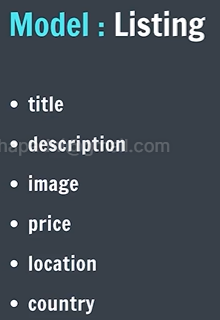
**PHASE - 1 (a)**

In this part we initialized db and established connections, performed CRUD operations on db, created APIs for db and createed model .

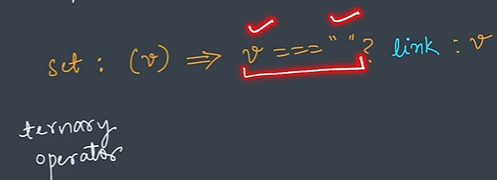


* npm init -y
* npm install express, ejs and moongose
* Use app.listen to listen to port
* Create basic API : get request to create root page
* Create and connect to mongodb using url from mongoose website…use then and catch along with main function to connect to db

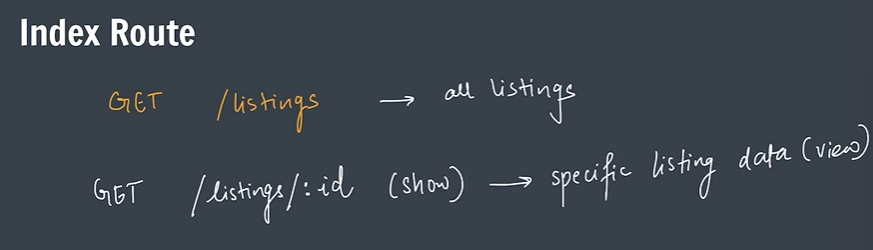




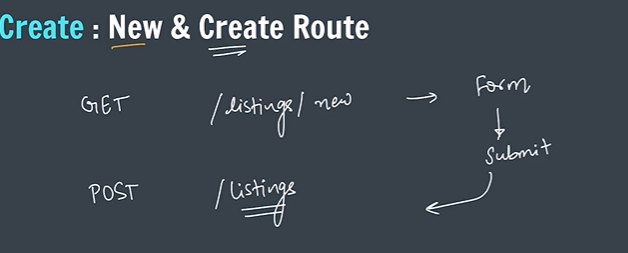
* We create folder models : create file listing.js in this which we will export and use in app.js
* In listing.js : create a schema listing.js and then create model listing with the defined schema
* Set default image for sample database



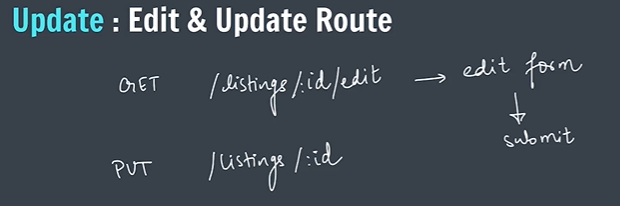
* We have our sample database data.js (can generate using chatgpt or some api) and we export the sample data at end in the form of an object.
* Then we create index.js for initialising db.
* INDEX ROUTE in app.js
* Create Views forlder…in that listing folder…and in that index.ejs



* Under CRUD operations, R is Read for which we will create show route to print data of individual listings.
* We use this : &#8377 for rupee symbol and we use num.toLocaleString(“en-IN”) to separate amount by commas.
* Under CRUD operations, C is create for which we will create two new routes : new route and create route.



* Under CRUD operations, U is for update for which we will create two new routes : edit route and update route.



* Under CRUD operations, D is for delete for which we will create a delete route.



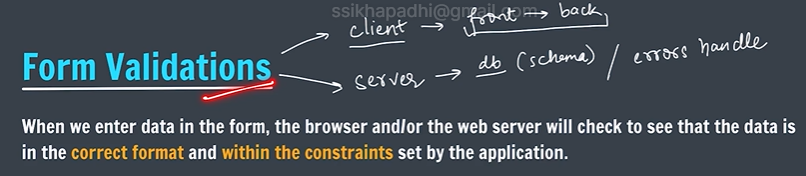
**PHASE - 1 (b)**

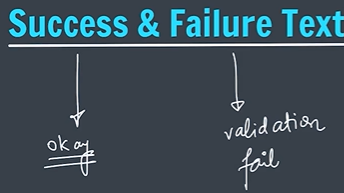
We add styling in this part.

* Here we will use a new package of npm, that is, ejs mate which helps to create multiple templates for web page. We create a boilerplate code and connect all ejs file with it.
* Then we work on nav-bar which is common for all pages so we will create it in boilerplate code.
* Then create the footer file. Its important to maintain modulaity, that is, to divide each part of website into different codes and file so that if we need to modify some part, we can directly modify the file.
* The next we style the index by creating cards of all listings. We will edit the index.ejs file for this.
* Next we style the create from page by editing new.ejs

**PHASE - 1 (c)**

* Validation: Here we will use custom styles for validation by bootstrap.





* Now we define our custom error handler by creating a middleware in app.js
* Next we create custom WrapAsync which is use to write the try-catch block in a better format.
* In next step, we will create custom express error to use wherever we need to handle error.
* Then we create a default error page error.ejs to present the errors in better way.
* Validation for schema: Here we will use a tool joi which is used to validate the schema we defined. Joi is the most powerful schema description language and data validator for JavaScript.