Project stack

1. Typescript
2. Typeorm
3. Express with authentication using sessions and cors
4. Postgres
5. NextJS

Following the Clean architecture.

Steps:

1. Setup folder structure

Following the clean architecture shown below:



Link to lecture: <https://www.youtube.com/watch?v=NeXQEJNWO5w&ab_channel=StreamAConStreamingConferences>

Summary is:

* entities are business rules they are the core and they have no dependency
* use cases are application business rules, they use the entities to do things,
* the controllers adapt the frameworks and abstract them from out use cases
* finally, the frameworks can be plugged in the final layer

The purpose of this architecture is to abstract layers from each other and be able to expand on the project with minimal damage.

Based on that, the folder structure I used is as follows:

We have two main folders:

* backend: contains the whole system and is structured on clean architecture
  + dist: contains the compiled JS code from TS🡪 deployable code
  + src:
    - server file:
      * establish connection with database
      * Listen on port: port will be an env variable
      * Routes requests to different routers
      * Contains main middleware: sessions, cors...etc
      * Depends on: express, dotenv, typeorm
    - Routers:
      * Each route has a router handler: user, files…etc
      * Router handlers use the express framework to handle routes
      * They take in the request object, use a helper to adapt it (abstract from express) and then pass it down to a controller
      * The controller will return a response object that the router can decipher and send back
      * Returns response to the browser
    - controllers:
      * abstracts the use cases layer from the adaptors layer
      * it is called by the routers and is passed an abstract http request object
      * it calls a use case and then takes its return and formulates an abstract http response object
      * basically, connects the http sequence to the business use case
    - use cases:
      * abstracts the entity management from the controllers
      * called by the controller and passed in the abstracted data from the body/method/params/queries…etc.
      * it creates an entity ready to use by using the entities folder and passing it the information from the req object
      * it connects to the DB and does the CRUD operation necessary for the use case.
      * Returns confirmation on the use case such as res like object or an error object
      * They are organized by type: user, product…etc.
    - Entities:
      * Handles the logic for the entity itself such as validation, formulation of data, and getters/setters
      * Exposes an object that represents an entity in the DB and can only be interacted with by other modules through this object
      * While the connection is alive the object should not unexpectedly change state (values) due to this arrangement
    - DB\_manager:
      * Basically the DB controller
      * abstract data definition and migrations
      * can be switched entirely with no effect on the system, it contains all typeorm code
    - Helpers: reusable functions that can be re-used and plugged in the rest of the code
    - Middleware: contains all middleware functions re-usable in multiple places, it is different from helpers because they are only specifically used as middleware and not in the logic
  + Types: stores all the local types created for use in this project such as interfaces and enums
  + Ormconfig: contains the configuration for typeorm
  + .env: contains the environmental variables for the project not to be exposed
* front end: is a web framework so it’s decoupled from the system as it can be changed later easily. It’s the outer layer and is done in Next JS for now

1. Setup environment
   1. Download & Install node/npm
   2. Install typescript globally: *npm I -g typescript*
   3. Install typeorm globally: *npm I -g typeorm* 
      1. You need to install reflect-metadata as well
   4. Download and install postgres
      1. Create a new user and DB: use pgadmin or the cli
   5. Install ts-node to run typeorm scripts
   6. Install Next js with typescript: there is a flag for ts
   7. Install dev dependencies:
      1. @types/express
      2. @types/connect-redis
      3. @types/cors
      4. @types/express-session
      5. @types/nodemailer
      6. @types/node
      7. Typescript
      8. Tslint
      9. Tslib
      10. Nodemon
   8. Dependencies:
2. "dependencies": {
3. "bcrypt": "^5.0.1",
4. "connect-redis": "^6.0.0",
5. "cors": "^2.8.5",
6. "dotenv": "^10.0.0",
7. "express": "^4.17.1",
8. "express-session": "^1.17.2",
9. "nodemailer": "^6.6.5",
10. "nodemon": "^2.0.12",
11. "pg": "^8.7.1",
12. "redis": "^3.1.2",
13. "reflect-metadata": "^0.1.13",
14. "typeorm": "^0.2.37"
15. },
16. Add scripts: main ones are:
    * 1. Build typescript: “tsc” or watch “tsc-w”
      2. Dev env: “nodemon dist/server.js” or whatever you called the entry point
      3. Typeorm gen migrations: "node --require ts-node/register ./node\_modules/typeorm/cli.js migration:generate -n migrationname -p"
      4. Build and run frontend:
      5. Production script:
17. Set up configurations:
    1. Orm config: should be in the project root, use JSON format, in it specify the db, in this case postgres (install pg)
       1. The entities & migrations should point to the dist folder and a JS file: "./dist/src/DB\_Manager/entity/\*\*/\*.js"
       2. Set synchronize to false it’s not good for production
    2. Tsconfig: you can use the strict rules added in the file for higher quality code
18. Set Up server:
    1. Listen on port
    2. Establish connection
    3. Set up router
    4. Set up middleware