Week 8 - LAQ's

Instructions

Explain mongoose Scheme and its methods

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward way to model your data and manage relationships between data. A Mongoose schema defines the structure of documents within a collection and provides a way to enforce data validation, define default values, and set up middleware for various operations. Here’s an overview of Mongoose schemas and some of their key methods:

Mongoose Schema

A Mongoose schema is defined using the Schema class provided by Mongoose. It outlines the shape of the documents in a collection, including the fields, their types, and any validation rules.

Creating a Schema

Here's a basic example of how to create a Mongoose schema:

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


const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  age: { type: Number, min: 0 },
  createdAt: { type: Date, default: Date.now }
});
```

Schema Methods

Mongoose schemas can have various methods associated with them. Here are some commonly used methods:

1. save():

- Used to save a document to the database.
- Example:

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

const user = new User({ name: 'Alice', email: 'alice@example.com', age: 25 });

user.save()
   .then(() => console.log('User  saved'))
   .catch(err => console.error(err));
```

2. find():

- Used to retrieve documents from a collection.
- Example:

```
User.find({ age: { $gte: 18 } })
   .then(users => console.log(users))
   .catch(err => console.error(err));
```

3. findById():

- Retrieves a document by its unique identifier.
- Example:

```
User.findById('someUser Id')
```

```
  .then(user => console.log(user))
  .catch(err => console.error(err));
```

4. updateOne():

- Updates a single document that matches the provided criteria.
- Example:

```
User.updateOne({ _id: 'someUser Id' }, { age: 26 })
  .then(result => console.log(result))
  .catch(err => console.error(err));
```

5. deleteOne():

- Deletes a single document that matches the criteria.
- Example:

```
User.deleteOne({ _id: 'someUser Id' })
  .then(result => console.log(result))
  .catch(err => console.error(err));
```

Schema Middleware

Mongoose schemas can also have middleware (also known as hooks), which are functions that are executed at certain stages of the document lifecycle. Common middleware types include:

- pre(): Runs before a certain operation (e.g., save, remove).
- post(): Runs after a certain operation.

Example of using middleware:

```
userSchema.pre('save', function(next) {
  // Perform some action before saving
```

```
    console.log('About to save a user');
    next();
});
```

Validation

Mongoose allows you to set validation rules directly in the schema definition. You can specify required fields, data types, minimum/maximum values, and custom validation logic.

Example of validation:

```
const userSchema = new mongoose.Schema({
    email: {
        type: String,
        required: true,
        validate: {
            validator: function(v) {
                return /.+\@.+\..+/.test(v); // Simple email regex
            },
            message: props => `${props.value} is not a valid email!`
        }
    }
});
```

Conclusion

Mongoose schemas are a powerful way to define the structure of your MongoDB documents, enforce validation, and manage relationships between data. With methods for creating, reading, updating, and deleting documents, as well as support for middleware and validation, Mongoose provides a robust framework for working with MongoDB in Node.js applications.