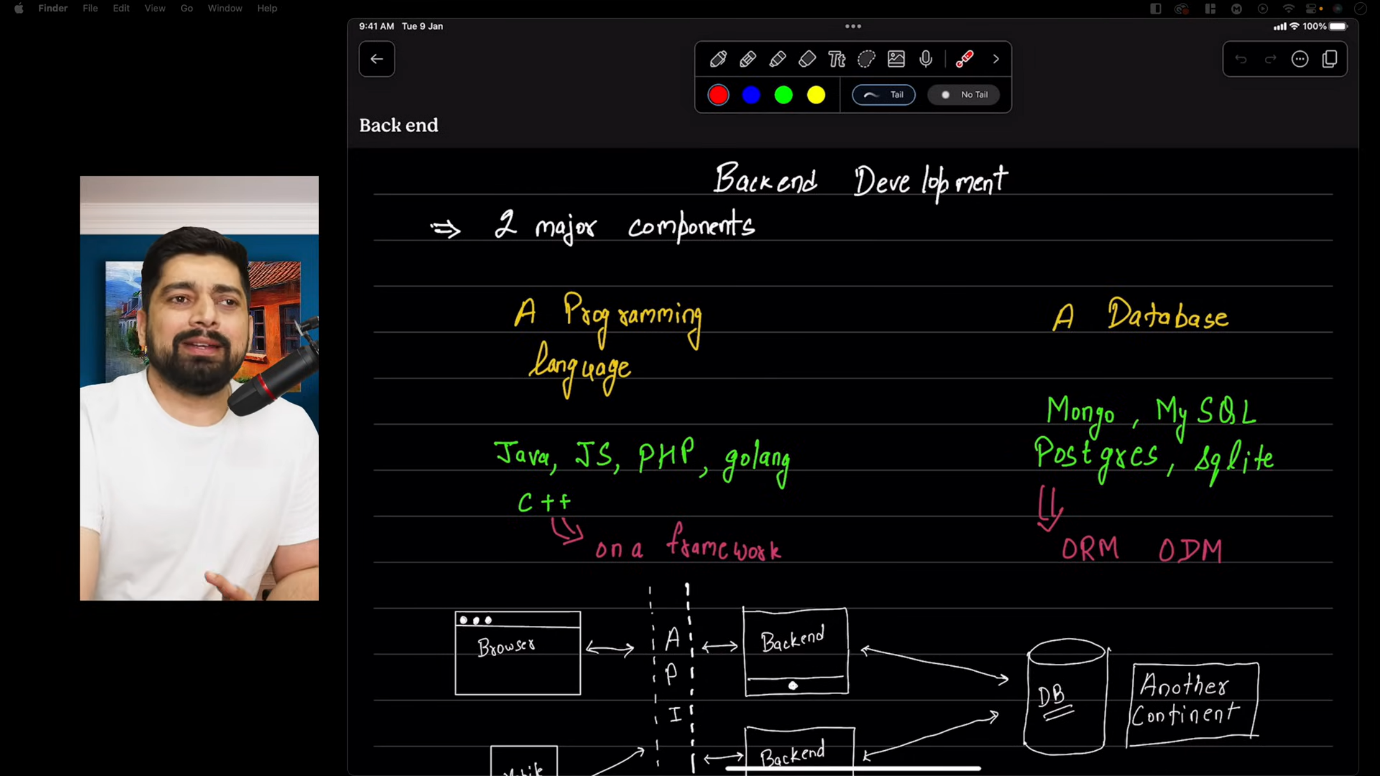
Backend Basics &

Javascript Backend Roadmap



We will need a programming language for writing business logic or data queries for storing or retrieving data from database. We may also need framework of that particular language for ease of writing complex code.

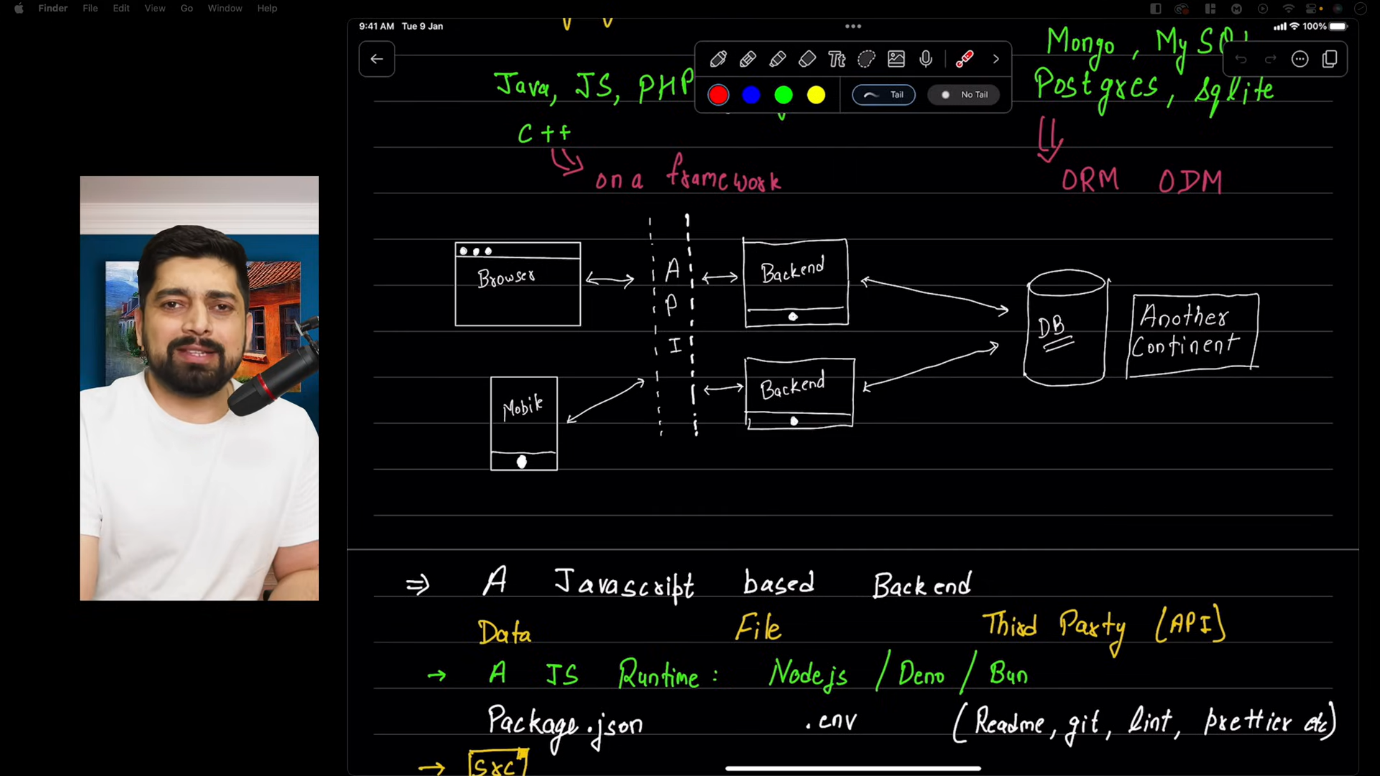
Languages (& their frameworks) for backend development =>

Java(springboot), Js(React,Angular,etc), PHP(Laravel, CakePHP), Golang(), C++(Crow)

Databases (& their frameworks) for backend development =>

Mongo, MySQL, Postgress, Sqlite and frameworks – (ORM, ODM)

Server is nothing but a software that serves.



**Flow of a full stack application =>**

**Frontend**(will do a query, like is 2+2 =4? If true send some response or add something in result, if false send something else **or**

we’ll get some data from frontend for checking, if correct send some response, otherwise send something else like error or any message)

**Database**(always resides in another continent)

**BACKEND** (We’ll write a backend on machine which is continuously running on servers. Suppose we visit a url like /login, /signup,etc. that will be detected by frameworks. And acc to the routes we visited, specific function for that route will be called from backend. **In Backend, we will generally write different functions , and they will interact with database , and we will send the response from database in form of an API**)

**API**(Response from backend who fetched it from database , eg- true, false, object , array,etc.)

NOW, MAYBE THESE CALLS(or queries from frontend) ARISE FROM BROWSER OR MOBILE

**Now, in JAVASCRIPT based backend we will deal with three things-**

**DATA** - Either we have to deal with data directly (username, password, number, strings, objects, arrays)

**FILE –** OR We have to handle file (image, pdf, video, etc)

**THIRD PARTY (API) –** (Dealing with third party APIs. Suppose we have to google login or we have to upload file on AWS or sending email, sms, calls, file upload download from third party APIs)

* **We’ll need a JS Runtime (Nodejs/Deno/Bun)**
* **FILE STRUCTURE :**

**Package.json -** This file will contain all the dependencies, libraries

**.env -** This will store environment variables.

**(Readme, git, lint, prettier, etc) -** Some other basic files.

* **src**

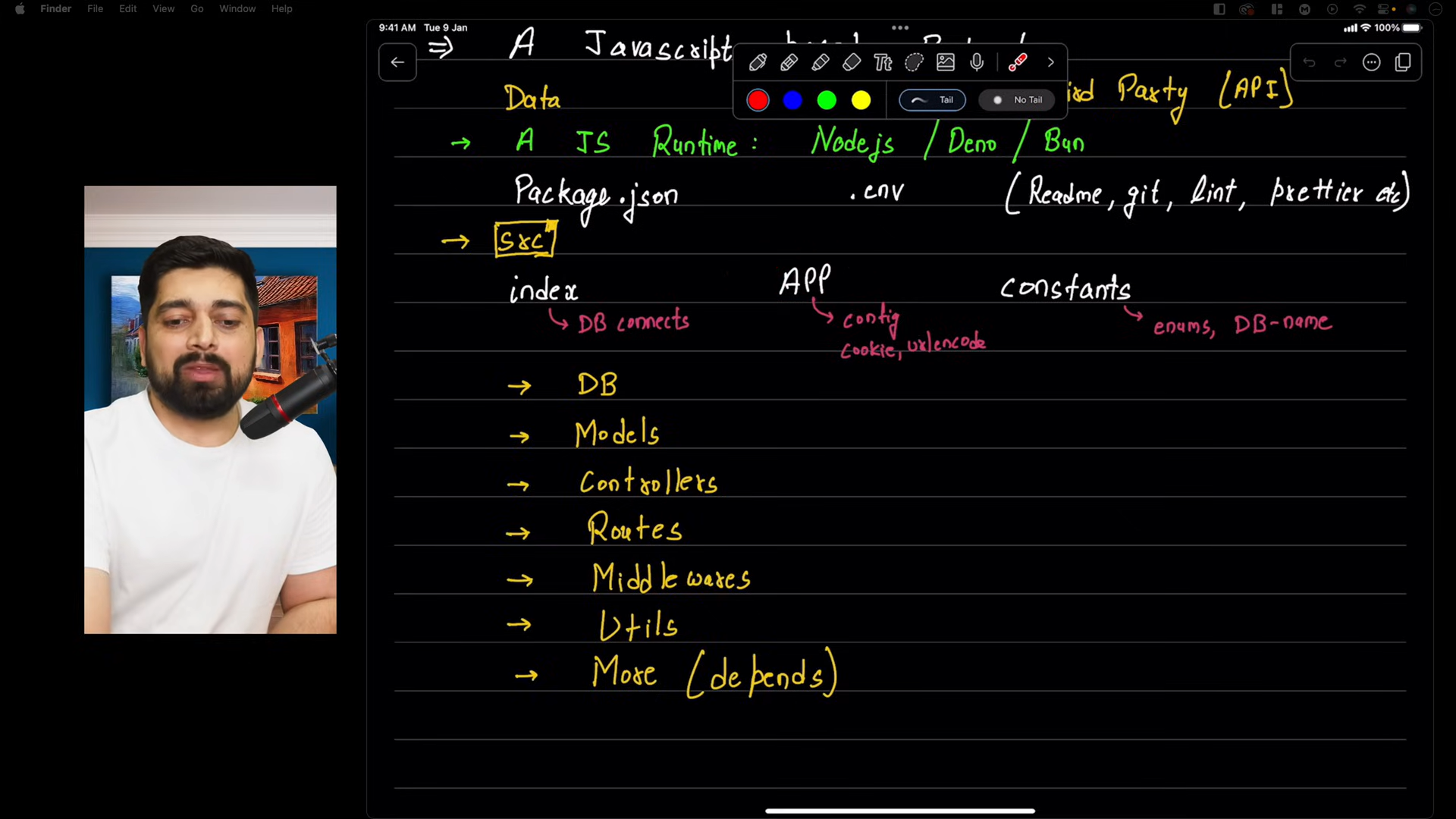
**index –** index file is the entry point of application . We’ll connect db in this file, when app starts

**App –** app file handles configurations(backend cookies, configurations, urlencode)

**constants –** is for storing constant names (enums, DB-name). Suppose we made a airticket booking application, and we have three type of seats aisle, middle, window.

And we’ll want to create enums for these three constants. If we don’t , then people can also select pilot seat

* **DB** **(FOLDER) -** This folder will contain all the code which will connect to the database
* **Models (FOLDER) –** This folder contain data models
* **Controllers (FOLDER) –** This will contain functions like MVC models (C in MVC is controller). These functions will take data and process it.
* **Routes (FOLDER) -** /SIGNUP, /LOGIN, ETC. (all code related to routes & functions which will be called on visiting them)
* **Middlewares (FOLDER) –** will discuss it later
* **Utils (FOLDER) –** utility folder is for utilities. This will handle work like handling functions which will be needed in many places like mailing(which will be needed at many places like in forgot password, congratulate signup, reset password) or file upload (it is also needed in many places in any full-stack app)
* **More(depends) (FOLDER)**



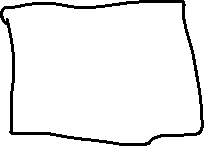
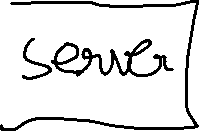
**How to deploy backend code in production**

1. Install nodejs
2. Now, we have to make a server, and if someone send a req from com/mobile(frontend)

And, for sending response to that request we will need two tech or package – express and mongoose.

Firstly, express will handle sending the response from server to comp/mobile for request.

Mongoose will handle database work.



Express

get

**Computer**

**or Mobile**

**Browser**

* Now, suppose if visit a url, a server is listening to the request, which will send something in response.

**SO, there should be something which is listening continuously for requests and express is the package which will handle listening to the requests.**

**/ - Home route**

**/login – login route**

* **There are many type of requests-**

**GET, Deleting data or any file or any entry from database, or many more types of requests.**

* **For now, we will see get request. When we enter anything in url, that is get request**
* **We have to make a server using express for listening to requests.** We have to listen to many routes like /, /login , etc.
* **CODE =>**

1. First, we have to make a empty node application, for that run command

**npm init (means node package manager se 1 application ko initialize krdo)**

This will create a file named **package.json** which contains all the common items like file name,version, desc, scripts, dependencies, etc.

And, we can use **npm install** to install any package.

Steps:

* Package name: firstBackend
* Version
* Desc: a basic app to deploy
* Entry point – index.js
* Test command: not now
* Git repo – later
* Keywords - node chai (for now)
* Author – bharti sahani
* License:

1. Make a file named index.js and write console .log for testing and to run this file, we can use **node index.js**

**In package.json, in scripts we have to add start command –**

**“start” – “node index.js”**

Now, whenever we’ll write npm run start in terminal, node index.js command will run.

1. Now, we have to make backend, and to make backend we will need express.

**npm i express,**  and after installing we can see express in dependencies section in package.json

1. Now, take basic code from express official site –

const express = require('express')

//require module syntax (1st method)

//import express from "express"  //import module syntax (2nd method)

//Now we'll mae a varaible named app using express

const app =  express()

const port = 3000 //virtual ports, server will listen on port 3000

//app will listen on routes eg: / route p listen kro, and send response if you get any request , and to send response it'll be provided with a callback

app.get('/',(req,res)=>{

    res.send('hello express')

})

app.get('/twitter',(req,res)=>{

    res.send('hello twitter')

})

//now app will listen on main port and will give a callback

app.listen(port,()=>{

    console.log(`example app listening on port ${port}`)

})

**HENCE WE’VE MADE A SERVER SUCCESSFULLY, NOW WE WILL DEPLOY IT.**

**BUT BEFORE DEPLOYING WE SHOULD KEEP SOME THINGS IN MIND-**

* We need to be careful with special variables (sensitive info like username, password, db info, etc)
* And, in our computer we are using 3000 port , but it is not necessary that on the other person’s computer 3000 port must be free, or maybe they are forcing their ports on our applications.

So we need a package - **dotenv**

**Npm install dotenv**

* **Now we will make a file .env, and we’ll put our sensitive info variables in it.**

PORT = 3000

* Now, in index.js, write

**require(‘dotenv’).config()** on top of the code

* And, now change port variables in index.js with **process.env.PORT**
* **We’ll push our code to git**

1. git init
2. Now before adding all the files to git, we will push imp and sensitive data to git ignore file
3. Make a file .gitignore and then add node\_modules and .env
4. Now git add .
5. Then, follow all the steps and push the code.

* **Now, got ot your app deploying platform-**

1. Select an app
2. Take code from github
3. Then set plan
4. Then, environment variables need to be set
5. Then follow the steps and build
6. And, after testing destroy the app to save balance/bill

**Connect Backend With Frontend**

1. In js , there are 2 types to import files- common js(basic) and Module js

**SYNTAX FOR BOTH –**

* **Common js (work synchronously)**

**const express = require(‘express’)**

* **Module js (work asynchronously)**

**import express from ‘express’**

**And, in package.json , add “type”:”module”,** now our server will assemble the file acc to module (as modules) and not acc to commonjs

1. res.send -> to send normal data

res.json -> to send json data

1. There are many bundleres like vite, create-react-app, parcel to create a react app and these are called **toolchains**.

And, these help in bundling all the JS files and convert it in form of html, css, jswhich is understandable by js.

1. Now, we’ll start our first basic fullstack project-

Make 2 folders backend and frontend.

In backend , open integrated terminal and make a server as previously done, using npm init , then npm express, then changing script for start in package.json and adding type:module in it.

Now, in frontend folder open separate terminal like backend and create a react app using vite. Then, npm install and write basic code. Now, to get data from backend, there are many ways- Axios, Fetch, ReactQuery,etc.

But, we will use Axios.

Axios is better bcz this library is written specially for web requests, and many more benefits (i.e. production level features). It also handles ,

* How we are receiving data
* How we are handling data
* What will happen if it fails to get data
* What will happen on loading
* To stop a request in between and add some API keys and then get data.

All these features are provided by Axios by default.

To install axios, run command – **npm i axios**

Then – **import axios from ‘axios’**

**And, the data that we get from response, we don’t need to pass or convert into json and stuff like that bcz all these things are handled by axios. We can get directly from response.data**

Also, make repo on github, make gitignore file before committing to github and add node\_modules,.env and any imp file in it and then commit.

Now, on first run we’ll get CORS error

**CORS**

* Means cross origin, (don’t allow outsiders).
* It provides safety to our application.
* If origin is different, then don’t allow the browser(req from outsider) to access our app.
* Only the links which are homely to our app are allowed
* **Origin same means url should be same, port should be same, everything should be same.**
* Eg- our server is working on diff port, and application is working on diff ports, hence we can get infinite no. of requests which can increase our bill/charges.
* Now, we have some solutions to solve this cross origin problem

1. Ask your backend developer to whitelist your url (with port no.). Backend dev can also \* mark to accept requests from any source/url/port.

**IP WHITELIST OR DOMAIN WHITELIST**

And to do this, install cors package from npm

***NOW, WHILE WHITELISTING WE SHOULD KEEP IN MIND, the whitelisting done in production and on localhost both because assume your vite application running on localhost port 5173 but when we will put our app on production it is not necessary that your backend will run on 3000, it can run on any other port too (unlike on local , in which server was running on 3000). And same with frontend, local is on 5173, but it is not necessary that vercel or Netlify will serve on same port, they can serve on diff ports than local port.***

1. **USE PROXY ON FRONTEND and**

**Follow standard practices like-**

**IMPORTANT POINT :**

Don’t write full url on both backend and frontend, write in standarised form like =>

/api/jokes, etc.

Because irrespective of ports and url, it has the endpoint, so it will show the same page because

We have standarised the url of api, but it will give the error url doesn’t exist. **So, we will use PROXY.**

**To add proxy, if we have created our app through create-react-app, then add proxy url in package .json**

**And if we have created our app through vite, then we’ve to add proxy in vite configuration file(vite.config.js), like:**

**Before handling CORS, we are facing this error with this code:**

import { useEffect, useState } from 'react'

import './App.css'

import axios from 'axios'

function App() {

  const [jokes, setJokes] = useState([])

  useEffect(()=>{

    axios.get('http://localhost:3000/jokes')

    .then((response)=>{

      setJokes(response.data)

    })

    .catch((error)=>{

      console.log(error)

    })

  })

  return (

   <div>

    <h1>First Full-Stack App</h1>

    {jokes.map((joke)=>(

      <div key = {joke.id}>

       <h2>{joke.title}</h2>

       <p>{joke.content}</p>

      </div>

    ))}

   </div>

  )

}

export default App

// Access to XMLHttpRequest at 'http://localhost:3000/jokes' from origin 'http://localhost:5173' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

//CORRECTED CODE

useEffect(()=>{

    axios.get('/api/jokes') //standardizing the urls of apis

    .then((response)=>{

      setJokes(response.data)

    })

    .catch((error)=>{

      console.log(error)

    })

  })

import express from 'express'

const app = express();

const port = process.env.PORT || 3000;

app.get('/',(req,res)=>{

    res.send('server is ready')

})

//get a list of 5 jokes

app.get('/jokes',(req,res)=>{

    const jokes = [

        {

            id:1,

            title:'1st joke',

            content:'this is 1st joke'

        },

        {

            id:2,

            title:'2nd joke',

            content:'this is 2nd joke'

        },

        {

            id:3,

            title:'3rd joke',

            content:'this is 3rd joke'

        },

        {

            id:4,

            title:'4th joke',

            content:'this is 4th joke'

        },

        {

            id:5,

            title:'5th joke',

            content:'this is 5th joke'

        }

    ]

    res.send(jokes);

})

app.listen(port,()=>{

    console.log(`server is running on port ${port}`)

})

//CORRECTED CODE

app.get('/api/jokes',(req,res)=>{ //standardizing urls of apis

    const jokes = [

        {

            id:1,

            title:'1st joke',

            content:'this is 1st joke'

        },

        {

            id:2,

            title:'2nd joke',

            content:'this is 2nd joke'

        },

        {

            id:3,

            title:'3rd joke',

            content:'this is 3rd joke'

        },

        {

            id:4,

            title:'4th joke',

            content:'this is 4th joke'

        },

        {

            id:5,

            title:'5th joke',

            content:'this is 5th joke'

        }

    ]

    res.send(jokes);

})

**ADDING PROXY IN VITE.CONFIG.JS =>**

It means whenever someone makes a req from this toolchain, all the requests which will contain /api, it’ll detect automatically and will append <http://localhost:3000> in starting.

Moreover, it will also apply a proxy(means our server will take it as a request coming from same url on which our server is working) that the request is coming from this url only

(ye proxy lg jaegi, mtlb hmare server ko lgega ki jo request ari uska origin yhi url h, jispe apna server bhi chlra h)

And, in production , we will change the vite configuration ki api ab hmare server k url se aari h jha bhi hosting hmne krrkhi h

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/

export default defineConfig({

  server:{

    proxy:{

     '/api': 'http://localhost:3000'

    }

  },

  plugins: [react()],

})

**AFTER CORRECTION OF CODE USING PONTS BELOW, WE’VE handled the cors**

**SUMMARY:**

IN THIS APP, WE ARE DOINF THESE STEPS TO HANDLE CORS ERROR

1) Standarise all the api urls in both backend and frontend

2) Use proxy (server ko ye lgwane k liye ki request same url se aari h, jha hmara backend ka server

chlra h)

**Data Modeling for backend with mongoose**

1. While starting to make a backend, keep these things in mind-

* What data will be stored i.e. (data fields like username, password, photos, dob, email, etc

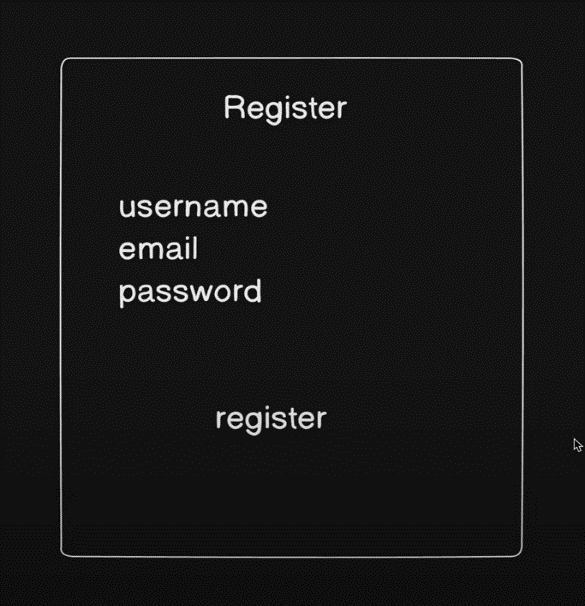
1. Walkthrough for preparation of backend while starting a project –

* Mongoose (important tool for storing data)

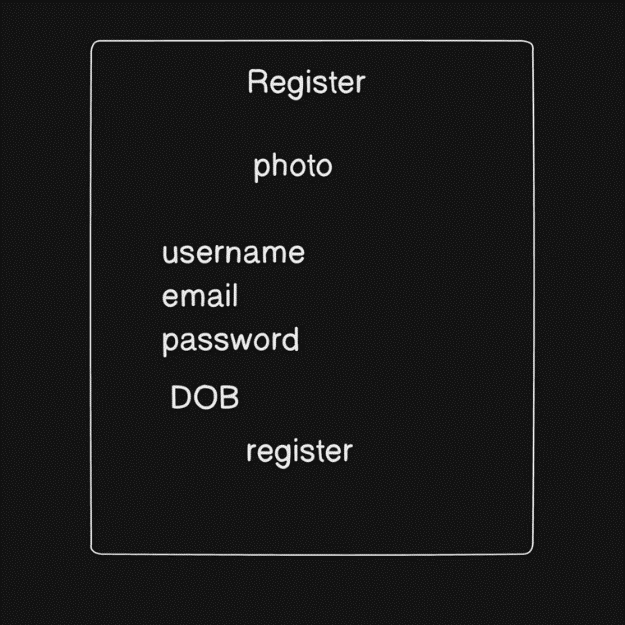
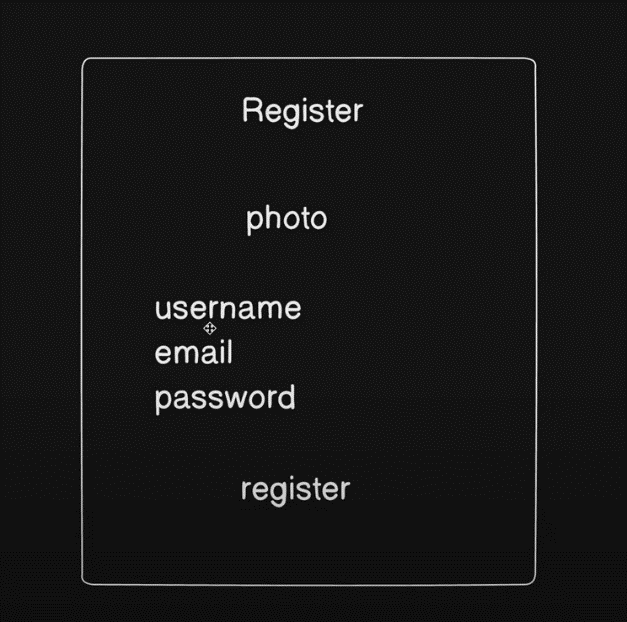
1. What data will be stored and how? This point will be discussed first each time while starting a project and for this we have many tools like moon modeler (expensive), eraser.io
2. DATA MODELING – means to define structure of data
3. In beginning, think about the screen through which data will be stord in database like register form screen ,

Because login form is validation of data from database, but register form is storing data in database.

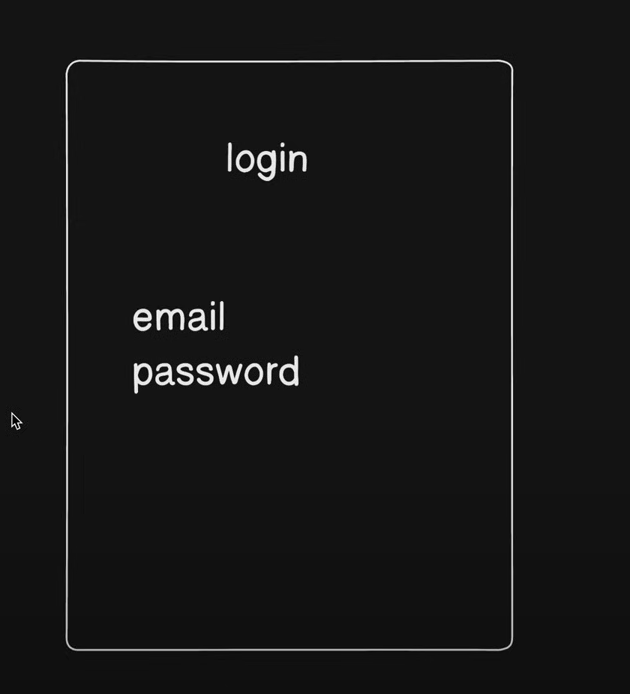
Make a diagram in notebook and note all datapoints. **Note all the fields which are needed.**

****

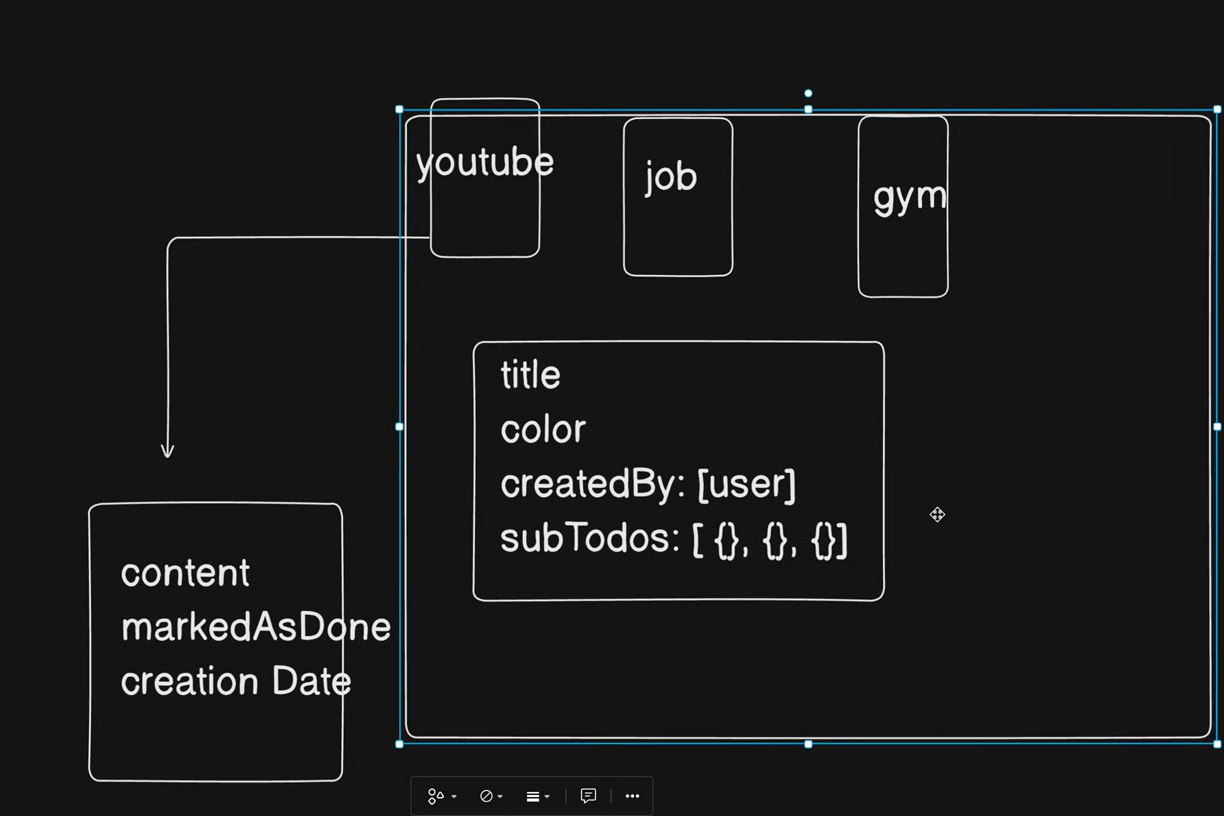




We can put different datapoints in register form like photo , dob, email, password, as we can see in 3 different data modelling diagrams for register screen



And as we know login is data validation because it always asks for data that is already stored in database. **AND, hence storing data points (register form screen) is more important than data validation on existing data. So, we will always handle data modelling of register form screen first.**



Suppose we are making a todo app. And, in our todo we will store todos like youtube work, job related work, gym. Now, these are main todos, and now we have to structure of subtodos and the main todos.

Like in main todos, we will store title of maintodo, color of that todo, created by means user, and list of subTodos as objects in an array.

Now, we will define structure of subtodos in each mainTodo=>

We will store content of subTodo, markedAsDone, created Date.

1. Now, we will practice on online nodejs environment using stackblitz platform.

* Install package - **npm i mongoose**
* Make a folder models in root directory, and a folder named todos in this models folder.
* Now, we want to make an application todos in backend. So, at first it should not be our concern how we will put our data in our application bcz that will be handled by controllers.
* Our first concern should be, how we will store all the data of todos in database. Just like movies which have characters hero villain relatives, etc. In database too, we have characters.
* So, we will make some files for characters, make a file named **user.js or user.models.js** (industry practice),

**2nd file -** todo.js or todo.models.js

**3rd file –** sub\_todo.js or sub\_todo.models.js

* What is a Schema? A schema is a JSON object that defines the structure and contents of your data.
* Mongoose schemas are how you tell Mongoose what your documents look like.
* Mongoose helps us to create schema.
* Mongoose models kse bnata h or export kse krta h, let’s see –
* **import mongoose from “mongoose”**
* **const userSchema = new mongoose.Schema({})**

Here, Schema is a method which takes an object

Now, we have to export it , as it will create this schema in mongodb. As of now, we are not connecting mongodb, because for now we are practice data modeling.

Now, we can’t directly export schema, as it is mongoose’s schema , there is a way to export it.

**We’ll create a model named User using mongoose.model and this mongoose.model is a method which will take 2 parameters –**

**A name for model, and a schema on which our model will be based.**

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

import mongoose from 'mongoose';

const userSchema = new mongoose.Schema({});

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

**IMP NOTE -**  now as we can see we have used capital letter in User, but after going in mongodb, it will be converted into lowercase users.

This is a standard practice of mongodb, then model name will be converted into plural and into lowercase.

User will be converted into users

Todo will be converted into todos

1. Now, we have created the model, now we will put our data fields in our schema, and all the properties for each data field in an object.

* username : String,

**or**

* username : {

type: string,

required: true,

lowercase: true,

unique: true,

and many more properties

}

**We can write data fields using any of above methods, direct or by defining properties of data fields in an object**

