# HOLLY 3.0

Hyper-Optimized Logic & Learning Yield

# Developer Documentation

## Complete Technical Reference & API Guide

**Hollywood Productions**

Steve "Hollywood" Dorego

Generated: December 2025

# HOLLY Developer Documentation

## Complete Technical Guide v1.0

**For:** Developers, DevOps Engineers, System Architects
**Last Updated:** December 2025
**Version:** 1.0.0

---

## Table of Contents



---

# 1. Getting Started

## 1.1 Prerequisites

**Required:** - Node.js >= 18.0.0 - npm >= 9.0.0 - PostgreSQL >= 14.0 - Git

**Recommended:** - VS Code with extensions: - Prisma - ESLint - Prettier - TypeScript

## 1.2 Quick Start

```
# Clone the repository
git clone https://github.com/iamhollywoodpro/Holly-AI.git
cd Holly-AI

# Install dependencies
npm install

# Set up environment variables
cp .env.example .env.local
# Edit .env.local with your credentials

# Generate Prisma Client
npx prisma generate

# Push database schema
npx prisma db push

# Start development server
npm run dev

# Open http://localhost:3000
```

## 1.3 Environment Variables

Create `.env.local` with the following:

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

# Clerk Authentication
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY="pk_test_..."
CLERK_SECRET_KEY="sk_test_..."

# Clerk URLs
NEXT_PUBLIC_CLERK_SIGN_IN_URL="/sign-in"
NEXT_PUBLIC_CLERK_SIGN_UP_URL="/sign-up"
NEXT_PUBLIC_CLERK_AFTER_SIGN_IN_URL="/dashboard"
NEXT_PUBLIC_CLERK_AFTER_SIGN_UP_URL="/dashboard"

# Optional: External APIs
OPENAI_API_KEY="sk-..."
ANTHROPIC_API_KEY="sk-ant-..."
```

---

# 2. Project Structure

```
Holly-AI/
├── app/                          # Next.js App Router
│   ├── api/                      # API Routes
│   │   ├── analytics/            # Analytics APIs
│   │   ├── creative/             # Creative APIs
│   │   ├── security/             # Security APIs
│   │   ├── moderation/           # Moderation APIs
│   │   ├── compliance/           # Compliance APIs
│   │   ├── audit/                # Audit APIs
│   │   └── orchestration/        # Orchestration APIs
│   ├── dashboard/                # Dashboard Pages
│   │   ├── page.tsx             # Main dashboard
│   │   ├── creative/             # Creative studio
│   │   ├── analytics/            # Analytics dashboard
│   │   ├── security/             # Security center
│   │   └── orchestration/        # Orchestration hub
│   ├── (auth)/                   # Auth pages
│   ├── layout.tsx                # Root layout
│   └── page.tsx                  # Home page
│
├── src/
│   ├── components/               # React Components
│   │   ├── dashboard/            # Dashboard-specific
│   │   │   ├── ui/              # UI primitives
│   │   │   ├── charts/          # Chart components
│   │   │   ├── metrics/         # Metric displays
│   │   │   ├── layout/          # Layout components
│   │   │   ├── notifications/ # Notifications
│   │   │   └── workflow/        # Workflow builder
│   │   └── [feature]/           # Feature components
│   │
│   ├── lib/                      # Core Libraries
│   │   ├── analytics/            # Analytics Engine
│   │   │   ├── metrics-aggregator.ts
│   │   │   ├── report-generator.ts
│   │   │   ├── dashboard-builder.ts
│   │   │   └── insights-engine.ts
│   │   ├── creative/             # Creative Engine
│   │   │   ├── asset-manager.ts
│   │   │   ├── content-generator.ts
```

```
│   │   │   ├── image-generator.ts
│   │   │   └── template-manager.ts
│   │   ├── security/          # Security System
│   │   │   ├── audit-logger.ts
│   │   │   ├── security-monitor.ts
│   │   │   ├── content-moderator.ts
│   │   │   └── compliance-manager.ts
│   │   ├── orchestration/     # Orchestration Engine
│   │   │   ├── agent-coordinator.ts
│   │   │   ├── workflow-engine.ts
│   │   │   ├── task-scheduler.ts
│   │   │   └── resource-allocator.ts
│   │   ├── api/               # API Clients
│   │   │   ├── client.ts
│   │   │   ├── analytics.ts
│   │   │   ├── creative.ts
│   │   │   ├── security.ts
│   │   │   └── orchestration.ts
│   │   └── websocket/         # WebSocket
│   │       └── notifications.ts
│   │
│   ├── hooks/                 # React Hooks
│   │   ├── useAnalytics.ts
│   │   ├── useCreative.ts
│   │   ├── useSecurity.ts
│   │   └── useOrchestration.ts
│   │
│   └── utils/                 # Utility functions
│
├── prisma/
│   └── schema.prisma          # Database schema
│
├── public/                    # Static assets
│
├── docs/                      # Documentation
│   ├── HOLLY_WHITE_PAPER.md
│   ├── HOLLY_INVESTOR_PITCH.md
│   └── DEVELOPER_DOCUMENTATION.md
│
├── .env.local                 # Environment variables
```

```
├── next.config.mjs            # Next.js config
├── tailwind.config.ts         # Tailwind config
├── tsconfig.json              # TypeScript config
└── package.json               # Dependencies
```

---

# 3. Development Setup

## 3.1 Local Development

```
# Install dependencies
npm install

# Generate Prisma Client
npx prisma generate

# Start dev server (with hot reload)
npm run dev

# Run in production mode
npm run build
npm start

# Check TypeScript types
npm run type-check

# Lint code
npm run lint

# Format code
npm run format
```

## 3.2 Database Setup

### Option 1: Local PostgreSQL

```
# Install PostgreSQL (macOS)
brew install postgresql@14
brew services start postgresql@14

# Create database
createdb holly_dev

# Update .env.local
DATABASE_URL="postgresql://localhost:5432/holly_dev"
```

**Option 2: Neon (Recommended)**

1. Sign up at https://neon.tech
2. Create a new project
3. Copy connection string to `DATABASE_URL`

**Push Schema:**

```
# Push schema to database
npx prisma db push

# Generate Prisma Client
npx prisma generate

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

## 3.3 Authentication Setup

**Clerk Setup:**

1. Sign up at https://clerk.com
2. Create a new application
3. Copy API keys to `.env.local`: `env NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY="pk_test_..." CLERK_SECRET_KEY="sk_test_..."`
4. Configure sign-in/up URLs in Clerk Dashboard

---

# 4. Core Concepts

## 4.1 Architecture Overview

HOLLY follows a **layered architecture**:

```
┌─────────────────────────────────────────┐
│  PRESENTATION (React Components)        │
├─────────────────────────────────────────┤
│  HOOKS (React Hooks + State)            │
├─────────────────────────────────────────┤
│  API CLIENTS (HTTP Layer)               │
├─────────────────────────────────────────┤
│  API ROUTES (Next.js Endpoints)         │
├─────────────────────────────────────────┤
│  BUSINESS LOGIC (Core Libraries)        │
├─────────────────────────────────────────┤
│  DATA ACCESS (Prisma ORM)               │
├─────────────────────────────────────────┤
│  DATABASE (PostgreSQL)                  │
└─────────────────────────────────────────┘
```

## 4.2 Data Flow

**Frontend → Backend:**

```
// 1. Component calls hook
const { create, loading } = useCreativeAsset();

// 2. Hook calls API client
await create({ name: 'My Asset', type: 'image' });

// 3. API client makes HTTP request
const response = await apiClient.post('/creative/assets', data);

// 4. API route authenticates & validates
const { userId } = await auth();
if (!userId) return unauthorized();

// 5. Business logic processes request
const asset = await assetManager.createAsset(userId, data);

// 6. Prisma ORM persists to database
await prisma.creativeAsset.create({ data });

// 7. Response flows back up the chain
return NextResponse.json({ success: true, asset });
```

## 4.3 Type Safety

**HOLLY is 100% TypeScript:**

```
// Types are defined at database level (Prisma)
model CreativeAsset {
  id   String @id @default(cuid())
  name String
  type String
  // ...
}

// Generated by Prisma
import { CreativeAsset } from '@prisma/client';

// Used in API layer
export interface Asset extends CreativeAsset {
  // Additional fields
}

// Used in frontend
const asset: Asset = await getAsset(id);
```

---

# 5. API Reference

## 5.1 API Conventions

**Base URL:** `/api`
**Authentication:** Clerk session (automatic)
**Content-Type:** `application/json`
**Response Format:** JSON

**Standard Responses:**

```
// Success
{
  "success": true,
  "data": { /* ... */ }
}

// Error
{
  "error": "Error message",
  "code": "ERROR_CODE"
}
```

# 5.2 Creative APIs

## Generate Image

```
POST /api/creative/image/generate
```

**Request:**

```
{
  "prompt": "A futuristic cityscape at sunset",
  "model": "dall-e-3",
  "width": 1024,
  "height": 1024
}
```

**Response:**

```
{
  "success": true,
  "jobId": "job_abc123"
}
```

## List Assets

```
GET /api/creative/assets?type=image&limit=10
```

**Response:**

```
{
  "assets": [
    {
      "id": "asset_123",
      "name": "My Image",
      "type": "image",
      "url": "https://...",
      "createdAt": "2025-12-06T..."
    }
  ]
}
```

### Delete Asset

```
DELETE /api/creative/asset/[id]
```

**Response:**

```
{
  "success": true
}
```

## 5.3 Analytics APIs

### Create Metric

```
POST /api/analytics/metrics
```

**Request:**

```
{
  "name": "user_signups",
  "metricType": "counter",
  "category": "growth"
}
```

**Response:**

```
{
  "success": true,
  "metricId": "metric_abc123"
}
```

## Get Metrics

```
GET /api/analytics/metrics?category=growth&limit=10
```

**Response:**

```
{
  "metrics": [
    {
      "id": "metric_123",
      "name": "user_signups",
      "value": 1250,
      "trend": "up",
      "changePercent": 15.3
    }
  ]
}
```

## Generate Report

```
POST /api/analytics/reports
```

**Request:**

```
{
  "name": "Monthly Revenue Report",
  "type": "revenue",
  "config": {
    "dateRange": "last_30_days"
  }
}
```

**Response:**

```
{
  "success": true,
  "reportId": "report_abc123"
}
```

# 5.4 Security APIs

## Get Security Report

```
GET /api/security/report
```

**Response:**

```
{
  "securityScore": 98,
  "activeThreats": 0,
  "blockedRequests": 127,
  "recentEvents": [
    {
      "type": "rate_limit_exceeded",
      "severity": "medium",
      "timestamp": "2025-12-06T..."
    }
  ]
}
```

## Moderate Content

```
POST /api/moderation/check
```

**Request:**

```
{
  "content": "Text to moderate",
  "type": "text"
}
```

**Response:**

```
{
  "safe": true,
  "reason": null
}
```

# 5.5 Orchestration APIs

## Create Agent

```
POST /api/orchestration/agents
```

**Request:**

```
{
  "name": "Code Generator",
  "type": "creative",
  "capabilities": ["code_generation", "testing"]
}
```

**Response:**

```
{
  "success": true,
  "agentId": "agent_abc123"
}
```

## Execute Workflow

```
POST /api/orchestration/workflows/[id]/execute
```

**Response:**

```
{
  "success": true,
  "workflow": {
    "id": "workflow_123",
    "status": "running",
    "progress": 25
  }
}
```

---

# 6. Database Schema

## 6.1 Prisma Basics

**Schema Definition:**

```
model User {
  id          String   @id @default(cuid())
  clerkUserId String   @unique
  email       String   @unique
  name        String?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  // Relations
  assets CreativeAsset[]

  @@map("users")
}
```

**Generated TypeScript Type:**

```
type User = {
  id: string;
  clerkUserId: string;
  email: string;
  name: string | null;
  createdAt: Date;
  updatedAt: Date;
}
```

## 6.2 Common Queries

**Create:**

```
const user = await prisma.user.create({
  data: {
    clerkUserId: userId,
    email: 'user@example.com',
    name: 'John Doe'
  }
});
```

**Read:**

```
// Find unique
const user = await prisma.user.findUnique({
  where: { clerkUserId: userId }
});

// Find many
const assets = await prisma.creativeAsset.findMany({
  where: { userId: user.id },
  orderBy: { createdAt: 'desc' },
  take: 10
});
```

**Update:**

```
const updated = await prisma.user.update({
  where: { id: userId },
  data: { name: 'New Name' }
});
```

**Delete:**

```
await prisma.creativeAsset.delete({
  where: { id: assetId }
});
```

**Relations:**

```
// Include related data
const user = await prisma.user.findUnique({
  where: { id: userId },
  include: {
    assets: true,
    projects: true
  }
});
```

## 6.3 Migrations

**Development:**

```
# Make schema changes in schema.prisma

# Push changes (development only)
npx prisma db push

# Generate client
npx prisma generate
```

**Production:**

```
# Create migration
npx prisma migrate dev --name add_new_field

# Apply migrations (CI/CD)
npx prisma migrate deploy
```

---

# 7. Authentication

## 7.1 Clerk Integration

**Frontend (Client Component):**

```
'use client';

import { useUser } from '@clerk/nextjs';

export function ProfileButton() {
  const { user, isSignedIn } = useUser();

  if (!isSignedIn) {
    return <SignInButton />;
  }

  return <div>Hello, {user.firstName}</div>;
}
```

**Backend (API Route):**

```
import { auth } from '@clerk/nextjs/server';
import { NextRequest, NextResponse } from 'next/server';

export async function GET(req: NextRequest) {
  const { userId } = await auth();

  if (!userId) {
    return NextResponse.json(
      { error: 'Unauthorized' },
      { status: 401 }
    );
  }

  // User is authenticated
  const data = await fetchUserData(userId);
  return NextResponse.json(data);
}
```

## 7.2 Protected Routes

**Middleware:**

```
// middleware.ts
import { clerkMiddleware } from '@clerk/nextjs/server';

export default clerkMiddleware();

export const config = {
  matcher: [
    '/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|d
    '/(api|trpc)(.*)',
  ],
};
```

**Page-Level Protection:**

```
// app/dashboard/page.tsx
import { auth } from '@clerk/nextjs/server';
import { redirect } from 'next/navigation';

export default async function DashboardPage() {
  const { userId } = await auth();

  if (!userId) {
    redirect('/sign-in');
  }

  return <Dashboard userId={userId} />;
}
```

---

# 8. Frontend Development

## 8.1 Component Structure

**Example Component:**

```
'use client';

import { useState } from 'react';
import { Card, CardHeader, CardTitle, CardContent } from '@/components/ui/Card';
import { Button } from '@/components/ui/Button';
import { useCreativeAsset } from '@/hooks/useCreative';

export function AssetList() {
  const { assets, loading, error, deleteAsset } = useCreativeAsset();
  const [selectedId, setSelectedId] = useState<string | null>(null);

  if (loading) {
    return <LoadingSpinner />;
  }

  if (error) {
    return <ErrorMessage error={error} />;
  }

  return (
    <Card>
      <CardHeader>
        <CardTitle>Assets</CardTitle>
      </CardHeader>
      <CardContent>
        <div className="grid gap-4">
          {assets.map((asset) => (
            <AssetCard
              key={asset.id}
              asset={asset}
              onDelete={() => deleteAsset(asset.id)}
            />
          ))}
        </div>
      </CardContent>
    </Card>
  );
}
```

## 8.2 Custom Hooks

**Pattern:**

```
// src/hooks/useResource.ts
import { useState, useEffect, useCallback } from 'react';
import * as api from '@/lib/api/resource';

export function useResource() {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const fetchData = useCallback(async () => {
    setLoading(true);
    setError(null);
    try {
      const result = await api.getResources();
      setData(result);
    } catch (err: any) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => {
    fetchData();
  }, [fetchData]);

  const create = useCallback(async (newData: any) => {
    const result = await api.createResource(newData);
    setData(prev => [...prev, result]);
    return result;
  }, []);

  const deleteResource = useCallback(async (id: string) => {
    await api.deleteResource(id);
    setData(prev => prev.filter(item => item.id !== id));
  }, []);

  return {
    data,
    loading,
```

```
    error,
    fetchData,
    create,
    delete: deleteResource
  };
}
```

## 8.3 Styling with Tailwind

**Best Practices:**

```
// Use className prop
<div className="flex items-center gap-4 rounded-lg bg-white p-4 shadow">
  <h2 className="text-xl font-semibold text-gray-900">Title</h2>
  <p className="text-sm text-gray-600">Description</p>
</div>

// Responsive design
<div className="grid grid-cols-1 gap-4 md:grid-cols-2 lg:grid-cols-3">
  {/* Content */}
</div>

// Hover states
<button className="rounded-lg bg-purple-600 px-4 py-2 text-white hover:bg-purple-700 acti
  Click Me
</button>

// Dark mode (if implemented)
<div className="bg-white dark:bg-gray-800 text-gray-900 dark:text-white">
  Content
</div>
```

---

# 9. Backend Development

## 9.1 Creating API Routes

**File Structure:**

```
app/api/resource/route.ts          # GET /api/resource, POST /api/resource
app/api/resource/[id]/route.ts     # GET /api/resource/:id, PATCH, DELETE
```

**Example Route:**

```
// app/api/resource/route.ts
import { auth } from '@clerk/nextjs/server';
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/prisma';

// GET /api/resource
export async function GET(req: NextRequest) {
  try {
    const { userId } = await auth();
    if (!userId) {
      return NextResponse.json(
        { error: 'Unauthorized' },
        { status: 401 }
      );
    }

    const resources = await prisma.resource.findMany({
      where: { userId }
    });

    return NextResponse.json({ resources });
  } catch (error) {
    console.error('GET /api/resource error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}

// POST /api/resource
export async function POST(req: NextRequest) {
  try {
    const { userId } = await auth();
    if (!userId) {
      return NextResponse.json(
        { error: 'Unauthorized' },
        { status: 401 }
      );
    }
```

```
    const body = await req.json();

    // Validate input
    if (!body.name) {
      return NextResponse.json(
        { error: 'Name is required' },
        { status: 400 }
      );
    }

    const resource = await prisma.resource.create({
      data: {
        userId,
        name: body.name,
        type: body.type || 'default'
      }
    });

    return NextResponse.json({
      success: true,
      resource
    });
  } catch (error) {
    console.error('POST /api/resource error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}
```

## 9.2 Business Logic Libraries

**Pattern:**

```
// src/lib/feature/manager.ts
import { prisma } from '@/lib/prisma';

export class ResourceManager {
  /**
   * Get resources for a user
   */
  async getUserResources(userId: string) {
    return prisma.resource.findMany({
      where: { userId },
      orderBy: { createdAt: 'desc' }
    });
  }

  /**
   * Create a new resource
   */
  async createResource(userId: string, data: {
    name: string;
    type: string;
  }) {
    // Business logic validation
    if (data.name.length < 3) {
      throw new Error('Name must be at least 3 characters');
    }

    return prisma.resource.create({
      data: {
        userId,
        ...data
      }
    });
  }

  /**
   * Delete a resource
   */
  async deleteResource(userId: string, resourceId: string) {
    // Check ownership
    const resource = await prisma.resource.findUnique({
```

```
      where: { id: resourceId }
    });

    if (!resource || resource.userId !== userId) {
      throw new Error('Resource not found or unauthorized');
    }

    return prisma.resource.delete({
      where: { id: resourceId }
    });
  }
}

// Singleton instance
export const resourceManager = new ResourceManager();
```

---

# 10. Testing

## 10.1 Unit Tests

**Setup:**

```
npm install --save-dev jest @testing-library/react @testing-library/jest-dom
```

**Example Test:**

```
// src/lib/utils/calculator.test.ts
import { calculate } from './calculator';

describe('Calculator', () => {
  test('adds two numbers', () => {
    expect(calculate(2, 3, 'add')).toBe(5);
  });

  test('handles division by zero', () => {
    expect(() => calculate(5, 0, 'divide')).toThrow('Division by zero');
  });
});
```

## 10.2 Integration Tests

**Example:**

```
// __tests__/api/resource.test.ts
import { GET, POST } from '@/app/api/resource/route';
import { NextRequest } from 'next/server';

describe('Resource API', () => {
  test('GET returns resources', async () => {
    const req = new NextRequest('http://localhost/api/resource');
    const response = await GET(req);
    const data = await response.json();

    expect(response.status).toBe(200);
    expect(data).toHaveProperty('resources');
  });
});
```

## 10.3 E2E Tests

**Using Playwright:**

```
// tests/e2e/dashboard.spec.ts
import { test, expect } from '@playwright/test';

test('dashboard loads correctly', async ({ page }) => {
  await page.goto('http://localhost:3000/dashboard');

  await expect(page.locator('h1')).toContainText('Dashboard');
  await expect(page.locator('[data-testid="metric-card"]')).toHaveCount(4);
});
```

---

# 11. Deployment

## 11.1 Vercel Deployment

**Automatic (Recommended):**

1. Push code to GitHub
2. Import project in Vercel
3. Configure environment variables
4. Deploy automatically on push to main

**Manual:**

```
# Install Vercel CLI
npm i -g vercel

# Login
vercel login

# Deploy
vercel

# Deploy to production
vercel --prod
```

## 11.2 Environment Variables

**Add in Vercel Dashboard:** - Settings → Environment Variables - Add all variables from `.env.local` - Redeploy after adding variables

## 11.3 Database Migrations

**Production Migration:**

```
# In vercel-build script (package.json)
"vercel-build": "prisma db push --accept-data-loss && prisma generate && next build"
```

**Note:** Use `prisma migrate deploy` for production migrations with proper migration history.

---

# 12. Troubleshooting

## 12.1 Common Issues

**Issue: Prisma Client not generated**

```
# Solution
npx prisma generate
```

**Issue: Database connection failed**

```
# Check DATABASE_URL in .env.local
# Test connection
npx prisma db push
```

**Issue: Authentication not working**

```
# Verify Clerk keys in .env.local
# Check Clerk Dashboard settings
# Clear browser cookies
```

**Issue: Build fails on Vercel**

```
# Check build logs
# Ensure all env vars are set
# Run build locally: npm run build
```

## 12.2 Debug Mode

**Enable verbose logging:**

```
// Add to api routes
console.log('[DEBUG]', { userId, data });
```

**Prisma query logging:**

```
// prisma/client.ts
const prisma = new PrismaClient({
  log: ['query', 'info', 'warn', 'error']
});
```

## 12.3 Performance Optimization

**Database:** - Add indexes for frequently queried fields - Use `select` to limit returned fields - Implement pagination

**Frontend:** - Use React.memo for expensive components - Implement virtual scrolling for large lists - Lazy load images and components

**API:** - Implement caching (React Query) - Use SWR for real-time data - Optimize database queries

---

# 13. Best Practices

## 13.1 Code Organization

 **DO:** - One component per file - Group related files in folders - Use index files for exports - Keep components under 300 lines

 **DON'T:** - Mix business logic in components - Create deeply nested folder structures - Use relative imports ( `../../../` )

## 13.2 TypeScript

✅ **DO:** - Define explicit return types - Use interfaces for objects - Leverage type inference - Use generics when appropriate

❌ **DON'T:** - Use `any` type - Disable type checking - Ignore TypeScript errors

## 13.3 Security

✅ **DO:** - Validate all user input - Use parameterized queries - Implement rate limiting - Log security events

❌ **DON'T:** - Trust client-side data - Expose sensitive data - Skip authentication checks - Store secrets in code

---

# 14. Resources

## Official Documentation

- Next.js: https://nextjs.org/docs
- Prisma: https://www.prisma.io/docs
- Clerk: https://clerk.com/docs
- Tailwind CSS: https://tailwindcss.com/docs

## Community

- GitHub Issues: Report bugs
- Discord: Community support
- Twitter: Updates and announcements

## Contact

**Technical Support:** [Support Email]
**Documentation:** [Docs URL]
**GitHub:** https://github.com/iamhollywoodpro/Holly-AI

---

**Document Version:** 1.0
**Last Updated:** December 2025
**Maintained By:** HOLLY Development Team

---

This documentation is continuously updated. Check the repository for the latest version.