2:47

we will be using react as our framework react router for navigation formik and yup for form and form validation Redux

toolkit for our state management with Redux persist to store in local storage and react drop zone for image uploads

for the back end we are going to be using node.js as our runtime, express.js as our backend framework Mongoose for managing our database Json web token for authentication and multer for file uploading

npm i npx use?

ChatGPT

npm (Node Package Manager): npm is a package manager for Node.js. It is used to install and manage packages (libraries, tools,

etc.) for Node.js applications. You use the npm command to interact with the npm registry, install dependencies, and manage your project.

npx:

npx is a tool that comes with npm, starting from version 5.2.0. It allows you to execute Node.js packages directly, without

having to install them globally. npx makes it easy to run binaries from packages as if they were installed globally, without

actually installing them globally.

npm i -g nodemon?

ChatGPT

The command npm i -g nodemon is used to install the nodemon package globally on your system

using npm.

Here's a breakdown of each part of the command:

npm: This is the Node Package Manager, a command-line tool for managing Node.js packages.

i: This is short for install, the command used to install packages.

-g: This flag stands for "global," and when used with the install command, it installs the specified package globally on your system. Global installations are typically used for packages that provide command-line utilities that you want to use across different projects.

nodemon: This is the name of the package you want to install. nodemon is a utility that monitors for changes in your source code files and automatically restarts your Node.js application when changes are detected. It's commonly used during development to streamline the development process.

mkdir server

cd server

server cmd + npm i express body-parser bcrypt cors dotenv gridfs-stream multer multer-gridfs-storage helmet morgan jsonwebtoken mongoose

The command npm i express body-parser bcrypt cors dotenv gridfs-stream multer multer-gridfs-storage helmet morgan jsonwebtoken mongoose is used to install

multiple Node.js packages for building a web application. Let's break down each package:

express: A web application framework for Node.js that simplifies

the process of building robust and scalable web applications.

body-parser: Middleware for handling HTTP request data, particularly parsing the body of incoming

requests. It is often used with Express.

bcrypt: A library for hashing passwords. It is commonly used for securely storing user passwords.

cors:

Middleware for handling Cross-Origin Resource Sharing (CORS) in Express applications. It allows or restricts access to resources on a web server

based on the domain making the request.

dotenv: A zero-dependency module that loads environment variables from a .env file into the process.env

object.

gridfs-stream: A Node.js library for streaming files to and from MongoDB's GridFS.

multer: Middleware for handling multipart/form-data, which is commonly used for

file uploads. It works well with Express.

multer-gridfs-storage: A Multer storage engine for MongoDB's GridFS. It works with Multer to store uploaded

files directly to GridFS.

helmet: A collection of middleware functions to help secure Express applications by setting various HTTP headers.

morgan: A logging

middleware for Express. It logs HTTP requests to the console, making it useful for debugging and monitoring.

jsonwebtoken: A library for generating

and verifying JSON Web Tokens (JWT). It is commonly used for authentication and authorization.

mongoose: A MongoDB object modeling tool for Node.js.

It provides a higher-level, schema-based abstraction for interacting with MongoDB.

By running the command npm i express body-parser bcrypt cors dotenv gridfs-stream

multer multer-gridfs-storage helmet morgan jsonwebtoken mongoose, you are installing these packages as dependencies for your Node.js project. These packages are commonly

used when building web applications with Node.js and Express, and they cover areas such as web server setup, data parsing, authentication,

logging, and more.

server cmd + npm init -y

ChatGPT

The command npm init -y is a shortcut for initializing a new Node.js project with default values, without

requiring you to go through the interactive setup prompted by npm init.

Here's what each part of the command does:

npm init: This command initializes a new Node.js project. It prompts you with a series of questions about your project (e.g., package name, version,

description, entry point, test command, etc.), and it creates a package.json file based on your responses.

-y: This is a shorthand option

that stands for "yes." When you include -y with npm init, it automatically accepts the default values for all the questions,

effectively skipping the interactive setup. This is useful when you want to quickly create a new project with default settings.

So, when you run npm init -y, you're telling npm to initialize a new project with default values, without requiring any manual input from you. This is especially handy for quickly setting up a new project when you're okay with the default configuration provided by npm.

package.json changes

{

"name": "server",

"version": "1.0.0",

"description": "",

"main": "index.js",

"type": "module", <--------------------- added

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC"

}

create index.js in server

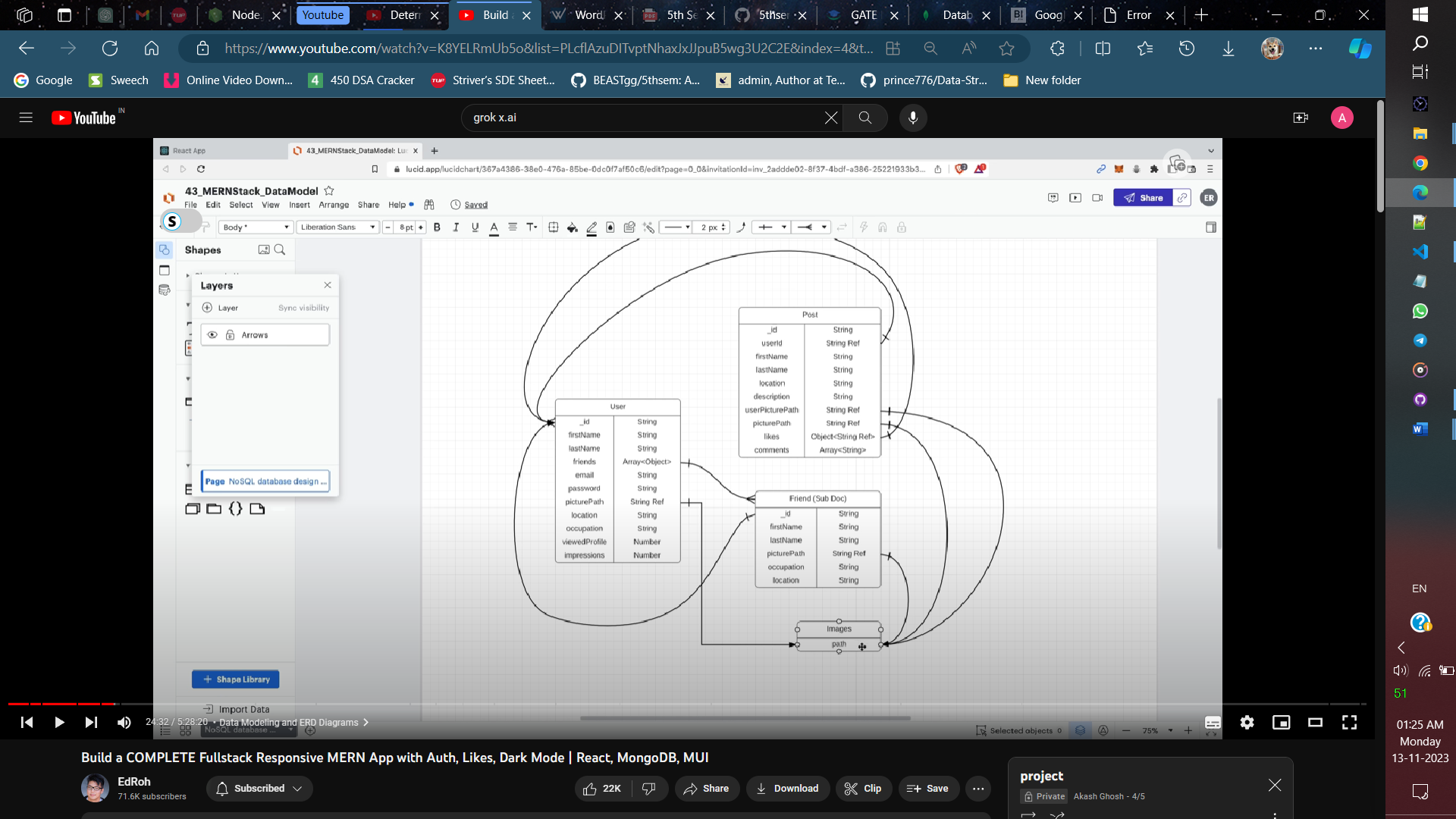
setup database at mongoDB website and coonect with application --make sure to add to use current IP Adress

create ".env" file in server

The **.env** file, short for "environment," is a configuration file used to store environment variables for a software application. Environment variables are key-value pairs that hold configuration settings or sensitive information needed by an application. The primary purpose of using an **.env** file is to separate configuration details from the code, making it easier to manage and deploy applications in different environments (such as development, testing, and production).

Planning for DB design and ER diagrams for DBMS

It will help us in developing relations



Now do Authentication vs Authorization (Hardest Part)

Authentication verifies the identity of a user or system.

Authorization determines what actions or resources an authenticated user or system is

allowed to access.

Example Scenario:

Authentication: When you log into your email account by providing your username and password, the system checks if

the credentials match the stored information to verify your identity.

Authorization: After authentication, the system checks your user profile to determine what

actions you're allowed to perform, such as reading, sending, or deleting emails. This is the authorization step.

Create folder models -> User.js (first do user Schema)

Create folder controllers -> auth.js

Registration work all done here and password is hashed

server -> Create folder public ->create folder assets (to store images)

server -> Create folder routes ->create file auth.js

route folder where we have the path and the routes for every type of feature

Now add logging section in controllers/auth.js it will be used for authentication

After authentication we will start for user authorization

Server/create folder middleware/ create file auth.js

Now its time to fetch detail for a user like friends, personal details, working etc so now we will do route setup

Controllers/create users.js it will store all the logic

Now we will handle user posts from index.js – user should be able to upload a picture

Routes/create file posts.js directs the user feeds

Controllers/create file posts.js working of like and comments

Models/create file Post.js stores the post details – user name of the post etc

Create folder data/create file index.js it just contains dummy data