# **Backend**

Server is a software that serves something.

**Working of Backend:**

Browser API (Response) Backend Database(Another Continent)

Data/File/Third Party API ` Handling in Backend

NodeJS -> Js Runtime

**Files in Backend:**

Package.json , .env, (readme,git,..)

Src:

-> index ->DB connect

->App -> configuration(main)

->constants ->enums, DB-name

**Directory that should be made:**

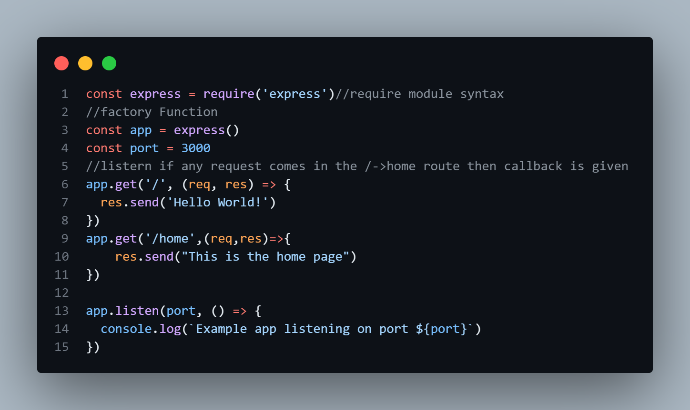
1. DB directory
2. Models -> data structure
3. Controllers ->methods and functionality
4. Routes -> /home or /about
5. Middle wares
6. Utils -> mail

get(request)

Computer Express server

response Listen: /:home route

/login : login setup

 **Express:**

1. Hello world

2. And production application:

Package -> dot env

->install: npm install dotenv

->creating a file .env

A. .env file:

PORT = 4000

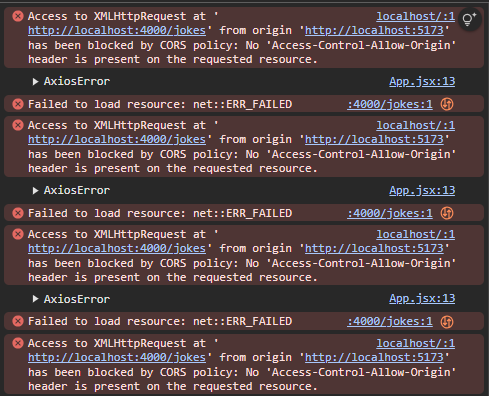
B.

Reads the port number from an environment variable or defaults to 3000



Starts the server and listens on the specified port

.

#Backend and Frontend



->Sends a GET request to the backend API /api/jokes.

->This request is proxied to the backend using Vite’s proxy configuration.

-jokes: Holds the list of jokes fetched from the backend.

-setjokes: A function to update the jokes state.

-Initial state is an empty array [].

useEffect: A React hook that runs side effects (like fetching data).

useState: A React hook to manage component-level state.

axios is used to make HTTP requests.

*$CORS policy means : only allowing which has same origin: not all are allowed:*

*: cross origin request:*

*Url, port then only they have same origin to connect the backend and frontend,*

*We can slove by making making the url/domain white list, or mark \**

***Solving Error:***

1. *Using the proxy in frontend:[vite.config.js]*



server.proxy:

->Redirects frontend requests starting with /api to the backend running at http://localhost:4000.

* This avoids Cross-Origin Resource Sharing (CORS) issues during development.

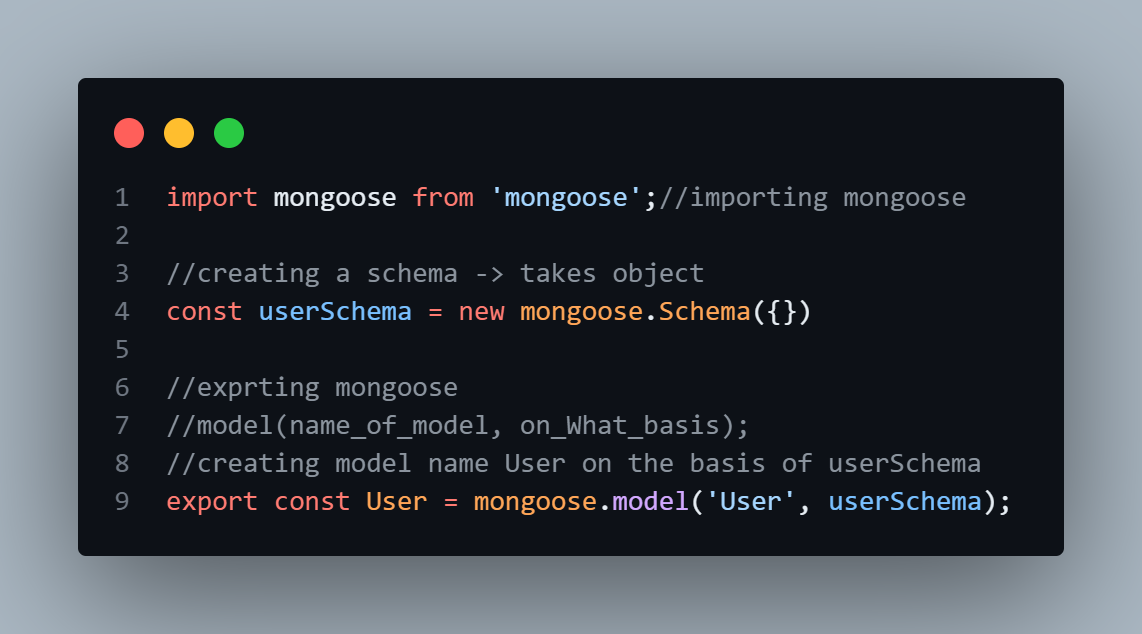


This hook runs after the component renders.

*When proxy is used, this request will act like the same as the origin then cros error will not occur since: ther are both from the same origin.*

# Mongoose [MongoDB]:

A popular ODM (Object Data Modeling) library for MongoDB and Node.js, used for database operations.



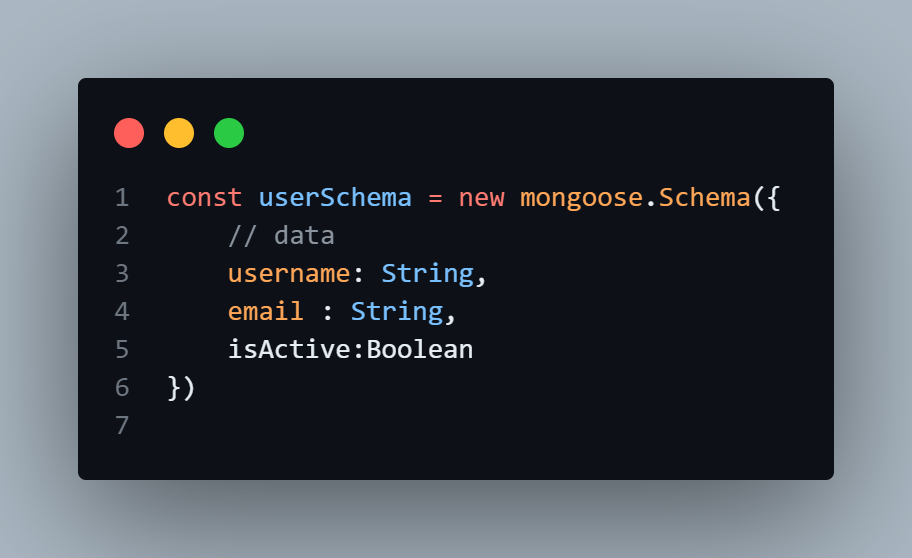
In database User will be converted to users

Data saving in mongoose:



required: [true, 'password is required']

A custom message can also be sent



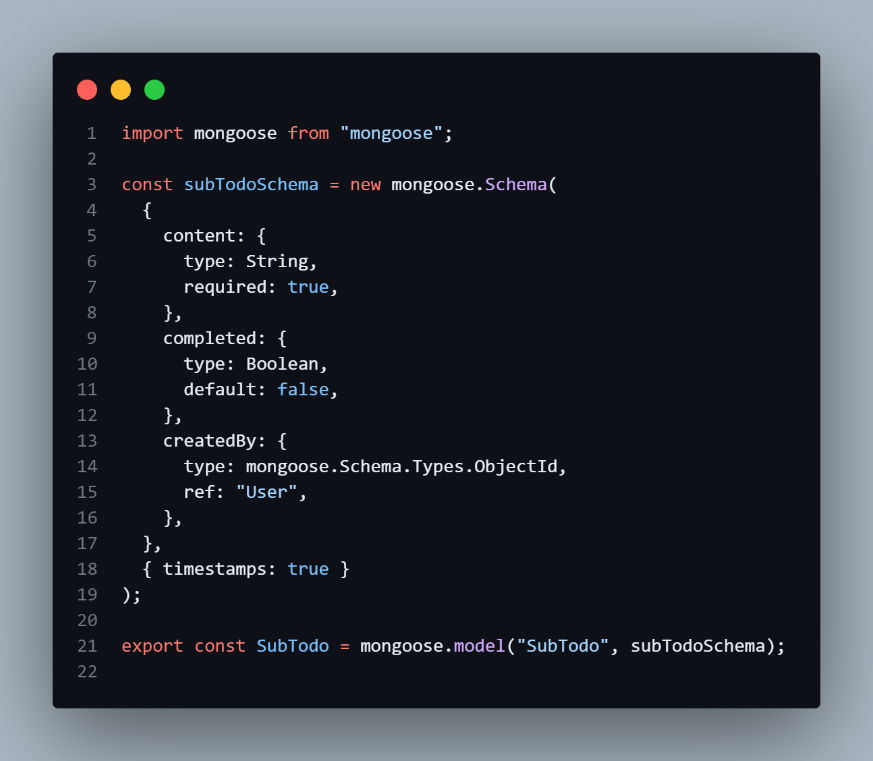
}, {timestamps: true})

To add created at time and updated at time in database

Type 1 OR, Type 2 [Most reliable]

Connecting documents[relate]:

1. User:user.models.js
2. todo.mode.js:
3. sub\_todos.models.js:



Note: Always use asyc await and try catch or promises to handle error in data base connection since it takes time and it’s assumed to be in another continent and error occurs during the connection of database.

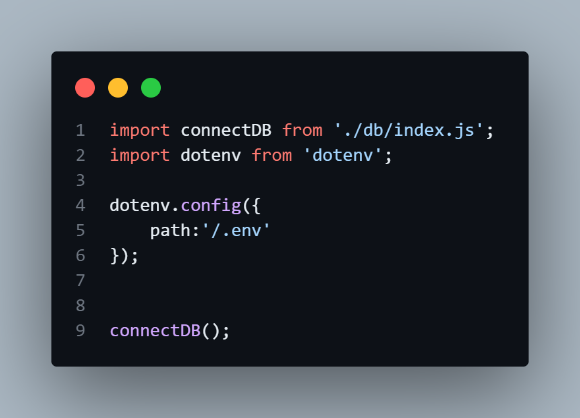
# Connection to Database:

* + - 1. First Approach:
      2. Better Approach

db/index.js->



connectionInstance.connection.host contains the host name (e.g., cluster0.mongodb.net), so this confirms the connection.

index.js -> main file constants.js

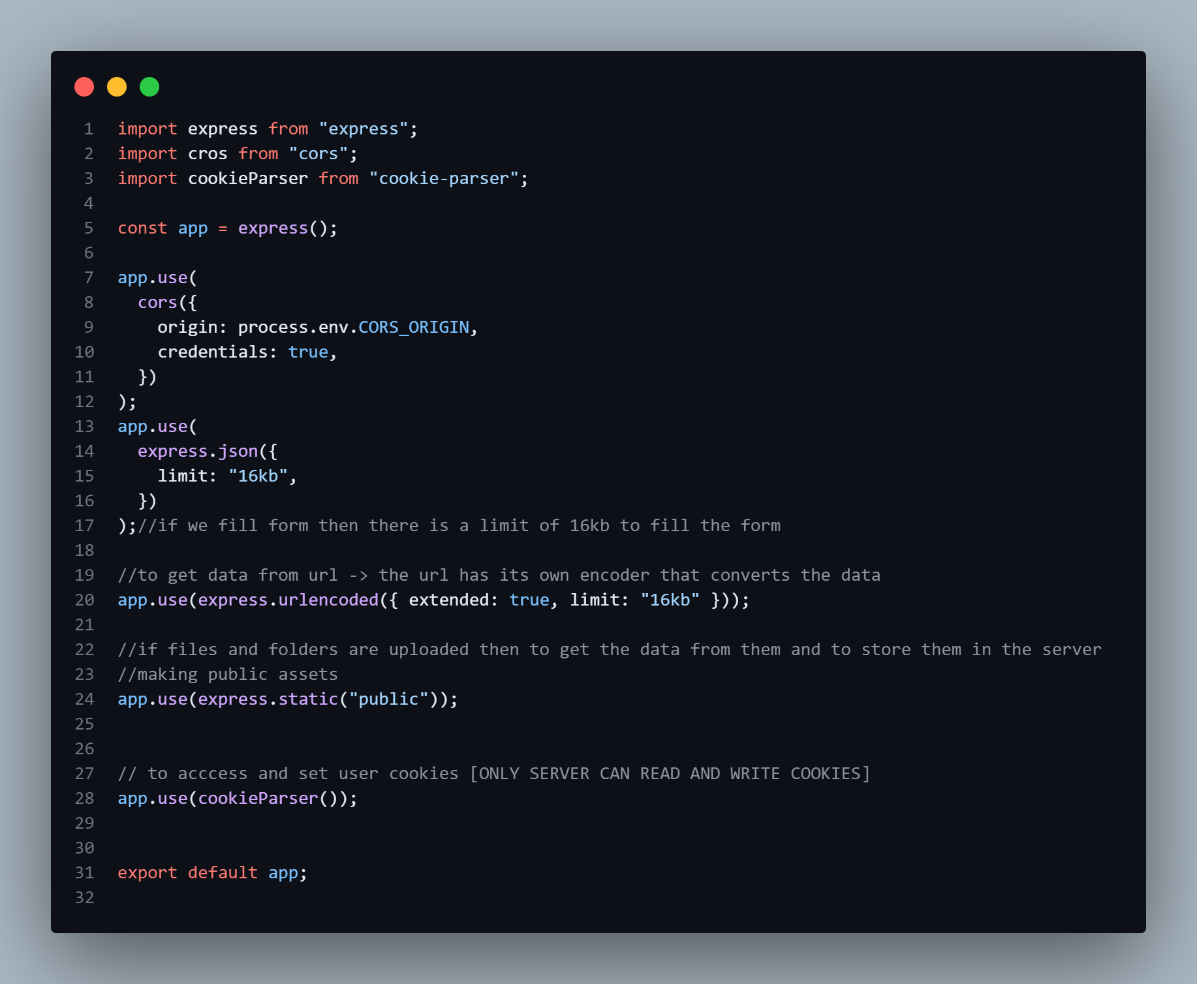
# Cookies:

Cookies are small pieces of data stored on a user's device by a web browser while they are browsing a website. They are used to remember information about the user or the session, enabling websites to provide a more personalized and consistent experience.

1. App.js

CORS\_ORIGIN = \*

This accepts all the request..



 **Purpose**:

* This is the foundation of your server. It defines the Express app and configures the middleware stack to handle requests, process incoming data, and serve static files.

 **Middleware explained**:

1. **CORS (cors)**: Allows cross-origin requests. It enables the server to accept requests from a specific origin (set by process.env.CORS\_ORIGIN) and share credentials like cookies.
2. **express.json and express.urlencoded**: These handle parsing of JSON and URL-encoded data in incoming requests with a data size limit of 16 KB.
3. **express.static**: Serves static files (like images, stylesheets, or JavaScript) from the public folder.
4. **cookie-parser**: Enables the server to read/write cookies in HTTP requests.

 **Usage**: This configured app is exported for use in your main server entry point (e.g., server.js or index.js). It processes and routes all incoming HTTP requests.

We can also user the white list to accept the specific **URL** only.

1. asyncHandler.js
2. apierror



 A custom error class to create detailed, consistent error responses.

 Extends the built-in Error class, adding properties like statusCode, errors, and a stack trace.

The ApiError class **extends** the built-in Error class, making it a subclass of Error.

#### **Why** super **is needed:**

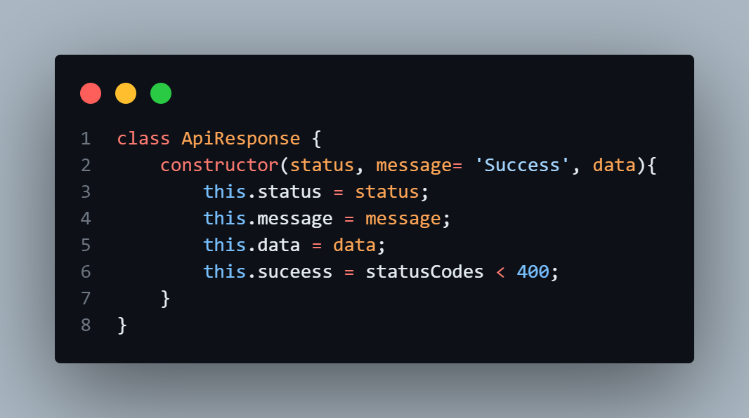
* When extending a parent class (Error), you must call the parent's constructor using super() to properly initialize the inherited properties.



 Ensures consistent error handling for asynchronous request handlers (e.g., when using async/await).

 If an error occurs in a request handler, it will automatically be passed to Express' error-handling middleware using next(err).

If the handler throws an error (e.g., due to a failed database call), the error is caught and forwarded to the error-handling middleware.

1. Apirespose

 A standardized way to send success responses.

 Encapsulates HTTP status, message, and data into a consistent format.