**Table of Content**

Table of Contents

[Typescript 2](#_Toc145191760)

[TypeScript দিয়ে রিয়েল লাইফ প্রজেক্ট করার প্রসেস সমুহ- 4](#_Toc145191761)

[Create A new Project From Scratch 5](#_Toc145191762)

[প্রজেক্ট সেটাপ করার জন্য প্রয়োজনীয় কমান্ড (01) Scratch to Advanced (Important) 9](#_Toc145191763)

[এই ওয়েবসাইট ফলো করা যেতে পারে ESLint এর জন্য- 14](#_Toc145191764)

[মেইন খেলা শুরু এখান থেকে 20](#_Toc145191765)

[12.4 (Setup Basic Logger) 22](#_Toc145191766)

[12.5 (Separate Success logs) 22](#_Toc145191767)

[12.6 (Beatify and Customize the log) 23](#_Toc145191768)

[12.7 (Manage Logs using a library) Final 24](#_Toc145191769)

[12.8 (Error Handling) 25](#_Toc145191770)

[Zod Installation 27](#_Toc145191771)

[Authentication of Project 33](#_Toc145191772)

[TYPESCRIPT JWT 34](#_Toc145191773)

[React দিয়ে প্রজেক্ট তৈরি করার নিয়ম (লেটেষ্ট) 35](#_Toc145191774)

[React Router Install করার টেকনিক- 35](#_Toc145191775)

[Redux Toolkit install 36](#_Toc145191776)

[CRUD operation By Redux Toolkit 38](#_Toc145191777)

[Next JS 41](#_Toc145191778)

[MySQL (POSTgre) 44](#_Toc145191779)

[JavaScript দিয়ে Backend 45](#_Toc145191780)

[( Javascript Error Handling) 51](#_Toc145191781)

[JavaScript Authentication of Project 52](#_Toc145191782)

[JAVASCIRPT JWT 53](#_Toc145191783)

### LEVEL-2 PENDINGS

1. module-06 , module-07 and module-08
2. 14.5 পর্যন্ত শেষ করা হয়েছে। ১৪.৬,১৫, থেকে শুরু হবে। পরবর্তীতে।

### Typescript

1. ভেরিয়েবল কী হবে সেইটার টাইপ যদি লিখতে হয় তাহলে ছোট হাতের লিখতে হয় যেমন- string, number, boolean etc.
2. যদি Schema তৈরি করতে চাই তাহলে হবে বড় হাতের String, Number, Boolean
3. আমি যখন ফাংশন কল করব তখন যে প্যারামিটার গুলো নিব সেগুলো কী টাইপের সেগুলো বলে দিব এবং ফাংশনটি কী রিটার্ন করবে সেইটাও বলে দিব। যেমন

const addTwo = (a: number, b: number): number => {

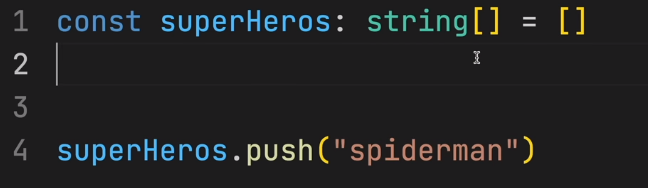
  return a + b;

};

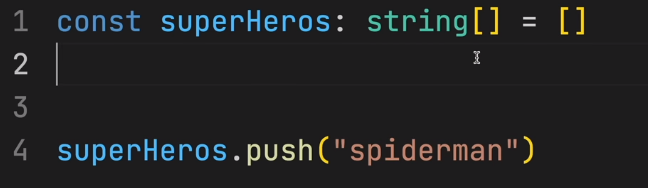
console.log(addTwo(4, 5));

export {};

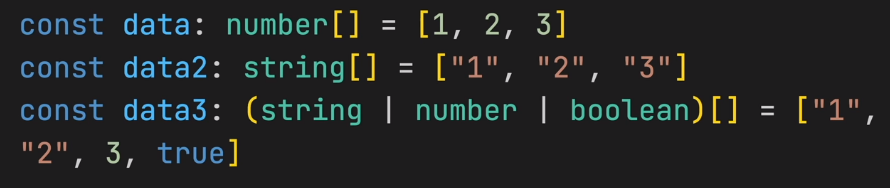
1. যখন দেখা যাবে একটি ফাংশন আমাকে দুই থেকে তিনটা প্রিমিটিভ টাইপ রিটার্ন করতেছে তাহলে আমি রিটার্ন টাইপ বলে দিতে পারব না
2. অ্যারের ক্ষেত্রে অ্যারে কী ধরনের হবে সেইটা বলে দিতে হবে। যেমন



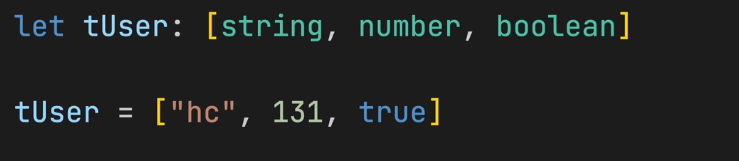
1. অ্যারের মধ্যে কোন অবজেক্ট থাকে তাহলে সেটিকে টাইপ দিয়ে কনভার্ট করে অ্যারে টাইপ ডিক্লেয়ার করে দিতে হবে। যেমন-



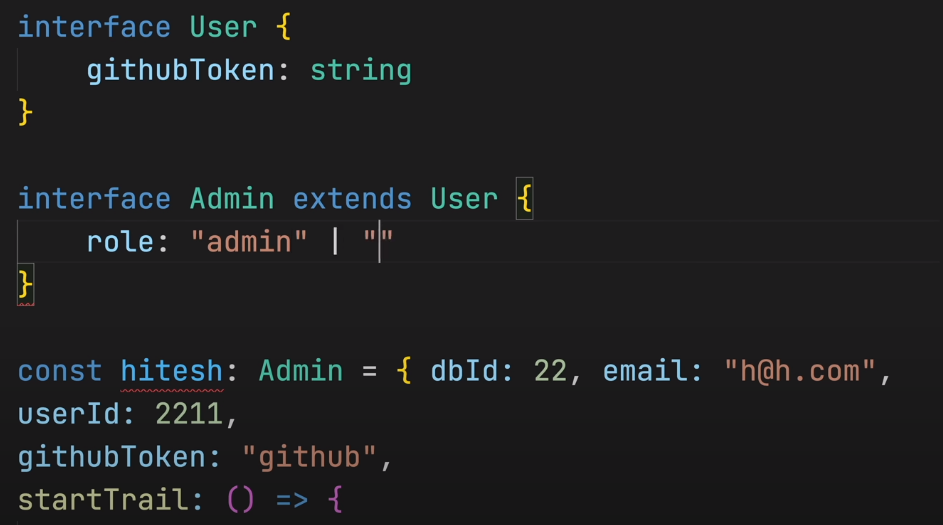
1. মাঝে মাঝে অ্যারের মধ্যে বিভিন্ন ডাটা টাইপ আসতে পারে যেমন



1. tupple এর ক্ষেত্রে নিচের নিয়ম অনুসরণ হবে। কারণ এইটা টাইপগুলো যে সিকুয়েন্স এ থাকবে সেই সিকুয়েন্সেই লিখতে হবে-

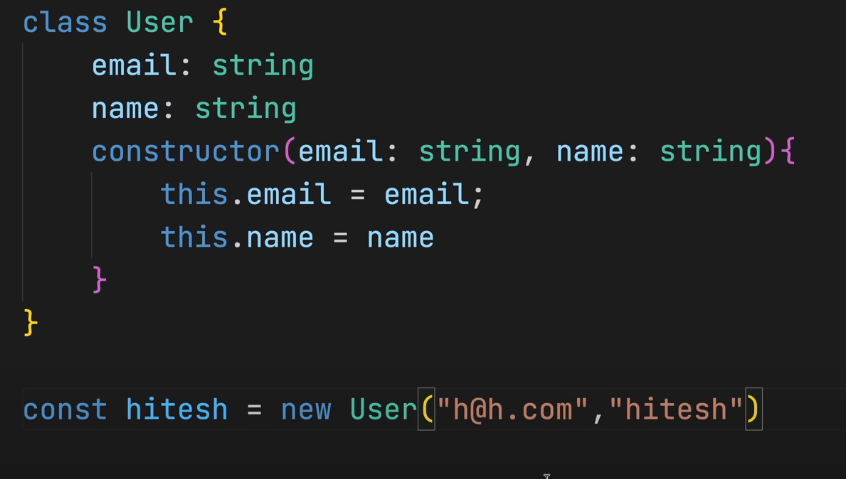


1. interface এ আমাদের সুবিধা হলো যে এইটাকে আমরা extends করতে পারি।

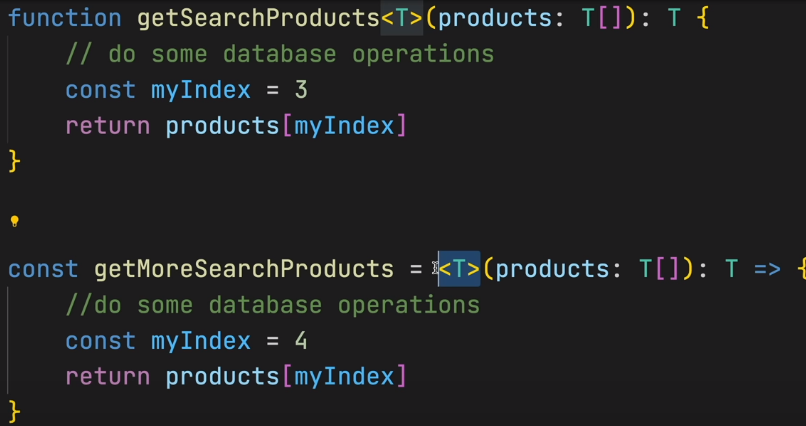


TypeScript দিয়ে রিয়েল লাইফ প্রজেক্ট করার প্রসেস সমুহ-

1. প্রথমে কমান্ড লাইন থেকে tsc –init এই কমান্ড দিতে হবে।
2. npm init –y লিখলে নোড দিয়ে একটা প্যাকেজ ম্যানেজার ইনষ্টল হবে।
3. তারপত src এবং dist নামক দুইটা ফোল্ডার বানাতে হবে ।
4. tscofig.json ফাইলে গিয়ে খুজতে হবে outDir নামক অপশান সেইটাকে আনকমেন্ট করে “./dist” এবং rootDir এর ভেতরে “./src”
5. tsc –w কমান্ড দিলে এটি লাইভ আপডেট দিবে (অপশনাল)
6. class এর ক্ষেত্রে আমরা যদি কোন প্যারামিটার নেই তাহলে সেইটা constructor এর মধ্যে থেকে নেই।



1. Generic function এ নিচের নিয়মে লিখতে হয়। এখানে T হলো যেকোন টাইপের অ্যারের ইনপুট নিতে পারবে-



### Create A new Project From Scratch

কোন প্রজেক্ট করতে গেলে আমাকে প্রথমে Requirement Analysis করতে হবে। এর ধাপগুলো হবে-

1. Functional Requirement
2. Model
3. Api End Points
4. ER Diagram

Authentication service

We are building a university management system where we have three types of roles such as Admin , Student , Faculty.

**Functional Requirements:**

* **Student**

* Student can login and log out
* Student can manage and update their profile.
* Student can update certain fields
* Student can enroll in a semester.
* Student can enroll in offered courses for a specific semester.
* Student can pay their tuition fees through offline or online.(Partial / Full Payment)
* Student can see their transaction histories.
* Student can see their class routines.
* Student can see their otice board and events
* Student can see their result (Full / Semester Wise).
* Student can evaluate their teachers.

* **Admin:**

* Admin can log in and log out
* Admin can manage and update their profile.
* Admin can only update certain fields.
* Admin can manage user accounts
  + Block/Unblock
  + Change Password
  + Forcefully Log out
* Admin can manage multiple process:
  + Semester
  + Offered Courses
  + Section
  + Faculty
  + Student
  + Building
  + Room
  + Payment
  + Permissions
  + Activity

* **Faculty:**

* Faculty can log in and log out
* Faculty can manage and update their profile.
* Faculty can only update certain fields.
* Faculty can manage user accounts
* Faculty can manage student grades
* Access to Academic and Personal Information.
* Faculty can manage their lecture resources

**Models**

**Permission:**

* title

**UserPermission:**

* permissionId
* userId

**User:**

* Id
* **role**
* Password
* createdAt
* updatedAt
* studentId || adminId || facultyId

**Student:**

* id
* name
  + fisrtName
  + middleName
  + lastName
* gender
* dateOfBirth
* guardian
* contactNo
* emergencyContactNo
* email
* presentAddress
* permanentAddress
* department
* Subject

**Admin:**

* Id
* name
  + fisrtName
  + middleName
  + lastName
* gender
* dateOfbirth
* email
* contactNo
* emergencyContactNo
* department
* designation

**Faculty:**

* Id
* name
  + firstName
  + midleName
  + lastName
* gender
* dateOfBirth
* Email
* contactNo
* emergencyContactNo
* department
* faculty
* designation

API END POINTS:

* **User**:
  + users/create-student (POST)
  + users/create-faculty (POST)
  + users/create-admin (POST)
  + users/my-profile (GET)
  + users/:id (GET)
  + users/:id (PATCH)
  + user/:id (DELETE)
  + users/:id/force-logged-out
  + users?page=1&limit=10 (GET)
  + users/:id/available-permissions?page=1&limit=10 (GET)
  + users/:id/assigned-permissions?page=1&limit=10 (GET)
  + users/:id/assign-permissions (POST)
  + users/:id/remove-permissions (POST)

* **Student:**
  + students?page=1&limit=10 (GET)
  + students/:id  (GET)
  + students/:id    (PATCH)

* **Faculty**:
  + faculties?page=1&limit=10 (GET)
  + faculties/:id  (GET)
  + faculties/:id    (PATCH)

* **Admin**:
  + admins?page=1&limit=10 (GET)
  + admins/:id  (GET)
  + admin/:id    (PATCH)

* **Permission**
  + permissions?page=1&limit=10 (GET)
  + permissions (POST)
  + permissions/:id (GET)
  + permissions/:id (PATCH)
  + permissions/:id (DELETE)

* **Auth**
  + auth/login (POST)
  + auth/refresh-token (POST)
  + auth/change-password (POST)
  + auth/forgot-password(POST)
  + auth/reset-password (POST)

----------------------------------------------------------

ER-Diagram

<https://apollo-level2-web-dev.github.io/university-management-auth-service-resources/>

### প্রজেক্ট সেটাপ করার জন্য প্রয়োজনীয় কমান্ড (01) Scratch to Advanced (Important)

1. প্রথমে কমান্ড লাইন থেকে tsc –init এই কমান্ড দিতে হবে।
2. npm init –y লিখলে নোড দিয়ে একটা প্যাকেজ ম্যানেজার ইনষ্টল হবে।
3. তারপত src এবং dist নামক দুইটা ফোল্ডার বানাতে হবে ।
4. tscofig.json ফাইলে গিয়ে খুজতে হবে outDir নামক অপশান সেইটাকে আনকমেন্ট করে “./dist” এবং rootDir এর ভেতরে “./src”
5. এরপর root folder এর মধ্যেই .gitignore ফাইল তৈরি করে এর ভেতরে node\_modules and .env লিখতে হবে।
6. tsc –w কমান্ড দিলে এটি লাইভ আপডেট দিবে (অপশনাল)
7. TypeScript- npm i typescript
8. express- npm i express –save
9. mongoose npm i mongoose –save
10. cors- npm i @types/cors
11. dotenv- npm i dotenv
12. ts-node-dev- npm i ts-node-dev এই কমান্ড package.json এ লিখতে হবে। এইখানে দেখতে হবে – server.ts কোন জায়গায় আছে।

"dev":"ts-node-dev --respawn --transpile-only src/server.ts"

app.ts

import cors from 'cors'

import express, { Application } from 'express'

const app: Application = express()

app.use(cors())

//parser

app.use(express.json())

app.use(express.urlencoded({ extended: true }))

app.use('/api/v1/', routes)

// //Testing

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

//   res.send('Hello From this project')

// })

export default app

app.ts ফাইলে route গুলো লিখার ফলে ফাইলটি পলিউট হয়ে যাওয়ার সম্ভাবনা বেশি থাকায় আমরা Routes গুলোকে আলাদা ফাইলে রাখব। src/app/routes নামে একটি ফোল্ডার বানিয়ে index.ts এ রাউটস গুলো রাখতে পারি।

import express from 'express'

import { BookRoutes } from '../../modules/books/books.routes'

import { userRoutes } from '../../modules/users/users.route'

const router = express.Router()

const moduleRoutes = [

  {

    path: '/users',

    route: userRoutes,

  },

  {

    path: '/books',

    route: BookRoutes,

  },

]

moduleRoutes.forEach(route => router.use(route.path, route.route))

export default router

index.ts এর মধ্যে Environment এর যে ভেরিয়েবল গুলো ছিল সেগুলো এক্সপোর্ট করে দিতে হবে-

import dotenv from 'dotenv'

import path from 'path'

dotenv.config({ path: path.join(process.cwd(), '.env') })

export default {

  env: process.env.NODE\_ENV,

  port: process.env.PORT,

  database\_url: process.env.DATABASE\_URL,

  default\_user\_pass: process.env.DEFAULT\_USER\_PASS,

  bycrpt\_salt\_rounds: process.env.BCRYPT\_SALT\_ROUNDS,

}

একটি ফিচার তৈরি করতে হলে –

interface --> Schema & Model 🡪services-> controller🡪route

src/module/featuresnameFolder

name.interface.ts হবে এইরকম-

import { Model } from 'mongoose'

export type IUser = {

  id: string

  role: string

  password: string

}

export type UserModel = Model<IUser>

name.model.ts হবে এইরকম-

import { Schema, model } from 'mongoose'

import { IUser, UserModel } from './users.interface'

const userSchema = new Schema<IUser>(

  {

    id: {

      type: String,

      required: true,

      unique: true,

    },

    role: {

      type: String,

      required: true,

    },

    password: {

      type: String,

      required: true,

    },

  },

  {

    timestamps: true,

  }

)

export const User = model<IUser, UserModel>('User', userSchema)

name.route.ts

import express from 'express'

import { UserController } from './users.controller'

const router = express.Router()

router.post('/create-user', UserController.createUser)

export const UserRoutes = router

name.services.ts (এইখানে কুয়েরী অথবা বিজনেস লজিক গুলা হবে)

import bcrypt from 'bcrypt'

import config from '../../../config/index'

import ApiError from '../../../errors/ApiError'

import { IUser } from './users.interface'

import { User } from './users.model'

import { generateUserId } from './users.utils'

const createUser = async (user: IUser): Promise<IUser | null> => {

  const id = await generateUserId()

  user.id = id

  if (!user.password) {

    user.password = config.default\_user\_pass as string

  }

  user.password = await bcrypt.hash(

    user.password,

    Number(config.bycrpt\_salt\_rounds)

  )

  const createdUser = await User.create(user)

  if (!createdUser) {

    //createUser

    throw new ApiError(400, 'failed to create user')

  }

  return createdUser

}

export const usersService = {

  createUser,

}

name.controller.ts

import { RequestHandler } from 'express'

import { usersService } from './users.service'

const createUser: RequestHandler = async (req, res, next) => {

  try {

    const { user } = req.body

    const result = await usersService.createUser(user)

    res.status(200).json({

      success: true,

      message: 'user created successfully!',

      data: result,

    })

  } catch (err) {

    next(err)

  }

}

export const UserController = {

  createUser,

}

Env ফাইলের মধ্যে

NODE\_ENV=development

PORT=5000

DATABASE\_URL=mongodb+srv://university-management:ZmLsSJPIyPz2vAsc@cluster0.x5fc9ya.mongodb.net/university-management?retryWrites=true&w=majority

তারপর টাইপ গুলো ডিফাইন করে দিতে হবে-

Request,Response এইগুলা

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

  res.send('Hello World!')

})

Requset এবং Response গুলো import করতে হবে –

import express, { Application, Request, Response } from 'express';

এইখান থেকে ।

### এই ওয়েবসাইট ফলো করা যেতে পারে ESLint এর জন্য-

<https://blog.logrocket.com/linting-typescript-eslint-prettier/>

"include": ["src"], // which files to compile

"exclude": ["node\_modules"], // which files to skip

এই দুটি জিনিষ tsconfig.json ফাইলে পেষ্ট করতে হবে।

পরবর্তীতে

npm install eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin --save-dev

এইটা ইনষ্টল করতে হবে।

রুট ফাইলে .eslintrc নামক ফাইল তৈরি করে তার মধ্য নিচের কোড পেষ্ট করতে হবে-

{

  "parser": "@typescript-eslint/parser",

  "parserOptions": {

    "ecmaVersion": 12,

    "sourceType": "module"

  },

  "plugins": ["@typescript-eslint"],

  "extends": [

    "eslint:recommended",

    "plugin:@typescript-eslint/recommended",

    "prettier"

  ],

  "rules": {

    "no-unused-vars": "error",

    "prefer-const": "error",

    "no-unused-expressions": "error",

    "no-undef": "error",

    "no-console": "off",

    "@typescript-eslint/consistent-type-definitions": ["error", "type"],

    "@typescript-eslint/explicit-module-boundary-types": "off"

  },

  "env": {

    "browser": true,

    "es2021": true,

    "node":true

  },

  "globals": {

    "process":"readonly"

  }

}

Package.json এ Scripts এর মধ্যে

// package.json

{

// ...

"scripts": {

"lint:check": "eslint --ignore-path .eslintignore --ext .js,.ts .",

"lint:fix":"exlint --fix",

"prettier:check": "prettier --ignore-path .gitignore --write \"\*\*/\*.+(js|ts|json)\"",

},

// ...

}

এই লাইন পেষ্ট করে দিতে হবে।

You’ll find that some files don’t need to be linted at all, such as your dist folder, so you can prevent linting by creating a .eslintignore file and adding the folders or files you want to ignore:

node\_modules

dist

তারপর

npm install --save-dev prettier

// .prettierrc

{

"semi": false,

"singleQuote": true,

"arrowParens": "avoid",

}

**Next, we are going to start formatting our code using Prettier in the command line:**

npx prettier --write src/app.ts

তারপর settings.json ফাইলে গিয়ে এইটা পেষ্ট করতে হবে-

// settings.json

 "[typescript]":{

        "editor.defaultFormatter": "esbenp.prettier-vscode",

        "editor.formatOnSave": true,

    },

npm install --save-dev eslint-config-prettier

.eslintrc চেঞ্জ করতে হবে যা আমরা ইতিমধ্যে কপি পেষ্ট করেছি।

আমরা প্রজেক্টের lint এর কোন প্রব্লেম থাকলে আমরা টার্মিনালে রান করতে পারি-

npm run lint:check

এরপর husky ইন্সটল করতে হবে-

npm install husky --save-dev

npx husky install

(এইখানে মনে রাখতে হবে গিটহাব কে ইনিশিয়ালাইজ করে রাখতে হবে) এমনে এমনে ইনষ্টল হবে না।

npx husky add .husky/pre-commit "npm test"

এইটা ইন্সটল করার পর আমরা দেখব এখানে pre-commit নামের একটা ফাইল আছে সেখানে npm test এর জায়গায় npx lint-staged লিখব ।

যদি প্রোগ্রাম রান না করে তাহলে – নিচের কোড টি দেখতে হবে-

#!/usr/bin/env sh

. "$(dirname -- "$0")/\_/husky.sh"

npx lint-staged

***১১.৭***

{ NOTE

ES-Lint প্রজেক্টের লিন্ট টেষ্ট করে ( কোন টা আনডিফাইন বা আনইউজড অবস্থায় আছে কী না সেইটা দেখে)

Prettier প্রজেক্টের কোডগুলো ফরম্যাট করে।

Husky গিটে পুশ করার আগে ES-Lint এবং Format check করে }

Package.JSON ফাইলে গিয়ে

"lint-prettier" :"npm lint:check && npm prettier:check",

লিখতে হবে

npm install –-save-dev lint-staged

এরপর আবার package.json ফাইলে গিয়ে লিখতে হবে-

"lint-stage":{

    "src/\*\*/\*.ts":"npm lint-prettier"

  },

যদি কোড রান হতে সমস্যা হয় তাহলে package.json ফাইল চেক দিতে হবে । দেখতে হবে নিচের মতো হয়েছে কী না?

{

  "name": "university-management-auth-service",

  "version": "1.0.0",

  "description": "",

  "main": "src/server.ts",

  "scripts": {

    "start": "ts-node-dev --respawn --transpile-only src/server.ts",

    "lint:check": "eslint --ignore-path .eslintignore --ext .js,.ts .",

    "lint:fix": "eslint . --fix",

    "prettier:check": "prettier --ignore-path .gitignore --write \"\*\*/\*.+(js|ts|json)\"",

    "prettier:fix": "prettier --write .",

    "lint-prettier": "yarn lint:check && yarn prettier:check",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "Programming Hero",

  "license": "ISC",

  "lint-staged": {

    "src/\*\*/\*.ts": "yarn lint-prettier"

  },

  "devDependencies": {

    "@types/cors": "^2.8.13",

    "@types/express": "^4.17.17",

    "eslint-config-prettier": "^8.8.0",

    "husky": "^8.0.3",

    "lint-staged": "^13.2.2",

    "prettier": "^2.8.8",

    "ts-node-dev": "^2.0.0",

    "typescript": "^5.0.4"

  },

  "dependencies": {

    "@typescript-eslint/eslint-plugin": "^5.59.7",

    "@typescript-eslint/parser": "^5.59.7",

    "cors": "^2.8.5",

    "dotenv": "^16.0.3",

    "eslint": "^8.41.0",

    "express": "^4.18.2",

    "mongoose": "^7.2.1"

  }

}

***১১.৮***

### মেইন খেলা শুরু এখান থেকে

**Create User Interface and Model:**

src🡪app🡪modules🡪users🡪user.interface.ts একটা ফাইল ক্রিয়েট করব।

export type IUser ={

    id:string,

    role:string,

    password:string

}

তারপর users.model.ts ফাইল ক্রিয়েট করব

import { Model, Schema, model } from 'mongoose'

import { IUser } from './users.interface'

type UserModel = Model<IUser, object>

const userSchema = new Schema<IUser>(

  {

    id: {

      type: String,

      required: true,

      unique: true,

    },

    role: {

      type: String,

      required: true,

    },

    password: {

      type: String,

      required: true,

    },

  },

  {

    timestamps: true,

  }

)

export const User = model<IUser, UserModel>('User', userSchema)

তারপর users.service.ts ফাইল ক্রিয়েট করব-

Default password তৈরি করতে হবে এইজন্য .env ফাইলে DEFAULT\_PASSWORD = DFDSF নিলাম। এবং এটিকে config/index.ts ফাইলে export default এর মধ্যে

default\_user\_pass: process.env.DEFAULT\_USER\_PASS,

এটিকে লিখব।

config/index.ts ফাইলে নিচের কোড লিখব – (utilities function গুলো লিখে রাখব)

import { User } from './users.model'

export const findLastUserId = async () => {

  const lastUser = await User.findOne({}, { id: 1, \_id: 0 })

    .sort({

      createdAt: -1,

    })

    .lean()

  return lastUser?.id

}

export const generateUserId = async () => {

  const currentId = (await findLastUserId()) || (0).toString().padStart(5, '0')

  const incrementId = (parseInt(currentId) + 1).toString().padStart(5, '0')

  return incrementId

}

**১২.১**

কন্সোল লগ যাতে প্রিন্ট না হয় এইজন্য eslintrc ফাইলে গিয়ে

“no-console”: “error”;

“no-undef”:”error”; এইটা অলরেডি কপি পেষ্ট করা আছে।

পরবর্তীতে vs-code এ গিয়ে setting.json ফাইলে –

"editor.codeActionsOnSave": {

    "source.fixAll.eslint": true,

    "source.organizeImports": true

  }

### 12.4 (Setup Basic Logger)

Winston ইন্সটল করতে হবে- npm I Winston

src/shared/logger.ts ফাইলে কপি পেষ্ট করতে হবে-

import winston from 'winston'

const logger = winston.createLogger({

  level: 'info',

  format: winston.format.json(),

  transports: [

    //

    // - Write all logs with importance level of `error` or less to `error.log`

    // - Write all logs with importance level of `info` or less to `combined.log`

    //

    new winston.transports.Console(),

    new winston.transports.File({ filename: 'error.log', level: 'error' }),

    new winston.transports.File({ filename: 'combined.log' }),

  ],

})

export default logger

তারপর app.ts ফাইলে যেগুলোতে কনসোল লগ আছে সেগুলোতে logger.info / logger.warn এইগুলা ব্যবহার করব । তারপর রান হলে এইটা একটা combined.log ফাইলে এইগুলা দেখা যাবে।

console এও লগ দেখার জন্য

new winston.transports.Console(),

এইটা অ্যাড করা লাগবে যেইটা অলরেডি অ্যাড করে রাখা হয়েছে।

### 12.5 (Separate Success logs)

import path from 'path'

import winston from 'winston'

const logger = winston.createLogger({

  level: 'info',

  format: winston.format.json(),

  transports: [

    new winston.transports.Console(),

    new winston.transports.File({

      filename: path.join(process.cwd(), 'logs', 'winston', 'success.log'),

      level: 'info',

    }),

  ],

})

const erroLogger = winston.createLogger({

  level: 'info',

  format: winston.format.json(),

  transports: [

    new winston.transports.Console(),

    new winston.transports.File({

      filename: path.join(process.cwd(), 'logs', 'winston', 'error.log'),

      level: 'error',

    }),

  ],

})

export { logger, erroLogger }

### 12.6 (Beatify and Customize the log)

import path from 'path'

import { createLogger, format, transports } from 'winston'

const { combine, timestamp, printf, prettyPrint } = format

//custom log format

const myFormat = printf(({ level, message, timestamp }) => {

  const date = new Date(timestamp)

  const hour = date.getHours()

  const minutes = date.getMinutes()

  const seconds = date.getSeconds()

  return `${date.toDateString()} ${hour}:${minutes}:${seconds} ${level}: ${message}`

})

const logger = createLogger({

  level: 'info',

  format: combine(timestamp(), myFormat, prettyPrint()),

  transports: [

    new transports.Console(),

    new transports.File({

      filename: path.join(process.cwd(), 'logs', 'winston', 'success.log'),

      level: 'info',

    }),

  ],

})

const erroLogger = createLogger({

  level: 'info',

  format: combine(timestamp(), myFormat, prettyPrint()),

  transports: [

    new transports.Console(),

    new transports.File({

      filename: path.join(process.cwd(), 'logs', 'winston', 'error.log'),

      level: 'error',

    }),

  ],

})

export { logger, erroLogger }

### 12.7 (Manage Logs using a library) Final

প্রথমে npm install winston-daily-rotate-file এইটা ইনষ্টল করতে হবে।

শুধু এইটা logger.ts ফাইলে পেষ্ট করলেই পুরো কাজ হয়ে যাবে।

import path from 'path'

import { createLogger, format, transports } from 'winston'

import DailyRotateFile from 'winston-daily-rotate-file'

const { combine, timestamp, printf, prettyPrint } = format

//custom log format

const myFormat = printf(({ level, message, timestamp }) => {

  const date = new Date(timestamp)

  const hour = date.getHours()

  const minutes = date.getMinutes()

  const seconds = date.getSeconds()

  return `${date.toDateString()} ${hour}:${minutes}:${seconds} ${level}: ${message}`

})

const logger = createLogger({

  level: 'info',

  format: combine(timestamp(), myFormat, prettyPrint()),

  transports: [

    new transports.Console(),

    new DailyRotateFile({

      filename: path.join(

        process.cwd(),

        'logs',

        'winston',

        'successes',

        '%DATE%-success.log'

      ),

      datePattern: 'YYYY-MM-DD-HH',

      zippedArchive: true,

      maxSize: '20m',

      maxFiles: '14d',

    }),

  ],

})

const erroLogger = createLogger({

  level: 'error',

  format: combine(timestamp(), myFormat, prettyPrint()),

  transports: [

    new transports.Console(),

    new DailyRotateFile({

      filename: path.join(

        process.cwd(),

        'logs',

        'winston',

        'successes',

        '%DATE%-error.log'

      ),

      datePattern: 'YYYY-MM-DD-HH',

      zippedArchive: true,

      maxSize: '20m',

      maxFiles: '14d',

    }),

  ],

})

export { logger, erroLogger }

### 12.8 (Error Handling)

প্রথমে src এর মধ্যে error নামক একটি ফোল্ডার খুলে সেখানে ApiError.ts এর মধ্যে নিচের কোড পেষ্ট করে দিতে পারি-

class ApiError extends Error {

  statusCode: number

  constructor(statusCode: number, message: string | undefined, stack = '') {

    super(message)

    this.statusCode = statusCode

    if (stack) {

      this.stack = stack

    } else {

      Error.captureStackTrace(this, this.constructor)

    }

  }

}

export default ApiError

src/interface/error.ts এর মধ্যে এরকম ইন্টারফেস তৈরি করতে হবে-

export type IGenericErrorMessage = {

  path: string | number

  message: string

}

src/interface/common.ts এর মধ্যে এরকম ইন্টারফেস তৈরি করতে হবে-

import { IGenericErrorMessage } from './error'

export type IGenericErrorResponse = {

  statusCode: number

  message: string

  errorMessages: IGenericErrorMessage[]

}

src/error/handleValidationError.ts এর মধ্যে নিচের দেওয়া কোড কপি পেষ্ট করতে হবে-

import mongoose from 'mongoose'

import { IGenericErrorResponse } from '../interface/common'

import { IGenericErrorMessage } from '../interface/error'

const handleValidationError = (

  error: mongoose.Error.ValidationError

): IGenericErrorResponse => {

  const errors: IGenericErrorMessage[] = Object.values(error.errors).map(

    (el: mongoose.Error.ValidatorError | mongoose.Error.CastError) => {

      return {

        path: el?.path,

        message: el?.message,

      }

    }

  )

  const statusCode = 400

  return {

    statusCode,

    message: 'Validation Error',

    errorMessages: errors,

  }

}

export default handleValidationError

src/error/handleCastError.ts এর মধ্যে নিচের দেওয়া কোড কপি পেষ্ট করতে হবে-

import mongoose from 'mongoose'

import { IGenericErrorMessage } from '../interfaces/error'

const handleCastError = (error: mongoose.Error.CastError) => {

  const errors: IGenericErrorMessage[] = [

    {

      path: error.path,

      message: 'Invalid Id',

    },

  ]

  const statusCode = 400

  return {

    statusCode,

    message: 'Cast Error',

    errorMessages: errors,

  }

}

export default handleCastError

### Zod Installation and Server.ts

প্রথমে npm i zod ইনষ্টল করতে হবে

src/error/handleZodError.ts এর মধ্যে নিচের দেওয়া কোড কপি পেষ্ট করতে হবে-

import { ZodError, ZodIssue } from 'zod'

import { IGenericErrorResponse } from '../interface/common'

import { IGenericErrorMessage } from '../interface/error'

const handleZodError = (error: ZodError): IGenericErrorResponse => {

  const errors: IGenericErrorMessage[] = error.issues.map((issue: ZodIssue) => {

    return {

      path: issue?.path[issue.path.length - 1],

      message: issue?.message,

    }

  })

  const statusCode = 400

  return {

    statusCode,

    message: 'Validation Error',

    errorMessages: errors,

  }

}

export default handleZodError

app ফোল্ডারের মধ্যে middlewares ফোল্ডার তৈরি করে সেখানে globalErrorHandler.ts

ফাইলের মধ্যে নিচের কোড পেষ্ট করতে হবে-

/\* eslint-disable no-unused-vars \*/

/\* eslint-disable no-unused-expressions \*/

/\* eslint-disable @typescript-eslint/no-unused-vars \*/

import { ErrorRequestHandler, NextFunction, Request, Response } from 'express'

import { ZodError } from 'zod'

import config from '../../config'

import ApiError from '../../error/ApiError'

import handleCastError from '../../error/handleCastError'

import handleValidationError from '../../error/handleValidationError'

import handleZodError from '../../error/handleZodError'

import { IGenericErrorMessage } from '../../interface/error'

const globalErrorHandler: ErrorRequestHandler = (

  error,

  req: Request,

  res: Response,

  next: NextFunction

) => {

  config.env === 'development'

    ? console.log(`🐱‍🏍 globalErrorHandler ~~`, { error })

    : console.log(`🐱‍🏍 globalErrorHandler ~~`, error) //errorlogger.error(`🐱‍🏍 globalErrorHandler ~~`, error);

  let statusCode = 500

  let message = 'Something went wrong !'

  let errorMessages: IGenericErrorMessage[] = []

  if (error?.name === 'ValidationError') {

    const simplifiedError = handleValidationError(error)

    statusCode = simplifiedError.statusCode

    message = simplifiedError.message

    errorMessages = simplifiedError.errorMessages

  } else if (error instanceof ZodError) {

    const simplifiedError = handleZodError(error)

    statusCode = simplifiedError.statusCode

    message = simplifiedError.message

    errorMessages = simplifiedError.errorMessages

  } else if (error?.name === 'CastError') {

    const simplifiedError = handleCastError(error)

    statusCode = simplifiedError.statusCode

    message = simplifiedError.message

    errorMessages = simplifiedError.errorMessages

  } else if (error instanceof ApiError) {

    statusCode = error?.statusCode

    message = error.message

    errorMessages = error?.message

      ? [

          {

            path: '',

            message: error?.message,

          },

        ]

      : []

  } else if (error instanceof Error) {

    message = error?.message

    errorMessages = error?.message

      ? [

          {

            path: '',

            message: error?.message,

          },

        ]

      : []

  }

  res.status(statusCode).json({

    success: false,

    message,

    errorMessages,

    stack: config.env !== 'production' ? error?.stack : undefined,

  })

}

export default globalErrorHandler

পেষ্ট করার পর app.ts ফাইলে গিয়ে সেখানে লিখতে হবে (মানে globalErrorHandler কে ডিফাইন করে দিতে হবে)-

app.user(globalErrorHandler)

সার্ভার যেন হঠাত বন্ধ না হয়ে যায় এইজন্য server.ts ফাইলের মধ্যে bootstrap function এর মধ্যে let server:Server ( import {Server} from ‘http’) -- এইটা করা আছে উপরের server.ts ফাইলের মধ্যে।

তারপর যেইখানে server listen করতেছে সেখানে server = app.listen করতে হবে।

তারপর ঐ bootstrap function এর মধ্যেই নিচের কোড লিখতে হবে--

 process.on('unhandledRejection', error => {

    console.log('Server is Closing...............')

    if (server) {

      server.close(() => {

        console.log(error) // errorlogger.error(error)

        process.exit(1)

      })

    } else {

      process.exit(1)

    }

  })

তারপর bootstrap ফাংশনের উপরে নিচের কোড লিখতে হবে ---

process.on('uncaughtException', () => {

  console.log('Uncaught Exception is Detected')

  process.exit(1)

})

তাহলে server.ts ফাইলের পুরো কোড হবে এমন ---

import { Server } from 'http'

import mongoose from 'mongoose'

import app from './app'

import config from './config'

process.on('uncaughtException', () => {

  console.log('Uncaught Exception is Detected')

  process.exit(1)

})

let server: Server

async function bootstrap() {

  try {

    await mongoose.connect(config.database\_url as string)

    console.log('Database Connected')

    server = app.listen(config.port, () => {

      console.log(`Server is Running at ${config.port}`)

    })

  } catch (err) {

    console.log('Error Occuered', err)

  }

  process.on('unhandledRejection', error => {

    console.log('Server is Closing...............')

    if (server) {

      server.close(() => {

        console.log(error) // errorlogger.error(error)

        process.exit(1)

      })

    } else {

      process.exit(1)

    }

  })

}

bootstrap()

process.on('SIGTERM', () => {

  console.log('Sigterm is received') //logger.info('Sigterm is received')

  if (server) {

    server.close()

  }

})

-----

তারপর যে যায়গায় আমরা zod ইমপ্লিমেন্ট করতে চাই সেই যায়গায় গিয়ে কাজ করতে হবে। ধরা যাক এখানে আমরা কোন অর্গানাইজেশনের create user করতেছি সেক্ষেত্রে ZOD ইমপ্লিমেন্ট করতে হবে-

তো user ফোল্ডারের মধ্যে user.valudation.ts নামে একটি ফাইল তৈরি করলাম-

সেখানে নিচের কোড কপি পেষ্ট করে দিব- (এইটা আসলে Edit, Update,Delete সব ক্ষেত্রেই ব্যবহার করা লাগবে)

import { z } from 'zod'

const createUserZodSchema = z.object({

  body: z.object({

    role: z.string({

      required\_error: 'role is required',

    }),

    password: z.string().optional(),

  }),

})

export const UserValidation = {

  createUserZodSchema,

}

তো এই টা আসলে ব্যবহার করতেছি যখন আমাকে user রিকুয়েষ্ট করতেছে user create করতে তাহলে একটি zod validation middleware লিখতে হবে router এর আগে ।

তো zod validation লিখব app/middleware/validateRequset.ts

validateRequest.ts ফাইলে নিচের কোড লিখব

import { NextFunction, Request, Response } from 'express'

import { AnyZodObject } from 'zod'

const validateRequest =

  (schema: AnyZodObject) =>

  async (req: Request, res: Response, next: NextFunction) : Promise<void>

=> {

    try {

      await schema.parseAsync({

        body: req.body,

        query: req.query,

        params: req.params,

        cookies: req.cookies,

      })

      return next()

    } catch (error) {

      next(error)

    }

  }

export default validateRequest

এত validateRequest রাউটারে middleware হিসেবে ব্যবহার করতে হবে-

router.post(

  '/create-user',

  validateRequest(UserValidation.createUserZodSchema),

  usersController.createUser

)

------------------xxxx-------------------xxxx--------------------xxxx----------------------

### Authentication of Project

প্রথমে Bcrypt ইনষ্টল করতে হবে- npm i bcrypt

.env ফাইলে গিয়ে BCRYPT\_SALT\_ROUNDS=12

লিখব।

তারপর index.ts ফাইলে গিয়ে bycrpt\_salt\_rounds: process.env.BCRYPT\_SALT\_ROUNDS,

লিখব

তারপর যে জায়গায় hash করব সেখানকার services ফাইলে গিয়ে কোড লিখব।

user.password = await bcrypt.hash(user.password,Number(config.bycrpt\_salt\_rounds))

এতে করে Database এ Hash ভাবে ষ্টোর হবে পাসওয়ার্ড।

যদি ডাটাবাসে একের অধিক ইউজারের জন্য পাসওয়ার্ড হ্যাশ করতে হয় (যেমন- Admin, Faculty, User) তাহলে সেইটা মডেলে গিয়ে করত হবে-

ModelSchema.pre('save', async function (next) {

  this.password = await bcrypt.hash(

    this.password,

    Number(config.bycrpt\_salt\_rounds)

  )

  next()

})

পাসওয়ার্ড যেহেতু খুব সেনসিটিভ তথ্য সেহেতু একে রিটার্ন হিসেবে দেওয়া উচিত না। এইজণ্য model এ গিয়ে password কে select:0 করে দিতে হবে।

const { password, ...userWithoutPassword } = createdUser.toObject()

এই লাইনটি লিখতে হবে যদি পাসওয়ার্ড আমরা না দেখাতে চাই।

এখানে password কে আমরা userWithoutPassword দ্বারা চেঞ্জ করে পাঠাচ্ছি।

### TYPESCRIPT JWT

১। প্রথমে npm i jsonwebtoken ইনষ্টল করতে হবে।

২। যে service ফাইলে কাজ করব সেইখানে প্রথমে ফাইল import করে নিব।

import jwt, { Secret } from 'jsonwebtoken'

তারপর নিচের প্রোগ্রামে মতো প্রোগ্রাম লিখব---

const loginUser = async (

  phoneNumber: string,

  password: string

): Promise<string> => {

  const userExist = await User.findOne({ phoneNumber })

  if (!userExist) {

    throw new ApiError(404, 'User not found')

  }

  const user = new User()

  const passwordMatched = await user.isPasswordMatched(

    password,

    userExist.password

  )

  if (!passwordMatched) {

    throw new ApiError(401, 'Invalid password')

  }

  const accessToken = generateAccessToken(

    userExist.\_id.toString(),

    userExist.role

  )

  return accessToken

}

এখানে generateAccessToken কে নিচে ফাংশন কল করতে হবে-

const generateAccessToken = (userId: string, role: string): string => {

  const token = jwt.sign({ userId, role }, config.jwt.jwt\_secret as Secret, {

    expiresIn: config.jwt.jwt\_expires,

  })

  return token

}

### React দিয়ে প্রজেক্ট তৈরি করার নিয়ম (লেটেষ্ট)

npm create vite@latest দিয়ে প্রজেক্ট ফাইল তৈরি করে নিতে হবে

এরপর ফোল্ডারের ভিতর গিয়ে কমান্ড উইন্ডো ওপেন করে npm I লিখে node\_modules ইন্সটল করতে হবে।

এখানে npm run dev দিয়ে সার্ভার রান করতে হবে। তবে এখানে সার্ভার রান করলে পোর্ট আসবে 5172 এইটাকে 3000 এ নিয়ে আসতে হলে vite.config.ts ফাইলে গিয়ে plugins এর নিচে

server:{

    port:3000

  }

লিখলে সার্ভার 3000 এ রান হবে।

### React Router Install করার টেকনিক-

npm i react-router-dom

তারপর main.tsx ফাইলে গিয়ে

App কে সরায়ে দিয়ে <RouterProvider router={routes}/>

import React from 'react'

import ReactDOM from 'react-dom/client'

import { RouterProvider } from 'react-router-dom'

import './index.css'

import routes from './routes'

ReactDOM.createRoot(document.getElementById('root')!).render(

  <React.StrictMode>

    <RouterProvider router={routes}/>

  </React.StrictMode>,

)

routes/index.ts

import { createBrowserRouter } from "react-router-dom";

import App from "../App";

import Home from "../page/Home";

import Login from "../page/login";

const routes = createBrowserRouter([

  {

    path:"/",

    element: <App/>

  },

  {

    path:"/home",

    element: <Home/>

  },

  {

    path:"/login",

    element: <Login/>

  }

]);

export default routes

### Redux Toolkit install

npm install @reduxjs/toolkit

npm install react-redux

redux/store.ts এর মধ্যে

import { configureStore } from "@reduxjs/toolkit";

export const store = configureStore({

  reducer: {},

});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

main.tsx এ গিয়ে

<Provider store={store}>

     <RouterProvider router={routes}/>

   </Provider>

store প্রোভাইড করে দিতে হবে গ্লোবালি

এরপর app/hooks.ts এই ফাইলের মধ্যে গিয়ে নিচের কোড পেষ্ট করে দিতে হবে—

import type { TypedUseSelectorHook } from "react-redux";

import { useDispatch, useSelector } from "react-redux";

import type { AppDispatch, RootState } from "./store";

// Use throughout your app instead of plain `useDispatch` and `useSelector`

export const useAppDispatch: () => AppDispatch = useDispatch;

export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;

তারপর ফিচার আকারে স্লাইস তৈরি করতে হবে—

features/featuresNameFolder/featureFileName.ts

import { createSlice } from "@reduxjs/toolkit";

interface bookInitalSchema {

  wishListedBooks: { bookName: string; readed: boolean; userEmail: string }[];

}

const initialState: bookInitalSchema = {

  wishListedBooks: [],

};

const wishListSLice = createSlice({

  name: "wishlist",

  initialState,

  reducers: {

    addToWishList:(state,action)=>{

        state.wishListedBooks.push(action.payload)

    }

  },

});

export const {} = wishListSLice.actions;

export default wishListSLice.reducer;

এই slice তৈরি করার পর এইটাকে store এ গিয়ে import করে কানেক্ট করতে হবে এইভাবে-

import { configureStore } from "@reduxjs/toolkit";

import postsReducer from './features/api/postReducer'

export const store = configureStore({

  reducer: {

    posts: postsReducer,

  },

middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(),

});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

এইভাবে কানেক্ট করার পর chromedevtool এ এইটার instance দেখা যাবে। তার মানে এইটা কাজ করতেছে।

### CRUD operation By Redux Toolkit

তো এইটার জন্য প্রথমে একটী features/api/apiSlice.ts এর মধ্যে api create করতে হবে- নিচের কোড গুলো পেষ্ট করতে হবে-

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";

export const apiSlice = createApi({

  reducerPath: "apiSlice",

  baseQuery: fetchBaseQuery({ baseUrl: "http://localhost:5000" }),

  endpoints: () => ({}),

});

এইখানে apiSlice তৈরি করে enpoints গুলো ফাকা রাখা যায় কারণ এইখানে ব্যাপারটা পলিউট হয়ে যেতে পারে এইজন্য endpoints গুলো আমরা আলাদা ফাইলে রাখব। সেজন্য fileNameApi.ts নামে একটা ফাইল তৈরি করে সেই ফাইলের ভিতর endpoints গুলো রাখব।

(data Get করার জন্য নিচের পদ্ধতি অবলম্বন করতে হবে-)

import { apiSlice } from "../api/apiSlice";

const bookApi = apiSlice.injectEndpoints({

  endpoints: (builder) => ({

    getAllBooks: builder.query({

      query: () => "/api/v1/books/",

    }),

  }),

});

export const { useGetAllBooksQuery } = bookApi;

(data Post করার জন্য নিচের পদ্ধতি অবলম্বন করতে হবে- )

import { apiSlice } from "../api/apiSlice";

const userApi = apiSlice.injectEndpoints({

  endpoints: (builder) => ({

    registerUser: builder.mutation({

      query: (data) => ({

        url: "/api/v1/users/create-user",

        method: "POST",

        body: data,

      }),

    }),

  }),

});

export const { useRegisterUserMutation } = userApi;

এখন এইটাকেও store এ গিয়ে কানেক্ট করে নিতে হবে নিচের কোডের মতো-

import { configureStore } from "@reduxjs/toolkit";

import wishListReducer from "./features/wishlist/wishListSlice";

import { apiSlice } from "./features/api/apiSlice";

export const store = configureStore({

  reducer: {

    wishList: wishListReducer,

    [apiSlice.reducerPath]:apiSlice.reducer //this line for apislice reducer

  },

  middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(),

});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

এখন যদি ফ্রন্ট এন্ডে Data নিয়ে আসতে চাই তাহলে সেই Component এ গিয়ে

const {data, isLoading} = useGetAllBooksQuery(undefined)

যদি ডাটা post করতে হয় তাহলে এইভাবে ডাটা নিয়ে আসতে হবে বা কাজ করতে হবে-

import { api } from "@/redux/api/apiSlice";

const productApi = api.injectEndpoints({

  endpoints: (builder) => ({

    getProducts: builder.query({

      query: () => "/products",

    }),

    singleProduct: builder.query({

      query: (id) => `/product/${id}`,

    }),

    postComment: builder.mutation({

      query: ({ id, data }) => ({

        url: `/comment/${id}`,

        method: "POST",

        body: data,

      }),

      invalidatesTags: ["comments"],

    }),

    getComment: builder.query({

      query: (id) => `/comment/${id}`,

      providesTags: ["comments"],

    }),

  }),

});

export const {

  useGetCommentQuery,

  useGetProductsQuery,

  usePostCommentMutation,

  useSingleProductQuery,

} = productApi;

এই গুলা আবার ফ্রন্ট এন্ডে কাজ করতে হয় নিচের মতো কোড করে-

  const { data, isLoading } = useGetAllBooksQuery(undefined);

 const navigate = useNavigate();

  const [loginUser, { isLoading, isError, isSuccess }] = useLoginUserMutation();

  const handleSubmit = async (e) => {

    e.preventDefault();

    const email = e.target.email.value;

    const password = e.target.password.value;

    const data = {

      email,

      password,

    };

    const response = await loginUser(data);

    if (response.data?.success) {

      const accessToken = response.data.data.accessToken;

      localStorage.setItem("accessToken", accessToken);

      navigate("/");

    }

  };

### Next JS

1. নেক্সট জেএস ইনষ্টল করার জন্য এই কমান্ড চালাতে হবে–

npx create-next-app@latest folder\_name

2. তারপর কিছু প্রশ্ন করবে সেগুলোর উত্তর দিতে হবে-

3. Next Js এ রাউট করার জন্য একটা ফোল্ডার নিয়ে সেই ফোল্ডারের ভেতর একটা ফাইল নিয়ে ফাইলের নামটি ছোট হাতের দিতে হবে। এবং সেই ফাইলের নামটি localhost:3000/ এর পর লিখে দিলে সেটি সেই পেজে চলে যাবে।

4. \_app.js ফাইলে গিয়ে নিচের কোড পেষ্ট করে দিতে হবে-

import '@/styles/globals.css'

export default function App({ Component, pageProps }) {

  const getLayout = Component.getLayout || ((page) => page);

  return getLayout(<Component {...pageProps} />);

}

1. src এর মধ্যে assets Folder তৈরি করতে হবে যেখানে ইমেজ ফাইল সহ অন্যান্য ফাইল গুলো থাকবে
2. src এর মধ্যে components ফোল্ডার তৈরি করতে হবে এবং সেখানে Layouts ফোল্ডারের মধ্যে Layout গুলো থাকবে যেমন RootLayout.js
3. এখন এই লেআউট কম্পোনেন্ট কে অন্য কম্পোনেন্টের সাথে কানেক্ট করার জন্য যে কম্পোনেন্টে কানেক্ট করব সেখানে গিয়ে নিচে গিয়ে নিচের এই কোডটি লিখতে হবে-

4. Product Id নিয়ে অনেক সময় ডিটেইলস পেজে যেতে হয় তখন products/[productId].js নামে একটি ফাইল খুলত হবে এবং সেখানে const router =useRouter() ফাংশন নিয়ে প্রোডাক্টের প্যারামিটার কুয়েরি থেকে নিতে হবে নিচের কোডের মতো হবে-

import { useRouter } from "next/router";

const productDetails = () => {

    const router = useRouter()

    return (

        <div>

            <h1>This is Details page of {router.query.productId}</h1>

        </div>

    );

};

export default productDetails;

1. যদি আমাদের কোন লে আউট থাকে তাহলে সেই লে আউটের মাঝে আমরা কন্টেন্ট গুলো চাইল্ড আকারে দিতে পারি নিচে প্রোগ্রামের উদাহরণ দেওয়া হলো-

import RootLayout from "@/components/Layouts/RootLayout";

import Banner from "@/components/UI/Banner";

import Head from "next/head";

const HomePage = () => {

  return (

    <>

      <Head>

        <title>PH-News Portal</title>

        <meta

          name="description"

          content="This is news portal of programming hero made by next-js"

        />

        <meta name="viewport" content="width=device-width, initial-scale=1" />

        <link rel="icon" href="/favicon.ico" />

      </Head>

      <Banner />

    </>

  );

};

export default HomePage;

HomePage.getLayout = function getLayout(page) {

  return <RootLayout>{page}</RootLayout>;

};

export const getStaticProps = async () => {

  const res = await fetch("http://localhost:5000/news")

  const data = await res.json();

  console.log(data);

  return {

    props: {

      allNews: data

    }

  }

}

1. ডাইনামিক ভাবে ID দিয়ে ডাটা নিয়ে আসতে চাই ডিটেইলস পেজে তাহলে getStaticPaths এবং getStaticProps ব্যবহার করতে হবে। নিচের কোড ফলো করতে হবে-

const ProductDetails = ({ products }) => {

    return (

        <div>

            {products.productName}

        </div>

    );

};

export const getStaticPaths = async () => {

    const res = await fetch("http://localhost:5000/api/products")

    const products = await res.json()

    console.log(products)

    const paths = products.data.map((product) => ({

        params: { productsId: product.\_id }

    }))

    return { paths, fallback: false }

}

export const getStaticProps = async (context) => {

    const { params } = context

    const res = await fetch(`http://localhost:5000/api/product/${params.productsId}`)

    const data = await res.json();

    console.log(data)

    return {

        props: {

            products: data

        }

    }

}

export default ProductDetails;

### MySQL (POSTgre)

ডাটাবেজ (postgres) ক্রিয়েট করার জন্য প্রথমে একটি ফোল্ডারের মধ্যে intro.sql নামক ফাইল নিয়ে সেটিকে VS Code এ ওপেন করতে হবে।

For Database Creating command below-

CREATE DATABASE databaseName;

For Creating a table in the Database the command below-

### JavaScript দিয়ে Backend

**প্রজেক্ট সেটাপ করার জন্য প্রয়োজনীয় কমান্ড (02) (JavaScript) Scratch to Advanced (Important)**

1. প্রথমে npm init –y দিতে হবে

2. npm i express cors nodemon dotenv mongodb jsonwebtoken mongoose

তারপর তিনটা ফাইল বানাতে হবে-

src/ server.ts , src/app.ts

src/config/index.ts

server.js ফাইলের মধ্যে ডাটাবেজ কানেক্ট করতে হবে-

import mongoose from 'mongoose'

import app from './app'

import config from './config/index'

import { erroLogger, logger } from './shared/logger'

async function bootstrap() {

  try {

    await mongoose.connect(config.database\_url as string)

    logger.info(`Database Connected`)

    app.listen(config.port, () => {

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

    })

  } catch (err) {

    erroLogger.error(`Error occured`, err)

  }

}

bootstrap()

app.js

const cors = require('cors')

const express = require('express')

const routes = require('./app/routes');

const globalErrorHandler = require('./app/middlewares/globalErrorHandler');

const app = express();

app.use(cors())

//parser

app.use(express.json())

app.use(express.urlencoded({ extended: true }))

app.use('/api/v1/', routes)

app.use(globalErrorHandler)

module.exports = app

app.js ফাইলে route গুলো লিখার ফলে ফাইলটি পলিউট হয়ে যাওয়ার সম্ভাবনা বেশি থাকায় আমরা Routes গুলোকে আলাদা ফাইলে রাখব। src/app/routes নামে একটি ফোল্ডার বানিয়ে routes.js এ রাউটস গুলো রাখতে পারি।

const express = require('express');

const ProductRoutes = require('../module/products/productRoute')

const router = express.Router()

const moduleRoutes = [

    {

        path: '/product',

        route: ProductRoutes,

    }

]

moduleRoutes.forEach(route => router.use(route.path, route.route))

module.exports = router

index.js এর মধ্যে Environment এর যে ভেরিয়েবল গুলো ছিল সেগুলো এক্সপোর্ট করে দিতে হবে-

const dotenv = require('dotenv')

const path = require('path')

dotenv.config({ path: path.join(process.cwd(), '.env') })

module.exports = {

    env: process.env.NODE\_ENV,

    port: process.env.PORT,

    database\_url: process.env.DATABASE\_URL,

    bycrpt\_salt\_rounds: process.env.BCRYPT\_SALT\_ROUNDS,

}

একটি ফিচার তৈরি করতে হলে –

Schema & Model 🡪services-> controller🡪route

src/module/featuresnameFolder

nameModel.js হবে এইরকম-

const mongoose = require("mongoose")

const productSchema = mongoose.Schema({

    product\_name: {

        type: String,

        required: true,

    },

    price: {

        type: Number,

        required: true,

    },

    old\_price: Number,

    brand: {

        type: String,

        required: true,

    },

    description: {

        type: String,

        required: true,

    },

    color: {

        type: String,

        required: true,

    },

    img: {

        type: String,

        required: true,

    },

    imgThree: String,

    availibility: {

        type: Boolean,

        required: true,

    },

    category: {

        type: String,

        required: true,

    },

    ratings: {

        type: Number,

    },

    size: [{

        type: String,

        required: true,

        enum: ['S', 'M', 'L', 'XL', 'XXL', 'XXXL']

    }],

},

    {

        timestamps: true,

    }

)

const Product = new mongoose.model("Products", productSchema)

module.exports = Product

nameRoute.js

const express = require('express');

const productController = require('./productController')

const router = express.Router();

router.post('/addproduct', productController.addProduct);

router.get('/', productController.getProduct)

module.exports = router

nameServices.js (এইখানে কুয়েরী অথবা বিজনেস লজিক গুলা হবে)

const ApiError = require("../../error/ApiError");

const Product = require("./productModel")

const addProduct = async (product) => {

    const createProduct = Product.create(product)

    if (!createProduct) {

        // alter by Api Error

        res.status(400).json({

            success: false,

            message: 'Product doesnot added successfully!'

        })

    }

    return createProduct;

}

const getProduct = async () => {

    const allProduct = await Product.find()

    if (allProduct.length === 0) {

        throw new ApiError(400, "There have no products")

    }

    return allProduct;

}

module.exports = {

    addProduct,

    getProduct

}

export const usersService = {

  createUser,

}

nameController.ts

const productService = require("./productService")

const addProduct = async (req, res, next) => {

    try {

        const product = req.body

        const result = await productService.addProduct(product)

        res.status(200).json({

            success: true,

            message: 'Product added successfully!',

            data: result,

        })

    } catch (err) {

        next(err)

    }

}

const getProduct = async (req, res, next) => {

    try {

        const result = await productService.getProduct()

        res.status(200).json({

            success: true,

            message: 'Product retrived successfully!',

            data: result,

        })

    } catch (err) {

        next(err)

    }

}

module.exports = {

    addProduct,

    getProduct

}

Env ফাইলের মধ্যে

DATABASE\_URL=mongodb+srv://kinbaaNaki:ODSepvZbf7H9qcCw@cluster0.x5fc9ya.mongodb.net/kinbaaNaki?retryWrites=true&w=majority

PORT=5000

NODE\_ENV=development

BCRYPT\_SALT\_ROUNDS=10

### ( Javascript Error Handling)

প্রথমে src এর মধ্যে error নামক একটি ফোল্ডার খুলে সেখানে ApiError.js এর মধ্যে নিচের কোড পেষ্ট করে দিতে পারি-

class ApiError extends Error {

    constructor(statusCode, message, stack = '') {

        super(message);

        this.statusCode = statusCode;

        if (stack) {

            this.stack = stack;

        } else {

            Error.captureStackTrace(this, this.constructor);

        }

    }

}

module.exports = ApiError;

src/error/handleValidationError.js এর মধ্যে নিচের দেওয়া কোড কপি পেষ্ট করতে হবে-

const mongoose = require('mongoose')

const handleValidationError = (error) => {

    const errors = Object.values(error.errors).map(

        (el) => {

            return {

                path: el?.path,

                message: el?.message,

            }

        }

    )

    const statusCode = 400

    return {

        statusCode,

        message: 'Validation Error',

        errorMessages: errors,

    }

}

module.exports = handleValidationError

src/error/handleCastError.js এর মধ্যে নিচের দেওয়া কোড কপি পেষ্ট করতে হবে-

const handleCastError = (error) => {

    const errors = [

        {

            path: error.path,

            message: 'Invalid Id',

        },

    ]

    const statusCode = 400

    return {

        statusCode,

        message: 'Cast Error',

        errorMessages: errors,

    }

}

module.exports = handleCastError;

### Pagination কীভাবে করা হয়

প্রথমে paginationHelper.js নামক একটি ফাইল ক্রিয়েট করতে হবে—

const pick = (obj, keys) => {

    const finalObj = {};

    for (const key of keys) {

        if (obj && Object.hasOwnProperty.call(obj, key)) {

            finalObj[key] = obj[key];

        }

    }

    return finalObj;

}

const paginationFields = ['page', 'limit', 'sortBy', 'sortOrder']

const calculatePagination = (options) => {

    const page = Number(options.page || 1);

    const limit = Number(options.limit);

    const skip = (page - 1) \* limit;

    const sortBy = options.sortBy || 'createdAt';

    const sortOrder = options.sortOrder || 'desc'

    return {

        page,

        limit,

        skip,

        sortBy,

        sortOrder

    }

}

module.exports = {

    pick,

    paginationFields,

    calculatePagination

}

যখন কোন কন্ট্রোলারে API হিট করবে তখন নিচের কোড কাজ করবে—

const getProduct = async (req, res, next) => {

    const paginationOptions = pick(req.query, paginationFields)

    try {

        const result = await productService.getProduct(paginationOptions)

        res.status(200).json({

            success: true,

            message: 'Product retrived successfully!',

            meta: result.meta,

            products: result.allProduct,

        })

    } catch (err) {

        next(err)

    }

}

এইখানে pick দিয়ে query এবং page, limit, sortBy, sortOrder এইগুলা পাঠায়ে দেওয়া হয়।

এরপর কন্ট্রোলার থেকে যখন সার্ভিসে ডাটা হিট করবে—

const getProduct = async (paginationOptions) => {

    const { page, limit, skip, sortBy, sortOrder } = calculatePagination(paginationOptions);

    const sortConditions = {};

    if (sortBy && sortOrder) {

        sortConditions[sortBy] = sortOrder;

    }

    const allProduct = await Product.find().sort(sortConditions).skip(skip).limit(limit)

    const total = await Product.countDocuments();

    if (allProduct.length === 0) {

        throw new ApiError(400, "There have no products")

    }

    return {

        meta: {

            page,

            limit,

            total,

        },

        allProduct

    };

}

### JavaScript Authentication of Project

প্রথমে Bcrypt ইনষ্টল করতে হবে- npm i bcrypt

.env ফাইলে গিয়ে BCRYPT\_SALT\_ROUNDS=12

লিখব।

তারপর index.js ফাইলে গিয়ে bycrpt\_salt\_rounds: process.env.BCRYPT\_SALT\_ROUNDS,

লিখব

তারপর যে জায়গায় hash করব সেখানকার services ফাইলে গিয়ে কোড লিখব।

user.password = await bcrypt.hash(user.password,Number(config.bycrpt\_salt\_rounds))

এতে করে Database এ Hash ভাবে ষ্টোর হবে পাসওয়ার্ড।

যদি ডাটাবাসে একের অধিক ইউজারের জন্য পাসওয়ার্ড হ্যাশ করতে হয় (যেমন- Admin, Faculty, User) তাহলে সেইটা মডেলে গিয়ে করত হবে-

ModelSchema.pre('save', async function (next) {

  this.password = await bcrypt.hash(

    this.password,

    Number(config.bycrpt\_salt\_rounds)

  )

  next()

})

পাসওয়ার্ড যেহেতু খুব সেনসিটিভ তথ্য সেহেতু একে রিটার্ন হিসেবে দেওয়া উচিত না। এইজণ্য model এ গিয়ে password কে select:0 করে দিতে হবে।

const { password, ...userWithoutPassword } = createdUser.toObject()

এই লাইনটি লিখতে হবে যদি পাসওয়ার্ড আমরা না দেখাতে চাই।

এখানে password কে আমরা userWithoutPassword দ্বারা চেঞ্জ করে পাঠাচ্ছি।

### JAVASCIRPT JWT

১। প্রথমে npm i jsonwebtoken ইনষ্টল করতে হবে।

২। যে service ফাইলে কাজ করব সেইখানে প্রথমে ফাইল import করে নিব।

const jwt = require('jsonwebtoken')

তারপর নিচের প্রোগ্রামে মতো প্রোগ্রাম লিখব---

const addUser = async (user) => {

    user.password = await bcrypt.hash(user.password, 10)

    const createUser = await User.create(user)

    if (!createUser) {

        throw new ApiError(501, 'User Doesnot Added!!')

    }

    const { password, ...userWithoutPassword } = createUser.toObject()

    const accessToken = generateAccessToken(

        userWithoutPassword.\_id.toString(),

        userWithoutPassword.role,

        userWithoutPassword.email,

        userWithoutPassword.username

    )

    return { accessToken, userWithoutPassword }

}

এখানে generateAccessToken কে নিচে ফাংশন কল করতে হবে-

const generateAccessToken = (userId, role) => {

    const token = jwt.sign({ userId, role }, config.jwt.jwt\_secret, {

        expiresIn: config.jwt.jwt\_expires,

    })

    return token

}

controller সেকশনে গিয়ে- cookies এ accessToken ইমপ্লিমেন্ট করে দিতে হবে।

const signUpUser = async (req, res, next) => {

    try {

        const user = req.body

        const result = await usersService.addUser(user)

        res.cookie('accessToken', result.accessToken, {

            httpOnly: true,

        });

        res.status(200).json({

            success: true,

            message: 'User added successfully!',

            data: result,

        })

    } catch (err) {

        next(err)

    }

}