Developer Manual

Table of Contents

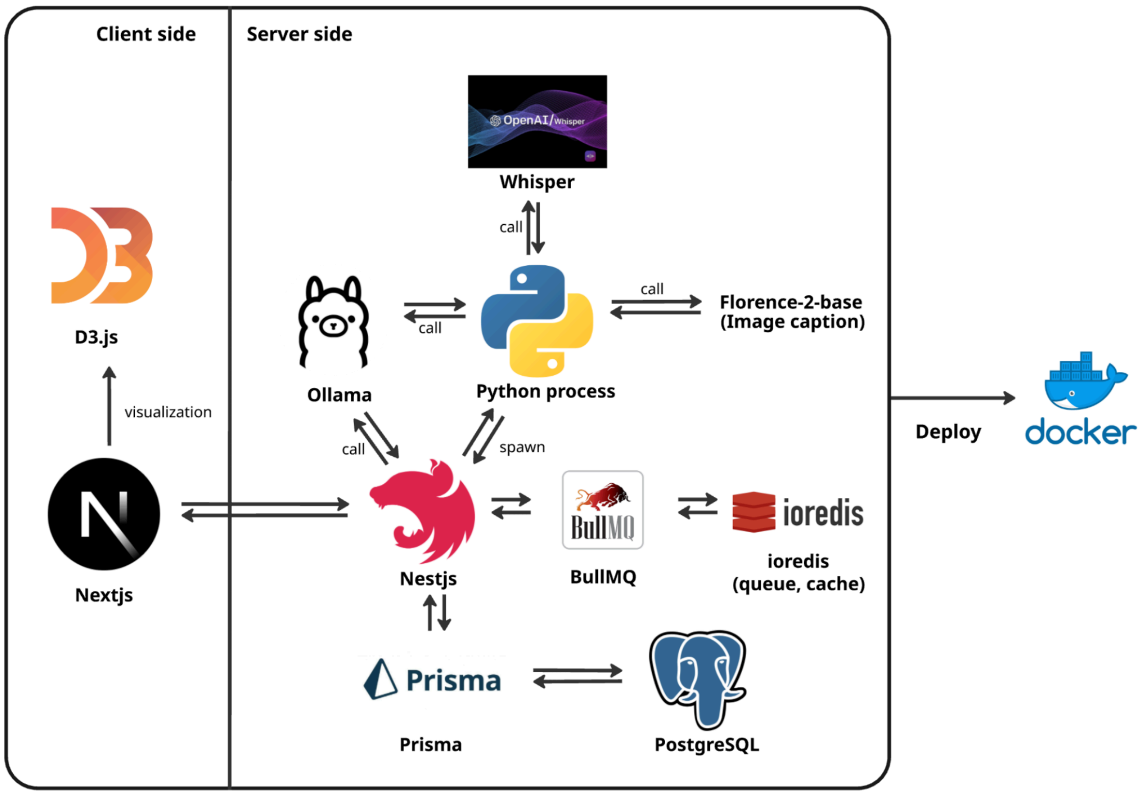
1. System Overview

This platform is a YouTube Video Summarization System that leverages Large Language Models (LLM) to:

- Summarize YouTube videos (speech-to-text ⟶ LLM summarization)
- Build an Ontology Graph of related topics extracted from video content
- Provide an interactive Hypertext experience where users click on topic words to explore related knowledge
- Support Chat with AI based on summarized content
- Include User Authentication with role-based access (Admin / User)

The system is composed of 4 backend microservices, 1 frontend application, and is orchestrated with Docker Compose.

2. Architecture Diagram



```
┌─────────────────────────────────────────────────────────────────
──┐
│                       Frontend (tmpfin)                         │
│                   Next.js · Port 8080                           │
└──────────────┬──────────────────┬──────────────────┬────────────
──┘
               │                  │                  │
               ▼                  ▼                  ▼
┌──────────────────┐  ┌──────────────────┐
┌──────────────────────────────────────┐
│ Auth-Service │ │  summarize      │ │  hypertext-onto-backend    │
│  Port 4005   │ │  _backend       │ │  (Ontology Service)        │
│  (JWT Auth)  │ │  Port 8081      │ │  Port 3001                 │
└──────────────────┘  └──────────┬───────┘
└────────────┬─────────────────────────┘
                         │                     │
                         │                     ▼
                         │          ┌──────────────────────────────────────┐
                         │          │  hypertext_backend           │
                         │          │  (LLM Processing)            │
                         │          │  Port 3002                   │
                         │          └────────────┬─────────────────────────┘
                         │                       │
                         ▼                       ▼
                 ┌──────────────────┐     ┌──────────────────┐
                 │   Redis       │      │  Ollama LLM │
                 │  Port 6379    │      │             │
                 └──────────────────┘     └──────────────────┘
                         │
                         ▼
               ┌─────────────────────────┐
               │  PostgreSQL        │
               │  (NeonDB Cloud)    │
               └─────────────────────────┘
```

3. Technology Stack

Layer	Technology
Runtime	Node.js 20
Framework	NestJS (all backend services)
Frontend	Next.js 15 + React 19 + TailwindCSS 4
Database	PostgreSQL (Neon serverless)
ORM	Prisma
Auth	JWT + Argon2 hashing + HTTP-only Cookies
LLM	Ollama (llama3:8b) via REST API
Queue	BullMQ + Redis
API Docs	Swagger (@nestjs/swagger)
Containerization	Docker + Docker Compose
CI/CD	GitHub Actions + GitHub Container Registry
Package Manager	npm / pnpm

4. Repository Overview

Repository	Role	Port	Package Manager
Auth-Service	Authentication & Authorization	4005	pnpm
summarize_backend	YouTube Summarization Pipeline	8081	npm
hypertext_backend	LLM Processing & Topic Graphs	3002	npm
hypertext-onto-backend	Ontology Graph Management	3001	npm
tmpfin	Frontend Web Application	8080	npm
final-deployment	Docker Compose Orchestration	—	—

5. Auth-Service

Purpose

Handles user registration, login, logout, token refresh, and user profile retrieval. Uses JWT tokens stored in HTTP-only cookies for security.

Directory Structure

```
Auth-Service/
├── src/
│   ├── main.ts                 # App bootstrap + Swagger setup
│   ├── app.module.ts           # Root module
│   ├── auth/
│   │   ├── auth.controller.ts  # Auth endpoints
│   │   ├── auth.service.ts     # Auth business logic
│   │   ├── auth.module.ts
│   │   ├── jwt.strategy.ts     # Passport JWT strategy
│   │   ├── jwt-auth.guard.ts   # Route guard
│   │   └── dto/
│   │       ├── login.dto.ts
│   │       └── register.dto.ts
│   ├── users/
│   │   ├── users.service.ts    # User CRUD operations
│   │   ├── users.module.ts
│   │   └── dto/
│   │       └── create-user.dto.ts
│   └── prisma/                 # Prisma ORM module
├── prisma/
│   └── schema.prisma
├── Dockerfile
└── package.json
```

API Endpoints

Method	Endpoint	Auth Required	Description
POST	/auth/register	No	Register a new user
POST	/auth/login	No	Login and receive cookies
POST	/auth/refresh	Cookie	Refresh access token via cookie
POST	/auth/logout	No	Clear auth cookies
GET	/auth/me	JWT / Cookie	Get current authenticated user info

Key Features

- Password Hashing: Argon2 for secure password storage
- JWT Tokens: Access token (7d expiry) + Refresh token (7d expiry) stored in HTTP-only cookies
- Role System: ADMIN and USER roles; admin can login as regular user
- User Color: Auto-generates consistent HSL color from user ID (for graph visualization)
- CORS: Configured for http://localhost:8080

Environment Variables

Variable	Description	Example
DATABASE_URL	PostgreSQL connection	postgresql://...
PORT	Server port	4005
JWT_SECRET	JWT signing secret	supersecretjwt

Local Development

```
cd Auth-Service
pnpm install
pnpm exec prisma generate
pnpm run start:dev
```

6. summarize_backend

Purpose

The core service of the platform. Handles YouTube video summarization by orchestrating a full pipeline: download → speech-to-text (Whisper) → scene captioning (Florence-2) → LLM summarization (Ollama). Also provides chat functionality and job queue management.

Directory Structure

```
summarize_backend/
├── src/
│   ├── main.ts                   # Bootstrap + Swagger (dev only)
│   ├── app.module.ts             # Root module
│   ├── auth/                     # JWT guard (validates Auth-Service tokens)
│   │   ├── jwt-auth.guard.ts
│   │   └── auth.module.ts
│   ├── summarize/
│   │   ├── summarize.controller.ts # Summary CRUD endpoints
│   │   ├── summarize.service.ts    # Summary business logic
│   │   ├── progress.controller.ts  # SSE progress streaming
│   │   ├── progress.service.ts
│   │   └── dto/
│   ├── chat/
│   │   ├── chat.controller.ts      # Chat endpoints + SSE streaming
│   │   ├── chat.service.ts         # Ollama-based chat
│   │   └── dto/
│   ├── queue/
│   │   ├── queue.controller.ts     # Queue management
│   │   ├── queue.service.ts        # BullMQ queue operations
│   │   └── queue.event.ts          # Queue event handling
│   ├── system-config/
│   │   ├── system-config.controller.ts  # System config (concurrency, uptime)
│   │   └── system-config.service.ts
│   ├── worker/
│   │   ├── index.ts                # Worker entry point
│   │   ├── processor.ts           # Job processor (Python pipeline)
│   │   └── worker-manager.ts
│   ├── cache/                     # Redis caching
│   └── shared/                    # Shared utilities
├── python/                        # Python scripts for ML pipeline
├── prisma/
│   └── schema.prisma
├── Dockerfile
└── Dockerfile.python-base         # Custom Python ML base image
```

API Endpoints

Summary Module

Method	Endpoint	Auth	Description
POST	/summary	JWT	Create a new summary job
GET	/summary	JWT	Get my summaries
GET	/summary/all	No	Get all summaries (with active workers)
GET	/summary/:id	No	Get summary by ID
GET	/summary/:id/ontology	No	Get keyword + summary for ontology
POST	/summary/:id/cancel	No	Cancel a summary job

Chat Module

Method	Endpoint	Auth	Description
POST	/chat	JWT	Create chat (non-streaming)
POST	/chat/stream	JWT	Create chat (SSE streaming)
GET	/chat/history/:summaryId	JWT	Get chat history by summary

Queue Module

Method	Endpoint	Auth	Description
POST	/queue/clear	No	Clear the job queue
GET	/queue/status	No	Get queue status

System Config Module

Method	Endpoint	Auth	Description
GET	/system-config/concurrency	No	Get concurrency level
POST	/system-config/concurrency	No	Set concurrency level
GET	/system-config/uptime	No	Get server uptime

Progress Module (SSE)

Method	Endpoint	Auth	Description
SSE	/jobs/:id/stream	No	Stream job progress via SSE

Key Features

- BullMQ Job Queue: Async processing with configurable concurrency via Redis
- Python ML Pipeline: Whisper ASR, Florence-2 scene captioning, Ollama summarization
- SSE Streaming: Real-time progress updates and chat responses
- Swagger: Available in non-production environments only (NODE_ENV !== 'production')
- Cookie-based Auth: Validates JWT tokens from Auth-Service

Key Environment Variables

Variable	Description	Default
DATABASE_URL	PostgreSQL connection string	—
PORT	Server port	8081
OLLAMA_API	Ollama API endpoint	http://ollama:11434/api/generate
OLLAMA_MODEL	LLM model name	llama3:8b
REDIS_HOST	Redis host	localhost
REDIS_PORT	Redis port	6379
PYTHON_BIN	Python binary path	python
ASR_DEVICE	Whisper device (cpu/cuda)	cpu
VL_DEVICE	Vision-Language device	cpu
WHISPER_MODEL	Whisper model variant	large-v3-turbo
BULL_CONCURRENCY	Max concurrent jobs	2
ONTOLOGY_SERVICE	Ontology service URL	http://ontology:3001

Worker Architecture

The summarize service runs in two modes:

15. API Server — node dist/src/main.js (handles HTTP requests)
16. Worker — node dist/src/worker/index.js (processes BullMQ jobs)

Both share the same Docker image (summarize_backend), differentiated by the command in Docker Compose.

Local Development

```
cd summarize_backend
npm install
npx prisma generate
npm run start:dev
```

7. hypertext_backend

Purpose

Handles LLM-powered topic extraction and hypertext graph building. When a user clicks on a keyword, this service generates related descriptions and topic graphs using Ollama.

Directory Structure

```
hypertext_backend/
├── src/
│   ├── main.ts                   # Bootstrap + Swagger
│   ├── app.module.ts             # Root module
│   ├── app.controller.ts        # Health checks
│   ├── hypertext/
│   │   ├── hypertext.controller.ts  # Hypertext CRUD
│   │   ├── hypertext.service.ts     # Topic + click handling
│   │   └── dto/
│   │       └── topic.dto.ts
│   ├── llm/
│   │   ├── llm.service.ts           # Ollama LLM integration
│   │   └── llm.module.ts
│   └── prisma/
├── prisma/
│   └── schema.prisma
└── Dockerfile
```

API Endpoints

Method	Endpoint	Description
GET	/	Hello message
GET	/health	System health check
GET	/health/ollama	Ollama connectivity check
POST	/hypertext/topic	Create topic + extract 10 related keywords (LLM)
POST	/hypertext/click	Handle hypertext click → generate description
GET	/hypertext/topic/:name	Get topic by name with relations
GET	/hypertext/topic-info/:name	Get main topic + top 10 related + description
GET	/hypertext/graph	Get all topics as graph (nodes + links)
GET	/hypertext/graph/user/:userId	Get user-specific topic graph

Key Features

- LLM Topic Extraction: Sends description to Ollama, extracts 10 related keywords
- Hypertext Click Handling: Generates new descriptions with caching (Redis)
- Graph Data: Returns nodes + links format for D3.js visualization
- Performance Logging: Detailed timing for DB lookups, cache hits, LLM calls
- Article Generation: Thai-language article summarization via LLM with strict formatting rules

Key Environment Variables

Variable	Description	Default
DATABASE_URL	PostgreSQL connection	—
PORT	Server port	3002
OLLAMA_API	Ollama API endpoint	http://localhost:11434/api/generate
OLLAMA_MODEL	LLM model name	llama3:8b
REDIS_HOST	Redis host	redis
REDIS_PORT	Redis port	6379

Local Development

```
cd hypertext_backend
npm install
npx prisma generate
npm run start:dev
```

8. hypertext-onto-backend

Purpose

Manages the Ontology Graph — the knowledge structure of all topics. Acts as a middleware between the frontend and hypertext_backend. Provides graph queries, user-topic associations, privacy-focused graph coloring, and data synchronization.

Directory Structure

```
hypertext-onto-backend/
├── src/
│   ├── main.ts                     # Bootstrap + Swagger
│   ├── app.module.ts               # Root module (with LoggingMiddleware)
│   ├── logging.middleware.ts       # HTTP request logging
│   ├── ontology/
│   │   ├── ontology.controller.ts     # Ontology CRUD (12 endpoints)
│   │   ├── ontology.service.ts        # Complex graph operations
│   │   ├── ontology.data.ts           # Demo/seed ontology data
│   │   └── dto/
│   │       ├── create-topic.dto.ts
│   │       ├── add-user-to-topic.dto.ts
│   │       ├── get-topic-id.dto.ts
│   │       ├── get-pedigree.dto.ts
│   │       └── get-users-colors.dto.ts
│   ├── hyperlink/
│   │   ├── hyperlink.controller.ts    # Static hyperlink data
│   │   ├── hyperlink.service.ts
│   │   └── hyperlink.data.ts          # Large hyperlink dataset
│   └── prisma/
├── prisma/
│   └── schema.prisma
└── Dockerfile
```

API Endpoints

Ontology Module

Method	Endpoint	Description
GET	/ontology	Get demo ontology data
GET	/ontology/node/:word	Find node in demo data
POST	/ontology/topic	Create/update topic (calls hypertext_backend LLM)
POST	/ontology/topic/assign-user	Assign user to a topic
POST	/ontology/topic/id	Get topic ID by name
POST	/ontology/click	Forward hypertext click to hypertext_backend
GET	/ontology/topics	Get all topics from DB as graph
GET	/ontology/world	Get world topics (with user details)
GET	/ontology/topics/user/:userId	Get topics for user (admin: blended colors)
GET	/ontology/topics/user/:userId/private	Get topics with privacy coloring
GET	/ontology/topic/:name	Get full topic details by name
GET	/ontology/pedigree/:name/:userId	Get all connected nodes (relatives)
POST	/ontology/users/colors	Get user colors for legends
POST	/ontology/sync	Sync topics from hypertext_backend

Hyperlink Module

Method	Endpoint	Description
GET	/hyperlink	Get all hyperlink keys
GET	/hyperlink/:word	Get hyperlink content

Key Features

- Request Logging Middleware: Logs method, URL, status, and duration
- Inter-service Communication: Forwards LLM requests to hypertext_backend
- Privacy Coloring: Different color schemes for exclusive vs. shared nodes
- Pedigree Queries: Discover all connected topics in the graph
- Sync Endpoint: Pull and merge data from hypertext_backend
- CORS: Configured for localhost:8080, 127.0.0.1:8080, localhost:3000

Key Environment Variables

Variable	Description	Default
DATABASE_URL	PostgreSQL connection	—
PORT	Server port	3001
HYPERTEXT_API_URL	hypertext_backend URL	http://hypertext-backend:3002

Local Development

```
cd hypertext-onto-backend
npm install
```

```
npx prisma generate
npm run start:dev
```

9. Frontend (tmpfin)

Purpose

The Next.js frontend that provides the user interface for the entire platform.

Tech Stack

- Next.js 15 with App Router
- React 19
- TailwindCSS 4 for styling
- D3.js for ontology graph visualization
- Framer Motion for animations
- Lucide React for icons
- React Toastify for notifications
- LangChain + Ollama for client-side AI features

Directory Structure

```
tmpfin/
├── app/
│   ├── page.tsx            # Main page (summary dashboard)
│   ├── layout.tsx          # Root layout
│   ├── globals.css         # Global styles
│   ├── login/              # Login page
│   ├── history/            # Summary history page
│   ├── admin/              # Admin dashboard
│   └── provider/           # Context providers
├── components/             # Reusable UI components (26 files)
├── contexts/               # React contexts
├── hooks/                  # Custom hooks
├── lib/                    # Utility libraries (10 files)
├── public/                 # Static assets
├── types/                  # TypeScript types
├── utils/                  # Utility functions
├── Dockerfile
└── package.json
```

Pages

Route	Description
/	Main dashboard — summarize videos
/login	User login page
/history	View past summaries
/admin	Admin panel (user management, config)

Key Environment Variables

Variable	Description
NEXT_PUBLIC_DEPLOY_ONTO_TEST	Ontology service URL
NEXT_PUBLIC_SUMMARY_SERVICE_ENDPOINT	Summary service URL
llmModel	Ollama model for client-side chat

Local Development

```
cd tmpfin
npm install
npm run dev
# → http://localhost:8080
```

10. Deployment (final-deployment)

Docker Compose Services

The docker-compose.yml defines the following services:

Service	Image Source	Port	Dependencies
redis	redis:7	6379	—
summarize	GHCR (CI/CD built)	8081	redis, ontology
summary_worker	Same as summarize	—	redis, ontology
hypertext-backend	GHCR (CI/CD built)	3002	redis
ontology	GHCR (CI/CD built)	3001	hypertext-backend
auth-service	GHCR (CI/CD built)	4005	redis

Service Startup Order

```
redis → hypertext-backend → ontology → summarize + summary_worker
                                   ↘ auth-service
```

Nginx Reverse Proxy (Optional)

An Nginx configuration is provided for API gateway routing:

Path	Forwards To
/summarize-service/	http://summarize:3000/
/ontology-service/	http://ontology:3000/
/	Health status message

CI/CD (GitHub Actions)

Two workflow files exist under .github/workflows/:

- ci.yml — Continuous Integration (build + test)
- docker.yml — Docker image build and push to GHCR

Running with Docker Compose

```
cd final-deployment

# Copy environment file
cp .env.example .env
# Edit .env with your actual values

# Start all services
docker compose up -d

# View logs
docker compose logs -f

# Stop all services
docker compose down
```

11. Database Schema

All services share the same PostgreSQL database (NeonDB). The shared schema includes:

Core Models

Model	Description
User	User accounts (id, username, password, role)
Summary	YouTube video summary jobs
SummaryOwner	Many-to-many: summary ↔ user ownership
OntologyTopic	Topic nodes in the knowledge graph
OntologyTopicRelation	Edges between topic nodes (with weight)
UserOntologyTopic	Many-to-many: user ↔ topic association
ChatSession	Chat sessions linked to summaries
ChatMessage	Individual chat messages (USER/ASSISTANT/SYSTEM)

Enums

Enum	Values
SummaryStatus	QUEUED, RUNNING, DONE, ERROR, CANCEL
UserRole	ADMIN, USER
ChatRole	USER, ASSISTANT, SYSTEM

Running Migrations

```
# Generate Prisma client
npx prisma generate

# Pull schema from database (development)
npx prisma db pull

# Open Prisma Studio (GUI)
npx prisma studio
```

12. Environment Variables

Complete `.env.example` (for final-deployment)

```
# Docker Registry
ORG=llm-summarization-project
SUMMARIZE_REPO=summarize_backend
ONTO_REPO=hypertext-onto-backend
HYPERTEXT_REPO=hypertext_backend
FRONTEND_REPO=tmpfinal
AUTH_REPO=auth-service
TAG=latest

# Database
DATABASE_URL="postgresql://user:pass@host:5432/dbname"

# Ollama LLM
OLLAMA_MODEL=llama3:8b
OLLAMA_API=http://ollama:11434/api/generate

# Summarize Backend
PORT=4001
PYTHON_BIN=python
ASR_DEVICE=cpu
VL_DEVICE=cpu
WHISPER_MODEL=large-v3-turbo
SCENE_THRESH=0.6
LANGUAGE=th
NODE_ENV=development
BULL_CONCURRENCY=2

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379

# Auth
JWT_SECRET=supersecretjwt

# Frontend
FRONTEND_ORIGIN=http://localhost:8080
NEXT_PUBLIC_DEPLOY_ONTO_TEST=http://localhost:80/ontology-service
NEXT_PUBLIC_SUMMARY_SERVICE_ENDPOINT=http://localhost:4001
```

13. API Documentation (Swagger)

All 4 backend services have Swagger integrated. Access the interactive API docs at:

Service	Swagger URL
Auth-Service	http://localhost:4005/swagger
summarize_backend	http://localhost:8081/swagger

hypertext_backend	http://localhost:3002/swagger
hypertext-onto-backend	http://localhost:3001/swagger

Note: summarize_backend only exposes Swagger when NODE_ENV !== 'production'.

Swagger Configuration Summary

Service	Package	swagger-ui-express	Bearer Auth	Cookie Auth
Auth-Service	@nestjs/swagger	⚠️ Not in deps	✅ Yes	✅ Yes
summarize_backend	@nestjs/swagger	✅ Yes	✅ Yes	❌ No
hypertext_backend	@nestjs/swagger	✅ Yes	❌ No	❌ No
hypertext-onto-backend	@nestjs/swagger	✅ Yes	❌ No	❌ No

14. Getting Started

Prerequisites

- Docker and Docker Compose
- Ollama with llama3:8b model pulled
- A PostgreSQL database (or NeonDB account)

Quick Start (Local Development)

All services are pre-built as Docker images and hosted on GitHub Container Registry (GHCR). The docker-compose.yml in final-deployment/ pulls these images automatically — no need to clone or build individual repositories.

1. Configure Environment

```
cd final-deployment

# Copy the example env and fill in your values
cp .env.example .env
```

Edit .env with your actual configuration:

```
# Required: your PostgreSQL connection string
DATABASE_URL="postgresql://user:pass@host:5432/dbname"

# Required: Ollama endpoint (use host machine IP if Ollama runs outside Docker)
OLLAMA_API=http://host.docker.internal:11434/api/generate
OLLAMA_MODEL=llama3:8b

# Required: JWT secret for auth
JWT_SECRET=supersecretjwt
```

2. Start Ollama (on host machine)

```
ollama serve
ollama pull llama3:8b
```

3. Pull and Start All Services

```
cd final-deployment

# Pull latest images from GHCR
docker compose pull

# Start all services in background
docker compose up -d
```

This single command starts all services:

Service	Container	Port
Redis	redis	6379
Auth Service	auth-service	4005
Hypertext Backend	hypertext-backend	3002
Ontology Service	ontology	3001
Summarize API	summarize	8081
Summarize Worker	summary_worker	—

4. Verify Services

```
# Check all containers are running
docker compose ps

# View logs
docker compose logs -f

# View logs for a specific service
docker compose logs -f summarize
```

5. Access the Application

Service	URL
Auth Swagger	http://localhost:4005/swagger
Summarize Swagger	http://localhost:8081/swagger
Hypertext Swagger	http://localhost:3002/swagger
Ontology Swagger	http://localhost:3001/swagger

6. Stop All Services

```
docker compose down
```

Source Code Development (Optional)

If you need to modify source code and run services locally without Docker:

```
# Auth Service
cd Auth-Service && pnpm install && pnpm exec prisma generate && pnpm run start:dev

# Hypertext Backend
cd hypertext_backend && npm install && npx prisma generate && npm run start:dev

# Ontology Service
cd hypertext-onto-backend && npm install && npx prisma generate && npm run
start:dev

# Summarize Backend
cd summarize_backend && npm install && npx prisma generate && npm run start:dev

# Frontend
cd tmpfin && npm install && npm run dev
```

Note: Running from source requires Node.js 20+, npm/pnpm, Redis, and Python 3.10+ (for the summarize worker's ML pipeline).