Appendix D: Data Definitions and Schemas

This appendix provides detailed specifications of the data models used within the Service Market Place Botswana platform, including their Mongoose schema definitions and considerations for data structure.

D.1 Database Schemas (Mongoose)

The platform utilizes MongoDB as its primary database, with data models defined using Mongoose schemas. Below are the detailed schemas for the main collections:

D.1.1 User Schema

Represents a user in the system, including both Seekers and Providers. Integrated with Clerk for authentication.

Code:
```
const userSchema = new mongoose.Schema({
  clerkId: {
    type: String,
    required: true,
    unique: true,
    index: true // Index for quick lookups by Clerk ID
  },
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true // Store emails in lowercase
  },
  phoneNumber: {
    type: String,
    required: false, // Optional
    unique: true,
    sparse: true // Allow multiple nulls for unique constraint
  },
  profilePicture: {
    type: String, // URL or path to the stored image
    required: false
  },
  bio: {
    type: String,
    required: false
  },
  userType: {
    type: String,
    enum: ['Seeker', 'Provider'],
    required: true,
    default: 'Seeker'
  },
  providerAttributes: { // Fields specific to Providers
    birthday: {
      type: Date,
      required: false
    },
    baseLocation: { // Provider's primary service area location
        type: {
            type: String,
            enum: ['Point'],
            required: true,
            default: 'Point'
        },
        coordinates: { // [longitude, latitude]
            type: [Number],
            required: true,
            index: '2dsphere' // Geospatial index for proximity searches
        }
    },
    selectedCategories: [{ // Categories the provider offers services in
        categoryId: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Category',
            required: true
        },
        attributes: { // Dynamic attributes specific to this category (e.g.,
'tool_brand': 'Bosch')
            type: Map, // Stores key-value pairs
            of: String
        }
    }],
    // Other provider-specific fields like skills, service area radius, etc.
  }
}, { timestamps: true }); // Adds createdAt and updatedAt

// Optional: Index userType for filtering
userSchema.index({ userType: 1 });
```

D.1.2 Job Schema

Represents a job posting created by a Seeker.

Code:
```
const jobSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  categoryId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Category',
    required: true
  },
  category: { // Denormalized category name for easier access
     type: String,
     required: true
  },
  budget: {
    type: mongoose.Types.Decimal128, // Use Decimal128 for currency
    required: true
  },
  location: { // Job location
    type: {
      type: String,
      enum: ['Point'],
      required: true,
      default: 'Point'
    },
    coordinates: { // [longitude, latitude]
        type: [Number],
        required: true,
        index: '2dsphere' // Geospatial index
    }
  },
  altLocationId: { // Optional reference to a predefined location or address
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Location' || 'Address', // Adjust ref based on your model
      required: false,
      nullable: true // Ensure it's explicitly nullable
  },
  seekerId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // Refers to the User model (Seeker)
    required: true,
    index: true // Index for filtering by seeker
  },
  providerId: { // Assigned provider after a bid is accepted
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // Refers to the User model (Provider)
    required: false, // Not required until bid is accepted
    nullable: true // Ensure it's explicitly nullable
  },
  agreedAmount: { // Final agreed amount after bid acceptance
     type: mongoose.Types.Decimal128,
     required: false, // Not required until bid is accepted
     nullable: true
  },
  status: {
    type: String,
    enum: ['Pending', 'Open', 'In Progress', 'Completed', 'Cancelled'],
    required: true,
    default: 'Pending', // Or 'Open', depending on approval workflow
    index: true // Index for filtering by status
  },
  attributes: { // Dynamic attributes provided by the seeker (e.g.,
'number_of_rooms': '3')
     type: Map, // Stores key-value pairs
     of: String,
     required: false,
     nullable: true
  },
  startedAt: { // Timestamp when status changes to 'In Progress'
    type: Date,
    required: false,
    nullable: true
  },
  completedAt: { // Timestamp when status changes to 'Completed'
    type: Date,
    required: false,
    nullable: true
  },
  // Optional: Add fields for deadlines, required skills, etc.
}, { timestamps: true }); // Adds createdAt and updatedAt

// Compound index for efficient lookups by seeker and status
jobSchema.index({ seekerId: 1, status: 1 });
```

D.1.3 Bid Schema

Represents a bid placed by a Provider on an 'Open' Job.

Code:
```
const bidSchema = new mongoose.Schema({
  jobId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Job',
    required: true,
    index: true // Index for querying bids for a specific job
  },
  seekerId: { // Denormalized seeker ID for easier access/validation
     type: mongoose.Schema.Types.ObjectId,
     ref: 'User',
     required: true,
     index: true // Index for querying bids received by a seeker
  },
  providerId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // Refers to the User model (Provider)
    required: true,
    index: true // Index for querying bids placed by a provider
  },
  amount: {
    type: mongoose.Types.Decimal128, // Use Decimal128 for currency
    required: true
  },
  description: { // Optional message from provider
```

```
     type: String,
     required: false,
     nullable: true
  },
  status: { // Status of the bid itself (newly added based on previous
discussion)
     type: String,
     enum: ['Pending', 'Accepted', 'Rejected'],
     required: true,
     default: 'Pending',
     index: true // Index for filtering bids by status
  }
}, { timestamps: true }); // Adds createdAt and updatedAt

// Optional: Compound index for quick check if a provider has bid on a
specific job
bidSchema.index({ jobId: 1, providerId: 1 }, { unique: true });
bidSchema.index({ providerId: 1, status: 1 }); // Index for querying bids
placed by provider filtered by status
```

D.1.4 Payment Schema

Represents a payment record associated with a completed job.

Code:

```
const paymentSchema = new mongoose.Schema({
  jobId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Job',
    required: true,
    unique: true // One payment record per job
  },
  seekerId: { // Seeker initiating payment
     type: mongoose.Schema.Types.ObjectId,
     ref: 'User',
     required: true
  },
  providerId: { // Provider receiving payment
     type: mongoose.Schema.Types.ObjectId,
     ref: 'User',
     required: true
  },
  amount: {
    type: mongoose.Types.Decimal128, // Use Decimal128
    required: true // Should match job's agreedAmount
  },
  paymentMethod: { // e.g., 'Mpesa', 'Bank Transfer', 'In App Wallet'
    type: String,
    required: true
  },
  paymentStatus: {
    type: String,
    enum: ['Pending', 'Completed', 'Failed'],
    required: true,
    default: 'Pending',
    index: true // Index for filtering payments by status
  },
  transactionReference: { // Reference from external payment system
    type: String,
    required: false,
    unique: true,
    sparse: true,
    nullable: true
  },
  // Optional: Add fields for payment date, gateway response, etc.
}, { timestamps: true }); // Adds createdAt and updatedAt

// Index for querying payments by user
paymentSchema.index({ seekerId: 1 });
paymentSchema.index({ providerId: 1 });
```

D.1.5 Category Schema

Represents a category of service offered on the platform.

Code:
```
const categorySchema = new mongoose.Schema({
  categoryName: {
    type: String,
    required: true,
    unique: true,
    trim: true // Remove leading/trailing whitespace
  },
  description: {
    type: String,
    required: false,
    nullable: true
  },
  // Optional: Add fields for required attributes for this category, icon,
etc.
  requiredAttributes: { // Define attributes expected for jobs/providers in
this category
     type: Map, // e.g., {'tool_brand': 'string', 'experience_years':
'number'}
     of: String, // Could store expected type or description
     required: false,
     nullable: true
  }
}, { timestamps: true }); // Adds createdAt and updatedAt
```

D.1.6 Location Schema

Represents predefined administrative locations (e.g., cities, towns) if not using dynamic GeoJSON points everywhere. (Based on `/getLocations` endpoint).

Code:
```
const locationSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    unique: true
  },
  coordinates: { // Center point of the location
      type: {
          type: String,
          enum: ['Point'],
          required: true,
          default: 'Point'
      },
      coordinates: { // [longitude, latitude]
          type: [Number],
          required: true
      }
  },
  // Optional: Add fields for description, area polygons, etc.
}, { timestamps: true }); // Adds createdAt and updatedAt

// Index for GeoJSON queries if locations represent areas
// locationSchema.index({ coordinates: '2dsphere' }); // Only if querying
based on these points
```

D.1.7 Address Schema

Represents specific street addresses that users or jobs might reference.

Code:

```
const addressSchema = new mongoose.Schema({
  userId: { // Optional: Associate address with a user
     type: mongoose.Schema.Types.ObjectId,
     ref: 'User',
     required: false,
     nullable: true,
     index: true // Index if querying addresses by user
  },
  street: {
    type: String,
    required: true
  },
  city: {
    type: String,
    required: true
  },
  state: { // Or District/Region
    type: String,
    required: false,
    nullable: true
  },
  postalCode: {
    type: String,
    required: false,
    nullable: true
  },
  country: {
    type: String,
    required: true
  },
  // Optional: Geo-encode address to store coordinates here too
  coordinates: {
      type: { type: String, enum: ['Point'] },
      coordinates: [Number], // [longitude, latitude]
      required: false,
      nullable: true
  }
}, { timestamps: true }); // Adds createdAt and updatedAt

// Compound index for efficient lookup by street/city
addressSchema.index({ street: 1, city: 1 });
```

D.1.8 Rating Schema

Represents a rating and review given by one user about another user for a specific job.

Code:

```
const ratingSchema = new mongoose.Schema({
  jobId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Job',
    required: true,
    unique: true // One rating record per job (from seeker to provider, or
vice versa)
  },
  reviewerId: { // User giving the rating
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    index: true // Index for querying ratings given by a user
  },
  revieweeId: { // User receiving the rating
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    index: true // Index for querying ratings received by a user
  },
  rating: {
    type: Number,
    required: true,
    min: 1, // Assuming a 1-5 scale
    max: 5
  },
  comment: { // Optional review text
    type: String,
    required: false,
    nullable: true
  },
  // Optional: Add field to indicate who is rating whom (e.g.,
'SeekerToProvider', 'ProviderToSeeker')
}, { timestamps: true }); // Adds createdAt and updatedAt

// Compound index for querying ratings for a specific job
```

```
ratingSchema.index({ jobId: 1 });
// Compound index for querying ratings between two users for a job (ensure
uniqueness if needed)
// ratingSchema.index({ jobId: 1, reviewerId: 1, revieweeId: 1 }, { unique:
true });
```

D.1.9 Notification Schema

Represents a notification sent to a user.

Code:

```
const notificationSchema = new mongoose.Schema({
  userId: { // User receiving the notification
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    index: true // Index for querying notifications for a user
  },
  type: { // Type of notification (e.g., 'bid', 'job_update', 'message',
'payment')
     type: String,
     required: true
  },
  title: { // Short title for the notification
     type: String,
     required: true
  },
  message: { // Main content of the notification
    type: String,
    required: true
  },
  read: {
    type: Boolean,
    required: true,
    default: false,
    index: true // Index for querying unread notifications
  },
  data: { // Optional: Store related IDs or data (e.g., { jobId: '...',
bidId: '...' })
     type: Map, // Stores key-value pairs
     of: mongoose.Schema.Types.Mixed, // Can store various types
     required: false,
     nullable: true
  },
  // Optional: Add fields for action URL, severity, expiry date etc.
}, { timestamps: true }); // Adds createdAt (when notification was created)

// Compound index for efficient lookup of unread notifications for a user
notificationSchema.index({ userId: 1, read: 1 });
```

D.2 Data Dictionary

D.2 Data Dictionary

The Data Dictionary provides a centralized repository of information about data elements within the system's database. It describes each field in detail, complementing the Mongoose schema definitions by offering a more human-readable explanation of the data's meaning, purpose, and characteristics. Including a formal data dictionary is beneficial for complex projects or larger development teams, although for smaller projects, detailed comments within the schema definitions (as shown in Section D.1) can often suffice.

Below are example entries for a few key fields from the core schemas to illustrate the typical format and content of a data dictionary for the Service Market Place Botswana platform:

Field Name	Schema/Model	Data Type	Description	Constraints	Relationships
_id	All	ObjectId (BSON Type)	Unique identifier automatically assigned to each document upon creation.	Auto-generated; Required; Primary Key.	N/A
clerkId	User	String	The unique ID assigned to the user by the Clerk authentication service.	Required; Unique; Indexed for efficient lookup.	External link to Clerk user record.
firstName	User	String	The user's given name.	Required.	N/A
lastName	User	String	The user's family name.	Required.	N/A
email	User	String	The user's primary email address.	Required; Unique; Stored in lowercase; Indexed.	Used for communication and potentially login/identification.
phoneNumber	User	String	The user's contact phone number.	Optional; Unique (sparse allows multiple users without a phone number).	Used for communication.
profilePicture	User	String	URL or file path pointing to the user's profile image.	Optional.	References an image file (external storage link).
bio	User	String	A short description or biography provided by the user.	Optional.	N/A

userType	User	String (enum)	The role of the user within the platform.	Required; Enum: 'Seeker', 'Provider'; Default: 'Seeker'; Indexed.	Determines user permissions and dashboard view.
providerAttributes	User	Object (Nested Schema)	Container for fields specific to Provider users.	Optional (only applicable if userType is 'Provider').	N/A
providerAttributes.birthday	User (Nested)	Date	The provider's date of birth.	Optional.	N/A
providerAttributes.baseLocation	User (Nested)	GeoJSON Point	The provider's base geographical coordinates representing their primary service area.	Required if userType is 'Provider'; Contains nested type ('Point') and coordinates [longitude, latitude] array; coordinates is Indexed (2dsphere).	Used for location-based matching.
providerAttributes.selectedCategories	User (Nested)	Array of Objects	A list of service categories the provider offers, including category-specific attributes.	Array of objects, each containing categoryId and attributes.	Each object references Category schema via categoryId.
providerAttributes.selectedCategories[].categoryId	User (Nested Array)	ObjectId (String)	References a specific service category the provider is associated with.	Required within the nested object; Must be a valid Category _id.	Refers to Category collection.

providerAttributes.selectedCategories[].attributes	User (Nested Array)	Map (String keys, String values)	Dynamic key-value pairs for attributes relevant to the specific category (e.g., {'tool_brand': 'Bosch'}).	Optional within the nested object.	N/A
title	Job	String	The title of the job posting.	Required.	N/A
description	Job	String	A detailed explanation of the tasks required for the job.	Required.	N/A
categoryId	Job	ObjectId (String)	References the main category this job belongs to.	Required; Must be a valid Category _id.	Refers to Category collection.
category	Job	String	Denormalized name of the job category for easier display without population.	Required; Should match the categoryName of the referenced category.	Derived from Category collection.
budget	Job	Decimal 128 (BSON Type)	The initial estimated cost or budget for the job, specified by the seeker.	Required; Stored with high precision for currency.	N/A
location	Job	GeoJSON Point	The specific geographical coordinates for where the job needs to be performed.	Required; Contains nested type ('Point') and coordinates [longitude, latitude] array; coordinates is Indexed (2dsphere).	Used for location-based matching.

altLocationId	Job	ObjectId (String)	Optional reference to a predefined Location or Address if GeoJSON is not used for the exact point.	Optional (nullable); Can reference Location or Address collection.	Refers to Location or Address collection.
seekerId	Job	ObjectId (String)	References the User who posted this job.	Required; Must be a valid User _id with userType 'Seeker'; Indexed.	Refers to User collection.
providerId	Job	ObjectId (String)	References the User (Provider) who was assigned to this job after a bid was accepted.	Optional (nullable until bid acceptance); Must be a valid User _id with userType 'Provider'.	Refers to User collection.
agreedAmount	Job	Decimal 128 (BSON Type)	The final agreed bid amount accepted by the seeker for this job.	Optional (nullable until bid acceptance); Stored with high precision.	N/A
status	Job	String (enum)	The current workflow state of the job posting.	Required; Enum: 'Pending', 'Open', 'In Progress', 'Completed', 'Cancelled'; Default: 'Pending'; Indexed.	Controls visibility and available actions.

attributes	Job	Map (String keys, String values)	Dynamic key-value pairs for job details relevant to the category but not part of the fixed schema.	Optional (nullable).	N/A
startedAt	Job	Date	Timestamp indicating when the job status was changed to 'In Progress'.	Optional (nullable).	N/A
completedAt	Job	Date	Timestamp indicating when the job status was changed to 'Completed'.	Optional (nullable).	N/A
jobId	Bid	ObjectId (String)	References the Job that this bid is placed on.	Required; Must be a valid Job _id; Indexed.	Refers to Job collection.
seekerId	Bid	ObjectId (String)	Denormalized reference to the Seeker who owns the job this bid is for.	Required; Must be a valid User _id with userType 'Seeker'; Indexed.	Refers to User collection (via Job).
providerId	Bid	ObjectId (String)	References the User (Provider) who submitted this bid.	Required; Must be a valid User _id with userType 'Provider'; Indexed; Compound unique index with jobId.	Refers to User collection.

amount	Bid	Decimal 128 (BSON Type)	The monetary amount the provider is bidding to complete the job.	Required; Stored with high precision.	N/A
description	Bid	String	An optional message from the provider regarding their bid.	Optional (nullable).	N/A
status	Bid	String (enum)	The current state of the bid itself.	Required; Enum: 'Pending', 'Accepted', 'Rejected'; Default: 'Pending'; Indexed.	Controls bid visibility and actions.
jobId	Payment	ObjectId (String)	References the Job for which this payment record was created.	Required; Must be a valid Job _id; Unique.	Refers to Job collection.
seekerId	Payment	ObjectId (String)	References the Seeker who initiated the payment.	Required; Must be a valid User _id with userType 'Seeker'.	Refers to User collection.
providerId	Payment	ObjectId (String)	References the Provider who is the recipient of the payment.	Required; Must be a valid User _id with userType 'Provider'.	Refers to User collection.
amount	Payment	Decimal 128 (BSON Type)	The amount of the payment transaction.	Required; Stored with high precision; Should typically match the job's agreedAmount.	N/A

paymentMethod	Payment	String	The method used for the payment transaction (e.g., 'Mpesa', 'Bank Transfer').	Required.	N/A
paymentStatus	Payment	String (enum)	The current state of the payment transaction.	Required; Enum: 'Pending', 'Completed', 'Failed'; Default: 'Pending'; Indexed.	N/A
transactionReference	Payment	String	A unique identifier or reference provided by the payment gateway or system.	Optional (nullable); Unique (sparse allows multiple nulls).	External link to payment gateway record.
categoryName	Category	String	The official name of the service category.	Required; Unique; Trimmed.	N/A
description	Category	String	A brief explanation of what services are included in this category.	Optional (nullable).	N/A
requiredAttributes	Category	Map (String keys, String values)	Defines key attributes expected or required for jobs/providers within this category (e.g., skill levels).	Optional (nullable); Map keys could be attribute names, values could indicate data type or description.	N/A

name	Location	String	The name of the predefined location (e.g., 'Gaborone', 'Francistown').	Required; Unique.	N/A
coordinates	Location	GeoJSON Point	The geographical coordinates representing the central point of the predefined location.	Required; Contains nested type ('Point') and coordinates [longitude, latitude] array.	Used for referencing predefined locations.
userId	Address	ObjectId (String)	References the User associated with this address.	Optional (nullable); Must be a valid User _id; Indexed.	Refers to User collection.
street	Address	String	The street address component.	Required.	N/A
city	Address	String	The city component of the address.	Required.	N/A
state	Address	String	The state, district, or region component of the address.	Optional (nullable).	N/A
postalCode	Address	String	The postal or zip code.	Optional (nullable).	N/A
country	Address	String	The country component of the address.	Required.	N/A
coordinates	Address	GeoJSON Point	Optional geographical coordinates derived from the address.	Optional (nullable); Contains nested type ('Point') and coordinates [longitude, latitude] array.	N/A

jobId	Rating	ObjectId (String)	References the Job related to this rating.	Required; Must be a valid Job _id; Unique (assuming one rating record per job).	Refers to Job collection.
reviewerId	Rating	ObjectId (String)	References the User who gave the rating.	Required; Must be a valid User _id; Indexed.	Refers to User collection.
revieweeId	Rating	ObjectId (String)	References the User who received the rating.	Required; Must be a valid User _id; Indexed.	Refers to User collection.
rating	Rating	Number	The numerical rating value.	Required; Must be an integer between 1 and 5 (assuming scale).	N/A
comment	Rating	String	The textual review provided by the reviewer.	Optional (nullable).	N/A
userId	Notification	ObjectId (String)	References the User who is the intended recipient of the notification.	Required; Must be a valid User _id; Indexed.	Refers to User collection.
type	Notification	String	The type of event or action that triggered the notification (e.g., 'bid_placed', 'job_accepted').	Required.	Used to categorize notifications.
title	Notification	String	A brief, user-friendly title for the notification.	Required.	N/A

message	Notification	String	The main content or body of the notification message.	Required.	N/A
read	Notification	Boolean	Indicates whether the user has viewed or interacted with the notification.	Required; Default: false; Indexed.	Used for filtering read/unread notifications.
data	Notification	Map (String keys, Mixed values)	Optional structured data relevant to the notification, such as related jobId or bidId.	Optional (nullable); Can store various data types; Useful for linking notifications to specific records.	May contain references to other collections (e.g., Job, Bid).
createdAt	All	Date	Timestamp indicating when the document was first created.	Automatically added by Mongoose timestamps: true; Immutable after creation.	N/A
updatedAt	All	Date	Timestamp indicating when the document was last updated.	Automatically added by Mongoose timestamps: true; Updated on modification.	N/A

D.3 GeoJSON Usage and Indexing Strategies

The platform utilizes GeoJSON `Point` types for storing geographical coordinates related to job locations and provider base locations.

- **GeoJSON `Point`:** This standard format `{ type: 'Point', coordinates: [longitude, latitude] }` is used to store single geographical points. It's crucial to remember that the `coordinates` array uses **[longitude, latitude]** order, not the common [latitude, longitude] order.
- **2dsphere Index:** A `2dsphere` index is applied to the `location.coordinates` field in the `Job` schema and the `providerAttributes.baseLocation.coordinates` field in the `User` schema. This special index is necessary to perform geospatial queries efficiently, such as:
    - Finding jobs within a certain radius of a provider's location.
    - Finding providers within a certain radius of a job location.
    - Calculating distances between points.

These indexing strategies ensure that location-based searches, which are fundamental to a service marketplace, are performant even as the number of jobs and users grows. Other standard indexes are applied to `ObjectId` fields used for common lookups (e.g., `seekerId`, `providerId`, `jobId`, `userId` in Notification) and fields used for filtering (e.g., `status`, `read`).