**4. Project Design**

# 目录

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

II. Components Design ........................ 3

1. Class Diagram ........................ 4

2. Component Descriptions ........................ 5

3. Component Interactions ........................ 15

4. Design Patterns ........................ 19

5. Component Dependencies ........................ 24

6. Component Extensibility ........................ 25

*[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**

## **II. Components Design**

This document outlines the component design of the ArtisanConnect platform, focusing on the class structure and relationships between key components.

### **1. Class Diagram**

The following class diagram illustrates the core domain model of the ArtisanConnect platform, showing the relationships between major classes and their attributes and methods:

![](data:None;base64,)

*mermaid chart*

### **2. Component Descriptions**

#### **2.1 User Component**

The User component represents individuals interacting with the platform in different roles (Customer, Provider, Admin).

**Key Responsibilities:**

• User authentication and authorization

• Profile management

• Role-based access control

**Key Methods:**

• validatePassword(password): Verifies user credentials during login

• createUser(userData): Creates a new user account

• findByEmail(email): Retrieves user information by email

• updateProfile(userData): Updates user profile information

**Implementation Files:**

• models/User.js: Mongoose schema and model definition

• app/api/auth/[...nextauth]/route.js: NextAuth authentication implementation

• app/(routes)/profile/page.jsx: User profile management UI

#### **2.2 BusinessList Component**

The BusinessList component represents service providers’ businesses listed on the platform.

**Key Responsibilities:**

• Business profile management

• Service listing

• Business discovery and search

**Key Methods:**

• createBusiness(businessData): Creates a new business listing

• findById(id): Retrieves business details by ID

• findByCategory(category): Finds businesses in a specific category

• updateStatus(id, status): Updates approval status (PENDING, APPROVED, REJECTED)

**Implementation Files:**

• models/BusinessList.js: Mongoose schema and model definition

• app/api/businesses/route.js: API endpoints for business management

• app/(routes)/details/[businessId]/page.jsx: Business detail page

#### **2.3 Booking Component**

The Booking component handles service appointments between customers and providers.

**Key Responsibilities:**

• Appointment scheduling

• Booking management

• Status tracking

**Key Methods:**

• createBooking(bookingData): Creates a new booking

• findByUser(userEmail): Retrieves bookings for a specific user

• findByBusiness(businessId): Retrieves bookings for a specific business

• updateStatus(id, status): Updates booking status

**Implementation Files:**

• models/Booking.js: Mongoose schema and model definition

• app/api/bookings/route.js: API endpoints for booking management

• app/(routes)/details/\_components/BookingSection.jsx: Booking UI component

#### **2.4 Review Component**

The Review component manages customer feedback and ratings for businesses.

**Key Responsibilities:**

• Rating collection

• Review management

• Rating calculation

**Key Methods:**

• createReview(reviewData): Creates a new review

• findByBusiness(businessId): Retrieves reviews for a specific business

• calculateAverageRating(businessId): Calculates average rating for a business

**Implementation Files:**

• models/Review.js: Mongoose schema and model definition

• app/api/reviews/route.js: API endpoints for review management

• app/\_components/ReviewForm.jsx: Review submission UI component

• app/\_components/ReviewList.jsx: Review display UI component

#### **2.5 Category Component**

The Category component organizes businesses into service categories.

**Key Responsibilities:**

• Category management

• Service classification

**Key Methods:**

• getAllCategories(): Retrieves all available categories

• findById(id): Retrieves category details by ID

• updateIcon(id, iconUrl): Updates category icon

**Implementation Files:**

• models/Category.js: Mongoose schema and model definition

• app/api/categories/route.js: API endpoints for category management

• app/\_components/CategoryList.jsx: Category display UI component

#### **2.6 ChatMessage Component**

The ChatMessage component enables communication between customers and service providers.

**Key Responsibilities:**

• Message exchange

• Conversation history

• Read status tracking

**Key Methods:**

• createMessage(messageData): Creates a new message

• findByBooking(bookingId): Retrieves messages for a specific booking

• markAsRead(id): Marks a message as read

**Implementation Files:**

• models/ChatMessage.js: Mongoose schema and model definition

• app/api/chat/[bookingId]/route.js: API endpoints for chat management

• app/\_components/ChatWindow.jsx: Chat UI component

#### **2.7 Portfolio Component**

The Portfolio component manages service providers’ work samples and project showcases.

**Key Responsibilities:**

• Portfolio management

• Image storage and retrieval

**Key Methods:**

• addPortfolioItem(portfolioData): Adds a new portfolio item

• findByBusiness(businessId): Retrieves portfolio items for a business

• updateItem(id, data): Updates portfolio item details

**Implementation Files:**

• models/Portfolio.js: Mongoose schema and model definition

• app/api/provider/portfolio/route.js: API endpoints for portfolio management

• app/\_components/PortfolioGallery.jsx: Portfolio display UI component

#### **2.8 UserStatus Component**

The UserStatus component tracks user online/offline status for real-time features.

**Key Responsibilities:**

• Online status tracking

• Activity monitoring

**Key Methods:**

• updateStatus(email, status): Updates user online status

• getOnlineProviders(): Retrieves currently online service providers

**Implementation Files:**

• models/UserStatus.js: Mongoose schema and model definition

• app/api/socket/route.js: WebSocket implementation for status updates

#### **2.9 Notification Component**

The Notification component manages system notifications for users.

**Key Responsibilities:**

• Notification generation

• Notification delivery

• Read status tracking

**Key Methods:**

• createNotification(notificationData): Creates a new notification

• findByUser(userEmail): Retrieves notifications for a user

• markAsRead(id): Marks a notification as read

**Implementation Files:**

• models/Notification.js: Mongoose schema and model definition

• app/\_services/EmailService.js: Email notification service

### **3. Component Interactions**

#### **3.1 Booking Flow**

The booking flow illustrates how components interact during the booking process:

1. User (Customer) selects a BusinessList to book

2. BookingSection component collects booking details

3. Booking component creates a new booking record

4. Notification component generates notifications for both customer and provider

5. ChatMessage component enables communication about the booking

#### **3.2 Review Flow**

The review flow shows component interactions during the review process:

1. User (Customer) submits a review for a BusinessList

2. Review component creates a new review record

3. BusinessList component updates the average rating

4. Notification component alerts the business owner about the new review

#### **3.3 Provider Registration Flow**

The provider registration flow demonstrates component interactions during business registration:

1. User (Provider) submits business details

2. BusinessList component creates a new business with PENDING status

3. Notification component alerts admins about the new business

4. User component updates the user’s role to PROVIDER

### **4. Design Patterns**

The component design incorporates several design patterns:

#### **4.1 Repository Pattern**

The models implement a repository pattern, encapsulating data access logic:

**javascript**

// Example from models/BusinessList.js

static async findByCategory(category) {

  return this.find({ category: category, approvalStatus: 'APPROVED' }).sort({ rating: -1 });

}

#### **4.2 Factory Pattern**

The component creation follows a factory pattern, particularly in API services:

**javascript**

// Example from app/\_services/GlobalApi.js

const getBusinessById = (id) => {

  return ApiService.get(`/businesses/${id}`);

};

#### **4.3 Observer Pattern**

The notification system implements an observer pattern:

**javascript**

// Example from app/api/bookings/route.js

await Notification.create({

  userEmail: provider.email,

  type: 'NEW\_BOOKING',

  message: `New booking from ${booking.userName}`,

  relatedId: booking.\_id

});

#### **4.4 Strategy Pattern**

The authentication system uses a strategy pattern with different providers:

**javascript**

// Example from app/api/auth/[...nextauth]/route.js

providers: [

  CredentialsProvider({

    // Authentication strategy implementation

  }),

  // Other potential auth providers

]

### **5. Component Dependencies**

The component dependencies are managed through:

1. **Direct Model References**: Mongoose models reference each other through IDs

2. **Service Layer**: API services abstract data access for UI components

3. **Context Providers**: React context for sharing state between components

4. **Event-based Communication**: WebSockets for real-time updates

### **6. Component Extensibility**

The components are designed for extensibility through:

1. **Modular Structure**: Components have single responsibilities

2. **Clear Interfaces**: Well-defined methods for interaction

3. **Dependency Injection**: Services are injected rather than hardcoded

4. **Configuration-driven Behavior**: Components adapt based on configuration

This component design provides a solid foundation for the ArtisanConnect platform, with clear responsibilities, relationships, and extension points.