📝 Blog Application - Complete Documentation

Version: 1.0
Date: August 2025
Author: GitHub Copilot
Repository: https://github.com/gh0st-bit/blog-site

📋 Table of Contents

🎯 Project Overview

The **Blog Application** is a modern, full-stack web application built with Next.js that allows users to create, read, update, and delete blog posts. It features a clean, responsive design and supports both local development and production deployment through Docker containerization.

Key Highlights

- **Full CRUD Operations** - Complete blog post management
- **Responsive Design** - Works on desktop, tablet, and mobile
- **Database Flexibility** - MongoDB with intelligent fallback to mock database
- **Production Ready** - Docker containerization with health checks
- **Type Safety** - Built with TypeScript for better code quality
- **Modern UI** - Tailwind CSS for styling

✨ Features

🔧 Core Functionality

- **Create Posts** - Rich text editor for writing blog posts
- **View Posts** - Clean, readable post display with preview modal
- **Edit Posts** - In-place editing with form validation
- **Delete Posts** - Safe deletion with confirmation prompts
- **Responsive Layout** - Mobile-first design approach

🎨 User Interface

- **Modern Design** - Clean, professional appearance
- **Interactive Elements** - Hover effects and smooth transitions
- **Loading States** - Visual feedback during operations
- **Error Handling** - User-friendly error messages
- **Modal Previews** - Full-screen post preview functionality

🔒 Technical Features

- **Type Safety** - TypeScript for error prevention
- **Auto-Save** - Form data persistence
- **Smart Fallbacks** - Automatic database switching
- **Health Monitoring** - Container health checks
- **Cross-Platform** - Windows, macOS, and Linux support

🛠 Technology Stack

Frontend

```
Framework: Next.js 15 (App Router)
Language: TypeScript
Styling: Tailwind CSS
State Management: React Hooks (useState, useEffect)
```

Backend

```
API: Next.js API Routes
Runtime: Node.js 18+
Database: MongoDB (with mock fallback)
ODM: Native MongoDB Driver
```

Development & Deployment

```
Container: Docker & Docker Compose
Package Manager: npm
Version Control: Git
CI/CD: GitHub Actions
```

Key Dependencies

```json
{
  "next": "15.1.4",
  "react": "^19.0.0",
  "mongodb": "^6.12.0",
  "tailwindcss": "^3.4.17",
  "typescript": "^5.7.2"
}
```

🏗 Architecture

Application Structure

```
blog-site/
├── src/
│   ├── app/                    # Next.js App Router
│   │   ├── api/posts/         # API endpoints
│   │   ├── layout.tsx         # Root layout
│   │   └── page.tsx           # Main page
│   ├── components/            # React components
│   │   ├── PostForm.tsx       # Create/Edit form
│   │   ├── PostList.tsx       # Posts display
│   │   └── BlogView.tsx       # Post preview modal
│   └── lib/                   # Utilities
│       ├── mongodb.ts         # Database connection
│       └── mockdb.ts          # Mock database
├── docs/                      # Documentation
├── .github/workflows/         # CI/CD pipelines
├── docker-compose.yml         # Container orchestration
├── Dockerfile                 # Container definition
└── README.md                  # Project documentation
```

Data Flow

```
User Interface (React)
        ↓
API Routes (Next.js)
        ↓
Database Layer (MongoDB/Mock)
        ↓
Data Storage (MongoDB/Memory)
```

Component Hierarchy

```
App (page.tsx)
├── PostForm (Create/Edit)
├── PostList (Display posts)
│   └── BlogView (Preview modal)
└── API Integration (CRUD operations)
```

🚀 Installation Guide

Prerequisites

```
# Required software
Node.js 18+
Docker (optional but recommended)
Git
```

Quick Start Options

Option 1: One-Liner Setup (Recommended)

Windows:

```
git clone https://github.com/gh0st-bit/blog-site && cd blog-site && start.bat
```

Mac/Linux:

```
git clone https://github.com/gh0st-bit/blog-site && cd blog-site && chmod +x start.sh &&
./start.sh
```

Option 2: Manual Installation

```
# 1. Clone repository
git clone https://github.com/gh0st-bit/blog-site
cd blog-site

# 2. Install dependencies
npm install

# 3. Setup environment
cp .env.example .env.local

# 4. Start development server
npm run dev
```

Option 3: Docker Setup

Legacy Docker (most Linux systems):

```
git clone https://github.com/gh0st-bit/blog-site
cd blog-site
docker-compose up --build -d
```

Modern Docker (newer systems):

```
git clone https://github.com/gh0st-bit/blog-site
cd blog-site
docker compose up --build -d
```

Option 4: Auto-Detection

```
git clone https://github.com/gh0st-bit/blog-site
cd blog-site
chmod +x docker-command-helper.sh
./docker-command-helper.sh
```

📖 Usage Instructions

Accessing the Application

After successful installation, access these URLs:

Service	URL	Credentials
Blog Application	http://localhost:3000	-
MongoDB Admin	http://localhost:8081	admin / admin123
Database	localhost:27017	-

Creating Blog Posts

1. **Click "Create New Post"** - Opens the creation form
2. **Enter Title** - Write your blog post title
3. **Add Content** - Write your blog post content
4. **Click "Publish"** - Saves the post to database

Managing Posts

Viewing Posts:

- All posts display on the main page
- Click "View" to see full post in modal
- Posts show title and truncated content

Editing Posts:

- Click "Edit" on any post
- Modify title and content
- Click "Save Changes" to update

Deleting Posts:

- Click "Delete" on any post
- Confirm deletion in the prompt
- Post is permanently removed

User Interface Elements

Main Interface:

- Header with application title
- "Create New Post" button
- Grid of existing posts
- Search and filter options

Post Form:

- Title input field
- Content textarea
- Save/Cancel buttons
- Form validation

Post Display:

- Title and excerpt
- Action buttons (View, Edit, Delete)
- Responsive card layout
- Loading states

📡 API Documentation

Base URL

```
Local Development: http://localhost:3000/api
Production: https://your-domain.com/api
```

Endpoints

GET /api/posts

Description: Retrieve all blog posts

Request:

```
GET /api/posts
Content-Type: application/json
```

Response:

```
[
  {
    "_id": "507f1f77bcf86cd799439011",
    "title": "My First Blog Post",
    "content": "This is the content of my first blog post...",
    "createdAt": "2025-08-10T10:00:00Z"
  }
]
```

POST /api/posts

Description: Create a new blog post

Request:

```
POST /api/posts
Content-Type: application/json

{
  "title": "New Blog Post",
  "content": "Content of the new blog post..."
}
```

Response:

```
{
  "acknowledged": true,
  "insertedId": "507f1f77bcf86cd799439012"
}
```

GET /api/posts/[id]

Description: Retrieve a specific blog post

Request:

```
GET /api/posts/507f1f77bcf86cd799439011
```

Response:

```
{
  "_id": "507f1f77bcf86cd799439011",
  "title": "My First Blog Post",
  "content": "This is the content...",
  "createdAt": "2025-08-10T10:00:00Z"
}
```

PUT /api/posts/[id]

Description: Update an existing blog post

Request:

```
PUT /api/posts/507f1f77bcf86cd799439011
Content-Type: application/json
```

```
{
  "title": "Updated Blog Post Title",
  "content": "Updated content..."
}
```

Response:

```json
{
  "acknowledged": true,
  "modifiedCount": 1
}
```

DELETE /api/posts/[id]

Description: Delete a blog post

Request:

```
DELETE /api/posts/507f1f77bcf86cd799439011
```

Response:

```json
{
  "acknowledged": true,
  "deletedCount": 1
}
```

Error Responses

```json
{
  "error": "Error message description",
  "status": 400
}
```

Common status codes:

- `200` - Success
- `400` - Bad Request
- `404` - Not Found
- `500` - Internal Server Error

🗄️ Database Schema

Post Collection

```
{
  _id: ObjectId,           // MongoDB unique identifier
  title: String,           // Blog post title (required)
  content: String,         // Blog post content (required)
  createdAt: Date,         // Creation timestamp
  updatedAt: Date          // Last update timestamp
}
```

Sample Document

```
{
  "_id": "507f1f77bcf86cd799439011",
  "title": "Getting Started with React",
  "content": "React is a popular JavaScript library for building user interfaces. In this
post, we'll explore the basics of React and how to get started with your first component...",
  "createdAt": "2025-08-10T10:00:00.000Z",
  "updatedAt": "2025-08-10T11:30:00.000Z"
}
```

Database Fallback Strategy

The application implements intelligent database switching:

1. **Primary:** Attempts MongoDB connection
2. **Fallback:** Uses in-memory mock database
3. **Seamless:** No user-facing differences

Mock Database Features:

- In-memory storage
- Same API interface
- Pre-populated sample data
- Automatic ID generation

🐳 Docker Setup

Container Architecture

```
┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐
│   Blog App      │  │   MongoDB       │  │  Mongo Express  │
│   (Next.js)     │  │   (Database)    │  │  (Admin UI)     │
│   Port: 3000    │  │   Port: 27017   │  │  Port: 8081     │
└─────────────────┘  └─────────────────┘  └─────────────────┘
        │                      │                      │
        └──────────────────────┼──────────────────────┘
                               │
                     ┌───────────────┐
                     │   Docker      │
                     │   Network     │
                     └───────────────┘
```

Services Configuration

App Service:

```
app:
  build: .
  ports: ["3000:3000"]
  environment:
    - MONGODB_URI=mongodb://mongo:27017/blog-db
    - NODE_ENV=production
  depends_on: [mongo]
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:3000"]
    interval: 30s
    timeout: 10s
    retries: 3
```

MongoDB Service:

```
mongo:
  image: mongo:7-jammy
  ports: ["27017:27017"]
  volumes: [mongo-data:/data/db]
  environment:
    - MONGO_INITDB_DATABASE=blog-db
  healthcheck:
    test: ["CMD", "mongosh", "--eval", "db.adminCommand('ping')"]
```

Mongo Express Service:

```
mongo-express:
  image: mongo-express:latest
  ports: ["8081:8081"]
  environment:
    - ME_CONFIG_MONGODB_SERVER=mongo
    - ME_CONFIG_BASICAUTH_USERNAME=admin
    - ME_CONFIG_BASICAUTH_PASSWORD=admin123
```

Docker Commands Reference

Starting Services:

```
# Build and start all services
docker-compose up --build -d

# Start without building
docker-compose up -d

# Start with logs visible
docker-compose up --build
```

Managing Services:

```
# Check service status
docker-compose ps

# View logs
docker-compose logs -f

# Stop services
docker-compose down

# Stop and remove volumes
docker-compose down -v
```

Debugging:

```
# View specific service logs
docker-compose logs app
docker-compose logs mongo

# Execute commands in containers
docker-compose exec app sh
docker-compose exec mongo mongosh
```

🔧 Troubleshooting

Common Issues & Solutions

Docker Permission Issues (Linux)

Problem: `permission denied while trying to connect to the Docker daemon`

Solution:

```
# Add user to docker group
sudo usermod -aG docker $USER

# Apply changes
newgrp docker

# Or restart session
logout && login
```

Port Already in Use

Problem: `port is already allocated`

Solution:

```
# Stop conflicting services
docker-compose down

# Check what's using the port
sudo lsof -i :3000
sudo lsof -i :27017

# Kill specific processes
sudo kill -9 <PID>
```

Database Connection Failed

Problem: Application can't connect to MongoDB

Solution:

1. **Check MongoDB status:**

   ```
   docker-compose logs mongo
   ```

2. **Restart MongoDB:**

```
docker-compose restart mongo
```

3. **Use mock database:**
   - Application automatically falls back
   - No configuration needed

Build Failures

Problem: Docker build fails

Solution:

```
# Clean Docker cache
docker system prune -f

# Rebuild from scratch
docker-compose down -v
docker-compose up --build --force-recreate
```

Missing Environment Files

Problem: `ERROR: Couldn't find env file`

Solution:

```
# Create missing file
echo "MONGODB_URI=mongodb://mongo:27017/blog-db" > .env.docker

# Or run setup script
./setup.sh
```

System-Specific Issues

Windows Issues

- **Docker Desktop not running:** Start Docker Desktop application
- **WSL2 problems:** Update WSL2 kernel
- **Path issues:** Use PowerShell as Administrator

macOS Issues

- **Docker permission:** Install Docker Desktop
- **Port binding:** Check for conflicting services

- **M1 chip:** Use `--platform linux/amd64` flag

Linux Issues

- **Docker not installed:** Install Docker and Docker Compose
- **Systemd issues:** Start Docker service manually
- **AppArmor conflicts:** Configure Docker security profiles

Performance Optimization

Memory Usage

```
# Check container memory usage
docker stats

# Limit container memory
docker-compose up --memory=512m
```

Disk Space

```
# Clean unused Docker resources
docker system prune -a

# Remove unused volumes
docker volume prune
```

Network Issues

```
# Reset Docker networks
docker network prune

# Recreate networks
docker-compose down && docker-compose up
```

💻 Development Guide

Development Workflow

Setting up Development Environment

```bash
# Clone repository
git clone https://github.com/gh0st-bit/blog-site
cd blog-site

# Install dependencies
npm install

# Setup environment
cp .env.example .env.local

# Start development server
npm run dev
```

File Structure Explanation

```
src/
├── app/                    # Next.js App Router
│   ├── layout.tsx         # Root layout component
│   ├── page.tsx           # Main page component
│   └── api/               # API endpoints
│       └── posts/         # Posts API routes
│           ├── route.ts   # GET, POST /api/posts
│           └── [id]/      # Dynamic routes
│               └── route.ts # GET, PUT, DELETE /api/posts/[id]
├── components/            # Reusable React components
│   ├── PostForm.tsx       # Create/edit form component
│   ├── PostList.tsx       # Posts listing component
│   └── BlogView.tsx       # Post preview modal
└── lib/                   # Utility libraries
    ├── mongodb.ts         # MongoDB connection
    └── mockdb.ts          # Mock database implementation
```

TypeScript Integration

Type Definitions:

```typescript
type Post = {
  _id?: string;
  title: string;
  content: string;
};
```

Component Props:

```
interface PostFormProps {
  post?: Post;
  onSave: () => void;
  onCancel?: () => void;
}
```

API Response Types:

```
interface ApiResponse {
  data?: any;
  error?: string;
  status: number;
}
```

Adding New Features

Creating a New Component:

```
// src/components/NewComponent.tsx
'use client';

import { useState } from 'react';

interface NewComponentProps {
  // Define props here
}

export default function NewComponent({ }: NewComponentProps) {
  return (
    <div>
      {/* Component JSX */}
    </div>
  );
}
```

Adding API Endpoints:

```
// src/app/api/new-endpoint/route.ts
import { NextResponse } from 'next/server';

export async function GET() {
  try {
    // Logic here
    return NextResponse.json({ success: true });
  } catch (error) {
    return NextResponse.json(
      { error: 'Failed to process request' },
      { status: 500 }
    );
  }
}
```

Code Quality Standards

ESLint Configuration:

```
{
  "extends": ["next/core-web-vitals"],
  "rules": {
    "@typescript-eslint/no-explicit-any": "error",
    "@typescript-eslint/no-unused-vars": "warn"
  }
}
```

Prettier Configuration:

```
{
  "semi": true,
  "trailingComma": "es5",
  "singleQuote": true,
  "printWidth": 80
}
```

Testing Strategy

Unit Testing Setup

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

# Create test file
touch src/components/__tests__/PostForm.test.tsx
```

Example Test

```
import { render, screen } from '@testing-library/react';
import PostForm from '../PostForm';
```

```
describe('PostForm', () => {
  it('renders form elements', () => {
    render(<PostForm onSave={() => {}} />);

    expect(screen.getByLabelText(/title/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/content/i)).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /save/i })).toBeInTheDocument();
  });
});
```

🚀 Deployment

Production Deployment Options

Option 1: Docker Production Deployment

```
# Build production images
docker-compose -f docker-compose.prod.yml up --build -d

# Scale services
docker-compose up --scale app=3 -d
```

Option 2: Vercel Deployment

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

# Deploy to Vercel
vercel --prod
```

Option 3: Traditional Server Deployment

```
# Build application
npm run build

# Start production server
npm start
```

```
# Use process manager
pm2 start npm --name "blog-app" -- start
```

Environment Configuration

Production Environment Variables:

```
# .env.production
MONGODB_URI=mongodb://production-server:27017/blog-db
NODE_ENV=production
NEXT_PUBLIC_API_URL=https://your-domain.com
```

Docker Production:

```
# .env.docker
MONGODB_URI=mongodb://mongo:27017/blog-db
NODE_ENV=production
NEXT_PUBLIC_API_URL=http://localhost:3000
```

Security Considerations

Database Security:

- Use strong MongoDB passwords
- Enable authentication
- Configure network restrictions
- Regular backup procedures

Application Security:

- Input validation and sanitization
- CORS configuration
- Rate limiting implementation
- HTTPS enforcement

Container Security:

- Use non-root users
- Minimal base images
- Regular security updates
- Network segmentation

Monitoring & Logging

Health Checks:

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:3000/api/health"]
  interval: 30s
  timeout: 10s
  retries: 3
```

Logging Configuration:

```
logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "3"
```

Monitoring Commands:

```
# Check application logs
docker-compose logs -f app

# Monitor resource usage
docker stats

# Health check status
docker-compose ps
```

📋 Command Reference

Development Commands

```
npm run dev          # Start development server
npm run build        # Build for production
npm start           # Start production server
npm run lint        # Run ESLint
npm run test        # Run tests
```

Docker Commands

Legacy Docker (docker-compose):

```
docker-compose up --build -d    # Build and start
docker-compose down             # Stop services
docker-compose logs -f          # View logs
docker-compose ps               # Check status
docker-compose restart app      # Restart specific service
```

Modern Docker (docker compose):

```
docker compose up --build -d    # Build and start
docker compose down             # Stop services
docker compose logs -f          # View logs
docker compose ps               # Check status
docker compose restart app      # Restart specific service
```

Git Commands

```
git clone https://github.com/gh0st-bit/blog-site
git pull origin main            # Get latest changes
git add .                       # Stage changes
git commit -m "message"         # Commit changes
git push origin main            # Push changes
```

System Commands

```
# Check running processes
ps aux | grep node
ps aux | grep docker

# Check port usage
netstat -tulpn | grep :3000
lsof -i :3000

# System resources
htop                            # Process monitor
df -h                          # Disk usage
free -h                        # Memory usage
```

🏁 Conclusion

The Blog Application represents a modern, full-stack web development approach combining:

- **Frontend Excellence:** React with TypeScript for type-safe, maintainable code
- **Backend Simplicity:** Next.js API routes for seamless full-stack development
- **Database Flexibility:** MongoDB with intelligent fallback strategies
- **Deployment Ready:** Docker containerization for consistent environments
- **Developer Experience:** Comprehensive tooling and documentation

Key Achievements

✅ **Production Ready** - Fully containerized with health checks
✅ **Cross Platform** - Works on Windows, macOS, and Linux
✅ **Type Safe** - TypeScript throughout the application
✅ **Responsive Design** - Mobile-first approach with Tailwind CSS
✅ **Smart Fallbacks** - Graceful degradation when services unavailable
✅ **Comprehensive Documentation** - Complete setup and usage guides

Next Steps

- **Performance Optimization:** Implement caching strategies
- **Feature Enhancement:** Add user authentication and authorization
- **Testing Coverage:** Implement comprehensive test suites
- **Monitoring:** Add application performance monitoring
- **SEO Optimization:** Implement meta tags and structured data

---

📞 Support & Resources

Repository Information

- **GitHub:** https://github.com/gh0st-bit/blog-site
- **Issues:** https://github.com/gh0st-bit/blog-site/issues
- **Documentation:** Available in `/docs` folder

Quick Links

- **Local Application:** http://localhost:3000
- **Database Admin:** http://localhost:8081 (admin/admin123)
- **API Documentation:** http://localhost:3000/api/posts

Documentation Files

- `README.md` - Quick start guide
- `INSTALL.md` - Installation instructions
- `DOCKER_COMMANDS_FIXED.md` - Docker troubleshooting
- `TYPESCRIPT_EXPLANATION.md` - TypeScript information

Generated by GitHub Copilot | August 2025