**4. Project Design**

# 目录

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

III. Database Schemas ........................ 3

1. Database Architecture ........................ 4

2. MongoDB Schemas ........................ 5

3. Additional MongoDB Schemas ........................ 13

4. Database Relationships ........................ 15

5. Database Indexing Strategy ........................ 16

6. Data Migration Strategy ........................ 17

7. Database Security ........................ 18

8. Database Backup Strategy ........................ 19

*[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**

## **III. Database Schemas**

This document outlines the database schemas for the ArtisanConnect platform, following Ian Sommerville’s approach to database design documentation. The diagrams are kept simple and focused on essential information to ensure clarity.

### **1. Database Architecture**

The ArtisanConnect platform uses MongoDB as its primary database system:

![](data:None;base64,)

*mermaid chart*

### **2. MongoDB Schemas**

MongoDB uses Mongoose schemas to define the structure of documents. The following diagrams illustrate the key schemas in the ArtisanConnect platform.

#### **2.1 User Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/User.js

const UserSchema = new mongoose.Schema({

  email: { type: String, required: true, unique: true },

  name: { type: String, required: true },

  password: { type: String, required: true },

  role: {

    type: String,

    enum: ['CUSTOMER', 'PROVIDER', 'ADMIN'],

    default: 'CUSTOMER'

  },

  emailVerified: { type: Boolean, default: false },

  createdAt: { type: Date, default: Date.now },

  updatedAt: { type: Date, default: Date.now }

});

#### **2.2 BusinessList Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/BusinessList.js

const BusinessListSchema = new mongoose.Schema({

  name: { type: String, required: true },

  description: { type: String, required: true },

  address: { type: String, required: true },

  contactEmail: { type: String, required: true },

  contactPhone: { type: String, required: true },

  images: [{ type: String }],

  category: { type: String, required: true },

  providerEmail: { type: String, required: true },

  approvalStatus: {

    type: String,

    enum: ['PENDING', 'APPROVED', 'REJECTED'],

    default: 'PENDING'

  },

  rating: { type: Number, default: 0 },

  totalReviews: { type: Number, default: 0 },

  createdAt: { type: Date, default: Date.now },

  updatedAt: { type: Date, default: Date.now },

  location: {

    type: { type: String, default: 'Point' },

    coordinates: [Number]

  }

});

#### **2.3 Booking Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/Booking.js

const BookingSchema = new mongoose.Schema({

  businessId: { type: String, required: true },

  userEmail: { type: String, required: true },

  userName: { type: String, required: true },

  bookingDate: { type: Date, required: true },

  time: { type: String, required: true },

  status: {

    type: String,

    enum: ['PENDING', 'CONFIRMED', 'COMPLETED', 'CANCELLED'],

    default: 'PENDING'

  },

  note: { type: String },

  createdAt: { type: Date, default: Date.now },

  updatedAt: { type: Date, default: Date.now }

});

#### **2.4 Review Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/Review.js

const ReviewSchema = new mongoose.Schema({

  businessId: { type: String, required: true },

  userEmail: { type: String, required: true },

  userName: { type: String, required: true },

  rating: { type: Number, required: true, min: 1, max: 5 },

  comment: { type: String },

  createdAt: { type: Date, default: Date.now }

});

#### **2.5 ChatMessage Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/ChatMessage.js

const ChatMessageSchema = new mongoose.Schema({

  bookingId: { type: String, required: true },

  senderId: { type: String, required: true },

  recipientId: { type: String, required: true },

  content: { type: String, required: true },

  timestamp: { type: Date, default: Date.now },

  isRead: { type: Boolean, default: false }

});

#### **2.6 Portfolio Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/Portfolio.js

const PortfolioSchema = new mongoose.Schema({

  businessId: { type: String, required: true },

  title: { type: String, required: true },

  description: { type: String },

  imageUrl: { type: String, required: true },

  createdAt: { type: Date, default: Date.now }

});

#### **2.7 Notification Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/Notification.js

const NotificationSchema = new mongoose.Schema({

  userEmail: { type: String, required: true },

  type: {

    type: String,

    enum: ['NEW\_BOOKING', 'BOOKING\_UPDATE', 'NEW\_REVIEW', 'APPROVAL\_UPDATE', 'MESSAGE'],

    required: true

  },

  message: { type: String, required: true },

  relatedId: { type: String },

  isRead: { type: Boolean, default: false },

  createdAt: { type: Date, default: Date.now }

});

### **3. Additional MongoDB Schemas**

#### **3.1 Category Schema**

![](data:None;base64,)

*mermaid chart*

**Schema Definition:**

**javascript**

// models/Category.js

const CategorySchema = new mongoose.Schema({

  name: { type: String, required: true },

  icon: { type: String, required: true },

  description: { type: String },

});

### **4. Database Relationships**

The following diagram illustrates the relationships between the main database entities:

![](data:None;base64,)

*mermaid chart*

### **5. Database Indexing Strategy**

To optimize query performance, the following indexes are implemented:

|  |  |  |  |
| --- | --- | --- | --- |
| Collection | Index Fields | Index Type | Purpose |
| Users | email | Unique | Fast user lookup by email |
| BusinessLists | category | Standard | Fast business lookup by category |
| BusinessLists | approvalStatus | Standard | Fast filtering of businesses by status |
| BusinessLists | location | Geospatial | Location-based search |
| Bookings | userEmail | Standard | Fast lookup of user’s bookings |
| Bookings | businessId | Standard | Fast lookup of business’s bookings |
| Reviews | businessId | Standard | Fast retrieval of business reviews |
| ChatMessages | bookingId | Standard | Fast retrieval of booking messages |

### **6. Data Migration Strategy**

For schema evolution and data migration, the platform follows these strategies:

1. **Versioned Schemas**: Schema versions are tracked in code

2. **Migration Scripts**: Dedicated scripts for data transformation

3. **Backward Compatibility**: New schemas maintain compatibility with old data

4. **Incremental Updates**: Migrations run incrementally to minimize downtime

Example migration script structure:

**javascript**

// migrations/20230601\_add\_location\_to\_businesses.js

module.exports = async function(db) {

  const businesses = await db.collection('businesslists').find({}).toArray();

  for (const business of businesses) {

    if (!business.location) {

      await db.collection('businesslists').updateOne(

        { \_id: business.\_id },

        { $set: {

            location: {

              type: 'Point',

              coordinates: [0, 0] // Default coordinates

            }

          }

        }

      );

    }

  }

  console.log('Migration completed: Added location field to businesses');

};

### **7. Database Security**

The database security strategy includes:

1. **Network Security**:

• MongoDB Atlas IP whitelisting

• VPC peering for cloud deployments

2. **Authentication**:

• Strong database user credentials

• Role-based database access

3. **Authorization**:

• Field-level encryption for sensitive data

• Document-level access controls

4. **Data Validation**:

• Schema validation rules

• Input sanitization before storage

### **8. Database Backup Strategy**

The database backup strategy ensures data durability:

1. **Automated Backups**:

• Daily full backups

• Point-in-time recovery for MongoDB Atlas

2. **Backup Retention**:

• 7 days of daily backups

• 4 weeks of weekly backups

• 3 months of monthly backups

3. **Backup Verification**:

• Weekly automated restore tests

• Integrity verification

This database schema design provides a solid foundation for the ArtisanConnect platform, with clear entity definitions, optimized performance, and robust security and backup strategies.