**4. Project Design**

# 目录

4. Project Design ........................ 2

I. Architectural Design ........................ 3

1. Architectural Model ........................ 4

2. Architectural Patterns ........................ 5

3. Subsystem Decomposition ........................ 9

4. Component and Connector View ........................ 19

5. Deployment Architecture ........................ 21

6. Data Architecture ........................ 22

7. Architectural Strategies ........................ 23

8. Architectural Decisions ........................ 28

9. Quality Attribute Analysis ........................ 29

10. Architecture Evolution ........................ 30

*[在Microsoft Word中，您可以右键点击上方目录区域并选择'更新域'来自动生成准确的页码]*

# **4. Project Design**

## **I. Architectural Design**

This document outlines the architectural design of the ArtisanConnect platform following Ian Sommerville’s approach to software architecture. The design is presented with clarity and simplicity to ensure understanding of the system’s structure and components.

### **1. Architectural Model**

The ArtisanConnect platform follows a **layered architecture** with **client-server** pattern, implemented using the **Next.js App Router** framework. This architecture separates concerns while enabling both server-side and client-side rendering for optimal performance and SEO.

**mermaid**

flowchart TD

    subgraph "Client Layer"

        UI["UI Components"]

        Hooks["React Hooks"]

        ClientUtils["Client Utilities"]

    end

    subgraph "Server Layer"

        AppRouter["Next.js App Router"]

        ServerComponents["Server Components"]

        APIRoutes["API Routes"]

        Middleware["Middleware"]

    end

    subgraph "Service Layer"

        GlobalAPI["Global API Service"]

        AuthService["Authentication Service"]

        EmailService["Email Service"]

        MongoAPI["MongoDB API"]

    end

    subgraph "Data Layer"

        MongoDB["MongoDB"]

        HygraphCMS["Hygraph CMS"]

        Cloudinary["Cloudinary Storage"]

    end

    UI <--> Hooks

    UI <--> ClientUtils

    Hooks <--> AppRouter

    ClientUtils <--> AppRouter

    AppRouter <--> ServerComponents

    AppRouter <--> APIRoutes

    AppRouter <--> Middleware

    ServerComponents <--> GlobalAPI

    APIRoutes <--> GlobalAPI

    APIRoutes <--> AuthService

    APIRoutes <--> EmailService

    APIRoutes <--> MongoAPI

    GlobalAPI <--> HygraphCMS

    MongoAPI <--> MongoDB

    EmailService <--> "External Email Provider"

    AuthService <--> "NextAuth"

    GlobalAPI <--> Cloudinary

    classDef clientLayer fill:#e3f2fd,stroke:#1565c0,stroke-width:1px

    classDef serverLayer fill:#e8f5e9,stroke:#2e7d32,stroke-width:1px

    classDef serviceLayer fill:#fff8e1,stroke:#ff8f00,stroke-width:1px

    classDef dataLayer fill:#f3e5f5,stroke:#7b1fa2,stroke-width:1px

    class UI,Hooks,ClientUtils clientLayer

    class AppRouter,ServerComponents,APIRoutes,Middleware serverLayer

    class GlobalAPI,AuthService,EmailService,MongoAPI serviceLayer

    class MongoDB,HygraphCMS,Cloudinary dataLayer

### **2. Architectural Patterns**

The ArtisanConnect platform incorporates several architectural patterns:

#### **2.1 Model-View-Controller (MVC)**

• **Model**: MongoDB schemas and Hygraph CMS content models

• **View**: React components and Next.js pages

• **Controller**: API routes and server components

#### **2.2 Service-Oriented Architecture**

Services are organized as independent modules with specific responsibilities:

• **GlobalApi.js**: Handles communication with Hygraph CMS

• **ApiService.js**: Manages internal API calls

• **EmailService.js**: Handles email notifications

• **MongoApi.js**: Interfaces with MongoDB

#### **2.3 Middleware Architecture**

The system uses middleware for cross-cutting concerns:

• **Authentication**: Verifies user identity

• **Authorization**: Enforces role-based access control

• **Logging**: Records system events

• **Error Handling**: Provides graceful error responses

### **3. Subsystem Decomposition**

The system is decomposed into the following major subsystems:

![](data:None;base64,)

*mermaid chart*

#### **3.1 User Management Subsystem**

• **Responsibility**: Manages user accounts, profiles, and preferences

• **Key Components**: User model, profile pages, settings

• **Interfaces**: User API endpoints, authentication hooks

#### **3.2 Authentication Subsystem**

• **Responsibility**: Handles user authentication and authorization

• **Key Components**: NextAuth integration, JWT handling, middleware

• **Interfaces**: Login/logout endpoints, session management

#### **3.3 Search & Discovery Subsystem**

• **Responsibility**: Enables users to find services and providers

• **Key Components**: Search filters, category browsing, location-based search

• **Interfaces**: Search API, category listing

#### **3.4 Booking Management Subsystem**

• **Responsibility**: Handles the creation and management of service bookings

• **Key Components**: Calendar, booking form, booking history

• **Interfaces**: Booking API endpoints

#### **3.5 Provider Management Subsystem**

• **Responsibility**: Manages service provider profiles and services

• **Key Components**: Provider registration, business profile, service management

• **Interfaces**: Provider API endpoints

#### **3.6 Review System Subsystem**

• **Responsibility**: Handles customer reviews and ratings

• **Key Components**: Review form, rating display, review moderation

• **Interfaces**: Review API endpoints

#### **3.7 Admin Dashboard Subsystem**

• **Responsibility**: Provides platform administration capabilities

• **Key Components**: User management, business approval, analytics

• **Interfaces**: Admin API endpoints

#### **3.8 Messaging System Subsystem**

• **Responsibility**: Enables communication between users and providers

• **Key Components**: Chat interface, notifications, message history

• **Interfaces**: Socket.IO endpoints, message API

#### **3.9 Payment Processing Subsystem**

• **Responsibility**: Handles secure payment transactions

• **Key Components**: Payment form, transaction history, invoicing

• **Interfaces**: Payment API endpoints, Stripe integration

### **4. Component and Connector View**

This view shows how components interact at runtime through connectors:

![](data:None;base64,)

*mermaid chart*

#### **4.1 Key Connectors**

1. **HTTP/HTTPS**: RESTful communication between frontend and API routes

2. **WebSockets**: Real-time bidirectional communication for chat and notifications

3. **Mongoose**: Object-Document Mapping for MongoDB interactions

4. **GraphQL**: Query language for Hygraph CMS interactions

### **5. Deployment Architecture**

The deployment architecture shows how the system components are distributed across physical infrastructure:

![](data:None;base64,)

*mermaid chart*

### **6. Data Architecture**

The data architecture illustrates the primary data entities and their relationships:

![](data:None;base64,)

*mermaid chart*

### **7. Architectural Strategies**

#### **7.1 Security Strategy**

• JWT-based authentication

• Role-based access control

• HTTPS for all communications

• Input validation and sanitization

#### **7.2 Scalability Strategy**

• Serverless architecture for automatic scaling

• Database connection pooling

• Optimized database queries with indexes

• Static generation for content-heavy pages

#### **7.3 Performance Strategy**

• Server-side rendering for SEO-critical pages

• Client-side rendering for interactive components

• Image optimization via Cloudinary

• Caching of frequent queries

#### **7.4 Reliability Strategy**

• Error boundary components

• Graceful degradation

• Automated backups

• Health checks and monitoring

### **8. Architectural Decisions**

|  |  |  |  |
| --- | --- | --- | --- |
| Decision | Options Considered | Selected Approach | Rationale |
| Frontend Framework | React, Vue, Angular | Next.js (React) | SEO requirements, developer familiarity, hybrid rendering |
| Database | PostgreSQL, MongoDB, Firebase | MongoDB | Schema flexibility, JSON document structure, scalability |
| Content Management | Custom CMS, Contentful, Hygraph | Hygraph CMS | GraphQL support, content modeling flexibility, developer experience |
| Authentication | Custom Auth, Firebase Auth, NextAuth | NextAuth | Integration with Next.js, multiple providers support, simplicity |
| Styling | CSS Modules, Styled Components, Tailwind | Tailwind CSS | Utility-first approach, rapid development, consistency |
| Deployment | AWS, Vercel, Netlify | Vercel | Optimized for Next.js, serverless functions, edge network |

### **9. Quality Attribute Analysis**

|  |  |  |
| --- | --- | --- |
| Quality Attribute | Architectural Support | Potential Risks |
| Performance | Server-side rendering, edge caching | Complex queries might slow response times |
| Security | Middleware for authentication, input validation | Session management complexity |
| Reliability | Error boundaries, graceful degradation | External service dependencies |
| Maintainability | Component-based structure, service layer | Knowledge transfer for complex patterns |
| Scalability | Serverless architecture, connection pooling | Database scaling limitations |

### **10. Architecture Evolution**

The architecture is designed to evolve in the following ways:

1. **Microservices Transition**: As the system grows, certain subsystems (e.g., messaging, payments) can be extracted into independent microservices.

2. **API Gateway**: Introduction of an API gateway to manage cross-cutting concerns as the number of services increases.

3. **Caching Layer**: Addition of Redis or similar caching solution for frequently accessed data.

4. **Event-Driven Architecture**: Evolution towards an event-driven architecture for better decoupling between services.

This architectural design provides a solid foundation that balances immediate needs with future scalability and maintainability.