Instituto Superior Tecnológico Sudamericano

INSTITUTO TECNOLÓGICO
SUDAMERICANO
¡Hacemos gente de talento!

DESARROLLO DE SOFTWARE
TECNOLOGÍA SUPERIOR

IMPLEMENTACIÓN DE UN ASISTENTE VIRTUAL BASADO EN RAG Y LLM COMO SOLUCIÓN DE NEGOCIO PARA LA ATENCIÓN AUTOMATIZADA DE SERVICIOS EDUCATIVOS

MANUAL DE PROGRAMADOR

Autor:

Pablo Moisés Cuenca

Director:

Ing. Cristhian Javier Villamarin Gaona

Loja - Ecuador
2025

Índices

1 Introducción

1.1 Bienvenida

Bienvenido al **Manual del Programador del Sistema de Chatbot ISTS**. Este documento proporciona documentación técnica completa para desarrolladores que trabajarán con el backend API del chatbot institucional.

El sistema está construido con **Go 1.25.1** utilizando arquitectura limpia (Clean Architecture), y proporciona un API REST robusto para gestionar un chatbot inteligente de WhatsApp con capacidades RAG (Retrieval Augmented Generation).

1.2 Objetivos del Manual

Este manual tiene los siguientes objetivos:

- Explicar la arquitectura del sistema y las decisiones de diseño
- Documentar el stack tecnológico y las dependencias
- Proporcionar guías para configurar el entorno de desarrollo
- Describir los patrones de código y convenciones utilizadas
- Explicar cómo extender y mantener el sistema

1.3 A Quién Está Dirigido

Este manual está dirigido a:

- **Desarrolladores backend** que trabajarán con el código Go
- **Arquitectos de software** evaluando o diseñando el sistema
- **DevOps engineers** encargados del despliegue y mantenimiento
- **Estudiantes** aprendiendo sobre a programar

1.4 Prerrequisitos

Para trabajar con este proyecto, debe tener conocimientos de:

- **Lenguaje Go** (nivel intermedio-avanzado)
- **PostgreSQL** y bases de datos relacionales
- **Arquitectura REST** y principios HTTP
- **Git** para control de versiones
- **Docker** para containerización (recomendado)
- **Conceptos de IA/ML** (embedding, RAG, LLMs) - básico

1.5 Estructura del Manual

El manual está organizado en las siguientes secciones:

1. **Visión General del Sistema**: Arquitectura y componentes principales
2. **Entorno de Desarrollo**: Instalación y configuración
3. **Arquitectura Detallada**: Capas, patrones y flujo de datos
4. **Módulos del Sistema**: Documentación de cada módulo funcional

5. **Base de Datos**: Schema y procedimientos almacenados
6. **API REST**: Endpoints y especificaciones
7. **Extensión y Mantenimiento**: Cómo agregar nuevas funcionalidades
8. **Despliegue**: Configuración de producción
9. **Referencia**: Apéndices y recursos adicionales

2 Visión General del Sistema

2.1 Arquitectura de Alto Nivel

El sistema APIGO Chatbot es una **aplicación backend monolítica** construida con Go que proporciona un API REST para gestionar un chatbot institucional de WhatsApp con capacidades de procesamiento de lenguaje natural e inteligencia artificial.

2.1.1 Componentes Principales

El sistema consta de los siguientes componentes:

1. **API REST Backend** (Go + Huma)
   - Gestión de parámetros dinámicos
   - CRUD de documentos y conocimiento
   - Endpoints de administración
   - Autenticación JWT
2. **Motor RAG** (Retrieval Augmented Generation)
   - Búsqueda semántica con embeddings vectoriales
   - Generación de respuestas con LLM (Groq/OpenAI)
   - Cache de chunks de conocimiento
3. **Integración WhatsApp**
   - Cliente WhatsApp (whatsmeow)
   - Gestión de sesiones y QR codes
   - Manejo de mensajes entrantes/salientes
   - Dispatchers por tipo de mensaje
4. **Base de Datos PostgreSQL**
   - Almacenamiento de datos estructurados
   - Extensión pgvector para búsquedas vectoriales
   - Stored procedures para lógica de negocio
   - Cache de parámetros en memoria
5. **Servicios Internos**
   - Cache en memoria (thread-safe)
   - Cliente HTTP para APIs externas
   - Generación de embeddings
   - Gestión de tokens JWT

- Sistema de logging estructurado

2.2 Stack Tecnológico

2.2.1 Backend
- **Go 1.25.1**: Lenguaje principal
- **Huma v2**: Framework REST con OpenAPI 3.1
- **Chi Router**: Router HTTP subyacente
- **pgx/v5**: Driver PostgreSQL de alto rendimiento
- **pgvector-go**: Operaciones vectoriales

2.2.2 Base de Datos
- **PostgreSQL 15+**: Base de datos principal
- **pgvector**: Extensión para embeddings vectoriales
- **uuid-ossp**: Generación de UUIDs
- **pgcrypto**: Funciones criptográficas

2.2.3 Integraciones
- **whatsmeow**: Cliente WhatsApp no oficial
- **Groq/OpenAI API**: Servicios LLM
- **OpenAI Embeddings**: text-embedding-3-small
- **Ollama** (opcional): Embeddings locales

2.2.4 DevOps
- **Docker**: Containerización
- **Git**: Control de versiones
- **Viper**: Gestión de configuración
- **Lumberjack**: Rotación de logs

2.3 Principios de Diseño
El sistema sigue estos principios arquitectónicos:

2.3.1 Clean Architecture
Separación estricta en capas:
- **Domain**: Entidades y contratos (interfaces)
- **Repository**: Acceso a datos
- **Use Case**: Lógica de negocio
- **API**: Presentación y routing

2.3.2 Database-First
- Toda la lógica de datos está en PostgreSQL
- Uso de stored procedures y funciones
- El código Go llama funciones DB por nombre e integra la lógica en el servicio

- Zero SQL raw en el código Go

2.3.3 Interface-Based Design
- Dependencias invertidas mediante interfaces
- Facilita testing y mocking
- Desacoplamiento entre capas

2.3.4 Thread-Safety
- Cache con sync.RWMutex
- Connection pooling (pgxpool)
- Goroutines seguras para WhatsApp handlers

2.4 Estado del Proyecto
Versión actual: 1.0 MVP

Módulos implementados:
- Sistema de parámetros
- Gestión de documentos
- Motor RAG (chunks + búsqueda)
- Integración WhatsApp
- Panel de conversaciones admin
- Autenticación JWT
- Panel de analytics
- Frontend admin web
- Logging estructurado

3 Entorno de Desarrollo

3.1 Requisitos del Sistema

3.1.1 Hardware Mínimo

- **CPU**: 2 cores
- **RAM**: 4 GB
- **Disco**: 2 GB libres

3.1.2 Hardware Recomendado

- **CPU**: 4+ cores
- **RAM**: 8+ GB
- **Disco**: 10+ GB (para logs y embeddings)

3.1.3 Software Requerido

- **Go**: 1.25.1 o superior
- **PostgreSQL**: 15 o superior
- **Git**: 2.30+
- **Docker**: 20.10+ (opcional pero recomendado)
- **Make**: Para comandos de desarrollo (opcional)

3.2 Instalación del Entorno

3.2.1 Instalación de Go

Linux:

```
wget https://go.dev/dl/go1.25.1.linux-amd64.tar.gz
sudo rm -rf /usr/local/go
sudo tar -C /usr/local -xzf go1.25.1.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
```

macOS:

```
brew install go@1.25
```

Verificación:

```
go version
# Salida esperada: go version go1.25.1 linux/amd64
```

3.2.2 Instalación de PostgreSQL

Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install postgresql-15 postgresql-contrib
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

macOS:

```
brew install postgresql@15
brew services start postgresql@15
```

Verificación:

```
psql --version
# Salida esperada: psql (PostgreSQL) 15.x
```

3.2.3 Instalación de pgvector

```
# Clonar repositorio
cd /tmp
git clone https://github.com/pgvector/pgvector.git
cd pgvector

# Compilar e instalar
make
sudo make install
```

3.3 Configuración del Proyecto

3.3.1 Clonar el Repositorio

```
git clone <repository-url>
cd apigo-chatbot
```

3.3.2 Instalar Dependencias

```
go mod download
```

3.3.3 Configurar Base de Datos

1. **Crear usuario y base de datos:**

```
sudo -u postgres psql
```

```
CREATE USER chatbot_user WITH PASSWORD 'your_password';
CREATE DATABASE chatbot_db OWNER chatbot_user;
\c chatbot_db
CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS pgcrypto;
\q
```

2. **Ejecutar migraciones:**

```
# Orden de ejecución
PGPASSWORD='your_password' psql -h localhost -U chatbot_user -d chatbot_db -
f db/00_database_setup.sql
PGPASSWORD='your_password' psql -h localhost -U chatbot_user -d chatbot_db -
f db/01_create_tables.sql
```

```
PGPASSWORD='your_password' psql -h localhost -U chatbot_user -d chatbot_db -
f db/02_parameters_procedures.sql
PGPASSWORD='your_password' psql -h localhost -U chatbot_user -d chatbot_db -
f db/03_knowledge_procedures.sql
PGPASSWORD='your_password' psql -h localhost -U chatbot_user -d chatbot_db -
f db/04_conversation_procedures.sql
# ... continuar con todos los archivos en db/
PGPASSWORD='your_password' psql -h localhost -U chatbot_user -d chatbot_db -
f db/initial_data.sql
```

3.3.4 Configurar config.json

Crear archivo `config.json` en la raíz del proyecto:

```
{
  "App": {
    "AppName": "APIGO Chatbot",
    "Env": "development",
    "Port": "8080",
    "Timeout": 30
  },
  "Database": {
    "Host": "localhost",
    "Port": "5432",
    "User": "chatbot_user",
    "Password": "your_password",
    "Name": "chatbot_db",
    "MaxConnections": 10,
  }
}
```

3.4 Compilar y Ejecutar

3.4.1 Desarrollo Local

```
# Ejecutar directamente con Go
go run cmd/main.go

# Output esperado:
# Starting APIGO Chatbot v1.0
# Server listening on :8080
# OpenAPI docs: http://localhost:8080/docs
```

3.4.2 Compilar Binario

```
# Build optimizado
go build -o main cmd/main.go
```

```
# Ejecutar
./main
```

3.4.3 Con Docker

```
# Build image
docker build -t apigo-chatbot:latest .

# Run container
docker run -d \
  -p 3434:8080 \
  -v /path/to/config.json:/config/chatbot/config.json \
  --name chatbot \
  apigo-chatbot:latest
```

3.5 Verificar Instalación

3.5.1 Health Check

```
curl http://localhost:8080/health
```

Respuesta esperada:

```
{
  "success": true,
  "code": "OK",
  "info": "Operación exitosa",
  "data": {}
}
```

3.5.2 OpenAPI Docs

Abrir navegador en: `http://localhost:8080/docs`

Debería ver la interfaz interactiva de Huma con todos los endpoints documentados.

3.5.3 Test de Endpoint

```
curl -X POST http://localhost:8080/api/parameters/get-all \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "test",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "test",
    "idDevice": "test",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z"
  }'
```

Respuesta esperada:

```
{
  "success": true,
  "code": "OK",
  "info": "Operación exitosa",
  "data": [ /* array de parámetros */ ]
}
```

4 Arquitectura Detallada

4.1 Patrón Clean Architecture

El sistema sigue estrictamente el patrón Clean Architecture (Arquitectura Limpia) de Robert C. Martin, con separación clara de responsabilidades en capas concéntricas.

4.1.1 Principios Fundamentales

1. **Independencia de Frameworks**: El core del negocio no depende de frameworks
2. **Testabilidad**: Lógica de negocio fácilmente testeable
3. **Independencia de UI**: API puede cambiar sin afectar lógica
4. **Independencia de Base de Datos**: Puede cambiar DB sin afectar reglas de negocio
5. **Independencia de Agentes Externos**: Lógica aislada de servicios externos

4.1.2 Regla de Dependencias

Las dependencias fluyen **hacia adentro**:

```
API Layer → UseCase Layer → Repository Layer → Domain Layer
                                                      ↑
                                                (Solo interfaces)
```

- Las capas externas dependen de las internas
- Las capas internas NO conocen las externas
- La comunicación se hace mediante interfaces

4.2 Estructura de Directorios

```
apigo-chatbot/
├── cmd/
│   └── main.go                # Entry point
├── api/
│   ├── route/                 # Definiciones de rutas Huma
│   │   ├── route.go           # Router principal
│   │   ├── parameter_router.go
│   │   ├── document_router.go
│   │   ├── chunk_router.go
│   │   ├── admin_auth_router.go
│   │   ├── admin_conversation_router.go
│   │   └── whatsapp_admin_router.go
│   ├── request/               # DTOs de entrada
│   │   ├── parameter.go
│   │   ├── document.go
│   │   ├── chunk.go
│   │   ├── admin_auth.go
│   │   └── admin_conversation.go
│   ├── middleware/            # HTTP middleware
│   │   ├── logger.go
│   │   ├── cors.go
```

```
│   │   ├── auth.go
│   │   └── jwt_auth_middleware.go
│   ├── dal/                    # Data Access Layer
│   │   └── postgres.go
│   └── common/                 # Tipos compartidos
├── domain/                     # Capa de dominio
│   ├── base.go                 # BaseRequest
│   ├── result.go               # Result[T]
│   ├── parameter.go            # Entities + Interfaces
│   ├── document.go
│   ├── chunk.go
│   ├── conversation.go
│   ├── admin.go
│   ├── whatsapp.go
│   ├── embedding.go
│   └── httpclient.go
├── repository/                 # Implementaciones Repository
│   ├── parameter_repository.go
│   ├── document_repository.go
│   ├── chunk_repository.go
│   ├── conversation_repository.go
│   ├── admin_repository.go
│   ├── whatsapp_session_repository.go
│   └── whatsapp_user_repository.go
├── usecase/                    # Lógica de negocio
│   ├── parameter_usecase.go
│   ├── document_usecase.go
│   ├── chunk_usecase.go
│   ├── conversation_usecase.go
│   ├── admin_usecase.go
│   └── whatsapp_usecase.go
├── internal/                   # Servicios internos
│   ├── cache/                  # Cache implementations
│   │   └── parameter_cache.go
│   ├── embedding/              # Embedding service
│   │   ├── openai.go
│   │   └── ollama.go
│   ├── httpclient/             # HTTP client
│   │   └── client.go
│   ├── jwttoken/               # JWT operations
│   │   └── jwt.go
│   ├── llm/                    # LLM service
│   │   └── groq.go
│   └── helper/                 # Utilidades
├── whatsapp/                   # WhatsApp integration
│   ├── client.go               # Cliente principal
```

```
│   ├── handler.go              # Message dispatcher
│   └── handlers/               # Message handlers
│       ├── rag_handler.go
│       ├── command_handler.go
│       ├── auth_handler.go
│       └── fallback_handler.go
├── config/                     # Configuración
│   └── config.go
├── db/                         # Migraciones y procedures
│   ├── 00_database_setup.sql
│   ├── 01_create_tables.sql
│   ├── 02_parameters_procedures.sql
│   ├── 03_knowledge_procedures.sql
│   ├── 04_conversation_procedures.sql
│   ├── 05_admin_procedures.sql
│   └── initial_data.sql
├── docs/                       # Documentación
├── config.json                 # Archivo de configuración
├── go.mod                      # Go modules
├── go.sum
├── Dockerfile
└── deploy.sh
```

4.3 Capa de Dominio (domain/)

La capa de dominio es el **corazón** del sistema. Contiene:

- **Entidades**: Estructuras de datos del negocio
- **Interfaces**: Contratos que otras capas deben implementar
- **Tipos de valor**: Result[T], errores de dominio

4.3.1 Características Clave

- **Zero dependencias externas**: Solo Go estándar
- **Definición de contratos**: Todas las interfaces aquí
- **Sin implementaciones**: Solo definiciones
- **Reutilizable**: Puede usarse en cualquier capa

4.3.2 Ejemplo: domain/parameter.go

```go
package domain

// Entity
type Parameter struct {
    ID          int64           `json:"id"`
    Name        string          `json:"name"`
    Code        string          `json:"code"`
    Data        json.RawMessage `json:"data"`
```

```
    Description string          `json:"description"`
    Active      bool            `json:"active"`
    CreatedAt   time.Time       `json:"createdAt"`
    UpdatedAt   time.Time       `json:"updatedAt"`
}

// Repository Interface (contrato)
type ParameterRepository interface {
    GetAll(ctx context.Context) Result[[]Parameter]
    GetByCode(ctx context.Context, code string) Result[Parameter]
    Add(ctx context.Context, name, code string, data json.RawMessage, desc
string) Result[string]
    Update(ctx context.Context, code, name string, data json.RawMessage,
desc string) Result[string]
    Delete(ctx context.Context, code string) Result[string]
}

// UseCase Interface (contrato)
type ParameterUseCase interface {
    GetAll(ctx context.Context) Result[[]Parameter]
    GetByCode(ctx context.Context, code string) Result[Parameter]
    Add(ctx context.Context, name, code string, data json.RawMessage, desc
string) Result[string]
    Update(ctx context.Context, code, name string, data json.RawMessage,
desc string) Result[string]
    Delete(ctx context.Context, code string) Result[string]
    ReloadCache(ctx context.Context) Result[string]
}

// Cache Interface (contrato)
type ParameterCache interface {
    Get(code string) (Parameter, bool)
    GetAll() []Parameter
    Set(code string, param Parameter)
    Delete(code string)
    Clear()
    LoadAll(params []Parameter)
}
```

4.4 Capa de Repositorio (repository/)

Implementa las interfaces `Repository` definidas en domain. Responsable de:

- Acceso a base de datos vía DAL
- Mapeo de resultados DB a entidades domain
- Manejo de errores de datos

4.4.1 Patrón Repository

Abstrae el acceso a datos, permitiendo:

- Cambiar implementación sin afectar use cases
- Mockear fácilmente en tests
- Centralizar lógica de acceso a datos

4.4.2 Ejemplo: repository/parameter_repository.go

```go
package repository

type parameterRepository struct {
    dal domain.DAL
}

func NewParameterRepository(dal domain.DAL) domain.ParameterRepository {
    return &parameterRepository{dal: dal}
}

func (r *parameterRepository) GetAll(ctx context.Context)
domain.Result[[]domain.Parameter] {
    params, err := dal.QueryRows[domain.Parameter](
        r.dal,
        ctx,
        "fn_get_all_parameters", // PostgreSQL function name
    )

    if err != nil {
        return domain.Result[[]domain.Parameter]{
            Success: false,
            Code:    "ERR_DB_QUERY",
            Message: err.Error(),
        }
    }

    return domain.Result[[]domain.Parameter]{
        Success: true,
        Code:    "OK",
        Data:    params,
    }
}

func (r *parameterRepository) GetByCode(ctx context.Context, code string)
domain.Result[domain.Parameter] {
    param, err := dal.QueryRow[domain.Parameter](
        r.dal,
        ctx,
```

```
        "fn_get_parameter_by_code",
        code,
    )

    if err != nil {
        return domain.Result[domain.Parameter]{
            Success: false,
            Code:    "ERR_PARAM_NOT_FOUND",
            Message: err.Error(),
        }
    }

    return domain.Result[domain.Parameter]{
        Success: true,
        Code:    "OK",
        Data:    param,
    }
}
```

4.5 Capa de Use Case (usecase/)

Contiene la **lógica de negocio**. Orquesta:

- Llamadas a repositorios
- Operaciones de cache
- Validaciones de negocio
- Composición de operaciones

4.5.1 Responsabilidades

- Implementar reglas de negocio
- Coordinar múltiples repositorios si es necesario
- Gestionar transacciones lógicas
- Aplicar timeouts y contextos

4.5.2 Ejemplo: usecase/parameter_usecase.go

```
package usecase

type parameterUseCase struct {
    repo  domain.ParameterRepository
    cache domain.ParameterCache
}

func NewParameterUseCase(
    repo domain.ParameterRepository,
    cache domain.ParameterCache,
) domain.ParameterUseCase {
```

```
    return &parameterUseCase{
        repo:  repo,
        cache: cache,
    }
}

func (uc *parameterUseCase) GetAll(ctx context.Context)
domain.Result[[]domain.Parameter] {
    // 1. Intentar obtener de cache
    cached := uc.cache.GetAll()
    if len(cached) > 0 {
        return domain.Result[[]domain.Parameter]{
            Success: true,
            Code:    "OK",
            Data:    cached,
        }
    }

    // 2. Si no hay en cache, obtener de DB
    result := uc.repo.GetAll(ctx)
    if !result.Success {
        return result
    }

    // 3. Actualizar cache
    for _, param := range result.Data {
        uc.cache.Set(param.Code, param)
    }

    return result
}

func (uc *parameterUseCase) Add(ctx context.Context, name, code string, data
json.RawMessage, desc string) domain.Result[string] {
    // 1. Agregar a DB
    result := uc.repo.Add(ctx, name, code, data, desc)
    if !result.Success {
        return result
    }

    // 2. Invalidar cache para forzar recarga
    uc.cache.Clear()

    return result
}
```

```go
func (uc *parameterUseCase) ReloadCache(ctx context.Context)
domain.Result[string] {
    // 1. Limpiar cache actual
    uc.cache.Clear()

    // 2. Obtener todos los parámetros de DB
    result := uc.repo.GetAll(ctx)
    if !result.Success {
        return domain.Result[string]{
            Success: false,
            Code:    result.Code,
            Message: result.Message,
        }
    }

    // 3. Cargar en cache
    uc.cache.LoadAll(result.Data)

    return domain.Result[string]{
        Success: true,
        Code:    "OK",
        Message: "Cache reloaded successfully",
    }
}
```

4.6 Capa de API (api/)

Capa de presentación que expone funcionalidad vía HTTP REST.

4.6.1 Componentes

1. **api/route/**: Definiciones de endpoints Huma
2. **api/request/**: DTOs de entrada con validación
3. **api/middleware/**: Middlewares HTTP
4. **api/dal/**: Data Access Layer abstraction

4.6.2 Huma Framework

Utiliza Huma v2 para:

- Auto-generación de OpenAPI 3.1
- Validación automática de requests
- Serialización/deserialización JSON
- Documentación interactiva

4.6.3 Ejemplo: api/route/parameter_router.go

```go
package route

func RegisterParameterRoutes(apiAPI huma.API, paramUseCase
```

```
domain.ParameterUseCase) {
    // GET ALL
    api.Register(humaAPI, huma.Operation{
        OperationID: "get-all-parameters",
        Method:      "POST",
        Path:        "/api/parameters/get-all",
        Summary:     "Get all parameters",
        Description: "Retrieves all active parameters from cache or
database",
        Tags:        []string{"Parameters"},
    }, func(ctx context.Context, input *struct {
        Body request.BaseRequest
    }) (*GetAllParametersResponse, error) {
        // Use case call
        result := paramUseCase.GetAll(ctx)

        // Map to response
        return &GetAllParametersResponse{
            Body: result,
        }, nil
    })

    // GET BY CODE
    api.Register(humaAPI, huma.Operation{
        OperationID: "get-parameter-by-code",
        Method:      "POST",
        Path:        "/api/parameters/get-by-code",
        Summary:     "Get parameter by code",
        Tags:        []string{"Parameters"},
    }, func(ctx context.Context, input *struct {
        Body request.GetParameterByCodeRequest
    }) (*GetParameterByCodeResponse, error) {
        result := paramUseCase.GetByCode(ctx, input.Body.Code)
        return &GetParameterByCodeResponse{Body: result}, nil
    })
}
```

4.6.4 Request DTOs (api/request/)

Estructuras con validación incorporada:

```
package request

type BaseRequest struct {
    IDSession   string    `json:"idSession" validate:"required"`
    IDRequest   string    `json:"idRequest" validate:"required,uuid"`
    Process     string    `json:"process" validate:"required"`
```

```
    IDDevice    string    `json:"idDevice" validate:"required"`
    PublicIP    string    `json:"publicIp" validate:"required,ip"`
    DateProcess time.Time `json:"dateProcess" validate:"required"`
}

type GetParameterByCodeRequest struct {
    BaseRequest
    Code string `json:"code" validate:"required,min=3,max=50"`
}

type AddParameterRequest struct {
    BaseRequest
    Name        string          `json:"name"
validate:"required,min=3,max=100"`
    Code        string          `json:"code"
validate:"required,min=3,max=50"`
    Data        json.RawMessage `json:"data" validate:"required"`
    Description string          `json:"description" validate:"max=500"`
}
```

Huma valida automáticamente estos campos antes de llamar al handler.

4.7 Data Access Layer (api/dal/)

Abstracción para llamadas a PostgreSQL. Proporciona funciones genéricas:

```
package dal

// QueryRows: Ejecuta función PostgreSQL que retorna múltiples filas
func QueryRows[T any](dal domain.DAL, ctx context.Context, funcName string,
args ...interface{}) ([]T, error)

// QueryRow: Ejecuta función PostgreSQL que retorna una fila
func QueryRow[T any](dal domain.DAL, ctx context.Context, funcName string,
args ...interface{}) (T, error)

// ExecProc: Ejecuta stored procedure (CALL statement)
func ExecProc[T any](dal domain.DAL, ctx context.Context, procName string,
args ...interface{}) (T, error)
```

4.7.1 Ventajas del DAL

- **Type-safe**: Usa generics de Go
- **DRY**: Elimina código repetitivo
- **Testeable**: Fácil de mockear
- **Database-agnostic**: Puede cambiar implementación

5 Módulos del Sistema

5.1 Sistema de Parámetros

Gestiona configuración dinámica del sistema mediante parámetros almacenados en DB.

5.1.1 Propósito

- Almacenar configuración que puede cambiar sin recompilar
- Parámetros JSON flexibles
- Cache en memoria para alto rendimiento
- CRUD completo vía API

5.1.2 Componentes

Tabla: `parameters`

```
CREATE TABLE parameters (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    code VARCHAR(50) NOT NULL UNIQUE,
    data JSONB NOT NULL,
    description TEXT,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Funciones PostgreSQL:

- `fn_get_all_parameters()`: Obtiene todos activos
- `fn_get_parameter_by_code(code)`: Obtiene por código
- `sp_create_parameter()`: Crea nuevo
- `sp_update_parameter()`: Actualiza existente
- `sp_delete_parameter()`: Soft delete

Cache: `internal/cache/parameter_cache.go`

- Thread-safe con `sync.RWMutex`
- Cache completo en memoria
- Invalidación manual o automática

5.1.3 Endpoints

Método	Path	Descripción
POST	/api/parameters/get-all	Obtener todos
POST	/api/parameters/get-by-code	Obtener por código
POST	/api/parameters/add	Crear parámetro
POST	/api/parameters/update	Actualizar parámetro

POST	/api/parameters/delete	Eliminar parámetro
POST	/api/parameters/reload-cache	Recargar cache

5.1.4 Ejemplo de Uso

Crear parámetro:

```
curl -X POST http://localhost:8080/api/parameters/add \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "admin",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "add-param",
    "idDevice": "admin-pc",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z",
    "name": "Horario de Atención",
    "code": "OFFICE_HOURS",
    "data": {
      "monday_friday": "08:00-17:00",
      "saturday": "08:00-13:00",
      "sunday": "closed"
    },
    "description": "Horarios de oficina"
  }'
```

Obtener parámetro:

```
curl -X POST http://localhost:8080/api/parameters/get-by-code \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "app",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "get-hours",
    "idDevice": "bot",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z",
    "code": "OFFICE_HOURS"
  }'
```

5.2 Motor de Conocimiento (RAG)

Sistema RAG (Retrieval Augmented Generation) para respuestas inteligentes.

5.2.1 Componentes

1. **Documentos** (`documents`)
   - Archivos fuente de conocimiento
   - Metadatos y estado de procesamiento

2. **Chunks** (`chunks`)
   - Fragmentos de texto indexados
   - Vector embeddings (1536 dims)
   - Metadatos para contexto
3. **Estadísticas** (`chunk_statistics`)
   - Métricas de uso de chunks
   - Frecuencia de consultas
   - Rankings de relevancia

5.2.2 Flujo RAG

1. **Usuario hace pregunta** → WhatsApp
2. **Generar embedding** → OpenAI/Ollama
3. **Búsqueda vectorial** → PostgreSQL pgvector
4. **Recuperar chunks relevantes** → Top K resultados
5. **Construir prompt** → Contexto + pregunta
6. **Generar respuesta** → LLM (Groq/OpenAI)
7. **Enviar respuesta** → WhatsApp

5.2.3 Búsqueda Semántica

Función PostgreSQL para similarity search:

```
CREATE OR REPLACE FUNCTION fn_similarity_search(
    query_embedding vector(1536),
    match_threshold float DEFAULT 0.7,
    match_count int DEFAULT 5
)
RETURNS TABLE (
    chunk_id bigint,
    document_id bigint,
    content text,
    similarity float
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.id,
        c.document_id,
        c.content,
        1 - (c.embedding <=> query_embedding) as similarity
    FROM chunks c
    WHERE c.active = true
      AND 1 - (c.embedding <=> query_embedding) > match_threshold
```

```
    ORDER BY c.embedding <=> query_embedding
    LIMIT match_count;
END;
$$ LANGUAGE plpgsql;
```

Operador <=>: Cosine distance de pgvector

5.2.4 Endpoints de Conocimiento

Documentos:

- `POST /api/documents/add`: Subir documento
- `POST /api/documents/get-all`: Listar documentos
- `POST /api/documents/get-by-id`: Obtener por ID
- `POST /api/documents/update`: Actualizar metadatos
- `POST /api/documents/delete`: Eliminar documento
- `POST /api/documents/process`: Procesar a chunks
- `POST /api/documents/search`: Buscar en documentos

Chunks:

- `POST /api/chunks/add`: Agregar chunk
- `POST /api/chunks/get-all`: Listar chunks
- `POST /api/chunks/get-by-document`: Por documento
- `POST /api/chunks/search`: Búsqueda semántica
- `POST /api/chunks/update`: Actualizar chunk
- `POST /api/chunks/delete`: Eliminar chunk
- `POST /api/chunks/bulk-add`: Agregar múltiples

Estadísticas:

- `POST /api/chunk-statistics/get-by-chunk`: Por chunk
- `POST /api/chunk-statistics/get-most-used`: Más usados
- `POST /api/chunk-statistics/increment`: Incrementar uso
- `POST /api/chunk-statistics/reset`: Resetear stats

5.3 Integración WhatsApp

Cliente WhatsApp completo usando `whatsmeow`.

5.3.1 Arquitectura

Cliente Principal (`whatsapp/client.go`):

- Gestión de sesión persistente
- Generación de QR code
- Conexión/desconexión
- Estado del cliente

Message Dispatcher (`whatsapp/handler.go`):

- Recibe mensajes entrantes

- Routing por tipo de mensaje
- Delega a handlers específicos

Handlers Especializados (`whatsapp/handlers/`):
- `rag_handler.go`: Preguntas generales (RAG)
- `command_handler.go`: Comandos especiales
- `auth_handler.go`: Autenticación de usuarios
- `fallback_handler.go`: Respuestas por defecto

5.3.2 Flujo de Mensajes

```
Usuario → WhatsApp → whatsmeow →
  MessageDispatcher →
    [Determinar tipo] →
      RAGHandler | CommandHandler | AuthHandler | FallbackHandler →
        [Procesar] →
          WhatsApp Client →
            Usuario
```

5.3.3 Gestión de Sesión

Tabla: `whatsapp_sessions`

```sql
CREATE TABLE whatsapp_sessions (
    id BIGSERIAL PRIMARY KEY,
    session_id VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(20),
    status VARCHAR(20) DEFAULT 'disconnected',
    qr_code TEXT,
    last_connected_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Estados:
- `disconnected`: Sin conexión
- `qr_generated`: QR code generado, esperando escaneo
- `connecting`: Estableciendo conexión
- `connected`: Conectado y funcionando
- `error`: Error de conexión

5.3.4 Endpoints de Control

- `POST /api/v1/whatsapp/start`: Iniciar cliente
- `POST /api/v1/whatsapp/stop`: Detener cliente
- `POST /api/v1/whatsapp/status`: Estado actual
- `POST /api/v1/whatsapp/qr`: Obtener QR code
- `POST /api/v1/whatsapp/logout`: Cerrar sesión

5.3.5 Ejemplo: Iniciar WhatsApp

```
curl -X POST http://localhost:8080/api/v1/whatsapp/start \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "admin",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "start-whatsapp",
    "idDevice": "server",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z"
  }'
```

Respuesta:

```
{
  "success": true,
  "code": "OK",
  "data": {
    "status": "qr_generated",
    "qrCode": "..."
  }
}
```

5.4 Panel de Administración

5.4.1 Autenticación JWT

Sistema de autenticación para administradores.

Tabla: `admins`

```
CREATE TABLE admins (
    id BIGSERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100),
    role VARCHAR(20) DEFAULT 'admin',
    active BOOLEAN DEFAULT true,
    last_login_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Endpoints:

- `POST /api/v1/admin/auth/login`: Login
- `POST /api/v1/admin/auth/logout`: Logout
- `POST /api/v1/admin/auth/refresh`: Refresh token
- `POST /api/v1/admin/auth/validate`: Validar token

Middleware: `api/middleware/jwt_auth_middleware.go`
- Valida JWT en header `Authorization: Bearer <token>`
- Extrae claims del token
- Verifica permisos

5.4.2 Panel de Conversaciones

Interfaz tipo WhatsApp para gestionar conversaciones.

Tablas:
- `conversations`: Lista de conversaciones
- `admin_conversation_messages`: Mensajes intercambiados

Endpoints:
- `POST /admin/conversations/get-all`: Listar conversaciones
- `POST /admin/conversations/get-messages`: Mensajes de una conversación
- `POST /admin/conversations/send-message`: Enviar mensaje
- `POST /admin/conversations/mark-read`: Marcar como leído
- `POST /admin/conversations/block`: Bloquear usuario
- `POST /admin/conversations/unblock`: Desbloquear
- `POST /admin/conversations/delete`: Eliminar conversación

Filtros disponibles:
- `all`: Todas las conversaciones
- `unread`: Solo con mensajes no leídos
- `blocked`: Usuarios bloqueados
- `active`: Conversaciones activas

5.4.3 Ejemplo: Obtener Conversaciones

```
curl -X POST http://localhost:8080/admin/conversations/get-all \
  -H "Authorization: Bearer <jwt_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "admin",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "get-conversations",
    "idDevice": "admin-panel",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z",
    "filter": "unread",
    "limit": 20,
    "offset": 0
  }'
```

6 Base de Datos

6.1 Schema Overview

El sistema utiliza PostgreSQL 15+ con las siguientes extensiones:

- `vector`: Búsquedas vectoriales (pgvector)
- `uuid-ossp`: Generación de UUIDs
- `pgcrypto`: Funciones criptográficas

6.1.1 Schemas

- `public`: Tablas y funciones de aplicación
- `ex`: Extensiones PostgreSQL (aisladas)

6.2 Sistema de Migraciones

El proyecto utiliza `golang-migrate` para gestionar cambios en el esquema de la base de datos de forma automática y versionada.

6.2.1 Configuración

Las migraciones se configuran en `config.json`:

```
{
  "Migration": {
    "AUTO_MIGRATE": true,  // Auto-ejecutar en startup
    "VERBOSE": true        // Logs detallados
  }
}
```

6.2.2 Migraciones Automáticas

Al iniciar la aplicación, las migraciones pendientes se ejecutan automáticamente:

```
// config/app.go
func runMigrations(env *Env) error {
    dsn := fmt.Sprintf("postgres://%s:%s@%s:%d/%s?sslmode=disable",
        env.Database.User, env.Database.Password,
        env.Database.Host, env.Database.Port, env.Database.Name)

    return migration.RunMigrations(migration.Config{
        AutoMigrate: env.Migration.AutoMigrate,
        Verbose:     env.Migration.Verbose,
        DSN:         dsn,
    })
}
```

Logs de ejemplo:

```
[MIGRATION] Auto-migration enabled, running pending migrations...
[MIGRATION] Start buffering 1/u database_setup
```

```
[MIGRATION] Finished 1/u database_setup (read 15ms, ran 102ms)
[MIGRATION] Current version: 14
[MIGRATION] Migrations completed successfully
```

6.2.3 Herramienta CLI

Para control manual de migraciones:

```
# Compilar la herramienta
go build -o migrate cmd/migrate/main.go

# Ver versión actual
./migrate -version

# Ejecutar migraciones pendientes
./migrate -up

# Revertir última migración
./migrate -down

# Forzar versión (recuperar estado "dirty")
./migrate -force 14
```

6.2.4 Crear Nueva Migración

1. Crear archivos de migración con el siguiente número de versión:

```
# Archivos de ejemplo
internal/migration/migrations/000015_add_analytics.up.sql
internal/migration/migrations/000015_add_analytics.down.sql
```

2. Escribir SQL en el archivo `.up.sql`:

```
-- 000015_add_analytics.up.sql
CREATE TABLE IF NOT EXISTS public.cht_analytics (
    id SERIAL PRIMARY KEY,
    event_name VARCHAR(100) NOT NULL,
    event_data JSONB,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX IF NOT EXISTS idx_analytics_event
ON cht_analytics(event_name);
```

3. Escribir rollback en `.down.sql`:

```
-- 000015_add_analytics.down.sql
DROP TABLE IF EXISTS cht_analytics CASCADE;
```

4. Recompilar aplicación (migraciones embebidas):

```
go build -o main cmd/main.go
./main  // Ejecuta migraciones automáticamente
```

6.2.5 Ubicación de Archivos

- **Migraciones**: `internal/migration/migrations/`
- **Runner**: `internal/migration/migration.go`
- **CLI**: `cmd/migrate/main.go`
- **Legacy SQL**: `db/` (solo referencia, NO ejecutar manualmente)

6.2.6 Versionado

La tabla `schema_migrations` rastrea el estado:

```
SELECT * FROM schema_migrations;

 version | dirty
---------+-------
      14 | f       -- f = limpio, t = requiere intervención
```

6.2.7 Buenas Prácticas

- Siempre usar `IF NOT EXISTS` / `IF EXISTS`
- Probar tanto UP como DOWN
- Mantener migraciones atómicas (un cambio lógico por archivo)
- Nunca editar migraciones ya aplicadas
- Hacer backup antes de migraciones grandes

Para más detalles: `docs/DATABASE_MIGRATIONS.md`

6.3 Tablas Principales

6.3.1 parameters

Configuración dinámica del sistema.

```
CREATE TABLE parameters (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    code VARCHAR(50) NOT NULL UNIQUE,
    data JSONB NOT NULL,
    description TEXT,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_parameters_code ON parameters(code);
CREATE INDEX idx_parameters_active ON parameters(active);
```

6.3.2 documents

Documentos fuente de conocimiento.

```
CREATE TABLE documents (
    id BIGSERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    file_name VARCHAR(255),
    file_path TEXT,
    file_type VARCHAR(50),
    file_size BIGINT,
    content TEXT,
    metadata JSONB,
    status VARCHAR(50) DEFAULT 'pending',
    processed_at TIMESTAMP,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_documents_status ON documents(status);
CREATE INDEX idx_documents_active ON documents(active);
```

Estados:

- `pending`: Cargado, no procesado
- `processing`: Siendo procesado a chunks
- `completed`: Procesado exitosamente
- `error`: Error en procesamiento

6.3.3 chunks

Fragmentos indexados con embeddings.

```
CREATE TABLE chunks (
    id BIGSERIAL PRIMARY KEY,
    document_id BIGINT REFERENCES documents(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    embedding vector(1536),
    chunk_index INT,
    tokens INT,
    metadata JSONB,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_chunks_document ON chunks(document_id);
CREATE INDEX idx_chunks_active ON chunks(active);
```

```
CREATE INDEX idx_chunks_embedding ON chunks USING ivfflat (embedding
vector_cosine_ops);
```

Índice vectorial: IVFFlat para búsquedas rápidas de similitud

6.3.4 chunk_statistics

Métricas de uso de chunks.

```
CREATE TABLE chunk_statistics (
    id BIGSERIAL PRIMARY KEY,
    chunk_id BIGINT REFERENCES chunks(id) ON DELETE CASCADE,
    query_count INT DEFAULT 0,
    last_queried_at TIMESTAMP,
    avg_similarity FLOAT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE UNIQUE INDEX idx_chunk_stats_chunk ON chunk_statistics(chunk_id);
```

6.3.5 conversations

Conversaciones con usuarios.

```
CREATE TABLE conversations (
    id BIGSERIAL PRIMARY KEY,
    chat_id VARCHAR(100) UNIQUE NOT NULL,
    user_id BIGINT REFERENCES whatsapp_users(id),
    phone_number VARCHAR(20),
    contact_name VARCHAR(100),
    is_group BOOLEAN DEFAULT false,
    last_message_at TIMESTAMP,
    message_count INT DEFAULT 0,
    unread_count INT DEFAULT 0,
    blocked BOOLEAN DEFAULT false,
    admin_intervened BOOLEAN DEFAULT false,
    temporary BOOLEAN DEFAULT false,
    expires_at TIMESTAMP,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_conversations_chat_id ON conversations(chat_id);
CREATE INDEX idx_conversations_user_id ON conversations(user_id);
CREATE INDEX idx_conversations_blocked ON conversations(blocked);
CREATE INDEX idx_conversations_unread ON conversations(unread_count) WHERE
unread_count > 0;
```

6.3.6 admin_conversation_messages

Mensajes entre admin y usuarios.

```
CREATE TABLE admin_conversation_messages (
    id BIGSERIAL PRIMARY KEY,
    conversation_id BIGINT REFERENCES conversations(id) ON DELETE CASCADE,
    admin_id BIGINT REFERENCES admins(id),
    message_type VARCHAR(20) DEFAULT 'text',
    content TEXT NOT NULL,
    from_admin BOOLEAN DEFAULT false,
    read BOOLEAN DEFAULT false,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    read_at TIMESTAMP,
    metadata JSONB
);

CREATE INDEX idx_admin_messages_conversation ON
admin_conversation_messages(conversation_id);
CREATE INDEX idx_admin_messages_sent_at ON
admin_conversation_messages(sent_at DESC);
CREATE INDEX idx_admin_messages_unread ON admin_conversation_messages(read)
WHERE NOT read;
```

6.3.7 whatsapp_sessions

Sesiones de WhatsApp.

```
CREATE TABLE whatsapp_sessions (
    id BIGSERIAL PRIMARY KEY,
    session_id VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(20),
    status VARCHAR(20) DEFAULT 'disconnected',
    qr_code TEXT,
    last_connected_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6.3.8 whatsapp_users

Usuarios de WhatsApp registrados.

```
CREATE TABLE whatsapp_users (
    id BIGSERIAL PRIMARY KEY,
    chat_id VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
    name VARCHAR(100),
    identity_number VARCHAR(20) UNIQUE,
    email VARCHAR(100),
```

```
    role VARCHAR(50),
    verified BOOLEAN DEFAULT false,
    blocked BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_whatsapp_users_chat_id ON whatsapp_users(chat_id);
CREATE INDEX idx_whatsapp_users_identity ON whatsapp_users(identity_number);
```

6.3.9 admins

Administradores del sistema.

```
CREATE TABLE admins (
    id BIGSERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100),
    role VARCHAR(20) DEFAULT 'admin',
    active BOOLEAN DEFAULT true,
    last_login_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_admins_username ON admins(username);
CREATE INDEX idx_admins_email ON admins(email);
```

6.4 Stored Procedures y Funciones

El sistema utiliza **database-first approach**: toda la lógica de datos está en PostgreSQL.

6.4.1 Guía de Mejores Prácticas SQL

6.4.1.1 Reglas de Nomenclatura

Variables en SQL/PLpgSQL:

- **Parámetros de entrada**: Prefijo `p_` (ejemplo: `p_id_usuario`, `p_estado`)
- **Variables de salida**: Prefijo `o_` (ejemplo: `o_id_parametro`) Especiales: `success BOOLEAN`, `code VARCHAR`
- **Variables locales**: Prefijo `v_` (ejemplo: `v_mod_id`, `v_param_id`)
- **Registros temporales**: Prefijo `r_` (ejemplo: `r_modulo`)
- **Contadores**: Prefijo `i_` (ejemplo: `i_contador`)
- **Booleanos**: Prefijo `bl_` o `is_` (ejemplo: `bl_existe`, `is_active`)
- **Constantes**: Prefijo `c_` (ejemplo: `c_estado_activo`)

Funciones y Procedimientos:

- **`sp_`**: Procedimientos almacenados (ejemplo: `sp_add_usuario`)
- **`fn_`**: Funciones (ejemplo: `fn_get_parametros`)
- **`vw_`**: Vistas (ejemplo: `vw_usuarios_permisos`)
- **`tr_`**: Triggers (ejemplo: `tr_update_timestamp`)

Convenciones:

- Los procedimientos siempre retornan:
  - `success`: BOOLEAN
  - `code`: VARCHAR (código único del error, ej. "OK", "ERR_NOT_FOUND")
- Usar `snake_case` en todos los identificadores
- Nombres descriptivos en inglés o español (consistente)

6.4.1.2 Convenciones Generales

- **Siempre comentar** cada bloque relevante
- **Indentación**: 4 espacios o tabs
- Palabras clave SQL en **minúsculas**
- Usar `DO $$ ... $$;` solo cuando es necesario
- Scripts deben ser **idempotentes** (usar `IF NOT EXISTS`)
- No usar abreviaturas ambiguas

6.4.1.3 Ejemplo: Script de Inserción

```
DO $$
DECLARE
    v_mod_id      INT;
    v_prm_id      INT;
BEGIN
    -- Obtener ID del módulo SISTEMA
    SELECT mod_id INTO v_mod_id
    FROM cht_modulos WHERE mod_codigo = 'SISTEMA';

    -- Siguiente prm_id disponible
    SELECT COALESCE(MAX(prm_id), 0) + 1 INTO v_prm_id
    FROM cht_parametros;

    -- Insertar parámetro si no existe
    IF NOT EXISTS (
        SELECT 1 FROM cht_parametros WHERE prm_nemonico = 'COD_OK'
    ) THEN
        INSERT INTO cht_parametros (
            prm_id, prm_fk_modulo, prm_nombre, prm_nemonico,
            prm_valor1, prm_valor2, prm_descripcion,
            prm_tipo, prm_activo
```

```
        ) VALUES (
            v_prm_id, v_mod_id, 'Operación exitosa', 'COD_OK',
            'Operación realizada correctamente.', '',
            'Mensaje estándar de éxito para operaciones.',
            'string', TRUE
        );
    END IF;
END
$$;
```

6.4.1.4 Ejemplo: Función con Trigger

```
-- Función para actualizar updated_at automáticamente
CREATE OR REPLACE FUNCTION fn_set_updated_at()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    NEW.updated_at := CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$;

-- Trigger que llama a la función
CREATE TRIGGER tr_update_post
BEFORE UPDATE ON post
FOR EACH ROW
EXECUTE FUNCTION fn_set_updated_at();
```

6.4.2 Patrón de Nomenclatura

- `fn_*`: Funciones que retornan datos (SELECT)
- `sp_*`: Stored procedures que modifican datos (INSERT/UPDATE/DELETE)

6.4.3 Ejemplos de Funciones

fn_get_all_parameters()

```
CREATE OR REPLACE FUNCTION fn_get_all_parameters()
RETURNS TABLE (
    id bigint,
    name varchar,
    code varchar,
    data jsonb,
    description text,
    active boolean,
    created_at timestamp,
    updated_at timestamp
)
```

```
AS $$
BEGIN
    RETURN QUERY
    SELECT
        p.id, p.name, p.code, p.data,
        p.description, p.active,
        p.created_at, p.updated_at
    FROM parameters p
    WHERE p.active = true
    ORDER BY p.name;
END;
$$ LANGUAGE plpgsql;
```

fn_similarity_search()

```
CREATE OR REPLACE FUNCTION fn_similarity_search(
    query_embedding vector(1536),
    match_threshold float DEFAULT 0.7,
    match_count int DEFAULT 5
)
RETURNS TABLE (
    chunk_id bigint,
    document_id bigint,
    content text,
    similarity float
)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        c.id,
        c.document_id,
        c.content,
        1 - (c.embedding <=> query_embedding) as similarity
    FROM chunks c
    WHERE c.active = true
      AND 1 - (c.embedding <=> query_embedding) > match_threshold
    ORDER BY c.embedding <=> query_embedding
    LIMIT match_count;
END;
$$ LANGUAGE plpgsql;
```

6.4.4 Ejemplos de Procedures

sp_create_parameter()

```
CREATE OR REPLACE PROCEDURE sp_create_parameter(
    OUT out_success boolean,
```

```
    OUT out_code varchar,
    OUT out_message text,
    IN in_name varchar,
    IN in_code varchar,
    IN in_data jsonb,
    IN in_description text DEFAULT NULL
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Verificar si ya existe
    IF EXISTS (SELECT 1 FROM parameters WHERE code = in_code) THEN
        out_success := false;
        out_code := 'ERR_PARAM_CODE_EXISTS';
        out_message := 'Parameter code already exists';
        RETURN;
    END IF;

    -- Insertar
    INSERT INTO parameters (name, code, data, description)
    VALUES (in_name, in_code, in_data, in_description);

    out_success := true;
    out_code := 'OK';
    out_message := 'Parameter created successfully';
END;
$$;
```

sp_add_chunk_with_embedding()

```
CREATE OR REPLACE PROCEDURE sp_add_chunk_with_embedding(
    OUT out_success boolean,
    OUT out_code varchar,
    OUT out_message text,
    OUT out_chunk_id bigint,
    IN in_document_id bigint,
    IN in_content text,
    IN in_embedding vector(1536),
    IN in_chunk_index int DEFAULT NULL,
    IN in_tokens int DEFAULT NULL,
    IN in_metadata jsonb DEFAULT NULL
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO chunks (
        document_id, content, embedding,
```

```
        chunk_index, tokens, metadata
    )
    VALUES (
        in_document_id, in_content, in_embedding,
        in_chunk_index, in_tokens, in_metadata
    )
    RETURNING id INTO out_chunk_id;

    out_success := true;
    out_code := 'OK';
    out_message := 'Chunk added successfully';
END;
$$;
```

6.5 Migraciones

Los archivos de migración están en `db/` y deben ejecutarse en orden:

1. `00_database_setup.sql`: Crea database, schemas, extensiones
2. `01_create_tables.sql`: Crea todas las tablas
3. `02_parameters_procedures.sql`: Procedures de parámetros
4. `03_knowledge_procedures.sql`: Procedures RAG (docs, chunks)
5. `04_conversation_procedures.sql`: Procedures de conversaciones
6. `05_admin_procedures.sql`: Procedures de admin
7. `06_whatsapp_procedures.sql`: Procedures de WhatsApp
8. `07_analytics_procedures.sql`: Procedures de analytics
9. `initial_data.sql`: Datos semilla

Script de migración completo:

```
#!/bin/bash
DB_USER="chatbot_user"
DB_NAME="chatbot_db"
DB_HOST="localhost"
DB_PORT="5432"

export PGPASSWORD="your_password"

for file in db/*.sql; do
  echo "Executing $file..."
  psql -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME -f $file
  if [ $? -ne 0 ]; then
    echo "Error executing $file"
    exit 1
  fi
done
```

```
echo "All migrations completed successfully"
```

7 Frontend Web Application

7.1 Visión General del Frontend

El sistema incluye una **aplicación web de administración** construida con React y TypeScript que proporciona una interfaz gráfica completa para gestionar el chatbot.

7.1.1 Tecnologías Principales

- **React 19.1.0**: Framework de UI
- **TypeScript 5.8.3**: Tipado estático
- **TanStack Router v1.128.0**: Routing basado en archivos
- **TanStack React Query v5.83.0**: Gestión de estado del servidor
- **Zustand v5.0.6**: Estado del cliente (autenticación)
- **Axios v1.10.0**: Cliente HTTP
- **TailwindCSS v4.1.15**: Estilos
- **Radix UI + shadcn/ui**: Componentes UI accesibles
- **Bun**: Runtime y build tool

7.1.2 Puerto y Acceso

- **Desarrollo**: `http://localhost:3000`
- **Producción**: `http://localhost:6600` (Docker)
- **Backend API**: `http://localhost:8080`

7.2 Estructura de Directorios

```
app-web-chatbot/
├── src/
│   ├── api/                    # Integración con backend
│   │   ├── entities/           # Modelos compartidos
│   │   ├── frontend-types/     # Tipos TypeScript (espejo de Go)
│   │   ├── http/               # Cliente Axios con interceptors
│   │   └── services/           # Funciones de servicio con React Query
│   │
│   ├── components/             # Componentes reutilizables
│   │   ├── layout/             # Layout (header, sidebar, nav)
│   │   └── ui/                 # Componentes UI (Radix + shadcn)
│   │
│   ├── config/                 # Configuración
│   │   ├── router.ts           # Setup TanStack Router
│   │   └── react-query.ts      # Configuración React Query
│   │
│   ├── context/                # React Context
│   │   ├── auth-context.tsx    # Contexto de autenticación
│   │   ├── theme-context.tsx   # Tema dark/light
│   │   └── search-context.tsx  # Búsqueda global
│   │
```

```
│   ├── features/              # Módulos por funcionalidad
│   │   ├── auth/              # Login, reset password
│   │   ├── chats/             # Interfaz de mensajería
│   │   ├── dashboard/         # Panel de control
│   │   ├── rag/               # Gestión de documentos RAG
│   │   ├── system/            # Admin (usuarios, roles, permisos)
│   │   ├── whatsapp/          # Integración WhatsApp
│   │   └── errors/            # Páginas de error
│   │
│   ├── hooks/                 # Custom React hooks
│   ├── stores/                # Zustand stores
│   ├── utils/                 # Funciones utilitarias
│   ├── routes/                # Estructura de rutas
│   │   ├── _authenticated/    # Rutas protegidas
│   │   ├── (auth)/            # Rutas públicas de auth
│   │   └── (errors)/          # Páginas de error
│   │
│   ├── index.tsx              # Entry point servidor Bun
│   ├── main.tsx               # Entry point app React
│   └── index.css              # Estilos globales
│
├── public/                    # Assets estáticos
├── dist/                      # Build de producción
├── Dockerfile                 # Imagen Docker con Nginx
├── deploy.sh                  # Script de despliegue
├── package.json
├── tsconfig.json
├── tailwind.config.ts
└── vite.config.ts
```

7.3 Arquitectura del Frontend

7.3.1 Patrón Arquitectónico

El frontend sigue una arquitectura basada en **Feature Modules** con separación de concerns:

```
UI Components → API Services → Backend REST API
       ↓              ↓
   React Query    Axios Client
   (Cache)       (HTTP)
```

7.3.2 Flujo de Datos

```
User Interaction
    ↓
React Component
    ↓
React Query Hook (useQuery/useMutation)
```

```
    ↓
API Service Function
    ↓
Axios Client (with interceptors)
    ↓
Backend API
    ↓
PostgreSQL
```

7.3.3 Gestión de Estado

Zustand para estado del cliente:
- Usuario autenticado
- Tokens de acceso y refresh
- Preferencias de UI

React Query para estado del servidor:
- Datos del backend
- Cache automático (10 segundos)
- Invalidación inteligente
- Retry logic

React Context para estado global:
- Tema (dark/light)
- Autenticación
- Búsqueda global

7.4 Sistema de Routing

7.4.1 TanStack Router

Utiliza **file-based routing** con generación automática:

```
// Generado automáticamente en routeTree.gen.ts
export const routeTree = rootRoute.addChildren([
  indexRoute,
  authenticatedRoute.addChildren([
    panelDeControlRoute,
    chatsRoute,
    ragRoute,
    whatsappRoute,
    // ...más rutas
  ]),
  authLayoutRoute.addChildren([
    loginRoute,
    forgotPasswordRoute,
  ]),
  errorsLayoutRoute.addChildren([
```

```
    notFoundRoute,
    unauthorizedRoute,
    // ...más errores
  ]),
])
```

7.4.2 Rutas Principales

Públicas:

```
/                              # Landing page
/iniciar-sesion                # Login
/olvide-mi-contrasena          # Reset password
```

Protegidas (requieren auth):

```
/_authenticated/panel-de-control    # Dashboard
/_authenticated/chats               # Mensajería
/_authenticated/rag                 # Documentos RAG
/_authenticated/whatsapp            # WhatsApp control
/_authenticated/usuarios            # Gestión usuarios
/_authenticated/estadisticas        # Analytics
/_authenticated/sistema/*           # Admin sistema
```

Errores:

```
/(errors)/404                  # No encontrado
/(errors)/401                  # No autorizado
/(errors)/403                  # Prohibido
/(errors)/500                  # Error servidor
/(errors)/503                  # Mantenimiento
```

7.4.3 Protección de Rutas

```
// Componente ProtectedRoute
const ProtectedRoute = ({ children }) => {
  const { isAuthenticated, isLoading } = useAuth()

  if (isLoading) return <LoadingSpinner />
  if (!isAuthenticated) return <Navigate to="/iniciar-sesion" />

  return children
}
```

7.5 Integración con Backend

7.5.1 Cliente Axios

Configuración base:

```
const apiClient = axios.create({
  baseURL: 'http://localhost:8080',
  headers: {
    'Content-Type': 'application/json',
    'X-App-Authorization': 'chatbot-web-app'
  }
})
```

7.5.2 Interceptors

Request Interceptor:

```
apiClient.interceptors.request.use(config => {
  const token = localStorage.getItem('accessToken')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

Response Interceptor (Token Refresh):

```
apiClient.interceptors.response.use(
  response => response,
  async error => {
    if (error.response?.status === 401) {
      // Intentar refresh token
      const refreshToken = localStorage.getItem('refreshToken')
      const newToken = await refreshAccessToken(refreshToken)

      // Reintentar request original
      error.config.headers.Authorization = `Bearer ${newToken}`
      return apiClient.request(error.config)
    }
    return Promise.reject(error)
  }
)
```

7.5.3 Servicios API

Estructura de servicio típica:

```
// src/api/services/parameters.service.ts
export const useGetAllParameters = () => {
  return useQuery({
    queryKey: ['parameters', 'all'],
    queryFn: async () => {
      const response = await apiClient.post('/api/parameters/get-all', {
        idSession: 'admin',
```

```
        idRequest: crypto.randomUUID(),
        process: 'get-all-params',
        idDevice: 'web',
        publicIp: '127.0.0.1',
        dateProcess: new Date().toISOString()
      })
      return response.data
    }
  })
}

export const useAddParameter = () => {
  const queryClient = useQueryClient()

  return useMutation({
    mutationFn: async (data: CreateParameterDTO) => {
      const response = await apiClient.post('/api/parameters/add', data)
      return response.data
    },
    onSuccess: () => {
      // Invalidar cache para refrescar datos
      queryClient.invalidateQueries({ queryKey: ['parameters'] })
      toast.success('Parámetro creado exitosamente')
    },
    onError: (error) => {
      toast.error('Error al crear parámetro')
    }
  })
}
```

7.5.4 Formato de Respuesta

Todas las respuestas siguen el formato `IResponse<T>`:

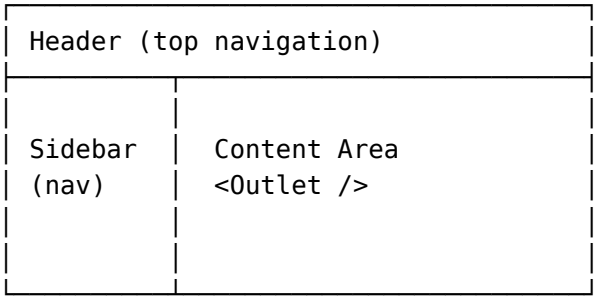
```
interface IResponse<T> {
  success: boolean
  code: string
  info?: string
  data?: T
}

// Validación
function isValidResponse<T>(response: any): response is IResponse<T> {
  return (
    typeof response === 'object' &&
    'success' in response &&
    'code' in response
```

```
  )
}
```

7.6 Componentes Principales

7.6.1 Layout

authenticated-layout.tsx



Componentes:

- `app-sidebar`: Navegación lateral colapsable
- `header`: Barra superior con breadcrumbs
- `nav-user`: Dropdown de perfil de usuario
- `top-nav`: Navegación dinámica por ruta

7.6.2 Dashboard

Panel de control principal con:

- Métricas de servicio (costos)
- Métricas de rendimiento (velocidad, tokens, usuarios)
- Gráficos de actividad
- Actividades recientes
- Navegación rápida

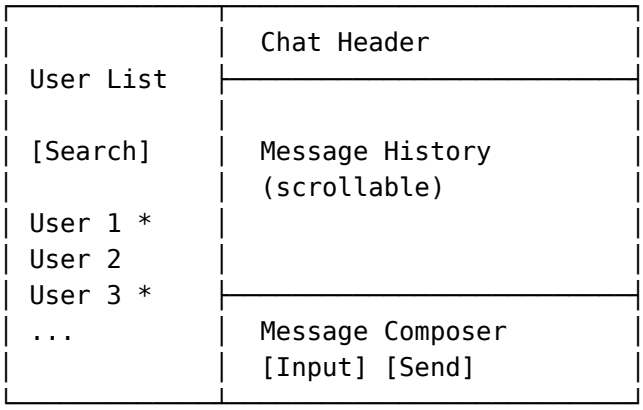
Componentes clave:

```tsx
// features/dashboard/index.tsx
export const DashboardPage = () => {
  const { data: metrics } = useGetMetrics()
  const { data: activities } = useGetRecentActivities()

  return (
    <div className="grid gap-4">
      <MetricsCards data={metrics} />
      <ChartsSection />
      <ActivitiesTable data={activities} />
    </div>
```

```
  )
}
```

7.6.3 Panel de Chats

Interfaz de mensajería tipo WhatsApp:




```
* = unread messages
```

Funcionalidades:

- Lista de usuarios con búsqueda
- Historial de conversación
- Indicadores de mensajes no leídos
- Composer con soporte de adjuntos
- Scroll automático a nuevos mensajes
- Responsive (vista móvil oculta lista en chat activo)

7.6.4 Gestión de Documentos RAG

Panel para administrar conocimiento del chatbot:

Secciones:

1. **Documentos**: Subir y gestionar archivos fuente
2. **Chunks**: Ver fragmentos indexados con embeddings
3. **Estadísticas**: Métricas de uso por chunk

Funciones:

```tsx
// features/rag/documents/
const useUploadDocument = () => {
  return useMutation({
    mutationFn: async (file: File) => {
      const formData = new FormData()
      formData.append('file', file)
```

```
      const response = await apiClient.post(
        '/api/documents/add',
        formData,
        { headers: { 'Content-Type': 'multipart/form-data' } }
      )
      return response.data
    }
  })
}

const useProcessDocument = () => {
  return useMutation({
    mutationFn: async (docId: number) => {
      const response = await apiClient.post(
        '/api/documents/process',
        { id: docId }
      )
      return response.data
    }
  })
}
```

7.6.5 Integración WhatsApp

Panel de control de WhatsApp:

Funcionalidades:

- Mostrar QR code para conexión
- Estado de sesión (conectado/desconectado)
- Iniciar/detener cliente
- Ver historial de mensajes

```
// features/whatsapp/
const useWhatsAppQR = () => {
  return useQuery({
    queryKey: ['whatsapp', 'qr'],
    queryFn: async () => {
      const response = await apiClient.post('/api/v1/whatsapp/qr')
      return response.data.data.qrCode // ...
    },
    refetchInterval: 5000, // Refrescar cada 5 segundos
    enabled: !isConnected
  })
}

const useStartWhatsApp = () => {
  return useMutation({
```

```
    mutationFn: async () => {
      const response = await apiClient.post('/api/v1/whatsapp/start')
      return response.data
    }
  })
}
```

7.6.6 Administración del Sistema

Módulos:

1. **Usuarios** (`/usuarios`):
   - Crear, editar, eliminar usuarios
   - Asignar roles y permisos
   - Ver historial de actividad
2. **Roles y Permisos** (`/sistema/permisos`):
   - Definir roles
   - Asignar permisos por módulo
   - Control de acceso granular
3. **Parámetros** (`/sistema/parametros`):
   - CRUD de parámetros del sistema
   - Edición JSON de datos
   - Activar/desactivar parámetros
4. **Funcionalidades** (`/sistema/funcionalidades`):
   - Gestionar módulos del sistema
   - Habilitar/deshabilitar features

7.7 Componentes UI (shadcn/ui)

El proyecto utiliza **shadcn/ui**, una colección de componentes reutilizables construidos sobre Radix UI.

7.7.1 Componentes Disponibles

En `src/components/ui/`:

- `button`: Botones con variantes (default, destructive, outline, ghost, link)
- `input`: Inputs de texto con validación
- `select`: Dropdowns y select
- `dialog`: Modales y diálogos
- `dropdown-menu`: Menús contextuales
- `toast`: Notificaciones (usando Sonner)
- `table`: Tablas con sorting y paginación
- `card`: Cards con header, content, footer

- `tabs`: Navegación por tabs
- `form`: Formularios con React Hook Form
- `badge`: Badges y labels
- `avatar`: Avatares de usuario
- `separator`: Separadores visuales
- `scroll-area`: Áreas scrollables estilizadas
- `sheet`: Sidesheets deslizables

7.7.2 Ejemplo de Uso

```
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Card, CardHeader, CardTitle, CardContent } from '@/components/ui/
card'

export const LoginForm = () => {
  const { mutate: login } = useLogin()

  return (
    <Card className="w-full max-w-md">
      <CardHeader>
        <CardTitle>Iniciar Sesión</CardTitle>
      </CardHeader>
      <CardContent>
        <form onSubmit={handleSubmit(login)}>
          <Input
            type="text"
            placeholder="Usuario"
            {...register('username')}
          />
          <Input
            type="password"
            placeholder="Contraseña"
            {...register('password')}
          />
          <Button type="submit" className="w-full">
            Ingresar
          </Button>
        </form>
      </CardContent>
    </Card>
  )
}
```

7.7.3 Tema y Estilos

TailwindCSS con custom config:

```
// tailwind.config.ts
export default {
  theme: {
    extend: {
      colors: {
        border: "hsl(var(--border))",
        input: "hsl(var(--input))",
        ring: "hsl(var(--ring))",
        background: "hsl(var(--background))",
        foreground: "hsl(var(--foreground))",
        primary: {
          DEFAULT: "hsl(var(--primary))",
          foreground: "hsl(var(--primary-foreground))",
        },
        // ...más colores
      },
      borderRadius: {
        lg: "var(--radius)",
        md: "calc(var(--radius) - 2px)",
        sm: "calc(var(--radius) - 4px)",
      },
    },
  },
  plugins: [require("tailwindcss-animate")],
}
```

Dark Mode: Soportado mediante clase `.dark` en root:

```
// context/theme-context.tsx
export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState(() =>
    localStorage.getItem('theme') || 'light'
  )

  useEffect(() => {
    document.documentElement.classList.toggle('dark', theme === 'dark')
  }, [theme])

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  )
}
```

7.8 Desarrollo y Build

7.8.1 Entorno de Desarrollo

Requisitos:

- Bun 1.0+ (o Node.js 22+)
- npm o yarn

Instalación:

```
cd app-web-chatbot
bun install
```

Ejecutar en desarrollo:

```
bun --hot src/index.tsx
```

Características en dev:

- Hot Module Replacement (HMR)
- React Query DevTools
- React Router DevTools
- Source maps habilitados
- Console logs de servidor

7.8.2 Build de Producción

```
bun run build.ts
```

Proceso de build:

1. Compila TypeScript a JavaScript
2. Minifica código
3. Optimiza assets
4. Genera source maps
5. Output en `dist/`

Contenido de dist/:

```
dist/
├── index.html
├── assets/
│   ├── index-[hash].js
│   ├── index-[hash].css
│   └── [otros assets]
└── [archivos estáticos]
```

7.8.3 Despliegue con Docker

Dockerfile:

```
# Build stage
FROM node:22-slim AS build
WORKDIR /app
COPY package.json bun.lockb ./
RUN npm install
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

nginx.conf:

```
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # SPA fallback
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Cache static assets
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
        expires 1M;
        add_header Cache-Control "public, immutable";
    }

    # Never cache index.html
    location = /index.html {
        add_header Cache-Control "no-store, no-cache, must-revalidate";
    }

    # Max upload size
    client_max_body_size 100M;
}
```

Script de despliegue (`deploy.sh`):

```
#!/bin/bash
```

```
IMAGE_NAME="img_front_chatbot"
CONTAINER_NAME="cnt-front-chatbot"
PORT=6600

echo "Building Docker image..."
docker build -t $IMAGE_NAME .

echo "Stopping existing container..."
docker stop $CONTAINER_NAME 2>/dev/null || true
docker rm $CONTAINER_NAME 2>/dev/null || true

echo "Starting new container..."
docker run -d \
  --name $CONTAINER_NAME \
  -p $PORT:80 \
  --env-file /config/appWebChatbot/.env \
  --restart unless-stopped \
  $IMAGE_NAME

echo "Deployment complete!"
echo "Frontend accessible at: http://localhost:$PORT"
```

Ejecutar despliegue:

```
./deploy.sh
```

7.9 Mejores Prácticas Frontend

7.9.1 Nomenclatura

- **Componentes**: PascalCase (`LoginForm.tsx`)
- **Archivos**: kebab-case (`login-form.tsx`)
- **Hooks**: camelCase con prefijo `use` (`useAuth`)
- **Constantes**: UPPER_SNAKE_CASE

7.9.2 Organización de Componentes

```
// Estructura de componente típica
import { useState } from 'react'
import { useQuery } from '@tanstack/react-query'
import { Button } from '@/components/ui/button'

interface MyComponentProps {
  id: number
  onComplete: () => void
}

export const MyComponent = ({ id, onComplete }: MyComponentProps) => {
```

```
  // 1. Hooks de estado
  const [isOpen, setIsOpen] = useState(false)

  // 2. Queries y mutations
  const { data, isLoading } = useQuery({...})
  const { mutate } = useMutation({...})

  // 3. Handlers
  const handleSubmit = () => {
    mutate(data, { onSuccess: onComplete })
  }

  // 4. Effects
  useEffect(() => {
    // side effects
  }, [dependencies])

  // 5. Render
  if (isLoading) return <Loading />

  return (
    <div>
      {/* JSX */}
    </div>
  )
}
```

7.9.3 Performance

- Usar `React.memo()` para componentes pesados
- Lazy loading de rutas con `React.lazy()`
- Virtualization para listas largas
- Debounce en búsquedas
- Optimistic updates en mutaciones

7.9.4 Accesibilidad

- Usar componentes Radix UI (accesibles por defecto)
- Atributos ARIA apropiados
- Navegación por teclado
- Contraste de colores adecuado
- Labels en formularios

8 API REST

8.1 OpenAPI Documentation

Toda la API está documentada automáticamente con OpenAPI 3.1 mediante Huma.

Acceso a documentación:

- **UI Interactiva**: `http://localhost:8080/docs`
- **OpenAPI JSON**: `http://localhost:8080/openapi.json`
- **OpenAPI YAML**: `http://localhost:8080/openapi.yaml`

8.2 Formato de Respuesta

Todas las respuestas siguen el formato `Result[T]`:

```
{
  "success": true,
  "code": "OK",
  "info": "Optional message",
  "data": { /* T */ }
}
```

En caso de error:

```
{
  "success": false,
  "code": "ERR_CODE",
  "info": "Error description",
  "data": null
}
```

8.3 Códigos de Respuesta

8.3.1 Códigos de Éxito

- `OK`: Operación exitosa general
- `CREATED`: Recurso creado exitosamente
- `UPDATED`: Recurso actualizado
- `DELETED`: Recurso eliminado

8.3.2 Códigos de Error

Parámetros:

- `ERR_PARAM_NOT_FOUND`: Parámetro no encontrado
- `ERR_PARAM_CODE_EXISTS`: Código ya existe
- `ERR_PARAM_INVALID`: Parámetro inválido

Documentos:

- `ERR_DOC_NOT_FOUND`: Documento no encontrado
- `ERR_DOC_PROCESSING`: Error procesando documento

- `ERR_DOC_INVALID_FORMAT`: Formato inválido

Chunks:
- `ERR_CHUNK_NOT_FOUND`: Chunk no encontrado
- `ERR_EMBEDDING_FAILED`: Fallo generando embedding

Autenticación:
- `ERR_AUTH_INVALID_CREDENTIALS`: Credenciales inválidas
- `ERR_AUTH_TOKEN_EXPIRED`: Token expirado
- `ERR_AUTH_UNAUTHORIZED`: No autorizado

Base de datos:
- `ERR_DB_QUERY`: Error en query
- `ERR_DB_CONNECTION`: Error de conexión

Genéricos:
- `ERR_VALIDATION`: Error de validación
- `ERR_INTERNAL`: Error interno del servidor

8.4 Endpoints Completos

8.4.1 Sistema

```
GET  /health                    # Health check
GET  /docs                      # OpenAPI UI
GET  /openapi.json             # OpenAPI spec
```

8.4.2 Parámetros

```
POST /api/parameters/get-all           # Obtener todos
POST /api/parameters/get-by-code       # Obtener por código
POST /api/parameters/add               # Crear
POST /api/parameters/update            # Actualizar
POST /api/parameters/delete            # Eliminar
POST /parameters/reload-cache           # Recargar cache
```

8.4.3 Documentos

```
POST /api/documents/add                # Subir documento
POST /api/documents/get-all            # Listar todos
POST /api/documents/get-by-id          # Obtener por ID
POST /api/documents/update             # Actualizar
POST /api/documents/delete             # Eliminar
POST /api/documents/process            # Procesar a chunks
POST /api/documents/search             # Buscar
```

8.4.4 Chunks

```
POST /api/chunks/add                   # Agregar chunk
POST /api/chunks/get-all               # Listar todos
```

```
POST /api/chunks/get-by-document       # Por documento
POST /api/chunks/search                # Búsqueda semántica
POST /api/chunks/update                # Actualizar
POST /api/chunks/delete                # Eliminar
POST /api/chunks/bulk-add              # Agregar múltiples
```

8.4.5 WhatsApp Admin

```
POST /api/v1/whatsapp/start             # Iniciar cliente
POST /api/v1/whatsapp/stop              # Detener cliente
POST /api/v1/whatsapp/status            # Estado
POST /api/v1/whatsapp/qr                # Obtener QR
POST /api/v1/whatsapp/logout            # Cerrar sesión
```

8.4.6 Admin Auth

```
POST /api/v1/admin/auth/login           # Login
POST /api/v1/admin/auth/logout          # Logout
POST /api/v1/admin/auth/refresh         # Refresh token
POST /api/v1/admin/auth/validate        # Validar token
```

8.4.7 Admin Conversations

```
POST /admin/conversations/get-all       # Listar conversaciones
POST /admin/conversations/get-messages  # Mensajes
POST /admin/conversations/send-message  # Enviar mensaje
POST /admin/conversations/mark-read     # Marcar leído
POST /admin/conversations/block         # Bloquear usuario
POST /admin/conversations/unblock       # Desbloquear
POST /admin/conversations/delete        # Eliminar conversación
```

8.5 Autenticación

Los endpoints del panel admin requieren JWT en header:

```
Authorization: Bearer <jwt_token>
```

Obtener token:

```
curl -X POST http://localhost:8080/api/v1/admin/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "admin-login",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "login",
    "idDevice": "browser",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z",
    "username": "admin",
    "password": "admin123"
  }'
```

Respuesta:

```
{
  "success": true,
  "code": "OK",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIs...",
    "expiresAt": "2025-10-28T10:00:00Z"
  }
}
```

Usar token:

```
curl -X POST http://localhost:8080/admin/conversations/get-all \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." \
  -H "Content-Type: application/json" \
  -d '{ ... }'
```

9 Extensión y Mantenimiento

9.1 Agregar Nueva Funcionalidad

Siga este workflow para agregar una nueva feature:

9.1.1 Paso 1: Diseñar Base de Datos

Crear archivo en `db/`:

```
-- db/08_nueva_feature_tables.sql

CREATE TABLE nueva_entidad (
    id BIGSERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    datos JSONB,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_nueva_entidad_nombre ON nueva_entidad(nombre);
```

Crear procedures:

```
-- db/09_nueva_feature_procedures.sql

CREATE OR REPLACE FUNCTION fn_get_all_nueva_entidad()
RETURNS TABLE (...)
AS $$
BEGIN
    -- Implementación
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE sp_create_nueva_entidad(...)
AS $$
BEGIN
    -- Implementación
END;
$$ LANGUAGE plpgsql;
```

9.1.2 Paso 2: Capa de Dominio

Crear `domain/nueva_entidad.go`:

```
package domain

// Entity
type NuevaEntidad struct {
    ID        int64           `json:"id"`
```

```
    Nombre    string          `json:"nombre"`
    Datos     json.RawMessage `json:"datos"`
    Active    bool            `json:"active"`
    CreatedAt time.Time       `json:"createdAt"`
}

// Repository Interface
type NuevaEntidadRepository interface {
    GetAll(ctx context.Context) Result[[]NuevaEntidad]
    GetByID(ctx context.Context, id int64) Result[NuevaEntidad]
    Create(ctx context.Context, nombre string, datos json.RawMessage)
Result[int64]
    Update(ctx context.Context, id int64, nombre string, datos
json.RawMessage) Result[string]
    Delete(ctx context.Context, id int64) Result[string]
}

// UseCase Interface
type NuevaEntidadUseCase interface {
    GetAll(ctx context.Context) Result[[]NuevaEntidad]
    GetByID(ctx context.Context, id int64) Result[NuevaEntidad]
    Create(ctx context.Context, nombre string, datos json.RawMessage)
Result[int64]
    Update(ctx context.Context, id int64, nombre string, datos
json.RawMessage) Result[string]
    Delete(ctx context.Context, id int64) Result[string]
}
```

9.1.3 Paso 3: Capa de Repositorio

Crear `repository/nueva_entidad_repository.go`:

```
package repository

type nuevaEntidadRepository struct {
    dal domain.DAL
}

func NewNuevaEntidadRepository(dal domain.DAL) domain.NuevaEntidadRepository
{
    return &nuevaEntidadRepository{dal: dal}
}

func (r *nuevaEntidadRepository) GetAll(ctx context.Context)
domain.Result[[]domain.NuevaEntidad] {
    entities, err := dal.QueryRows[domain.NuevaEntidad](
        r.dal,
```

```
        ctx,
        "fn_get_all_nueva_entidad",
    )

    if err != nil {
        return domain.Result[[]domain.NuevaEntidad]{
            Success: false,
            Code:    "ERR_DB_QUERY",
            Message: err.Error(),
        }
    }

    return domain.Result[[]domain.NuevaEntidad]{
        Success: true,
        Code:    "OK",
        Data:    entities,
    }
}

// Implementar otros métodos...
```

9.1.4 Paso 4: Capa de Use Case

Crear `usecase/nueva_entidad_usecase.go`:

```
package usecase

type nuevaEntidadUseCase struct {
    repo domain.NuevaEntidadRepository
}

func NewNuevaEntidadUseCase(repo domain.NuevaEntidadRepository)
domain.NuevaEntidadUseCase {
    return &nuevaEntidadUseCase{repo: repo}
}

func (uc *nuevaEntidadUseCase) GetAll(ctx context.Context)
domain.Result[[]domain.NuevaEntidad] {
    // Agregar lógica de negocio aquí si es necesario
    // Por ejemplo: cache, validaciones, transformaciones

    return uc.repo.GetAll(ctx)
}

func (uc *nuevaEntidadUseCase) Create(ctx context.Context, nombre string,
datos json.RawMessage) domain.Result[int64] {
    // Validaciones de negocio
```

```
    if nombre == "" {
        return domain.Result[int64]{
            Success: false,
            Code:    "ERR_VALIDATION",
            Message: "Nombre es requerido",
        }
    }

    return uc.repo.Create(ctx, nombre, datos)
}

// Implementar otros métodos...
```

9.1.5 Paso 5: Request DTOs

Crear `api/request/nueva_entidad.go`:

```
package request

type CreateNuevaEntidadRequest struct {
    BaseRequest
    Nombre string          `json:"nombre" validate:"required,min=3,max=100"`
    Datos  json.RawMessage `json:"datos" validate:"required"`
}

type GetNuevaEntidadByIDRequest struct {
    BaseRequest
    ID int64 `json:"id" validate:"required,min=1"`
}

type UpdateNuevaEntidadRequest struct {
    BaseRequest
    ID     int64           `json:"id" validate:"required,min=1"`
    Nombre string          `json:"nombre" validate:"required,min=3,max=100"`
    Datos  json.RawMessage `json:"datos" validate:"required"`
}

type DeleteNuevaEntidadRequest struct {
    BaseRequest
    ID int64 `json:"id" validate:"required,min=1"`
}
```

9.1.6 Paso 6: Router

Crear `api/route/nueva_entidad_router.go`:

```
package route
```

```go
import (
    "context"
    "github.com/danielgtaylor/api/v2"
    "api-chatbot/api/request"
    "api-chatbot/domain"
)

// Response types
type GetAllNuevaEntidadResponse struct {
    Body domain.Result[[]domain.NuevaEntidad]
}

type CreateNuevaEntidadResponse struct {
    Body domain.Result[int64]
}

func RegisterNuevaEntidadRoutes(apiAPI huma.API, useCase
domain.NuevaEntidadUseCase) {
    // GET ALL
    api.Register(humaAPI, huma.Operation{
        OperationID: "get-all-nueva-entidad",
        Method:      "POST",
        Path:        "/api/nueva-entidad/get-all",
        Summary:     "Get all nueva entidad",
        Tags:        []string{"NuevaEntidad"},
    }, func(ctx context.Context, input *struct {
        Body request.BaseRequest
    }) (*GetAllNuevaEntidadResponse, error) {
        result := useCase.GetAll(ctx)
        return &GetAllNuevaEntidadResponse{Body: result}, nil
    })

    // CREATE
    api.Register(humaAPI, huma.Operation{
        OperationID: "create-nueva-entidad",
        Method:      "POST",
        Path:        "/api/nueva-entidad/create",
        Summary:     "Create nueva entidad",
        Tags:        []string{"NuevaEntidad"},
    }, func(ctx context.Context, input *struct {
        Body request.CreateNuevaEntidadRequest
    }) (*CreateNuevaEntidadResponse, error) {
        result := useCase.Create(ctx, input.Body.Nombre, input.Body.Datos)
        return &CreateNuevaEntidadResponse{Body: result}, nil
    })
```

```
    // Agregar otros endpoints...
}
```

9.1.7 Paso 7: Registrar en main.go

Modificar `cmd/main.go`:

```
// Inicializar repository
nuevaEntidadRepo := repository.NewNuevaEntidadRepository(dal)

// Inicializar use case
nuevaEntidadUseCase := usecase.NewNuevaEntidadUseCase(nuevaEntidadRepo)

// Registrar routes
route.RegisterNuevaEntidadRoutes(apiAPI, nuevaEntidadUseCase)
```

9.1.8 Paso 8: Probar

```
# Aplicar migraciones
psql -U chatbot_user -d chatbot_db -f db/08_nueva_feature_tables.sql
psql -U chatbot_user -d chatbot_db -f db/09_nueva_feature_procedures.sql

# Recompilar y ejecutar
go build -o main cmd/main.go
./main

# Probar endpoint
curl -X POST http://localhost:8080/api/nueva-entidad/get-all \
  -H "Content-Type: application/json" \
  -d '{
    "idSession": "test",
    "idRequest": "550e8400-e29b-41d4-a716-446655440000",
    "process": "test",
    "idDevice": "test",
    "publicIp": "127.0.0.1",
    "dateProcess": "2025-10-27T10:00:00Z"
  }'
```

9.2 Mejores Prácticas

9.2.1 Nomenclatura

- **Archivos**: snake_case (`parameter_repository.go`)
- **Tipos**: PascalCase (`ParameterRepository`)
- **Funciones públicas**: PascalCase (`GetAll`)
- **Funciones privadas**: camelCase (`validateInput`)
- **Constantes**: UPPER_SNAKE_CASE (`MAX_RETRIES`)

9.2.2 Organización de Código

- Una interfaz por archivo en domain/
- Implementaciones privadas (lowercase struct)
- Constructores públicos `New*`
- Errores específicos por módulo

9.2.3 Manejo de Errores

```
// Retornar Result con código específico
if err != nil {
    return domain.Result[T]{
        Success: false,
        Code:    "ERR_SPECIFIC_CODE",
        Message: err.Error(),
    }
}

// Log de errores
logger.Error("Failed to process",
    "error", err,
    "context", additionalInfo)
```

9.2.4 Logging

```
import "api-chatbot/domain"

// Usar logger inyectado
logger := domain.GetLogger()

logger.Info("Operation started", "user", userID)
logger.Error("Operation failed", "error", err)
logger.Debug("Debug info", "data", data)
```

9.2.5 Contextos y Timeouts

```
// Siempre usar contextos
func (uc *useCase) DoSomething(ctx context.Context, params ...) Result {
    // Establecer timeout
    ctx, cancel := context.WithTimeout(ctx, 30*time.Second)
    defer cancel()

    // Pasar contexto a todas las llamadas
    result := uc.repo.Query(ctx, ...)

    return result
}
```

10 Despliegue

10.1 Despliegue con Docker

10.1.1 Dockerfile

```
FROM golang:1.25.1-alpine AS builder

WORKDIR /app

# Dependencias
COPY go.mod go.sum ./
RUN go mod download

# Código fuente
COPY . .

# Build optimizado
RUN CGO_ENABLED=0 GOOS=linux go build -ldflags="-w -s" -a -installsuffix cgo
-o main cmd/main.go

# Imagen final
FROM alpine:latest

RUN apk --no-cache add ca-certificates

WORKDIR /root/

COPY --from=builder /app/main .

EXPOSE 8080

CMD ["./main"]
```

10.1.2 docker-compose.yml

```
version: '3.8'

services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: chatbot_user
      POSTGRES_PASSWORD: your_password
      POSTGRES_DB: chatbot_db
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./db:/docker-entrypoint-initdb.d
```

```
    ports:
      - "5432:5432"
    networks:
      - chatbot_network

  api:
    build: .
    depends_on:
      - postgres
    ports:
      - "3434:8080"
    volumes:
      - ./config.json:/config/chatbot/config.json
      - ./logs:/logs
    environment:
      CONFIG_PATH: /config/chatbot/config.json
    networks:
      - chatbot_network
    restart: unless-stopped

volumes:
  postgres_data:

networks:
  chatbot_network:
    driver: bridge
```

10.1.3 deploy.sh

```
#!/bin/bash

echo "Building Docker image..."
docker build -t apigo-chatbot:latest .

echo "Stopping existing container..."
docker stop chatbot 2>/dev/null || true
docker rm chatbot 2>/dev/null || true

echo "Starting new container..."
docker run -d \
  --name chatbot \
  -p 3434:8080 \
  -v /path/to/config.json:/config/chatbot/config.json \
  -v /path/to/logs:/logs \
  --restart unless-stopped \
  apigo-chatbot:latest
```

```
echo "Deployment complete!"
docker logs -f chatbot
```

10.2 Variables de Entorno

```
# Archivo .env
DB_HOST=localhost
DB_PORT=5432
DB_USER=chatbot_user
DB_PASSWORD=your_password
DB_NAME=chatbot_db

JWT_SECRET=your-jwt-secret-key
OPENAI_API_KEY=your-openai-key
GROQ_API_KEY=your-groq-key

LOG_PATH=/var/log/chatbot
```

Cargar en config.json usando `${ENV_VAR}`:

```
{
  "Database": {
    "Host": "${DB_HOST}",
    "Password": "${DB_PASSWORD}"
  }
}
```

10.3 Monitoreo y Logs

10.3.1 Estructura de Logs

```
/logs/
  chatbot.log          # Log actual
  chatbot-2025-10-26.log  # Rotado
  chatbot-2025-10-25.log
```

10.3.2 Rotación de Logs

Configurado en `config.json`:

```
{
  "Log": {
    "Path": "/logs",
    "FileName": "chatbot.log",
    "MaxSize": 100,       // MB
    "MaxBackups": 5,      // Archivos
    "MaxAge": 30,         // Días
    "Compress": true
  }
}
```

10.3.3 Ver Logs en Tiempo Real

```
# Docker
docker logs -f chatbot

# Local
tail -f /logs/chatbot.log

# Filtrar errores
tail -f /logs/chatbot.log | grep ERROR
```

10.4 Backup de Base de Datos

```
#!/bin/bash
# backup.sh

DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups"
DB_NAME="chatbot_db"
DB_USER="chatbot_user"

mkdir -p $BACKUP_DIR

pg_dump -U $DB_USER -d $DB_NAME -F c -b -v -f "$BACKUP_DIR/
chatbot_db_$DATE.backup"

# Comprimir
gzip "$BACKUP_DIR/chatbot_db_$DATE.backup"

# Eliminar backups antiguos (>30 días)
find $BACKUP_DIR -name "*.backup.gz" -mtime +30 -delete

echo "Backup completed: chatbot_db_$DATE.backup.gz"
```

Configurar cron:

```
# Backup diario a las 2 AM
0 2 * * * /path/to/backup.sh
```

10.5 Restaurar Backup

```
#!/bin/bash
# restore.sh

BACKUP_FILE=$1
DB_NAME="chatbot_db"
DB_USER="chatbot_user"

if [ -z "$BACKUP_FILE" ]; then
    echo "Usage: ./restore.sh <backup_file.backup.gz>"
```

```
    exit 1
fi

# Descomprimir
gunzip -k $BACKUP_FILE

# Restaurar
pg_restore -U $DB_USER -d $DB_NAME -v ${BACKUP_FILE%.gz}

echo "Restore completed"
```

11 Referencia

11.1 Recursos Adicionales

11.1.1 Documentación Oficial

- **Go**: https://go.dev/doc/
- **PostgreSQL**: https://www.postgresql.org/docs/
- **Huma**: https://huma.rocks/
- **pgvector**: https://github.com/pgvector/pgvector
- **whatsmeow**: https://github.com/tulir/whatsmeow

11.1.2 Herramientas Útiles

- **pgAdmin**: GUI para PostgreSQL
- **Postman**: Cliente API REST
- **DBeaver**: IDE de bases de datos
- **Grafana**: Dashboards de monitoreo

11.2 Troubleshooting

11.2.1 Problema: No conecta a base de datos

Síntomas:

```
ERROR: failed to connect to database
```

Soluciones:

- Verificar que PostgreSQL esté corriendo: `systemctl status postgresql`
- Verificar credenciales en `config.json`
- Verificar firewall: `sudo ufw allow 5432`
- Verificar `pg_hba.conf` para permitir conexiones

11.2.2 Problema: pgvector no funciona

Síntomas:

```
ERROR: type "vector" does not exist
```

Solución:

```
\c chatbot_db
CREATE EXTENSION vector;
```

11.2.3 Problema: Embeddings fallan

Síntomas:

```
ERROR: failed to generate embedding
```

Soluciones:

- Verificar API key en `config.json`

- Verificar conectividad a API: `curl https://api.openai.com/v1/models`
- Verificar límites de rate limit
- Cambiar provider a Ollama si OpenAI no disponible

11.2.4 Problema: WhatsApp no conecta

Síntomas:

```
ERROR: failed to connect to WhatsApp
```

Soluciones:

- Generar nuevo QR code
- Verificar sesión no expirada
- Eliminar sesión antigua: `rm -rf whatsapp_session/`
- Reiniciar cliente

11.3 Glosario Técnico

API: Application Programming Interface

Clean Architecture: Patrón arquitectónico con separación en capas

DAL: Data Access Layer

DTO: Data Transfer Object

Embedding: Representación vectorial de texto

JWT: JSON Web Token

LLM: Large Language Model

Middleware: Función que procesa requests antes de handlers

pgvector: Extensión PostgreSQL para vectores

RAG: Retrieval Augmented Generation

Repository Pattern: Patrón de acceso a datos

REST: Representational State Transfer

Use Case: Lógica de negocio encapsulada

Vector Search: Búsqueda por similitud vectorial

11.4 Contacto y Soporte

Para soporte técnico o consultas:

Email: soporte-tecnico@ists.edu.ec

Repositorio: [URL del repositorio Git]

Documentación adicional: Carpeta `docs/` del proyecto

Este manual está sujeto a actualizaciones periódicas.
Consulte la versión más reciente en el repositorio del proyecto.