**Major project documentation**

1. app.js: main file. Contains all routes and mongodb connection
2. Create a listing model: listing.js. Type of schema.
3. Initialize Database: data.js.
4. index.js: initialization logic.
5. Index route: index.ejs.
6. Show route: show.ejs
7. Create route: new button pe click karne ke baad ek form milega which you when submit will be a post request. new.ejs.
8. Update route: when you click on edit button, it will be a put request. edit.ejs.
9. Delete route
10. Creating a boilerplate: EJS Mate. Boilerplate.ejs. Include it on every ejs page.
11. Style.css.
12. Create a navbar using bootstrap. Navbar.ejs. Include it in boilerplate code. Same for footer.
13. Create a footer. Footer.ejs.
14. Add styling of both footer and navbar concurrently.
15. Created cards using bootstrap and styled them.
16. New and edit page styling using bootstrap forms. Style show page
17. Form validations: should send data in correct format (client-side validation) and follow db schema(backend). Use bootstrap form validation for every field. For both edit and new pages as well.
18. Custom wrapasync: wrapasync.js. better than try-catch block.
19. Custom express error: Expresserror.js. For all other routes (\*). Used with app.use path
20. Error.ejs: used to show errors to user in a nicer format. Use bootstrap alerts. Used with app.use
21. Schema validations: joi used (npm package) schema.js
22. Create a function validate listing to convert the schema validation in form of middleware.
23. Created review.js in models: 1) set up the reviews form in show ejs.

2) Submitting the form: create a route in app.js

1. Client-side validation for reviews. Validate review same as listing validation. Create a joi in schema.js and then require in app.js and create a function. (validateReview)
2. Showing reviews on show.ejs page. Used populate method in listing.js. Adding styling to reviews. Deleting the reviews. Used mongo $pull operator.
3. Deleting listing: reviews also get deleted along that. Used a post middleware in listing.js
4. Used express router. Created listing.js and review.js. Basically, we shifted the specific routes from app.js to them. We need to require and “use” them in app.js
5. Made cookies.
6. Created Session and used connect-flash. For connect-flash 1) use flash in app.js and define middleware (success and failure) 2) in listing.js flashed a message 3) included in flash.ejs
7. Created flash.ejs. Included in boilerplate. Connect-flash in review.js
8. Used Passport for authentication. Created a User model (username, password, email). Passport local mongoose will add a username and password field by default. That’s why we define only email. Read docs in npm
9. User signup routes. GET and POST. User login. In login use passport authenticate.
10. req.authenticate in routes/listing to only allow authenticated users to create listing. Created middleware.js and wrote req. authenticate middleware (isLoggedIn) in it. Used in listing new route.
11. Created Logout method using passport in user.js route. Edit navbar to add sign up, login and logout. If logged in then no need to show signup and login page only logout page is necessary.
12. Automatically login after signup. No need to enter information twice. Used passport login method in user signup route. `save redirect url` function in middlewares.
13. Associating an owner to each listing. Added owner in index.js using map. Edited show.ejs as well.
14. Created a middleware isOwner. Shifted validate listing in middleware file.
15. Setting authorization for reviews just like we did for listings. Modified review model and show.ejs file. Used nested populate to show reviews with their respective author. Created isReviewAuthor middleware (same as isOwner middleware).
16. Created controller for listing. Basically, we shifted the routes from models listing to controllers listing. Nothing else. Basic copy pastes everything after wrapsync or where req, res starts. Same for reviews and user.
17. Router. Basically, combine the paths with have some root common. In listing file in route. Same for user.
18. Restyle ratings. Created rating.css. Edited show ejs file.
19. Image upload.
20. Added enctype in new ejs file. Installed multer and used in listing routes.
21. Cloud setup. Cloudinary. .env file: to store environment variables. Used npm package. env to access .env file contents in app.js. created cloudConfig.js file.
22. Flow is we can send form’s file to backend which parses and sends to Cloudinary and we get a link of our saved file.
23. Image edit in edit form. We should be able to upload ne image instead of seeing new image. Handling image in backend. Image preview for edit page (blurred the image so that few pixels are required for preview in listings controller).

Script.js is made for the logic of “needs validation” class of bootstrap

Protect routes by adding middlewares after their routes.

Starabiltiy in LunarLogic github library for rating.

https://console.cloudinary.com/app/c-79b30bdb4daac3be5180e99d626a27/assets/media\_library/folders/cbede4dec6894b1832ba9dc9396dc8c6bb?view\_mode=mosaic