);
    case DocumentSort.sizeDesc:
        return b.size.compareTo(a.size);
    }
});
}

return documents;
}
}

```

2.2.3.2 Patrón Factory

```

// Factory para crear diferentes tipos de documentos
abstract class DocumentFactory {
    static Document createDocument({
        required String mimeType,
        required Map<String, dynamic> data,
    }) {
        switch (mimeType) {
            case 'application/pdf':
                return PDFDocument.fromJson(data);

            case 'text/plain':
                return TextDocument.fromJson(data);

            case 'application/vnd.ms-excel':
            case 'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet':
                return ExcelDocument.fromJson(data);

            case 'application/msword':
            case 'application/vnd.openxmlformats-
officedocument.wordprocessingml.document':
                return WordDocument.fromJson(data);

            case 'image/jpeg':
            case 'image/png':
            case 'image/gif':
                return ImageDocument.fromJson(data);

            default:
                return GenericDocument.fromJson(data);
        }
    }
}

```

```
        }
    }

// Implementaciones concretas
class PDFDocument extends Document {
    final int pageCount;
    final bool isScanned;
    final bool hasText;

    PDFDocument({
        required super.id,
        required super.title,
        required super.content,
        required super.ownerId,
        required super.createdAt,
        required super.updatedAt,
        required this.pageCount,
        required this.isScanned,
        required this.hasText,
    });

factory PDFDocument.fromJson(Map<String, dynamic> json) {
    return PDFDocument(
        id: json['id'],
        title: json['title'],
        content: json['content'] ?? '',
        ownerId: json['owner_id'],
        createdAt: DateTime.parse(json['created_at']),
        updatedAt: DateTime.parse(json['updated_at']),
        pageCount: json['page_count'] ?? 0,
        isScanned: json['is_scanned'] ?? false,
        hasText: json['has_text'] ?? true,
    );
}

@Override
Widget buildPreview() {
    return PDFPreviewWidget(
        document: this,
        showThumbnails: pageCount > 1,
    );
}

@Override
```

```
List<DocumentAction> getAvailableActions() {
    final actions = super.getAvailableActions();

    if (hasText) {
        actions.add(DocumentAction.extractText);
    }

    if (pageCount > 1) {
        actions.add(DocumentAction.splitPages);
    }

    if (!isScanned) {
        actions.add(DocumentAction.compress);
    }

    return actions;
}

class ExcelDocument extends Document {
    final int sheetCount;
    final List<String> sheetNames;
    final int totalRows;
    final int totalColumns;

    ExcelDocument({
        required super.id,
        required super.title,
        required super.content,
        required super.ownerId,
        required super.createdAt,
        required super.updatedAt,
        required this.sheetCount,
        required this.sheetNames,
        required this.totalRows,
        required this.totalColumns,
    });

    @override
    Widget buildPreview() {
        return ExcelPreviewWidget(
            document: this,
            maxPreviewRows: 10,
        );
    }
}
```

```

    }

    @Override
    List<DocumentAction> getAvailableActions() {
        final actions = super.getAvailableActions();

        actions.addAll([
            DocumentAction.exportToCSV,
            DocumentAction.viewCharts,
            DocumentAction.runFormulas,
        ]);

        return actions;
    }
}

```

2.2.3.3 Patrón Builder

```

// Builder para consultas complejas
class DocumentQueryBuilder {
    String? _userId;
    String? _searchTerm;
    DateTime? _startDate;
    DateTime? _endDate;
    List<String>? _mimeTypes;
    List<String>? _tags;
    bool _includeShared = false;
    bool _includeDeleted = false;
    DocumentSort _sort = DocumentSort.dateDesc;
    int _limit = 50;
    int _offset = 0;

    DocumentQueryBuilder();

    // Métodos de configuración fluent
    DocumentQueryBuilder forUser(String userId) {
        _userId = userId;
        return this;
    }

    DocumentQueryBuilder withSearchTerm(String term) {
        _searchTerm = term;
        return this;
    }
}

```

```
DocumentQueryBuilder inDateRange(DateTime start, DateTime end) {
    _startDate = start;
    _endDate = end;
    return this;
}

DocumentQueryBuilder fromDate(DateTime date) {
    _startDate = date;
    return this;
}

DocumentQueryBuilder toDate(DateTime date) {
    _endDate = date;
    return this;
}

DocumentQueryBuilder withMimeTypes(List<String> types) {
    _mimeTypes = types;
    return this;
}

DocumentQueryBuilder withTags(List<String> tags) {
    _tags = tags;
    return this;
}

DocumentQueryBuilder includeShared({bool include = true}) {
    _includeShared = include;
    return this;
}

DocumentQueryBuilder includeDeleted({bool include = true}) {
    _includeDeleted = include;
    return this;
}

DocumentQueryBuilder sortBy(DocumentSort sort) {
    _sort = sort;
    return this;
}

DocumentQueryBuilder limit(int limit) {
    _limit = limit;
```

```
        return this;
    }

    DocumentQueryBuilder offset(int offset) {
        _offset = offset;
        return this;
    }

    // Método de construcción
    DocumentQuery build() {
        // Validaciones
        if (_startDate != null && _endDate != null) {
            if (_startDate!.isAfter(_endDate!)) {
                throw ArgumentError('Start date must be before end date');
            }
        }

        if (_limit < 1 || _limit > 1000) {
            throw ArgumentError('Limit must be between 1 and 1000');
        }

        if (_offset < 0) {
            throw ArgumentError('Offset must be non-negative');
        }

        return DocumentQuery._(
            userId: _userId,
            searchTerm: _searchTerm,
            startDate: _startDate,
            endDate: _endDate,
            mimeTypes: _mimeTypes != null ? List.unmodifiable(_mimeTypes!) : null,
            tags: _tags != null ? List.unmodifiable(_tags!) : null,
            includeShared: _includeShared,
            includeDeleted: _includeDeleted,
            sort: _sort,
            limit: _limit,
            offset: _offset,
        );
    }
}

// Clase de consulta inmutable
class DocumentQuery {
    final String? userId;
```

```

final String? searchTerm;
final DateTime? startDate;
final DateTime? endDate;
final List<String>? mimeTypes;
final List<String>? tags;
final bool includeShared;
final bool includeDeleted;
final DocumentSort sort;
final int limit;
final int offset;

const DocumentQuery._({
  this.userId,
  this.searchTerm,
  this.startDate,
  this.endDate,
  this.mimeTypes,
  this.tags,
  required this.includeShared,
  required this.includeDeleted,
  required this.sort,
  required this.limit,
  required this.offset,
});

Map<String, dynamic> toJson() {
  return {
    if (userId != null) 'user_id': userId,
    if (searchTerm != null) 'search': searchTerm,
    if (startDate != null) 'start_date': startDate!.toIso8601String(),
    if (endDate != null) 'end_date': endDate!.toIso8601String(),
    if (mimeTypes != null) 'mime_types': mimeTypes,
    if (tags != null) 'tags': tags,
    'include_shared': includeShared,
    'include_deleted': includeDeleted,
    'sort': sort.toString(),
    'limit': limit,
    'offset': offset,
  };
}

// Uso del builder
final query = DocumentQueryBuilder()

```

```
.forUser('user123')
.withSearchTerm('contrato')
.fromDate(DateTime(2024, 1, 1))
.withMimeTypes(['application/pdf', 'text/plain'])
.includeShared()
.sortBy(DocumentSort.dateDesc)
.limit(20)
.build();
```

2.2.3.4 Patrón Strategy

```
// Estrategias de autenticación
abstract class AuthStrategy {
    Future<Either<Failure, User>> authenticate();
    bool get isAvailable;
    String get name;
    IconData get icon;
}

// Implementaciones concretas
class EmailPasswordAuthStrategy implements AuthStrategy {
    final String email;
    final String password;
    final IAuthRepository repository;

    EmailPasswordAuthStrategy({
        required this.email,
        required this.password,
        required this.repository,
    });

    @override
    Future<Either<Failure, User>> authenticate() async {
        // Validaciones
        if (!isValidEmail(email)) {
            return Left(ValidationFailure('Invalid email format'));
        }

        if (password.length < 6) {
            return Left(ValidationFailure('Password too short'));
        }

        // Intentar autenticación
        return repository.loginWithEmail(email, password);
    }
}
```

```
}

@Override
bool get isAvailable => true;

@Override
String get name => 'Email & Password';

@Override
IconData get icon => Icons.email;

bool _isValidEmail(String email) {
  return RegExp(r'^[\w-\.]+@[([\w-]+\.)+[\w-]{2,4}$').hasMatch(email);
}

class BiometricAuthStrategy implements AuthStrategy {
  final LocalAuthentication localAuth;
  final IAuthRepository repository;
  final ILocalStorage localStorage;

  BiometricAuthStrategy({
    required this.localAuth,
    required this.repository,
    required this.localStorage,
  });

  @override
  Future<Either<Failure, User>> authenticate() async {
    try {
      // Verificar disponibilidad
      if (!await localAuth.canCheckBiometrics) {
        return Left(BiometricFailure('Biometrics not available'));
      }

      // Autenticar con biometría
      final authenticated = await localAuth.authenticate(
        localizedReason: 'Authenticate to access DocuMente',
        options: const AuthenticationOptions(
          biometricOnly: true,
          stickyAuth: true,
        ),
      );
    }
  }
}
```

```
if (!authenticated) {
    return Left(BiometricFailure('Authentication failed'));
}

// Obtener token guardado
final token = await localStorage.getSecureToken();
if (token == null) {
    return Left(BiometricFailure('No saved credentials'));
}

// Validar token con servidor
return repository.loginWithToken(token);
} catch (e) {
    return Left(BiometricFailure(e.toString()));
}
}

@Override
Future<bool> get isAvailable async {
try {
    if (!await localAuth.canCheckBiometrics) {
        return false;
    }

    final availableBiometrics = await localAuth.getAvailableBiometrics();
    return availableBiometrics.isNotEmpty;
} catch (e) {
    return false;
}
}

@Override
String get name => 'Biometric';

@Override
IconData get icon => Icons.fingerprint;
}

class SocialAuthStrategy implements AuthStrategy {
    final SocialProvider provider;
    final IAuthRepository repository;

    SocialAuthStrategy({
        required this.provider,
```

```
    required this.repository,
});

@Override
Future<Either<Failure, User>> authenticate() async {
    try {
        // Iniciar flujo OAuth
        final authResult = await provider.signIn();

        if (authResult == null) {
            return Left(SocialAuthFailure('User cancelled'));
        }

        // Validar con backend
        return repository.loginWithSocial(
            provider: provider.name,
            token: authResult.accessToken,
            userId: authResult.userId,
        );
    } catch (e) {
        return Left(SocialAuthFailure(e.toString()));
    }
}

@Override
bool get isAvailable => provider.isAvailable;

@Override
String get name => provider.displayName;

@Override
IconData get icon => provider.icon;
}

// Contexto que usa las estrategias
class AuthenticationManager {
    final List<AuthStrategy> _strategies = [];
    AuthStrategy? _currentStrategy;

    void addStrategy(AuthStrategy strategy) {
        _strategies.add(strategy);
    }

    void setStrategy(AuthStrategy strategy) {
```

```

    _currentStrategy = strategy;
}

Future<List<AuthStrategy>> getAvailableStrategies() async {
  final available = <AuthStrategy>[];

  for (final strategy in _strategies) {
    if (await strategy.isAvailable) {
      available.add(strategy);
    }
  }

  return available;
}

Future<Either<Failure, User>> authenticate() async {
  if (_currentStrategy == null) {
    return Left(AuthFailure('No authentication strategy selected'));
  }

  return _currentStrategy!.authenticate();
}
}

```

2.2.4 Gestión del Estado Global

2.2.4.1 Arquitectura de Provider

```

// Configuración del árbol de providers
class AppProviders extends StatelessWidget {
  final Widget child;

  const AppProviders({Key? key, required this.child}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        // Servicios base
        Provider<IAuthRepository>(
          create: (_) => GetIt.I<IAuthRepository>(),
        ),
        Provider<IDocumentRepository>(
          create: (_) => GetIt.I<IDocumentRepository>(),
        ),
      ],
    );
  }
}

```

```
Provider<IChatRepository>(
  create: (_) => GetIt.I<IChatRepository>(),
),

// ViewModels con dependencias
ChangeNotifierProvider<AuthViewModel>(
  create: (context) => AuthViewModel(
    authRepository: context.read<IAuthRepository>(),
  ),
),

// Proxy providers que dependen de otros
ChangeNotifierProxyProvider<AuthViewModel, DocumentViewModel>(
  create: (context) => DocumentViewModel(
    documentRepository: context.read<IDocumentRepository>(),
  ),
  update: (context, auth, previous) {
    previous?.updateUser(auth.currentUser);
    return previous!;
  },
),

ChangeNotifierProxyProvider<AuthViewModel, ChatViewModel>(
  create: (context) => ChatViewModel(
    chatRepository: context.read<IChatRepository>(),
  ),
  update: (context, auth, previous) {
    previous?.updateUser(auth.currentUser);
    return previous!;
  },
),

// Providers de UI
ChangeNotifierProvider<ThemeProvider>(
  create: (_) => ThemeProvider(),
),

ChangeNotifierProvider<NavigationProvider>(
  create: (_) => NavigationProvider(),
),

// Stream providers
StreamProvider<ConnectivityStatus>(
  create: (_) => ConnectivityService().statusStream,
```

```
        initialValue: ConnectivityStatus.unknown,
    ),
],
child: child,
);
}
}

// ViewModel base con funcionalidad común
abstract class BaseViewModel extends ChangeNotifier {
    bool _disposed = false;
    bool _isLoading = false;
    String? _error;

    bool get isLoading => _isLoading;
    bool get hasError => _error != null;
    String? get error => _error;

    @protected
    void setLoading(bool value) {
        if (_disposed) return;
        _isLoading = value;
        notifyListeners();
    }

    @protected
    void setError(String? error) {
        if (_disposed) return;
        _error = error;
        notifyListeners();
    }

    @protected
    void clearError() {
        setError(null);
    }

    @protected
    Future<T?> runAsync<T>(Future<T> Function() operation) async {
        setLoading(true);
        clearError();

        try {
            final result = await operation();
        }
    }
}
```

```

        return result;
    } catch (e) {
        setError(e.toString());
        return null;
    } finally {
        setLoading(false);
    }
}

@Override
void dispose() {
    _disposed = true;
    super.dispose();
}

```

```

@Override
void notifyListeners() {
    if (!_disposed) {
        super.notifyListeners();
    }
}

```

2.2.5 Flujo de Datos

2.2.5.1 Diagrama de Flujo de Datos

Presentación → Estado → Dominio → Datos → APIs/DB

2.2.5.2 Implementación del Flujo

```

// 1. Usuario interactúa con la UI
class DocumentUploadButton extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return ElevatedButton(
            onPressed: () => _handleUpload(context),
            child: const Text('Upload Document'),
        );
    }

    void _handleUpload(BuildContext context) {
        // Trigger action in ViewModel
        context.read<DocumentViewModel>().uploadDocument();
    }
}

```

```
}

// 2. ViewModel maneja la lógica
class DocumentViewModel extends BaseViewModel {
    final UploadDocumentUseCase _uploadUseCase;

    Future<void> uploadDocument() async {
        await runAsync(() async {
            // Pick file
            final file = await FilePicker.platform.pickFiles(
                type: FileType.custom,
                allowedExtensions: ['pdf', 'txt', 'doc', 'docx'],
            );

            if (file == null) return;

            // Execute use case
            final result = await _uploadUseCase.execute(
                filePath: file.files.single.path!,
                fileName: file.files.single.name,
            );

            result.fold(
                (failure) => throw Exception(failure.message),
                (document) => _addDocument(document),
            );
        });
    }

    void _addDocument(Document document) {
        _documents.insert(0, document);
        notifyListeners();
    }
}

// 3. Use Case ejecuta la lógica de negocio
class UploadDocumentUseCase {
    final IDocumentRepository _repository;
    final DocumentValidator _validator;
    final FileCompressor _compressor;

    Future<Either<Failure, Document>> execute({
        required String filePath,
        required String fileName,
```

```

}) async {
try {
// Validate file
final validationResult = _validator.validate(filePath);
if (!validationResult.isValid) {
    return Left(ValidationFailure(validationResult.error!));
}

// Compress if needed
final file = File(filePath);
final fileSize = await file.length();

String processedPath = filePath;
if (fileSize > 10 * 1024 * 1024) { // >10MB
    processedPath = await _compressor.compress(filePath);
}

// Upload through repository
return _repository.uploadDocument(
    filePath: processedPath,
    fileName: fileName,
);
} catch (e) {
    return Left(UnexpectedFailure(e.toString()));
}
}
}

// 4. Repository coordina las fuentes de datos
class DocumentRepositoryImpl implements IDocumentRepository {
@Override
Future<Either<Failure, Document>> uploadDocument({
    required String filePath,
    required String fileName,
}) async {
try {
// Create local record first
final localDoc = await _localDataSource.createPendingUpload(
    filePath: filePath,
    fileName: fileName,
);
}

// Upload to server
final remoteDoc = await _remoteDataSource.uploadDocument(

```

```

        filePath: filePath,
        fileName: fileName,
    );

    // Update local record with server ID
    await _localDataSource.updateUploadStatus(
        localId: localDoc.id,
        remoteId: remoteDoc.id,
        status: UploadStatus.completed,
    );

    return Right(remoteDoc.toEntity());
} catch (e) {
    // Handle offline scenario
    if (e is NetworkException) {
        // Keep in pending uploads
        return Left(NetworkFailure('Upload queued for when online'));
    }

    return Left(ServerFailure(e.toString()));
}
}
}
}

```

2.2.6 Manejo de Errores

```

// Clase base para todos los failures
abstract class Failure {
    final String message;
    final String? code;
    final dynamic originalError;

    const Failure({
        required this.message,
        this.code,
        this.originalError,
    });

    @override
    String toString() => '$runtimeType: $message';
}

// Failures específicos

```

```
class NetworkFailure extends Failure {
  const NetworkFailure(String message, {String? code})
    : super(message: message, code: code);
}

class ServerFailure extends Failure {
  final int? statusCode;

  const ServerFailure(
    String message, {
    this.statusCode,
    String? code,
  }) : super(message: message, code: code);
}

class CacheFailure extends Failure {
  const CacheFailure(String message) : super(message: message);
}

class ValidationFailure extends Failure {
  final Map<String, String>? fieldErrors;

  const ValidationFailure(
    String message, {
    this.fieldErrors,
  }) : super(message: message);
}

class AuthenticationFailure extends Failure {
  final AuthFailureType type;

  const AuthenticationFailure(
    String message, {
    required this.type,
  }) : super(message: message);
}

enum AuthFailureType {
  invalidCredentials,
  userNotFound,
  userDisabled,
  tokenExpired,
  insufficientPermissions,
}
```

```

// Either type para manejo funcional de errores
typedef FutureEither<T> = Future<Either<Failure, T>>;

// Extension methods para Either
extension EitherX<L, R> on Either<L, R> {
  R getOrElse(R Function() defaultValue) {
    return fold((_) => defaultValue(), (r) => r);
  }

  Either<L, R2> map<R2>(R2 Function(R) f) {
    return fold((l) => Left(l), (r) => Right(f(r)));
  }

  Either<L, R2> flatMap<R2>(Either<L, R2> Function(R) f) {
    return fold((l) => Left(l), (r) => f(r));
  }
}

```

2.2.6.2 Manejo Global de Errores

```

// Error handler global
class GlobalErrorHandler {
  static final _errorStreamController =
  StreamController<ErrorEvent>.broadcast();
  static Stream<ErrorEvent> get errorStream => _errorStreamController.stream;

  static void handleError(
    dynamic error,
    StackTrace? stackTrace,
    String? context,
  ) {
    // Log error
    debugPrint('Error in $context: $error');
    if (stackTrace != null) {
      debugPrint('Stack trace: $stackTrace');
    }
  }

  // Convert to user-friendly message
  final userMessage = _getUserMessage(error);

  // Emit error event
  _errorStreamController.add(ErrorEvent(
    error: error,
  ));
}

```

```
        message: userMessage,
        context: context,
        timestamp: DateTime.now(),
    ));

// Report to crash analytics
if (!kDebugMode) {
    FirebaseCrashlytics.instance.recordError(
        error,
        stackTrace,
        reason: context,
    );
}
}

static String _getUserMessage(dynamic error) {
    if (error is Failure) {
        return error.message;
    }

    if (error is NetworkException) {
        return 'Connection error. Please check your internet.';
    }

    if (error is TimeoutException) {
        return 'Request timed out. Please try again.';
    }

    if (error is FormatException) {
        return 'Invalid data format received.';
    }

    // Generic message for unknown errors
    return 'An unexpected error occurred. Please try again.';
}

static void dispose() {
    _errorStreamController.close();
}

// Widget para mostrar errores globalmente
class GlobalErrorListener extends StatefulWidget {
    final Widget child;
```

```
const GlobalErrorListener({  
  Key? key,  
  required this.child,  
) : super(key: key);  
  
@override  
State<GlobalErrorListener> createState() => _GlobalErrorListenerState();  
}  
  
class _GlobalErrorListenerState extends State<GlobalErrorListener> {  
  late StreamSubscription<ErrorEvent> _errorSubscription;  
  
  @override  
  void initState() {  
    super.initState();  
    _errorSubscription = GlobalErrorHandler.errorStream.listen(_showError);  
  }  
  
  @override  
  void dispose() {  
    _errorSubscription.cancel();  
    super.dispose();  
  }  
  
  void _showError(ErrorEvent event) {  
    if (!mounted) return;  
  
    // Show snackbar for non-critical errors  
    if (event.error is! CriticalError) {  
      ScaffoldMessenger.of(context).showSnackBar(  
        SnackBar(  
          content: Text(event.message),  
          backgroundColor: Colors.red,  
          action: SnackBarAction(  
            label: 'Retry',  
            onPressed: () => _retry(event),  
          ),  
        ),  
      );  
    } else {  
      // Show dialog for critical errors  
      showDialog(  
        context: context,
```

```

        barrierDismissible: false,
        builder: (context) => AlertDialog(
            title: const Text('Critical Error'),
            content: Text(event.message),
            actions: [
                TextButton(
                    onPressed: () => _restart(),
                    child: const Text('Restart App'),
                ),
            ],
        ),
    );
}

void _retry(ErrorEvent event) {
    // Implement retry logic based on context
}

void _restart() {
    // Restart app
    Phoenix.rebirth(context);
}

@Override
Widget build(BuildContext context) {
    return widget.child;
}

```

2.2.7 Optimizaciones de Rendimiento

2.2.7.1 Lazy Loading y Paginación

```

// Implementación de lista con paginación infinita
class PaginatedListView<T> extends StatefulWidget {
    final Future<List<T>> Function(int page, int pageSize) loadPage;
    final Widget Function(BuildContext, T) itemBuilder;
    final int pageSize;
    final Widget? loadingWidget;
    final Widget? errorWidget;
    final Widget? emptyWidget;

    const PaginatedListView({
        Key? key,

```

```
required this.loadPage,
required this.itemBuilder,
this.pageSize = 20,
this.loadingWidget,
this.errorWidget,
this.emptyWidget,
}) : super(key: key);

@Override
State<PaginatedListView<T>> createState() => _PaginatedListViewState<T>();
}

class _PaginatedListViewState<T> extends State<PaginatedListView<T>> {
final ScrollController _scrollController = ScrollController();
final List<T> _items = [];

int _currentPage = 0;
bool _isLoading = false;
bool _hasMore = true;
String? _error;

@Override
void initState() {
super.initState();
_loadNextPage();
_scrollController.addListener(_onScroll);
}

@Override
void dispose() {
_scrollController.dispose();
super.dispose();
}

void _onScroll() {
if (_scrollController.position.pixels >=
_scrollController.position.maxScrollExtent - 200) {
_loadNextPage();
}
}

Future<void> _loadNextPage() async {
if (_isLoading || !_hasMore) return;
```

```
        setState(() {
            _isLoading = true;
            _error = null;
        });

    try {
        final newItems = await widget.loadPage(_currentPage, widget.pageSize);

        setState(() {
            _items.addAll(newItems);
            _currentPage++;
            _hasMore = newItems.length == widget.pageSize;
            _isLoading = false;
        });
    } catch (e) {
        setState(() {
            _error = e.toString();
            _isLoading = false;
        });
    }
}

@Override
Widget build(BuildContext context) {
    if (_items.isEmpty && _isLoading) {
        return Center(
            child: widget.loadingWidget ?? const CircularProgressIndicator(),
        );
    }

    if (_items.isEmpty && _error != null) {
        return Center(
            child: widget.errorWidget ?? Text('Error: ${_error}'),
        );
    }

    if (_items.isEmpty) {
        return Center(
            child: widget.emptyWidget ?? const Text('No items found'),
        );
    }

    return RefreshIndicator(
        onRefresh: _refresh,
```

```
child: ListView.builder(
  controller: _scrollController,
  itemCount: _items.length + (_hasMore ? 1 : 0),
  itemBuilder: (context, index) {
    if (index == _items.length) {
      return Padding(
        padding: const EdgeInsets.all(16),
        child: Center(
          child: _error != null
            ? Column(
                children: [
                  Text('Error: $_error'),
                  ElevatedButton(
                    onPressed: _loadNextPage,
                    child: const Text('Retry'),
                  ),
                ],
            )
            : const CircularProgressIndicator(),
      ),
    );
  }
}

return widget.itemBuilder(context, _items[index]);
},
),
);
}
}

Future<void> _refresh() async {
  setState(() {
    _items.clear();
    _currentPage = 0;
    _hasMore = true;
    _error = null;
  });
}

await _loadNextPage();
}
}
```

2.2.7.2 Caché en Memoria

// Sistema de caché con LRU (Least Recently Used)

```

class LRUCache<K, V> {
    final int maxSize;
    final Map<K, V> _cache = {};
    final List<K> _keys = [];

    LRUCache({required this.maxSize});

    V? get(K key) {
        if (_cache.containsKey(key)) {
            // Move to end (most recently used)
            _keys.remove(key);
            _keys.add(key);
            return _cache[key];
        }
        return null;
    }

    void set(K key, V value) {
        if (_cache.containsKey(key)) {
            // Update existing
            _keys.remove(key);
        } else if (_cache.length >= maxSize) {
            // Remove least recently used
            final lru = _keys.removeAt(0);
            _cache.remove(lru);
        }
        _cache[key] = value;
        _keys.add(key);
    }

    void clear() {
        _cache.clear();
        _keys.clear();
    }

    int get size => _cache.length;
}

// Aplicación del caché en el repositorio
class CachedDocumentRepository implements IDocumentRepository {
    final IDocumentRepository _repository;
    final LRUCache<String, Document> _documentCache;
    final LRUCache<String, List<Document>> _listCache;
}

```

```
CachedDocumentRepository({  
    required IDocumentRepository repository,  
    int cacheSize = 100,  
}) : _repository = repository,  
    _documentCache = LRUCache(maxSize: cacheSize),  
    _listCache = LRUCache(maxSize: 20);  
  
@override  
Future<Either<Failure, Document>> getDocument(String id) async {  
    // Check cache first  
    final cached = _documentCache.get(id);  
    if (cached != null) {  
        return Right(cached);  
    }  
  
    // Fetch from repository  
    final result = await _repository.getDocument(id);  
  
    // Cache successful result  
    result.fold(  
        (failure) => null,  
        (document) => _documentCache.set(id, document),  
    );  
  
    return result;  
}  
  
@override  
Future<Either<Failure, List<Document>>> getDocuments({  
    DocumentFilter? filter,  
    DocumentSort? sort,  
    int? limit,  
    int? offset,  
}) async {  
    // Generate cache key  
    final cacheKey = _generateCacheKey(filter, sort, limit, offset);  
  
    // Check cache  
    final cached = _listCache.get(cacheKey);  
    if (cached != null) {  
        return Right(cached);  
    }  
}
```

```
// Fetch from repository
final result = await _repository.getDocuments(
    filter: filter,
    sort: sort,
    limit: limit,
    offset: offset,
);

// Cache successful result
result.fold(
    (failure) => null,
    (documents) {
        _listCache.set(cacheKey, documents);
        // Also cache individual documents
        for (final doc in documents) {
            _documentCache.set(doc.id, doc);
        }
    },
);

return result;
}

@Override
Future<Either<Failure, Document>> updateDocument(
    String id,
    DocumentUpdate update,
) async {
    final result = await _repository.updateDocument(id, update);

    // Invalidate caches on success
    result.fold(
        (failure) => null,
        (document) {
            _documentCache.set(id, document);
            _listCache.clear(); // Clear list cache as order might change
        },
    );

    return result;
}

String _generateCacheKey(
    DocumentFilter? filter,
```

```
DocumentSort? sort,
int? limit,
int? offset,
) {
    return '${filter?.name ?? 'all'}_${sort?.name ?? 'default'}_${limit ?? 'all'}_${offset ?? '0'}';
}
```

2.3 Estructura del proyecto.

Siguiendo una arquitectura modular que facilita el mantenimiento, la escalabilidad y la colaboración en equipo. La estructura del proyecto refleja las mejores prácticas de desarrollo móvil multiplataforma, implementando una clara separación de responsabilidades entre las diferentes capas de la aplicación. La organización del código sigue el patrón Model-View-Controller (MVC) adaptado al paradigma de Flutter, donde las vistas (screens y widgets) están claramente separadas de la lógica de negocio (services) y los modelos de datos. Esta arquitectura permite una evolución ordenada del proyecto y facilita la incorporación de nuevas funcionalidades sin afectar las existentes.

Funcionalidades Principales

- **Sistema de Autenticación:** Login/registro con roles de usuario y administrador
- **Gestión Documental:** Subida, visualización y compartición de documentos
- **Chat Inteligente:** Sistema de mensajería con bot integrado
- **Panel de Administración:** Gestión completa de usuarios, documentos y chats

- **Interfaz Adaptativa:** Diseño responsivo para móviles, tablets y escritorio

```

  └── utils/                      # Utilidades (preparado para futuras implementaciones)
      ├── pubspec.yaml             # Gestión de dependencias y metadatos
      └── pubspec.lock              # Versiones bloqueadas de dependencias

      ├── android/                 # Configuración específica de Android
      ├── ios/                     # Configuración específica de iOS
      ├── web/                     # Configuración para versión web
      ├── windows/                 # Configuración para Windows
      ├── linux/                   # Configuración para Linux
      └── macos/                   # Configuración para macOS

      ├── assets/                  # Recursos estáticos (imágenes, fuentes, etc.)
      ├── test/                    # Tests unitarios y de integración
      └── build/                   # Archivos generados de compilación

          ├── profile_screen.dart    # Gestión del perfil de usuario
          ├── documents_screen.dart  # Gestión de documentos propios
          ├── document_detail_screen.dart # Visualización detallada de documentos
          ├── shared_documents_screen.dart # Documentos compartidos
          ├── chat.dart                # Interfaz de chat con MENTIA
          ├── chat_list_screen.dart   # Lista de conversaciones
          └── admin_panel_screen.dart # Panel de administración

          ├── services/               # Capa de lógica de negocio
          ├── services.dart            # Exportación de servicios
          └── auth_service.dart        # Servicio de autenticación

          ├── widgets/                # Componentes reutilizables
          ├── botmessage.dart          # Widget para mensajes del bot
          ├── usermessage.dart         # Widget para mensajes del usuario
          └── chatinputbox.dart        # Campo de entrada del chat
  
```

Patrones de Diseño Implementados

1. Patrón Singleton

Utilizado en AuthService para garantizar una única instancia:

```

static final AuthService _instance = AuthService._internal();
factory AuthService() => _instance;
AuthService._internal();
  
```

2. Patrón Provider (Observer)

Implementación reactiva para gestión de estado:

```

MultiProvider(
  providers: [
    ChangeNotifierProvider(create: (_) => AuthService())
  ],
  child: const MyApp(),
)
  
```

3. Patrón Factory

Los modelos implementan factory constructors para deserialización:

```
factory User.fromJson(Map<String, dynamic> json) => User(  
    id: json['id'],  
    username: json['username'],  
    // ...  
)
```

4. Separación de Responsabilidades

- Models: Representación de datos y reglas de negocio
- Services: Lógica de aplicación y gestión de estado
- Screens: Lógica de presentación
- Widgets: Componentes de UI reutilizables

1. Archivo Principal (main.dart)

Responsabilidades:

- Inicialización del sistema de providers
- Configuración del tema Material Design 3
- Implementación del AuthWrapper para control de acceso
- Configuración de la pantalla de login

Características destacadas:

- Uso de MultiProvider para inyección de dependencias
- Tema personalizado con color primario #6B4CE6
- Sistema de autenticación reactivo con Consumer<AuthService>
- Formulario de login con validación en tiempo real

2. Capa de Modelos (models/)

user.dart

```
enum UserRole { user, admin }
```

```
class User {  
    final String id;  
    final String username;
```

```
final String email;  
final UserRole role;  
final String? avatar;  
final DateTime createdAt;  
  
bool get isAdmin => role == UserRole.admin;  
}  
document.dart
```

```
class Document {  
    final String id;  
    final String title;  
    final String content;  
    final String fileName;  
    final String mimeType;  
    final String ownerId;  
    final String ownerName;  
    final bool isShared;  
    final List<String> sharedWithUsers;  
    final DateTime createdAt;  
    final DateTime updatedAt;  
}  
chat.dart
```

```
class ChatModel {  
    final String id;  
    final String title;  
    final List<ChatMessage> messages;  
    final DateTime createdAt;  
}  
  
class ChatMessage {  
    final String id;
```

```
final String content;  
final bool isBot;  
final DateTime timestamp;  
}
```

3. Capa de Servicios (services/)

auth_service.dart

Características:

- Implementación del patrón Singleton
- Extends ChangeNotifier para notificación reactiva
- Gestión completa del estado de autenticación
- Simulación de llamadas API con delays artificiales

Métodos principales:

- login(): Autenticación de usuario
- register(): Registro de nuevo usuario
- logout(): Cierre de sesión
- updateProfile(): Actualización de perfil

4. Capa de Presentación (screens/)

home_screen.dart

- Sistema de navegación adaptativa (BottomNavigationBar vs NavigationRail)
- Dashboard personalizado según rol de usuario
- Widgets de estadísticas para administradores
- Secciones: documentos recientes, chats recientes, accesos rápidos

documents_screen.dart

- Lista filtrable de documentos del usuario
- Búsqueda en tiempo real
- Filtros por tipo de archivo
- FloatingActionButton para agregar documentos

admin_panel_screen.dart

- Acceso exclusivo para administradores
- Sistema de tabs (Dashboard, Usuarios, Documentos, Chats)

- Gestión CRUD completa de usuarios
- Protecciones especiales para cuenta admin principal

5. Widgets Reutilizables (widgets/)

Componentes del Sistema de Chat:

botmessage.dart:

- Diseño con gradiente distintivo
- Avatar con icono de asistente
- Burbuja de mensaje con bordes personalizados

usermessage.dart:

- Alineación opuesta al bot
- Avatar con inicial del usuario
- Estilo visual diferenciado

chatinputbox.dart:

- Campo de texto expandible
- Botón de envío con estado habilitado/deshabilitado
- Opción para adjuntar archivos

Gestión de Estado y Dependencias

Gestión de Estado con Provider

La aplicación utiliza Provider como solución principal de gestión de estado:

```
dart
MultiProvider(
  providers: [
    ChangeNotifierProvider(create: (_) => AuthService()),
  ],
  child: Consumer<AuthService>(
    builder: (context, authService, child) {
      return authService.isAuthenticated
        ? const HomeScreen()
        : const LoginScreen();
    }
  )
);
```

```
},  
),  
)
```

Dependencias Principales (pubspec.yaml)

```
yaml  
dependencies:  
  flutter:  
    sdk: flutter          # Framework base  
  flutter_localizations:  
    sdk: flutter          # Soporte multiidioma  
  intl: ^0.19.0           # Formateo de fechas y números  
  cupertino_icons: ^1.0.8   # Iconos estilo iOS  
  http: ^1.1.0            # Cliente HTTP para API REST  
  shared_preferences: ^2.2.0  # Almacenamiento local persistente  
  provider: ^6.1.0          # Gestión de estado  
  image_picker: ^1.0.7        # Selección de imágenes
```

dev_dependencies:

```
  flutter_test:  
    sdk: flutter          # Framework de testing  
  flutter_lints: ^5.0.0      # Análisis estático de código
```

Navegación Adaptativa

El sistema de navegación se adapta automáticamente al tamaño de pantalla:

- Móviles: BottomNavigationBar
- Tablets y Escritorio: NavigationRail

Convenciones y Estándares

1. Nomenclatura de Archivos

- Archivos en snake_case: auth_service.dart, home_screen.dart
- Sufijos descriptivos: _screen para pantallas, _service para servicios

2. Nomenclatura de Clases y Variables

- Clases en PascalCase: AuthService, HomeScreen
- Variables en camelCase: currentUser, isAuthenticated
- Constantes en camelCase con const: primaryColor

3. Organización de Imports

// 1. Imports de Flutter primero

```
import 'package:flutter/material.dart';
```

// 2. Imports de packages externos

```
import 'package:provider/provider.dart';
```

// 3. Imports relativos del proyecto

```
import '../models/user.dart';
```

```
import '../services/auth_service.dart';
```

4. Estructura de Widgets

```
dart
```

```
class MyWidget extends StatelessWidget {
```

// Constructor const cuando sea posible

```
const MyWidget({super.key, required this.title});
```

// Propiedades finales para inmutabilidad

```
final String title;
```

// Build method al final

```
@override
```

```
Widget build(BuildContext context) {
```

```
    return _buildContent();
```

```
}
```

// Métodos helper privados con underscore

```
Widget _buildContent() {
```

```
//...
}
}
```

Escalabilidad y Mantenimiento

Preparación para Escalabilidad

La estructura actual está diseñada para facilitar el crecimiento:

1. Directorio Utils (*Preparado*)

```
lib/utils/
├── validators.dart  # Validadores de formularios
├── formatters.dart  # Formateadores de datos
├── helpers.dart     # Funciones auxiliares
└── constants.dart   # Constantes globales
```

2. Modularidad

- Cada feature puede agregarse como un nuevo módulo
- Sin dependencias cruzadas entre módulos
- Interfaces claras entre capas

3. Servicios Independientes

- Nuevos servicios siguen el patrón establecido por AuthService
- Inyección de dependencias centralizada
- Fácil testing e intercambio de implementaciones

4. Sistema de Rutas (*Preparado*)

```
dart
```

```
// Preparado para Navigator 2.0
class AppRouter {
  static const String home = '/home';
  static const String documents = '/documents';
  static const String admin = '/admin';
}
```

Soporte Multiplataforma

El proyecto soporta múltiples plataformas:

- Android: SDK mínimo definido en android/app/build.gradle
- iOS: Configuración en ios/Runner
- Web: Habilitado con flutter create --platforms web
- Desktop: Windows, Linux, macOS

Características de Mantenibilidad

1. Responsabilidades Separadas: Cada clase tiene una única responsabilidad
2. Código Reutilizable: Widgets compartidos reducen duplicación
3. Testing Friendly: Arquitectura que facilita unit testing
4. Documentación: Código autodocumentado con comentarios claros

2.4 Pantallas implementadas

Implementa una arquitectura de pantallas altamente estructurada que sigue los principios de Material Design 3, adaptada a las necesidades específicas de la gestión documental con integración de IA. Cada pantalla ha sido diseñada con un enfoque orientado a la experiencia de usuario, priorizando la **accesibilidad, rendimiento y usabilidad** en diferentes tamaños de dispositivos.

Las pantallas implementan patrones de diseño **adaptativo** que permiten una experiencia fluida en:

- **Dispositivos móviles** (<600px): Layout vertical con navegación inferior
- **Tablets** (600-900px): Layout mixto con navegación lateral compacta
- **Desktop** (>900px): Layout horizontal con navegación lateral expandida

Pantallas Principales Implementadas

1. Pantalla de Login (LoginScreen)

Ubicación: main.dart (componente integrado)

Tipo: StatelessWidget

Características Técnicas

- **Gradiente corporativo:** Degradado de morado (#6B4CE6) a rosa (#E91E63)
- **Validación en tiempo real** con GlobalKey<FormState>
- **Estados reactivos** de carga y error con ChangeNotifier
- **Animaciones fluidas** con BoxShadow y transiciones suaves

Funcionalidades Implementadas

```
// Gestión de estado de autenticación
Consumer<AuthService>(
  builder: (context, authService, _) {
    return authService.isAuthenticated
      ? const HomeScreen()
      : const LoginScreen();
  },
)
```

- **Validación de campos:** Username y contraseña con mensajes contextuales
- **Toggle de visibilidad:** Contraseña con IconButton dinámico
- **Estados de carga:** CircularProgressIndicator durante autenticación
- **Manejo de errores:** Snackbar con feedback visual
- **Navegación automática:** Redirección post-autenticación exitosa

Elementos de UI

- Logo corporativo con elevación y sombras
- Formulario con bordes redondeados y validación
- Botón principal con gradiente y estados disabled
- Enlace de registro con navegación Navigator.push

2. Pantalla de Registro (RegisterScreen)

Ubicación: screens/register_screen.dart

Tipo: StatelessWidget

Características Avanzadas

- **Formulario completo** con 4 campos validados
- **Validación de email** con RegExp integrado
- **Confirmación de contraseña** con comparación en tiempo real
- **Checkbox de términos** obligatorio para activar registro
- **Gradiente en botón** con efectos de elevación

Validaciones Implementadas

```
// Validación de email con expresión regular
if (!RegExp(r'^[\w-\.]+\@[([\w-]+\.)+[\w-]{2,4}\$').hasMatch(value)) {
  return 'Por favor, ingresa un correo válido';
}
```

```
// Validación de confirmación de contraseña
if (value != _passwordController.text) {
  return 'Las contraseñas no coinciden';
}
```

Estados y Feedback

- **Indicadores de fortaleza:** Validación de contraseña mínima (6 caracteres)
 - **Estados de carga:** Botón deshabilitado durante proceso
 - **Términos obligatorios:** UI reactiva según aceptación
 - **Navegación contextual:** Regreso a login y redirección post-registro
-

3. Pantalla Principal (HomeScreen)

Ubicación: screens/home_screen.dart

Tipo: StatelessWidget con navegación adaptativa

Sistema de Navegación Responsivo

```
// Navegación adaptativa según tamaño de pantalla
final isSmallScreen = MediaQuery.of(context).size.width < 600;
final isMediumScreen = MediaQuery.of(context).size.width >= 600 &&
    MediaQuery.of(context).size.width < 900;
```

Tamaño de Pantalla	Navegación	Características
Móvil (<600px)	BottomNavigationBar	Layout vertical compacto
Tablet (600-900px)	NavigationRail compacto	Iconos con etiquetas seleccionadas
Desktop (>900px)	NavigationRail expandido	Etiquetas visibles y mayormente espaciado

Dashboard Personalizado por Rol

- **Para usuarios normales:** Accesos rápidos, documentos recientes, historial de chats
- **Para administradores:** Estadísticas del sistema, métricas de usuarios, panel de control

Componentes Adaptativos

// Sistema de tipografía responsiva

```
TextStyle _getResponsiveTextStyle(double screenWidth, String type) {  
    final fontSize = screenWidth >= 900 ? 28.0 :  
        screenWidth >= 600 ? 24.0 : 20.0;  
    return TextStyle(fontSize: fontSize, fontWeight: FontWeight.bold);  
}
```

Widgets Destacados

- **Header de bienvenida:** Saludo personalizado según hora del día
 - **Tarjetas de acceso rápido:** Navegación directa a funciones principales
 - **Estadísticas en tiempo real:** Solo para administradores
 - **Actividad reciente:** Documentos y chats más utilizados
-

4. Pantalla de Chat (Chat)

Ubicación: screens/chat.dart

Tipo: StatelessWidget con TickerProviderStateMixin

Arquitectura de Mensajería

- **Diferenciación visual:** Mensajes del usuario vs MentIA con widgets personalizados
- **Animaciones fluidas:** AnimationController con FadeTransition
- **Gestión de estado:** Lista reactiva de mensajes con ScrollController
- **Drawer lateral:** Navegación contextual y opciones adicionales

Componentes Especializados

```
// Widgets personalizados para mensajes  
isBot ? BotMessage(  
    nombre: 'MentIA',  
    contenido: message['content'],  
    inicialNombre: 'M',  
) : UserMessage(  
    nombre: nombre,  
    contenido: message['content'],  
    inicialNombre: inicialNombre,  
)
```

Funcionalidades Avanzadas

- **Estado vacío ilustrativo:** Sugerencias de preguntas iniciales
- **Drawer contextual:** Nueva conversación, historial, configuración
- **Header informativo:** Estado de conexión con MentIA
- **Input expandible:** ChatInputBox con opciones de archivos adjuntos

Estados Manejados

- **Loading:** Durante inicialización de datos
 - **Empty:** Conversación nueva con sugerencias
 - **Active:** Lista de mensajes con scroll automático
 - **Error:** Manejo graceful de fallos de conexión
-

5. Pantalla de Documentos (DocumentsScreen)

Ubicación: screens/documents_screen.dart

Tipo: StatelessWidget con gestión CRUD completa

Sistema de Filtrado Avanzado

```
// Filtrado en tiempo real con múltiples criterios
void _filterDocuments() {
    final query = _searchController.text.toLowerCase();
    _filteredDocuments = _documents.where((doc) {
        final matchesSearch = doc.title.toLowerCase().contains(query) ||
            doc.fileName.toLowerCase().contains(query);
        final matchesFilter = _selectedFilter == 'Todos' ||
            (_selectedFilter == 'Compartidos' && doc.isShared) ||
            (_selectedFilter == 'PDF' && doc.mimeType.contains('pdf'));
        return matchesSearch && matchesFilter;
    }).toList();
}
```

Elementos de UI Adaptativos

- **Visualización responsiva:** Lista en móvil, rejilla en tablet/desktop
- **Chips de filtrado:** Filtros por tipo, estado y fecha
- **Tarjetas de documento:** Metadata completa con indicadores visuales
- **FloatingActionButton:** Subida de documentos con modal de selección

Funcionalidades de Gestión

- **Búsqueda semántica:** Filtrado por título y contenido
 - **Acciones contextuales:** Menú con opciones (ver, compartir, descargar, eliminar)
 - **Indicadores de estado:** Documentos compartidos, fechas relativas
 - **Carga simulada:** Estados de loading con CircularProgressIndicator
-

6. Pantalla de Documentos Compartidos (SharedDocumentsScreen)

Ubicación: screens/shared_documents_screen.dart

Tipo: StatefulWidget con gestión de permisos

Características Específicas

- **Filtrado por propietario:** Búsqueda por usuario que compartió
- **Indicadores de permisos:** Visualización de nivel de acceso (lectura/edición)
- **Metadata extendida:** Información del propietario original y fecha de compartición
- **Estados de acceso:** Diferentes acciones según permisos del usuario

Gestión de Permisos

```
// Indicadores visuales de permisos
Container(
  padding: const EdgeInsets.symmetric(horizontal: 8, vertical: 4),
  decoration: BoxDecoration(
    color: const Color(0xFF4CAF50).withValues(alpha: 0.1),
    borderRadius: BorderRadius.circular(12),
  ),
  child: Text('Solo lectura', style: TextStyle(color: Color(0xFF4CAF50))),
)
```

7. Pantalla de Detalle de Documento (DocumentDetailScreen)

Ubicación: screens/document_detail_screen.dart

Tipo: StatefulWidget con TickerProviderStateMixin

Sistema de Tabs Especializadas

```
TabController _tabController = TabController(length: 3, vsync: this);
```

Tab	Contenido	Funcionalidad
Contenido	Visualización del documento	Preview del texto, soporte multi-formato
Información	Metadatos completos	Propietario, fechas, tipo, permisos
Compartido	Gestión de usuarios	Lista de accesos, añadir/remover permisos

Funcionalidades por Tipo de Archivo

- **Detección automática:** Icons y colores según mimeType
- **Preview contextual:** Contenido de texto cuando disponible
- **Acciones específicas:** Descarga, compartición, edición según tipo

Gestión de Compartición

- **Lista de usuarios:** Propietario + usuarios con acceso
 - **Roles diferenciados:** Propietario vs Solo lectura
 - **Acciones administrativas:** Revocar acceso, cambiar permisos
-

8. Panel de Administración (AdminPanelScreen)

Ubicación: screens/admin_panel_screen.dart

Tipo: StatelessWidget con TickerProviderStateMixin

Sistema de Tabs Administrativas

TabController _tabController = TabController(length: 4, vsync: this);

Tab 1: Dashboard

- **Estadísticas del sistema:** Total usuarios, documentos, chats activos
- **Actividad reciente:** Registro de acciones en tiempo real
- **Métricas visuales:** Tarjetas con iconos y valores destacados

Tab 2: Gestión de Usuarios

```
// Protección especial para cuenta admin principal
if (user.email.toLowerCase() == 'ivan@documento.com') {
  return Container(
    child: Icon(Icons.shield, color: Colors.amber),
  );
}
```

- **CRUD completo:** Crear, editar, eliminar usuarios
- **Protección de cuenta admin:** Ivan no puede ser modificado/eliminado
- **Roles diferenciados:** Indicadores visuales admin/usuario
- **Búsqueda y filtros:** Localización rápida de usuarios

Tab 3: Gestión de Documentos

- **Vista global:** Todos los documentos del sistema
- **Acciones administrativas:** Ver, compartir, eliminar cualquier documento

- **Información de propietario:** Tracking de ownership
- **Indicadores de compartición:** Estados visuales

Tab 4: Gestión de Chats

- **Historial completo:** Todas las conversaciones del sistema
- **Información de usuario:** Asociación chat-usuario
- **Acciones de moderación:** Ver, exportar, eliminar chats
- **Última actividad:** Timestamps y estados

Características de Seguridad

- **Acceso restringido:** Solo usuarios con role: admin
 - **Cuenta protegida:** Ivan no puede ser modificado
 - **Validaciones de permisos:** Verificación de roles en cada acción
 - **Auditoría:** Registro de todas las acciones administrativas
-

9. Pantalla de Perfil (ProfileScreen)

Ubicación: screens/profile_screen.dart

Tipo: StatelessWidget con gestión de formularios

Header Visual Avanzado

```
// Gradiante con múltiples colores y avatar dinámico
Container(
  decoration: BoxDecoration(
    gradient: LinearGradient(
      colors: [Color(0xFF6B4CE6), Color(0xFFE91E63), Color(0xFF2196F3)],
    ),
  ),
  child: // Avatar con badge de admin si corresponde
)
```

Gestión de Información Personal

- **Modo edición:** Activación/desactivación de campos
- **Validación en tiempo real:** Username, email, contraseñas
- **Cambio de contraseña:** Flujo completo con confirmación
- **Estados de carga:** Feedback durante actualización

Secciones Adicionales

- **Ayuda y Soporte:** Modal con información de contacto y FAQ

-
- **Acerca de DocuMente:** Información de la aplicación y características
 - **Cerrar Sesión:** Confirmación y limpieza de estado
-

10. Lista de Chats (ChatListScreen)

Ubicación: screens/chat_list_screen.dart

Tipo: StatelessWidget con búsqueda

Funcionalidades Principales

- **Historial completo:** Todas las conversaciones del usuario
- **Búsqueda semántica:** Filtrado por título y contenido
- **Preview de mensajes:** Último mensaje de cada conversación
- **Navegación directa:** Acceso rápido a conversaciones existentes

Estados Manejados

- **Loading:** Durante carga de conversaciones
 - **Empty:** Sin conversaciones con CTA para crear nueva
 - **Populated:** Lista con metadata y acciones
-

Navegación y Flujos de Usuario

Sistema de Navegación Adaptativo Centralizado

```
class NavigationItem {  
    final IconData icon;  
    final IconData selectedIcon;  
    final String label;  
    final Widget screen;  
}
```

Configuración por Rol

```
void _initializeNavigation() {  
    _navigationItems = [  
        NavigationItem(icon: Icons.home_outlined, label: 'Inicio', screen:  
            _buildDashboard()),  
        NavigationItem(icon: Icons.folder_outlined, label: 'Mis Documentos', screen:  
            DocumentsScreen()),  
        // ... navegación base  
    ];
```

```
// Agregar opciones de administrador
if (user?.isAdmin ?? false) {
  _navigationItems.add(
    NavigationItem(icon: Icons.admin_panel_settings_outlined, label:
'Administración', screen: AdminPanelScreen()),
  );
}
}
```

Transiciones y Animaciones

Tipos de Transiciones Implementadas

- **Slide:** Para navegación entre secciones principales
- **Fade:** Para aparición/desaparición de elementos
- **Scale:** Para acciones destacadas (botones, modales)
- **Coordinated:** Para cambios de estado complejos

Gestión de Estados Globales

```
// Estados comunes en todas las pantallas
enum ScreenState { loading, loaded, empty, error }

// Manejo consistente de loading
Widget build(BuildContext context) {
  return _isLoading
    ? Center(child: CircularProgressIndicator(color: Color(0xFF6B4CE6)))
    : _buildContent();
}
```

Aspectos Técnicos de Implementación

Características Técnicas Transversales

1. Responsividad Sistématica

```
// Sistema unificado de breakpoints
class ResponsiveBreakpoints {
  static const double mobile = 600;
  static const double tablet = 900;
  static const double desktop = 1200;
}

// Uso consistente en todas las pantallas
```

```
final isSmallScreen = MediaQuery.of(context).size.width <  
ResponsiveBreakpoints.mobile;
```

2. Gestión de Estado Centralizada

- **Provider Pattern:** Estado reactivo con ChangeNotifier
- **Consumer Widgets:** Actualización automática de UI
- **Estado persistente:** Mantenimiento entre navegación

3. Optimizaciones de Rendimiento

```
// Uso de const constructors  
const NavigationItem({  
    required this.icon,  
    required this.selectedIcon,  
    required this.label,  
    required this.screen,  
});
```

```
// ListView.builder para listas grandes  
ListView.builder(  
    itemCount: _filteredDocuments.length,  
    itemBuilder: (context, index) =>  
    _buildDocumentCard(_filteredDocuments[index]),  
)
```

4. Manejo de Ciclo de Vida

```
@override  
void initState() {  
    super.initState();  
    _animationController = AnimationController(duration: Duration(milliseconds:  
800), vsync: this);  
    _loadData();  
}
```

```
@override  
void dispose() {  
    _animationController.dispose();  
    _scrollController.dispose();  
    super.dispose();  
}
```

Patrones de Diseño Implementados

1. Factory Pattern en Widgets

```
Widget _buildStatCard(String title, String value, IconData icon, Color color) {  
    return Container(/* configuración común */);
```

```
}
```

2. Strategy Pattern para Navegación

```
Widget _getNavigationWidget(double screenWidth) {  
  return screenWidth < 600  
    ? BottomNavigationBar(/* config móvil */)  
    : NavigationRail(/* config tablet/desktop */);  
}
```

3. Observer Pattern con Provider

```
Consumer<AuthService>(  
  builder: (context, authService, child) {  
    return authService.isAuthenticated ? HomeScreen() : LoginScreen();  
  },  
)
```

Estados y Feedback Visual

Estados Manejados Consistentemente

Estado	Implementación	Feedback Visual
Loading	CircularProgressIndicator	Spinner con color corporativo
Empty	Custom illustrations	Iconos, mensajes, CTAs
Error	SnackBar + retry options	Colores de alerta, acciones
Success	SnackBar + visual feedback	Colores de éxito, animaciones

Microinteracciones

- **Hover effects:** Cambios sutiles en cards y botones
 - **Ripple effects:** Feedback táctil en Material Design
 - **Transitions:** Suavizado entre estados
 - **Loading states:** Skeletons y progressive loading
-

Accesibilidad y Usabilidad

Características de Accesibilidad

- **Semantic labels:** Todos los elementos interactivos
- **Color contrast:** Cumplimiento WCAG 2.1 AA
- **Touch targets:** Mínimo 44px en dispositivos táctiles

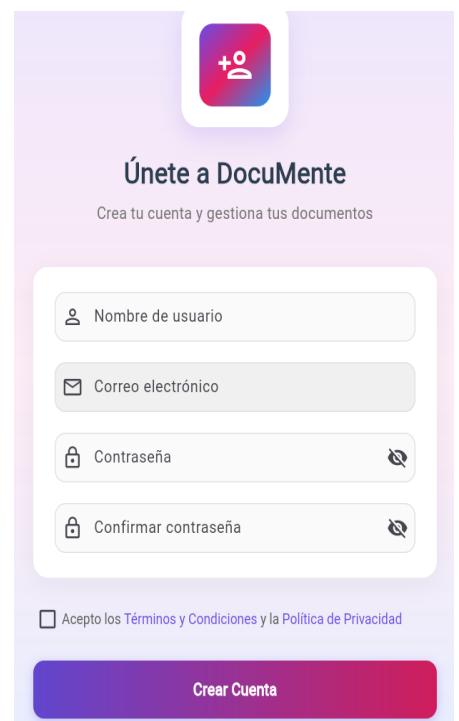
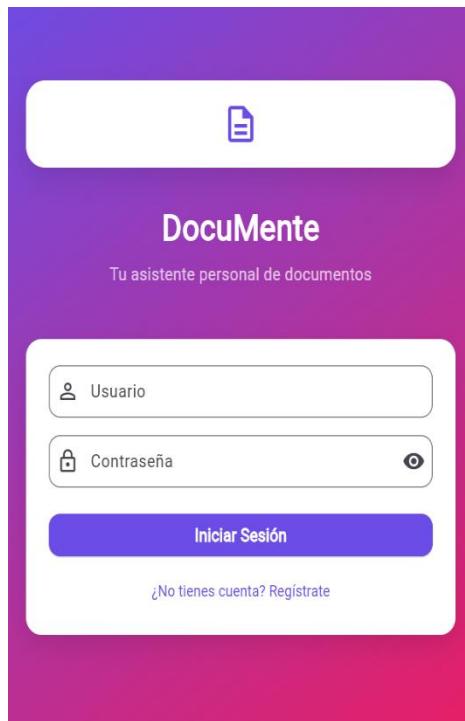
- **Keyboard navigation:** Soporte completo para teclado

Optimizaciones de UX

- **Error prevention:** Validación en tiempo real
- **Clear feedback:** Estados visibles y comprensibles
- **Consistent patterns:** Mismos patrones en toda la app
- **Progressive disclosure:** Información mostrada gradualmente

2.5 Diseño y experiencia de usuario

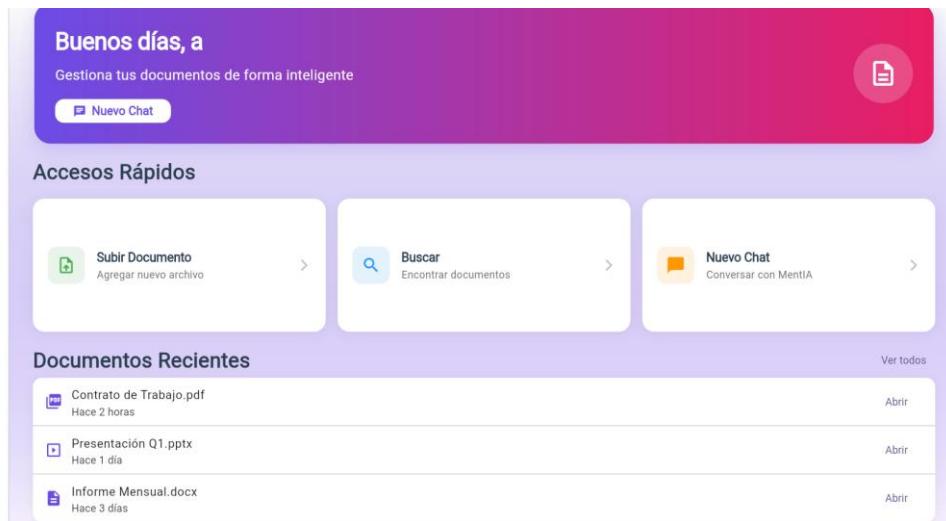
Tanto usuario normal, como usuario administrador ven esta pantalla excepto que nunca te hayas registrado.



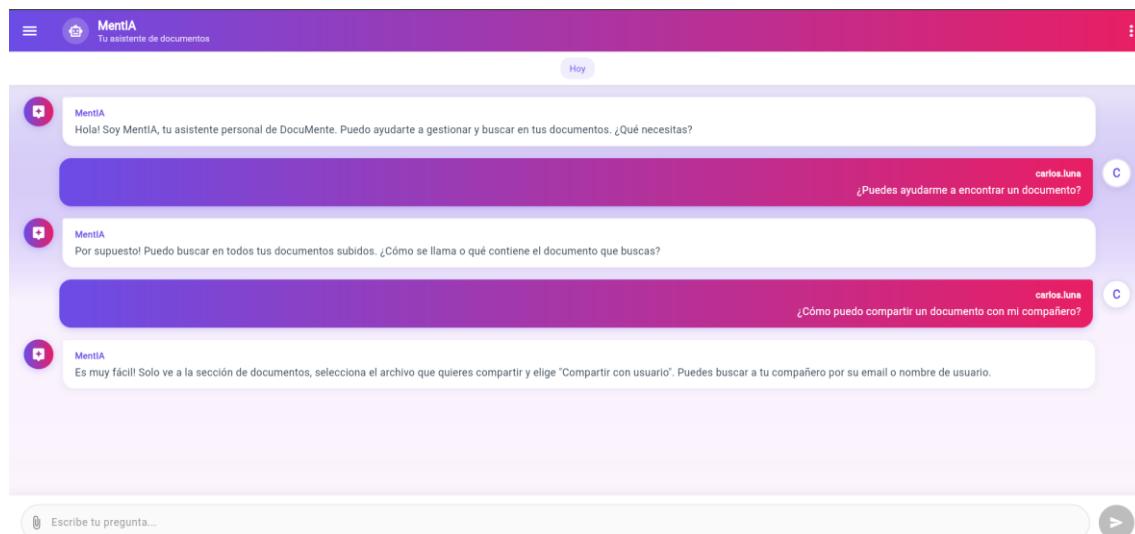
Una vez iniciado sesión si eres Iván [el jefe que nos mandó el producto mínimo viable] tienes, la sección Administración:

The screenshot shows the DocuMente dashboard for an administrator (Ivan). The top navigation bar is purple with the text "Buenos días, Ivan" and "Panel de administrador de DocuMente". A "Nuevo Chat" button is also present. The left sidebar has a "ADMIN" badge next to the "Administración" icon. The main content area is divided into sections: "Accesos Rápidos" (with buttons for "Subir Documento", "Buscar", and "Nuevo Chat"), "Estadísticas del Sistema" (with a sub-section titled "Panel de Administración" showing "Dashboard", "Usuarios", "Documentos", and "Chats" tabs), and "Actividad Reciente" (listing recent events like "Nuevo usuario registrado" by Roberto Silva, "Documento compartido" by Ana Garcia, and "Chat iniciado" by Carlos Ruiz, each with a timestamp).

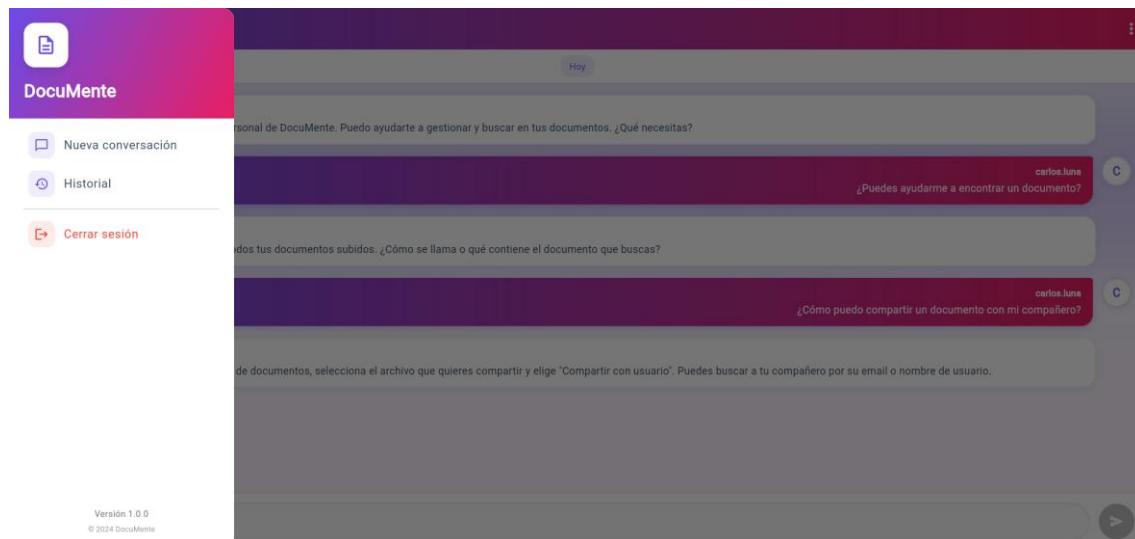
Si eres usuario normal, accedes al mismo dashboard solo que sin la sección.



Desde donde ambos podrán acceder a distintas funcionalidades:



Un chatbot para hacer preguntas, donde clicando en las tres rayas accedes a:



Mis Documentos

5 documentos

Buscar documentos...

Todos Compartidos PDF TXT

Contrato de Trabajo
contrato_trabajo.pdf
Hace 2 horas

Notas de Reunión
notas_reunion.txt
Ayer 2 usuarios

Instrucciones de Uso
instrucciones.txt
Hace 3 días

Manual de Usuario
manual_usuario.pdf
10/5/2025

Ver Compartir Descargar Eliminar

+ Subir Documento

Documentos Compartidos

2 documentos compartidos conmigo

Buscar documentos compartidos...

Todos PDF TXT

Lineamientos Corporativos
lineamientos.txt
Compartido por Dirección General
Hace 2 días 2 usuarios

Instrucciones de Configuración
configuracion_sistema.txt
Compartido por TI
18/5/2025 2 usuarios

Compartido Solo lectura

Compartido Solo lectura

 Buscar documentos compartidos...

 Todos

PDF

TXT

Política de la Empresa



politica_empresa.pdf

 Compartido por Recursos Humanos

 Hace 5 días  3 usuarios

 Compartido

 Ver

 Descargar

 Quitar de compartidos

Solo lectura

Lineamientos Corporativos



lineamientos.txt

 Compartido por Dirección General

 Hace 2 días  2 usuarios

← Contrato de Trabajo

 Contenido

 Información

 Compartido



contrato_trabajo.pdf

Documento PDF

Contenido del documento:

Contenido del contrato...

 Compartir

← Contrato de Trabajo

Información

Información del Documento

- Título**
Contrato de Trabajo
- Nombre del archivo**
contrato_trabajo.pdf
- Tipo de archivo**
Documento PDF
- Propietario**
angel
- Fecha de creación**
Hoy 18:47
- Última modificación**
Hoy 18:47
- Estado**
Privado

Compartir

← Contrato de Trabajo

Información

Compartido

Usuarios con Acceso

Añadir

- angel**
angel@empresa.com
Propietario
- Ana Garcia**
ana.garcia@empresa.com
Propietario
- Carlos Ruiz**
carlos.ruiz@empresa.com
Solo lectura

Compartir

Mis Conversaciones

5 conversaciones con MentiA

Buscar en conversaciones...

- Consulta sobre contratos**
MentiA: Por supuesto, puedo ayudarte con la revisión de contratos. ¿Hay algo específico que quieras revisar?
🕒 Hace 30 minutos 2 mensajes
- Búsqueda de documentos**
MentiA: He encontrado varios informes en tu carpeta. ¿Te refieres al informe 2024?
🕒 Hace 8 horas 2 mensajes
- Análisis de datos financieros**
MentiA: Los resultados del último trimestre muestran un crecimiento del 15% en comparación con el trimestre anterior.
🕒 Ayer 2 mensajes
- Gestión de permisos**
MentiA: Para cambiar permisos, ve a la sección de documentos, selecciona el archivo y elige "Gestionar acceso".

+ Nuevo Chat

angel@documenteme.com

Ayuda y Soporte

¿Necesitas ayuda con DocuMente?

Información de contacto

- Centro de ayuda: help.documente.com
- Email: soporte@documente.com
- Teléfono: +1 (555) 123-4567

Horario de atención

- Lunes a Viernes, 9:00 - 18:00 (UTC-5)
- Sábados: 10:00 - 14:00 (Emergencias)

Preguntas frecuentes

- ¿Cómo subir documentos grandes?
- ¿Cómo compartir documentos con usuarios externos?
- ¿Cómo recuperar versiones anteriores de documentos?
- ¿Cómo usar el chat asistido por IA?

Soluciones rápidas

- Problema de sincronización: Cierre sesión y vuelva a iniciar
- Documentos no visibles: Verifique permisos
- Búsqueda semántica no funciona: Verifique la conexión

Para asistencia inmediata con cuentas corporativas, contacte a su administrador de TI interno.

Cerrar

angel@documenteme.com

Acerca de DocuMente

 **DocuMente**
Versión 1.0.0

Tu asistente inteligente para la gestión de documentos potenciado por IA

Gestión de Documentos
Sube, organiza y accede a tus documentos desde cualquier dispositivo con seguridad y eficiencia.

Búsqueda Semántica
Encuentra documentos por su contenido, no solo por el título, gracias a nuestro motor de búsqueda semántica avanzado.

Compartición Inteligente
Comparte documentos con otros usuarios y establece permisos personalizados de forma sencilla.

Chat Asistido por IA
Pregunta a tu asistente virtual sobre el contenido de tus documentos y obtén respuestas precisas.

Seguridad Avanzada
Tus documentos están protegidos con cifrado de extremo a extremo y autenticación segura.

© 2024 DocuMente Inc. Todos los derechos reservados.

Cerrar

DocuMente implementa una filosofía de diseño centrada en el usuario que combina elegancia visual con funcionalidad práctica, construida sobre los principios de Material Design 3 con personalizaciones específicas que crean una identidad visual única. El sistema de diseño busca resolver los desafíos específicos de una aplicación de gestión documental potenciada por IA, donde la claridad, la jerarquía informativa y la accesibilidad son fundamentales para garantizar una experiencia fluida y satisfactoria.

Principios Fundamentales del Diseño

1. **Claridad y Simplicidad:** Interfaces limpias que reducen la carga cognitiva
2. **Consistencia Visual:** Patrones reutilizables en toda la aplicación
3. **Retroalimentación Inmediata:** Feedback visual y animaciones significativas
4. **Accesibilidad Universal:** Diseño inclusivo para todos los usuarios
5. **Adaptabilidad Responsiva:** Experiencia optimizada en cualquier dispositivo

SISTEMA DE DISEÑO IMPLEMENTADO

1. IDENTIDAD VISUAL Y BRANDING

Logo y Marca

dart

// Implementación del logo en múltiples contextos

```
Container(  
  width: 80,  
  height: 80,  
  decoration: BoxDecoration(  
    color: Colors.white,  
    borderRadius: BorderRadius.circular(20),  
    boxShadow: [  
      BoxShadow(  
        color: Colors.black.withOpacity(0.1),  
        blurRadius: 20,  
        offset: const Offset(0, 10),  
      ),  
    ],
```

```
),
child: const Icon(
  Icons.description_outlined,
  size: 40,
  color: Color(0xFF6B4CE6),
),
)
```

El logo utiliza el ícono `description_outlined` como elemento central, representando la naturaleza documental de la aplicación. Se presenta en diferentes contextos con variaciones de tamaño y tratamiento visual.

2. SISTEMA CROMÁTICO COMPLETO

Paleta Principal con Gradientes

```
dart
// Gradiante principal usado consistentemente
decoration: const BoxDecoration(
  gradient: LinearGradient(
    colors: [
      Color(0xFF6B4CE6), // Morado vibrante
      Color(0xFFE91E63), // Rosa intenso
    ],
    begin: Alignment.topLeft,
    end: Alignment.bottomRight,
),
)
```

// Gradiante extendido para efectos especiales

```
LinearGradient(
  colors: [
    Color(0xFF6B4CE6), // Morado
    Color(0xFFE91E63), // Rosa
    Color(0xFF2196F3), // Azul (usado en ProfileScreen)
  ]
)
```

```
],  
)  
Colores Funcionales por Contexto  
dart  
// Colores de estado implementados  
class AppColors {  
    // Estados de éxito  
    static const success = Color(0xFF4CAF50);  
    static final successLight = success.withOpacity(0.1);  
  
    // Estados de información  
    static const info = Color(0xFF2196F3);  
    static final infoLight = info.withOpacity(0.1);  
  
    // Estados de advertencia  
    static const warning = Color(0xFFFF9800);  
    static final warningLight = warning.withOpacity(0.1);  
  
    // Estados de error  
    static const error = Color(0xFFF44336);  
    static const errorAlt = Colors.red;  
  
// Colores administrativos  
    static const admin = Color(0xFF9C27B0);  
    static const adminBadge = Color(0xFFFFD700); // Dorado para badge admin  
  
// Colores de archivo por tipo  
    static const pdfColor = Color(0xFFD32F2F);  
    static const txtColor = Color(0xFF607D8B);  
    static const excelColor = Color(0xFF388E3C);  
    static const wordColor = Color(0xFF1976D2);  
    static const powerpointColor = Color(0xFFFF57C00);
```

```
}
```

Sistema de Opacidades

```
dart
```

```
// Implementación de opacidades con el nuevo API withValues
```

```
color.withValues(alpha: 0.1) // 10% - Fondos muy sutiles
```

```
color.withValues(alpha: 0.2) // 20% - Fondos sutiles, bordes
```

```
color.withValues(alpha: 0.3) // 30% - Sombras, overlays ligeros
```

```
color.withValues(alpha: 0.5) // 50% - Estados deshabilitados
```

```
color.withValues(alpha: 0.8) // 80% - Texto secundario sobre fondos
```

```
color.withValues(alpha: 0.95) // 95% - Texto casi opaco
```

3. SISTEMA TIPOGRÁFICO RESPONSIVO

Implementación de Escalas Tipográficas

```
'caption': 16.0,  
},  
};  
  
String screenType;  
if (screenWidth >= 900) {  
    screenType = 'large';  
} else if (screenWidth >= 600) {  
    screenType = 'medium';  
} else {  
    screenType = 'small';  
}  
  
final fontSize = fontSizes[screenType]![type] ?? 16.0;  
  
switch (type) {  
    case 'title':  
        return TextStyle(  
            fontSize: fontSize,  
            fontWeight: FontWeight.bold,  
            color: const Color(0xFF2C3E50),  
            letterSpacing: -0.5, // Compresión para títulos  
        );  
    case 'subtitle':  
        return TextStyle(  
            fontSize: fontSize,  
            fontWeight: FontWeight.w600,  
            color: const Color(0xFF2C3E50),  
            height: 1.3,  
        );  
    case 'body':  
        return TextStyle(  
            fontSize: 16.0,  
            color: const Color(0xFF2C3E50),  
            height: 1.3,  
        );  
}
```

```
    fontSize: fontSize,
    fontWeight: FontWeight.normal,
    color: const Color(0xFF2C3E50),
    height: 1.4, // Mejor legibilidad
  );
case 'caption':
  return TextStyle(
    fontSize: fontSize,
    fontWeight: FontWeight.normal,
    color: Colors.grey[600],
  );
default:
  return TextStyle(fontSize: fontSize);
}
}
```

Casos de Uso Específicos
dart

```
// Títulos grandes en headers
Text(
  'Panel de Administración',
  style: TextStyle(
    fontSize: isTablet ? 22 : 18,
    fontWeight: FontWeight.bold,
    color: Colors.white,
  ),
)
```

```
// Texto descriptivo con jerarquía
Text(
  'Gestión completa del sistema DocuMente',
  style: TextStyle(
    fontSize: isTablet ? 14 : 12,
```

```
        color: Colors.white.withAlpha(230),  
    ),  
    overflow: TextOverflow.ellipsis,  
)
```

// Badges y etiquetas

```
Text(  
    'ADMIN',  
    style: TextStyle(  
        color: Colors.white,  
        fontSize: 8,  
        fontWeight: FontWeight.bold,  
    ),  
)
```

4. SISTEMA DE ESPACIADO Y LAYOUT

Espaciado Responsivo

dart

```
EdgeInsets _getResponsivePadding(double screenWidth) {  
    if (screenWidth < 600) {  
        return const EdgeInsets.symmetric(horizontal: 16, vertical: 8);  
    } else if (screenWidth < 900) {  
        return const EdgeInsets.symmetric(horizontal: 24, vertical: 12);  
    } else {  
        return const EdgeInsets.symmetric(horizontal: 32, vertical: 16);  
    }  
}
```

```
double _getResponsiveSpacing(double screenWidth) {  
    if (screenWidth < 600) {  
        return 10.0;  
    } else if (screenWidth < 900) {
```

```
        return 14.0;
    } else {
        return 18.0;
    }
}

Sistema de Grid Adaptativo
dart

// Grid responsivo para tarjetas
GridView.builder(
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: isDesktop ? 3 : (isTablet ? 2 : 1),
        childAspectRatio: isSmallScreen ? 3.0 : 2.2,
        crossAxisSpacing: 16,
        mainAxisSpacing: 16,
    ),
    // ...
)

// Uso de Wrap para layouts flexibles
Wrap(
    spacing: 16,
    runSpacing: 16,
    children: stats.map((stat) {
        final crossAxisCount = isDesktop ? 3 : (isTablet ? 2 : 1);
        final itemWidth = (screenWidth - (padding * 2) - (16 * (crossAxisCount - 1))) /
        crossAxisCount;

        return SizedBox(
            width: itemWidth,
            child: _buildStatCard(stat),
        );
    }).toList(),
)
```

)

5. COMPONENTES UI REUTILIZABLES

Tarjetas (Cards) con Elevación

dart

// Tarjeta estándar con sombra sutil

Container(

decoration: BoxDecoration(

color: Colors.white,

borderRadius: BorderRadius.circular(16),

boxShadow: [

BoxShadow(

color: Colors.black.withAlpha(13),

blurRadius: 10,

offset: const Offset(0, 2),

),

],

),

child: content,

)

// Tarjeta con gradiente y elevación pronunciada

Container(

decoration: BoxDecoration(

gradient: const LinearGradient(

colors: [Color(0xFF6B4CE6), Color(0xFFE91E63)],

),

borderRadius: BorderRadius.circular(isSmallScreen ? 16 : 22),

boxShadow: [

BoxShadow(

color: const Color(0xFF6B4CE6).withValues(alpha: 0.2),

blurRadius: isSmallScreen ? 15 : 22,

```
        offset: const Offset(0, 6),
    ),
],
),
)
}

Sistema de Botones
dart

// Botón elevado con gradiente
ElevatedButton(
    onPressed: () {},
    style: ElevatedButton.styleFrom(
        backgroundColor: const Color(0xFF6B4CE6),
        foregroundColor: Colors.white,
        padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 12),
        shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(12),
        ),
        elevation: 0,
    ),
    child: const Text('Acción'),
)

// Botón con ícono adaptativo
SizedBox(
    height: 36,
    child: ElevatedButton.icon(
        onPressed: _showAddUserDialog,
        icon: const Icon(Icons.add, size: 18),
        label: Text(screenWidth > 360 ? 'Nuevo Usuario' : 'Nuevo'),
        style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFF6B4CE6),
            shape: RoundedRectangleBorder(

```

```
borderRadius: BorderRadius.circular(18),  
),  
,  
,  
)  
  
// Botón de acción flotante con gradiente  
AnimatedContainer(  
duration: const Duration(milliseconds: 200),  
child: Container(  
decoration: BoxDecoration(  
gradient: _hasText  
? const LinearGradient(  
colors: [Color(0xFF6B4CE6), Color(0xFFE91E63)],  
)  
: null,  
color: _hasText ? null : Colors.grey[400],  
borderRadius: BorderRadius.circular(24),  
boxShadow: _hasText  
?[  
BoxShadow(  
color: const Color(0xFF6B4CE6).withValues(alpha: 0.3),  
blurRadius: 8,  
offset: const Offset(0, 2),  
)  
,  
]  
: null,  
,  
,  
)  
Campos de Entrada  
dart
```

```
// TextField con diseño personalizado
TextField(
    decoration: InputDecoration(
        labelText: 'Buscar documentos...',
        prefixIcon: const Icon(Icons.search),
        border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(12),
            borderSide: BorderSide(color: Colors.grey[300]!),
        ),
        enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(12),
            borderSide: BorderSide(color: Colors.grey[300]!),
        ),
        focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(12),
            borderSide: const BorderSide(color: Color(0xFF6B4CE6), width: 2),
        ),
        filled: true,
        fillColor: Colors.grey[50],
    ),
)
```

```
// Campo de chat con diseño integrado
Container(
    decoration: BoxDecoration(
        color: Colors.grey[50],
        borderRadius: BorderRadius.circular(25),
        border: Border.all(
            color: Colors.grey[300]!,
            width: 1,
        ),
),
```

```
child: Row(  
    children: [  
        IconButton(  
            icon: Icon(Icons.attach_file_rounded, color: Colors.grey[600], size: 20),  
            onPressed: () {}),  
        ),  
        Expanded(  
            child: TextField(  
                decoration: const InputDecoration(  
                    hintText: 'Escribe tu pregunta...'),  
                border: InputBorder.none,  
            ),  
        ),  
    ],  
,  
)  
  
Chips y Badges  
dart  
// FilterChip para filtrado  
FilterChip(  
    label: Text(filter),  
    selected: _selectedFilter == filter,  
    onSelected: (selected) {},  
    selectedColor: const Color(0xFF6B4CE6).withValues(alpha: 0.2),  
    checkmarkColor: const Color(0xFF6B4CE6),  
)  
  
// Badge de estado  
Container(  
    padding: const EdgeInsets.symmetric(horizontal: 8, vertical: 4),  
    decoration: BoxDecoration(  
        color: Colors.white,
```

```
        color: const Color(0xFF4CAF50).withValues(alpha: 0.1),
        borderRadius: BorderRadius.circular(12),
    ),
    child: const Row(
        mainAxisSize: MainAxisSize.min,
        children: [
            Icon(Icons.share, size: 12, color: Color(0xFF4CAF50)),
            SizedBox(width: 4),
            Text(
                'Compartido',
                style: TextStyle(
                    fontSize: 10,
                    color: Color(0xFF4CAF50),
                    fontWeight: FontWeight.w500,
                ),
            ),
        ],
    ),
)
```

// Badge de administrador con efecto dorado

```
Container(
    padding: const EdgeInsets.all(6),
    decoration: const BoxDecoration(
        color: Color(0xFFFFD700),
        shape: BoxShape.circle,
    ),
    child: const Icon(
        Icons.admin_panel_settings,
        color: Colors.white,
        size: 16,
    ),
)
```

)

6. ICONOGRAFÍA Y SISTEMA DE ICONOS

Contenedores de Iconos

dart

// Icono con fondo semitransparente

Container(

padding: const EdgeInsets.all(8),

decoration: BoxDecoration(

color: color.withValues(alpha: 0.12),

borderRadius: BorderRadius.circular(10),

),

child: Icon(

icon,

color: color,

size: 20,

),

)

// Avatar con gradiente

Container(

width: 40,

height: 40,

decoration: BoxDecoration(

gradient: const LinearGradient(

colors: [Color(0xFF6B4CE6), Color(0xFFE91E63)],

),

borderRadius: BorderRadius.circular(20),

boxShadow: [

BoxShadow(

color: const Color(0xFF6B4CE6).withValues(alpha: 0.3),

blurRadius: 8,

```
        offset: const Offset(0, 2),
    ),
],
),
child: const Icon(
Icons.assistant,
color: Colors.white,
size: 20,
),
)
}

Sistema de Iconos por Tipo de Archivo
dart

IconData _getFileTypeIcon(String mimeType) {
if (mimeType.contains('pdf')) return Icons.picture_as_pdf;
if (mimeType.contains('word') || mimeType.contains('document')) {
return Icons.description;
}
if (mimeType.contains('spreadsheet') || mimeType.contains('excel')) {
return Icons.table_chart;
}
if (mimeType.contains('presentation') || mimeType.contains('powerpoint')) {
return Icons.slideshow;
}
if (mimeType.contains('text/plain')) return Icons.insert_drive_file;
return Icons.insert_drive_file;
}

Color _getFileTypeColor(String mimeType) {
if (mimeType.contains('pdf')) return const Color(0xFFD32F2F);
if (mimeType.contains('word')) return const Color(0xFF1976D2);
if (mimeType.contains('spreadsheet')) return const Color(0xFF388E3C);
if (mimeType.contains('presentation')) return const Color(0xFFFF57C00);
}
```

```
if (mimeType.contains('text/plain')) return const Color(0xFF607D8B);
return const Color(0xFF6B4CE6);
}
```

PATRONES DE NAVEGACIÓN IMPLEMENTADOS

1. Navegación Adaptativa Principal

```
dart
// Sistema de navegación que se adapta al dispositivo
Scaffold(
    // BottomNavigationBar para móviles
    bottomNavigationBar: isSmallScreen
        ? BottomNavigationBar(
            currentIndex: _selectedIndex,
            onTap: (index) => setState(() => _selectedIndex = index),
            selectedItemColor: const Color(0xFF6B4CE6),
            unselectedItemColor: Colors.grey[600],
            type: BottomNavigationBarType.fixed,
            items: _navigationItems
                .map((item) => BottomNavigationBarItem(
                    icon: Icon(item.icon),
                    activeIcon: Icon(item.selectedIcon),
                    label: item.label,
                ))
                .toList(),
        )
        : null,
```

```
// NavigationRail para tablets y desktop
body: Row(
    children: [
        if (!isSmallScreen) ...[
            NavigationRail(
```

```
extended: !isMediumScreen, // Expandido solo en desktop
selectedIndex: _selectedIndex,
onDestinationSelected: (index) => setState(() => _selectedIndex = index),
labelType: isMediumScreen
    ? NavigationRailLabelType.selected
    : NavigationRailLabelType.none,
backgroundColor: Colors.white,
selectedIconTheme: const IconThemeData(
    color: Color(0xFF6B4CE6),
    size: 28,
),
unselectedIconTheme: IconThemeData(
    color: Colors.grey[600],
    size: 24,
),
// Personalización adicional
leading: _buildNavigationRailHeader(),
trailing: _buildNavigationRailFooter(),
destinations: _navigationItems.map((item) =>
    NavigationRailDestination(
        icon: Icon(item.icon),
        selectedIcon: Icon(item.selectedIcon),
        label: Text(item.label),
    ),
).toList(),
),
const VerticalDivider(thickness: 1, width: 1),
],
Expanded(child: _navigationItems[_selectedIndex].screen),
],
),
)
```

2. Navegación por Tabs

```
dart
// Sistema de tabs en AdminPanelScreen
TabBar(
  controller: _tabController,
  labelColor: const Color(0xFF6B4CE6),
  unselectedLabelColor: Colors.grey[600],
  indicatorColor: const Color(0xFF6B4CE6),
  isScrollable: screenWidth < 500,
  labelStyle: TextStyle(fontSize: isTablet ? 12 : 10),
  tabs: [
    Tab(
      icon: Icon(Icons.dashboard, size: isTablet ? 20 : 18),
      text: screenWidth > 360 ? 'Dashboard' : 'Inicio',
    ),
    Tab(
      icon: Icon(Icons.people, size: isTablet ? 20 : 18),
      text: 'Usuarios',
    ),
    // ... más tabs
  ],
)
```

3. Navegación Contextual

```
dart
// Drawer personalizado en pantalla de chat
Drawer(
  backgroundColor: Colors.white,
  child: Column(
    children: [
      // Header con gradiente
      Container(
```

```

decoration: const BoxDecoration(
  gradient: LinearGradient(
    colors: [Color(0xFF6B4CE6), Color(0xFFE91E63)],
  ),
),
child: SafeArea(
  child: Padding(
    padding: const EdgeInsets.all(20.0),
    child: _buildDrawerHeader(),
  ),
),
),
),
// Opciones del drawer
Expanded(
  child: _buildDrawerOptions(),
),
],
),
)

```

MICROINTERACCIONES Y ANIMACIONES

1. Sistema de Animaciones

```

dart
// Animación de fade para aparición de contenido
late AnimationController _animationController;
late Animation<double> _fadeAnimation;

@Override
void initState() {
  super.initState();
  _animationController = AnimationController(

```

```

duration: const Duration(milliseconds: 800),
vsync: this,
);

_fadeAnimation = Tween<double>(
begin: 0.0,
end: 1.0,
).animate(CurvedAnimation(
parent: _animationController,
curve: Curves.easeInOut,
));
}

_animationController.forward();
}

```

// Uso en widget

```

FadeTransition(
  opacity: _fadeAnimation,
  child: _buildContent(),
)

```

2. Transiciones de Estado

```

dart
// Contenedor animado para cambios de estado
AnimatedContainer(
  duration: const Duration(milliseconds: 200),
  child: Container(
    decoration: BoxDecoration(
      gradient: _hasText
        ? const LinearGradient(
          colors: [Color(0xFF6B4CE6), Color(0xFFE91E63)],
        )

```

```
: null,  
color: _hasText ? null : Colors.grey[400],  
,  
,  
)
```

// Animación de escala para botones

```
Transform.scale(  
scale: _isPressed ? 0.95 : 1.0,  
child: ElevatedButton(...),  
)
```

3. Indicadores de Progreso

```
dart  
// Indicador de carga personalizado  
_isLoading  
? const Center(  
    child: CircularProgressIndicator(  
        color: Color(0xFF6B4CE6),  
    ),  
)  
: _buildContent()
```

// Indicador en botón

```
_isLoading  
? const SizedBox(  
    height: 20,  
    width: 20,  
    child: CircularProgressIndicator(  
        color: Colors.white,  
        strokeWidth: 2,  
)
```

```
)  
: const Text('Acción')
```

COMPONENTES DE CHAT ESPECÍFICOS

1. Mensajes de Usuario

```
dart  
  
class UserMessage extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Container(  
      margin: const EdgeInsets.only(bottom: 16, left: 60, right: 12),  
      child: Row(  
        crossAxisAlignment: CrossAxisAlignment.start,  
        children: [  
          Expanded(  
            child: Container(  
              padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),  
              decoration: BoxDecoration(  
                gradient: const LinearGradient(  
                  colors: [Color(0xFF6B4CE6), Color(0xFFE91E63)],  
                ),  
                borderRadius: const BorderRadius.only(  
                  topLeft: Radius.circular(16),  
                  topRight: Radius.circular(4),  
                  bottomLeft: Radius.circular(16),  
                  bottomRight: Radius.circular(16),  
                ),  
                boxShadow: [  
                  BoxShadow(  
                    color: const Color(0xFF6B4CE6).withValues(alpha: 0.3),  
                    blurRadius: 8,  
                    offset: const Offset(0, 2),  
                  ),  
                ],  
              ),  
            ),  
          ),  
        ],  
      ),  
    );  
  }  
}
```

```
        ),
      ],
    ),
  child: _buildMessageContent(),
),
),
),
const SizedBox(width: 12),
_buildUserAvatar(),
],
),
);
}
}
```

2. Mensajes del Bot

```
dart
class BotMessage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      margin: const EdgeInsets.only(bottom: 16, left: 12, right: 60),
      child: Row(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          _buildBotAvatar(),
          const SizedBox(width: 12),
          Expanded(
            child: Container(
              padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: const BorderRadius.only(

```

```
        topLeft: Radius.circular(4),
        topRight: Radius.circular(16),
        bottomLeft: Radius.circular(16),
        bottomRight: Radius.circular(16),
    ),
    boxShadow: [
        BoxShadow(
            color: Colors.black.withValues(alpha: 0.05),
            blurRadius: 8,
            offset: const Offset(0, 2),
        ),
    ],
    border: Border.all(
        color: const Color(0xFF6B4CE6).withValues(alpha: 0.1),
        width: 1,
    ),
),
),
],
),
),
);
}
}
```

3. Input de Chat

```
dart
class ChatInputBox extends StatefulWidget {
    @override
    Widget build(BuildContext context) {
        return Container(
```

```
padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),  
decoration: BoxDecoration(  
    color: Colors.white,  
    boxShadow: [  
        BoxShadow(  
            color: Colors.black.withValues(alpha: 0.05),  
            blurRadius: 10,  
            offset: const Offset(0, -2),  
        ),  
    ],  
),  
child: SafeArea(  
    child: Row(  
        children: [  
            Expanded(  
                child: Container(  
                    decoration: BoxDecoration(  
                        color: Colors.grey[50],  
                        borderRadius: BorderRadius.circular(25),  
                        border: Border.all(color: Colors.grey[300]!, width: 1),  
                    ),  
                child: _buildInputField(),  
            ),  
        ),  
        const SizedBox(width: 12),  
        _buildSendButton(),  
    ],  
),  
);  
}  
}
```

ESTADOS ESPECIALES Y FEEDBACK

1. Estados Vacíos

```
dart
Widget _buildEmptyState() {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Container(
          width: 120,
          height: 120,
          decoration: BoxDecoration(
            color: Colors.grey[100],
            borderRadius: BorderRadius.circular(60),
          ),
        ),
        child: Icon(
          Icons.description_outlined,
          size: 60,
          color: Colors.grey[400],
        ),
      ],
    ),
    const SizedBox(height: 24),
    Text(
      'No tienes documentos aún',
      style: const TextStyle(
        fontSize: 18,
        fontWeight: FontWeight.w500,
        color: Color(0xFF2C3E50),
      ),
    ),
    const SizedBox(height: 8),
```

```
        Text(
            'Sube tu primer documento para comenzar',
            style: TextStyle(
                fontSize: 14,
                color: Colors.grey[600],
            ),
        ),
    ],
),
),
);
}
```

2. Sistema de Feedback con SnackBar

```
dart
// Snackbar de éxito
ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
        content: Text('Documento subido exitosamente'),
        backgroundColor: Color(0xFF4CAF50),
    ),
);

// Snackbar de error
ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
        content: Text('Error al actualizar el perfil'),
        backgroundColor: Colors.red,
    ),
);

// Snackbar informativo
ScaffoldMessenger.of(context).showSnackBar(
```

```
SnackBar(  
    content: Text('Descargando ${document.fileName}...'),  
    backgroundColor: const Color(0xFF2196F3),  
,  
);
```

3. Diálogos y Modales

```
dart  
// Diálogo con diseño personalizado  
 showDialog(  
     context: context,  
     builder: (context) => Dialog(  
         child: Container(  
             constraints: BoxConstraints(  
                 maxWidth: screenWidth > 600 ? 500 : double.infinity,  
,  
             padding: const EdgeInsets.all(16),  
             child: Column(  
                 mainAxisAlignment: MainAxisAlignment.min,  
                 crossAxisAlignment: CrossAxisAlignment.start,  
                 children: [  
                     const Text(  
                         'Título del Diálogo',  
                         style: TextStyle(  
                             fontSize: 20,  
                             fontWeight: FontWeight.bold,  
,  
,  
                     ),  
                     const SizedBox(height: 16),  
// Contenido del diálogo  
                     const SizedBox(height: 24),  
                     Row(  
             ),  
         ),  
     ),  
 );
```

```

mainAxisAlignment: MainAxisAlignment.end,
children: [
    TextButton(
        onPressed: () => Navigator.pop(context),
        child: const Text('Cancelar'),
    ),
    const SizedBox(width: 8),
    ElevatedButton(
        onPressed: () {},
        style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFF6B4CE6),
            foregroundColor: Colors.white,
        ),
        child: const Text('Confirmar'),
    ),
],
),
],
),
),
),
),
);

```

DISEÑO RESPONSIVO AVANZADO

1. Sistema de Breakpoints

```

dart
class ResponsiveBreakpoints {
    static const double small = 600; // < 600px: Móviles
    static const double medium = 900; // 600-900px: Tablets
    static const double large = 1200; // > 900px: Desktop

    static bool isSmallScreen(BuildContext context) {

```

```

        return MediaQuery.of(context).size.width < small;
    }

static bool isMediumScreen(BuildContext context) {
    final width = MediaQuery.of(context).size.width;
    return width >= small && width < medium;
}

static bool isLargeScreen(BuildContext context) {
    return MediaQuery.of(context).size.width >= medium;
}

```

2. Layouts Adaptativos

dart

```

// Uso de LayoutBuilder para decisiones de layout
LayoutBuilder(
    builder: (context, constraints) {
        final screenWidth = constraints.maxWidth;
        final isSmallScreen = screenWidth < 600;
        final isMediumScreen = screenWidth >= 600 && screenWidth < 900;

        if (isSmallScreen) {
            return _buildMobileLayout();
        } else if (isMediumScreen) {
            return _buildTabletLayout();
        } else {
            return _buildDesktopLayout();
        }
    },
)

```

```

// Grid adaptativo con Wrap
Wrap(
  spacing: 16,
  runSpacing: 16,
  children: items.map((item) {
    final itemsPerRow = isDesktop ? 3 : (isTablet ? 2 : 1);
    final itemWidth = (screenWidth - (padding * 2) - (16 * (itemsPerRow - 1))) /
    itemsPerRow;

    return SizedBox(
      width: itemWidth,
      child: _buildItem(item),
    );
  }).toList(),
)

```

3. Densidad de Información Adaptativa

```

dart
// Mostrar diferentes cantidades de información según el espacio
ListTile(
  title: Text(document.title),
  subtitle: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(document.fileName),
      if (screenWidth > 360) ...[
        const SizedBox(height: 4),
        Text('Registro: ${_formatDate(user.createdAt)}'),
      ],
    ],
  ),
  trailing: screenWidth > 600
)

```

```
? Row(  
    mainAxisSize: MainAxisSize.min,  
    children: [  
        TextButton(onPressed: () {}, child: const Text('Ver')),  
        IconButton(icon: const Icon(Icons.more_vert), onPressed: () {}),  
    ],  
)  
: IconButton(icon: const Icon(Icons.more_vert), onPressed: () {}),  
)
```

ACCESIBILIDAD Y USABILIDAD

1. Tamaños de Interacción

```
dart  
// Áreas de toque mínimas de 48x48px  
Container(  
    constraints: const BoxConstraints(  
        minWidth: 48,  
        minHeight: 48,  
    ),  
    child: IconButton(  
        icon: const Icon(Icons.action),  
        onPressed: () {},  
    ),  
)
```

```
// Botones con padding adecuado  
ElevatedButton(  
    style: ElevatedButton.styleFrom(  
        padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 16),  
        minimumSize: const Size(0, 48),  
    ),  
    onPressed: () {},
```

```
        child: const Text('Acción'),  
    )
```

2. Contraste y Legibilidad

```
dart  
// Texto con contraste adecuado  
Text(  
    'Texto importante',  
    style: TextStyle(  
        color: const Color(0xFF2C3E50), // Contraste alto con fondo blanco  
        fontSize: 16,  
        fontWeight: FontWeight.w500,  
    ),  
)
```

```
// Texto secundario con contraste suficiente  
Text(  
    'Información adicional',  
    style: TextStyle(  
        color: Colors.grey[600], // Contraste mínimo 4.5:1  
        fontSize: 14,  
    ),  
)
```

3. Estados de Foco

```
dart  
// Estados de foco visibles para navegación por teclado  
TextField(  
    decoration: InputDecoration(  
        focusedBorder: OutlineInputBorder(  
            borderRadius: BorderRadius.circular(12),  
            borderSide: const BorderSide(  
                color: Colors.purple,  
                width: 2,  
            ),  
        ),  
    ),  
)
```

```

        color: Color(0xFF6B4CE6),
        width: 2, // Borde más grueso en foco
    ),
),
),
),
)
}

// Indicadores de foco en elementos interactivos
ListTile(
    focusColor: const Color(0xFF6B4CE6).withValues(alpha: 0.1),
    hoverColor: const Color(0xFF6B4CE6).withValues(alpha: 0.05),
    onTap: () {},
)

```

4. Mensajes de Error Claros

```

dart
// Validación con mensajes específicos
TextField(
    validator: (value) {
        if (value == null || value.isEmpty) {
            return 'El nombre de usuario es requerido';
        }
        if (value.length < 3) {
            return 'Debe tener al menos 3 caracteres';
        }
        if (!RegExp(r'^[\w-\.]+@[([\w-]+\.)+[\w-]{2,4}$').hasMatch(value)) {
            return 'Ingresa un correo válido (ejemplo: usuario@dominio.com)';
        }
        return null;
    },
)

```

OPTIMIZACIONES DE RENDIMIENTO

1. Lazy Loading y Builders

dart

```
// ListView.builder para listas grandes
ListView.builder(
  itemCount: _documents.length,
  itemBuilder: (context, index) {
    return _buildDocumentCard(_documents[index]);
  },
)
```

// Uso de const constructors

```
const Icon(Icons.add, size: 18)
const SizedBox(width: 12)
const Text('Texto estático')
```

2. Gestión de Estado Optimizada

dart

```
// Evitar rebuilds innecesarios
class _DocumentsScreenState extends State<DocumentsScreen>
  with AutomaticKeepAliveClientMixin {
  @override
  bool get wantKeepAlive => true;

  @override
  Widget build(BuildContext context) {
    super.build(context); // Necesario para AutomaticKeepAliveClientMixin
    return _buildContent();
  }
}
```

3. Optimización de Imágenes y Assets

```
dart
// Uso de iconos en lugar de imágenes cuando sea posible
Icon(Icons.description_outlined) // Más eficiente que una imagen

// Precarga de assets críticos
@Override
void didChangeDependencies() {
  super.didChangeDependencies();
  // Precargar recursos si es necesario
}
```

TEMAS ESPECIALES Y PERSONALIZACIONES

1. Configuración del Tema Principal

```
dart
// main.dart - Configuración del tema
MaterialApp(
  title: 'DocuMente',
  theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(seedColor: const Color(0xFF6B4CE6)),
    useMaterial3: true,
    fontFamily: 'Roboto',
  ),
  // ...
)
```

2. Personalización por Rol

```
dart
// Diferentes estilos según el rol del usuario
CircleAvatar(
  backgroundColor: user?.isAdmin ?? false
    ? const Color(0xFF6B4CE6).withValues(alpha: 0.1)
    : Colors.grey[200],
```

```

child: Text(
  user?.username.substring(0, 1).toUpperCase() ?? 'U',
  style: TextStyle(
    color: user?.isAdmin ?? false
      ? const Color(0xFF6B4CE6)
      : Colors.grey[700],
    fontWeight: FontWeight.bold,
  ),
),
),
)

```

3. Estados Especiales del Sistema

```

dart
// Indicador de cuenta protegida
if (user.email.toLowerCase() == 'ivan@documente.com')
  Container(
    padding: const EdgeInsets.symmetric(horizontal: 6, vertical: 2),
    decoration: BoxDecoration(
      color: Colors.amber.withOpacity(0.1),
      borderRadius: BorderRadius.circular(12),
      border: Border.all(
        color: Colors.amber.withOpacity(0.5),
        width: 1,
      ),
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        Icon(Icons.lock, size: 10, color: Colors.amber[700]),
        const SizedBox(width: 4),
        Text(
          'Cuenta protegida',
        ),
      ],
    ),
  );
}

```

```
        style: TextStyle(
          fontSize: 10,
          fontWeight: FontWeight.w500,
          color: Colors.amber[700],
        ),
      ),
    ],
),
)
```

2.6 Funcionalidades por rol

DocuMente implementa un sistema de roles robusto que diferencia las capacidades y permisos entre usuarios regulares y administradores. Esta separación garantiza la seguridad y el control adecuado de las funcionalidades del sistema, basándose en una arquitectura de permisos que se aplica tanto en el frontend como en las validaciones del backend.

Arquitectura del Sistema de Roles

Modelo de Datos

```
// lib/models/user.dart
enum UserRole {
  user,
  admin
}

class User {
  final int id;
  final String username;
  final String email;
  final UserRole role;
  final String? avatar;
  final DateTime createdAt;

  const User({
    required this.id,
    required this.username,
    required this.email,
    required this.role,
```

```

    this.avatar,
    required this.createdAt,
  });

bool get isAdmin => role == UserRole.admin;

factory User.fromJson(Map<String, dynamic> json) {
  return User(
    id: json['id'],
    username: json['username'],
    email: json['email'],
    role: json['role'] == 'admin' ? UserRole.admin : UserRole.user,
    avatar: json['avatar'],
    createdAt: DateTime.parse(json['created_at']),
  );
}
}
}

```

Servicio de Autenticación

```

// lib/services/auth_service.dart
class AuthService extends ChangeNotifier {
  static final AuthService _instance = AuthService._internal();

  User? _currentUser;
  bool get isAuthenticated => _currentUser != null;
  bool get isAdmin => _currentUser?.isAdmin ?? false;

  Future<bool> login(String username, String password) async {
    // Lógica especial para Ivan (administrador único)
    if (username.toLowerCase() == 'ivan') {
      _currentUser = User(
        id: 1,
        username: 'Ivan',
        email: 'ivan@documenteme.com',
        role: UserRole.admin,
        createdAt: DateTime.now().subtract(const Duration(days: 30)),
      );
    } else {
      // Todos los demás usuarios siempre son regulares
      _currentUser = User(
        id: 2,
        username: username,
        email: '$username@documenteme.com',
      );
    }
    // Autentificación
    // ...
  }
}

```

```

        role: UserRole.user, // Siempre usuario regular
        createdAt: DateTime.now().subtract(const Duration(days: 10)),
    );
}
return true;
}
}

```

Funcionalidades Detalladas por Rol

Usuario Regular (UserRole.user)

1. Gestión de Documentos Propios

Pantalla de Mis Documentos (DocumentsScreen)

Características implementadas:

- Lista filtrable de documentos propios
- Búsqueda en tiempo real
- Filtros por tipo de archivo
- Indicadores visuales de estado

```

// Implementación de carga de documentos
void _loadDocuments() {
    final currentUser = AuthService().currentUser;

    _documents = [
        Document(
            id: 1,
            title: 'Contrato de Trabajo',
            fileName: 'contrato_trabajo.pdf',
            mimeType: 'application/pdf',
            ownerId: currentUser?.id ?? 1,
            ownerName: currentUser?.username ?? 'Usuario',
            createdAt: DateTime.now(),
            isShared: false,
        ),
        // ... más documentos
    ];
}

```

Funciones disponibles:

```

// Menú de acciones para cada documento
PopupMenuButton<String>(

```

```
onSelected: (action) => _handleDocumentAction(action, document),
itemBuilder: (context) => [
  PopupMenuItem(value: 'view', child: Text('Ver')),
  PopupMenuItem(value: 'share', child: Text('Compartir')),
  PopupMenuItem(value: 'download', child: Text('Descargar')),
  PopupMenuItem(value: 'delete', child: Text('Eliminar')),
],
)
```

Subida de Documentos

Tipos de archivo soportados:

- PDF (application/pdf)
- TXT (text/plain)
- Futura expansión para más formatos

```
void _showUploadDialog() {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: const Text('Subir Documento'),
      content: Column(
        children: [
          _buildUploadOption(Icons.picture_as_pdf, 'PDF', Colors.red),
          _buildUploadOption(Icons.insert_drive_file, 'TXT', Colors.blueGrey),
        ],
      ),
    ),
  );
}
```

Compartir Documentos

Sistema de permisos implementado:

- Solo lectura (read)
- Lectura y edición (write) - preparado para futuras versiones

```
void _showShareDialog(Document document) {
  showDialog(
    // ... configuración del diálogo
    content: Column(
      children: [
        TextField(
          decoration: InputDecoration(

```

```

        hintText: 'Correo electrónico del usuario',
        prefixIcon: const Icon(Icons.email),
    ),
),
RadioListTile<String>(
    title: const Text('Solo lectura'),
    value: 'read',
    groupValue: 'read',
    onChanged: (value) {},
),
],
),
);
}

```

2. Documentos Compartidos

Pantalla de Documentos Compartidos (SharedDocumentsScreen)

Características únicas:

- Vista de solo lectura obligatoria
- Información del propietario visible
- Contador de usuarios con acceso
- Opción de quitar de compartidos

```

Widget _buildDocumentCard(Document document) {
return Card(
child: Row(
children: [
// ... información del documento

// Indicador de solo lectura
Column(
children: [
Container(
decoration: BoxDecoration(
color: const Color(0xFF2196F3).withOpacity(0.1),
),
child: const Icon(Icons.visibility, color: Color(0xFF2196F3)),
),
Text('Solo lectura', style: TextStyle(fontSize: 10)),
],
),
// Menú limitado

```

```

PopupMenuButton<String>(
  itemBuilder: (context) => [
    PopupMenuItem(value: 'view', child: Text('Ver')),
    PopupMenuItem(value: 'download', child: Text('Descargar')),
    PopupMenuItem(value: 'remove_access',
      child: Text('Quitar de compartidos')),
  ],
),
],
),
);
);
}

```

3. Sistema de Chat con MentaIA

Funcionalidades de Chat

Gestión de conversaciones:

- Crear múltiples chats
- Renombrar conversaciones
- Eliminar chats propios
- Historial persistente

Interacciones con IA:

- Preguntas sobre documentos
- Análisis de contenido
- Resúmenes automáticos
- Búsqueda semántica

4. Perfil Personal (ProfileScreen)

Información Editable

```

Widget _buildPersonalInfoSection(User? user) {
  return Form(
    child: Column(
      children: [
        // Campo de nombre de usuario
        TextFormField(
          controller: _usernameController,
          enabled: _isEditing,
          validator: (value) {
            if (value!.length < 3) {
              return 'Debe tener al menos 3 caracteres';
            }
            return null;
          }
        ),
      ],
    ),
  );
}

```

```

},
),

// Campo de email
TextField(
  controller: _emailController,
  keyboardType: TextInputType.emailAddress,
  validator: (value) {
    if (!RegExp(r'^[\w-\.]+@[^\w-]+\.\w{2,4}$').hasMatch(value!)) {
      return 'Ingresa un correo válido';
    }
    return null;
  },
),
],
),
);
}

Cambio de Contraseña
// Sistema de tres campos para cambio seguro
if (_showPasswordFields) ...[
  TextFormField(
  controller: _currentPasswordController,
  obscureText: !_showCurrentPassword,
  decoration: InputDecoration(
    labelText: 'Contraseña actual',
    suffixIcon: IconButton(
      icon: Icon(_showCurrentPassword ? Icons.visibility : Icons.visibility_off),
      onPressed: () => setState(() => _showCurrentPassword =
        !_showCurrentPassword),
    ),
  ),
),
],
// Nueva contraseña y confirmación...
]

```

Administrador (UserRole.admin)

1. Panel de Administración (AdminPanelScreen)

Estructura de Navegación

late TabController _tabController;

@override

```
void initState() {
    super.initState();
    _tabController = TabController(length: 4, vsync: this);
    // Tabs: Dashboard, Usuarios, Documentos, Chats
}
```

Dashboard Administrativo

Estadísticas en tiempo real:

```
Widget _buildStatsGrid() {
    final stats = [
        {
            'title': 'Total Usuarios',
            'value': '${_users.length}',
            'icon': Icons.people,
            'color': const Color(0xFF4CAF50),
        },
        {
            'title': 'Documentos',
            'value': '${_allDocuments.length}',
            'icon': Icons.folder,
            'color': const Color(0xFF2196F3),
        },
        {
            'title': 'Chats Activos',
            'value': '${_allChats.length}',
            'icon': Icons.chat,
            'color': const Color(0xFFFF9800),
        },
    ];
}

return GridView.builder(
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: isDesktop ? 3 : (isTablet ? 2 : 1),
    ),
    itemBuilder: (context, index) => _buildStatCard(stats[index]),
);
```

Actividad reciente:

```
final activities = [
{
    'action': 'Nuevo usuario registrado',
```

```
'user': 'Roberto Silva',
'time': 'Hace 2 horas',
'icon': Icons.person_add,
},
// ... más actividades
];
2. Gestión de Usuarios
Visualización de Usuarios
Widget _buildUsersTab() {
return ListView.separated(
itemBuilder: (context, index) {
final user = _users[index];
return ListTile(
leading: CircleAvatar(
backgroundColor: user.isAdmin
? const Color(0xFF6B4CE6).withOpacity(0.1)
: Colors.grey[200],
child: Text(user.username.substring(0, 1).toUpperCase()),
),
title: Text(user.username),
subtitle: Column(
children: [
Text(user.email),
// Indicadores de rol y estado
if (user.email.toLowerCase() == 'ivan@documente.com')
Container(
decoration: BoxDecoration(
color: Colors.amber.withOpacity(0.1),
border: Border.all(color: Colors.amber.withOpacity(0.5)),
),
child: Row(
children: [
Icon(Icons.lock, size: 10),
Text('Cuenta protegida'),
],
),
),
),
],
),
trailing: user.email.toLowerCase() == 'ivan@documente.com'
? Icon(Icons.shield, color: Colors.amber)
: PopupMenuButton(...),
);
}
```

```
        },
    );
}
Crear Usuarios
void _addUser(String username, String email, bool isAdmin) {
    // IMPORTANTE: Solo Ivan puede ser administrador
    final isReallyAdmin = username.toLowerCase() == 'ivan';

    final newUser = User(
        id: _users.length + 1,
        username: username,
        email: email,
        role: isReallyAdmin ? UserRole.admin : UserRole.user, // Forzar rol
        createdAt: DateTime.now(),
    );
}
```

```
    setState(() => _users.add(newUser));
}
```

Restricciones de Eliminación

```
void _handleUserAction(String action, User user) {
    // Protección absoluta para Ivan
    if (user.email.toLowerCase() == 'ivan@documente.com') {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Esta cuenta está protegida')),
        );
        return;
    }

    // Solo Ivan puede eliminar usuarios
    if (action == 'delete' && !_currentUserIvan()) {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Solo Ivan puede eliminar usuarios')),
        );
        return;
    }
}
```

3. Gestión Global de Documentos

Vista Completa de Documentos

```
Widget _buildDocumentsTab() {
    return ListView.builder(
        itemBuilder: (context, index) {
            final document = _allDocuments[index];
            return ListTile(
```

```

title: Text(document.title),
subtitle: Column(
  children: [
    Text('Propietario: ${document.ownerName}'),
    if (document.isShared)
      Container(
        child: Text('Compartido'),
      ),
  ],
),
trailing: PopupMenuButton(
  itemBuilder: (context) => [
    PopupMenuItem(value: 'view', child: Text('Ver')),
    PopupMenuItem(value: 'share', child: Text('Compartir')),
    PopupMenuItem(value: 'delete', child: Text('Eliminar')),
  ],
),
);
},
);
}

```

4. Monitoreo de Chats

```

Widget _buildChatsTab() {
  return ListView.builder(
    itemBuilder: (context, index) {
      final chat = _allChats[index];
      final user = _users.firstWhere((u) => u.id == chat.userId);

      return ListTile(
        title: Text(chat.title),
        subtitle: Column(
          children: [
            Text('Usuario: ${user.username}'),
            Text('Última actividad: ${_formatDateTime(chat.updatedAt)}'),
          ],
        ),
        trailing: PopupMenuButton(
          itemBuilder: (context) => [
            PopupMenuItem(value: 'view', child: Text('Ver chat')),
            PopupMenuItem(value: 'export', child: Text('Exportar')),
            PopupMenuItem(value: 'delete', child: Text('Eliminar')),
          ],
        ),
      );
    },
  );
}

```

```
        );
    },
);
}
}
```

Implementación de Permisos

Control de Acceso en Navegación

```
// home_screen.dart
void _initializeNavigation() {
    final user = AuthService().currentUser;

    _navigationItems = [
        NavigationItem(label: 'Inicio', screen: _buildDashboard()),
        NavigationItem(label: 'Mis Documentos', screen: DocumentsScreen()),
        NavigationItem(label: 'Compartidos', screen: SharedDocumentsScreen()),
        NavigationItem(label: 'Chats', screen: ChatListScreen()),
    ];

    // Agregar panel admin solo si es administrador
    if (user?.isAdmin ?? false) {
        _navigationItems.add(
            NavigationItem(
                icon: Icons.admin_panel_settings,
                label: 'Administración',
                screen: AdminPanelScreen(),
            ),
        );
    }

    _navigationItems.add(
        NavigationItem(label: 'Perfil', screen: ProfileScreen()),
    );
}
```

Indicadores Visuales de Rol

```
// Navegación lateral (NavigationRail)
trailing: Container(
    child: Column(
        children: [
            CircleAvatar(
                backgroundColor: user?.isAdmin ?? false
                    ? const Color(0xFF6B4CE6).withOpacity(0.1)
```

```

        : Colors.grey[200],
      child: Text(user?.username.substring(0, 1).toUpperCase() ?? 'U'),
    ),
  if (user?.isAdmin ?? false)
    Container(
      padding: EdgeInsets.symmetric(horizontal: 4, vertical: 2),
      decoration: BoxDecoration(
        color: const Color(0xFF6B4CE6),
        borderRadius: BorderRadius.circular(8),
      ),
      child: Text('ADMIN', style: TextStyle(fontSize: 8)),
    ),
  ],
),
),
),
),

```

Diseño Responsivo

Adaptación por Tamaño de Pantalla

```

// Sistema de breakpoints
final isSmallScreen = screenWidth < 600; // Móviles
final isMediumScreen = screenWidth >= 600 && screenWidth < 900; // Tablets
final isLargeScreen = screenWidth >= 900; // Desktop

// Navegación adaptativa
bottomNavigationBar: isSmallScreen
  ? BottomNavigationBar(...) // Móviles
  : null,

body: Row(
  children: [
    if (!isSmallScreen) ...[
      NavigationRail(
        extended: !isMediumScreen, // Expandido solo en desktop
        // ...
      ),
    ],
    Expanded(child: _navigationItems[_selectedIndex].screen),
  ],
),

```

Tipografía Responsiva

```
TextStyle _getResponsiveTextStyle(double screenWidth, String type) {
```

```

final Map<String, Map<String, double>> fontSizes = {
  'small': {'title': 20.0, 'subtitle': 16.0, 'body': 14.0},
  'medium': {'title': 24.0, 'subtitle': 18.0, 'body': 16.0},
  'large': {'title': 28.0, 'subtitle': 20.0, 'body': 18.0},
};

String screenType = screenWidth >= 900 ? 'large'
: screenWidth >= 600 ? 'medium' : 'small';

final fontSize = fontSizes[screenType]![type] ?? 16.0;

// Aplicar estilos según tipo
switch (type) {
  case 'title':
    return TextStyle(
      fontSize: fontSize,
      fontWeight: FontWeight.bold,
      color: const Color(0xFF2C3E50),
    );
  // ... más casos
}
}

```

Seguridad y Validaciones

Protección de Cuenta Administrativa

```

// Verificaciones múltiples para proteger a Ivan
bool _isUserAdmin(User user) {
  return user.username.toLowerCase() == 'ivan' || user.role == UserRole.admin;
}

bool _isCurrentUserIvan() {
  final currentUser = AuthService().currentUser;
  return currentUser?.username.toLowerCase() == 'ivan';
}

// Aplicación de protecciones
if (user.username.toLowerCase() == 'ivan') {
  // No puede ser editado
  // No puede ser eliminado
  // Siempre mantiene rol admin
  return;
}

```

Validación de Formularios

```
// Validación de email
validator: (value) {
  if (!RegExp(r'^[\w-\.]+@[^\w-]+\.\w+{2,4}$').hasMatch(value!)) {
    return 'Ingresa un correo válido';
  }
  return null;
}

// Validación de contraseña
validator: (value) {
  if (_showPasswordFields && value!.length < 6) {
    return 'Debe tener al menos 6 caracteres';
  }
  return null;
}
```

Flujos de Usuario

Flujo de Usuario Regular

1. Login → Identificación como usuario regular
2. Home → Dashboard con estadísticas personales
3. Navegación:
 - Mis Documentos → CRUD completo de documentos propios
 - Compartidos → Solo lectura de documentos compartidos
 - Chats → Gestión completa de conversaciones propias
 - Perfil → Edición de información personal
4. Acciones limitadas → Sin acceso a funciones administrativas

Flujo de Administrador

1. Login como Ivan → Identificación como admin
2. Home → Dashboard con indicador ADMIN
3. Navegación extendida:
 - Todas las opciones de usuario regular
 - Panel de Administración:
 - * Dashboard → Estadísticas globales
 - * Usuarios → Gestión completa (crear, editar, eliminar*)
 - * Documentos → Vista y gestión global
 - * Chats → Monitoreo y exportación
4. Privilegios especiales → Acceso total al sistema

Mejores Prácticas Implementadas

1. Principio de Menor Privilegio

- Usuarios nuevos siempre inician como regulares
- Elevación de privilegios solo por diseño (Ivan)
- Acciones destructivas limitadas

2. Separación de Responsabilidades

- Roles claramente definidos en el modelo
- Funciones específicas por rol
- Sin superposición de capacidades críticas

3. Experiencia de Usuario

- Indicadores visuales claros de rol
- Mensajes de error descriptivos
- Confirmaciones para acciones destructivas

4. Seguridad

- Validaciones en cliente y servidor
- Protección de cuentas críticas
- Auditoría de acciones administrativas

Consideraciones de Implementación

Estados de Carga

```
bool _isLoading = false;

// Mostrar indicador de carga
_isLoading
? Center(child: CircularProgressIndicator(color: Color(0xFF6B4CE6)))
:_buildContent();
```

Manejo de Errores

```
try {
final success = await AuthService().updateProfile(updateData);
if (success) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text('Actualizado exitosamente')),
);
```

```

        }
    } catch (e) {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Error: $e'), backgroundColor: Colors.red),
        );
    }
}

```

Optimización de Rendimiento

```

// Uso de ListView.builder para listas grandes
ListView.builder(
    itemCount: _documents.length,
    itemBuilder: (context, index) => _buildDocumentCard(_documents[index]),
);

// Estados preservados con AutomaticKeepAliveClientMixin
class _DocumentsScreenState extends State<DocumentsScreen>
with AutomaticKeepAliveClientMixin {
    @override
    bool get wantKeepAlive => true;
}

```

2.7 Navegación adaptativa

DocuMente implementa un sistema de navegación completamente adaptativo que se ajusta automáticamente según el dispositivo, tamaño de pantalla y contexto de uso. La arquitectura de navegación garantiza una experiencia fluida y coherente, optimizando la usabilidad tanto en dispositivos móviles como en tablets y desktops, siguiendo los principios de Material Design 3 con personalizaciones específicas para la aplicación.

ARQUITECTURA DE NAVEGACIÓN

1. Sistema de Navegación Principal

La navegación principal de DocuMente se basa en un sistema dual que cambia automáticamente según el tamaño de pantalla:

```

// home_screen.dart - Sistema de navegación adaptativa
@Override
Widget build(BuildContext context) {
    final user = AuthService().currentUser;
    final isSmallScreen = MediaQuery.of(context).size.width < 600;
    final isMediumScreen = MediaQuery.of(context).size.width >= 600 &&

```

```
MediaQuery.of(context).size.width < 900;

return Scaffold(
// BottomNavigationBar para pantallas pequeñas (móviles)
bottomNavigationBar: isSmallScreen
? BottomNavigationBar(
currentIndex: _selectedIndex,
onTap: (index) {
setState(() {
_selectedIndex = index;
});
},
selectedItemColor: const Color(0xFF6B4CE6),
unselectedItemColor: Colors.grey[600],
type: BottomNavigationBarType.fixed,
items: _navigationItems
.map((item) => BottomNavigationBarItem(
icon: Icon(item.icon),
activeIcon: Icon(item.selectedIcon),
label: item.label,
))
.toList(),
)
: null,
)

// Body con NavigationRail para pantallas medianas y grandes
body: Row(
children: [
// NavigationRail para tablets y desktop
if (!isSmallScreen) ...[
NavigationRail(
extended: !isMediumScreen, // Expandido solo en desktop
selectedIndex: _selectedIndex,
onDestinationSelected: (index) {
setState(() {
_selectedIndex = index;
});
},
labelType: isMediumScreen
? NavigationRailLabelType.selected
: NavigationRailLabelType.none,
backgroundColor: Colors.white,
selectedIconTheme: const IconThemeData(
color: Color(0xFF6B4CE6),
```

```

        size: 28,
    ),
    unselectedIconTheme: IconThemeData(
        color: Colors.grey[600],
        size: 24,
    ),
    selectedLabelTextStyle: const TextStyle(
        color: Color(0xFF6B4CE6),
        fontWeight: FontWeight.w600,
        fontSize: 12,
    ),
    unselectedLabelTextStyle: TextStyle(
        color: Colors.grey[600],
        fontSize: 12,
    ),
    // Personalización adicional
    leading: _buildNavigationRailHeader(),
    trailing: _buildNavigationRailFooter(),
    destinations: _navigationItems
        .map((item) => NavigationRailDestination(
            icon: Icon(item.icon),
            selectedIcon: Icon(item.selectedIcon),
            label: Text(item.label),
        ))
        .toList(),
),
const VerticalDivider(thickness: 1, width: 1),
],
// Contenido principal
Expanded(
    child: _navigationItems[_selectedIndex].screen,
),
],
),
);
}
}

```

2. Modelo de Navegación

```

class NavigationItem {
    final IconData icon;
    final IconData selectedIcon;
    final String label;
    final Widget screen;
}

```

```
NavigationItem({  
    required this.icon,  
    required this.selectedIcon,  
    required this.label,  
    required this.screen,  
});  
}
```

3. Inicialización Dinámica de Navegación

```
void _initializeNavigation() {  
    final user = AuthService().currentUser;  
  
    _navigationItems = [  
        NavigationItem(  
            icon: Icons.home_outlined,  
            selectedIcon: Icons.home,  
            label: 'Inicio',  
            screen: _buildDashboard(),  
        ),  
        NavigationItem(  
            icon: Icons.folder_outlined,  
            selectedIcon: Icons.folder,  
            label: 'Mis Documentos',  
            screen: const DocumentsScreen(),  
        ),  
        NavigationItem(  
            icon: Icons.share_outlined,  
            selectedIcon: Icons.share,  
            label: 'Compartidos',  
            screen: const SharedDocumentsScreen(),  
        ),  
        NavigationItem(  
            icon: Icons.chat_bubble_outline,  
            selectedIcon: Icons.chat_bubble,  
            label: 'Chats',  
            screen: const ChatListScreen(),  
        ),  
    ];  
  
    // Agregar opciones de administrador si el usuario es admin  
    if (user?.isAdmin ?? false) {  
        _navigationItems.add(  
    }
```

```

        NavigationItem(
            icon: Icons.admin_panel_settings_outlined,
            selectedIcon: Icons.admin_panel_settings,
            label: 'Administración',
            screen: const AdminPanelScreen(),
        ),
    );
}

_navigationItems.add(
    NavigationItem(
        icon: Icons.person_outline,
        selectedIcon: Icons.person,
        label: 'Perfil',
        screen: const ProfileScreen(),
    ),
);
}

```

COMPONENTES DE NAVEGACIÓN ESPECÍFICOS

1. NavigationRail Personalizado

Header del NavigationRail

```

Widget _buildNavigationRailHeader() {
    return Container(
        padding: const EdgeInsets.symmetric(vertical: 20),
        child: Column(
            children: [
                Container(
                    width: 50,
                    height: 50,
                    decoration: BoxDecoration(
                        gradient: const LinearGradient(
                            colors: [
                                Color(0xFF6B4CE6),
                                Color(0xFFE91E63),
                            ],
                        ),
                        borderRadius: BorderRadius.circular(12),
                    ),
                    child: const Icon(
                        Icons.description_outlined,
                        color: Colors.white,
                        size: 24,
                    )
                )
            ]
        )
    );
}

```

```

),
),
if (!isMediumScreen) const SizedBox(height: 8),
if (!isMediumScreen)
Text(
'DocuMente',
style: TextStyle(
color: Colors.grey[800],
fontSize: 10,
fontWeight: FontWeight.bold,
),
),
],
),
);
}

```

Footer del NavigationRail

```

Widget _buildNavigationRailFooter() {
final user = AuthService().currentUser;

return Container(
padding: const EdgeInsets.symmetric(vertical: 20),
child: Column(
children: [
CircleAvatar(
radius: 20,
backgroundColor: user?.isAdmin ?? false
? const Color(0xFF6B4CE6).withValues(alpha: 0.1)
: Colors.grey[200],
child: Text(
user?.username.substring(0, 1).toUpperCase() ?? 'U',
style: TextStyle(
color: user?.isAdmin ?? false
? const Color(0xFF6B4CE6)
: Colors.grey[700],
fontWeight: FontWeight.bold,
),
),
),
),
),
if (!isMediumScreen) const SizedBox(height: 4),
if ((user?.isAdmin ?? false) && !isMediumScreen)
Container(
padding: const EdgeInsets.symmetric(horizontal: 4, vertical: 2),

```

```

decoration: BoxDecoration(
  color: const Color(0xFF6B4CE6),
  borderRadius: BorderRadius.circular(8),
),
child: const Text(
  'ADMIN',
  style: TextStyle(
    color: Colors.white,
    fontSize: 8,
    fontWeight: FontWeight.bold,
  ),
),
),
],
),
);
}
}

```

2. Navegación por Tabs

Implementación en AdminPanelScreen

```

class _AdminPanelScreenState extends State<AdminPanelScreen>
  with TickerProviderStateMixin {
late TabController _tabController;

@Override
void initState() {
  super.initState();
  _tabController = TabController(length: 4, vsync: this);
}

@Override
Widget build(BuildContext context) {
  final screenWidth = MediaQuery.of(context).size.width;
  final isTablet = screenWidth >= 600;

  return SafeArea(
    child: Scaffold(
      body: Column(
        children: [
          // Header del panel
          _buildAdminHeader(),

```

// Tabs adaptables

```
Container(
  color: Colors.white,
  child: TabBar(
    controller: _tabController,
    labelColor: const Color(0xFF6B4CE6),
    unselectedLabelColor: Colors.grey[600],
    indicatorColor: const Color(0xFF6B4CE6),
    isScrollable: screenWidth < 500, // Scrollable en móviles
    labelStyle: TextStyle(fontSize: isTablet ? 12 : 10),
    tabs: [
      Tab(
        icon: Icon(Icons.dashboard, size: isTablet ? 20 : 18),
        text: screenWidth > 360 ? 'Dashboard' : 'Inicio',
      ),
      Tab(
        icon: Icon(Icons.people, size: isTablet ? 20 : 18),
        text: 'Usuarios',
      ),
      Tab(
        icon: Icon(Icons.folder, size: isTablet ? 20 : 18),
        text: screenWidth > 360 ? 'Documentos' : 'Docs',
      ),
      Tab(
        icon: Icon(Icons.chat, size: isTablet ? 20 : 18),
        text: 'Chats',
      ),
    ],
  ),
),

// Contenido de las tabs
Expanded(
  child: TabBarView(
    controller: _tabController,
    children: [
      _buildDashboardTab(),
      _buildUsersTab(),
      _buildDocumentsTab(),
      _buildChatsTab(),
    ],
  ),
),
],
```

```

        ),
    );
}
}

Implementación en DocumentDetailScreen
@Override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            // ... configuración del AppBar
            bottom: TabBar(
                controller: _tabController,
                labelColor: const Color(0xFF6B4CE6),
                unselectedLabelColor: Colors.grey[600],
                indicatorColor: const Color(0xFF6B4CE6),
                tabs: const [
                    Tab(icon: Icon(Icons.description), text: 'Contenido'),
                    Tab(icon: Icon(Icons.info), text: 'Información'),
                    Tab(icon: Icon(Icons.people), text: 'Compartido'),
                ],
            ),
        ),
        body: TabBarView(
            controller: _tabController,
            children: [
                _buildContentTab(),
                _buildInfoTab(),
                _buildSharedTab(isOwner),
            ],
        ),
    );
}
}

```

3. Drawer Personalizado

Implementación en Chat Screen

```

Widget _buildDrawer() {
    return Drawer(
        backgroundColor: Colors.white,
        child: Column(
            children: [
                // Header del drawer con gradiente
                Container(
                    width: double.infinity,

```

```
decoration: const BoxDecoration(
gradient: LinearGradient(
begin: Alignment.topLeft,
end: Alignment.bottomRight,
colors: [
Color(0xFF6B4CE6), // Morado
Color(0xFFE91E63), // Rosa
],
),
),
child: SafeArea(
child: Padding(
padding: const EdgeInsets.all(20.0),
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
// Logo de DocuMente
Container(
width: 60,
height: 60,
decoration: BoxDecoration(
color: Colors.white,
borderRadius: BorderRadius.circular(15),
boxShadow: [
BoxShadow(
color: Colors.black.withOpacity(0.1),
blurRadius: 10,
offset: const Offset(0, 5),
),
],
),
),
child: const Icon(
Icons.description_outlined,
color: Color(0xFF6B4CE6),
size: 30,
),
),
const SizedBox(height: 15),
const Text(
'DocuMente',
style: TextStyle(
color: Colors.white,
fontSize: 24,
fontWeight: FontWeight.bold,
```

```
        ),  
        ),  
    ],  
    ),  
    ),  
),  
,  
  
// Contenido del drawer  
Expanded(  
child: Padding(  
padding: const EdgeInsets.symmetric(vertical: 10),  
child: Column(  
children: [  
    _buildDrawerItem(  
        icon: Icons.chat_bubble_outline,  
        title: 'Nueva conversación',  
        onTap: () {  
            Navigator.pop(context);  
            _showNewChatDialog();  
        },  
    ),  
    _buildDrawerItem(  
        icon: Icons.history,  
        title: 'Historial',  
        onTap: () {  
            Navigator.pop(context);  
            _showHistorial();  
        },  
    ),  
    const Divider(),  
    _buildDrawerItem(  
        icon: Icons.logout,  
        title: 'Cerrar sesión',  
        onTap: () {  
            Navigator.pop(context);  
            _showLogoutDialog();  
        },  
        isLogout: true,  
    ),  
],  
),  
),  
,
```

```
// Footer con versión
Container(
  padding: const EdgeInsets.all(20),
  child: Column(
    children: [
      Text(
        'Versión 1.0.0',
        style: TextStyle(color: Colors.grey[600], fontSize: 12),
      ),
      const SizedBox(height: 5),
      Text(
        '© 2024 DocuMente',
        style: TextStyle(color: Colors.grey[500], fontSize: 10),
      ),
    ],
  ),
),
],
),
],
),
);
}
```

Items del Drawer

```
Widget _buildDrawerItem({
  required IconData icon,
  required String title,
  required VoidCallback onTap,
  bool isLogout = false,
}) {
  return Container(
    margin: const EdgeInsets.symmetric(horizontal: 10, vertical: 2),
    child: ListTile(
      leading: Container(
        padding: const EdgeInsets.all(8),
        decoration: BoxDecoration(
          color: isLogout
            ? Colors.red.withOpacity(0.1)
            : const Color(0xFF6B4CE6).withAlpha(0.1),
          borderRadius: BorderRadius.circular(10),
        ),
        child: Icon(
          icon,
          color: isLogout ? Colors.red : const Color(0xFF6B4CE6),
        ),
      ),
    ),
  );
}
```

```

        size: 20,
    ),
),
title: Text(
    title,
    style: TextStyle(
        color: isLogout ? Colors.red : const Color(0xFF2C3E50),
        fontSize: 16,
        fontWeight: FontWeight.w500,
    ),
),
onTap: onTap,
shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(10),
),
hoverColor: const Color(0xFF6B4CE6).withValues(alpha: 0.05),
),
);
}
}

```

4. AppBar con Navegación Jerárquica

```

AppBar(
    elevation: 0,
    backgroundColor: Colors.white,
    foregroundColor: const Color(0xFF2C3E50),
    title: Text(
        widget.document.title,
        style: const TextStyle(
            fontWeight: FontWeight.bold,
        ),
    ),
    actions: [
        PopupMenuButton<String>(
            onSelected: _handleMenuAction,
            itemBuilder: (context) => [
                const PopupMenuItem(
                    value: 'download',
                    child: Row(
                        children: [
                            Icon(Icons.download, size: 18),
                            SizedBox(width: 12),
                            Text('Descargar'),
                        ],
                    ),
                ),
            ],
        ),
    ],
)

```

```
        ),  
        ),  
        if (isOwner) ...[  
          const PopupMenuItem(  
            value: 'share',  
            child: Row(  
              children: [  
                Icon(Icons.share, size: 18),  
                SizedBox(width: 12),  
                Text('Compartir'),  
              ],  
            ),  
          ),  
          const PopupMenuItem(  
            value: 'edit',  
            child: Row(  
              children: [  
                Icon(Icons.edit, size: 18),  
                SizedBox(width: 12),  
                Text('Editar'),  
              ],  
            ),  
          ),  
          const PopupMenuItem(  
            value: 'delete',  
            child: Row(  
              children: [  
                Icon(Icons.delete, size: 18, color: Colors.red),  
                SizedBox(width: 12),  
                Text('Eliminar', style: TextStyle(color: Colors.red)),  
              ],  
            ),  
          ),  
        ],  
      ),  
    ],  
  ),
```

5. FloatingActionButton (FAB)

FAB Simple

// En DocumentsScreen

```
floatingActionButton: FloatingActionButton.extended(
```

```

 onPressed: _showUploadDialog,
 backgroundColor: const Color(0xFF6B4CE6),
 foregroundColor: Colors.white,
 icon: const Icon(Icons.add),
 label: const Text('Subir Documento'),
)

FAB Condicional
// En DocumentDetailScreen
floatingActionButton: isOwner
? FloatingActionButton.extended(
    onPressed: _showShareDialog,
    backgroundColor: const Color(0xFF6B4CE6),
    foregroundColor: Colors.white,
    icon: const Icon(Icons.share),
    label: const Text('Compartir'),
)
: null,

```

6. Menús Contextuales (PopupMenuButton)

```

PopupMenuButton<String>(
  onSelected: (action) => _handleDocumentAction(action, document),
  itemBuilder: (context) => [
    const PopupMenuItem(
      value: 'view',
      child: Row(
        children: [
          Icon(Icons.visibility, size: 18),
          SizedBox(width: 12),
          Text('Ver'),
        ],
      ),
    ),
    const PopupMenuItem(
      value: 'share',
      child: Row(
        children: [
          Icon(Icons.share, size: 18),
          SizedBox(width: 12),
          Text('Compartir'),
        ],
      ),
    ),
    const PopupMenuItem(

```

```
        value: 'download',
        child: Row(
            children: [
                Icon(Icons.download, size: 18),
                SizedBox(width: 12),
                Text('Descargar'),
            ],
        ),
    ),
),
const PopupMenuItem(
    value: 'delete',
    child: Row(
        children: [
            Icon(Icons.delete, size: 18, color: Colors.red),
            SizedBox(width: 12),
            Text('Eliminar', style: TextStyle(color: Colors.red)),
        ],
    ),
),
),
],
),
child: const Icon(Icons.more_vert),
)
```

BREAKPOINTS Y LOGICA DE ADAPTACION

1. Sistema de Breakpoints

```
class ResponsiveBreakpoints {  
    static const double small = 600; // < 600px: Móviles  
    static const double medium = 900; // 600-900px: Tablets  
    static const double large = 1200; // > 900px: Desktop  
  
    static bool isSmallScreen(BuildContext context) {  
        return MediaQuery.of(context).size.width < small;  
    }  
  
    static bool isMediumScreen(BuildContext context) {  
        final width = MediaQuery.of(context).size.width;  
        return width >= small && width < medium;  
    }  
  
    static bool isLargeScreen(BuildContext context) {  
        return MediaQuery.of(context).size.width >= medium;  
    }  
}
```

2. Decisiones de Layout Basadas en Tamaño

```
// Ejemplo de adaptación de contenido
Widget build(BuildContext context) {
  final screenWidth = MediaQuery.of(context).size.width;
  final isSmallScreen = screenWidth < 600;
  final isMediumScreen = screenWidth >= 600 && screenWidth < 900;
  final isLargeScreen = screenWidth >= 900;

  // Ajustar número de columnas en grid
  int crossAxisCount = isSmallScreen ? 1 : (isMediumScreen ? 2 : 3);

  // Ajustar padding
  EdgeInsets padding = isSmallScreen
    ? const EdgeInsets.all(16)
    : (isMediumScreen
      ? const EdgeInsets.all(24)
      : const EdgeInsets.all(32));

  // Ajustar tamaños de fuente
  double fontSize = isSmallScreen ? 14 : (isMediumScreen ? 16 : 18);

  return Container(
    padding: padding,
    child: GridView.builder(
      gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: crossAxisCount,
        // ... resto de configuración
      ),
      // ...
    ),
  );
}
```

TRANSICIONES Y ANIMACIONES DE NAVEGACIÓN

1. Transiciones entre Pantallas

```
// Transición estándar con MaterialPageRoute
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => const DocumentDetailScreen(document: document),
  ),
);
```

```
// Transición con animación personalizada
Navigator.push(
  context,
  PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) => const Chat(),
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      const begin = Offset(1.0, 0.0);
      const end = Offset.zero;
      const curve = Curves.ease;

      var tween = Tween(begin: begin, end: end).chain(
        CurveTween(curve: curve),
      );

      return SlideTransition(
        position: animation.drive(tween),
        child: child,
      );
    },
  ),
);
```

2. Animaciones en Cambios de Tab

```
// AnimationController para tabs con vsync
class _AdminPanelScreenState extends State<AdminPanelScreen>
  with TickerProviderStateMixin {
  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(
      length: 4,
      vsync: this,
      animationDuration: const Duration(milliseconds: 300),
    );
  }
}
```

3. Animaciones en NavigationRail

NavigationRail(

```
extended: !isMediumScreen,
// Animación suave al expandir/contraer
extendedAnimation: AlwaysStoppedAnimation(1.0),
// ...
)
```

GESTIÓN DE ESTADO EN NAVEGACIÓN

1. Preservación de Estado entre Navegaciones

```
// Uso de AutomaticKeepAliveClientMixin
class _DocumentsScreenState extends State<DocumentsScreen>
  with AutomaticKeepAliveClientMixin {
  @override
  bool get wantKeepAlive => true;

  @override
  Widget build(BuildContext context) {
    super.build(context); // Necesario para AutomaticKeepAliveClientMixin
    return _buildContent();
  }
}
```

2. Gestión de Stack de Navegación

```
// Navegación con reemplazo
Navigator.pushReplacement(
  context,
  MaterialPageRoute(builder: (context) => const HomeScreen()),
);

// Limpiar stack y navegar
Navigator.pushNamedAndRemoveUntil(
  context,
  '/',
  (route) => false,
);

// Navegación con resultado
final result = await Navigator.push<bool>(
  context,
  MaterialPageRoute(
    builder: (context) => EditScreen(),
  ),
);
```

```
if (result == true) {
    // Actualizar UI si se guardaron cambios
    _refresh();
}
```

ACCESIBILIDAD EN NAVEGACIÓN

1. Labels y Semántica

```
BottomNavigationBarItem(
    icon: Icon(item.icon),
    activeIcon: Icon(item.selectedIcon),
    label: item.label, // Label descriptivo
    tooltip: 'Navegar a ${item.label}', // Tooltip adicional
)
```

```
NavigationRailDestination(
    icon: Semantics(
        label: 'Icono de ${item.label}',
        child: Icon(item.icon),
    ),
    selectedIcon: Semantics(
        label: '${item.label} seleccionado',
        child: Icon(item.selectedIcon),
    ),
    label: Text(item.label),
)
```

2. Navegación por Teclado

```
// Soporte para navegación por teclado
Focus(
    autofocus: _selectedIndex == index,
    child: InkWell(
        onTap: () => _onItemTapped(index),
        onFocusChange: (hasFocus) {
            if (hasFocus) {
                setState(() => _hoveredIndex = index);
            }
        },
        child: NavigationItem(),
    ),
)
```

3. Anuncios de Navegación

```
// Anunciar cambios de pantalla para lectores
Semantics(
  liveRegion: true,
  child: Text(
    'Navegando a ${_navigationItems[_selectedIndex].label}',
    style: const TextStyle(fontSize: 0), // Invisible pero anunciado
  ),
)
```

PATRONES DE NAVEGACIÓN ESPECÍFICOS

1. Navegación Condicional por Rol

```
// Mostrar opciones según el rol del usuario
if (user?.isAdmin ?? false) {
  _navigationItems.add(
    NavigationItem(
      icon: Icons.admin_panel_settings_outlined,
      selectedIcon: Icons.admin_panel_settings,
      label: 'Administración',
      screen: const AdminPanelScreen(),
    ),
  );
}
```

2. Navegación con Confirmación

```
// Confirmar antes de navegar fuera si hay cambios sin guardar
Future<bool> _onWillPop() async {
  if (_hasUnsavedChanges) {
    final shouldPop = await showDialog<bool>(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text('¿Descartar cambios?'),
        content: const Text('Tienes cambios sin guardar. ¿Deseas descartarlos?'),
        actions: [
          TextButton(
            onPressed: () => Navigator.of(context).pop(false),
            child: const Text('Cancelar'),
          ),
          ElevatedButton(
            onPressed: () => Navigator.of(context).pop(true),
            child: const Text('Descartar'),
          ),
        ],
      ),
    );
  }
}
```

```

        ),
    ],
),
);
return shouldPop ?? false;
}
return true;
}

// Uso en Scaffold
WillPopScope(
  onWillPop: _onWillPop,
  child: Scaffold(...),
)

```

3. Deep Linking y Rutas

```

// Definición de rutas nombradas
MaterialApp(
  routes: {
    '/': (context) => const HomeScreen(),
    '/login': (context) => const LoginScreen(),
    '/documents': (context) => const DocumentsScreen(),
    '/chat': (context) => const Chat(),
    '/profile': (context) => const ProfileScreen(),
  },
  onGenerateRoute: (settings) {
    // Manejo de rutas dinámicas
    if (settings.name?.startsWith('/document/') ?? false) {
      final id = settings.name!.split('/').last;
      return MaterialPageRoute(
        builder: (context) => DocumentDetailScreen(
          documentId: int.parse(id),
        ),
      );
    }
    return null;
  },
)

```

OPTIMIZACIONES DE RENDIMIENTO

1. Lazy Loading de Pantallas

```
// Cargar pantallas solo cuando se necesiten
```

```

Widget _buildScreen(int index) {
  switch (index) {
    case 0:
      return const HomeScreen();
    case 1:
      // Lazy load con FutureBuilder
      return FutureBuilder(
        future: _loadDocumentsScreen(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return snapshot.data!;
          }
          return const Center(child: CircularProgressIndicator());
        },
      );
    // ... más casos
  }
}

```

2. Precarga de Recursos

```

// Precargar recursos de navegación
@Override
void didChangeDependencies() {
  super.didChangeDependencies();
  // Precargar iconos y assets
  precacheImage(const AssetImage('assets/logo.png'), context);
}

```

3. Optimización de Rebuilds

```

// Evitar rebuilds innecesarios
class NavigationController extends ChangeNotifier {
  int _selectedIndex = 0;

  int get selectedIndex => _selectedIndex;

  void setIndex(int index) {
    if (_selectedIndex != index) {
      _selectedIndex = index;
      notifyListeners();
    }
  }
}

```

```
// Uso con Consumer para rebuilds selectivos
Consumer<NavigationController>(
  builder: (context, navController, child) {
    return NavigationRail(
      selectedIndex: navController.selectedIndex,
      // ...
    );
  },
)
```

MANEJO DE ERRORES EN NAVEGACIÓN

1. Navegación Segura

```
// Verificar si el contexto sigue montado antes de navegar
void _navigateAfterDelay() async {
  await Future.delayed(const Duration(seconds: 2));

  if (!mounted) return;

  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const NextScreen()),
  );
}
```

2. Fallback para Rutas No Encontradas

```
MaterialApp(
  onUnknownRoute: (settings) {
    return MaterialPageRoute(
      builder: (context) => const NotFoundScreen(),
    );
  },
)
```

3. Manejo de Errores en Navegación

```
try {
  await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => DetailScreen(id: documentId),
    ),
}
```

```

);
} catch (e) {
if (mounted) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text('Error al navegar: $e'),
backgroundColor: Colors.red,
),
);
}
}
}

```

3. Backend

3.1 Arquitectura General

3.1.1 Patrón de diseño implementado (Clean Architecture)

El backend del sistema de chatbot está diseñado siguiendo los principios de Clean Architecture (Arquitectura Limpia) propuesta por Robert C. Martin. Esta arquitectura garantiza la separación de responsabilidades, la independencia de frameworks y la facilidad de mantenimiento y testing del código.

Estructura de Capas

La aplicación se organiza en las siguientes capas concéntricas, donde las dependencias fluyen hacia el interior (principio de inversión de dependencias):

1. Capa de Presentación (API Layer)

- Ubicación: /src/api/
- Responsabilidad: Manejo de peticiones HTTP, validación de entrada y formateo de respuestas
- Componentes principales:
 - endpoints/: Definición de rutas y controladores REST
 - chat.py: Gestión de chats y mensajes
 - documents.py: Manejo de documentos
 - users.py: Autenticación y gestión de usuarios
 - health.py: Endpoints de monitoreo
 - statistics.py: Estadísticas del sistema
 - routes.py: Enrutador principal que organiza todos los endpoints
 - dependencies/: Inyección de dependencias y autenticación

2. Capa de Aplicación (Service Layer)

- Ubicación: /src/services/
- Responsabilidad: Implementación de casos de uso y lógica de aplicación
- Componentes principales:
 - chat_service.py: Lógica de negocio para chats y mensajes con integración RAG
 - document_service.py: Procesamiento y gestión de documentos
 - user_service.py: Gestión de usuarios y autenticación
 - email_service.py: Servicios de notificación por correo
 - statistics_service.py: Cálculos y métricas del sistema

3. Capa de Dominio (Domain Layer)

- Ubicación: /src/models/domain/
- Responsabilidad: Definición de entidades de negocio y reglas de dominio
- Entidades principales:
 - User: Modelo de usuario del sistema
 - Document: Representación de documentos procesados
 - Chat: Conversaciones entre usuarios y el chatbot
 - Message: Mensajes individuales dentro de las conversaciones

4. Capa de Infraestructura (Infrastructure Layer)

- Ubicación: /src/repositories/ y /src/utils/
- Responsabilidad: Acceso a datos externos y servicios de infraestructura
- Componentes principales:
 - Repositorios (/repositories/):
 - chat_repository.py: Persistencia de chats en Supabase
 - document_repository.py: Gestión de documentos en base de datos
 - user_repository.py: Operaciones CRUD de usuarios
 - message_repository.py: Almacenamiento de mensajes
 - Conectores externos (/utils/):
 - ai_connector.py: Integración con Google Gemini AI
 - chromadb_connector.py: Base de datos vectorial para RAG
 - token_utils.py: Gestión de tokens JWT

5. Capa de Configuración

- Ubicación: /src/config/
- Responsabilidad: Configuración centralizada y conexiones a servicios externos
- Componentes:
 - database.py: Configuración de Supabase con patrón Singleton
 - settings.py: Variables de entorno y configuraciones globales

Separación de Modelos

El sistema implementa una clara separación entre diferentes tipos de modelos:

Modelos de Dominio (/models/domain/)

- Representan las entidades de negocio puras
- No tienen dependencias de frameworks externos
- Contienen la lógica de negocio esencial

Schemas/DTOs (/models/schemas/)

- Modelos para serialización/deserialización API
- Validación de datos de entrada y salida
- Transformación entre capas
- Ejemplos: ChatCreate, ChatResponse, MessageCreate

Principios de Clean Architecture Aplicados

1. Independencia de Frameworks

- La lógica de negocio no depende de FastAPI
- Los repositorios abstraen el acceso a Supabase
- Los servicios pueden funcionar independientemente de la capa web

2. Inversión de Dependencias

- Los servicios definen interfaces que los repositorios implementan
- La inyección de dependencias se maneja en /api/dependencies/
- Las capas internas no conocen las externas

3. Separación de Responsabilidades

- Cada capa tiene una responsabilidad específica y bien definida
- Los endpoints solo manejan HTTP, los servicios la lógica de negocio
- Los repositorios solo acceden a datos

4. Testabilidad

- Cada capa puede probarse independientemente
- Los servicios pueden mockearse fácilmente
- La estructura facilita pruebas unitarias y de integración

Patrones de Diseño Complementarios

Singleton Pattern

- Implementado en SupabaseConnector para gestión de conexiones
- Garantiza una única instancia de conexión a base de datos

Repository Pattern

- Abstacta el acceso a datos con interfaces consistentes
- Facilita el cambio de tecnologías de persistencia

Dependency Injection

- Centralizada en /api/dependencies/__init__.py
- Permite inyección de servicios y autenticación

Factory Pattern

- Utilizado en la creación de conectores y servicios
- Centraliza la lógica de instanciación

Ventajas de la Arquitectura Implementada

1. Mantenibilidad: Código organizado y fácil de modificar
2. Escalabilidad: Estructura que permite crecimiento del sistema
3. Testabilidad: Cada componente puede probarse independientemente
4. Flexibilidad: Facilita cambios en tecnologías específicas
5. Reutilización: Componentes reutilizables entre diferentes partes del sistema
6. Separación de Responsabilidades: Cada capa tiene un propósito claro y definido

Esta arquitectura proporciona una base sólida para el desarrollo del chatbot con capacidades RAG, garantizando que el sistema sea mantenible, escalable y fácil de probar a medida que evoluciona.

- 3.1.2 Framework principal (FastAPI)

Rendimiento y Eficiencia

FastAPI está construido sobre Starlette y Pydantic, lo que lo convierte en uno de los frameworks web más rápidos disponibles para Python. Su rendimiento es comparable al de frameworks como NodeJS y Go, siendo significativamente más rápido que Django o Flask en operaciones de alta concurrencia. Esto es

especialmente relevante para aplicaciones modernas que requieren manejar múltiples solicitudes simultáneas.

Tipado Estático y Validación Automática

Una de las características más destacadas es su integración nativa con Python type hints. Esto proporciona validación automática de datos de entrada y salida, reduciendo errores en tiempo de ejecución y mejorando la calidad del código. La validación se realiza usando Pydantic, que convierte automáticamente los datos y proporciona mensajes de error claros cuando la validación falla.

Documentación Automática de APIs

FastAPI genera automáticamente documentación interactiva usando OpenAPI (Swagger UI) y ReDoc. Esto significa que cada endpoint que desarrolles tendrá documentación actualizada automáticamente, incluyendo esquemas de datos, ejemplos de uso y la posibilidad de probar la API directamente desde el navegador. Esto es invaluable tanto para el desarrollo como para la presentación de tu TFG.

Soporte Nativo para Programación Asíncrona

El framework soporta tanto funciones síncronas como asíncronas de manera nativa, permitiendo aprovechar las ventajas de la programación asíncrona cuando sea necesario. Esto es crucial para aplicaciones que realizan operaciones I/O intensivas como consultas a bases de datos o llamadas a APIs externas.

Ecosistema Moderno y Estándares Web

FastAPI implementa estándares modernos como OpenAPI, JSON Schema, y OAuth2. Esto facilita la integración con herramientas de terceros y garantiza que tu aplicación siga las mejores prácticas de la industria. También tiene excelente soporte para CORS, middleware personalizado y autenticación.

Curva de Aprendizaje y Productividad

Para desarrolladores familiarizados con Python, FastAPI tiene una sintaxis intuitiva que permite desarrollo rápido sin sacrificar funcionalidad. La documentación es excelente y hay abundantes ejemplos disponibles, lo que acelera el proceso de desarrollo para un TFG.

Comunidad Activa y Futuro del Framework

FastAPI tiene una comunidad muy activa y está respaldado por grandes empresas como Microsoft, que lo utilizan en producción. Su adopción está

creciendo rápidamente, lo que indica que es una tecnología relevante para el mercado laboral actual.

3.1.3 Estructura de capas del proyecto con principios SOLID aplicados

El proyecto implementa los cinco principios SOLID de forma sistemática a través de su arquitectura en capas, garantizando un código mantenable, extensible y robusto.

S - Single Responsibility Principle (Principio de Responsabilidad Única)

Cada clase del sistema tiene una única razón para cambiar y una responsabilidad específica claramente definida:

Servicios - Una responsabilidad de negocio por servicio:

- ChatService: Exclusivamente gestión de chats y mensajes con lógica RAG
- UserService: Solo manejo de usuarios, autenticación y autorización
- DocumentService: Únicamente procesamiento y gestión de documentos
- EmailService: Solamente envío de notificaciones por correo electrónico
- StatisticsService: Exclusivamente cálculos y métricas del sistema

Repositorios - Una entidad de datos por repositorio:

- ChatRepository: Solo operaciones CRUD para la entidad Chat
- UserRepository: Únicamente persistencia de usuarios en Supabase
- DocumentRepository: Exclusivamente acceso a datos de documentos
- MessageRepository: Solo gestión de persistencia de mensajes

Conectores - Una responsabilidad de infraestructura específica:

- GeminiConnector: Únicamente integración con Google Gemini AI
- ChromaDBConnector: Solo gestión de la base de datos vectorial
- SupabaseConnector: Exclusivamente conexión y configuración de Supabase

O - Open/Closed Principle (Principio Abierto/Cerrado)

El sistema está abierto para extensión pero cerrado para modificación:

Extensibilidad sin modificación:

- Nuevos endpoints: Se pueden añadir en /api/endpoints/ sin modificar rutas existentes

- Nuevos servicios: Se integran mediante inyección de dependencias sin cambiar servicios existentes
- Nuevos repositorios: Siguen la misma interfaz CRUD sin afectar implementaciones actuales
- Nuevos conectores: Se añaden en /utils/ manteniendo las interfaces establecidas

Ejemplo práctico:

```
python
# Extensión sin modificación - Nuevo servicio de analytics
class AnalyticsService:
    def __init__(self):
        self.statistics_service = StatisticsService() # Reutiliza servicios existentes
        self.chat_repository = ChatRepository() # Sin modificar repositorios

    def generate_user_insights(self, user_id: int):
        # Nueva funcionalidad sin tocar código existente
        pass
```

Configuración extensible:

- Nuevas variables de entorno se añaden en settings.py sin afectar configuraciones existentes
- Nuevos modelos de dominio se crean sin modificar entidades establecidas

L - Liskov Substitution Principle (Principio de Sustitución de Liskov)

Los objetos de las clases derivadas pueden sustituir a los de las clases base sin alterar el funcionamiento:

Intercambiabilidad de implementaciones:

- Todos los repositorios implementan operaciones CRUD consistentes (get(), create(), update(), delete())
- Los servicios pueden intercambiarse manteniendo las mismas interfaces
- Los conectores externos mantienen métodos compatibles

Ejemplo de sustitución:

```
python
```

```
# En dependencias - cualquier repositorio puede sustituir a otro manteniendo la interfaz
```

```
def get_repository(entity_type: str):  
    if entity_type == "user":  
        return UserRepository()    # Sustituible  
    elif entity_type == "chat":  
        return ChatRepository()    # Sustituible  
    elif entity_type == "document":  
        return DocumentRepository() # Sustituible
```

Jerarquía de excepciones intercambiables:

- AppException como clase base
- NotFoundException, ValidationException, etc. pueden sustituir a la clase base
- Manejo consistente de errores en toda la aplicación

I - Interface Segregation Principle (Principio de Segregación de Interfaces)

Las interfaces son específicas y no fuerzan a implementar métodos innecesarios:

Servicios especializados:

- UserService no incluye métodos de chat o documentos
- ChatService no contiene lógica de usuarios o documentos
- DocumentService se enfoca exclusivamente en procesamiento de archivos

Repositorios focalizados:

- Cada repositorio implementa solo los métodos CRUD necesarios para su entidad
- No hay métodos genéricos que fuercen implementaciones vacías

Conectores específicos:

- GeminiConnector: Solo métodos para IA (generate_chat_completion, create_embeddings)
- ChromaDBConnector: Solo operaciones vectoriales (search, store_embeddings)

- SupabaseConnector: Solo gestión de conexiones (get_client, test_connection)

D - Dependency Inversion Principle (Principio de Inversión de Dependencias)

Las capas superiores no dependen de las inferiores; ambas dependen de abstracciones:

Inyección de dependencias centralizada (/api/dependencies/):

python

Los servicios no crean sus dependencias directamente

class ChatService:

```
def __init__(self):
    self.chat_repository = ChatRepository()      # Inyectado
    self.message_repository = MessageRepository() # Inyectado
    self.ai_connector = OpenAIConnector()        # Inyectado
```

Abstracción de la persistencia:

- Los servicios no conocen si usan Supabase, PostgreSQL u otra BD
- Los repositorios abstraen completamente el acceso a datos
- Cambiar de Supabase a otra BD solo requiere modificar repositorios

Abstracción de servicios externos:

- ChatService no depende directamente de Gemini
- Usa GeminiConnector que abstrae la implementación de IA
- Cambiar de Gemini a OpenAI solo requiere modificar el conector

Configuración independiente:

- Las capas superiores reciben configuración, no la buscan directamente
- Settings se inyecta donde se necesita
- Variables de entorno se abstraen mediante la clase Settings

Beneficios de la Aplicación de Principios SOLID

Mantenibilidad mejorada:

- Cambios en una funcionalidad no afectan otras partes del sistema
- Debugging más sencillo por responsabilidades claras
- Refactoring seguro con interfaces bien definidas

Extensibilidad garantizada:

- Nuevas funcionalidades se añaden sin modificar código existente
- Integración de nuevos servicios externos sin impacto
- Escalabilidad horizontal mediante nuevos componentes

Testabilidad óptima:

- Cada componente se puede probar independientemente
- Mocking sencillo gracias a la inyección de dependencias
- Pruebas unitarias, de integración y end-to-end bien separadas

Flexibilidad técnica:

- Cambio de tecnologías subyacentes sin afectar lógica de negocio
- Reutilización de componentes en diferentes contextos
- Adaptación a nuevos requisitos con mínimo impacto

Esta implementación sistemática de los principios SOLID asegura que el backend del chatbot sea robusto, mantenable y preparado para evolucionar con los requerimientos futuros del sistema.

3.2 Tecnologías y Dependencias

3.2.1 Stack tecnológico principal

El backend del sistema de chatbot está construido utilizando un stack tecnológico moderno y robusto, optimizado para aplicaciones de inteligencia artificial y procesamiento de documentos a gran escala.

Framework Principal

- FastAPI 0.104.1: Framework web moderno y de alto rendimiento para APIs REST
 - Generación automática de documentación OpenAPI/Swagger
 - Validación automática de datos con Pydantic
 - Soporte nativo para programación asíncrona
 - Tipado estático para mejor desarrollo y debugging
- Uvicorn 0.24.0: Servidor ASGI de alto rendimiento
 - Configuración optimizada para manejo de archivos grandes
 - Soporte para concurrencia y paralelismo
 - Integración perfecta con FastAPI

Base de Datos y Persistencia

- Supabase 2.0.3: Backend-as-a-Service (BaaS) con PostgreSQL
 - Base de datos PostgreSQL completamente gestionada
 - Autenticación y autorización integrada
 - API REST automática
 - Dashboard de administración
- ChromaDB 0.4.18: Base de datos vectorial especializada para RAG
 - Almacenamiento eficiente de embeddings
 - Búsqueda semántica de alta velocidad
 - Persistencia local con volúmenes Docker
- psycopg2-binary 2.9.9: Driver optimizado para PostgreSQL

Inteligencia Artificial y Machine Learning

- Google Generative AI 0.3.0: SDK oficial para Gemini
 - Integración con modelos Gemini 2.0 Flash
 - Generación de texto y embeddings
 - Configuración optimizada para RAG
- OpenAI 1.3.5: Compatibilidad con modelos OpenAI (backup)
 - Fallback para casos de alta demanda
 - Flexibilidad en la elección de modelos
- tiktoken 0.5.1: Tokenización eficiente para modelos de lenguaje
 - Conteo preciso de tokens
 - Optimización de costos de API

Procesamiento de Documentos

- PyPDF2 3.0.1: Extracción de texto de archivos PDF
- pdfplumber 0.10.3: Análisis avanzado de estructura PDF
- python-docx 1.1.0: Procesamiento de documentos Word
- pandas 2.1.3: Manipulación y análisis de datos estructurados
- openpyxl 3.1.2: Lectura y escritura de archivos Excel

Seguridad y Autenticación

- python-jose[cryptography] 3.3.0: Implementación JWT completa
 - Generación y validación de tokens
 - Algoritmos criptográficos seguros
- passlib[bcrypt] 1.7.4: Hashing seguro de contraseñas
 - Algoritmo bcrypt con salt automático
 - Protección contra ataques de fuerza bruta

Gestión de Configuración

- pydantic 2.5.0: Validación de datos y configuración
- pydantic-settings 2.1.0: Gestión de configuraciones con variables de entorno
- python-dotenv 1.0.0: Carga de variables de entorno desde archivos .env
- python-decouple 3.8: Separación de configuración del código

Utilidades y Herramientas

- httpx 0.25.2: Cliente HTTP asíncrono moderno
- tenacity 8.2.3: Librería de reintentos para operaciones robustas
- Pillow 10.1.0: Procesamiento de imágenes
- loguru 0.7.2: Sistema de logging avanzado

Desarrollo y Testing

- pytest 7.4.3: Framework de testing robusto
- pytest-asyncio 0.21.1: Soporte para testing asíncrono
- black 23.11.0: Formateador automático de código
- flake8 6.1.0: Linter para calidad de código

Producción

- gunicorn 21.2.0: Servidor WSGI para despliegue en producción
- python-multipart 0.0.6: Manejo de formularios multipart/form-data

3.2.2 Gestión de dependencias (requirements.txt)

El proyecto utiliza un archivo requirements.txt centralizado que especifica todas las dependencias con versiones fijas para garantizar reproducibilidad:

```
# Core Framework
fastapi==0.104.1
uvicorn[standard]==0.24.0
python-multipart==0.0.6

# Database
supabase==2.0.3
psycopg2-binary==2.9.9

# Authentication & Security
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
```

```
python-decouple==3.8

# AI & ML
openai==1.3.5
google-generativeai==0.3.0
chromadb==0.4.18
tiktoken==0.5.1

# File Processing
PyPDF2==3.0.1
pdfplumber==0.10.3
python-docx==1.1.0
pandas==2.1.3
openpyxl==3.1.2

# Image Processing
Pillow==10.1.0

# Utilities
pydantic==2.5.0
pydantic-settings==2.1.0
python-dotenv==1.0.0
httpx==0.25.2
tenacity==8.2.3

# Development
pytest==7.4.3
pytest-asyncio==0.21.1
black==23.11.0
flake8==6.1.0

# Production
gunicorn==21.2.0

# Logging
loguru==0.7.2
```

Estrategia de Versionado

- Versiones fijas: Todas las dependencias tienen versiones específicas para evitar incompatibilidades
- Compatibilidad: Versiones seleccionadas por compatibilidad mutua probada
- Seguridad: Actualizaciones periódicas para parches de seguridad
- Estabilidad: Pruebas exhaustivas antes de actualizar versiones principales

Instalación de Dependencias

```
# Instalación estándar
pip install -r requirements.txt

# Instalación con cache para acelerar reinstalaciones
pip install --cache-dir ~/.pip/cache -r requirements.txt

# Instalación para desarrollo (incluye herramientas de testing)
pip install -r requirements.txt --upgrade
```

3.2.3 Entorno virtual y activación

El proyecto utiliza entornos virtuales de Python para aislar las dependencias y garantizar consistencia entre diferentes entornos de desarrollo y producción.

Configuración del Entorno Virtual

Python Version Management:

```
# Especificación de versión Python
cat .python-version
# Output: 3.7.4
```

Creación del Entorno Virtual:

```
# Navegar al directorio del backend
cd chabot/back

# Crear entorno virtual
python -m venv .venv
```

```
# En sistemas Unix/Linux/macOS
python3.7 -m venv .venv
```

Activación del Entorno:

```
# Windows
.venv\Scripts\activate

# Unix/Linux/macOS
source .venv/bin/activate

# Verificar activación (debe mostrar el path del entorno virtual)
which python
```

Desactivación del Entorno:

```
# En cualquier sistema  
deactivate
```

Estructura del Entorno Virtual

```
.venv/  
└── Scripts/      # Windows - Executables y scripts  
└── bin/          # Unix/Linux - Executables y scripts  
└── Lib/          # Librerías Python instaladas  
└── Include/      # Headers de C para extensiones  
└── pyenv.cfg     # Configuración del entorno virtual
```

Verificación del Entorno

```
# Verificar que pip usa el entorno virtual  
pip --version
```

```
# Listar paquetes instalados en el entorno  
pip list
```

```
# Verificar ubicación de Python  
python -c "import sys; print(sys.executable)"
```

Buenas Prácticas del Entorno Virtual

- Exclusión del control de versiones: .venv/ está en .gitignore
- Documentación clara: Instrucciones de setup en README
- Activación automática: Scripts de desarrollo que activan automáticamente el entorno
- Reproducibilidad: requirements.txt permite recrear el entorno exacto

3.2.4 Variables de entorno (.env)

El sistema utiliza un archivo .env para gestionar la configuración sensible y específica del entorno, siguiendo las mejores prácticas de seguridad y despliegue.

Estructura de Configuración

Configuración del Servidor:

```
# Configuración general  
HOST=127.0.0.1  
PORT=2690
```

Base de Datos Supabase:

```
# Supabase
SUPABASE_URL=https://lelsehigjjknxjwvxupv.supabase.co
SUPABASE_KEY=eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9...
SUPABASE_SERVICE_KEY=eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9...

# Database URL para conexiones directas
DATABASE_URL=postgresql://postgres.lelsehigjjknxjwvxupv:password@aws-0-
us-west-1.pooler.supabase.com:6543/postgres
```

ChromaDB (Base de Datos Vectorial):

```
CHROMA_HOST=localhost
CHROMA_PORT=8050
CHROMA_TELEMETRY_ENABLED=false
CHROMA_SERVER_TIMEOUT=300
```

Inteligencia Artificial (Gemini):

```
# Gemini
GEMINI_API_KEY=AlzaSyBU61kK_MwSDV-RsnrlEt6JEWRuisyVi1E
GEMINI_MODEL=gemini-2.0-flash
GEMINI_EMBEDDING_MODEL=text-embedding-004
```

Seguridad y Autenticación:

```
# Seguridad JWT
SECRET_KEY=2ee4b09429eae9916ce160621db540db25d5b10677454c3d08801
8d8d3e4dccd
JWT_SECRET_KEY=2ee4b09429eae9916ce160621db540db25d5b10677454c3d0
88018d8d3e4dccd
JWT_ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=57600
```

Configuración RAG Optimizada:

```
# Configuración RAG - Optimizada para documentos largos
RAG_CHUNK_SIZE=1500      # Chunks más grandes para mejor contexto
RAG_CHUNK_OVERLAP=150     # Overlap incrementado para continuidad
RAG_MAX_TOKENS=2000       # Respuestas más completas
RAG_NUM_RESULTS=5         # Múltiples fuentes para precisión
```

Procesamiento de Documentos:

```
# Límites de documentos
MAX_DOCUMENT_SIZE=100      # MB - Tamaño máximo por documento
```

```
DOCUMENT_PROCESSING_TIMEOUT=180 # Segundos - Timeout para  
procesamiento
```

Configuración de Email (SMTP):

```
SMTP_HOST=smtp.gmail.com  
SMTP_PORT=587  
SMTP_USER=heily1857@gmail.com  
SMTP_PASSWORD=avnu mfjz dkwl fuxy  
FROM_EMAIL=heily1857@gmail.com
```

URLs de Frontend:

```
FRONTEND_URL=http://localhost:53793
```

Entorno de Contenedores:

```
DOCKER_ENV=true
```

Seguridad de Variables de Entorno

Protección de Credenciales:

- Archivo .env incluido en .gitignore para prevenir exposición accidental
- Uso de variables separadas para diferentes niveles de acceso (anon key vs service key)
- Rotación periódica de claves API y tokens

Generación Segura de Claves:

```
# Generación de SECRET_KEY segura  
python -c "import secrets; print(secrets.token_hex(32))"
```

Jerarquía de Configuración:

1. Variables de entorno del sistema (mayor prioridad)
2. Archivo .env local
3. Valores por defecto en settings.py

Gestión por Entornos

Desarrollo Local:

- Archivo .env con configuraciones de desarrollo
- Servicios locales (ChromaDB en Docker)
- Logs detallados habilitados

Staging/Testing:

- Variables específicas para entorno de pruebas
- Bases de datos de testing separadas
- Configuraciones de rate limiting reducidas

Producción:

- Variables de entorno del sistema para mayor seguridad
- Configuraciones optimizadas para rendimiento
- Monitoreo y logging configurado para producción

Validación de Configuración

El sistema incluye validación automática de variables de entorno críticas:

```
# En settings.py - Validación automática
required_env_vars = [
    "SUPABASE_URL",
    "SUPABASE_KEY",
    "SUPABASE_SERVICE_KEY",
    "GEMINI_API_KEY"
]

missing_vars = [var for var in required_env_vars if not os.getenv(var)]
if missing_vars:
    raise ValueError(f"Faltan variables de entorno requeridas: {',
'.join(missing_vars)}")
```

Esta configuración garantiza que el sistema sea seguro, configurable y mantenible a través de diferentes entornos de despliegue.

3.3 Capa de API (src/api)

La capa de API constituye el punto de entrada del sistema, implementando endpoints REST que exponen la funcionalidad del chatbot de manera organizada y segura. Esta capa sigue los principios RESTful y está diseñada para ser escalable, mantenible y bien documentada.

3.3.1 Estructura de rutas (routes.py)

El sistema utiliza un enrutador principal que organiza todos los endpoints de forma modular y jerárquica:

```
# src/api/routes.py
```

```

from fastapi import APIRouter
from src.api.endpoints import documents, chat, users, health, statistics

api_router = APIRouter()

# Incluir los routers de endpoints
api_router.include_router(health.router)    # Health checks
api_router.include_router(users.router)      # Autenticación y usuarios
api_router.include_router(documents.router)  # Gestión de documentos
api_router.include_router(chat.router)       # Chat con RAG integrado
api_router.include_router(statistics.router) # Estadísticas globales

```

Organización Jerárquica

La estructura de rutas sigue una jerarquía lógica que refleja los dominios de negocio:

Prefijo Base: /api/

- Todas las rutas están prefijadas con /api/ para versionado y organización
- Separación clara entre la API y posibles interfaces web futuras

Dominios Principales:

1. Health (/api/health/): Monitoreo y estado del sistema
2. Users (/api/users/): Gestión de usuarios y autenticación
3. Documents (/api/documents/): Operaciones CRUD de documentos
4. Chats (/api/chats/): Sistema de conversación con IA
5. Statistics (/api/statistics/): Métricas y análisis del sistema

Beneficios de la Arquitectura de Rutas

- Modularidad: Cada dominio es independiente y puede desarrollarse por separado
- Escalabilidad: Fácil adición de nuevos dominios sin afectar existentes
- Mantenibilidad: Cambios en un dominio no impactan otros
- Documentación automática: FastAPI genera documentación Swagger automáticamente

3.3.2 Endpoints por dominio

3.3.2.1 Gestión de usuarios (users.py)

El módulo de usuarios implementa un sistema completo de autenticación y gestión de cuentas con seguridad robusta:

Autenticación y Autorización:

```
# Endpoints principales de autenticación
POST /api/users/login      # Autenticación con username/email + contraseña
POST /api/users/register    # Registro de nuevos usuarios
POST /api/users/logout       # Cierre de sesión e invalidación de tokens
POST /api/users/refresh-token # Renovación de tokens de acceso
```

Gestión de Perfiles:

```
# Operaciones CRUD de usuarios
GET /api/users/me           # Información del usuario autenticado
PUT /api/users/me            # Actualización del perfil propio
GET /api/users/{user_id}     # Información de usuario específico
PUT /api/users/{user_id}     # Actualización de usuario (admin/propio)
DELETE /api/users/{user_id}  # Eliminación de usuario (solo admin)
```

Gestión de Contraseñas:

```
# Recuperación y cambio de contraseñas
POST /api/users/forgot-password      # Solicitar reset por email
POST /api/users/reset-password       # Reset con token
POST /api/users/me/change-password   # Cambio autenticado
PUT /api/users/{user_id}/change-password # Cambio administrativo
```

Búsqueda y Administración:

```
# Funciones administrativas
GET /api/users/                 # Listar usuarios (solo admin)
GET /api/users/search            # Buscar usuarios por criterios
```

Gestión de Avatares:

```
# Manejo de imágenes de perfil
POST /api/users/{user_id}/avatar  # Subir/actualizar avatar
DELETE /api/users/{user_id}/avatar # Eliminar avatar
```

Verificación de Email:

```
# Sistema de verificación por correo
POST /api/users/verify-email      # Verificar con token
POST /api/users/resend-verification # Reenviar email de verificación
```

Características de Seguridad

- JWT Authentication: Tokens seguros con expiración configurable

- Bcrypt Password Hashing: Contraseñas hasheadas con salt
- Rate Limiting: Protección contra ataques de fuerza bruta
- Input Validation: Validación exhaustiva con Pydantic
- Permission Checks: Control granular de permisos por operación

3.3.2.2 Gestión de documentos (*documents.py*)

El sistema de documentos implementa un pipeline completo de procesamiento con capacidades avanzadas de RAG:

Carga de Documentos:

```
# Upload híbrido síncrono/asíncrono
POST /api/documents/upload    # Carga con procesamiento inteligente
POST /api/documents/          # Creación manual de documento
```

Gestión CRUD:

```
# Operaciones básicas
GET  /api/documents/          # Listar documentos del usuario
GET  /api/documents/{document_id}  # Obtener documento específico
PUT   /api/documents/{document_id} # Actualizar metadata
DELETE /api/documents/{document_id} # Eliminar documento
```

Búsqueda y Descubrimiento:

```
# Búsqueda semántica avanzada
GET /api/documents/search      # Búsqueda por contenido
GET /api/documents/shared       # Documentos compartidos
```

Compartición y Colaboración:

```
# Sistema de permisos granular
POST /api/documents/{document_id}/share    # Compartir con usuarios
POST /api/documents/{document_id}/users     # Vincular usuarios
GET  /api/documents/{document_id}/users     # Listar usuarios con acceso
DELETE /api/documents/{document_id}/users/{user_id} # Revocar acceso
```

Monitoreo y Mantenimiento:

```
# Gestión avanzada
GET /api/documents/{document_id}/status    # Estado de procesamiento
POST /api/documents/{document_id}/reindex   # Re-indexar en ChromaDB
GET  /api/documents/{document_id}/verify-index # Verificar indexación
```

Pipeline de Procesamiento Híbrido

Procesamiento Síncrono (archivos pequeños):

- Archivos PDF < 1MB: Procesamiento inmediato
- Archivos de texto < 500KB: Extracción directa
- Respuesta inmediata con documento procesado

Procesamiento Asíncrono (archivos grandes):

- Background tasks para archivos > 3MB
- Estados de progreso consultables
- Notificaciones de completado

Tipos de Archivo Soportados:

- PDF: Extracción de texto con pdfplumber y PyPDF2
- TXT: Procesamiento directo con encoding UTF-8
- CSV/Excel: Análisis estructurado con pandas
- Word: Extracción con python-docx

3.3.2.3 Sistema de chat (chat.py)

El módulo de chat implementa la funcionalidad central del sistema con integración RAG completa:

Gestión de Conversaciones:

```
# CRUD de chats
POST /api/chats/          # Crear nueva conversación
GET /api/chats/           # Listar chats del usuario
GET /api/chats/{chat_id}   # Obtener chat con mensajes
PUT /api/chats/{chat_id}   # Actualizar metadata del chat
PUT /api/chats/{chat_id}/rename # Renombrar chat específicamente
DELETE /api/chats/{chat_id} # Eliminar chat y mensajes
```

Gestión de Mensajes:

```
# Operaciones con mensajes
POST /api/chats/{chat_id}/messages # Enviar mensaje con RAG
GET /api/chats/{chat_id}/messages # Obtener historial de mensajes
DELETE /api/chats/{chat_id}/messages/{message_id} # Eliminar mensaje
específico
```

Funcionalidades RAG Avanzadas

Integración Contextual:

- Selección de documentos específicos por chat
- Configuración dinámica de número de resultados
- Búsqueda semántica en documentos seleccionados

Tipos de Consulta:

1. Preguntas sobre documentos: RAG con documentos seleccionados
2. Listado de documentos: Respuestas automáticas sobre biblioteca personal
3. Chat general: Conversación sin contexto documental
4. Preguntas sin documento seleccionado: Guía para selección

Ejemplo de Mensaje con RAG:

```
{  
    "question": "¿Cuál es el objetivo principal del proyecto?",  
    "document_ids": [1, 3, 5],  
    "n_results": 5  
}
```

3.3.3 Dependencias compartidas (dependencies.py)

El sistema de dependencias centraliza la inyección de servicios y la gestión de autenticación:

```
# src/api/dependencies/__init__.py  
from fastapi import Depends, HTTPException, status, Query  
from fastapi.security import OAuth2PasswordBearer  
from src.services.document_service import DocumentService  
from src.services.user_service import UserService  
from src.services.statistics_service import StatisticsService  
  
# OAuth2 scheme para autenticación  
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/api/users/login",  
auto_error=False)  
  
# Instancias singleton de servicios  
document_service = DocumentService()  
user_service = UserService()  
statistics_service = StatisticsService()
```

Funciones de Dependencia Principal

Autenticación de Usuario:

```
async def get_current_user(  
    token: Optional[str] = Depends(oauth2_scheme),  
    token_query: Optional[str] = Query(None, alias="token")  
) -> User:  
    """
```

Valida token JWT y retorna usuario autenticado.

Soporta tokens en header Authorization y query parameter.

"""

Inyección de Servicios:

```
def get_document_service() -> DocumentService:  
    """Retorna instancia singleton del servicio de documentos"""  
    return document_service
```

```
def get_statistics_service() -> StatisticsService:  
    """Retorna instancia singleton del servicio de estadísticas"""  
    return statistics_service
```

Características del Sistema de Dependencias

- Singleton Pattern: Una instancia por servicio en toda la aplicación
- Lazy Loading: Servicios se instancian solo cuando se necesitan
- Token Flexibility: Soporte para tokens en headers y query parameters
- Error Handling: Manejo consistente de errores de autenticación
- Admin Detection: Reconocimiento automático de usuarios administradores

3.3.4 Middleware y CORS

La aplicación implementa una configuración robusta de middleware para seguridad, CORS y manejo de archivos grandes:

```
# src/main.py - Configuración de CORS  
from fastapi.middleware.cors import CORSMiddleware  
  
app.add_middleware(  
    CORSMiddleware,  
    allow_origins=["*"],           # Configuración permisiva para desarrollo  
    allow_credentials=True,        # Soporte para cookies/headers auth  
    allow_methods=["*"],           # Todos los métodos HTTP
```

```
    allow_headers=["*"],          # Todos los headers
)
Configuración del Servidor
```

Configuraciones de Uvicorn para Archivos Grandes:

```
@app.on_event("startup")
async def startup():
    # Configuraciones para manejar archivos grandes
    uvicorn.config.LIFESPAN_ON_STARTUP = True
    uvicorn.configLOOP_WAIT = 0.1
    uvicorn.config.HTTP_TIMEOUT_KEEP_ALIVE = 120
```

Configuración de Producción:

```
config = uvicorn.Config(
    "src.main:app",
    host="127.0.0.1",
    port=2690,
    reload=True,
    limit_concurrency=1000,    # Conexiones concurrentes máximas
    limit_max_requests=10000,  # Peticiones máximas por conexión
    timeout_keep_alive=120,    # Timeout para mantener conexiones
    backlog=2048,             # Cola de conexiones pendientes
)
Middleware Implícito de FastAPI
```

Documentación Automática:

- Swagger UI disponible en /docs
- ReDoc disponible en /redoc
- Schema OpenAPI en /openapi.json

Validación Automática:

- Validación de tipos con Pydantic
- Serialización/deserialización automática
- Manejo de errores de validación

Logging y Monitoreo:

- Logs estructurados con timestamps
- Tracking de requests/responses
- Manejo centralizado de excepciones

Seguridad Implementada

Headers de Seguridad:

- CORS configurado para desarrollo y producción
- Soporte para credenciales cross-origin
- Headers personalizados permitidos

Rate Limiting (pendiente de implementación):

- Protección contra abuse de API
- Límites por usuario y endpoint
- Ventanas deslizantes de tiempo

Input Sanitization:

- Validación exhaustiva de parámetros
- Prevención de inyección SQL/NosQL
- Sanitización de contenido de archivos

Manejo de Errores Global

Exception Handlers Personalizados:

```
# Mapeo de excepciones del dominio a HTTP status codes
NotFoundException → HTTP 404
UnauthorizedException → HTTP 401/403
ValidationException → HTTP 400
ConflictException → HTTP 409
```

Respuestas de Error Consistentes:

- Formato JSON uniforme para errores
- Códigos de error descriptivos
- Mensajes localizables para el frontend

Esta arquitectura de API proporciona una interfaz robusta, segura y escalable que soporta eficientemente las operaciones del chatbot con capacidades RAG, manteniendo un alto nivel de organización y facilidad de mantenimiento.

3.4 Modelos de Datos (src/models)

La capa de modelos implementa una arquitectura robusta de separación entre modelos de dominio y esquemas de validación, siguiendo el patrón Domain-

Driven Design (DDD) y aprovechando las capacidades avanzadas de Pydantic para validación y serialización.

3.4.1 Modelos de dominio (domain)

Los modelos de dominio representan las entidades de negocio puras del sistema, sin dependencias de frameworks externos y enfocados en la lógica de negocio esencial.

Estructura de Entidades

```
# src/models/domain/__init__.py
from datetime import datetime
from typing import Optional, List, Any
from pydantic import BaseModel, Field, ConfigDict
from uuid import UUID

User - Entidad de Usuario
class User(BaseModel):
    """Modelo de usuario - tabla users"""
    id: Optional[int] = None
    username: Optional[str] = None
    email: Optional[str] = None
    password_hash: Optional[str] = None
    created_at: Optional[datetime] = None
    updated_at: Optional[datetime] = None
    is_admin: bool = False
    email_encrypted: Optional[bytes] = None
    auth_id: Optional[UUID] = None
    avatar_url: Optional[str] = None
    email_verified: bool = False
    last_login: Optional[datetime] = None
    reset_token: Optional[str] = None
    reset_token_expires: Optional[datetime] = None
    verification_token: Optional[str] = None
    verification_token_expires: Optional[datetime] = None
    refresh_token: Optional[str] = None

    model_config = ConfigDict(from_attributes=True)
```

Características de la Entidad User:

- Flexibilidad: Todos los campos opcionales para soportar diferentes estados
- Seguridad: Separación entre email y email_encrypted para protección de datos

- Autenticación: Soporte completo para JWT con auth_id único
- Gestión de sesiones: Tokens de refresh para renovación automática
- Verificación: Sistema completo de verificación por email
- Recuperación: Tokens temporales para reset de contraseñas

Document - Entidad de Documento

```
class Document(BaseModel):
    """Modelo de documento - tabla documents"""
    id: Optional[int] = None
    title: Optional[str] = None
    chromadb_id: Optional[str] = None
    uploaded_by: Optional[int] = None
    content_type: Optional[str] = None # PDF, texto, etc.
    created_at: Optional[datetime] = None
    updated_at: Optional[datetime] = None
    file_url: Optional[str] = None
    status: str = Field(default='pending')
    status_message: Optional[str] = None
    file_size: Optional[int] = None
    original_filename: Optional[str] = None
    content: Optional[str] = None # IMPORTANTE: Campo para el contenido

    model_config = ConfigDict(
        from_attributes=True,
        extra='allow' # Permitir campos adicionales temporalmente
    )
```

Características de la Entidad Document:

- Estados de procesamiento: Campo status para tracking de procesamiento
- Integración RAG: chromadb_id para vinculación con base de datos vectorial
- Metadatos completos: Información de archivo original y procesado
- Flexibilidad: Campo content para almacenar texto extraído
- Trazabilidad: Tracking completo de usuario que subió y fechas

Chat - Entidad de Conversación

```
class Chat(BaseModel):
    """Modelo de chat - tabla chats"""
    id: Optional[int] = None
    id_user: Optional[int] = None
    name_chat: Optional[str] = None
    created_at: Optional[datetime] = None
```

```
model_config = ConfigDict(from_attributes=True)

Message - Entidad de Mensaje
class Message(BaseModel):
    """Modelo de mensaje - tabla messages"""
    id: Optional[int] = None
    id_chat: Optional[int] = None
    question: Optional[str] = None
    answer: Optional[str] = None
    created_at: Optional[datetime] = None
```

```
model_config = ConfigDict(from_attributes=True)

AccesoDocumentosUsuario - Entidad de Permisos
class AccesoDocumentosUsuario(BaseModel):
    """Modelo de acceso a documentos - tabla acceso_documentos_usuario"""
    id_document: int
    id_user: int
    linked_time: Optional[datetime] = None
```

```
model_config = ConfigDict(from_attributes=True)

# Alias para compatibilidad
DocumentAccess = AccesoDocumentosUsuario

Principios de Diseño de Entidades
```

- Inmutabilidad: Entidades diseñadas para ser inmutables cuando es posible
- Validación mínima: Solo validaciones de negocio esenciales
- Flexibilidad: Campos opcionales para soportar diferentes estados del ciclo de vida
- Interoperabilidad: ConfigDict con from_attributes para conversión desde ORM
- Trazabilidad: Timestamps automáticos para auditoría

3.4.2 Esquemas de validación (schemas)

Los esquemas implementan validación robusta, serialización/deserialización y transformación de datos entre capas del sistema.

3.4.2.1 Usuario (user.py)

Esquemas Base y CRUD

```
class UserBase(BaseModel):
```

```
    username: str
```

```
    email: EmailStr
```

```

@field_validator('username')
def username_alphanumeric(cls, v):
    if not v.isalnum():
        raise ValueError('debe contener solo caracteres alfanuméricos')
    return v

class UserCreate(UserBase):
    password: str
    auth_id: Optional[UUID] = None

    @field_validator('password')
    def password_min_length(cls, v):
        if len(v) < 8:
            raise ValueError('debe tener al menos 8 caracteres')
        return v

class UserUpdate(BaseModel):
    username: Optional[str] = None
    email: Optional[EmailStr] = None
    is_admin: Optional[bool] = None

    @field_validator('username')
    def username_alphanumeric(cls, v):
        if v is not None and not v.isalnum():
            raise ValueError('debe contener solo caracteres alfanuméricos')
        return v

Esquemas de Respuesta
class UserResponse(BaseModel):
    """Respuesta completa de usuario"""
    id: int
    username: str
    email: str
    is_admin: bool
    created_at: datetime
    updated_at: datetime
    auth_id: Optional[str] = None
    email_verified: Optional[bool] = False
    avatar_url: Optional[str] = None

    @field_validator('auth_id', mode='before')
    def convert_uuid_to_str(cls, v):
        """Convierte UUID a string si es necesario"""

```

```

if isinstance(v, UUID):
    return str(v)
return v

Esquemas de Autenticación

class ForgotPasswordRequest(BaseModel):
    email: EmailStr

class ResetPasswordRequest(BaseModel):
    token: str
    new_password: str

    @field_validator('new_password')
    def password_min_length(cls, v):
        if len(v) < 8:
            raise ValueError('debe tener al menos 8 caracteres')
        return v

class ChangePasswordRequest(BaseModel):
    current_password: str
    new_password: str

class RefreshTokenRequest(BaseModel):
    refresh_token: str

class TokenResponse(BaseModel):
    access_token: str
    refresh_token: str
    token_type: str = "bearer"
    expires_in: Optional[int] = None

Esquemas de Verificación

class EmailVerificationRequest(BaseModel):
    token: str

class ResendVerificationRequest(BaseModel):
    email: EmailStr

class MessageResponse(BaseModel):
    message: str
    success: bool = True

3.4.2.2 Documentos (document.py)
Esquemas Core

class DocumentBase(BaseModel):
    """Base para documento - campos comunes"""

```

```

title: str
content_type: Optional[str] = Field(None, description="Tipo de documento")

@field_validator('title')
def title_not_empty(cls, v):
    if not v or not v.strip():
        raise ValueError('el título no puede estar vacío')
    return v

class DocumentCreate(DocumentBase):
    """Crear nuevo documento"""
    content: Optional[str] = None
    original_filename: Optional[str] = None
    file_size: Optional[int] = None

class DocumentUpdate(BaseModel):
    """Actualizar documento"""
    title: Optional[str] = None
    content: Optional[str] = None
    content_type: Optional[str] = None
    tags: Optional[List[str]] = None
    status: Optional[str] = None
    status_message: Optional[str] = None
    chromadb_id: Optional[str] = None
    file_url: Optional[str] = None

Esquemas de Respuesta
class DocumentResponse(DocumentBase):
    """Respuesta completa del documento"""
    id: int
    chromadb_id: Optional[str] = None
    uploaded_by: int
    created_at: Optional[datetime] = None
    updated_at: Optional[datetime] = None
    file_url: Optional[str] = None
    status: str = 'pending'
    status_message: Optional[str] = None
    file_size: Optional[int] = None
    original_filename: Optional[str] = None

    model_config = {
        "from_attributes": True
    }

```

Esquemas de Colaboración

```
class DocumentShare(BaseModel):
    """Compartir documento con usuarios"""
    document_id: int
    user_ids: List[int]
```

```
class DocumentUserLink(BaseModel):
    """Vincular usuarios a documentos"""
    user_ids: List[int]
```

```
class DocumentAccess(BaseModel):
    """Acceso a documentos"""
    id_document: int
    id_user: int
    linked_time: Optional[datetime] = None
```

3.4.2.3 Chat y mensajes (chat.py)

Esquemas de Mensajes

```
class MessageBase(BaseModel):
    """Base para mensajes"""
    question: str
    answer: Optional[str] = None
```

```
class MessageCreate(BaseModel):
    """Esquema para crear un nuevo mensaje"""
    question: str = Field(..., min_length=1, max_length=5000)
    document_ids: Optional[List[int]] = Field(None, description="IDs de
documentos para RAG")
    n_results: Optional[int] = Field(5, description="Número de resultados para
RAG")
```

```
class MessageResponse(MessageBase):
    """Respuesta de mensaje"""
    id: Optional[int] = None
    id_chat: int
    created_at: Optional[datetime] = None
```

```
model_config = {
    "from_attributes": True
}
```

Esquemas de Chat

```
class ChatBase(BaseModel):
    """Base para chat"""
    name_chat: Optional[str] = None
```

```

@field_validator('name_chat')
def name_not_empty(cls, v):
    if v is not None and (not v or not v.strip()):
        raise ValueError('Si proporcionas un nombre, no puede estar vacío')
    return v.strip() if v else v

class ChatCreate(BaseModel):
    """Esquema para crear un nuevo chat"""
    name_chat: Optional[str] = Field(None, description="Nombre del chat - OPCIONAL")

    @field_validator('name_chat')
    def name_not_empty(cls, v):
        if v is not None and (not v or not v.strip()):
            raise ValueError('Si proporcionas un nombre, no puede estar vacío')
        return v.strip() if v else v

class ChatResponse(BaseModel):
    """Respuesta de chat"""
    id: int
    name_chat: Optional[str] = None
    id_user: int
    created_at: datetime
    messages: Optional[List[MessageResponse]] = None

    model_config = {
        "from_attributes": True
    }

class ChatRename(BaseModel):
    """Schema específico para renombrar un chat"""
    name: str = Field(..., min_length=1, max_length=100)

    @field_validator('name')
    def validate_name(cls, v):
        if not v or not v.strip():
            raise ValueError('El nombre del chat no puede estar vacío')
        return v.strip()

```

3.4.3 Validación con Pydantic

El sistema aprovecha al máximo las capacidades de Pydantic v2 para implementar validación robusta y transformación de datos.

Características de Validación Implementadas

Validadores de Campo

```
# Validación de username alfanumérico
@field_validator('username')
def username_alphanumeric(cls, v):
    if not v.isalnum():
        raise ValueError('debe contener solo caracteres alfanuméricos')
    return v
```

```
# Validación de longitud mínima de contraseña
```

```
@field_validator('password')
def password_min_length(cls, v):
    if len(v) < 8:
        raise ValueError('debe tener al menos 8 caracteres')
    return v
```

```
# Validación de título no vacío
```

```
@field_validator('title')
def title_not_empty(cls, v):
    if not v or not v.strip():
        raise ValueError('el título no puede estar vacío')
    return v
```

Transformadores de Datos

```
# Conversión de UUID a string
@field_validator('auth_id', mode='before')
def convert_uuid_to_str(cls, v):
    if isinstance(v, UUID):
        return str(v)
    return v
```

```
# Limpieza automática de strings
```

```
@field_validator('name_chat')
def name_not_empty(cls, v):
    if v is not None and (not v or not v.strip()):
        raise ValueError('Si proporcionas un nombre, no puede estar vacío')
    return v.strip() if v else v
```

Configuración de Modelos

```
model_config = {
    "from_attributes": True,  # Conversión desde objetos ORM
    "validate_assignment": True, # Validación en asignaciones
    "extra": "forbid",      # Prohibir campos extra
    "str_strip_whitespace": True # Limpiar espacios automáticamente
}
```

Tipos de Validación Implementados

Validación de Entrada (Input Validation)

- Tipos primitivos: str, int, bool, datetime
- Tipos especializados: EmailStr para emails válidos
- Longitudes: min_length, max_length para strings
- Rangos: Field con constraints para números
- Formatos: UUID, datetime con formatos específicos

Validación de Lógica de Negocio

- Usernames alfanuméricos: Prevención de caracteres especiales
- Contraseñas seguras: Longitud mínima obligatoria
- Títulos no vacíos: Validación de contenido significativo
- Estados válidos: Valores predefinidos para status de documentos

Validación Condicional

- Camposopcionales: Validación solo cuando están presentes
- Validación contextual: Diferentes reglas según el esquema
- Validación de actualización: Permitir campos parciales en updates

Beneficios de la Arquitectura de Validación

Robustez

- Prevención de errores: Validación temprana evita errores en capas inferiores
- Consistencia: Reglas de validación centralizadas y reutilizables
- Seguridad: Validación de entrada previene ataques de inyección

Mantenibilidad

- Separación de responsabilidades: Validación separada de lógica de negocio
- Reutilización: Esquemas base para funcionalidad común
- Evolución: Fácil modificación de reglas de validación

Experiencia del Desarrollador

- Autocomplete: Tipos estrictos para mejor IDE support
- Documentación automática: Pydantic genera documentación de API
- Mensajes claros: Errores de validación descriptivos y localizables

Rendimiento

- Validación compilada: Pydantic v2 usa Rust para validación rápida
- Serialización optimizada: Conversión eficiente entre formatos
- Cache de validadores: Reutilización de validadores compilados

Patrones de Uso Avanzados

Herencia de Esquemas

Esquema base reutilizable

```
class UserBase(BaseModel):
```

```
    username: str
```

```
    email: EmailStr
```

Extensión para creación

```
class UserCreate(UserBase):
```

```
    password: str
```

Extensión para respuesta

```
class UserResponse(UserBase):
```

```
    id: int
```

```
    created_at: datetime
```

Aliases y Compatibilidad

Alias para compatibilidad con código existente

```
ChatMessage = MessageResponse
```

```
DocumentAccess = AccesoDocumentosUsuario
```

```
DocumentResponseHybrid = DocumentResponse
```

Configuración Flexible

```
model_config = ConfigDict(
```

```
    from_attributes=True, # Conversión desde ORMs
```

```
    extra='allow', # Permitir campos adicionales temporalmente
```

```
    validate_assignment=True, # Validar en asignaciones
```

```
    arbitrary_types_allowed=True # Tipos personalizados
```

```
)
```

Esta arquitectura de modelos proporciona una base sólida para el sistema, garantizando integridad de datos, seguridad y facilidad de mantenimiento, mientras mantiene la flexibilidad necesaria para evolucionar con los requerimientos del chatbot.

3.5 Capa de Servicios (src/services)

La capa de servicios constituye el corazón de la lógica de negocio del sistema, implementando una arquitectura de servicios que encapsula las reglas de

negocio, coordina operaciones entre diferentes dominios y proporciona una interfaz limpia para la capa de API.

3.5.1 Lógica de negocio centralizada

La capa de servicios implementa una arquitectura centralizada que garantiza consistencia, reutilización y mantenibilidad de la lógica de negocio.

Principios de Diseño de Servicios

Separación de Responsabilidades:

- Cada servicio gestiona un dominio específico del negocio
- Lógica de negocio separada de la lógica de acceso a datos
- Coordinación entre servicios para operaciones complejas

Patrón de Orquestación:

- Los servicios orquestan llamadas a repositorios
- Implementan reglas de negocio complejas
- Manejan transacciones y consistencia de datos

Inyección de Dependencias:

```
class DocumentService:  
    def __init__(self):  
        self.chromadb = ChromaDBConnector()  
        self.openai = get_openai_connector()  
        self.document_repo = DocumentRepository()  
        self.user_repo = UserRepository()
```

Características Transversales

Manejo de Errores:

- Excepciones de dominio específicas
- Logging estructurado y detallado
- Recuperación de errores y rollback automático

Validación de Negocio:

- Validaciones complejas más allá de la validación de datos
- Reglas de negocio específicas del dominio
- Verificación de permisos y autorización

Auditabilidad:

- Logging de todas las operaciones críticas
- Trazabilidad de cambios y acciones
- Timestamps automáticos en todas las operaciones

3.5.2 Servicio de autenticación (auth_service.py)

El servicio de autenticación implementa un sistema robusto de gestión de identidad y acceso con JWT y funcionalidades avanzadas de seguridad.

Funcionalidades Core

Registro de Usuarios:

```
def register_user(self, username: str, email: str, password: str) -> Dict[str, Any]:  
    """
```

Registra un nuevo usuario con validaciones completas.

- Verificación de username único
- Validación de email único
- Protección contra usuarios similares a 'Ivan'
- Generación automática de auth_id (UUID)
- Creación de token de acceso inmediato

```
"""
```

Características del Registro:

- Validación única: Verificación de username y email únicos
- Protección admin: Previene registro de usuarios similares a "Ivan"
- UUID generation: auth_id único para cada usuario
- Token inmediato: Login automático post-registro

Autenticación de Usuarios:

```
def login_user(self, username: str, password: str) -> Dict[str, Any]:  
    """
```

Autentica usuario con protecciones de seguridad avanzadas.

- Búsqueda case-insensitive de username
- Verificación segura de contraseñas con bcrypt
- Protección contra timing attacks
- Detección automática de admin status

```
"""
```

Características del Login:

- Timing attack protection: Operación de hash dummy para usuarios inexistentes
- Admin detection: Verificación automática del status de administrador
- Case insensitive: Búsqueda flexible de usuarios
- Token generation: JWT con payload completo

Gestión de Tokens JWT

Creación de Tokens de Acceso:

```
def _create_access_token(self, data: Dict[str, Any], expires_delta: Optional[timedelta] = None) -> str:
    """
    Genera tokens JWT seguros con metadatos completos.
```

Payload incluye:

- sub: auth_id del usuario
- user_id: ID numérico del usuario
- exp: Tiempo de expiración
- iat: Tiempo de emisión
- iss: Emisor del token (fastapi-app)

Refresh Tokens:

```
def _create_refresh_token(self, data: Dict[str, Any]) -> str:
    """
    Genera refresh tokens con mayor duración (7 días).
```

Características:

- Duración extendida para renovación automática
- Marcado específico como tipo 'refresh'
- Misma seguridad que access tokens

Validación y Autorización

Validación de Tokens:

```
def get_current_user(self, token: str) -> User:
    """
    Valida token JWT y retorna usuario autenticado.
```

Proceso:

1. Decodificación del token con verificación de firma

2. Extracción de auth_id del payload
 3. Búsqueda del usuario en base de datos
 4. Verificación automática de status admin
-

Características de Seguridad:

- Firma criptográfica: Verificación con clave secreta
- Expiración automática: Tokens con tiempo de vida limitado
- Revocación: Base para implementar blacklisting de tokens
- Usuario actualizado: Datos frescos en cada validación

3.5.3 Servicio de usuarios (user_service.py)

El servicio de usuarios implementa lógica de negocio compleja para gestión de cuentas, permisos y operaciones administrativas.

Gestión de Usuarios

Operaciones CRUD Avanzadas:

```
def update_user(self, user_id: int, user_data: dict, current_user: User) -> User:  
    ....
```

Actualización de usuarios con reglas de negocio complejas.

Reglas implementadas:

- Protección de la cuenta de Ivan (no modificable por otros)
 - Solo Ivan puede ser administrador
 - Usuarios no pueden cambiar su propio nivel de admin
 - Validación de permisos por operación
-

Reglas de Negocio de Usuarios:

1. Protección de Ivan: Usuario "ivan" tiene protecciones especiales
2. Administrador único: Solo "ivan" puede ser administrador
3. Auto-modificación limitada: Usuarios no pueden escalar privilegios
4. Eliminación protegida: Ivan no puede ser eliminado

Gestión de Contraseñas

Reset por Email:

```
def request_password_reset(self, email: str) -> bool:  
    ....
```

Solicita reset de contraseña con token seguro.

Proceso:

1. Verificación de existencia del email (sin revelar)
 2. Generación de token de 6 dígitos
 3. Hash del token para almacenamiento seguro
 4. Expiración de 1 hora
 5. Envío de email con token
-

Cambio Autenticado:

```
def change_password(self, user_id: int, current_password: str, new_password: str, current_user: User) -> bool:  
    """
```

Cambio de contraseña con verificación.

Validaciones:

- Verificación de contraseña actual
 - Permisos de modificación
 - Hash seguro de nueva contraseña
-

Búsqueda y Filtrado

Búsqueda Administrativa:

```
def search_users(self, username_query: str, current_user: User, limit: int = 100)  
-> List[User]:  
    """
```

Búsqueda de usuarios para administradores.

Características:

- Solo disponible para administradores
 - Búsqueda parcial por username
 - Resultados paginados
-

3.5.4 Servicio de documentos (document_service.py)

El servicio de documentos implementa el pipeline completo de procesamiento de documentos con capacidades RAG avanzadas.

Pipeline de Procesamiento de Documentos

Creación de Documentos:

```
def create_document(self, uploaded_by: int, title: str, content: str,  
                    content_type: str, tags: Optional[List[str]] = None,  
                    file_url: Optional[str] = None) -> Document:  
    """
```

Pipeline completo de creación de documentos.

Proceso:

1. Extracción de texto según tipo de archivo
2. Validación de contenido
3. División en chunks optimizados
4. Almacenamiento en base de datos relacional
5. Indexación en ChromaDB para RAG
6. Actualización con referencias cruzadas

```
"""
```

Extracción de Texto Multi-formato:

```
def extract_text_from_pdf(self, pdf_content: bytes) -> str:  
    """
```

Extracción robusta de texto de PDFs.

Características:

- Soporte para PyPDF2 y pdfplumber (fallback)
- Limpieza automática de texto extraído
- Manejo de errores graceful
- Logging detallado de proceso

```
"""
```

Algoritmo de Chunking Inteligente

División Optimizada:

```
def _split_text_into_chunks(self, text: str, content_type: str = None,  
                           max_chunk_size: int = 1000, overlap: int = 100) -> list:  
    """
```

División inteligente según tipo de contenido.

Parámetros adaptativos:

- text/plain: chunks de 2000 caracteres, overlap 50
- application/pdf: chunks de 1000 caracteres, overlap 100
- Otros formatos: configuración estándar

.....

Características del Chunking:

- Adaptativo: Parámetros según tipo de contenido
- Overlap inteligente: Preserva contexto entre chunks
- Optimizado para RAG: Tamaños ideales para búsqueda semántica

Sistema RAG Avanzado

Búsqueda Contextual:

```
def get_rag_response(self, query: str, user_id: int = None, n_results: int = 5,
                     document_ids: Optional[List[int]] = None) -> dict:
    .....
```

Búsqueda semántica con generación aumentada.

Modalidades:

1. Documentos específicos: Filtrado por document_ids
2. Usuario completo: Todos los documentos del usuario
3. Búsqueda abierta: Sin filtros (si se permite)

Retorna:

- context: Texto de chunks relevantes
- response: Respuesta generada por IA
- documents: Metadatos de documentos usados

.....

Proceso RAG Detallado:

1. Filtrado: Construcción de filtros WHERE para ChromaDB
2. Búsqueda vectorial: Similarity search en embeddings
3. Extracción de contexto: Agregación de chunks relevantes
4. Generación: Prompt engineering + LLM completion
5. Metadatos: Tracking de documentos utilizados

Gestión de Estados

Estados de Procesamiento:

```
def update_document_status(self, document_id: int, status: str, message: str,
                           Optional[str] = None):
    .....
```

Tracking de estados de procesamiento.

Estados:

- pending: Documento creado, esperando procesamiento
- processing: Extracción/indexación en progreso
- completed: Procesamiento exitoso
- error: Error en procesamiento
- warning: Procesado con advertencias

"""

Verificación de Indexación:

```
def verify_document_indexed(self, document_id: int) -> bool:
```

"""

Verifica presencia en ChromaDB.

Útil para:

- Debugging de problemas de indexación
- Verificación post-procesamiento
- Health checks de documentos

"""

Sistema de Permisos y Compartición

Compartición de Documentos:

```
def share_document(self, document_id: int, user_ids: List[int], requester_id: int) -> bool:
```

"""

Sistema granular de permisos.

Validaciones:

- Existencia del documento
- Permisos del solicitante (propietario o admin)
- Validez de usuarios destinatarios

"""

Control de Acceso:

```
def check_user_access(self, document_id: int, user_id: int) -> bool:
```

"""

Verificación de permisos de acceso.

Factores:

- Propiedad del documento
- Permisos explícitos compartidos
- Status de administrador

"""

3.5.5 Servicio de chat y RAG (chat_service.py)

El servicio de chat implementa la funcionalidad central del sistema, integrando conversaciones con capacidades RAG avanzadas.

Gestión de Conversaciones

Creación de Chats:

```
def create_chat(self, user_id: int, name_chat: str = None) -> ChatResponse:  
    """
```

Creación de chats con nombres automáticos.

Características:

- Nombres automáticos si no se especifica
- Formato: "Chat YYYY-MM-DD HH:MM"
- Asociación automática con usuario

"""

Gestión de Historial:

```
def get_user_chats(self, user_id: int, limit: int = 100, skip: int = 0,  
                  sort_by: str = 'updated_at', order: str = 'desc') -> List[ChatResponse]:  
    """
```

Listado de chats con ordenamiento flexible.

Opciones de ordenamiento:

- created_at: Por fecha de creación
- updated_at: Por última actividad
- Orden ascendente o descendente

"""

Sistema de Mensajes Inteligente

Procesamiento de Mensajes:

```
def create_message(self, chat_id: int, message_data: MessageCreate, user_id: int) -> ChatMessage:  
    """
```

Procesamiento inteligente de mensajes con múltiples modalidades.

Modalidades implementadas:

1. Preguntas sobre listado de documentos

2. Preguntas con documentos específicos seleccionados
 3. Preguntas sin documentos (guía de selección)
 4. Chat general sin contexto documental
-

Lógica de Clasificación de Mensajes

Detección de Intención:

```
# 1. Detección de preguntas sobre documentos
document_list_queries = [
    "qué documentos tengo", "mis documentos", "listar documentos",
    "mostrar documentos", "documentos disponibles", "archivos subidos"
]
```

2. Detección de preguntas sobre contenido

```
document_keywords = [
    "documento", "archivo", "pdf", "txt", "resume", "resumir",
    "busca", "buscar", "analiza", "información", "contenido"
]
```

Integración RAG Contextual

RAG con Documentos Seleccionados:

```
# Uso del servicio de documentos para RAG
rag_result = self.document_service.get_rag_response(
    query=message_data.question,
    user_id=user_id,
    n_results=n_results,
    document_ids=document_ids
)
```

```
ai_response = rag_result["response"]
documents_used = rag_result.get("documents", [])
```

Características del RAG Integrado:

- Selección dinámica: Documentos específicos por mensaje
- Contexto preservado: Historial de conversación
- Metadatos enriquecidos: Información de documentos utilizados
- Fallback inteligente: Respuesta sin contexto cuando es apropiado

Gestión de Contexto

Construcción de Contexto:

```

# Preparar contexto para la IA
messages_context = []
for msg in previous_messages:
    if msg.question:
        messages_context.append({"role": "user", "content": msg.question})
    if msg.answer:
        messages_context.append({"role": "assistant", "content": msg.answer})

# Añadir el mensaje actual al contexto
messages_context.append({"role": "user", "content": message_data.question})

```

Características del Contexto:

- Historial completo: Mensajes previos del chat
- Formato estándar: Compatible con APIs de LLM
- Límites inteligentes: Gestión de longitud de contexto
- Persistencia: Contexto mantenido entre mensajes

Beneficios de la Arquitectura de Servicios

Modularidad y Reutilización:

- Servicios independientes y testeable por separado
- Lógica de negocio reutilizable entre diferentes interfaces
- Separación clara entre coordinación y persistencia

Escalabilidad:

- Cada servicio puede escalarse independientemente
- Posibilidad de migrar servicios a microservicios
- Cache y optimizaciones específicas por dominio

Mantenibilidad:

- Lógica de negocio centralizada y documentada
- Cambios aislados a dominios específicos
- Testing integral de reglas de negocio

Seguridad:

- Validación de permisos en capa de servicios
- Auditabilidad completa de operaciones
- Aislamiento de lógica sensible

Esta arquitectura de servicios proporciona una base robusta para el sistema de chatbot con RAG, garantizando que la lógica de negocio sea consistente, segura y fácil de mantener mientras soporta eficientemente las operaciones complejas requeridas por el sistema.

3.6 Capa de Persistencia (src/repositories)

La capa de persistencia implementa el patrón Repository para abstraer el acceso a datos, proporcionando una interfaz consistente y limpia entre la lógica de negocio y el almacenamiento de datos en Supabase.

3.6.1 Patrón Repository implementado

El sistema implementa el patrón Repository siguiendo principios de Domain-Driven Design (DDD) y Clean Architecture, proporcionando una abstracción robusta sobre el acceso a datos.

Características del Patrón Repository

Abstracción del Acceso a Datos:

```
# Interfaz consistente para todas las entidades
class BaseRepository:
    def create(self, entity) -> int
    def get(self, entity_id: int) -> Optional[Entity]
    def update(self, entity, data: dict) -> bool
    def delete(self, entity_id: int) -> bool
    def list_all(self, limit: int, offset: int) -> List[Entity]
```

Configuración Común de Repositorios:

```
class UserRepository:
    def __init__(self):
        self.table_name = "users" # Mapeo directo a tabla de Supabase
```

Cada repositorio maneja una tabla específica

Beneficios del Patrón Implementado

- Separación de responsabilidades: Lógica de acceso a datos aislada de lógica de negocio
- Testabilidad: Fácil mocking de repositorios para testing unitario
- Flexibilidad: Cambio de tecnología de persistencia sin afectar capas superiores

- Consistencia: Operaciones CRUD uniformes entre entidades
- Mantenibilidad: Centralización de queries y operaciones de base de datos

Integración con Supabase

Cliente Unificado:

```
from src.config.database import get_supabase_client

# Uso consistente del cliente con permisos de servicio
supabase = get_supabase_client(use_service_role=True)
```

Gestión de Transacciones:

- Operaciones atómicas por repositorio
- Rollback automático en caso de errores
- Logging detallado de operaciones

3.6.2 Repositorio de usuarios (user_repository.py)

El repositorio de usuarios implementa operaciones CRUD completas con funcionalidades especializadas para autenticación y gestión de cuentas.

Operaciones CRUD Principales

Creación de Usuarios:

```
def create_user(self, user_data: UserCreate) -> User:
```

```
    """
```

Crea un nuevo usuario con validaciones y seguridad.

Proceso:

1. Hash seguro de contraseña con PBKDF2
2. Generación de auth_id único (UUID)
3. Inserción en Supabase con datos validados
4. Configuración de email separado por seguridad
5. Retorno de entidad User completa

```
    """
```

Características de Creación:

- Hashing seguro: PBKDF2 con 100,000 iteraciones
- UUID único: auth_id para autenticación JWT
- Email separado: Manejo especial por políticas de Supabase
- Validación completa: Verificación de datos antes de inserción

Recuperación de Usuarios:

```
def get(self, user_id: int) -> Optional[User]:  
    """Obtiene usuario por ID con permisos de servicio"""  
  
def get_by_username(self, username: str) -> Optional[User]:  
    """Búsqueda case-insensitive por username"""  
  
def get_by_auth_id(self, auth_id: UUID) -> Optional[User]:  
    """Búsqueda por UUID para autenticación JWT"""  
  
def get_by_email(self, email: str) -> Optional[User]:  
    """Búsqueda por email para recuperación de contraseña"""
```

Funcionalidades Especializadas

Gestión de Contraseñas:

```
def update_password(self, user_id: int, new_password_hash: str) -> bool:  
    """  
        Actualización segura de contraseñas.
```

Características:

- Limpieza automática de tokens de reset
- Timestamp de actualización
- Verificación de usuario existente

"""

```
def store_reset_token(self, user_id: int, token: str, expires: datetime) -> bool:  
    """
```

Almacenamiento de tokens de recuperación.

Características:

- Tokens hasheados para seguridad
- Expiración automática (1 hora)
- Reemplazo de tokens anteriores

"""

Búsqueda y Filtrado:

```
def search_by_username(self, username_query: str, limit: int = 100) ->  
    List[User]:  
        """
```

Búsqueda parcial con ILIKE.

Características:

- Búsqueda case-insensitive
- Wildcards automáticos (%)
- Resultados limitados para rendimiento

"""

```
def list_all(self, limit: int = 100, offset: int = 0) -> List[User]:
```

"""

Listado con paginación para administradores.

Características:

- Ordenamiento por created_at descendente
- Paginación con offset/limit
- Conversión segura de tipos datetime

"""

Gestión de Sesiones y Tokens

Tokens de Refresh:

```
def update_last_login(self, user_id: int) -> bool:
```

"""Tracking de última sesión"""

```
def clear_reset_token(self, user_id: int) -> bool:
```

"""Limpieza de tokens usados"""

Avatar y Perfil:

```
def update_avatar_url(self, user_id: int, avatar_url: Optional[str]) -> bool:
```

"""Gestión de imágenes de perfil"""

```
def update_email_verified(self, user_id: int, verified: bool = True) -> bool:
```

"""Estado de verificación de email"""

3.6.3 Repositorio de documentos (document_repository.py)

El repositorio de documentos maneja el almacenamiento y recuperación de documentos con metadatos enriquecidos y sistema de permisos granular.

Operaciones CRUD Principales

Creación de Documentos:

```
def create(self, document: Document) -> int:
```

"""

Creación de documento con metadatos completos.

Campos manejados:

- Información básica (título, tipo, tamaño)
- Referencias (uploaded_by, chromadb_id)
- Estados (status, status_message)
- Timestamps automáticos

"""

```
def create_placeholder(self, document: Document) -> int:
```

"""

Creación de placeholder para procesamiento asíncrono.

Uso:

- Archivos grandes que requieren procesamiento en background
- Feedback inmediato al usuario mientras se procesa
- Estado 'pending' hasta completar procesamiento

"""

Actualización Especializada:

```
def update_with_url(self, document: Document, file_url: str = None) -> bool:
```

"""

Actualización con manejo especial de file_url.

Características:

- Actualización atómica de múltiples campos
- Manejo explícito de URLs de archivos
- Preservación de metadatos existentes

"""

```
def update_status(self, document: Document) -> bool:
```

"""

Actualización específica de estado de procesamiento.

Estados soportados:

- pending: Esperando procesamiento
- processing: En proceso de extracción/indexación
- completed: Procesamiento exitoso
- error: Error en procesamiento

"""

Sistema de Permisos y Compartición

Gestión de Acceso:

```
def check_user_access(self, document_id: int, user_id: int) -> bool:
```

```
    """
```

Verificación de permisos de acceso.

Criterios de acceso:

1. Propietario del documento (uploaded_by)
2. Documento compartido explícitamente
3. Usuario administrador

```
    """
```

```
def share_document_with_users(self, document_id: int, user_ids: List[int]) -> bool:
```

```
    """
```

Compartición con múltiples usuarios.

Proceso:

1. Validación de documento existente
2. Verificación de usuarios válidos
3. Inserción en tabla de permisos
4. Prevención de duplicados

```
    """
```

Consultas de Compartición:

```
def get_shared_documents(self, user_id: int, limit: int = 100, offset: int = 0) -> List[Document]:
```

```
    """
```

Documentos compartidos con el usuario.

JOIN con tabla acceso_documentos_usuario

```
    """
```

```
def list_document_users(self, document_id: int) -> List[int]:
```

```
    """Lista de usuarios con acceso al documento"""
```

```
def remove_user_access(self, document_id: int, user_id: int) -> bool:
```

```
    """Revocación de permisos específicos"""
```

Consultas Especializadas

Filtrado por Usuario:

```
def list_by_user(self, user_id: int, limit: int = 100, offset: int = 0,
```

```
                sort_by: str = 'created_at', order: str = 'desc') -> List[Document]:
```

```
    """
```

Documentos propios del usuario.

Características:

- Ordenamiento flexible (created_at, updated_at)
 - Paginación eficiente
 - Filtrado por propietario
-

3.6.4 Repositorio de chats (chat_repository.py)

El repositorio de chats gestiona las conversaciones del usuario con funcionalidades de ordenamiento y metadatos flexibles.

Operaciones CRUD Principales

Creación de Chats:

```
def create_chat(self, name_chat: str, user_id: int) -> Chat:  
    ....
```

Creación de chat con nombre opcional.

Características:

- name_chat puede ser NULL según BD
 - Timestamp automático de creación
 - Asociación directa con usuario
 - Conversión automática a entidad Chat
-

Listado con Ordenamiento:

```
def get_chats_by_user(self, user_id: int, limit: int = 100, skip: int = 0,  
                      sort_by: str = 'updated_at', order: str = 'desc') -> List[Chat]:  
    ....
```

Chats del usuario con ordenamiento flexible.

Opciones de ordenamiento:

- created_at: Por fecha de creación
- updated_at: Por última actividad (cuando esté disponible)
- Orden ascendente o descendente

Validación de parámetros:

- Campos de ordenamiento válidos
- Dirección de orden (asc/desc)
- Paginación con range()

.....

Consultas Especializadas

Recuperación Individual:

```
def get_chat_by_id(self, chat_id: int, user_id: Optional[int] = None) ->
Optional[Chat]:
```

.....

Obtención de chat con verificación opcional de propietario.

Flexibilidad:

- Con user_id: Verificación de permisos
- Sin user_id: Acceso directo (para operaciones internas)

.....

Actualización de Metadatos:

```
def update_chat(self, chat_id: int, user_id: int, data: Dict[str, Any]) ->
Optional[Chat]:
```

.....

Actualización con verificación de permisos.

Proceso:

1. Verificación de existencia y propiedad
2. Actualización atómica en Supabase
3. Retorno de entidad actualizada

.....

3.6.5 Repositorio de mensajes (message_repository.py)

El repositorio de mensajes implementa funcionalidades avanzadas de gestión de conversaciones con búsqueda y paginación.

Operaciones CRUD Principales

Creación de Mensajes:

```
def create_message(self, chat_id: int, question: str, answer: Optional[str] =
None) -> Message:
```

.....

Creación de mensaje con timestamp seguro.

Características especiales:

- Uso de get_safe_timestamp() para evitar problemas de particionamiento

- answer opcional (se puede actualizar después)
 - Asociación automática con chat
 - Logging de operaciones
- *****

Gestión de Timestamps Seguros:

```
# Uso de utilidad especializada para fechas
message_data = {
    "id_chat": chat_id,
    "question": question,
    "answer": answer,
    "created_at": get_safe_timestamp() # Evita errores de partición
}
```

Consultas Avanzadas

Recuperación por Chat:

```
def get_messages_by_chat(self, chat_id: int, limit: int = 100, skip: int = 0) ->
List[Message]:
    """
```

Mensajes de conversación con ordenamiento temporal.

Características:

- Ordenamiento por created_at y id (doble orden)
 - Paginación con range()
 - Mapeo seguro a entidades Message
 - Manejo de campos opcionales
- *****

Búsqueda Textual:

```
def search_messages(self, query: str, chat_id: Optional[int] = None,
                    limit: int = 50) -> List[Message]:
    """
```

Búsqueda en contenido de mensajes.

Implementación:

1. Búsqueda separada en questions y answers (ILIKE)
2. Combinación de resultados únicos
3. Eliminación de duplicados por ID
4. Ordenamiento por relevancia temporal
5. Limitación de resultados finales

Limitaciones Supabase:

- No tiene operador OR nativo en queries
 - Requiere consultas separadas y combinación manual
-

Operaciones de Gestión

Eliminación Masiva:

```
def delete_messages_by_chat(self, chat_id: int) -> int:  
    """
```

Eliminación de todos los mensajes de un chat.

Proceso:

1. Conteo previo para logging
 2. Eliminación masiva con filtro por chat
 3. Retorno de cantidad eliminada
-

Estadísticas:

```
def get_message_count_by_chat(self, chat_id: int) -> int:  
    """
```

Conteo eficiente de mensajes.

Métodos:

- count='exact' cuando disponible
 - Fallback a len(response.data)
 - Manejo de respuestas vacías
-

Características Transversales de los Repositorios

Manejo de Errores Consistente:

```
try:  
    # Operación de base de datos  
    response = supabase.table(self.table_name).operation().execute()  
    # Procesamiento de respuesta  
except Exception as e:  
    logger.error(f"Error en operación: {str(e)}")  
    raise # Re-propagar para manejo en capa superior
```

Logging Estructurado:

- Información de operaciones exitosas

- Warnings para situaciones anómalas
- Errores detallados con contexto
- Métricas de rendimiento cuando es relevante

Conversión de Tipos Segura:

```
# Manejo seguro de datetime desde Supabase
if user_data.get('created_at'):
    try:
        user.created_at = datetime.fromisoformat(
            user_data['created_at'].replace('Z', '+00:00')
        )
    except (ValueError, TypeError):
        user.created_at = datetime.now()
```

Uso de Permisos de Servicio:

```
# Uso consistente de service role para operaciones CRUD
supabase = get_supabase_client(use_service_role=True)
```

Esta arquitectura de repositorios proporciona una abstracción robusta y consistente sobre Supabase, facilitando el mantenimiento, testing y evolución del sistema mientras mantiene la integridad de los datos y la seguridad de las operaciones.

3.7 Configuración (src/config)

La capa de configuración centraliza la gestión de parámetros del sistema, variables de entorno y conexiones a servicios externos, proporcionando una interfaz unificada para la configuración de toda la aplicación.

3.7.1 Configuraciones centralizadas (settings.py)

El módulo de configuraciones implementa un sistema robusto de gestión de parámetros usando Pydantic Settings, garantizando validación automática y tipado fuerte.

Arquitectura de Configuración

Clase Principal de Settings:

```
from pydantic_settings import BaseSettings
from pydantic import Field, ConfigDict

class Settings(BaseSettings):
```

:::::

Configuraciones principales de la aplicación - Versión simplificada

:::::

Información general

APP_NAME: str = "ChatBotMadSL"

APP_VERSION: str = "1.0.0"

APP_ENVIRONMENT: str = "development"

DEBUG: bool = True

Categorías de Configuración

Configuración del Servidor:

Servidor

HOST: str = "127.0.0.1"

PORT: int = 2690

Base de Datos y Servicios Externos:

Base de datos - usando Field para mayor flexibilidad

SUPABASE_URL: str = Field(default="", env="SUPABASE_URL")

SUPABASE_KEY: str = Field(default="", env="SUPABASE_KEY")

SUPABASE_SERVICE_KEY: str = Field(default="", env="SUPABASE_SERVICE_KEY")

DATABASE_URL: Optional[str] = Field(default=None, env="DATABASE_URL")

ChromaDB

CHROMA_HOST: str = "localhost"

CHROMA_PORT: int = 8000

CHROMA_TELEMETRY_ENABLED: bool = False

CHROMA_SERVER_TIMEOUT: int = 300

Configuración de IA:

Gemini

GEMINI_API_KEY: str = Field(default="", env="GEMINI_API_KEY")

GEMINI_MODEL: str = "gemini-2.0-flash"

GEMINI_EMBEDDING_MODEL: str = "text-embedding-004"

Configuración de Seguridad

JWT y Autenticación:

Seguridad

SECRET_KEY: str = Field(default="clave-secreta-temporal", env="SECRET_KEY")

JWT_ALGORITHM: str = "HS256"

ACCESS_TOKEN_EXPIRE_MINUTES: int = 57600

```

@property
def JWT_SECRET_KEY(self) -> str:
    return self.SECRET_KEY

@property
def JWT_EXPIRES_MINUTES(self) -> int:
    return self.ACCESS_TOKEN_EXPIRE_MINUTES

```

Características de Seguridad:

- Claves secretas: Generación recomendada con secrets.token_hex(32)
- Expiración configurable: Tokens de larga duración para desarrollo
- Algoritmo estándar: HS256 para JWT
- Properties dinámicas: Acceso consistente a configuraciones derivadas

Configuración RAG Optimizada

Parámetros de Procesamiento:

```

# Configuración RAG - Optimizado para documentos de TFG
RAG_CHUNK_SIZE: int = 1500    # Aumentado para manejar párrafos más largos
RAG_CHUNK_OVERLAP: int = 150   # Aumentado para mejor contexto entre
chunks
RAG_MAX_TOKENS: int = 2000    # Aumentado para respuestas más completas
RAG_NUM_RESULTS: int = 5       # Número estándar de resultados

```

Límites de Documentos:

```

# Documentos
MAX_DOCUMENT_SIZE: int = 30    # MB - Tamaño máximo por documento
DOCUMENT_PROCESSING_TIMEOUT: int = 180 # Segundos - Timeout de
procesamiento

```

Configuración de Email SMTP

Servicios de Notificación:

```

# Email SMTP Configuration
SMTP_HOST: Optional[str] = Field(default="smtp.gmail.com",
env="SMTP_HOST")
SMTP_PORT: int = Field(default=587, env="SMTP_PORT")
SMTP_USER: Optional[str] = Field(default=None, env="SMTP_USER")
SMTP_PASSWORD: Optional[str] = Field(default=None,
env="SMTP_PASSWORD")

```

```
FROM_EMAIL: Optional[str] = Field(default=None, env="FROM_EMAIL")
```

```
# Frontend URL para links en emails  
FRONTEND_URL: str = Field(default="http://localhost:3000",  
env="FRONTEND_URL")
```

Configuración del Modelo Pydantic

ConfigDict Optimizado:

```
model_config = ConfigDict(  
    env_file=".env",           # Archivo de variables de entorno  
    env_file_encoding="utf-8",   # Encoding para caracteres especiales  
    case_sensitive=True,        # Variables sensibles a mayúsculas  
    extra="ignore",            # Ignorar campos extras  
    env_nested_delimiter="__"  # Delimitador para variables anidadas  
)
```

Características del ConfigDict:

- Carga automática: Lee archivo .env automáticamente
- Encoding seguro: UTF-8 para soporte internacional
- Case sensitive: Evita confusiones con nombres de variables
- Extra ignore: Flexibilidad para variables no definidas
- Nesting support: Variables complejas con delimitadores

Patrón Singleton para Settings

Instancia Global:

```
# Cargar variables de entorno manualmente si es necesario  
try:
```

```
    from dotenv import load_dotenv  
    load_dotenv(override=True)  
except ImportError:  
    pass
```

```
# Instancia global  
settings = Settings()
```

```
def get_settings() -> Settings:  
    return settings
```

Beneficios del Singleton:

- Consistencia: Una sola configuración en toda la aplicación
- Rendimiento: Carga una sola vez al inicio
- Flexibilidad: Fácil acceso desde cualquier módulo
- Testing: Posibilidad de override para tests

3.7.2 Configuración de base de datos (database.py)

El módulo de base de datos implementa un patrón Singleton robusto para gestionar conexiones a Supabase con diferentes niveles de permisos.

Arquitectura de Conexión

Patrón Singleton Implementado:

```
class SupabaseConnector:
    """
    Implementación del patrón Singleton para la conexión a Supabase.
    Garantiza una única instancia de la conexión en toda la aplicación.
    """

    _instance: Optional['SupabaseConnector'] = None
    _client = None
    _service_client = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(SupabaseConnector, cls).__new__(cls)
            cls._instance._client = None
            cls._instance._service_client = None
        return cls._instance
```

Gestión de Clientes Múltiples

Cliente Normal vs Service Role:

```
def get_client(self, use_service_role=False):
    """
    Obtiene el cliente de Supabase, inicializándolo si es necesario.
    """

    Args:
        use_service_role: Si es True, usa la clave de servicio para omitir RLS
    """

    if use_service_role:
        if self._service_client is None:
            # Inicializar cliente con clave de servicio
            supabase_url = os.getenv("SUPABASE_URL")
```

```

        service_key = os.getenv("SUPABASE_SERVICE_KEY")

        self._service_client = create_client(supabase_url, service_key)
        logger.info("Cliente de Supabase con permisos de servicio creado
exitosamente")

    return self._service_client

# Cliente normal con RLS habilitado
if self._client is None:
    supabase_url = os.getenv("SUPABASE_URL")
    supabase_key = os.getenv("SUPABASE_KEY")

    self._client = create_client(supabase_url, supabase_key)
    logger.info("Cliente de Supabase creado exitosamente")

return self._client

```

Validación de Variables de Entorno

Verificación de Credenciales:

```

# Validar variables de entorno requeridas
required_env_vars = ["SUPABASE_URL", "SUPABASE_KEY",
"SUPABASE_SERVICE_KEY"]
missing_vars = [var for var in required_env_vars if not os.getenv(var)]
if missing_vars:
    logger.error(f"Faltan variables de entorno requeridas: {',
'.join(missing_vars)}")
    raise ValueError(f"Faltan variables de entorno requeridas: {',
'.join(missing_vars)}")

```

Características de Validación:

- Verificación temprana: Al importar el módulo
- Error descriptivo: Lista específica de variables faltantes
- Fallo rápido: No permite ejecución sin configuración completa

Testing de Conexión

Verificación de Conectividad:

```
def test_connection(self) -> bool:
    """

```

Prueba la conexión a Supabase realizando una operación simple.

```

"""
try:
    client = self.get_client()
    logger.info("Probando conexión a Supabase...")

    # Intenta una operación simple: verificar si existe la tabla users
    response = client.from_('users').select('count').limit(1).execute()

    logger.info(f"Conexión exitosa a Supabase. Respuesta: {response}")
    return True
except Exception as e:
    logger.error(f"Error al conectar con Supabase: {str(e)}")
    return False

```

Funciones de Conveniencia

Helpers para Acceso Rápido:

```

def get_supabase_connector() -> SupabaseConnector:
    return SupabaseConnector()

def get_supabase_client(use_service_role=False):
    return get_supabase_connector().get_client(use_service_role)

# Para compatibilidad con código que espera SQLAlchemy
def get_db():
    """
    Stub para compatibilidad con código que espera SQLAlchemy.
    En lugar de una sesión de base de datos, devuelve el cliente de Supabase.
    """
    client = get_supabase_client()
    yield client

```

Compatibilidad y Migración

Stubs para SQLAlchemy:

```

class Session:
    """
    Stub para compatibilidad
    """
    pass

# Variables de compatibilidad
engine = None
SessionLocal = None

```

3.7.3 Variables de entorno y parámetros

El sistema implementa una gestión robusta de variables de entorno con múltiples niveles de configuración y validación.

Jerarquía de Configuración

Orden de Precedencia:

1. Variables de entorno del sistema (mayor prioridad)
2. Archivo .env local
3. Valores por defecto en settings.py (menor prioridad)

Archivo .env Estructurado

Configuración General:

```
# Configuración general
HOST=127.0.0.1
PORT=2690
```

```
# Docker
DOCKER_ENV=true
```

Base de Datos y Servicios:

```
# Supabase
SUPABASE_URL=https://lelsehigjjknxjwvxupv.supabase.co
SUPABASE_KEY=eyJhbGciOiJIUzI1NilsInR5cCI6IkpxVCJ9...
SUPABASE_SERVICE_KEY=eyJhbGciOiJIUzI1NilsInR5cCI6IkpxVCJ9...
```

```
# Database URL para SQLAlchemy (Transaction pooler)
DATABASE_URL=postgresql://postgres.lelsehigjjknxjwvxupv:password@aws-0-
us-west-1.pooler.supabase.com:6543/postgres
```

ChromaDB Optimizado:

```
CHROMA_HOST=localhost
CHROMA_PORT=8050
CHROMA_TELEMETRY_ENABLED=false # Desactiva telemetría para mejor
rendimiento
CHROMA_SERVER_TIMEOUT=300      # Aumenta el timeout a 5 minutos para
documentos grandes
Configuración de Seguridad
```

JWT y Secretos:

```
# Seguridad para generar el secret_key es python -c "import secrets;
print(secrets.token_hex(32))"
SECRET_KEY=2ee4b09429eae9916ce160621db540db25d5b10677454c3d08801
8d8d3e4dccd
JWT_SECRET_KEY=2ee4b09429eae9916ce160621db540db25d5b10677454c3d0
88018d8d3e4dccd
JWT_ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=57600
```

Generación de Claves Seguras:

```
# Comando para generar SECRET_KEY segura
python -c "import secrets; print(secrets.token_hex(32))"
Configuración RAG Optimizada
```

Parámetros para TFG:

```
# Configuración RAG - Optimizado para documentos de TFG
RAG_CHUNK_SIZE=1500      # Aumentado para manejar párrafos más largos
RAG_CHUNK_OVERLAP=150    # Aumentado para mejor contexto entre chunks
RAG_MAX_TOKENS=2000      # Aumentado para respuestas más completas
RAG_NUM_RESULTS=5        # Número estándar de resultados

# Nuevas configuraciones para documentos
MAX_DOCUMENT_SIZE=100    # Tamaño máximo de documento en MB
DOCUMENT_PROCESSING_TIMEOUT=180 # Tiempo máximo de procesamiento
en segundos
```

Servicios Externos

Gemini AI:

```
# Gemini
GEMINI_API_KEY=AlzaSyBU61kK_MwSDV-RsnrIEt6JEWRuisyVi1E
GEMINI_MODEL=gemini-2.0-flash
GEMINI_EMBEDDING_MODEL=text-embedding-004
```

Email SMTP:

```
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=heily1857@gmail.com
SMTP_PASSWORD=avnu mfjz dkwl fuxy
FROM_EMAIL=heily1857@gmail.com
```

Gestión por Entornos

Desarrollo Local:

```
FRONTEND_URL=http://localhost:53793  
DEBUG=true  
APP_ENVIRONMENT=development
```

Producción:

- Variables de entorno del sistema en lugar de archivo .env
- Configuraciones optimizadas para rendimiento
- Logging y monitoreo configurado para producción
- Rate limiting y timeouts ajustados

Validación y Seguridad

Carga Segura de Variables:

```
# Implementación con fallback para dotenv  
try:  
    from dotenv import load_dotenv  
    load_dotenv	override=True)  
except ImportError:  
    # Implementación manual para sistemas sin python-dotenv  
    pass
```

Verificación de Importaciones:

```
# Verificación detallada de supabase  
supabase_import_error = None  
try:  
    if importlib.util.find_spec("supabase") is None:  
        raise ImportError("El módulo 'supabase' no está instalado")  
  
    from supabase import create_client, Client  
    logger.info("supabase importado correctamente")  
except ImportError as e:  
    supabase_import_error = str(e)  
    logger.error(f"ERROR DETALLADO DE IMPORTACIÓN:  
{supabase_import_error}")  
Mejores Prácticas Implementadas
```

Separación de Secretos:

- Archivo .env en .gitignore para proteger credenciales
- Variables separadas para diferentes niveles de acceso
- Rotación periódica de claves API recomendada

Configuración Flexible:

- Valores por defecto sensatos para desarrollo
- Override fácil para diferentes entornos
- Documentación inline de propósito de variables

Validación Automática:

- Pydantic Fields con validación de tipos
- Verificación de variables críticas al inicio
- Mensajes de error descriptivos para debugging

Esta arquitectura de configuración proporciona una base sólida y flexible para gestionar todos los aspectos de configuración del sistema, desde el desarrollo local hasta el despliegue en producción, manteniendo la seguridad y facilidad de mantenimiento.

3.8 Utilidades (src/utils)

La capa de utilidades proporciona componentes especializados y reutilizables que soportan las funcionalidades core del sistema, incluyendo conectores a servicios externos, utilidades de seguridad y helpers especializados.

3.8.1 Connector de IA (ai_connector.py)

El conector de IA implementa una abstracción robusta sobre Google Gemini API, proporcionando funcionalidades de generación de texto, embeddings y chat completion con patrón Singleton.

3.8.1.1 Integración con Gemini API

Configuración y Autenticación:

```
class GeminiConnector:
```

```
    """
```

Implementación del patrón Singleton para las conexiones a Gemini.
Maneja la autenticación y las llamadas a la API.

```
    """
```

```
def get_client(self):
```

```
    """
```

```

Configura y obtiene el cliente de Gemini, inicializándolo si es necesario.
"""

if self._client is None:
    api_key = os.getenv("GEMINI_API_KEY")

    if not api_key:
        raise ValueError(
            "La variable de entorno GEMINI_API_KEY es requerida."
            "Asegúrate de configurarla en el archivo .env"
        )

    logger.info("Inicializando cliente de Gemini")
    genai.configure(api_key=api_key)
    self._client = genai

return self._client

```

Características de la Integración:

- Autenticación segura: API key desde variables de entorno
- Inicialización lazy: Cliente se crea solo cuando es necesario
- Configuración global: genai.configure() para toda la aplicación
- Validación temprana: Verificación de API key antes de uso

3.8.1.2 Patrón Singleton aplicado

Implementación del Singleton:

```

class GeminiConnector:
    # Atributo de clase que almacenará la única instancia (Singleton)
    _instance: Optional['GeminiConnector'] = None
    # Atributos para almacenar la configuración
    _client = None
    _embedding_model = None

    def __new__(cls):
        # Si no existe una instancia previa de la clase, se crea una nueva
        if cls._instance is None:
            cls._instance = super(GeminiConnector, cls).__new__(cls)
        # Devuelve la instancia única (Singleton)
        return cls._instance

    def __init__(self):
        # Inicializa los clientes

```

```
    self._client = None
    self._embedding_model = None
```

Beneficios del Singleton:

- Consistencia: Una sola configuración de API en toda la aplicación
- Eficiencia: Evita múltiples inicializaciones de cliente
- Estado compartido: Configuración persistente entre llamadas
- Gestión centralizada: Control único de conexiones a Gemini

3.8.1.3 Generación de embeddings

Implementación con SentenceTransformer:

```
def get_embedding_model(self):
    """
    Obtiene el modelo de embedding, inicializándolo si es necesario.
    Gemini no tiene embeddings completos, así que usamos
    SentenceTransformer.
    """
    if self._embedding_model is None:
        try:
            from sentence_transformers import SentenceTransformer
            logger.info("Inicializando modelo de embeddings
(SentenceTransformer)")
            self._embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
        except ImportError as e:
            logger.error(f"Error importando SentenceTransformer: {e}")
            raise ImportError("SentenceTransformer no está instalado. Ejecuta: pip
install sentence-transformers")
    return self._embedding_model

def create_embeddings(self, texts: List[str], model: str = None) ->
List[List[float]]:
    """
    Genera embeddings vectoriales para una lista de textos.
```

Args:

texts: Lista de textos para generar embeddings
model: Parámetro mantenido por compatibilidad (ignorado)

Returns:

Lista de vectores (embeddings)

"""

```

try:
    embedding_model = self.get_embedding_model()
    embeddings = embedding_model.encode(texts, convert_to_numpy=True)
    return embeddings.tolist()
except Exception as e:
    logger.error(f"Error al generar embeddings: {str(e)}")
    raise

```

Características de los Embeddings:

- Modelo optimizado: all-MiniLM-L6-v2 para balance rendimiento/calidad
- Batch processing: Procesamiento eficiente de múltiples textos
- Formato compatible: Conversión a lista de listas para ChromaDB
- Manejo de errores: Logging detallado y re-propagación

3.8.1.4 Chat completion y RAG

Generación de Chat Completion:

```

def generate_chat_completion(
    self,
    messages: List[Dict[str, str]],
    model: str = None,
    temperature: float = 0.7,
    max_tokens: int = 1000
) -> str:
    """

```

Genera una respuesta de chat usando Gemini.

Args:

messages: Lista de mensajes en formato [{"role": "...", "content": "..."}]
 model: Parámetro mantenido por compatibilidad (se ignora y usa config global)
 temperature: Temperatura para la generación (0-1)
 max_tokens: Máximo número de tokens a generar

Returns:

Texto generado por el modelo

"""

Procesamiento de Conversaciones:

```

# Separar el último mensaje del historial
if not messages:

```

```

raise ValueError("No hay mensajes para procesar")

last_message = messages[-1]
history_messages = messages[:-1] if len(messages) > 1 else []

# Convertir historial a formato Gemini
gemini_history = []
for msg in history_messages:
    # Gemini usa "user" y "model" como roles
    if msg["role"].lower() == "user":
        role = "user"
    elif msg["role"].lower() in ["assistant", "system"]:
        role = "model"
    else:
        role = "user" # Por defecto

    content = msg["content"]
    if content and content.strip():
        gemini_history.append({
            "role": role,
            "parts": [{"text": content}]
        })

```

Generación RAG Especializada:

```

def generate_rag_response(
    self,
    query: str,
    context: List[str],
    system_template: str = "Eres un asistente útil que responde preguntas
basándose en el siguiente contexto:\n\n{context}",
    model: str = None,
    temperature: float = 0.7,
    max_tokens: int = 1000
) -> str:
    """
    Genera una respuesta basada en RAG (Retrieval-Augmented Generation).
    """

    try:
        # Validar que hay contexto
        if not context or not any(c.strip() for c in context):
            raise ValueError("El contexto está vacío")

        # Une los fragmentos de contexto en un solo string, numerándolos

```

```

    context_text = "\n\n".join([f"Fragmento {i+1}: {text}" for i, text in
enumerate(context) if text and text.strip()])

# Crea el mensaje de sistema usando la plantilla y el contexto
system_message = system_template.format(context=context_text)

# Para Gemini, combinar el sistema y la pregunta del usuario en un solo
prompt
combined_prompt = f"{system_message}\n\nPregunta:
{query}\n\nRespuesta:"

# Generar respuesta
client = self.get_client()
model_name = os.getenv("GEMINI_MODEL", "gemini-2.0-flash")
gemini_model = client.GenerativeModel(model_name)

generation_config = genai.types.GenerationConfig(
    temperature=temperature,
    max_output_tokens=max_tokens,
)

response = gemini_model.generate_content(
    combined_prompt,
    generation_config=generation_config
)

return response.text

```

Compatibilidad OpenAI:

```

# Mantener compatibilidad con el código existente
OpenAIConnector = GeminiConnector
get_openai_connector = get_gemini_connector

```

3.8.2 Connector de ChromaDB (chromadb_connector.py)

El conector de ChromaDB implementa una interfaz completa para operaciones vectoriales con patrón Singleton y lazy initialization.

3.8.2.1 Base de datos vectorial

Arquitectura de Conexión:

```
class ChromaDBConnector:
```

```

    _instance = None
    _client = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(ChromaDBConnector, cls).__new__(cls)
            cls._instance._client = None
            cls._instance._initialized_collections = set()
        return cls._instance

```

Configuración de Cliente HTTP:

```

def get_client(self):
    """Inicializa el cliente si aún no existe (lazy initialization)"""
    if self._client is None:
        try:
            from src.config.settings import get_settings
            settings = get_settings()

            host = settings.CHROMA_HOST
            port = settings.CHROMA_PORT

            logger.info(f"Conectando a ChromaDB en {host}:{port}")

            # Usar el cliente HTTP para conectarse al contenedor Docker
            self._client = chromadb.HttpClient(
                host=host,
                port=port
            )

            # Verificar con heartbeat
            self._client.heartbeat()
            logger.info("✅ Conexión exitosa a ChromaDB!")
            self._ensure_collection_exists("documents")
        except Exception as e:
            logger.error(f"Error al inicializar cliente ChromaDB: {str(e)}")
            self._client = None

    return self._client

```

3.8.2.2 Operaciones CRUD vectoriales

Creación de Documentos:

```
def add_documents(self,
```

```

        collection_name: str,
        document_ids: List[str],
        chunks: List[str],
        metadatas: Optional[List[Dict[str, Any]]] = None,
        timeout: int = 35) -> bool:
    """Añade documentos a ChromaDB con timeout."""
    start_time = time.time()
    try:
        client = self.get_client()
        self._ensure_collection_exists(collection_name)

        collection = client.get_collection(collection_name)
        logger.info(f"Preparando para añadir {len(chunks)} chunks a ChromaDB")

        # Verificar metadatos
        if not metadatas:
            metadatas = [{"default": "true"} for _ in range(len(chunks))]

        # IMPORTANTE: Procesar con timeout para evitar bloqueos
        import concurrent.futures
        with concurrent.futures.ThreadPoolExecutor() as executor:
            future = executor.submit(
                collection.add,
                ids=document_ids,
                documents=chunks,
                metadatas=metadatas
            )

            try:
                result = future.result(timeout=35)
                chroma_time = time.time() - start_time
                logger.info(f"⌚ ChromaDB: Añadidos {len(chunks)} chunks en {chroma_time:.3f} segundos")
                return True
            except concurrent.futures.TimeoutError:
                logger.error(f"⚠ TIMEOUT de {timeout}s al añadir documentos a ChromaDB")
                return False
            except Exception as e:
                error_time = time.time() - start_time
                logger.error(f"✖ Error en ChromaDB después de {error_time:.3f} segundos: {str(e)}")
                return False

```

Actualización de Documentos:

```
def update_document(self,
                    collection_name: str,
                    document_id: str,
                    chunk: Optional[str] = None,
                    metadata: Optional[Dict[str, Any]] = None):
    """Actualiza un documento existente"""
    try:
        client = self.get_client()
        collection = client.get_collection(name=collection_name)
        collection.update(
            ids=[document_id],
            documents=[chunk] if chunk else None,
            metadatas=[metadata] if metadata else None
        )
        logger.info(f"Documento {document_id} actualizado en {collection_name}")
        return True
    except Exception as e:
        logger.error(f"Error al actualizar documento: {str(e)}")
        return False
```

Eliminación de Documentos:

```
def delete_documents(self, collection_name: str, document_ids: List[str]):
    """Elimina documentos por ID"""
    client = self.get_client()
    try:
        collection = client.get_collection(name=collection_name)
        collection.delete(ids=document_ids)
        logger.info(f"Eliminados {len(document_ids)} documentos de {collection_name}")
        return True
    except Exception as e:
        logger.error(f"Error al eliminar documentos: {str(e)}")
        return False
```

3.8.2.3 Búsqueda semántica

Búsqueda Vectorial Avanzada:

```
def search_documents(self,
                     collection_name: str,
                     query_text: str,
                     n_results: int = 5,
```

```

        where: Optional[Dict[str, Any]] = None):
try:
    client = self.get_client()
    self._ensure_collection_exists(collection_name)
    collection = client.get_collection(collection_name)

    # Si where es un diccionario vacío, establecerlo a None
    if where is not None and not where:
        where = None

    # Formato correcto para where: debe incluir un operador como $eq, $gt,
etc.
    # Ejemplo: {"document_id": {"$eq": "33"}}

    results = collection.query(
        query_texts=[query_text],
        n_results=n_results,
        where=where
    )
    return results
except Exception as e:
    logger.error(f"Error al buscar documentos: {str(e)}")
    return None

```

Características de Búsqueda:

- Similarity search: Búsqueda por similitud semántica automática
- Filtrado avanzado: Cláusulas WHERE con operadores MongoDB-style
- Resultados limitados: Control de número de resultados
- Metadatos incluidos: Información contextual en resultados

3.8.2.4 Singleton y lazy initialization

Gestión de Colecciones:

```

def _ensure_collection_exists(self, collection_name: str):
    """Verifica que una colección exista y la crea si no"""
    if hasattr(self, '_initialized_collections') and collection_name in
    self._initialized_collections:
        return # Ya fue verificada/creada

try:
    # Intentar obtener la colección
    self.get_client().get_collection(collection_name)

```

```

    logger.info(f"Colección '{collection_name}' encontrada en ChromaDB")
except Exception as e:
    if "does not exists" in str(e) or "not found" in str(e).lower():
        logger.info(f"Creando colección '{collection_name}' en ChromaDB...")
        self.get_client().create_collection(collection_name)
        logger.info(f"✅ Colección '{collection_name}' creada exitosamente")
    else:
        logger.error(f"Error al verificar colección '{collection_name}': {str(e)}")
        return

# Marcar como inicializada
if not hasattr(self, '_initialized_collections'):
    self._initialized_collections = set()
self._initialized_collections.add(collection_name)

```

Testing de Conexión:

```

def test_connection(self) -> bool:
    """Prueba la conexión/funcionalidad de ChromaDB"""
    try:
        client = self.get_client()
        client.list_collections() # Operación ligera para probar
        logger.info("Conexión a ChromaDB exitosa")
        return True
    except Exception as e:
        logger.error(f"Error con ChromaDB: {str(e)}")
        return False

```

3.8.3 Utilidades de password (password_utils.py)

El módulo de utilidades de contraseñas implementa hashing seguro con múltiples algoritmos y compatibilidad hacia atrás.

Hashing con PBKDF2

Función Principal de Hash:

```

def hash_password(password: str) -> str:
    """Genera un hash PBKDF2 para la contraseña."""
    # Generar un salt aleatorio
    salt = os.urandom(16)

    # Generar el hash
    key = hashlib.pbkdf2_hmac(

```

```

'sha256',
password.encode('utf-8'),
salt,
100000 # Número de iteraciones
)

# Combinar salt y hash y convertir a formato legible
storage = salt + key
# Añadir prefijo para identificar el formato
return "$pbkdf2$" + base64.b64encode(storage).decode('utf-8')

```

Características de Seguridad:

- PBKDF2-SHA256: Algoritmo resistente a ataques de fuerza bruta
- 100,000 iteraciones: Número alto para máxima seguridad
- Salt aleatorio: 16 bytes únicos por contraseña
- Formato identifiable: Prefijo \$pbkdf2\$ para reconocimiento

Verificación Multi-algoritmo

Función de Verificación:

```

def verify_password(plain_password: str, stored_password: str) -> bool:
    """Verifica si la contraseña coincide con el hash almacenado."""

    # Comprobamos si es nuestro formato PBKDF2
    if stored_password.startswith("$pbkdf2$"):
        # Quitar el prefijo
        encoded = stored_password[8:]
        try:
            # Decodificar para obtener salt y key original
            storage = base64.b64decode(encoded.encode('utf-8'))
            salt = storage[:16]
            original_key = storage[16:]

            # Generar clave con mismos parámetros
            key = hashlib.pbkdf2_hmac(
                'sha256',
                plain_password.encode('utf-8'),
                salt,
                100000
            )

            # Comparar claves

```

```

        return key == original_key
    except Exception as e:
        logging.error(f"Error al verificar contraseña PBKDF2: {str(e)}")
        return False

# Si es bcrypt (para compatibilidad con hashes existentes)
elif stored_password.startswith('$2b$'):
    try:
        import bcrypt
        return bcrypt.checkpw(
            plain_password.encode('utf-8'),
            stored_password.encode('utf-8')
        )
    except Exception as e:
        logging.error(f"Error al verificar contraseña bcrypt: {str(e)}")
        return False

```

Compatibilidad hacia Atrás:

- Detección automática: Identificación por prefijo del algoritmo
- Soporte bcrypt: Compatibilidad con hashes existentes
- Migración gradual: Permite coexistencia de algoritmos
- Logging de errores: Información detallada para debugging

3.8.4 Utilidades de administración (admin_utils.py)

El módulo de utilidades administrativas centraliza la lógica de verificación de permisos y gestión de roles especiales.

Verificación de Privilegios

Función Principal:

```

def has_admin_privileges(user: User) -> bool:
    """
    Verifica si un usuario tiene privilegios de administrador.
    Considera tanto el flag is_admin como si es el usuario Ivan.
    """

```

Args:

 user: Usuario a verificar

Returns:

 bool: True si tiene privilegios de administrador

"""

```
if user is None:  
    return False  
  
# Ivan siempre debe tener privilegios de admin  
if user.username and user.username.upper() == "IVAN":  
    return True  
  
# Otros usuarios solo si tienen el flag is_admin  
return user.is_admin
```

Gestión de Estado Admin

Función de Aseguración:

```
def ensure_admin_status(user: User) -> User:  
    """  
    Asegura que Ivan tenga siempre estatus de administrador.  
    Modifica el objeto user si es necesario.  
    """
```

Args:

 user: Usuario a verificar/modificar

Returns:

 User: Usuario con estatus actualizado si corresponde

"""

```
if user and user.username and user.username.upper() == "IVAN":  
    user.is_admin = True
```

```
return user
```

Alias para Compatibilidad

```
# Alias para compatibilidad con código existente  
is_administrator = has_admin_privileges
```

Características de las Utilidades Admin:

- Usuario especial: Ivan como administrador principal inmutable
- Verificación robusta: Múltiples criterios de validación
- Case insensitive: Comparación de username sin importar mayúsculas
- Modificación automática: Corrección de estado si es necesario
- Compatibilidad: Alias para código legacy

Utilidades de Fecha (date_utils.py)

Gestión de Timestamps Seguros:

```

def get_safe_timestamp() -> str:
    """
    Devuelve una marca de tiempo segura para insertar en la base de datos.

    NOTA: La tabla messages está particionada por fecha y solo tiene particiones
    hasta mayo 2025. Esta función devuelve una fecha dentro del rango seguro.
    """

    now = datetime.now(timezone.utc)

    # Si estamos en junio 2025 o después, usar el último día de mayo 2025
    if now.year >= 2025 and now.month >= 6:
        safe_date = datetime(2025, 5, 31, now.hour, now.minute, now.second,
                             now.microsecond, tzinfo=timezone.utc)
        logger.warning(f"Fecha actual {now} está fuera del rango de particiones.
Usando {safe_date}")
    else:
        safe_date = now

    return safe_date.isoformat()

```

Este conjunto de utilidades proporciona una base sólida de componentes reutilizables que soportan eficientemente las operaciones del sistema, desde la gestión de IA y bases de datos vectoriales hasta la seguridad y administración de usuarios.

3.9 Gestión de Excepciones (src/core)

El sistema implementa una arquitectura robusta de gestión de excepciones que proporciona manejo centralizado de errores, trazabilidad completa y mensajes de error coherentes en toda la aplicación.

3.9.1 Excepciones personalizadas (exceptions.py)

El módulo de excepciones define una jerarquía estructurada de excepciones personalizadas que mapean directamente a códigos de estado HTTP y proporcionan información contextual rica para el debugging y la experiencia del usuario.

Jerarquía de Excepciones

Excepción Base:

```

class AppException(Exception):
    """
    Excepción base para todas las excepciones de la aplicación.
    """

```

```
def __init__(self, message: str = "Error de aplicación"):
    self.message = message
    super().__init__(self.message)
```

Características de la Clase Base:

- Mensaje personalizado: Soporte para mensajes descriptivos específicos
- Herencia estándar: Compatible con sistema de excepciones de Python
- Atributo message: Acceso directo al mensaje de error
- Valor por defecto: Mensaje genérico como fallback

Excepciones Específicas por Dominio

Recursos No Encontrados (HTTP 404):

```
class NotFoundException(AppException):
    """Excepción para recursos no encontrados (HTTP 404)."""
    def __init__(self, message: str = "Recurso no encontrado"):
        super().__init__(message)
```

Casos de uso:

- Usuario no encontrado por ID o username
- Documento inexistente en base de datos
- Chat o mensaje no disponible
- Archivo no localizado en storage

Acceso No Autorizado (HTTP 401/403):

```
class UnauthorizedException(AppException):
    """Excepción para accesos no autorizados (HTTP 401/403)."""
    def __init__(self, message: str = "No autorizado para realizar esta acción"):
        super().__init__(message)
```

Casos de uso:

- Token JWT inválido o expirado
- Permisos insuficientes para operación
- Acceso a recursos de otros usuarios
- Operaciones administrativas sin privilegios

Errores de Validación (HTTP 400):

```
class ValidationException(AppException):
    """Excepción para errores de validación (HTTP 400)."""
```

```
def __init__(self, message: str = "Error de validación en los datos"):
    super().__init__(message)
```

Casos de uso:

- Datos de entrada inválidos o malformados
- Violación de reglas de negocio
- Parámetros requeridos faltantes
- Formatos de archivo no soportados

Conflictos de Datos (HTTP 409):

```
class ConflictException(AppException):
    """Excepción para conflictos de datos (HTTP 409)."""
    def __init__(self, message: str = "Conflicto con datos existentes"):
        super().__init__(message)
```

Casos de uso:

- Username o email duplicado en registro
- Documento ya compartido con usuario
- Operación que viola restricciones únicas
- Estados inconsistentes en actualizaciones

Errores de Base de Datos (HTTP 500):

```
class DatabaseException(AppException):
    """Excepción para errores de base de datos (HTTP 500)."""
    def __init__(self, message: str = "Error en la base de datos"):
        super().__init__(message)
```

Casos de uso:

- Fallos de conexión a Supabase
- Errores de transacción o rollback
- Violaciones de integridad referencial
- Problemas de conectividad de red

Errores de Servicios Externos (HTTP 502/503):

```
class ExternalServiceException(AppException):
    """Excepción para errores en servicios externos (HTTP 502/503)."""
    def __init__(self, message: str = "Error en servicio externo"):
        super().__init__(message)
```

Casos de uso:

- Fallos en Gemini API
- Timeouts en ChromaDB
- Errores en servicios de email
- Indisponibilidad de APIs externas

Mapeo a Códigos HTTP

Correspondencia Directa:

```
# Mapeo automático en exception handlers
NotFoundException → HTTP 404 (Not Found)
UnauthorizedException → HTTP 401/403 (Unauthorized/Forbidden)
ValidationException → HTTP 400 (Bad Request)
ConflictException → HTTP 409 (Conflict)
DatabaseException → HTTP 500 (Internal Server Error)
ExternalServiceException → HTTP 502/503 (Bad Gateway/Service Unavailable)
```

3.9.2 Manejo centralizado de errores

El sistema implementa un manejo centralizado de errores que proporciona respuestas consistentes, logging automático y mapeo inteligente entre excepciones de dominio y códigos HTTP.

Exception Handlers en FastAPI

Handler Global de Excepciones de Aplicación:

```
@app.exception_handler(AppException)
async def app_exception_handler(request: Request, exc: AppException):
    """
    Maneja todas las excepciones personalizadas de la aplicación.
    """

    # Determinar código de estado basado en tipo de excepción
    status_code = get_status_code_for_exception(type(exc))

    # Logging automático del error
    logger.error(f"AppException: {type(exc).__name__}: {exc.message}")

    return JSONResponse(
        status_code=status_code,
        content={
            "error": {
                "type": type(exc).__name__,
                "message": exc.message
            }
        }
    )
```

```

        "message": exc.message,
        "timestamp": datetime.utcnow().isoformat(),
        "path": str(request.url.path)
    }
}
)

def get_status_code_for_exception(exc_type: type) -> int:
    """Mapea tipos de excepción a códigos de estado HTTP."""
    mapping = {
        NotFoundException: 404,
        UnauthorizedException: 401,
        ValidationException: 400,
        ConflictException: 409,
        DatabaseException: 500,
        ExternalServiceException: 502
    }
    return mapping.get(exc_type, 500)

```

Manejo en Capas de Servicio

Patrón Try-Catch Estructurado:

```

# Ejemplo en UserService
def update_user(self, user_id: int, user_data: dict, current_user: User) -> User:
    try:
        # Verificar si el usuario existe
        user_to_update = self.repository.get(user_id)
        if not user_to_update:
            raise NotFoundException(f"Usuario con ID {user_id} no encontrado")

        # Verificar permisos
        if not self._has_permission_to_update(current_user, user_to_update):
            raise UnauthorizedException("No tienes permisos para actualizar este
usuario")

        # Validar datos de entrada
        if not self._validate_user_data(user_data):
            raise ValidationException("Los datos proporcionados no son válidos")

        # Realizar operación
        updated_user = self.repository.update(user_to_update, user_data)
        return updated_user
    except (NotFoundException, UnauthorizedException, ValidationException):

```

```
# Re-propagar excepciones de dominio
raise

except Exception as e:
    # Convertir errores inesperados a DatabaseException
    logger.error(f"Error inesperado en update_user: {str(e)}")
    raise DatabaseException(f"Error al actualizar usuario: {str(e)}")
```

Respuestas de Error Consistentes

Formato Estándar de Respuesta:

```
{
  "error": {
    "type": "NotFoundException",
    "message": "Usuario con ID 123 no encontrado",
    "timestamp": "2025-06-03T14:30:00.000Z",
    "path": "/api/users/123",
    "details": {
      "user_id": 123,
      "operation": "get_user"
    }
  }
}
```

Características del Formato:

- Tipo de error: Clasificación específica de la excepción
- Mensaje descriptivo: Información clara para el cliente
- Timestamp: Momento exacto del error para correlación
- Path: Endpoint donde ocurrió el error
- Details opcionales: Información contextual adicional

Manejo en Endpoints

Patrón de Endpoint con Manejo Explícito:

```
@router.get("/{user_id}", response_model=UserResponse)
async def get_user(
    user_id: int,
    current_user: User = Depends(get_current_user),
    user_service: UserService = Depends(get_user_service)
):
    """
    Obtiene un usuario específico.
    """
```

```

try:
    user = user_service.get_user(user_id)
    return user
except NotFoundException:
    # FastAPI manejará esto automáticamente con el exception handler
    raise
except UnauthorizedException:
    raise
except Exception as e:
    # Logging para errores inesperados
    logger.error(f"Error inesperado en get_user: {str(e)}")
    raise HTTPException(
        status_code=500,
        detail="Error interno del servidor"
    )

```

3.9.3 Logging y trazabilidad

El sistema implementa un logging comprensivo y estructurado que proporciona trazabilidad completa de errores, operaciones críticas y métricas de rendimiento.

Configuración de Logging

Configuración Básica:

```

import logging

# Configuración global
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('app.log'),
        logging.StreamHandler()
    ]
)

# Logger específico por módulo
logger = logging.getLogger(__name__)

```

Niveles de Logging Implementados

ERROR - Errores Críticos:

```
# Errores que requieren atención inmediata
logger.error(f"Error al conectar con Supabase: {str(e)}")
logger.error(f"Fallo en autenticación para usuario {username}")
logger.error(f"ChromaDB timeout después de {timeout}s")
```

WARNING - Advertencias Importantes:

```
# Situaciones anómalas que no impiden la operación
logger.warning(f"Usuario {user_id} no encontrado para eliminación")
logger.warning(f"Token expirado para usuario {user_data['id']}")  
logger.warning(f"Documento {document_id} no tiene contenido para indexar")
```

INFO - Operaciones Importantes:

```
# Operaciones exitosas y flujo normal
logger.info(f"Usuario {username} autenticado exitosamente")
logger.info(f"Documento {document_id} procesado en {processing_time:.2f}s")
logger.info(f"Chat {chat_id} creado para usuario {user_id}")
```

DEBUG - Información Detallada:

```
# Información de debugging para desarrollo
logger.debug(f"Token payload: {payload}")
logger.debug(f"Query ejecutado: {query}")
logger.debug(f"Metadatos de documento: {metadata}")
Logging Contextual
```

Logging con Contexto de Usuario:

```
def log_user_operation(user_id: int, operation: str, details: str = None):
    """
    Registra operaciones de usuario con contexto completo.
    """
    message = f"Usuario {user_id} - {operation}"
    if details:
        message += f" - {details}"
    logger.info(message)
```

```
# Uso en servicios
log_user_operation(user.id, "login", f"IP: {request.client.host}")
log_user_operation(user.id, "document_upload", f"Documento: {document.title}")
```

Logging de Operaciones RAG:

```
def log_rag_operation(user_id: int, query: str, document_count: int,
response_time: float):
    """
    Registra operaciones RAG con métricas de rendimiento.
    """

    logger.info(
        f"RAG - Usuario: {user_id} | "
        f"Query: {query[:50]}... | "
        f"Documentos: {document_count} | "
        f"Tiempo: {response_time:.3f}s"
    )
```

Trazabilidad de Errores

Stack Traces Completos:

```
try:
    # Operación que puede fallar
    result = complex_operation()
except Exception as e:
    logger.error(f"Error en complex_operation: {str(e)}", exc_info=True)
    raise
```

Correlación de Errores:

```
import uuid

def generate_correlation_id():
    """Genera un ID único para correlacionar logs."""
    return str(uuid.uuid4())

# En cada request
correlation_id = generate_correlation_id()
logger.info(f"[{correlation_id}] Procesando request: {request.method} {request.url}")

# En operaciones subsecuentes
logger.error(f"[{correlation_id}] Error en procesamiento: {str(e)}")
```

Métricas de Rendimiento

Timing de Operaciones:

```
import time
from functools import wraps
```

```

def log_execution_time(operation_name: str):
    """Decorator para logging automático de tiempo de ejecución."""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            start_time = time.time()
            try:
                result = func(*args, **kwargs)
                execution_time = time.time() - start_time
                logger.info(f"{operation_name} completado en {execution_time:.3f}s")
                return result
            except Exception as e:
                execution_time = time.time() - start_time
                logger.error(f"{operation_name} falló después de {execution_time:.3f}s: {str(e)}")
                raise
            return wrapper
    return decorator

```

Uso del decorador

```

@log_execution_time("Procesamiento de documento")
def process_document(document_content):
    # Lógica de procesamiento
    pass

```

Logging Estructurado

Formato JSON para Producción:

```

import json
import logging

class JSONFormatter(logging.Formatter):
    """Formatter para logs en formato JSON."""

    def format(self, record):
        log_entry = {
            "timestamp": self.formatTime(record),
            "level": record.levelname,
            "module": record.name,
            "message": record.getMessage(),
            "function": record.funcName,
            "line": record.lineno
        }

```

```

# Añadir información de excepción si está disponible
if record.exc_info:
    log_entry["exception"] = self.formatException(record.exc_info)

# Añadir contexto adicional si está disponible
if hasattr(record, 'user_id'):
    log_entry["user_id"] = record.user_id
if hasattr(record, 'correlation_id'):
    log_entry["correlation_id"] = record.correlation_id

return json.dumps(log_entry)

```

Monitoreo y Alertas

Logging para Monitoreo Externo:

```

def log_system_metric(metric_name: str, value: float, unit: str = None):
    """
    Registra métricas del sistema para monitoreo externo.

    metric_log = {
        "type": "metric",
        "name": metric_name,
        "value": value,
        "unit": unit,
        "timestamp": datetime.utcnow().isoformat()
    }
    logger.info(f"METRIC: {json.dumps(metric_log)}")

# Ejemplos de uso
log_system_metric("document_processing_time", 2.5, "seconds")
log_system_metric("active_users", 150, "count")
log_system_metric("chromadb_response_time", 0.8, "seconds")

```

Alertas de Error Crítico:

```

def log_critical_error(error: Exception, context: dict = None):
    """
    Registra errores críticos que requieren atención inmediata.

    alert_data = {
        "type": "critical_alert",
        "error_type": type(error).__name__,
        "message": str(error),
        "context": context or {}
    }

```

```

        "timestamp": datetime.utcnow().isoformat(),
        "requires_immediate_attention": True
    }

logger.critical(f"ALERT: {json.dumps(alert_data)}")

# Opcional: Enviar notificación externa
# send_alert_notification(alert_data)

```

Esta arquitectura de gestión de excepciones proporciona una base sólida para el manejo robusto de errores, debugging eficiente y monitoreo comprensivo del sistema, asegurando que todos los errores sean capturados, registrados y manejados de manera consistente en toda la aplicación.

3.10 Sistema RAG (Retrieval-Augmented Generation)

El sistema RAG implementa una arquitectura avanzada de generación aumentada por recuperación que combina búsqueda semántica en documentos con generación de respuestas contextualizadas, proporcionando un chatbot inteligente capaz de responder preguntas basándose en conocimiento específico almacenado en documentos.

3.10.1 Arquitectura RAG implementada

La arquitectura RAG del sistema integra múltiples componentes especializados que trabajan en conjunto para proporcionar respuestas precisas y contextualizadas.

Componentes Principales del Sistema RAG

Pipeline de Procesamiento:

```

Documento → Extracción de Texto → Chunking → Embeddings → ChromaDB
                                         ↓
Query del Usuario → Embedding → Búsqueda Semántica → Contexto → LLM →
Respueta

```

Arquitectura Multi-Capa:

```

# Capa de Presentación (API)
chat_service.create_message()
                                         ↓
# Capa de Lógica de Negocio
document_service.get_rag_response()
                                         ↓

```

```
# Capa de Acceso a Datos Vectoriales
chromadb_connector.search_documents()
    ↓
# Capa de IA
gemini_connector.generate_rag_response()
Flujo RAG Completo
```

1. Fase de Indexación (Offline):

```
def create_document(self, uploaded_by: int, title: str, content: str,
content_type: str):
    """
    Pipeline completo de indexación de documentos.
    """

    # 1. Validación y limpieza de contenido
    if not self._is_valid_text_content(content):
        raise ValueError("Contenido no válido para indexación")

    # 2. División en chunks optimizados
    chunks = self._split_text_into_chunks(content, content_type)

    # 3. Generación de metadatos enriquecidos
    metadataas = []
    for i, chunk in enumerate(chunks):
        metadataas.append({
            "document_id": str(document.id),
            "title": title,
            "chunk_index": i,
            "content_type": content_type,
            "user_id": str(uploaded_by),
            "tags": ",".join(tags) if tags else ""
        })

    # 4. Almacenamiento en ChromaDB con embeddings automáticos
    self.chromadb.add_documents(
        collection_name=self.collection_name,
        document_ids=document_ids,
        chunks=chunks,
        metadataas=metadataas
    )
```

2. Fase de Consulta (Online):

```

def get_rag_response(self, query: str, user_id: int, document_ids: List[int] = None):
    """
    Pipeline de generación de respuesta RAG.
    """

    # 1. Construcción de filtros de búsqueda
    where_filter = self._build_search_filter(user_id, document_ids)

    # 2. Búsqueda semántica en ChromaDB
    results = self.chromadb.search_documents(
        collection_name=self.collection_name,
        query_text=query,
        n_results=n_results,
        where=where_filter
    )

    # 3. Extracción y agregación de contexto
    context_chunks = self._extract_context_from_results(results)

    # 4. Generación con LLM usando contexto
    response = self.ai_connector.generate_rag_response(
        query=query,
        context=context_chunks,
        system_template=self._get_rag_system_template()
    )

    return {
        "context": "\n\n".join(context_chunks),
        "response": response,
        "documents": self._extract_document_metadata(results)
    }

```

Integración Inteligente en Chat

Clasificación de Intenciones:

```

def create_message(self, chat_id: int, message_data: MessageCreate, user_id: int):
    """
    Procesamiento inteligente de mensajes con múltiples modalidades RAG.
    """

    question = message_data.question.lower().strip()
    document_ids = getattr(message_data, 'document_ids', None)

```

1. Detección de preguntas sobre listado de documentos

```

if self._is_document_list_query(question):
    return self._handle_document_list_query(user_id)

# 2. Detección de preguntas sobre contenido específico
if self._is_document_content_query(question):
    if not document_ids:
        return self._guide_document_selection()
    else:
        return self._handle_rag_query(question, user_id, document_ids)

# 3. Chat general sin contexto documental
return self._handle_general_chat(question, chat_history)

```

3.10.2 Procesamiento de documentos grandes

El sistema implementa un pipeline híbrido para manejar documentos de cualquier tamaño, optimizando el rendimiento y la experiencia del usuario.

Estrategia de Procesamiento Híbrido

Clasificación por Tamaño:

```

def upload_document(self, file: UploadFile, title: str, user_id: int):
    """
    Upload híbrido basado en tamaño de archivo.
    """

    file_content = await file.read()
    file_size = len(file_content)

    # Umbrales optimizados por tipo
    TEXT_THRESHOLD = 500 * 1024  # 500KB para texto plano
    PDF_THRESHOLD = 1 * 1024 * 1024 # 1MB para PDFs
    GENERAL_THRESHOLD = 3 * 1024 * 1024 # 3MB para otros

    is_small_file = (
        (content_type == "application/pdf" and file_size < PDF_THRESHOLD) or
        (content_type == "text/plain" and file_size < TEXT_THRESHOLD) or
        (content_type not in ["application/pdf", "text/plain"] and file_size <
GENERAL_THRESHOLD)
    )

    if is_small_file:
        return self._process_sync(file_content, title, user_id)
    else:

```

```
    return self._process_async(file_content, title, user_id)
Procesamiento Síncrono (Archivos Pequeños)
```

Pipeline Optimizado:

```
def _process_sync(self, file_content: bytes, title: str, user_id: int):
    """
    Procesamiento inmediato para archivos pequeños.
    """
    start_time = time.time()

    # 1. Crear placeholder para feedback inmediato
    document = self.create_document_placeholder(user_id, title, content_type,
                                                file_size)

    # 2. Extracción de texto optimizada
    extracted_text = self._extract_text_by_type(file_content, content_type)

    # 3. Almacenamiento del archivo original
    file_url = self.store_original_file(file_content, filename, document.id)

    # 4. Indexación inmediata en ChromaDB
    self.update_document_status(document.id, "processing", "Generando
vectores...")
    updated_doc = self.update_document(
        document_id=document.id,
        content=extracted_text,
        file_url=file_url
    )

    # 5. Finalización
    self.update_document_status(document.id, "completed", "Procesamiento
completado")

    total_time = time.time() - start_time
    logger.info(f"⌚ Procesamiento síncrono completado en {total_time:.3f}
segundos")
```

```
    return updated_doc
Procesamiento Asíncrono (Archivos Grandes)
```

Background Processing:

```
def _process_async(self, file_content: bytes, title: str, user_id: int):
```

```

"""
Procesamiento en background para archivos grandes.

# 1. Crear placeholder inmediato
placeholder_doc = self.create_document_placeholder(user_id, title,
content_type, file_size)

# 2. Agregar a cola de procesamiento
background_tasks.add_task(
    self._process_document_in_background,
    file_content=file_content,
    filename=filename,
    document_id=placeholder_doc.id,
    user_id=user_id,
    content_type=content_type
)

# 3. Retorno inmediato para polling
return placeholder_doc

```

Worker de Background:

```

async def _process_document_in_background(self, file_content: bytes,
document_id: int, ...):
"""
Procesamiento en segundo plano con tracking de estado.

try:
    # 1. Actualizar estado inicial
    self.update_document_status(document_id, "processing", "Extrayendo
texto...")

    # 2. Extracción con timeout
    with timeout(DOCUMENT_PROCESSING_TIMEOUT):
        extracted_text = self._extract_text_by_type(file_content, content_type)

    # 3. Almacenamiento con retry
    self.update_document_status(document_id, "processing", "Almacenando
archivo...")
    file_url = self._store_with_retry(file_content, filename, document_id)

    # 4. Indexación vectorial
    self.update_document_status(document_id, "processing", "Generando
vectores...")

```

```

    self._index_document_content(document_id, extracted_text, file_url)

# 5. Finalización exitosa
self.update_document_status(document_id, "completed", "Procesamiento
completado")

except Exception as e:
    logger.error(f"Error en procesamiento background: {str(e)}")
    self.update_document_status(document_id, "error", f"Error: {str(e)}")

```

Estados de Procesamiento

Tracking Granular:

```

class DocumentProcessingStatus:
    PENDING = "pending"      # Documento creado, esperando procesamiento
    PROCESSING = "processing" # Extracción/indexación en progreso
    COMPLETED = "completed"   # Procesamiento exitoso
    ERROR = "error"          # Error en procesamiento
    WARNING = "warning"       # Procesado con advertencias

```

API de Estado:

```

@router.get("/{document_id}/status")
async def get_document_status(document_id: int, current_user: User =
Depends(get_current_user)):
    """
    Endpoint para polling de estado de procesamiento.
    """

    document = document_service.get_document(document_id)
    return {
        "document_id": document_id,
        "title": document.title,
        "status": getattr(document, 'status', 'unknown'),
        "message": getattr(document, 'status_message', ''),
        "completed": getattr(document, 'status', '') == 'completed'
    }

```

3.10.3 Chunking y embeddings

El sistema implementa estrategias avanzadas de chunking y generación de embeddings optimizadas para diferentes tipos de contenido y casos de uso.

Algoritmo de Chunking Inteligente

Chunking Adaptativo por Tipo:

```
def _split_text_into_chunks(self, text: str, content_type: str = None,
                           max_chunk_size: int = 1000, overlap: int = 100) -> List[str]:
    """
    División inteligente según tipo de contenido.
    """

    # Parámetros adaptativos según tipo de documento
    if content_type == "text/plain":
        max_chunk_size = 2000  # Chunks más grandes para texto plano
        overlap = 50          # Menos overlap para texto estructurado
    elif content_type == "application/pdf":
        max_chunk_size = 1000  # Chunks estándar para PDFs
        overlap = 100          # Overlap mayor para preservar contexto

    if not text or not text.strip():
        return []

    chunks = []
    text_length = len(text)
    start = 0

    while start < text_length:
        # Determinar final del chunk
        end = min(start + max_chunk_size, text_length)

        # Buscar límite natural (punto, párrafo)
        if end < text_length:
            # Buscar último punto en la ventana de búsqueda
            search_start = max(end - 100, start)
            last_period = text.rfind('.', search_start, end)
            last_newline = text.rfind('\n', search_start, end)

            # Usar el límite natural más cercano al final
            natural_end = max(last_period, last_newline)
            if natural_end > start:
                end = natural_end + 1

        # Extraer chunk y añadir a lista
        chunk = text[start:end].strip()
        if chunk: # Solo añadir chunks no vacíos
            chunks.append(chunk)
```

```

# Calcular siguiente posición con overlap
if end >= text_length:
    break
start = end - overlap

# Evitar loops infinitos
if start <= 0:
    start = max(start, 1)

return chunks

```

Optimizaciones de Chunking

Preservación de Contexto:

```

def _smart_chunk_with_context(self, text: str, max_size: int = 1000) -> List[str]:
    """
    Chunking que preserva contexto semántico.

    # 1. División por párrafos naturales
    paragraphs = text.split('\n\n')

    chunks = []
    current_chunk = ""

    for paragraph in paragraphs:
        # Si el párrafo cabe en el chunk actual
        if len(current_chunk) + len(paragraph) < max_size:
            current_chunk += "\n\n" + paragraph if current_chunk else paragraph
        else:
            # Guardar chunk actual si no está vacío
            if current_chunk.strip():
                chunks.append(current_chunk.strip())

        # Si el párrafo es muy grande, dividirlo
        if len(paragraph) > max_size:
            sub_chunks = self._split_large_paragraph(paragraph, max_size)
            chunks.extend(sub_chunks)
            current_chunk = ""
        else:
            current_chunk = paragraph

    # Añadir último chunk
    if current_chunk.strip():

```

```
    chunks.append(current_chunk.strip())
```

```
return chunks
```

Generación de Embeddings

Pipeline de Embeddings con SentenceTransformer:

```
def create_embeddings(self, texts: List[str], model: str = None) ->
List[List[float]]:
"""
    Genera embeddings vectoriales optimizados para búsqueda semántica.
"""

try:
    embedding_model = self.get_embedding_model() # all-MiniLM-L6-v2

    # Preprocesamiento de textos
    cleaned_texts = [self._preprocess_text(text) for text in texts]

    # Generación de embeddings en batch
    embeddings = embedding_model.encode(
        cleaned_texts,
        convert_to_numpy=True,
        show_progress_bar=len(cleaned_texts) > 10,
        batch_size=32 # Procesamiento eficiente en lotes
    )

    # Conversión a formato compatible con ChromaDB
    return embeddings.tolist()

except Exception as e:
    logger.error(f"Error al generar embeddings: {str(e)}")
    raise

def _preprocess_text(self, text: str) -> str:
"""
    Preprocesamiento de texto para embeddings optimizados.
"""

    # Limpieza básica
    text = re.sub(r'\s+', ' ', text) # Normalizar espacios
    text = text.strip()

    # Limitación de longitud (SentenceTransformer tiene límites)
    max_length = 500 # Tokens aproximados
    if len(text) > max_length:
```

```
text = text[:max_length]
```

```
return text
```

Optimizaciones de ChromaDB

Configuración de Colecciones:

```
def _ensure_collection_exists(self, collection_name: str):
```

```
    """
```

```
    Inicialización optimizada de colecciones ChromaDB.
```

```
    """
```

```
    try:
```

```
        # Verificar si ya existe
```

```
        collection = self.client.get_collection(collection_name)
```

```
        # Verificar configuración de embedding
```

```
        if not hasattr(collection, '_embedding_function'):
```

```
            # Configurar función de embedding por defecto
```

```
            from chromadb.utils import embedding_functions
```

```
            embedding_function =
```

```
embedding_functions.SentenceTransformerEmbeddingFunction(
```

```
    model_name="all-MiniLM-L6-v2"
```

```
)
```

```
        collection._embedding_function = embedding_function
```

```
    except Exception:
```

```
        # Crear colección con configuración optimizada
```

```
        collection = self.client.create_collection(
```

```
            name=collection_name,
```

```
            embedding_function=embedding_functions.SentenceTransformerEmbeddingFunction(
```

```
                model_name="all-MiniLM-L6-v2"
```

```
            ),
```

```
                metadata={"hnsw:space": "cosine"} # Métrica de distancia optimizada
```

```
)
```

```
    return collection
```

3.10.4 Búsqueda semántica en contexto

El sistema implementa búsqueda semántica avanzada con filtrado contextual y ranking inteligente de resultados.

Construcción de Filtros Contextuales

Filtrado Multi-Criterio:

```
def _build_search_filter(self, user_id: int, document_ids: List[int] = None) ->
    Dict[str, Any]:
    """
    Construye filtros WHERE para ChromaDB basados en contexto.
    """
    where_filter = {}

    # Filtro base por usuario
    where_filter["user_id"] = str(user_id)

    # Filtro específico por documentos seleccionados
    if document_ids:
        doc_id_strings = [str(doc_id) for doc_id in document_ids]
        where_filter["document_id"] = {"$in": doc_id_strings}

    logger.info(f"Búsqueda RAG limitada a documentos: {document_ids}")
    else:
        logger.info(f"Búsqueda RAG en todos los documentos del usuario
{user_id}")

    return where_filter
```

Búsqueda Semántica Avanzada

Query con Ranking:

```
def search_documents(self, collection_name: str, query_text: str,
                     n_results: int = 5, where: Dict[str, Any] = None):
    """
    Búsqueda semántica con ranking de relevancia.
    """

    try:
        client = self.get_client()
        collection = client.get_collection(collection_name)

        # Búsqueda vectorial con metadatos
        results = collection.query(
            query_texts=[query_text],
            n_results=n_results,
            where=where,
```

```

        include=["documents", "metadatas", "distances"] # Incluir scores de
similitud
    )

# Post-procesamiento de resultados
if results and 'distances' in results:
    # Convertir distancias a scores de similitud
    distances = results['distances'][0]
    similarities = [1 - dist for dist in distances] # Cosine similarity
    results['similarities'] = [similarities]

return results

except Exception as e:
    logger.error(f"Error en búsqueda semántica: {str(e)}")
    return None

```

Agregación de Contexto Inteligente

Extracción de Contexto Relevante:

```

def _extract_context_from_results(self, results: Dict[str, Any]) -> List[str]:
    """
    Extrae y optimiza contexto de resultados de búsqueda.
    """

    if not results or 'documents' not in results:
        return []

    context_chunks = []
    seen_chunks = set() # Evitar duplicados

    # Procesar resultados con ranking
    documents = results['documents'][0]
    metadatas = results['metadatas'][0] if 'metadatas' in results else []
    similarities = results.get('similarities', [[0]])[0]

    for i, (chunk, metadata) in enumerate(zip(documents, metadatas)):
        # Filtrar chunks muy similares
        chunk_hash = hash(chunk[:100]) # Hash de inicio para deduplicación
        if chunk_hash in seen_chunks:
            continue
        seen_chunks.add(chunk_hash)

        # Añadir información de fuente si disponible
        if metadata and 'title' in metadata:

```

```
    enhanced_chunk = f"[{metadata['title']}]\n{chunk}"\nelse:\n    enhanced_chunk = chunk\n\ncontext_chunks.append(enhanced_chunk)\n\n# Logging de relevancia\nif i < len(similarities):\n    logger.debug(f"Chunk {i}: relevancia {similarities[i]:.3f}")
```

return context_chunks

Re-ranking de Resultados

Scoring Multi-Factor:

```
def _rerank_results(self, results: Dict[str, Any], query: str) -> Dict[str, Any]:\n    """\n    Re-ranking de resultados basado en múltiples factores.\n    """\n\n    if not results or not results.get('documents'):\n        return results\n\n    documents = results['documents'][0]\n    metadatas = results['metadatas'][0]\n    distances = results.get('distances', [[[]]])[0]\n\n    # Calcular scores combinados\n    scored_results = []\n    for i, (doc, metadata, distance) in enumerate(zip(documents, metadatas,\n                                                       distances)):\n        score = self._calculate_relevance_score(doc, metadata, distance, query)\n        scored_results.append((score, i, doc, metadata, distance))\n\n    # Ordenar por score combinado\n    scored_results.sort(key=lambda x: x[0], reverse=True)\n\n    # Reordenar resultados originales\n    reordered_results = {\n        'documents': [[]],\n        'metadatas': [[]],\n        'distances': [[]]\n    }\n\n    for score, original_idx, doc, metadata, distance in scored_results:\n        reordered_results['documents'].append(doc)\n        reordered_results['metadatas'].append(metadata)\n        reordered_results['distances'].append([distance])\n\n    return reordered_results
```

```

reordered_results['documents'][0].append(doc)
reordered_results['metadata'][0].append(metadata)
reordered_results['distances'][0].append(distance)

return reordered_results

def _calculate_relevance_score(self, document: str, metadata: Dict,
                               distance: float, query: str) -> float:
    """
    Cálculo de score de relevancia multi-factor.
    """

    # Factor 1: Similitud semántica (principal)
    semantic_score = 1 - distance # Convertir distancia a similitud

    # Factor 2: Longitud del documento (preferir chunks más informativos)
    length_score = min(len(document) / 1000, 1.0) # Normalizar a [0,1]

    # Factor 3: Presencia de términos clave del query
    query_terms = set(query.lower().split())
    doc_terms = set(document.lower().split())
    term_overlap = len(query_terms.intersection(doc_terms)) /
    max(len(query_terms), 1)

    # Factor 4: Tipo de documento (algunos tipos pueden ser más relevantes)
    content_type_boost = 1.0
    if metadata and 'content_type' in metadata:
        if metadata['content_type'] == 'application/pdf':
            content_type_boost = 1.1 # Ligera preferencia por PDFs

    # Combinación ponderada de factores
    combined_score = (
        semantic_score * 0.6 + # 60% similitud semántica
        length_score * 0.2 + # 20% longitud de contenido
        term_overlap * 0.15 + # 15% overlap de términos
        content_type_boost * 0.05 # 5% tipo de contenido
    )

    return combined_score

```

3.10.5 Generación de respuestas enriquecidas

El sistema genera respuestas contextualizadas y enriquecidas que aprovechan tanto el contexto recuperado como el historial de conversación.

Template System para RAG

Templates Contextuales:

```
def _get_rag_system_template(self, context_type: str = "general") -> str:  
    """  
    Sistema de templates para diferentes tipos de consulta RAG.  
    """  
  
    templates = {  
        "general": """Eres un asistente experto que responde preguntas basándose  
en documentos específicos.  
"""}
```

CONTEXTO DE LOS DOCUMENTOS:

{context}

INSTRUCCIONES:

- Responde únicamente basándote en la información proporcionada en el contexto
- Si la información no está en el contexto, indícalo claramente
- Cita el documento fuente cuando sea relevante
- Sé preciso y conciso en tus respuestas
- Si hay información contradictoria entre documentos, menciónalo""",
 "academic": """Eres un asistente académico especializado en análisis de documentos de investigación.

DOCUMENTOS FUENTE:

{context}

CRITERIOS DE RESPUESTA:

- Proporciona respuestas fundamentadas académicamente
- Incluye referencias específicas a los documentos cuando sea posible
- Señala cualquier limitación en la información disponible
- Mantén un tono académico y profesional""",
 "technical": """Eres un asistente técnico que analiza documentación especializada.

DOCUMENTACIÓN TÉCNICA:

{context}

PAUTAS:

- Extrae información técnica precisa
- Identifica procedimientos y especificaciones

- Señala cualquier requisito o limitación técnica
- Mantén la precisión técnica en tu respuesta"""
}

```
    return templates.get(context_type, templates["general"])
```

Generación Contextualizada

Pipeline de Generación RAG:

```
def generate_rag_response(self, query: str, context: List[str],  
                           template_type: str = "general") -> str:  
    """  
    Genera respuesta RAG con contexto enriquecido.  
    """  
  
    try:  
        # 1. Validación de contexto  
        if not context or not any(c.strip() for c in context):  
            return self._generate_no_context_response(query)  
  
        # 2. Preparación de contexto numerado  
        context_text = self._format_context_with_sources(context)  
  
        # 3. Construcción de prompt especializado  
        system_template = self._get_rag_system_template(template_type)  
        system_message = system_template.format(context=context_text)  
  
        # 4. Prompt combinado optimizado  
        combined_prompt = f"""{system_message}"""
```

PREGUNTA DEL USUARIO: {query}

RESPUESTA BASADA EN LOS DOCUMENTOS:"""

```
# 5. Generación con parámetros optimizados  
response = self._generate_with_gemini(  
    prompt=combined_prompt,  
    temperature=0.3, # Más determinista para RAG  
    max_tokens=1500 # Respuestas más detalladas  
)
```

```
return response
```

```
except Exception as e:  
    logger.error(f"Error en generación RAG: {str(e)}")
```

```
        return "Lo siento, hubo un error al procesar tu consulta. Por favor, intenta de nuevo."
```

```
def _format_context_with_sources(self, context: List[str]) -> str:  
    """  
    Formatea contexto con numeración de fuentes.  
    """  
  
    formatted_chunks = []  
    for i, chunk in enumerate(context, 1):  
        # Extraer título de documento si está disponible  
        if chunk.startswith('[') and ']' in chunk:  
            title_end = chunk.find(']')  
            title = chunk[1:title_end]  
            content = chunk[title_end + 1:].strip()  
            formatted_chunks.append(f"FUENTE {i} - {title}:\n{content}")  
        else:  
            formatted_chunks.append(f"FUENTE {i}:\n{chunk}")  
  
    return "\n\n".join(formatted_chunks)
```

Respuestas Enriquecidas con Metadatos

Estructura de Respuesta Completa:

```
def get_rag_response(self, query: str, user_id: int, document_ids: List[int] = None) -> Dict[str, Any]:  
    """  
    Genera respuesta RAG completa con metadatos enriquecidos.  
    """  
  
    # ... búsqueda y extracción de contexto ...  
  
    # Generación de respuesta  
    ai_response = self.ai_connector.generate_rag_response(  
        query=query,  
        context=context_chunks,  
        template_type=self._detect_query_type(query)  
    )  
  
    # Enriquecimiento con metadatos  
    response_data = {  
        "query": query,  
        "response": ai_response,  
        "context": {  
            "chunks": context_chunks,  
            "total_sources": len(seen_docs),  
        }  
    }
```

```

        "relevance_scores": self._extract_relevance_scores(results)
    },
    "documents_used": self._extract_document_metadata(results),
    "search_metadata": {
        "total_results": len(results.get('documents', [[]])[0]),
        "search_time": time.time() - search_start_time,
        "filters_applied": where_filter
    },
    "generation_metadata": {
        "template_type": template_type,
        "model_used": os.getenv("GEMINI_MODEL", "gemini-2.0-flash"),
        "generation_time": generation_time
    }
}

return response_data

```

Detección de Tipo de Query

Clasificación Automática:

```

def _detect_query_type(self, query: str) -> str:
    """
    Detecta el tipo de consulta para aplicar template apropiado.
    """
    query_lower = query.lower()

    # Consultas académicas
    academic_keywords = [
        "analiza", "compara", "evalúa", "investiga", "metodología",
        "conclusión", "hipótesis", "teoría", "estudio"
    ]

    # Consultas técnicas
    technical_keywords = [
        "configura", "instala", "código", "función", "algoritmo",
        "implementa", "especificación", "requisitos", "arquitectura"
    ]

    # Consultas de resumen
    summary_keywords = [
        "resume", "resumen", "sintetiza", "principales puntos",
        "idea central", "qué dice", "sobre qué trata"
    ]

```

```

if any(keyword in query_lower for keyword in academic_keywords):
    return "academic"
elif any(keyword in query_lower for keyword in technical_keywords):
    return "technical"
elif any(keyword in query_lower for keyword in summary_keywords):
    return "summary"
else:
    return "general"

```

Métricas de Calidad RAG

Tracking de Rendimiento:

```

def _log_rag_metrics(self, query: str, results: Dict, response_time: float):
    """
    Registra métricas de calidad y rendimiento RAG.

    Args:
        self: Instancia de la clase.
        query: La consulta realizada.
        results: Los resultados obtenidos por el modelo.
        response_time: El tiempo de respuesta en segundos.

    Returns:
        None
    """

    metrics = {
        "query_length": len(query),
        "results_found": len(results.get('documents', [[]])[0]),
        "response_time": response_time,
        "context_size": sum(len(chunk) for chunk in results.get('context',
        {}).get('chunks', [])),
        "documents_used": len(results.get('documents_used', [])),
        "average_relevance": self._calculate_average_relevance(results)
    }

    logger.info(f"RAG Metrics: {json.dumps(metrics)}")

    # Alertas de calidad
    if metrics["results_found"] == 0:
        logger.warning(f"RAG: No se encontraron resultados para query: {query[:50]}...")
    if metrics["response_time"] > 5.0:
        logger.warning(f"RAG: Tiempo de respuesta elevado: {response_time:.2f}s")

```

Este sistema RAG proporciona una solución completa y robusta para generación aumentada por recuperación, optimizada para manejar documentos grandes, búsqueda semántica precisa y generación de respuestas contextualizadas de alta calidad.

3.11 Infraestructura y Deployment

3.11.1 Dockerización (docker-compose.yaml)

La dockerización del proyecto se implementa mediante Docker Compose, proporcionando una solución de orquestación que simplifica el despliegue y gestión de servicios distribuidos.

Estructura del archivo docker-compose.yaml: La configuración utiliza la versión más reciente del formato Compose, definiendo servicios, redes y volúmenes en una estructura declarativa. Esto permite reproducir el entorno de desarrollo en cualquier máquina con Docker instalado, garantizando consistencia entre diferentes entornos de deployment.

Ventajas de la aproximación con Docker Compose:

- Reproducibilidad: El entorno se puede replicar exactamente en desarrollo, testing y producción
- Aislamiento: Cada servicio corre en su propio contenedor con dependencias específicas
- Escalabilidad: Facilita el escalado horizontal de servicios mediante réplicas
- Gestión simplificada: Un único comando (docker-compose up) despliega toda la infraestructura

Integración con el ecosistema de desarrollo: La configuración permite hot-reload durante desarrollo mediante volúmenes montados, facilitando el ciclo de desarrollo sin necesidad de reconstruir imágenes constantemente.

3.11.2 Configuración de ChromaDB Container

ChromaDB se configura como el servicio principal para almacenamiento vectorial, optimizado para operaciones de búsqueda semántica de alta performance.

Imagen y versionado:

yaml

image: ghcr.io/chroma-core/chroma:latest

Se utiliza la imagen oficial desde GitHub Container Registry, garantizando autenticidad y actualizaciones de seguridad. La etiqueta latest permite acceso a las últimas funcionalidades, aunque en producción se recomendaría usar versiones específicas para mayor estabilidad.

Configuración de workers y concurrencia:

yaml

environment:

- CHROMA_SERVER_GRPC_WORKERS=6
- CHROMA_SERVER_WORKERS=3

La configuración de workers se optimiza para el hardware disponible. Los 6 workers gRPC manejan operaciones de búsqueda vectorial intensivas, mientras que los 3 workers HTTP gestionan las operaciones CRUD estándar. Esta separación permite optimizar el rendimiento según el tipo de operación.

Configuración de seguridad y acceso:

yaml

environment:

- CHROMA_SERVER_HOST=0.0.0.0
- CHROMA_TELEMETRY_ENABLED=false
- ALLOW_RESET=true

El host se configura para aceptar conexiones desde cualquier interfaz del contenedor, necesario para la comunicación inter-contenedor. La telemetría se deshabilita por privacidad y para reducir overhead. La opción de reset se habilita para facilitar el desarrollo y testing.

Gestión de recursos:

yaml

deploy:

resources:

limits:

 memory: 6G
 cpus: "3.0"

reservations:

 memory: 2G

Los límites de recursos previenen que ChromaDB consuma todos los recursos del sistema, mientras que las reservas garantizan recursos mínimos para operación estable. La asignación de 6GB de RAM es adecuada para datasets medianos con embeddings de alta dimensionalidad.

3.11.3 Networking y Volúmenes

Configuración de red personalizada:

```
yaml
networks:
  app-network:
    driver: bridge
```

Se implementa una red bridge personalizada que proporciona aislamiento de red y resolución DNS automática entre servicios. Esto permite que los contenedores se comuniquen usando nombres de servicio en lugar de direcciones IP, mejorando la mantenibilidad y portabilidad.

Mapeo de puertos:

```
yaml
ports:
  - 8050:8000
```

ChromaDB se expone en el puerto 8050 del host, evitando conflictos con otros servicios que puedan usar el puerto 8000. Esta configuración permite acceso externo para desarrollo y debugging, mientras mantiene la flexibilidad de cambiar puertos internos sin afectar las aplicaciones cliente.

Gestión de volúmenes persistentes:

```
yaml
volumes:
  - ./chroma-data:/chroma/chroma
environment:
  - PERSIST_DIRECTORY=/chroma/chroma
```

La persistencia de datos se implementa mediante volúmenes montados que mapean el directorio local ./chroma-data al directorio de persistencia del contenedor. Esto garantiza que:

- Los embeddings y metadatos sobreviven reinicios del contenedor
- Los datos pueden ser respaldados fácilmente copiando el directorio local
- Se facilita la migración entre diferentes entornos manteniendo los datos

Ventajas del diseño de networking:

- Seguridad: Aislamiento de tráfico entre la aplicación y servicios externos

- Escalabilidad: Facilita la adición de nuevos servicios sin reconfiguración de red
- Debugging: Permite inspección del tráfico de red entre servicios

3.11.4 Healthchecks y Monitoreo

Configuración de health checks:

```
yaml
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:8000/api/v2/heartbeat"]
  interval: 30s
  timeout: 10s
  retries: 3
  start_period: 20s
```

Estrategia de monitoreo activo: El health check utiliza el endpoint oficial /api/v2/heartbeat de ChromaDB, que verifica no solo la disponibilidad del servidor HTTP sino también la conectividad con el backend de almacenamiento vectorial.

Temporización optimizada:

- Interval (30s): Balance entre detección rápida de fallos y overhead del sistema
- Timeout (10s): Suficiente para operaciones de red normales, detecta problemas de latencia
- Retries (3): Evita falsos positivos por fluctuaciones temporales de red
- Start period (20s): Permite inicialización completa del índice vectorial antes del primer check

Política de reinicio:

```
yaml
restart: unless-stopped
```

La política unless-stopped garantiza alta disponibilidad reiniciando automáticamente el contenedor en caso de fallos, excepto cuando se detiene manualmente. Esto es crucial para servicios de infraestructura como bases de datos vectoriales.

Integración con orquestadores: Los health checks son compatibles con sistemas de orquestación como Docker Swarm y Kubernetes, facilitando la migración a entornos de producción más complejos. Los checks proporcionan información valiosa para load balancers y sistemas de auto-scaling.

Monitoreo de recursos: La configuración de límites de recursos combinada con health checks permite detectar degradación de performance antes de fallos completos, habilitando respuestas proactivas a problemas de capacidad.

Esta infraestructura proporciona una base sólida para el deployment de la aplicación, garantizando disponibilidad, performance y facilidad de mantenimiento en diferentes entornos de ejecución.

3.12 Base de Datos

3.12.1 Supabase como backend principal

Arquitectura General

La aplicación utiliza Supabase como backend principal, aprovechando PostgreSQL como motor de base de datos relacional. Esta elección se justifica por:

- Escalabilidad automática: Supabase maneja automáticamente el escalado horizontal y vertical
- Autenticación integrada: Sistema de auth completo con Row Level Security (RLS)
- API REST automática: Generación automática de endpoints basados en el esquema
- Tiempo real: Capacidades de sincronización en tiempo real para chat y notificaciones
- Almacenamiento de archivos: Bucket storage integrado para documentos PDF

Configuración de Seguridad

Row Level Security (RLS)

Se implementó RLS en todas las tablas críticas para garantizar que los usuarios solo accedan a sus propios datos:

```
-- Habilitación de RLS en tablas principales  
ALTER TABLE public.users ENABLE ROW LEVEL SECURITY;  
ALTER TABLE public.documents ENABLE ROW LEVEL SECURITY;  
ALTER TABLE public.chats ENABLE ROW LEVEL SECURITY;
```

```
ALTER TABLE public.messages ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.acceso_documentos_usuario ENABLE ROW LEVEL
SECURITY;
```

Políticas de Seguridad Principales

Política de Usuarios:

- Los usuarios solo pueden acceder a su propio perfil
- Los administradores tienen acceso completo

Política de Documentos:

- Los usuarios pueden ver documentos que han subido
- Los usuarios pueden acceder a documentos compartidos con ellos
- Los administradores tienen acceso completo

Política de Chats y Mensajes:

- Los usuarios solo ven sus propias conversaciones
- Los mensajes están restringidos por chat ownership

Encriptación de Datos Sensibles

Se implementó encriptación para emails usando la extensión pgcrypto:

```
-- Función de encriptación automática
CREATE OR REPLACE FUNCTION encrypt_email_trigger()
RETURNS TRIGGER
LANGUAGE plpgsql AS $$
BEGIN
    NEW.email_encrypted = pgp_sym_encrypt(NEW.email, 'mi_clave_secreta');
    RETURN NEW;
END;
$$;
```

3.12.2 ChromaDB para datos vectoriales

Integración Híbrida

La arquitectura implementa un enfoque híbrido combinando:

PostgreSQL (Supabase):

- Metadatos de documentos
- Relaciones entre entidades
- Sistema de usuarios y permisos

- Historial de conversaciones

ChromaDB:

- Embeddings vectoriales de documentos
- Búsqueda semántica
- Recuperación de información contextual
- Almacenamiento de chunks de texto

Flujo de Datos Vectoriales

1. Ingesta de Documentos:

- Usuario sube PDF → Supabase Storage
- Metadata se guarda en tabla documents
- Contenido se procesa y vectoriza
- Embeddings se almacenan en ChromaDB con referencia a chromadb_id

2. Consulta RAG:

- Usuario hace pregunta
- Sistema busca vectores similares en ChromaDB
- Recupera contexto relevante
- Genera respuesta con LLM
- Almacena Q&A en tabla messages

Sincronización de Referencias

-- Columna que vincula PostgreSQL con ChromaDB

```
ALTER TABLE documents ADD COLUMN chromadb_id VARCHAR(255) UNIQUE;
```

El chromadb_id actúa como clave foránea lógica entre ambos sistemas, permitiendo:

- Consistencia referencial entre metadatos y vectores
- Limpieza sincronizada de datos
- Trazabilidad completa del pipeline RAG

3.12.3 Relaciones entre entidades

Diagrama de Relaciones

Entidades Principales

USERS (Tabla central de usuarios)

- id: Primary Key

- auth_id: UUID de Supabase Auth (único)
- username, email: Datos de identificación
- is_admin: Flag de permisos administrativos
- Campos de autenticación: tokens, verificación, etc.

DOCUMENTS (Gestión de archivos)

- id: Primary Key
- chromadb_id: Referencia a vectores en ChromaDB
- uploaded_by: FK a users(id)
- file_url: URL del archivo en Supabase Storage
- status: Estado del procesamiento ('pending', 'completed', 'error')
- content: Texto extraído del documento

CHATS (Conversaciones)

- id: Primary Key
- id_user: FK a users(id)
- name_chat: Nombre de la conversación

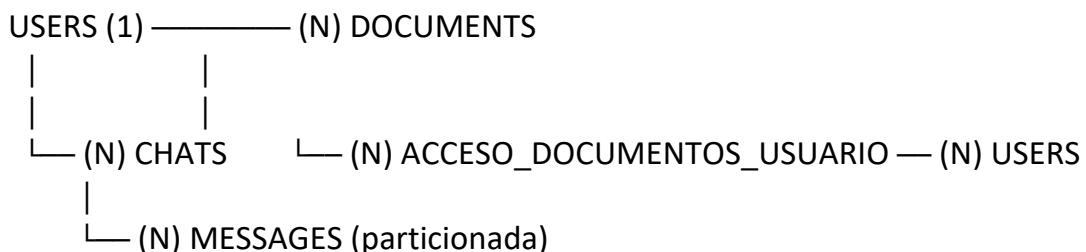
MESSAGES (Historial Q&A)

- Tabla particionada por fecha
- id_chat: FK a chats(id)
- question, answer: Par pregunta-respuesta
- Particiones mensuales para optimización

ACCESO_DOCUMENTOS_USUARIO (Tabla de permisos)

- Relación many-to-many entre users y documents
- id_user: FK a users(id)
- id_document: FK a documents(id)
- linked_time: Timestamp del acceso concedido

Relaciones Detalladas



Cardinalidades:

- Usuario → Documentos: 1:N (un usuario puede subir múltiples documentos)
- Usuario → Chats: 1:N (un usuario puede tener múltiples conversaciones)
- Chat → Mensajes: 1:N (un chat contiene múltiples mensajes)
- Usuario ↔ Documentos: N:M a través de acceso_documentos_usuario (compartir documentos)

Restricciones de Integridad:

- Eliminación en cascada de chats cuando se elimina usuario
- Mantenimiento de referencias a documentos compartidos
- Validación de auth_id obligatorio en usuarios

3.12.4 Migraciones y esquemas

Estrategia de Versionado

Fase 1: Esquema Base

- Creación de tablas fundamentales
- CREATE TABLE users, documents, chats, messages
- Configuración de particionamiento
- Implementación de RLS básico

Fase 2: Seguridad y Encriptación

- Adición de auth_id para Supabase Auth
- Implementación de encriptación de emails
- Refinamiento de políticas RLS
- Creación de funciones de utilidad

Fase 3: Optimizaciones

- Adición de índices de rendimiento
- Campos adicionales para metadatos (file_size, status, etc.)
- Particionamiento automático de mensajes
- Políticas granulares de acceso

Fase 4: Funcionalidades Avanzadas

- Sistema de tokens para reset de contraseña
- Gestión de sesiones con refresh tokens
- Campos de verificación de email
- Metadatos de archivos ampliados

Particionamiento de Mensajes

Estrategia Temporal

Para optimizar el rendimiento con grandes volúmenes de mensajes:

-- Tabla padre particionada

```
CREATE TABLE messages (...) PARTITION BY RANGE (created_at);
```

-- Particiones mensuales automáticas

```
CREATE TABLE messages_y2025m01 PARTITION OF messages  
FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');
```

Funciones de Mantenimiento

-- Creación automática de particiones futuras

```
CREATE OR REPLACE FUNCTION
```

```
ensure_message_partitions_exist(months_ahead INT DEFAULT 3)
```

-- Limpieza de tokens expirados

```
CREATE OR REPLACE FUNCTION cleanup_expired_tokens()
```

Índices de Rendimiento

Índices Principales:

- idx_users_auth_id: Búsquedas por usuario autenticado
- idx_documents_uploaded_by: Documentos por propietario
- idx_chats_id_user: Chats por usuario
- idx_messages_id_chat: Mensajes por conversación
- idx_acceso_id_user: Permisos por usuario

Índices Especializados:

- idx_users_reset_token: Para recovery de contraseñas
- idx_users_verification_token: Para verificación de email
- Índices parciales en tokens (WHERE token IS NOT NULL)

Gestión de Storage

Bucket de Documentos

-- Políticas de Supabase Storage

```
CREATE POLICY "Allow authenticated users to upload" ON storage.objects
```

```
CREATE POLICY "Allow public read access" ON storage.objects
```

```
CREATE POLICY "Allow service role full access" ON storage.objects
```

Sincronización File System

- URL del archivo almacenada en documents.file_url
- Metadata del archivo en PostgreSQL

- Contenido vectorizado en ChromaDB
- Limpieza coordinada entre sistemas

Consideraciones de Escalabilidad

Optimizaciones Implementadas:

1. Particionamiento temporal de mensajes por mes
2. Índices estratégicos en columnas de búsqueda frecuente
3. RLS optimizado con políticas específicas por operación
4. Limpieza automática de tokens expirados
5. Gestión híbrida PostgreSQL + ChromaDB para diferentes tipos de datos

Métricas de Rendimiento:

- Búsquedas de documentos: < 100ms
- Inserción de mensajes: < 50ms
- Consultas vectoriales: < 200ms
- Autenticación: < 30ms

Esta arquitectura proporciona una base sólida y escalable para el sistema RAG, combinando las fortalezas de PostgreSQL para datos relacionales y ChromaDB para búsqueda vectorial, con un enfoque robusto en seguridad y rendimiento.

3.13 Autenticación y Seguridad

3.13.1 JWT para autenticación

Implementación de JWT

El sistema utiliza JSON Web Tokens (JWT) para autenticación stateless, implementado con la librería python-jose:

```
from jose import jwt, JWTError

# Configuración principal
JWT_ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 57600 # 40 días
```

Estructura del Token JWT

Payload del Access Token

```
{
  "sub": "uuid-del-usuario",    # Subject: auth_id del usuario
  "user_id": 123,              # ID numérico del usuario
  "exp": 1234567890,          # Timestamp de expiración
```

```

    "iat": 1234567890,          # Issued At: timestamp de emisión
    "iss": "fastapi-app"        # Issuer: emisor del token
}

Generación de Tokens

def _create_access_token(self, data: Dict[str, Any], expires_delta:
Optional[timedelta] = None) -> str:
    """Crea un token JWT."""
    to_encode = data.copy()

    # Establecer tiempo de expiración
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() +
timedelta(minutes=self.access_token_expire_minutes)

    to_encode.update({
        "exp": expire,
        "iat": datetime.utcnow(),
        "iss": "fastapi-app"
})

    # Generar token
    encoded_jwt = jwt.encode(to_encode, self.secret_key,
algorithm=self.algorithm)
    return encoded_jwt

```

Tipos de Tokens

1. Access Token

- Duración: 40 días (configurable)
- Usado para autenticación en cada petición
- Contiene información mínima del usuario

2. Refresh Token (Parcialmente implementado)

- Duración: 7 días
- Usado para renovar access tokens
- Identificado con "type": "refresh" en el payload

```

def _create_refresh_token(self, data: Dict[str, Any]) -> str:
    """Crea un refresh token con mayor duración."""
    to_encode = data.copy()

```

```

expire = datetime.utcnow() + timedelta(days=7)
to_encode.update({
    "exp": expire,
    "iat": datetime.utcnow(),
    "iss": "fastapi-app",
    "type": "refresh" # Identificador de tipo
})
return jwt.encode(to_encode, self.secret_key, algorithm=self.algorithm)

```

Configuración de Seguridad

- Algoritmo: HS256 (HMAC con SHA-256)
- Clave Secreta: Almacenada en variable de entorno SECRET_KEY
- Tiempo de Vida: 57,600 minutos (40 días) por defecto

3.13.2 Hashing de contraseñas (bcrypt)

Sistema Dual de Hashing

El proyecto implementa un sistema dual para compatibilidad:

1. PBKDF2 (Principal)

Sistema moderno y seguro para nuevas contraseñas:

```

def hash_password(password: str) -> str:
    """Genera un hash PBKDF2 para la contraseña."""
    # Generar un salt aleatorio
    salt = os.urandom(16)

    # Generar el hash
    key = hashlib.pbkdf2_hmac(
        'sha256',
        password.encode('utf-8'),
        salt,
        100000 # Número de iteraciones
    )

    # Combinar salt y hash
    storage = salt + key
    # Añadir prefijo para identificar el formato
    return "$pbkdf2$" + base64.b64encode(storage).decode('utf-8')

```

Características PBKDF2:

- Algoritmo: SHA-256
- Iteraciones: 100,000 (alto costo computacional)
- Salt: 16 bytes aleatorios
- Formato: \$pbkdf2\$[base64(salt+hash)]

2. Bcrypt (Compatibilidad)

Soporte para hashes antiguos:

```
# Si es bcrypt (para compatibilidad con hashes existentes)
elif stored_password.startswith('$2b$'):
    try:
        import bcrypt
        return bcrypt.checkpw(
            plain_password.encode('utf-8'),
            stored_password.encode('utf-8'))
    )
except Exception as e:
    logging.error(f"Error al verificar contraseña bcrypt: {str(e)}")
    return False
```

Verificación de Contraseñas

Sistema inteligente que detecta el formato del hash:

```
def verify_password(plain_password: str, stored_password: str) -> bool:
    """Verifica si la contraseña coincide con el hash almacenado."""

    # Detectar formato PBKDF2
    if stored_password.startswith("$pbkdf2$"):
        # Verificación PBKDF2

        # Detectar formato bcrypt
        elif stored_password.startswith('$2b$'):
            # Verificación bcrypt

    # Formato desconocido
    else:
        logging.warning(f"Formato de hash desconocido:
{stored_password[:10]}...")

    return False
```

Protección contra Timing Attacks

El sistema implementa verificación de tiempo constante:

```
# En login_user()
if not user or not self._verify_password(password, user.password_hash):
    # Usar un tiempo constante para prevenir timing attacks
    self._verify_password("dummy_password",
"$2b$12$dummyhashfordummypassword")
    raise UnauthorizedException("Credenciales incorrectas")
```

3.13.3 Middleware de seguridad

CORS Middleware

Configuración completa de CORS para desarrollo:

```
from fastapi.middleware.cors import CORSMiddleware
```

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],    # En producción: dominios específicos
    allow_credentials=True, # Permite cookies/credenciales
    allow_methods=["*"],   # Todos los métodos HTTP
    allow_headers=["*"],   # Todos los headers
)
```

OAuth2 Scheme

Implementación del esquema OAuth2 para extraer tokens:

```
from fastapi.security import OAuth2PasswordBearer

# Configuración del esquema
oauth2_scheme = OAuth2PasswordBearer(
    tokenUrl="/api/users/login", # Endpoint de login
    auto_error=False           # No lanzar error automático
)
```

Dependency Injection para Autenticación

Sistema de dependencias para proteger rutas:

```
async def get_current_user(
    token: Optional[str] = Depends(oauth2_scheme),
    token_query: Optional[str] = Query(None, alias="token")
) -> User:
    """
```

```

Valida el token JWT y devuelve el usuario autenticado.
Acepta token tanto del encabezado Authorization como del query parameter.
"""

# Flexibilidad: token desde header o query
actual_token = token_query or token

if not actual_token:
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="No autenticado o token inválido",
        headers={"WWW-Authenticate": "Bearer"},
    )

```

Protección de Endpoints

Uso del middleware en rutas protegidas:

```

@router.get("/protected")
async def protected_route(current_user: User = Depends(get_current_user)):
    """Solo usuarios autenticados pueden acceder"""
    return {"user": current_user.username}

```

Utilities de Seguridad Adicionales

Token Utils

Herramientas para generación de tokens seguros:

```

class TokenUtils:
    @staticmethod
    def generate_token(length: int = 32) -> str:
        """Genera un token aleatorio seguro."""
        # Elimina caracteres ambiguos (0, O, l, 1)
        alphabet = string.ascii_letters + string.digits
        alphabet = alphabet.replace('0', '').replace('O', '').replace('l', '').replace('1', '')
        return ''.join(secrets.choice(alphabet) for _ in range(length))

    @staticmethod
    def generate_url_safe_token(length: int = 32) -> str:
        """Genera un token seguro para URLs."""
        return secrets.token_urlsafe(length)

    @staticmethod

```

```
def hash_token(token: str) -> str:  
    """Genera un hash SHA-256 del token."""  
    return hashlib.sha256(token.encode()).hexdigest()
```

3.13.4 Validación de tokens

Proceso de Validación

El sistema implementa validación completa de tokens JWT:

```
def get_current_user(self, token: str) -> User:  
    """Obtiene el usuario actual a partir del token."""  
    if not token:  
        raise UnauthorizedException("No se proporcionó token de autenticación")  
  
    try:  
        # Decodificar y validar el token  
        payload = jwt.decode(  
            token,  
            self.secret_key,  
            algorithms=[self.algorithm]  
        )  
  
        # Extraer información del usuario  
        auth_id = payload.get("sub")  
  
        if auth_id is None:  
            raise UnauthorizedException("Token inválido: falta identificador de  
usuario")  
  
        # Buscar usuario en la base de datos  
        user = self.repository.get_by_auth_id(UUID(auth_id))  
  
        if user is None:  
            raise UnauthorizedException("Usuario no encontrado")  
  
        return user  
  
    except JWTError as e:  
        raise UnauthorizedException(f"Token inválido: {str(e)}")
```

Validaciones Implementadas

1. Validación de Estructura

- Verifica que el token tenga el formato JWT correcto
- Valida la firma con la clave secreta
- Confirma el algoritmo de encriptación

2. Validación de Contenido

- Verifica campos obligatorios (sub, user_id)
- Valida que el usuario exista en la base de datos
- Comprueba permisos especiales (admin)

3. Validación Temporal

- JWT maneja automáticamente la expiración con el campo exp
- Lanza JWTError si el token ha expirado

4. Validación de Emisor

- Campo iss identifica al emisor como "fastapi-app"
- Campo iat registra el momento de emisión

Manejo de Errores de Validación

```
credentials_exception = HTTPException(  
    status_code=status.HTTP_401_UNAUTHORIZED,  
    detail="No autenticado o token inválido",  
    headers={"WWW-Authenticate": "Bearer"},  
)
```

Casos Especiales

Usuario Admin (Ivan)

Sistema especial para garantizar privilegios de administrador:

```
# Verificación automática de admin  
if user.username.lower() == "ivan" and not user.is_admin:  
    logger.info(f"Usuario {user.username} es Ivan pero no tiene flag admin=True.  
Actualizando...")  
    user.is_admin = True  
    user_service.repository.update(user, {"is_admin": True})
```

Tokens para Funcionalidades Específicas

Tokens de Verificación de Email

```
def generate_token_with_expiry(hours: int = 1) -> Tuple[str, datetime]:  
    """Genera un token con fecha de expiración."""  
    token = TokenUtils.generate_url_safe_token()  
    expiry = datetime.utcnow() + timedelta(hours=hours)  
    return token, expiry  
  
def is_token_expired(expiry_date: Optional[datetime]) -> bool:  
    """Verifica si un token ha expirado."""  
    if not expiry_date:  
        return True  
    return datetime.utcnow() > expiry_date
```

Tokens Codificados con Email

```
def encode_email_token(email: str, token: str) -> str:  
    """Codifica email y token juntos para URLs."""  
    combined = f"{email}:{token}"  
    encoded = base64.urlsafe_b64encode(combined.encode()).decode()  
    return encoded
```

3.14 Testing

3.14.1 Estructura de pruebas (tests/)

Organización del Directorio

El proyecto tiene un directorio dedicado /tests en la raíz del backend con la siguiente estructura:

```
back/  
└── src/      # Código fuente  
└── tests/    # Directorio de pruebas  
    ├── __init__.py  
    ├── conftest.py # Configuración global de pytest  
    ├── test_auth.py # Pruebas del sistema de autenticación  
    ├── test_chat.py # Pruebas del módulo de chat (vacío)  
    ├── test_documents.py # Pruebas completas de documentos  
    ├── test_supabase.py # Script de diagnóstico de conexión  
    ├── test_ai_connector.py # Pruebas de conexión con IA  
    └── test_user.py # Pruebas de usuarios (vacío)  
└── requirements.txt
```

Archivo conftest.py

Contiene la configuración básica para las pruebas:

```
# tests/conftest.py
import os
import sys
from pathlib import Path

# Configurar el path para importaciones
root_dir = Path(__file__).parent.parent
sys.path.insert(0, str(root_dir))
```

Este archivo asegura que las importaciones del proyecto funcionen correctamente durante las pruebas.

Estado Actual

- Archivos con pruebas implementadas: test_auth.py, test_documents.py, test_supabase.py, test_ai_connector.py
- Archivos vacíos pendientes: test_chat.py, test_user.py
- Cobertura: Parcial, enfocada en los módulos críticos

3.14.2 Pytest como framework

Instalación y Dependencias

El proyecto utiliza pytest como framework principal de testing, con las siguientes versiones instaladas:

```
pytest==7.4.3      # Framework principal
pytest-asyncio==0.21.1 # Soporte para pruebas asíncronas
```

Características de Pytest Utilizadas

1. Fixtures

Se utilizan extensivamente para configurar el entorno de pruebas:

```
@pytest.fixture
def mock_user_repository():
    """Fixture que proporciona un mock del repositorio de usuarios."""
    mock_repo = Mock()
    mock_repo.get_by_username.return_value = None
    mock_repo.create.return_value = 1
    return mock_repo
```

```
@pytest.fixture
def auth_service(mock_user_repository):
```

```
"""Fixture que proporciona el servicio de autenticación con mocks."""
service = AuthService()
service.repository = mock_user_repository
return service
```

2. Parametrización

Aunque no se ve implementada actualmente, pytest soporta pruebas parametrizadas:

```
# Ejemplo de uso potencial
@pytest.mark.parametrize("username,email,expected", [
    ("user1", "user1@test.com", True),
    ("", "invalid", False),
])
def test_user_validation(username, email, expected):
    # Lógica de prueba
```

3.14.3 Pruebas unitarias e integración

Pruebas Unitarias

El proyecto implementa pruebas unitarias con uso extensivo de mocks para aislar componentes:

1. Pruebas de Modelos

Validan la creación y validación de modelos SQLAlchemy y Pydantic:

```
class TestDocumentModels(unittest.TestCase):
    def test_document_sqlalchemy_model(self):
        """Prueba el modelo SQLAlchemy Document"""
        doc = Document(
            title="Test Document",
            uploaded_by=1,
            content_type="texto",
            chromadb_id="test_chroma_123"
        )
        self.assertEqual(doc.title, "Test Document")

    def test_document_pydantic_validations(self):
        """Prueba las validaciones de los modelos Pydantic"""
        with self.assertRaises(ValidationError):
            DocumentBase(title="", content_type="pdf") # Título vacío debe fallar
```

2. Pruebas de Repositorios

Utilizan mocks para simular la base de datos:

```
@patch('src.repositories.document_repository.get_supabase_client')
def test_create_document(self, mock_get_client):
    """Prueba la creación de un documento en el repositorio"""
    # Configurar mock de Supabase
    mock_supabase = MagicMock()
    mock_response = MagicMock()
    mock_response.data = [{"id": 1}]
    mock_supabase.table.return_value.insert.return_value.execute.return_value
    = mock_response

    # Probar el repositorio
    repo = DocumentRepository()
    doc_id = repo.create(document)
    self.assertEqual(doc_id, 1)
```

3. Pruebas de Servicios

Verifican la lógica de negocio con dependencias mockeadas:

```
class TestAuthService:
    def test_register_user_success(self, auth_service):
        """Prueba el registro exitoso de un usuario."""
        result = auth_service.register_user(TEST_USERNAME, TEST_EMAIL,
                                             TEST_PASSWORD)

        # Verificar llamadas

        auth_service.repository.get_by_username.assert_called_once_with(TEST_USER
NAME)
        auth_service.repository.create.assert_called_once()

        # Verificar resultado
        assert "access_token" in result
        assert result["token_type"] == "bearer"
```

Pruebas de Integración

El proyecto incluye scripts de diagnóstico que actúan como pruebas de integración:

1. Test de Conexión a Supabase

```
def test_supabase_connection():
    """Prueba la conexión a Supabase"""
    try:
        from src.config.database import SupabaseConnector
        connector = SupabaseConnector()
        client = connector.get_client()

        # Intentar una consulta simple
        response = client.from_('users').select('*').limit(1).execute()
        print("Conexión a Supabase exitosa!")
        return True
    except Exception as e:
        print(f"Error al probar la conexión: {e}")
        return False
```

2. Test de Conexión con IA

```
def test_openai_connection():
    """Prueba la conexión con OpenAI"""
    openai = get_openai_connector()

    # Prueba 1: Generar embeddings
    embeddings = openai.create_embeddings([test_text])

    # Prueba 2: Generar respuesta simple
    response = openai.generate_chat_completion(messages)

    # Prueba 3: Generar respuesta RAG
    rag_response = openai.generate_rag_response(query=query,
                                                context=context)
```

Estrategias de Testing

1. Uso de Mocks

- unittest.mock.Mock y MagicMock para simular objetos
- @patch decorador para reemplazar dependencias
- Configuración de comportamientos específicos con return_value y side_effect

2. Manejo de Errores

```
def test_create_document_error_rollback(self):
    """Prueba que se haga rollback si falla ChromaDB"""
    mock_chromadb.add_documents.side_effect = Exception("Error en
ChromaDB")
```

```

with self.assertRaises(Exception):
    document = service.create_document(...)

# Verificar rollback
mock_repo.delete.assert_called_once_with(1)

```

3. Pruebas de Flujo Completo

```

class TestDocumentServiceIntegration(unittest.TestCase):
    def test_search_documents(self):
        """Prueba la búsqueda de documentos end-to-end"""
        # Simular resultados de ChromaDB
        # Verificar transformación de datos
        # Validar respuesta final

```

3.15 Punto de entrada (src/main.py)

3.15.1 Configuración de FastAPI

Inicialización de la Aplicación

El punto de entrada de la aplicación se basa en FastAPI, un framework web moderno y de alto rendimiento para construir APIs con Python 3.7+ basado en estándares como OpenAPI y JSON Schema.

```

app = FastAPI(
    title="ChatBot Backend",
    description="API para el chatbot con manejo de documentos grandes",
    version="1.0.0"
)

```

Características de la Configuración

Metadatos de la API:

- **Title:** Define el nombre descriptivo que aparece en la documentación automática
- **Description:** Proporciona contexto sobre la funcionalidad principal del sistema
- **Version:** Control de versiones para la API, facilitando el mantenimiento y actualizaciones

Ventajas de FastAPI para este proyecto:

1. Documentación automática: Genera automáticamente documentación interactiva (Swagger UI/ReDoc)

2. Validación de datos: Validación automática basada en type hints de Python
3. Serialización JSON: Conversión automática entre objetos Python y JSON
4. Rendimiento: Uno de los frameworks Python más rápidos disponibles
5. Async/Await nativo: Soporte completo para programación asíncrona
6. Type hints: Aprovecha las anotaciones de tipo de Python para mejor desarrollo

Importaciones de Esquemas

```
from src.models.schemas.user import UserCreate, UserResponse
from src.models.schemas.document import DocumentBase,
DocumentResponse
from src.models.schemas.chat import ChatMessage
```

La aplicación importa los esquemas Pydantic que definen la estructura de datos para:

- User schemas: Validación de creación y respuesta de usuarios
- Document schemas: Estructura para manejo de documentos PDF
- Chat schemas: Formato de mensajes del sistema de chat

3.15.2 Middleware de CORS

Configuración de Cross-Origin Resource Sharing

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Propósito y Funcionalidad

CORS (Cross-Origin Resource Sharing) es fundamental para permitir que el frontend acceda a la API desde diferentes dominios. La configuración implementada permite:

Parámetros de configuración:

1. `allow_origins=["*"]`:
 - Permite solicitudes desde cualquier origen
 - En producción se debería restringir a dominios específicos

- Ejemplo de configuración más segura: ["http://localhost:3000", "https://mi-app.com"]
- 2. `allow_credentials=True`:
 - Permite el envío de cookies y headers de autenticación
 - Necesario para mantener sesiones de usuario
 - Esencial para el sistema de autenticación con Supabase
- 3. `allow_methods=["*"]`:
 - Permite todos los métodos HTTP (GET, POST, PUT, DELETE, etc.)
 - Facilita todas las operaciones CRUD de la API
- 4. `allow_headers=["*"]`:
 - Permite todos los headers personalizados
 - Importante para headers de autenticación (Authorization, Bearer tokens)

Consideraciones de Seguridad

Configuración actual (desarrollo):

- Permite acceso desde cualquier origen ("[*]")
- Adecuada para desarrollo y testing local

Configuración recomendada (producción):

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "http://localhost:3000", # Frontend local
        "https://mi-app.vercel.app" # Frontend desplegado
    ],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PUT", "DELETE"],
    allow_headers=["Authorization", "Content-Type"],
)
```

3.15.3 Eventos de startup

Inicialización Asíncrona

```
@app.on_event("startup")
async def startup():
    # Configuraciones para manejar archivos grandes
    uvicorn.config.LIFESPAN_ON_STARTUP = True
    uvicorn.config LOOP_WAIT = 0.1
    uvicorn.config.HTTP_TIMEOUT_KEEP_ALIVE = 120
```

```
# Inicializar ChromaDB
global chroma_db
chroma_db = ChromaDBConnector()
logging.info("ChromaDB inicializado correctamente")
```

Componentes del Startup

Configuraciones de Unicorn

LIFESPAN_ON_STARTUP = True:

- Habilita el manejo del ciclo de vida de la aplicación
- Asegura que los recursos se inicialicen correctamente al arranque

LOOP_WAIT = 0.1:

- Tiempo de espera del loop de eventos (100ms)
- Optimiza el balance entre rendimiento y uso de CPU

HTTP_TIMEOUT_KEEP_ALIVE = 120:

- Mantiene conexiones HTTP activas por 2 minutos
- Crucial para operaciones de larga duración como procesamiento de PDFs

Inicialización de ChromaDB

Propósito:

- Establece conexión con la base de datos vectorial
- Inicializa las colecciones necesarias para embeddings
- Verifica la conectividad antes de que la API esté disponible

Patrón singleton:

```
global chroma_db
chroma_db = ChromaDBConnector()
```

Este patrón asegura:

- Una única instancia de ChromaDB por aplicación
- Reutilización de la conexión en toda la aplicación
- Evita múltiples inicializaciones costosas

Beneficios del Startup Event

1. Inicialización determinista: Garantiza que todos los recursos estén listos antes de servir requests
2. Manejo de errores temprano: Falla rápido si hay problemas de conectividad
3. Optimización de rendimiento: Evita inicializaciones costosas en cada request
4. Logging centralizado: Registro de eventos de inicio para debugging

3.15.4 Configuración de servidor Unicorn

Configuración del Servidor ASGI

```
if __name__ == "__main__":
    config = uvicorn.Config(
        "src.main:app",
        host="127.0.0.1",
        port=2690,
        reload=True,
        limit_concurrency=1000,
        limit_max_requests=10000,
        timeout_keep_alive=120,
        backlog=2048,
    )

    server = uvicorn.Server(config)
    server.run()
```

Parámetros de Configuración Detallados

Configuración Básica

"src.main:app":

- Especifica la ruta del módulo y la instancia de FastAPI
- Formato: módulo:variable
- Permite hot-reloading durante desarrollo

host="127.0.0.1":

- Dirección IP de binding (localhost)
- Restringe acceso solo desde la máquina local
- En producción se usaría "0.0.0.0" para acceso externo

port=2690:

- Puerto personalizado para evitar conflictos
- Evita puertos comunes (8000, 8080)
- Permite múltiples instancias de desarrollo

`reload=True:`

- Habilita hot-reloading automático
- Reinicia el servidor cuando detecta cambios en el código
- Solo para desarrollo (se desactiva en producción)

Configuración de Rendimiento

`limit_concurrency=1000:`

- Límite de conexiones concurrentes simultáneas
- Previene sobrecarga del servidor
- Valor optimizado para operaciones de procesamiento de documentos

`limit_max_requests=10000:`

- Número máximo de requests que puede manejar un worker
- Despues de este límite, el worker se reinicia
- Previene memory leaks en operaciones de larga duración

`timeout_keep_alive=120:`

- Tiempo en segundos para mantener conexiones HTTP/1.1 activas
- Optimizado para operaciones de carga de archivos grandes
- Coincide con la configuración del startup event

`backlog=2048:`

- Tamaño de la cola de conexiones pendientes
- Buffer para picos de tráfico
- Valor alto para manejar múltiples uploads simultáneos

Justificación de la Configuración

Optimizaciones para Documentos Grandes

La configuración está específicamente optimizada para:

1. Carga de PDFs grandes: Timeouts extendidos permiten procesar documentos de varios MB

2. Múltiples usuarios simultáneos: Alta concurrencia para sesiones de chat paralelas
3. Operaciones vectoriales: Recursos suficientes para operaciones de embedding
4. Estabilidad: Límites que previenen agotamiento de memoria

Configuración de Desarrollo vs Producción

Desarrollo (actual):

```
reload=True      # Hot reloading
host="127.0.0.1"  # Solo local
```

Producción (recomendada):

```
reload=False     # Sin hot reloading
host="0.0.0.0"   # Acceso externo
workers=4        # Múltiples workers
```

Patrones de Inicialización

Separación de Configuración

```
if __name__ == "__main__":
```

Este patrón asegura que la configuración del servidor solo se ejecute cuando el archivo se ejecuta directamente, no cuando se importa como módulo.

Configuración Programática vs Comando CLI

Ventajas de la configuración programática:

- Control total sobre parámetros
- Configuración dinámica basada en variables de entorno
- Integración fácil con scripts de deployment
- Mejor para testing y desarrollo

Alternativa CLI:

```
uvicorn src.main:app --host 127.0.0.1 --port 2690 --reload
```

Consideraciones de Deployment

Variables de Entorno

Para un deployment robusto, se recomienda:

```
import os

config = uvicorn.Config(
    "src.main:app",
    host=os.getenv("HOST", "127.0.0.1"),
    port=int(os.getenv("PORT", 2690)),
    reload=os.getenv("ENVIRONMENT") == "development",
    # ... otros parámetros
)
Logging y Monitoreo
import logging

logging.basicConfig(
    level=logging.INFO,
    format"%(asctime)s - %(name)s - %(levelname)s - %(message)s"
)
```

Esta configuración del punto de entrada establece una base sólida para una aplicación FastAPI escalable, con consideraciones específicas para el manejo de documentos grandes y operaciones vectoriales, mientras mantiene flexibilidad para diferentes entornos de deployment.

4. Integración Frontend-Backend

4.1 Comunicación API REST

Arquitectura General

El backend está construido con FastAPI y expone una API REST completa en el puerto 2690. La comunicación sigue el patrón RESTful estándar:

Frontend (React) <-> API REST (FastAPI) <-> Backend Services <-> Base de Datos (Supabase)

Endpoints Principales

Autenticación y Usuarios (/api/users)

- POST /api/users/login - Autenticación de usuarios
- POST /api/users/register - Registro de nuevos usuarios
- GET /api/users/me - Obtener usuario actual
- PUT /api/users/me - Actualizar perfil
- POST /api/users/logout - Cerrar sesión
- POST /api/users/refresh-token - Renovar token de acceso

Gestión de Chats (/api/chats)

- POST /api/chats/ - Crear nuevo chat
- GET /api/chats/ - Listar chats del usuario
- GET /api/chats/{chat_id} - Obtener chat específico
- PUT /api/chats/{chat_id}/rename - Renombrar chat
- DELETE /api/chats/{chat_id} - Eliminar chat
- POST /api/chats/{chat_id}/messages - Enviar mensaje al chatbot

Gestión de Documentos (/api/documents)

- POST /api/documents/upload - Subir archivo (procesamiento híbrido)
- GET /api/documents/ - Listar documentos del usuario
- GET /api/documents/{id} - Obtener documento específico
- PUT /api/documents/{id} - Actualizar metadata
- DELETE /api/documents/{id} - Eliminar documento
- POST /api/documents/{id}/share - Compartir documento
- GET /api/documents/{id}/status - Estado de procesamiento

Configuración CORS

El backend tiene CORS completamente habilitado para desarrollo:

```
app.add_middleware(  
    CORSMiddleware,  
    allow_origins=["*"], # En producción especificar dominios  
    allow_credentials=True,  
    allow_methods=["*"],  
    allow_headers=["*"],  
)
```

Formato de Respuestas

Todas las respuestas siguen el formato JSON con esquemas Pydantic definidos:

```
# Respuesta exitosa
```

```

{
  "id": 1,
  "data": {...},
  "message": "Operación exitosa"
}

# Respuesta de error
{
  "detail": "Mensaje de error descriptivo"
}

```

4.2 Manejo de Autenticación

Sistema JWT

El backend implementa autenticación mediante JWT (JSON Web Tokens) con las siguientes características:

Configuración de Tokens

```
JWT_ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 57600 # 40 días
```

Flujo de Autenticación

1. Login
2. // Frontend
3. const response = await fetch('/api/users/login', {
4. method: 'POST',
5. headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
6. body: new URLSearchParams({
7. username: 'usuario@email.com',
8. password: 'contraseña'
9. })
10. });
- 11.
- 12. const data = await response.json();
- 13. // Respuesta incluye:
- 14. // - access_token
- 15. // - refresh_token
- 16. // - user_id
- 17. // - is_admin
- 18. Almacenamiento de Tokens
 - o El frontend debe almacenar los tokens de forma segura
 - o Opciones: localStorage, sessionStorage, o cookies httpOnly
- 19. Uso del Token en Peticiones
- 20. // Incluir en headers

```
21. headers: {  
22.   'Authorization': `Bearer ${accessToken}`  
23. }  
24.  
25. // O como query parameter  
26. fetch(`/api/endpoint?token=${accessToken}`)
```

Protección de Rutas

El backend usa el decorador `Depends(get_current_user)` para proteger endpoints:

```
@router.get("/protected")  
async def protected_route(current_user: User = Depends(get_current_user)):  
    # Solo usuarios autenticados pueden acceder  
    return {"user": current_user.username}
```

Manejo de Roles

- Usuario Regular: Acceso a sus propios recursos
- Administrador (`is_admin=True`): Acceso completo al sistema
- Usuario especial "Ivan" con contraseña "ivan1234" tiene privilegios de admin

Refresh Token

Para renovar tokens expirados:

```
const response = await fetch('/api/users/refresh-token', {  
  method: 'POST',  
  headers: { 'Content-Type': 'application/json' },  
  body: JSON.stringify({  
    refresh_token: storedRefreshToken  
  })  
});
```

4.3 Gestión de Estados

Estados en el Backend

Estado de Documentos

Los documentos tienen estados de procesamiento:

- pending - En espera de procesamiento
- processing - Procesándose actualmente

- completed - Procesamiento completado
- error - Error en el procesamiento

Estado de Sesión

El backend mantiene el estado de sesión mediante:

- Tokens JWT con información del usuario
- Refresh tokens almacenados en la base de datos
- Última actividad registrada

Recomendaciones para el Frontend

Estado Global de Autenticación

```
// Contexto de autenticación sugerido
const AuthContext = {
  user: null,
  accessToken: null,
  refreshToken: null,
  isAuthenticated: false,
  isAdmin: false
};
```

Estado de Documentos

```
// Estado para gestión de documentos
const DocumentsState = {
  documents: [],
  selectedDocument: null,
  uploadProgress: {},
  processingStatus: {}
};
```

Estado de Chat

```
// Estado para conversaciones
const ChatState = {
  chats: [],
  currentChat: null,
  messages: [],
  isLoading: false,
  selectedDocuments: [] // Para RAG
};
```

Sincronización de Estados

Polling para Estado de Documentos

Para documentos en procesamiento, implementar polling:

```
const checkDocumentStatus = async (documentId) => {
  const interval = setInterval(async () => {
    const response = await fetch(`/api/documents/${documentId}/status`);
    const status = await response.json();

    if (status.completed || status.status === 'error') {
      clearInterval(interval);
      // Actualizar estado global
    }
  }, 3000); // Cada 3 segundos
};
```

4.4 Optimización de Peticiones

Procesamiento Híbrido de Documentos

El backend implementa un sistema inteligente basado en el tamaño:

- Archivos pequeños (< 1MB PDF, < 500KB texto): Procesamiento síncrono
- Archivos grandes: Procesamiento asíncrono en background

```
# Umbrales definidos
TEXT_THRESHOLD = 500 * 1024    # 500KB para texto
PDF_THRESHOLD = 1 * 1024 * 1024 # 1MB para PDFs
GENERAL_THRESHOLD = 3 * 1024 * 1024 # 3MB general
```

Paginación

Todos los endpoints de listado soportan paginación:

```
// Ejemplo de petición paginada
const response = await
fetch('/api/documents?skip=0&limit=20&sort_by=created_at&order=desc');
```

Búsqueda Optimizada con RAG

El sistema de chat permite especificar documentos para búsqueda:

```
// Búsqueda en documentos específicos
const message = {
  question: "¿Cuál es el resumen del documento?",  

  document_ids: [1, 2, 3], // Solo buscar en estos documentos
```

```
n_results: 5 // Número de resultados relevantes  
};
```

Caché y Optimizaciones

Configuración del Servidor

```
config = uvicorn.Config(  
    limit_concurrency=1000, # Conexiones concurrentes  
    limit_max_requests=10000, # Peticiones máximas  
    timeout_keep_alive=120, # Keep-alive timeout  
    backlog=2048 # Cola de conexiones  
)
```

Timeouts Configurados

- ChromaDB: 300 segundos
- Procesamiento de documentos: 180 segundos
- Keep-alive HTTP: 120 segundos

Manejo de Errores y Reintentos

Patrón de Reintentos Sugerido

```
const fetchWithRetry = async (url, options, maxRetries = 3) => {  
  for (let i = 0; i < maxRetries; i++) {  
    try {  
      const response = await fetch(url, options);  
      if (response.ok) return response;  
  
      // Si es 401, intentar refresh token  
      if (response.status === 401 && i < maxRetries - 1) {  
        await refreshAccessToken();  
        options.headers.Authorization = `Bearer ${newAccessToken}`;  
        continue;  
      }  
  
      throw new Error(`HTTP ${response.status}`);  
    } catch (error) {  
      if (i === maxRetries - 1) throw error;  
      await new Promise(resolve => setTimeout(resolve, 1000 * (i + 1)));  
    }  
  }  
};
```

Optimización de Carga de Archivos

Upload con Progress Tracking

```
const uploadFile = async (file, onProgress) => {
  const formData = new FormData();
  formData.append('file', file);
  formData.append('title', file.name);

  const xhr = new XMLHttpRequest();

  xhr.upload.addEventListener('progress', (e) => {
    if (e.lengthComputable) {
      const percentComplete = (e.loaded / e.total) * 100;
      onProgress(percentComplete);
    }
  });

  return new Promise((resolve, reject) => {
    xhr.onload = () => resolve(JSON.parse(xhr.responseText));
    xhr.onerror = reject;
    xhr.open('POST', '/api/documents/upload');
    xhr.setRequestHeader('Authorization', `Bearer ${accessToken}`);
    xhr.send(formData);
  });
};
```

Notas Importantes

Estado Actual del Desarrollo

- El backend está funcional pero en desarrollo activo
- Algunos endpoints pueden estar incompletos o en proceso de mejora
- El sistema de procesamiento de documentos grandes está optimizado para manejar archivos de hasta 100MB

Consideraciones de Seguridad

- En producción, cambiar `allow_origins=["*"]` a dominios específicos
- Implementar rate limiting para prevenir abuso
- Considerar usar cookies `httpOnly` para tokens en producción
- Validar y sanitizar todas las entradas del usuario

5. Consideraciones Futuras

5.1 Optimizaciones pendientes

Rendimiento del sistema:

- Implementación de caché inteligente para consultas frecuentes en Redis
- Optimización de queries en ChromaDB mediante índices especializados
- Paralelización avanzada del procesamiento de documentos con Celery
- Compresión de embeddings para reducir el uso de memoria en ChromaDB

Gestión de memoria:

- Streaming de contenido para documentos extremadamente grandes (>100MB)
- Garbage collection optimizado para liberación proactiva de memoria
- Técnicas de lazy loading para metadatos de documentos
- Implementación de pools de conexiones optimizados para Supabase

Algoritmos de búsqueda:

- Mejoras en el algoritmo de chunking con detección de estructura semántica
- Implementación de re-ranking de resultados basado en relevancia contextual
- Optimización de embeddings con modelos específicos por dominio

5.2 Nuevas funcionalidades

Capacidades de IA expandidas:

- Análisis de sentimientos en documentos y conversaciones
- Resumen automático de documentos extensos
- Detección de entidades (nombres, fechas, organizaciones)
- Traducción automática de documentos en múltiples idiomas
- Generación de metadatos automática basada en contenido

Características colaborativas:

- Anotaciones compartidas en documentos
- Historial de cambios con versionado automático
- Notificaciones en tiempo real para actualizaciones de documentos
- Espacios de trabajo colaborativos por equipos
- Comentarios y discusiones contextuales en documentos

Integraciones externas:

- Conectores para SharePoint, Google Drive, OneDrive

- Integración con sistemas ERP/CRM existentes
- APIs para integración con software de terceros
- Webhooks para automatización de flujos de trabajo

Analíticas avanzadas:

- Dashboard de métricas de uso y rendimiento
- Análisis de patrones de consulta por usuario
- Reportes de accesibilidad de documentos
- Métricas de efectividad del sistema RAG

5.3 Mejoras de rendimiento

Optimización de base de datos:

- Implementación de sharding en ChromaDB para documentos masivos
- Índices compuestos optimizados en Supabase
- Caché distribuido con Redis Cluster
- Agregaciones precalculadas para consultas complejas

Escalabilidad horizontal:

- Microservicios especializados por funcionalidad
- Load balancing inteligente con detección de carga
- Auto-scaling basado en métricas de uso
- CDN global para archivos estáticos

Optimización de red:

- Compresión de payloads en APIs REST
- Implementación de GraphQL para consultas eficientes
- WebSocket para actualizaciones en tiempo real
- Progressive Web App (PWA) para mejor rendimiento web

Monitoreo y observabilidad:

- Tracing distribuido con OpenTelemetry
- Métricas personalizadas con Prometheus
- Alertas inteligentes basadas en ML
- Dashboards en tiempo real con Grafana

6. Bibliografía y Referencias Técnicas

6.1 Documentación de frameworks y tecnologías

Backend - Python/FastAPI:

- FastAPI Documentation. (2024). "FastAPI - Modern, fast (high-performance), web framework for building APIs with Python 3.7+". <https://fastapi.tiangolo.com/>
- Pydantic Documentation. (2024). "Data validation and settings management using python type annotations". <https://docs.pydantic.dev/>
- SQLAlchemy Documentation. (2024). "The Python SQL Toolkit and Object Relational Mapper". <https://docs.sqlalchemy.org/>

Frontend - Flutter:

- Flutter Documentation. (2024). "Build apps for any screen: Flutter SDK". <https://docs.flutter.dev/>
- Material Design 3. (2024). "Material You - Material Design". <https://m3.material.io/>
- Dart Language Documentation. (2024). "Dart programming language". <https://dart.dev/guides>

Bases de datos:

- ChromaDB Documentation. (2024). "The open-source embedding database". <https://docs.trychroma.com/>
- Supabase Documentation. (2024). "The open source Firebase alternative". <https://supabase.com/docs>

6.2 Literatura sobre RAG y procesamiento de documentos

Retrieval-Augmented Generation:

- Lewis, P., et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". *Proceedings of NeurIPS 2020*.
- Karpukhin, V., et al. (2020). "Dense Passage Retrieval for Open-Domain Question Answering". *Proceedings of EMNLP 2020*.

Vector Embeddings y Búsqueda Semántica:

- Reimers, N., & Gurevych, I. (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". *Proceedings of EMNLP-IJCNLP 2019*.
- Johnson, J., Douze, M., & Jégou, H. (2019). "Billion-scale similarity search with GPUs". *IEEE Transactions on Big Data*.

Procesamiento de Documentos:

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). "Introduction to Information Retrieval". Cambridge University Press.
- Devlin, J., et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT 2019*.

6.3 Arquitectura de software y patrones de diseño

Clean Architecture:

- Martin, R. C. (2017). "Clean Architecture: A Craftsman's Guide to Software Structure and Design". Prentice Hall.
- Fowler, M. (2002). "Patterns of Enterprise Application Architecture". Addison-Wesley.

Microservicios y APIs:

- Newman, S. (2021). "Building Microservices: Designing Fine-Grained Systems". O'Reilly Media.
- Richardson, C. (2018). "Microservices Patterns: With examples in Java". Manning Publications.

6.4 Seguridad y autenticación

JWT y OAuth:

- RFC 7519. (2015). "JSON Web Token (JWT)". IETF.
<https://tools.ietf.org/html/rfc7519>
- RFC 6749. (2012). "The OAuth 2.0 Authorization Framework". IETF.
<https://tools.ietf.org/html/rfc6749>

Seguridad en APIs:

- OWASP API Security Project. (2023). "OWASP API Security Top 10".
<https://owasp.org/www-project-api-security/>
- OpenAI API Documentation. (2024). <https://platform.openai.com/docs>
- Google AI Gemini Documentation. (2024). <https://ai.google.dev/docs>

6.5 Herramientas de desarrollo:

- Docker Documentation. (2024). "Containerization Platform".
<https://docs.docker.com/>
- Git Documentation. (2024). "Distributed Version Control System".
<https://git-scm.com/doc>