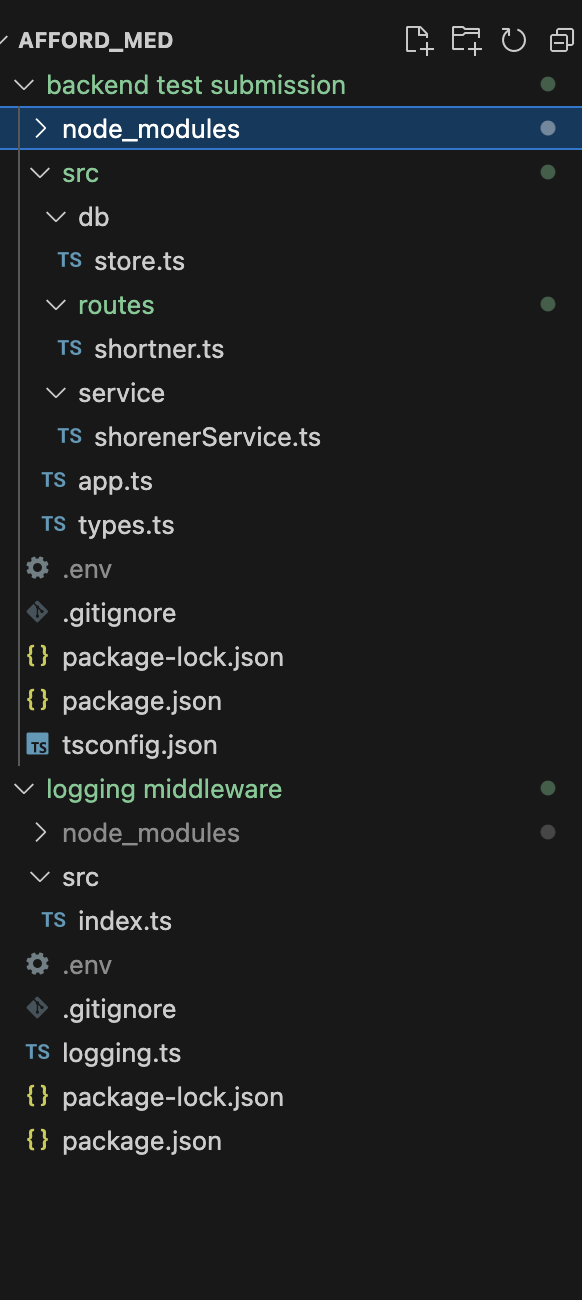
**Afford Medical Assignment For Backend :**

File Structure for ShortUrl :



Logging.ts:

import axios from 'axios'

import dotenv from 'dotenv'

dotenv.config()

const Log\_Endpoint = 'http://20.244.56.144/evaluation-service/logs'

const Allowed\_Stacks = ['backend' , 'frontend']

const Allowed\_Levels = ['debug' , 'info', 'warn', 'error' , 'fatal']

const Allowed\_Packages = [

'cache',

'controller',

'cron job',

'db',

'domain',

'handler',

'repository'

]

const Auth\_Token = process.env.Logging\_Api\_Token || " "

interface Log\_Params {

stack: 'backend' | 'frontend',

level: 'debug' | 'info' | 'warn' | 'error' | 'fatal',

package: typeof Allowed\_Packages[number],

message: string,

}

export async function log({stack , level , package: pkg , message}: Log\_Params):

Promise<void>{

try{

if(!Allowed\_Stacks.includes(stack)){

throw new Error(`Stack "${stack}" is not allowed.`)

}

if(!Allowed\_Levels.includes(level)){

throw new Error(`Level "${level}" is not allowed.`)

}

if(!Allowed\_Packages.includes(pkg)){

throw new Error(`Package "${pkg}" is not allowed.`)

}

await axios.post(

Log\_Endpoint,

{

stack,

level,

package: pkg,

message

},

{

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${Auth\_Token}`

},

}

)

}catch(err){

const error = err as Error

console.log('Logging failed: ' + error.message)

}

}

.env in backend test submission :

PORT=3000

HOST=http://localhost:3000

App.ts :

import express from 'express'

import dotenv from 'dotenv'

import shortenereRoutes from './routes/shortner'

dotenv.config()

const app = express()

const PORT = process.env.PORT || 3000

app.use(express.json())

app.use('/' , shortenereRoutes)

app.listen(PORT, () => {

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

})

ShorenerService (shortnerService) :

import {nanoid} from 'nanoid'

import geoip from 'geoip-lite'

import {urlStore , ShortURL} from '../db/store'

import {log} from '../../../logging middleware/logging'

const Host = process.env.HOST || 'http://localhost:3000'

export async function createShortURL(

longURL: string,

validity?: number,

customCode? : string

) {

validity = validity ?? 30;

const code = customCode || nanoid(6)

if(urlStore[code]){

await log({

stack: 'backend',

level: 'error',

package: 'handler',

message: `ShortCode already exists: ${code}`

})

throw new Error('ShortCode already exists. Please choose another one.')

}

const now = new Date()

const expires = new Date(now.getTime()+validity\*60000)

urlStore[code] = {

longURL,

createdAt: now,

expiresAt: expires,

clicks: []

}

await log({

stack: 'backend',

level: 'info',

package: 'repository',

message: `ShortURL created with code: ${code}`

})

return{

shortLink: `${Host}/${code}`,

expiry: expires.toISOString(),

}

}

export async function getAnalytics(code: string){

const data = urlStore[code]

if(!data){

await log({

stack: 'backend',

level: 'error',

package: 'repository',

message: `Analytics request failed for shortcode: ${code}`

})

throw new Error('Shortcode not found')

}

return{

totalClicks: data.clicks.length,

originalURL: data.longURL,

createdAt: data.createdAt.toISOString(),

expiresAt: data.expiresAt.toISOString(),

clicks: data.clicks,

}

}

export async function handleRedirect(code: string, req: any){

const data = urlStore[code]

if(!data){

await log({

stack: 'backend',

level: 'error',

package: 'handler',

message: `Redirect failed for shortcode: ${code}`

})

throw new Error('Shortcode not found')

}

if(new Date() > data.expiresAt){

await log({

stack: 'backend',

level: 'error',

package: 'handler',

message: `Link expired for shortcode: ${code}`

})

throw new Error('Link has expired')

}

const ip = req.ip

const geo = geoip.lookup(ip)

const location = geo ? `${geo.city}, ${geo.country}` : 'Unknown'

data.clicks.push({

timestamp: new Date(),

referrer: req.get('Referrer') || 'Direct',

location: location,

})

await log({

stack: 'backend',

level: 'info',

package: 'handler',

message: `Redirecting for shortcode: ${code} from ${location}`

})

return data.longURL

}

Store.ts :

export interface ShortURL{

longURL: string

createdAt: Date

expiresAt: Date

clicks: Array<{

timestamp: Date

referrer: string

location: string

}>

}

export const urlStore: Record<string , ShortURL> = {}

shortner.ts:

import express from 'express'

import {createShortURL , getAnalytics , handleRedirect} from '../service/shorenerService'

const router = express.Router()

router.post('/shorturls' , async(req,res)=>{

try{

const {url , validity , shortcode} = req.body

const result = await createShortURL(url , validity , shortcode)

res.status(201).json(result)

}

catch(err){

const error = err as Error

res.status(400).json({error: error.message})

}

})

router.get('/shorturls/:code' , async(req,res)=>{

try{

const result = await getAnalytics(req.params.code)

res.json(result)

}

catch(err){

const error = err as Error

res.status(404).json({error: error.message})

}

})

router.get('/:code' , async(req,res)=>{

try{

const result = await handleRedirect(req.params.code , req)

res.redirect(result)

}

catch(err){

const error = err as Error

res.status(404).json({error: error.message})

}

})

export default router

.env in logging middleware :

Logging\_Api\_Token = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..Tt8ojcevL4c676IvfiOyOn3ZTE4UIq3dSQC1jm2bSgA

Screenshot of postman API for shortUrl :

