Mongoose Exercise

Exercises

OVERVIEW

In this exercise, you will have the opportunity to practice using Mongoose, a flexible MongoDB object modeling (ODM) library for Node.js. You will focus on implementing CRUD (Create, Read, Update, Delete) operations and exploring the concept of referencing other documents within a MongoDB database.

GOALS

1. Creating Mongoose Models and Schemas
2. Implementing CRUD Operations
3. Understanding Referencing documents in MongoDB

SPECIFICATIONS

In this exercise, you will create an Event Management System that allows users to manage and organize events. The system will provide features for creating events and managing them.

Exercises

Setup

- Create a github repository and clone it
  - make sure it was initialized with a README.md file and a .gitignore file for node
- Initialize the package.json inside the root directory
- Create a cluster on mongoDB Atlas if you haven't got one running

Exercise 01

- Create a basic express server
    - use express.json() middleware
    - setup environment variables using dotenv` package
        - Add the mongodb connection string as an environment variable, make sure the connection string has the credentials of a user that has access to the cluster
        - Make sure the cluster allows connection from anywhere or at least from your own ip address
- Create a `db.js` file in config folder and setup a connection to the db
    - Check the documentation if you need help or reference the slides
- Require the db connection file in index.js
    - If the connection fails you make sure that the environment variable has a value, if you require db,js before dotenv/config then it will be undefined
- Create a `models/user.js` file that contains the schema and model for the users collection
    - **Create** the **userSchema** with the following fields
        - **name** => String, Required
        - **email** => String, Unique, Required
        - **age** => Number, min: 18,
        - **phoneNumber** => String, Unique, Required
        - **isActive =>** Boolean, default: true
        - **timeStamps**
    - **Create** and **export** the user model

Exercise 02

- In routes folder create users.js
    - You will need to create a router for /api/users don't forget to create controllers folder and create users.js that will export all the routes handler functions
    - Create an endpoint that accepts a **POST** request on path '**/api/users**` to save a new user in the db
        - HINT: Use the user model to create a new user, check the documentation or slides for more information
    - Create an endpoint that accepts a **GET** request on path '**/api/users**` to retrieve all the **active** users from the db

        - HINT: read the documentation for more information about querying
        - OPTIONAL: use query parameters to provide the value for the active query parameter `/users?active=true` or `/users?active=false`
    - Create an endpoint that accepts a **GET** request on path '**/users/:id**` to retrieve the user with the matching url parameter from the db
    - Create an endpoint that accepts a **PUT** request on path '**/users/:id**` to update the user with the matching url parameter from the db
    - Create an endpoint that accepts a **DELETE** request on path '**/users/:id**` to update the user with the matching url parameter from the db

Exercise 03

- Create a `models/event.js` file that contains the schema and model for the events collection
    - **Create** the **eventSchema** with the following fields
        - **name**=> String, Required
        - **description**=> String, Required
        - **location**=> String, Required,
    - Update the schema to add a new field named `organizer` that references a user User from the users collection
        - mongoose.Schema.Types.ObjectId is a mongoose type for document objectId
        - the ref key takes the name of the model name of the other collection (Should be the same name as the exported model)
- **Create** and **export** the event model

Exercise 04

- In routes folder create events.js
    - You will need to create a router for /api/events don't forget to create events.js in the controllers folder that will export all the routes handler functions
    - Create an endpoint that accepts a **POST** request on path '**/api/events**` to save a new event in the db
        - make sure that the objectId belongs to a user (copy a user _id)
    - Create an endpoint that accepts a **GET** request on path '**/api/events**` to retrieve all the events from the db

    - Create an endpoint that accepts a **GET** request on path ‘**/api/events/:id**` to retrieve the event with the matching url parameter from the db
        - You will need to use populate on the organizer field so it brings the user document as well
    - Create endpoint that accepts a **PUT** request on path **‘/api/events/:id’** to update event
    - Create endpoint that accepts a **DELETE** request on path **‘/api/events/:id’** to delete event

Exercise 05

- Update the event schema to have a new field named `attendees` that references array of users who are attending the event
    - mongoose.Schema.Types.ObjectId is a mongoose type for document objectId
    - the ref key takes the name of the model name of the other collection (Should be the same name as the exported model)
- Create an endpoint that accepts a **PATCH** request on path ‘**/api/events/:id/join**` to add a user to the attendees array for the event with the matching url parameter from the db
    - HINT: send the user id using the request body, you could use the $push to push a new value to the array while doing findOneAndUpdate
    - or you could fetch the event then update it using `Array.push()`and then save it using `.save()`

Exercise 06

- Update the **GET** request on path ‘**/api/events/:id**` to populate the attendees array to show all the users info