Bidzi Database Schema

Created by	Ⓢ Shuja Ali
Created time	@September 11, 2023 2:27 PM
Tags	

User Model

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  // User Information
  username: {
    type: String,
    required: true,
    unique: true,
    trim: true,
    minlength: 3,
    maxlength: 50,
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    trim: true,
    match: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/,
  },
  password: {
    type: String,
    required: true,
    minlength: 8,
    // Hashed and salted password will be stored
  },
  role: {
    type: String,
    enum: ['admin', 'buyer', 'seller'],
    required: true,
  },
  // Profile Information
  firstName: {
    type: String,
    trim: true,
    maxlength: 50,
  },
  lastName: {
    type: String,
    trim: true,
    maxlength: 50,
  },
  profilePicture: {
    type: String,
    // Store the URL to the user's profile picture
  },
  // LinkedIn Integration
  linkedinId: {
    type: String,
    // Store the LinkedIn user ID for integration
  },
  // KYC Verification
  kycStatus: {
    type: String,
    enum: ['pending', 'approved', 'rejected'],
```

```
    default: 'pending',
  },
  // Notifications
  emailVerified: {
    type: Boolean,
    default: false,
  },
  // Activity and Security
  lastLogin: {
    type: Date,
  },
  failedLoginAttempts: {
    type: Number,
    default: 0,
  },
  // Timestamps
  createdAt: {
    type: Date,
    default: Date.now,
  },
  updatedAt: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('User', userSchema);
```

Listing Model

```
const mongoose = require('mongoose');

const listingSchema = new mongoose.Schema({
  // Basic Information
  title: {
    type: String,
    required: true,
    trim: true,
    maxlength: 100,
  },
  description: {
    type: String,
    required: true,
    trim: true,
    maxlength: 1000,
  },
  category: {
    type: String,
    required: true,
    trim: true,
    maxlength: 50,
  },
  location: {
    type: String,
    trim: true,
    maxlength: 100,
  },
  // Business Details
  askingPrice: {
    type: Number,
    required: true,
  },
  annualRevenue: {
    type: Number,
  },
```

```
  annualProfit: {
    type: Number,
  },
  // Seller Information
  seller: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  // Listing Status
  status: {
    type: String,
    enum: ['draft', 'published', 'under offer', 'sold'],
    default: 'draft',
  },
  // Document Uploads
  documents: [
    {
      name: String, // Name of the document
      url: String, // URL to access the document
    },
  ],
  // Analytics
  views: {
    type: Number,
    default: 0,
  },
  // Timestamps
  createdAt: {
    type: Date,
    default: Date.now,
  },
  updatedAt: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('Listing', listingSchema);
```

NDA Model

```
const mongoose = require('mongoose');

const ndaSchema = new mongoose.Schema({
  // Unique identifier for the NDA
  ndaNumber: {
    type: String,
    required: true,
    unique: true,
  },

  // Seller who initiated the NDA
  seller: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // Reference to the User model
    required: true,
  },

  // Buyer who received the NDA
  buyer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // Reference to the User model
```

```
    required: true,
  },

  // Listing associated with the NDA
  listing: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Listing', // Reference to the Listing model
    required: true,
  },

  // Status of the NDA (e.g., "Pending," "Accepted," "Rejected")
  status: {
    type: String,
    enum: ['Pending', 'Accepted', 'Rejected'],
    default: 'Pending',
    required: true,
  },

  // Date when the NDA was created
  createdDate: {
    type: Date,
    default: Date.now,
  },

  // Date when the NDA was last updated
  updatedDate: {
    type: Date,
    default: Date.now,
  },

  // Text content of the NDA agreement
  agreementText: {
    type: String,
    required: true,
  },

  // Signature of the seller (base64 encoded image or other format)
  sellerSignature: {
    type: String,
    required: true,
  },

  // Signature of the buyer (base64 encoded image or other format)
  buyerSignature: {
    type: String,
    required: true,
  },

  // Comments or notes related to the NDA
  comments: {
    type: String,
  },
});

const NDA = mongoose.model('NDA', ndaSchema);

module.exports = NDA;
```

LOI Model

```
const mongoose = require('mongoose');

const loiSchema = new mongoose.Schema({
```

```
// Unique identifier for the LOI
loiNumber: {
  type: String,
  required: true,
  unique: true,
},

// Seller who received the LOI
seller: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User', // Reference to the User model
  required: true,
},

// Buyer who initiated the LOI
buyer: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User', // Reference to the User model
  required: true,
},

// Listing associated with the LOI
listing: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Listing', // Reference to the Listing model
  required: true,
},

// Status of the LOI (e.g., "Draft," "Submitted," "Accepted," "Rejected")
status: {
  type: String,
  enum: ['Draft', 'Submitted', 'Accepted', 'Rejected'],
  default: 'Draft',
  required: true,
},

// Date when the LOI was created
createdDate: {
  type: Date,
  default: Date.now,
},

// Date when the LOI was last updated
updatedDate: {
  type: Date,
  default: Date.now,
},

// Proposed terms and conditions in the LOI
termsAndConditions: {
  type: String,
  required: true,
},

// Purchase price offered in the LOI
purchasePrice: {
  type: Number,
  required: true,
},

// Deposit amount offered in the LOI
depositAmount: {
  type: Number,
  required: true,
},

// Expiry date of the LOI
expiryDate: {
  type: Date,
  required: true,
},
```

```
  // Comments or notes related to the LOI
  comments: {
    type: String,
  },

  attachments: [
    {
      filename: String, // Original file name
      path: String,     // Path to the stored file
    }
  ],

  loiAgreement: {
      filename: String, // Original file name
      path: String,     // Path to the stored file
  }
});

const LOI = mongoose.model('LOI', loiSchema);

module.exports = LOI;
```

Data Room Model

```
const mongoose = require('mongoose');

const dataRoomSchema = new mongoose.Schema({
  // Reference to the listing associated with this Data Room
  listing: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Listing',
    required: true,
  },

  // Questions and answers related to due diligence
  dueDiligence: [
    {
      question: {
        type: String,
        required: true,
      },
      answer: {
        type: String,
        required: true,
      },
      documents: [
        {
          filename: String, // Original file name
          path: String,     // Path to the stored file
        }
      ],
    },
  ],

  // Documents uploaded by the seller for due diligence
  uploadedDocuments: [
    {
      filename: String, // Original file name
      path: String,     // Path to the stored file
    },
  ],

  // Date when the Data Room was created
```

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

  // Date when the Data Room was last updated
  updatedAt: {
    type: Date,
    default: Date.now,
  },
});

const DataRoom = mongoose.model('DataRoom', dataRoomSchema);

module.exports = DataRoom;
```

SPA Model

```
const mongoose = require('mongoose');

const spaSchema = new mongoose.Schema({
  // Reference to the listing associated with this SPA
  listing: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Listing',
    required: true,
  },

  // Reference to the buyer involved in this SPA
  buyer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },

  // Clauses and comments within the SPA
  clauses: [
    {
      text: {
        type: String,
        required: true,
      },
      modifiedText: String, // If the clause has been modified by the seller
      status: {
        type: String,
        enum: ['accepted', 'modified', 'rejected'],
        default: 'pending',
      },
    },
  ],

  // Status of the SPA
  status: {
    type: String,
    enum: ['accepted', 'rejected', 'pending'],
    default: 'pending',
  },

  // Date when the SPA was created
  createdAt: {
    type: Date,
    default: Date.now,
  },
```

```
  // Date when the SPA was last updated
  updatedAt: {
    type: Date,
    default: Date.now,
  },
});

const SPA = mongoose.model('SPA', spaSchema);

module.exports = SPA;
```

Chat Model

```
const mongoose = require('mongoose');

const chatSchema = new mongoose.Schema({
  // Reference to the listing associated with this chat
  listing: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Listing',
    required: true,
  },

  // Reference to the seller involved in this chat
  seller: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },

  // Reference to the buyer involved in this chat
  buyer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },

  // Messages exchanged in the chat
  messages: [
    {
      sender: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true,
      },
      text: {
        type: String,
        required: true,
      },
      createdAt: {
        type: Date,
        default: Date.now,
      },
    },
  ],

  // Date when the chat was created
  createdAt: {
    type: Date,
    default: Date.now,
  },
```

```
  // Date when the chat was last updated
  updatedAt: {
    type: Date,
    default: Date.now,
  },
});

const Chat = mongoose.model('Chat', chatSchema);

module.exports = Chat;
```

Resources Model

```
const mongoose = require('mongoose');

const resourceSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  content: {
    type: String,
    required: true,
  },
  tags: {
    type: [String],
    required: true,
  },
  link: {
    type: String, // Assuming the link is a string (URL)
    required: false, // You can change this to true if it's required
  },
  created_at: {
    type: Date,
    default: Date.now,
  },
  updated_at: {
    type: Date,
    default: Date.now,
  },
});

const Resource = mongoose.model('Resource', resourceSchema);

module.exports = Resource;
```

Analytics Query Model

```
const mongoose = require('mongoose');

const analyticsSchema = new mongoose.Schema({
  // Timestamp to track when the analytics data was recorded
  timestamp: {
    type: Date,
    default: Date.now,
```

```
  },
  // Type of analytics data (e.g., 'MarketplaceSearch', 'LocationSearch', 'IndustrySearch', 'AskingPriceSearch', 'FAQSearch')
  type: {
    type: String,
    enum: ['MarketplaceSearch', 'LocationSearch', 'IndustrySearch', 'AskingPriceSearch', 'FAQSearch'],
    required: true,
  },
  // Query or keyword used for the search
  query: {
    type: String,
    required: true,
  },
  // Count or frequency of the search
  count: {
    type: Number,
    default: 1, // You can increment this count for multiple occurrences of the same query
  },
  // User role (e.g., 'Admin', 'Buyer', 'Seller') who performed the search
  userRole: {
    type: String,
    enum: ['Admin', 'Buyer', 'Seller'],
    required: true,
  },
});

const Analytics = mongoose.model('Analytics', analyticsSchema);

module.exports = Analytics;
```

Payment Model

```
const mongoose = require('mongoose');

const paymentSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  paymentType: {
    type: String,
    enum: ['Buyer', 'Seller'],
    required: true,
  },
  amountPaid: {
    type: Number,
    required: true,
  },
  paymentCategory: {
    type: String,
    enum: ['Subscription', 'One-Time'],
    required: true,
  },
  paymentDate: {
    type: Date,
    default: Date.now,
  },
});

module.exports = mongoose.model('Payment', paymentSchema);
```

Escrow Model

```
const mongoose = require('mongoose');

const escrowSchema = new mongoose.Schema({
  // Reference to the buyer associated with this escrow
  buyer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  // Reference to the seller associated with this escrow
  seller: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  // Reference to the SPA (Sales Purchase Agreement) associated with this escrow
  spa: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'SPA',
    required: true,
  },
  // Status of the escrow process (e.g., 'pending', 'completed', 'closed')
  status: {
    type: String,
    enum: ['pending', 'completed', 'closed'],
    default: 'pending',
  },
  // Reference to the Trade License document uploaded by the seller
  tradeLicenseDocument: {
    type: String, // You can store the document's URL or any identifier
    required: false, // It becomes required when the transfer is in progress
  },
  // Timestamp when the ownership transfer was completed
  transferCompletedAt: {
    type: Date,
    required: false, // Becomes required when the transfer is completed
  },
  // Timestamp when the transaction was confirmed
  transactionConfirmedAt: {
    type: Date,
    required: false, // Becomes required when the transaction is confirmed
  },
});

const Escrow = mongoose.model('Escrow', escrowSchema);

module.exports = Escrow;
```