Payload CMS v3 Integration Progress

This document tracks the progress of integrating Payload CMS v3 with SQLite into the existing Next.js 15.4.5 application.

Project Overview

- **Project Path**: `/home/ubuntu/ptnextjs`
- **Next.js Version**: 15.4.5
- **Target CMS**: Payload CMS v3
- **Database**: SQLite
- **Integration Goal**: Headless CMS for content management with static site performance

Integration Steps Progress

✅ Step 1: Set Up Payload CMS v3 with SQLite

Status: ✅ COMPLETED
Completion Date: 2025-01-28

Tasks Completed:
- [x] Created `/cms` folder at project root
- [x] Initialized Node.js app inside `/cms` with `npm init -y`
- [x] Installed required dependencies: `payload@3.49.1`, `@payloadcms/db-sqlite@3.49.1`, `express@5.1.0`
- [x] Installed development dependencies: `typescript`, `@types/node`, `ts-node`
- [x] Created `payload.config.ts` with minimal SQLite configuration
- [x] Configured Payload to use SQLite with filename `./payload.db`
- [x] Created server setup file (`server.mjs`) for running Payload
- [x] Set up TypeScript configuration (`tsconfig.json`)
- [x] Created environment configuration (`.env.example`)
- [x] Verified all dependencies are properly installed

Files Created:
- `/cms/package.json` - Node.js package configuration
- `/cms/payload.config.ts` - Payload CMS configuration with SQLite adapter
- `/cms/server.mjs` - Express server setup for Payload admin
- `/cms/tsconfig.json` - TypeScript configuration
- `/cms/.env.example` - Environment variables template
- `/cms/.gitignore` - Git ignore for CMS-specific files

Technical Implementation:
- **Database**: SQLite adapter configured to use `./payload.db`
- **Admin Panel**: Accessible at `http://localhost:3001/admin`
- **API Endpoint**: Available at `http://localhost:3001/api`
- **Authentication**: Basic user collection with email/password auth
- **TypeScript**: Full TypeScript support with proper type generation

Next Steps Ready:
- Basic user collection configured for admin access
- SQLite database adapter properly configured
- Server setup ready for development and testing
- Ready to add custom collections in Step 2

✅ Step 2: Define Content Collections

Status: ✅ COMPLETED
Completion Date: 2025-01-28

Tasks Completed:
- [x] Created comprehensive Blog Posts collection with all required fields
- [x] Created Partners collection with complete company information
- [x] Created Products collection with partner relationships
- [x] Created Categories collection for content organization
- [x] Created Authors collection for blog post attribution
- [x] Created Media collection for file uploads with image optimization
- [x] Enhanced Users collection with role-based access control
- [x] Defined proper field schemas matching existing TypeScript interfaces
- [x] Set up relationships between collections (Products → Partners, Posts → Authors & Categories)
- [x] Configured drafts and versioning for content management workflow

Collections Created:
- **Blog Posts** (`blog-posts`): title, slug, excerpt, content (richText), author (relationship), publishedAt, category (relationship), tags (array), image (upload), featured, readTime
- **Partners** (`partners`): name, category (relationship), description, logo (upload), website, founded, location, tags (array), featured
- **Products** (`products`): name, partner (relationship), category (relationship), description, image (upload), features (array), price, tags (array)
- **Categories** (`categories`): name, slug, description, type (blog/product/partner), color
- **Authors** (`authors`): name, role, bio, image (upload), email, linkedin, twitter
- **Media** (`media`): Upload collection with image optimization (thumbnail, card, tablet sizes)
- **Users** (`users`): Enhanced with role-based access (admin, editor, author)

Technical Features:
- Perfect field mapping from existing TypeScript interfaces to Payload field types
- Maintained compatibility with existing dynamic routes (`/blog/[slug]` , `/products/[id]` , `/partners/[id]`)
- Public read access for all content collections
- Comprehensive admin interface with useful default columns and descriptions

---

✅ Step 3: Add Dev-Only Payload Startup Logic

Status: ✅ COMPLETED
Completion Date: 2025-01-28

Tasks Completed:
- [x] Created development-only startup script at `scripts/start-payload-dev.ts`

- [x] Implemented environment check (`NODE_ENV === 'development'`) to conditionally start Payload
- [x] Added Express server setup that works alongside Next.js dev server
- [x] Configured different port allocation (3001 for Payload, 3000 for Next.js)
- [x] Added proper error handling and logging for development workflow
- [x] Created helper scripts for running both servers simultaneously
- [x] Added required Payload dependencies to main project
- [x] Ensured production builds remain unaffected

Files Created:
- `scripts/start-payload-dev.ts` - Development-only Payload CMS startup script
- `scripts/dev-with-cms.sh` - Helper script to run both Next.js and Payload CMS
- `scripts/start-cms-only.sh` - Helper script to run only Payload CMS

Technical Implementation:
- **Environment Safety**: Script only runs when `NODE_ENV === 'development'`
- **Port Management**: Payload CMS runs on port 3001, Next.js on port 3000
- **Dynamic Config Loading**: Imports CMS configuration from `/cms/payload.config.ts`
- **Process Management**: Proper SIGINT/SIGTERM handling for graceful shutdown
- **Development Experience**: Clear logging and helpful startup messages

Usage Commands:

```
# Run both Next.js and Payload CMS together
./scripts/dev-with-cms.sh

# Run only Payload CMS (for CMS development)
./scripts/start-cms-only.sh

# Run only Next.js (standard development)
npm run dev

# Run Payload CMS directly with tsx
npx tsx scripts/start-payload-dev.ts
```

Production Safety:
- Script automatically exits if not in development mode
- No impact on `npm run build` or production deployments
- Dependencies added only affect development workflow

⏳ Step 4: Set Up Payload Admin Interface

Status: ⏳ PENDING
Target Date: TBD

Planned Tasks:
- [ ] Configure admin panel access
- [ ] Set up authentication
- [ ] Create admin user account
- [ ] Test CRUD operations in admin interface

⏳ Step 5: Create Data Migration Scripts

Status: ⏳ PENDING
Target Date: TBD

Planned Tasks:
- [ ] Create migration script for existing blog data
- [ ] Create migration script for existing partner data
- [ ] Create migration script for existing product data
- [ ] Test data migration integrity
- [ ] Backup existing static data

⏳ Step 6: Integrate Payload with Next.js API Routes

Status: ⏳ PENDING
Target Date: TBD

Planned Tasks:
- [ ] Create API routes for fetching CMS data
- [ ] Replace static data imports with CMS API calls
- [ ] Maintain existing API structure for compatibility
- [ ] Test data fetching in development environment

⏳ Step 7: Update Next.js Components

Status: ⏳ PENDING
Target Date: TBD

Planned Tasks:
- [ ] Update blog components to use CMS data
- [ ] Update partner components to use CMS data
- [ ] Update product components to use CMS data
- [ ] Maintain static generation compatibility
- [ ] Test all pages and components

⏳ Step 8: Testing and Final Integration

Status: ⏳ PENDING
Target Date: TBD

Planned Tasks:
- [ ] Test complete integration workflow
- [ ] Verify static generation still works
- [ ] Performance testing and optimization
- [ ] Create documentation for content management
- [ ] Deploy and test in production environment

Technical Notes

Current Project Structure

```
/home/ubuntu/ptnextjs/
├── app/                      # Next.js app directory
├── components/               # React components
├── lib/                     # Utility functions and data
├── data/                    # Static data files (to be migrated)
├── cms/                     # 🆕 Payload CMS directory
│   ├── payload.config.ts    # CMS configuration
│   ├── package.json         # CMS dependencies
│   └── payload.db           # SQLite database (generated)
└── [other Next.js files]
```

Dependencies Added

- `payload` - Core Payload CMS
- `@payloadcms/db-sqlite` - SQLite adapter
- `express` - Server framework for Payload

Configuration Highlights

- SQLite database: `./payload.db`
- Admin panel: Will be accessible at `/admin`
- API endpoint: Will be at `/api`
- TypeScript configuration included

Next Steps

1. **Immediate**: Begin Step 2 - Define Content Collections
2. **Priority**: Focus on maintaining existing static site performance
3. **Testing**: Ensure no breaking changes to current functionality

---

Last Updated: 2025-01-28
Next Review: After Step 2 completion