# PROMPT PARA REPLIT AGENT - G4 HUB SAAS

```markdown

# CONTEXT

I'm building G4 Hub, a multi-tenant SaaS platform that provides complete e-commerce automation for LATAM businesses. G4 Hub connects online stores (WooCommerce/Shopify) with ERP systems (Contífico) and automates the entire post-sale flow: inventory sync, automated invoicing, and logistics management.

This is a migration from an existing WordPress plugin (STCI) to a scalable SaaS platform that will serve hundreds of businesses across Latin America.

# PROJECT VISION

G4 Hub aims to be the "Shopify for LATAM automation" - a comprehensive platform that handles:

- \*\*Inventory Synchronization\*\*: Bidirectional sync between stores and ERPs

- \*\*Automated Invoicing\*\*: Generate invoices in Contífico with multi-country tax compliance

- \*\*Logistics Integration\*\*: Automated shipping with Servientrega and other LATAM carriers

- \*\*Public APIs\*\*: Extensible platform for third-party integrations

# CURRENT STATE

- Existing WordPress plugin with inventory sync functionality

- External license server (Node.js + SQLite) already built

- Need to transform into multi-tenant SaaS architecture

- Target: 500+ customers by Year 3 with $1.2M ARR

# GOAL - PHASE 1: SaaS FOUNDATION

Create the foundational multi-tenant SaaS infrastructure with authentication, database, and basic dashboard.

# TECHNICAL REQUIREMENTS

## Architecture Stack:

```yaml

Backend:

- Node.js 18+ with Express

- PostgreSQL 14+ with connection pooling

- Redis for caching and job queues

- JWT authentication with refresh tokens

- Docker containers for development

Frontend:

- Next.js 14 with App Router

- TypeScript for type safety

- Tailwind CSS + shadcn/ui components

- React Query for state management

- NextAuth.js for authentication

Infrastructure:

- Multi-tenant database architecture

- RESTful APIs with proper versioning

- Rate limiting per tenant

- Comprehensive error handling

- Structured logging

```

## Database Schema (PostgreSQL):

```sql

-- Core multi-tenant tables

CREATE TABLE tenants (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

subdomain VARCHAR(100) UNIQUE NOT NULL,

plan\_type VARCHAR(50) DEFAULT 'starter', -- 'starter' | 'professional' | 'enterprise'

status VARCHAR(20) DEFAULT 'active', -- 'active' | 'suspended' | 'cancelled'

settings JSONB DEFAULT '{}',

api\_key VARCHAR(255) UNIQUE NOT NULL,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE users (

id SERIAL PRIMARY KEY,

tenant\_id INTEGER REFERENCES tenants(id) ON DELETE CASCADE,

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

name VARCHAR(255) NOT NULL,

role VARCHAR(50) DEFAULT 'admin', -- 'admin' | 'user'

email\_verified BOOLEAN DEFAULT FALSE,

last\_login\_at TIMESTAMP,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE stores (

id SERIAL PRIMARY KEY,

tenant\_id INTEGER REFERENCES tenants(id) ON DELETE CASCADE,

platform VARCHAR(20) NOT NULL, -- 'woocommerce' | 'shopify'

store\_name VARCHAR(255) NOT NULL,

store\_url VARCHAR(500) NOT NULL,

api\_credentials JSONB NOT NULL, -- Encrypted credentials

sync\_config JSONB DEFAULT '{}',

status VARCHAR(20) DEFAULT 'active', -- 'active' | 'inactive' | 'error'

last\_sync\_at TIMESTAMP,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW(),

UNIQUE(tenant\_id, store\_url)

);

CREATE TABLE sync\_logs (

id SERIAL PRIMARY KEY,

tenant\_id INTEGER REFERENCES tenants(id) ON DELETE CASCADE,

store\_id INTEGER REFERENCES stores(id) ON DELETE CASCADE,

sync\_type VARCHAR(20) NOT NULL, -- 'pull' | 'push'

status VARCHAR(20) NOT NULL, -- 'success' | 'error' | 'partial'

synced\_count INTEGER DEFAULT 0,

error\_count INTEGER DEFAULT 0,

duration\_ms INTEGER,

details JSONB DEFAULT '{}',

error\_message TEXT,

created\_at TIMESTAMP DEFAULT NOW()

);

-- Indexes for performance

CREATE INDEX idx\_tenants\_subdomain ON tenants(subdomain);

CREATE INDEX idx\_users\_tenant\_id ON users(tenant\_id);

CREATE INDEX idx\_users\_email ON users(email);

CREATE INDEX idx\_stores\_tenant\_id ON stores(tenant\_id);

CREATE INDEX idx\_sync\_logs\_tenant\_store ON sync\_logs(tenant\_id, store\_id);

CREATE INDEX idx\_sync\_logs\_created\_at ON sync\_logs(created\_at DESC);

```

## Project Structure:

```

g4-hub-saas/

├── backend/

│ ├── src/

│ │ ├── app.js # Express app setup

│ │ ├── server.js # Entry point

│ │ ├── config/

│ │ │ ├── database.js # PostgreSQL connection

│ │ │ ├── redis.js # Redis connection

│ │ │ └── environment.js # Environment variables

│ │ ├── middleware/

│ │ │ ├── auth.js # JWT authentication

│ │ │ ├── tenant.js # Tenant isolation

│ │ │ ├── rateLimit.js # Rate limiting

│ │ │ ├── validation.js # Request validation

│ │ │ └── errorHandler.js # Global error handling

│ │ ├── models/

│ │ │ ├── Tenant.js # Tenant model (Sequelize)

│ │ │ ├── User.js # User model

│ │ │ ├── Store.js # Store model

│ │ │ └── SyncLog.js # Sync log model

│ │ ├── controllers/

│ │ │ ├── authController.js # Authentication endpoints

│ │ │ ├── tenantController.js # Tenant management

│ │ │ ├── userController.js # User management

│ │ │ └── storeController.js # Store management

│ │ ├── routes/

│ │ │ ├── auth.js # Auth routes

│ │ │ ├── tenants.js # Tenant routes

│ │ │ ├── users.js # User routes

│ │ │ └── stores.js # Store routes

│ │ ├── services/

│ │ │ ├── authService.js # Auth business logic

│ │ │ ├── encryptionService.js # Data encryption

│ │ │ └── emailService.js # Email notifications

│ │ └── utils/

│ │ ├── jwt.js # JWT utilities

│ │ ├── logger.js # Winston logging

│ │ ├── validation.js # Joi validation schemas

│ │ └── helpers.js # Common utilities

│ ├── tests/ # Jest tests

│ ├── package.json

│ └── Dockerfile

├── frontend/

│ ├── src/

│ │ ├── app/

│ │ │ ├── (auth)/

│ │ │ │ ├── login/page.tsx

│ │ │ │ ├── register/page.tsx

│ │ │ │ └── layout.tsx

│ │ │ ├── dashboard/

│ │ │ │ ├── page.tsx # Main dashboard

│ │ │ │ ├── stores/

│ │ │ │ │ ├── page.tsx # Store list

│ │ │ │ │ └── [id]/page.tsx # Store details

│ │ │ │ ├── settings/page.tsx

│ │ │ │ └── layout.tsx

│ │ │ ├── globals.css

│ │ │ ├── layout.tsx # Root layout

│ │ │ └── page.tsx # Landing page

│ │ ├── components/

│ │ │ ├── ui/ # shadcn/ui components

│ │ │ ├── auth/

│ │ │ │ ├── LoginForm.tsx

│ │ │ │ ├── RegisterForm.tsx

│ │ │ │ └── AuthLayout.tsx

│ │ │ ├── dashboard/

│ │ │ │ ├── Sidebar.tsx

│ │ │ │ ├── Header.tsx

│ │ │ │ ├── StoreCard.tsx

│ │ │ │ └── SyncStatus.tsx

│ │ │ └── common/

│ │ │ ├── LoadingSpinner.tsx

│ │ │ └── ErrorBoundary.tsx

│ │ ├── lib/

│ │ │ ├── auth.ts # NextAuth config

│ │ │ ├── api.ts # API client (Axios)

│ │ │ ├── utils.ts # Utility functions

│ │ │ └── validations.ts # Zod schemas

│ │ ├── hooks/

│ │ │ ├── useAuth.ts # Auth hook

│ │ │ ├── useStores.ts # Stores hook

│ │ │ └── useTenant.ts # Tenant hook

│ │ └── types/

│ │ └── index.ts # TypeScript types

│ ├── package.json

│ ├── tailwind.config.js

│ ├── next.config.js

│ └── Dockerfile

├── scripts/

│ ├── init-db.sql # Database initialization

│ ├── seed-data.sql # Sample data

│ └── migration-001.sql # Database migrations

├── docker/

│ └── docker-compose.yml # Development environment

├── docs/

│ ├── API.md # API documentation

│ └── DEPLOYMENT.md # Deployment guide

├── .env.example

├── README.md

└── package.json # Root package.json

```

## Key Features to Implement:

### 1. Multi-tenant Authentication System:

```javascript

// JWT with refresh token pattern

// Tenant isolation in all queries

// Role-based access control

// API key management for public APIs

// Rate limiting per tenant

```

### 2. Core API Endpoints:

```

POST /api/auth/register # Tenant registration

POST /api/auth/login # User login

POST /api/auth/refresh # Token refresh

GET /api/auth/me # Current user info

GET /api/tenants/current # Current tenant info

PUT /api/tenants/current # Update tenant

GET /api/tenants/usage # Usage statistics

GET /api/stores # List tenant stores

POST /api/stores # Create new store

GET /api/stores/:id # Get store details

PUT /api/stores/:id # Update store

DELETE /api/stores/:id # Delete store

POST /api/stores/:id/test # Test store connection

GET /api/sync/logs # Sync history

POST /api/sync/manual # Manual sync trigger

```

### 3. Frontend Dashboard Features:

```

- Tenant registration/login flow

- Main dashboard with key metrics

- Store management (CRUD)

- Real-time sync status

- Settings management

- Responsive design for mobile

- Dark/light mode toggle

- Loading states and error handling

```

## CRITICAL REQUIREMENTS:

### Security:

- All API credentials must be encrypted at rest

- Tenant data isolation (no cross-tenant data access)

- SQL injection prevention with parameterized queries

- XSS protection with input sanitization

- CORS configuration for frontend domain

- Rate limiting: 1000 requests/hour per tenant

### Performance:

- Database connection pooling (max 20 connections)

- Redis caching for frequently accessed data

- API response times < 200ms for 95th percentile

- Efficient pagination for large datasets

- Optimized database queries with proper indexes

### Error Handling:

- Comprehensive error logging with Winston

- Structured error responses with error codes

- Graceful error boundaries in React

- Retry logic for external API calls

- Health check endpoints for monitoring

## EXPECTED OUTPUT:

Please create this complete Phase 1 implementation with these files in priority order:

### 1. Backend Infrastructure:

- `backend/package.json` with all required dependencies

- `backend/src/config/database.js` - PostgreSQL with Sequelize

- `backend/src/config/redis.js` - Redis connection

- `backend/src/models/` - All Sequelize models with relationships

- `backend/src/middleware/auth.js` - JWT authentication middleware

- `backend/src/middleware/tenant.js` - Tenant isolation middleware

- `backend/src/controllers/authController.js` - Complete auth endpoints

- `backend/src/app.js` - Express app with all middleware

- `backend/src/server.js` - Server startup with error handling

### 2. Database Setup:

- `scripts/init-db.sql` - Complete schema with indexes

- `scripts/seed-data.sql` - Sample tenants and users for testing

### 3. Frontend Application:

- `frontend/package.json` - Next.js 14 with all dependencies

- `frontend/src/app/layout.tsx` - Root layout with providers

- `frontend/src/components/auth/LoginForm.tsx` - Complete login form

- `frontend/src/components/auth/RegisterForm.tsx` - Tenant registration

- `frontend/src/app/dashboard/page.tsx` - Main dashboard

- `frontend/src/lib/api.ts` - Axios client with auth interceptors

- `frontend/tailwind.config.js` - Tailwind with shadcn/ui

### 4. Development Environment:

- `docker/docker-compose.yml` - PostgreSQL + Redis + App services

- `.env.example` - All required environment variables

- `README.md` - Setup and development instructions

### 5. Testing Setup:

- Basic Jest test setup for backend

- Test examples for auth endpoints

- Database test utilities

## SUCCESS CRITERIA:

- ✅ User can register a new tenant account

- ✅ User can login and receive JWT tokens

- ✅ Dashboard loads with tenant-specific data

- ✅ All database queries include tenant isolation

- ✅ API rate limiting works correctly

- ✅ Docker environment starts without errors

- ✅ Frontend handles auth states properly

- ✅ All API responses follow consistent format

- ✅ Error handling works on both frontend and backend

## VALIDATION CHECKLIST:

- [ ] Can create tenant via registration form

- [ ] JWT tokens work with proper expiration

- [ ] Protected routes redirect to login when unauthorized

- [ ] Dashboard shows tenant-specific information

- [ ] Database queries are properly scoped by tenant\_id

- [ ] API rate limiting prevents abuse

- [ ] Environment variables are properly configured

- [ ] Docker containers start and connect successfully

Start with the database schema and models, then authentication system, followed by the basic API endpoints. Ensure tenant isolation is implemented from the beginning - every database query MUST include tenant\_id filtering.

Focus on creating a solid, secure foundation that can scale to handle thousands of tenants. The architecture should be clean, well-documented, and easy to extend for future phases.

```

\*\*Este prompt está optimizado para que el Replit Agent entienda completamente el contexto de G4 Hub y genere una base sólida para el SaaS. El Agent debería crear toda la infraestructura foundacional con autenticación multi-tenant, dashboard básico y configuración de desarrollo lista para usar.\*\*

```