User Management System

# Technologies used:

Express.js for creating Api’s with nodejs, MongoDb for database, Mongoose to communicate with the database as a ODM(Object Data Model), Postman/Thunderclient for Api testing.

# Project Setup In Local Environment:

1. Clone the repository –

git clone <https://github.com/mohiyaddeen7/userManagementSystemTask.git>

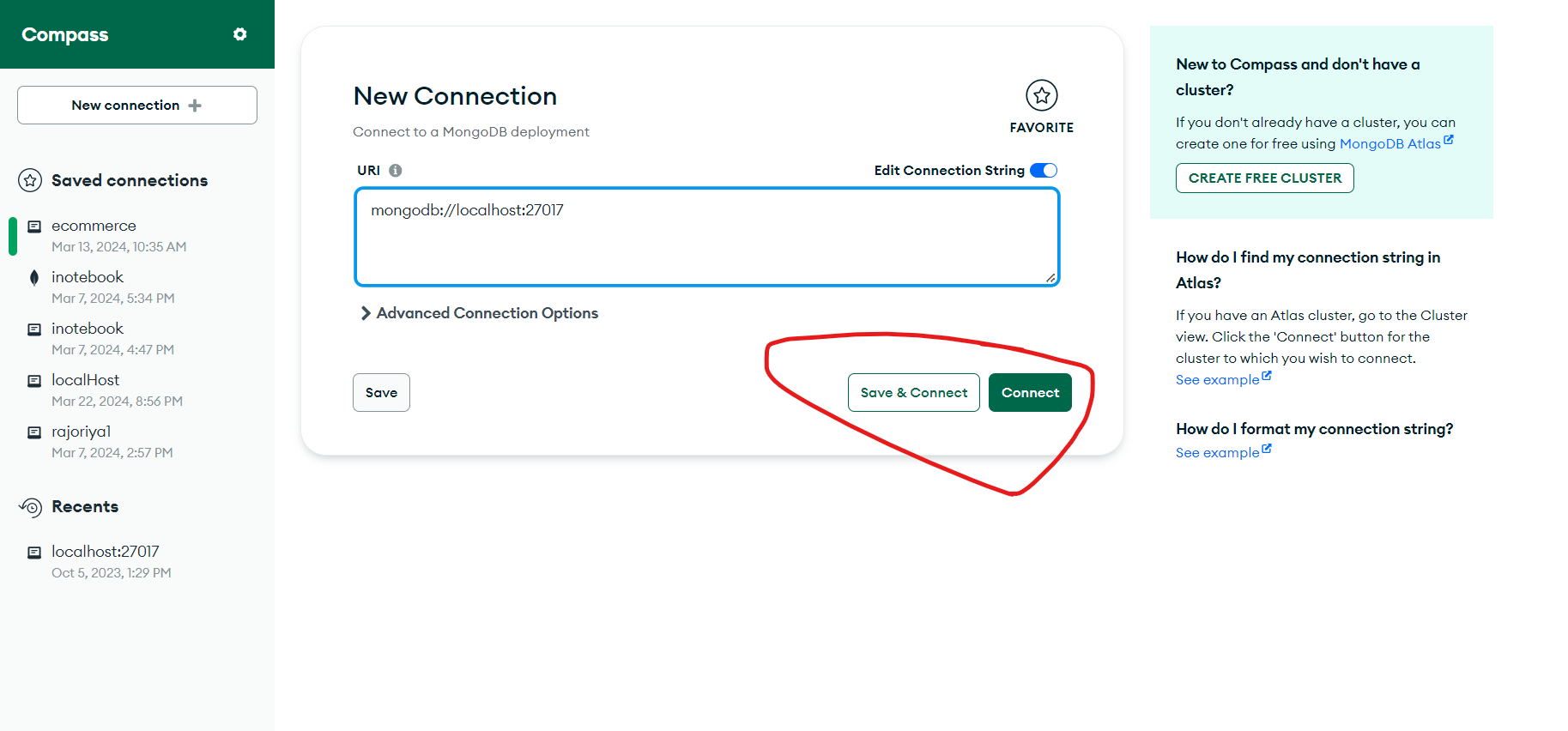
1. Install all the npm packages by opening the project in vs code and running the command in vs code terminal –

npm install-server

(for this to run you should have node, and npm installed in your os, to install npm and node - [npm and node install guide](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm))

1. Now you need to setup your mongodb as I have used mongoDb as a database, you need to have mongoDB compass installed, after installing mongoDb compass create a new connection and as

You can see you will have your mongoURI like this : mongodb://localhost:27017 copy this before clicking on save and connect



1. Now in the project directory create a .env file with this fields  
   mongodbURI = mongodb://localhost:27017/userManagementDb

port=5000

1. Now we have all things setup now you can run the server by running the command

npm start-server

1. And now our server is successfully started if you see

App listening on : http://localhost:5000

Connected to database successfully

1. Now you can test the api’s with the predefined collections that you can find in the docs folder inside api collection with a file name – userManagementSystemApi
2. You can import this collection in postman or thunderclient to test the api’s

# Api Endpoints:

POST -> /api/users/addUser – adds user to the database with fields in the body of the request

that are –

name(String, Name cannot be more than 50 characters long, Required) ,   
email(String,valid email address, Unique, Required),   
age(Integer, Age must be at least 18 years and Age cannot be more than 100 years, Required), gender(String, Enum- Gender must be either Male, Female, or Other, Required),   
address(String, Address cannot be more than 100 characters long., Optional),   
mobileNo(String, Please fill a valid 10-digit mobile number,Unique, Required)

GET -> /api/users/getUsers – gets all the users who are not marked as deleted

GET -> /api/users/getUsersByFilter – gets all the users with proper validation for query parameters and proper pagination in the response.

Accepted query parameters are :

page(Integer,it indicates page number),  
 limit(Integer, it indicates number of records per page),  
age(Integer,must be a number),  
name(String),userId(ObjectId),  
email(String, must be a valid email address),   
gender(String, Gender must be either Male, Female, or Other),  
mobileNo(Integer, Please fill a valid 10-digit mobile number),  
deleted(Boolean(true,false),it indicates that whether the records that are marked deleted should be retrieved(true) or marked as not deleted(false) and by default its false)

example request endpoint - http://localhost:5000/api/users/getUsersByFilter?page=1&limit=5&gender=Male&deleted=true

PUT -> /api/users/updateUser/userId=:userId – updates user record by userId and the fields that need to be updated are in the body of the request

PUT -> /api/users/updateUser/emailId=: emailed - updates user record by emailId and the fields that need to be updated are in the body of the request

DELETE -> /api/users /softDelete/userId=:userId – (Soft Deletion) – marks the user as deleted whose userId matches the given userId

DELETE -> /api/users /softDelete/ emailId =: emaild - (Soft Deletion) – marks the user as deleted whose emailId matches the given emailId

DELETE -> /api/users /hardDelete/userId=:userId - (Hard Deletion) – removes the user from the databse whose userId matches the given userId

DELETE -> /api/users /hardDelete/ emailId =: emailed - (Hard Deletion) – removes the user from the databse whose emailId matches the given emailId