**ПРИЛОЖЕНИЕ А**

**(обязательное)**

# **Листинг программного кода**

import stringHash from 'string-hash'  
  
export const ACCOUNTS = [  
 {  
 role: 'client',  
 email: 'avatar\_dj@mail.ru',  
 password: '12345',  
 passwordHash: stringHash('12345').toString(),  
 },  
 {  
 role: 'client',  
 email: 'indigo@gmail.com',  
 password: '1097535489fg',  
 passwordHash: stringHash('1097535489fg').toString(),  
 },  
 {  
 role: 'courier',  
 email: 'kessler.polly@mraz.org',  
 password: '99999fdffff',  
 passwordHash: stringHash('99999fdffff').toString(),  
 },  
 {  
 role: 'courier',  
 email: 'delphia.gutmann@hotmail.com',  
 password: 'jfpwsadkijfsdifh343',  
 passwordHash: stringHash('jfpwsadkijfsdifh343').toString(),  
 },  
 {  
 role: 'manager',  
 email: 'mireya16@turcotte.com',  
 password: '90583040gffgfg5',  
 passwordHash: stringHash('90583040gffgfg5').toString(),  
 },  
 {  
 role: 'manager',  
 email: 'nayeli.jacobi@yahoo.com',  
 password: '54fsd5f4sg6833348422\_\_fds',  
 passwordHash: stringHash('54fsd5f4sg6833348422\_\_fds').toString(),  
 },  
]

export const PATH\_TO\_DATABASE\_DIR = './db'  
export const PATH\_TO\_DATABASE = `${PATH\_TO\_DATABASE\_DIR}/pizzeria.db`  
export const PATH\_TO\_PUBLIC = './public'  
export const PATH\_TO\_PUBLIC\_IMAGES = `${PATH\_TO\_PUBLIC}/images`  
  
export const SERVER\_PORT = 4444  
export const CLIENT\_ERROR\_STATUS\_CODE = 400  
  
export enum RoutesPaths {  
 any = '\*',  
 root = '/',  
 backend = '/backend',  
 static = '/static',  
 images = '/images',  
 home = '/home',  
 signIn = '/sign-in',  
 signUp = '/sign-up',  
 catalog = '/catalog',  
 profile = '/profile',  
 comments = '/comments',  
 getAll = '/get-all',  
 addOne = '/add-one',  
 deleteOne = '/delete-one',  
 // client  
 clientMenu = '/menu/client',  
 getOrdersByClientId = '/get-orders-by-client-id',  
 declineOrder = '/decline-order',  
 // courier  
 courierMenu = '/menu/courier',  
 getOrdersByCourierId = '/get-orders-by-courier-id',  
 acceptOrder = '/accept-order',  
 // manager  
 managerMenu = '/menu/manager',  
 getGroupedReports = '/get-grouped-reports',  
 editCourierSalary = '/edit-courier-salary',  
}  
  
export enum ApiRouters {  
 api = '/api',  
 getAll = '/',  
 getSingle = '/:id',  
 putSingle = '/:id',  
 postSingle = '/',  
 deleteSingle = '/:id',  
}

export const messages = {  
 SERVER\_LISTENING: (port: number) => `Server listening at port ${port}`,  
 SUCCESSFULLY\_ADDED: 'Successfully added.',  
 SUCCESSFULLY\_UPDATED: 'Successfully updated.',  
 SUCCESSFULLY\_DELETED: 'Successfully deleted.',  
 DATABASE\_ERROR: 'Database error.',  
 DATABASE\_ERROR\_GET\_ALL: 'Get all.',  
 DATABASE\_ERROR\_GET\_ONE: 'Get one.',  
 DATABASE\_ERROR\_ADD: 'Add.',  
 DATABASE\_ERROR\_UPDATE: 'Update.',  
 DATABASE\_ERROR\_DELETE: 'Delete.',  
 NO\_SUCH\_ACCOUNT\_FOUND: 'No such account found.',  
 NO\_SUCH\_USER\_FOUND: 'No such user found.',  
}

export interface IModel {  
 id: number  
}  
  
export interface IUser extends IModel {  
 accountId: number  
}  
  
export interface IWorker extends IUser {  
 name: string  
 salary: number  
}  
  
export type AccountDTO = IModel & {  
 email: string  
 passwordHash: string  
}  
  
export type ClientDTO = IUser & {  
 name: string  
 phoneNumber: string  
 description: string  
}  
  
export type CourierDTO = IWorker & {  
 name: string  
 description: string  
}  
  
export type ManagerDTO = IWorker & {  
 name: string  
 description: string  
}  
  
export type CommentDTO = IModel & {  
 clientId: number  
 content: string  
 date: string  
}  
  
export type OrderDTO = IModel & {  
 pizzaId: number  
 clientId: number  
 courierId: number  
 statusId: number  
 address: string  
 startDate: string  
 endDate: string  
}  
  
export type PizzaDTO = IModel & {  
 name: string  
 description: string  
 price: number  
 imageUrl: string  
}  
  
export type ReportDTO = IModel & {  
 orderId: number  
 date: string  
 description: string  
}  
  
export type StatusDTO = IModel & {  
 type: string  
}

export enum AccountSchema {  
 id = 'id',  
 email = 'email',  
 password = 'passwordHash',  
}  
  
export enum ClientSchema {  
 id = 'id',  
 accountId = 'accountId',  
 name = 'name',  
 phone = 'phoneNumber',  
 description = 'description',  
}  
  
export enum CommentSchema {  
 id = 'id',  
 clientId = 'clientId',  
 content = 'content',  
 date = 'date',  
}  
  
export enum CourierSchema {  
 id = 'id',  
 accountId = 'accountId',  
 name = 'name',  
 salary = 'salary',  
 description = 'description',  
}  
  
export enum ManagerSchema {  
 id = 'id',  
 accountId = 'accountId',  
 name = 'name',  
 salary = 'salary',  
 description = 'description',  
}  
  
export enum OrderSchema {  
 id = 'id',  
 pizzaId = 'pizzaId',  
 clientId = 'clientId',  
 courierId = 'courierId',  
 statusId = 'statusId',  
 address = 'address',  
 startDate = 'startDate',  
 endDate = 'endDate',  
}  
  
export enum PizzaSchema {  
 id = 'id',  
 name = 'name',  
 description = 'description',  
 price = 'price',  
 imageUrl = 'imageUrl',  
}  
  
export enum ReportSchema {  
 id = 'id',  
 orderId = 'orderId',  
 date = 'date',  
 description = 'description',  
}  
  
export enum StatusSchema {  
 id = 'id',  
 type = 'type',  
}

export enum TableRoutes {  
 accounts = '/accounts',  
 clients = '/clients',  
 comments = '/comments',  
 couriers = '/couriers',  
 managers = '/managers',  
 orders = '/orders',  
 pizzas = '/pizzas',  
 reports = '/reports',  
 statuses = '/statuses',  
}  
  
export enum TableNames {  
 accounts = 'Accounts',  
 clients = 'Clients',  
 comments = 'Comments',  
 couriers = 'Couriers',  
 managers = 'Managers',  
 orders = 'Orders',  
 pizzas = 'Pizzas',  
 reports = 'Reports',  
 statuses = 'Statuses',  
}

import { Sequelize } from 'sequelize'  
import { PATH\_TO\_DATABASE } from '../constants/constants'  
  
export const sequelize = new Sequelize('sqlite::memory:', {  
 host: 'localhost', dialect: 'sqlite', pool: {  
 max: 5, min: 0, idle: 10000,  
 }, storage: PATH\_TO\_DATABASE,  
})

import \* as fs from 'fs'  
import { PATH\_TO\_DATABASE\_DIR } from '../constants/constants'  
import { ACCOUNTS } from '../constants/accounts'  
  
export const generateAccountsJson = () =>  
 fs.writeFileSync(`${PATH\_TO\_DATABASE\_DIR}/accounts.json`, JSON.stringify(ACCOUNTS, null, ' '))

import \* as fs from 'fs'  
import request from 'request'  
import { PATH\_TO\_PUBLIC\_IMAGES } from '../constants/constants'  
  
export const downloadImage = (uri: string, filePath: string, callback: () => void = () => {}) => {  
 request.head(uri, function (err: any, res: request.Response) {  
 console.log('content-type:', res.headers['content-type'])  
 console.log('content-length:', res.headers['content-length'])  
  
 request(uri).pipe(fs.createWriteStream(filePath)).on('close', callback)  
 })  
}  
  
export const downloadAllImages = (imgInfos: { fileName: string, url: string }[], callback = () => console.log('Successfully downloaded.')) => {  
 imgInfos.forEach(({ fileName, url }) => downloadImage(url, `${PATH\_TO\_PUBLIC\_IMAGES}/${fileName}.jpg`))  
 callback()  
}

import express from 'express'  
import { CLIENT\_ERROR\_STATUS\_CODE } from '../constants/constants'  
  
export const sendError = (errorMessage: string, res: express.Response) => res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: errorMessage })

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { AccountSchema } from '../constants/schemas'  
  
export const AccountModel = sequelize.define<any, any>(TableNames.accounts, {  
 [AccountSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [AccountSchema.email]: {  
 type: Sequelize.TEXT,  
 unique: true,  
 allowNull: false,  
 },  
 [AccountSchema.password]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.accounts,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { AccountSchema, ClientSchema } from '../constants/schemas'  
import { AccountModel } from './account.model'  
  
export const ClientModel = sequelize.define<any, any>(TableNames.clients, {  
 [ClientSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [ClientSchema.accountId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: AccountModel,  
 key: AccountSchema.id,  
 },  
 unique: true,  
 },  
 [ClientSchema.name]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [ClientSchema.phone]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [ClientSchema.description]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.clients,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { ClientSchema, CommentSchema } from '../constants/schemas'  
import { ClientModel } from './client.model'  
  
export const CommentModel = sequelize.define<any, any>(TableNames.comments, {  
 [CommentSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [CommentSchema.clientId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: ClientModel,  
 key: ClientSchema.id,  
 },  
 },  
 [CommentSchema.content]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [CommentSchema.date]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.comments,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { AccountSchema, CourierSchema } from '../constants/schemas'  
import { AccountModel } from './account.model'  
  
export const CourierModel = sequelize.define<any, any>(TableNames.couriers, {  
 [CourierSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [CourierSchema.accountId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: AccountModel,  
 key: AccountSchema.id,  
 },  
 unique: true,  
 },  
 [CourierSchema.name]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [CourierSchema.salary]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 },  
 [CourierSchema.description]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.couriers,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { AccountSchema, ManagerSchema } from '../constants/schemas'  
import { AccountModel } from './account.model'  
  
export const ManagerModel = sequelize.define<any, any>(TableNames.managers, {  
 [ManagerSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [ManagerSchema.accountId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: AccountModel,  
 key: AccountSchema.id,  
 },  
 unique: true,  
 },  
 [ManagerSchema.name]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [ManagerSchema.salary]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 },  
 [ManagerSchema.description]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.managers,  
 timestamps: false,  
})

import { ModelWrapper } from './modelWrapper'  
import { AccountModel as AccountM } from './account.model'  
import { ClientModel as ClientM } from './client.model'  
import { CommentModel as CommentM } from './comment.model'  
import { CourierModel as CourierM } from './courier.model'  
import { ManagerModel as ManagerM } from './manager.model'  
import { OrderModel as OrderM } from './order.model'  
import { PizzaModel as PizzaM } from './pizza.model'  
import { ReportModel as ReportM } from './report.model'  
import { StatusModel as StatusM } from './status.model'  
  
export const AccountModel = new ModelWrapper(AccountM)  
export const ClientModel = new ModelWrapper(ClientM)  
export const CommentModel = new ModelWrapper(CommentM)  
export const CourierModel = new ModelWrapper(CourierM)  
export const ManagerModel = new ModelWrapper(ManagerM)  
export const OrderModel = new ModelWrapper(OrderM)  
export const PizzaModel = new ModelWrapper(PizzaM)  
export const ReportModel = new ModelWrapper(ReportM)  
export const StatusModel = new ModelWrapper(StatusM)

export class ModelWrapper {  
 \_model: any  
  
 constructor(model: any) {  
 this.\_model = model  
 }  
  
 findAll = async (params: object) => (await this.\_model.findAll(params)).map((x: any) => x.dataValues)  
  
 findOne = async (params: object) => (await this.\_model.findOne(params))?.dataValues  
  
 create = async (params: object) => (await this.\_model.create(params)).dataValues  
  
 update = async (params: object, filter: object) => this.\_model.update(params, filter)  
  
 destroy = async (params: object) => this.\_model.destroy({ where: params })  
}

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { ClientSchema, CourierSchema, OrderSchema, PizzaSchema, StatusSchema } from '../constants/schemas'  
import { PizzaModel } from './pizza.model'  
import { ClientModel } from './client.model'  
import { CourierModel } from './courier.model'  
import { StatusModel } from './status.model'  
  
export const OrderModel = sequelize.define<any, any>(TableNames.orders, {  
 [OrderSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 allowNull: false,  
 unique: true,  
 },  
 [OrderSchema.pizzaId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: PizzaModel,  
 key: PizzaSchema.id,  
 },  
 },  
 [OrderSchema.clientId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: ClientModel,  
 key: ClientSchema.id,  
 },  
 },  
 [OrderSchema.courierId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: CourierModel,  
 key: CourierSchema.id,  
 },  
 },  
 [OrderSchema.statusId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: StatusModel,  
 key: StatusSchema.id,  
 },  
 },  
 [OrderSchema.address]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [OrderSchema.startDate]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [OrderSchema.endDate]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.orders,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { PizzaSchema } from '../constants/schemas'  
  
export const PizzaModel = sequelize.define<any, any>(TableNames.pizzas, {  
 [PizzaSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [PizzaSchema.name]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [PizzaSchema.description]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [PizzaSchema.price]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 },  
 [PizzaSchema.imageUrl]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.pizzas,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { OrderSchema, ReportSchema } from '../constants/schemas'  
import { OrderModel } from './order.model'  
  
export const ReportModel = sequelize.define<any, any>(TableNames.reports, {  
 [ReportSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [ReportSchema.orderId]: {  
 type: Sequelize.INTEGER,  
 allowNull: false,  
 references: {  
 model: OrderModel,  
 key: OrderSchema.id,  
 },  
 unique: true,  
 },  
 [ReportSchema.date]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
 [ReportSchema.description]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.reports,  
 timestamps: false,  
})

import Sequelize from 'sequelize'  
import { sequelize } from '../dbConnection/dbConnection'  
import { TableNames } from '../constants/types'  
import { StatusSchema } from '../constants/schemas'  
  
export const StatusModel = sequelize.define<any, any>(TableNames.statuses, {  
 [StatusSchema.id]: {  
 type: Sequelize.INTEGER,  
 primaryKey: true,  
 autoIncrement: true,  
 unique: true,  
 allowNull: false,  
 },  
 [StatusSchema.type]: {  
 type: Sequelize.TEXT,  
 allowNull: false,  
 },  
}, {  
 tableName: TableNames.statuses,  
 timestamps: false,  
})

import express, { Request, Response } from 'express'  
import { ApiRouters, CLIENT\_ERROR\_STATUS\_CODE } from '../../constants/constants'  
import { messages } from '../../constants/messages'  
import { ModelWrapper } from '../../models/modelWrapper'  
  
export const errorHandler = (reason: object, errorType: string, res: Response) =>  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ error: {  
 header: messages.DATABASE\_ERROR,  
 type: errorType,  
 body: reason,  
 } })  
  
export const generateApiRouter = (  
 model: ModelWrapper,  
 identifierName: string) => {  
 const apiRouter = express.Router()  
  
 apiRouter.get(ApiRouters.getAll, (req: Request, res: Response) => {  
 model.findAll({})  
 .then(data => res.json(data))  
 .catch(reason => errorHandler(reason, messages.DATABASE\_ERROR\_GET\_ALL, res))  
 })  
  
 apiRouter.get(ApiRouters.getSingle, (req: Request, res: Response) => {  
 model.findOne({  
 where: { [identifierName]: +req.params.id },  
 }).then(data => res.json(data))  
 .catch(reason => errorHandler(reason, messages.DATABASE\_ERROR\_GET\_ONE, res))  
 })  
  
 apiRouter.post(ApiRouters.postSingle, (req: Request, res: Response) => {  
 model.create({ ...req.body })  
 .then(data => res.json(data))  
 .catch(reason => errorHandler(reason, messages.DATABASE\_ERROR\_ADD, res))  
 })  
  
 apiRouter.put(ApiRouters.putSingle, (req: Request, res: Response) => {  
 model.update({ ...req.body }, { where: { [identifierName]: +req.params.id } })  
 .then(data => res.json(data))  
 .catch(reason => errorHandler(reason, messages.DATABASE\_ERROR\_UPDATE, res))  
 })  
  
 apiRouter.delete(ApiRouters.deleteSingle, (req: Request, res: Response) => {  
 model.destroy({ where: { [identifierName]: +req.params.id } })  
 .then(data => res.json(data))  
 .catch(reason => errorHandler(reason, messages.DATABASE\_ERROR\_DELETE, res))  
 })  
  
 return apiRouter  
}

import express from 'express'  
import { TableRoutes } from '../../constants/types'  
import {  
 AccountSchema,  
 ClientSchema,  
 CommentSchema,  
 CourierSchema,  
 ManagerSchema,  
 OrderSchema,  
 PizzaSchema,  
 ReportSchema,  
 StatusSchema,  
} from '../../constants/schemas'  
import {  
 AccountModel,  
 ClientModel,  
 CommentModel,  
 CourierModel,  
 ManagerModel,  
 OrderModel,  
 PizzaModel,  
 ReportModel,  
 StatusModel,  
} from '../../models/models'  
import { generateApiRouter } from './generic.router'  
import { ModelWrapper } from '../../models/modelWrapper'  
  
type ModelData = {  
 model: ModelWrapper  
 identifierName: string  
 route: TableRoutes  
}  
  
const models: ModelData[] = [  
 { model: AccountModel, identifierName: AccountSchema.id, route: TableRoutes.accounts },  
 { model: ClientModel, identifierName: ClientSchema.id, route: TableRoutes.clients },  
 { model: CommentModel, identifierName: CommentSchema.id, route: TableRoutes.comments },  
 { model: CourierModel, identifierName: CourierSchema.id, route: TableRoutes.couriers },  
 { model: ManagerModel, identifierName: ManagerSchema.id, route: TableRoutes.managers },  
 { model: OrderModel, identifierName: OrderSchema.id, route: TableRoutes.orders },  
 { model: PizzaModel, identifierName: PizzaSchema.id, route: TableRoutes.pizzas },  
 { model: ReportModel, identifierName: ReportSchema.id, route: TableRoutes.reports },  
 { model: StatusModel, identifierName: StatusSchema.id, route: TableRoutes.statuses },  
]  
  
export const apiRouter = models.reduce((router: express.Router, {  
 model,  
 identifierName,  
 route }: ModelData) => router.use(route, generateApiRouter(model, identifierName)), express.Router())

import express from 'express'  
import { CLIENT\_ERROR\_STATUS\_CODE, RoutesPaths } from '../../../constants/constants'  
import { ClientModel, CourierModel, OrderModel, PizzaModel, ReportModel, StatusModel } from '../../../models/models'  
import { CourierDTO, OrderDTO, PizzaDTO, ReportDTO, StatusDTO } from '../../../constants/models'  
import { ClientSchema, OrderSchema } from '../../../constants/schemas'  
import { Op } from 'sequelize'  
  
export const clientMenuRouter = express.Router()  
  
clientMenuRouter.post(RoutesPaths.getOrdersByClientId, async (req: express.Request, res: express.Response) => {  
 try {  
 const filter = req.body.isFilterApplied ?  
 { [OrderSchema.statusId]: { [Op.eq]: 1 } }  
 : { [OrderSchema.statusId]: { [Op.not]: 1 } }  
 const filteredOrders: OrderDTO[] = await OrderModel.findAll({  
 where: {  
 [OrderSchema.clientId]: req.body.clientId,  
 ...filter,  
 },  
 })  
 const statuses: StatusDTO[] = await StatusModel.findAll({})  
 const pizzas: PizzaDTO[] = await PizzaModel.findAll({})  
 const couriers: CourierDTO[] = await CourierModel.findAll({})  
 const reports: ReportDTO[] = await ReportModel.findAll({})  
 const client: CourierDTO = await ClientModel.findOne({  
 where: {  
 [ClientSchema.id]: req.body.clientId,  
 },  
 })  
  
 const fullFilteredOrders = filteredOrders.map(order => ({  
 ...order,  
 pizza: pizzas.find(pizza => pizza.id === order.pizzaId),  
 status: statuses.find(status => status.id === order.statusId),  
 courier: couriers.find(courier => courier.id === order.courierId),  
 client: client,  
 report: reports.find(report => report.orderId === order.id),  
 }))  
 res.json(fullFilteredOrders)  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})  
  
clientMenuRouter.post(RoutesPaths.declineOrder, async (req: express.Request, res: express.Response) => {  
 try {  
 await OrderModel.update({  
 [OrderSchema.statusId]: 2,  
 }, {  
 where: {  
 id: req.body.id,  
 },  
 })  
 res.json([])  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})

import express from 'express'  
import { CLIENT\_ERROR\_STATUS\_CODE, RoutesPaths } from '../../../constants/constants'  
import { ClientModel, CourierModel, OrderModel, PizzaModel, ReportModel, StatusModel } from '../../../models/models'  
import { ClientDTO, CourierDTO, OrderDTO, PizzaDTO, ReportDTO, StatusDTO } from '../../../constants/models'  
import { CourierSchema, OrderSchema } from '../../../constants/schemas'  
import { Op } from 'sequelize'  
  
export const courierMenuRouter = express.Router()  
  
courierMenuRouter.post(RoutesPaths.getOrdersByCourierId, async (req: express.Request, res: express.Response) => {  
 try {  
 const filter = req.body.isFilterApplied ?  
 { [OrderSchema.statusId]: { [Op.eq]: 1 } }  
 : { [OrderSchema.statusId]: { [Op.not]: 1 } }  
 const filteredOrders: OrderDTO[] = await OrderModel.findAll({  
 where: {  
 [OrderSchema.courierId]: req.body.courierId,  
 ...filter,  
 },  
 })  
 const statuses: StatusDTO[] = await StatusModel.findAll({})  
 const pizzas: PizzaDTO[] = await PizzaModel.findAll({})  
 const clients: ClientDTO[] = await ClientModel.findAll({})  
 const reports: ReportDTO[] = await ReportModel.findAll({})  
 const courier: CourierDTO = await CourierModel.findOne({  
 where: {  
 [CourierSchema.id]: req.body.courierId,  
 },  
 })  
  
 const fullFilteredOrders = filteredOrders.map(order => ({  
 ...order,  
 pizza: pizzas.find(pizza => pizza.id === order.pizzaId),  
 status: statuses.find(status => status.id === order.statusId),  
 courier: courier,  
 client: clients.find(client => client.id === order.clientId),  
 report: reports.find(report => report.orderId === order.id),  
 }))  
 res.json(fullFilteredOrders)  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})  
  
courierMenuRouter.post(RoutesPaths.acceptOrder, async (req: express.Request, res: express.Response) => {  
 try {  
 await OrderModel.update({  
 [OrderSchema.statusId]: 3,  
 }, {  
 where: {  
 id: req.body.orderId,  
 },  
 })  
 await ReportModel.create({ ...req.body })  
 res.json([])  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})

import express from 'express'  
import { CLIENT\_ERROR\_STATUS\_CODE, RoutesPaths } from '../../../constants/constants'  
import { ClientDTO, CourierDTO, OrderDTO, PizzaDTO, ReportDTO, StatusDTO } from '../../../constants/models'  
import { ClientModel, CourierModel, OrderModel, PizzaModel, ReportModel, StatusModel } from '../../../models/models'  
import { CourierSchema } from '../../../constants/schemas'  
  
export const managerMenuRouter = express.Router()  
  
managerMenuRouter.get(RoutesPaths.getGroupedReports, async (req: express.Request, res: express.Response) => {  
 try {  
 const reports: ReportDTO[] = await ReportModel.findAll({})  
 const orders: OrderDTO[] = await OrderModel.findAll({})  
 const pizzas: PizzaDTO[] = await PizzaModel.findAll({})  
 const couriers: CourierDTO[] = await CourierModel.findAll({})  
 const statuses: StatusDTO[] = await StatusModel.findAll({})  
 const clients: ClientDTO[] = await ClientModel.findAll({})  
  
 const mapShortOrderToFull = (shortOrder: OrderDTO) => ({  
 ...shortOrder,  
 pizza: pizzas.find(pizza => pizza.id === shortOrder.pizzaId),  
 client: clients.find(client => client.id === shortOrder.clientId),  
 status: statuses.find(status => status.id === shortOrder.statusId),  
 })  
  
 const groupedReports = couriers.map((courier) => {  
 const filteredOrders = orders.filter(order => order.courierId === courier.id)  
 const filteredReports = reports.filter(report => filteredOrders.some(order => order.id === report.orderId))  
 .map(report => ({  
 ...report,  
 order: mapShortOrderToFull(orders.find(order => order.id === report.orderId)),  
 }))  
 return {  
 courier,  
 reports: filteredReports,  
 }  
 })  
  
 res.json(groupedReports)  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})  
  
managerMenuRouter.post(RoutesPaths.editCourierSalary, async (req: express.Request, res: express.Response) => {  
 try {  
 await CourierModel.update({  
 [CourierSchema.salary]: req.body.salary,  
 }, {  
 where: {  
 [CourierSchema.id]: req.body.courierId,  
 },  
 })  
 res.json([])  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})

import express from 'express'  
import { CLIENT\_ERROR\_STATUS\_CODE, RoutesPaths } from '../../constants/constants'  
import { CourierModel, OrderModel, PizzaModel, StatusModel } from '../../models/models'  
import { CourierDTO, OrderDTO, StatusDTO } from '../../constants/models'  
  
export const catalogRouter = express.Router()  
  
catalogRouter.get(RoutesPaths.root, async (req: express.Request, res: express.Response) => {  
 try {  
 const pizzas = await PizzaModel.findAll({})  
 res.json(pizzas)  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})  
  
catalogRouter.post(RoutesPaths.addOne, async (req: express.Request, res: express.Response) => {  
 try {  
 const couriers: CourierDTO[] = await CourierModel.findAll({})  
 const randomCourier = couriers[Math.floor(Math.random() \* couriers.length)]  
 const pendingStatus: StatusDTO = await StatusModel.findOne({ where: { id: 1 } })  
 const createdOrder: OrderDTO = await OrderModel.create({  
 pizzaId: req.body.pizzaId,  
 clientId: req.body.clientId,  
 courierId: randomCourier.id,  
 statusId: pendingStatus.id,  
 address: req.body.address,  
 startDate: req.body.startDate,  
 endDate: new Date(req.body.endDate).toUTCString(),  
 })  
 res.json(createdOrder)  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})

import express from 'express'  
import { CLIENT\_ERROR\_STATUS\_CODE, RoutesPaths } from '../../constants/constants'  
import { ClientModel, CommentModel } from '../../models/models'  
import { ClientDTO, CommentDTO } from '../../constants/models'  
  
export const commentsRouter = express.Router()  
  
commentsRouter.get(RoutesPaths.getAll, async (req: express.Request, res: express.Response) => {  
 try {  
 const comments: CommentDTO[] = await CommentModel.findAll({})  
 const clients: ClientDTO[] = await ClientModel.findAll({})  
 res.json(comments.map(comment => ({  
 ...comment,  
 clientName: clients.find((client) => client.id === comment.clientId).name,  
 })))  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})  
  
commentsRouter.post(RoutesPaths.addOne, async (req: express.Request, res: express.Response) => {  
 try {  
 const addedCommentData = req.body  
 const addedCommentDTO: CommentDTO = await CommentModel.create(addedCommentData)  
 const client: ClientDTO = await ClientModel.findOne({ where: { id: addedCommentDTO.clientId } })  
  
 res.json({  
 ...addedCommentDTO,  
 clientName: client.name,  
 })  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})  
  
commentsRouter.delete(RoutesPaths.deleteOne, async (req: express.Request, res: express.Response) => {  
 try {  
 const { id } = req.body  
 await CommentModel.destroy({ id })  
 const comments: CommentDTO[] = await CommentModel.findAll({})  
 const clients: ClientDTO[] = await ClientModel.findAll({})  
 res.json(comments.map(comment => ({  
 ...comment,  
 clientName: clients.find((client) => client.id === comment.clientId).name,  
 })))  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})

import express from 'express'  
import { RoutesPaths } from '../../constants/constants'  
import { signInRouter } from './signIn.router'  
import { signUpRouter } from './signUp.router'  
import { catalogRouter } from './catalog.router'  
import { commentsRouter } from './comments.router'  
import { clientMenuRouter } from './menu/clientMenu.router'  
import { courierMenuRouter } from './menu/courierMenu.router'  
import { managerMenuRouter } from './menu/managerMenu.router'  
  
export const othersRouter = express.Router()  
  
othersRouter.use(RoutesPaths.signIn, signInRouter)  
othersRouter.use(RoutesPaths.signUp, signUpRouter)  
othersRouter.use(RoutesPaths.catalog, catalogRouter)  
othersRouter.use(RoutesPaths.comments, commentsRouter)  
othersRouter.use(RoutesPaths.clientMenu, clientMenuRouter)  
othersRouter.use(RoutesPaths.courierMenu, courierMenuRouter)  
othersRouter.use(RoutesPaths.managerMenu, managerMenuRouter)

import express from 'express'  
import { AccountModel, ClientModel, CourierModel, ManagerModel } from '../../models/models'  
import { AccountSchema, ClientSchema, CourierSchema, ManagerSchema } from '../../constants/schemas'  
import stringHash from 'string-hash'  
import { AccountDTO, ClientDTO, CourierDTO, ManagerDTO } from '../../constants/models'  
import { messages } from '../../constants/messages'  
import { sendError } from '../../helpers/server.helper'  
import { RoutesPaths } from '../../constants/constants'  
  
export const signInRouter = express.Router()  
  
type SignInData = {  
 email: string  
 password: string  
}  
  
signInRouter.post(RoutesPaths.root, async (req: express.Request, res: express.Response) => {  
 try {  
 const signInData: SignInData = { ...req.body }  
 const account: AccountDTO | null = await AccountModel.findOne({  
 where: {  
 [AccountSchema.email]: signInData.email,  
 [AccountSchema.password]: stringHash(signInData.password).toString(),  
 },  
 })  
  
 if (!account) {  
 return sendError(messages.NO\_SUCH\_ACCOUNT\_FOUND, res)  
 }  
  
 const client: ClientDTO | null = await ClientModel.findOne({  
 where: {  
 [ClientSchema.accountId]: account.id,  
 },  
 })  
  
 if (client) {  
 return res.json({ account, client })  
 }  
  
 const courier: CourierDTO | null = await CourierModel.findOne({  
 where: {  
 [CourierSchema.accountId]: account.id,  
 },  
 })  
  
 if (courier) {  
 return res.json({ account, courier })  
 }  
  
 const manager: ManagerDTO | null = await ManagerModel.findOne({  
 where: {  
 [ManagerSchema.accountId]: account.id,  
 },  
 })  
  
 if (manager) {  
 return res.json({ account, manager })  
 }  
  
 console.log(account, client, courier, manager)  
  
 sendError(messages.NO\_SUCH\_USER\_FOUND, res)  
 } catch (e: any) {  
 sendError(messages.NO\_SUCH\_USER\_FOUND, res)  
 }  
})

import express from 'express'  
import { AccountDTO, ClientDTO } from '../../constants/models'  
import { AccountModel, ClientModel } from '../../models/models'  
import { AccountSchema, ClientSchema } from '../../constants/schemas'  
import stringHash from 'string-hash'  
import { CLIENT\_ERROR\_STATUS\_CODE, RoutesPaths } from '../../constants/constants'  
  
export const signUpRouter = express.Router()  
  
type SignUpData = {  
 name: string  
 phoneNumber: string  
 description: string  
 email: string  
 password: string  
}  
  
signUpRouter.post(RoutesPaths.root, async (req: express.Request, res: express.Response) => {  
 try {  
 const signUpData: SignUpData = { ...req.body }  
 const account: AccountDTO = await AccountModel.create({  
 [AccountSchema.email]: signUpData.email,  
 [AccountSchema.password]: stringHash(signUpData.password).toString(),  
 })  
  
 const client: ClientDTO = await ClientModel.create({  
 [ClientSchema.name]: signUpData.name,  
 [ClientSchema.accountId]: account.id,  
 [ClientSchema.phone]: signUpData.phoneNumber,  
 [ClientSchema.description]: signUpData.description,  
 })  
  
 res.json({ account, client })  
 } catch (e: any) {  
 res.status(CLIENT\_ERROR\_STATUS\_CODE).json({ message: e.message })  
 }  
})

import express from 'express'  
import cors from 'cors'  
import { apiRouter } from './routers/api/routers.api'  
import { messages } from './constants/messages'  
import {  
 ApiRouters,  
 PATH\_TO\_PUBLIC\_IMAGES,  
 RoutesPaths,  
 SERVER\_PORT,  
} from './constants/constants'  
import { generateAccountsJson } from './helpers/accounts.helper'  
import { othersRouter } from './routers/others/routers.others'  
  
const isGenerateAccountsJson = false  
if (isGenerateAccountsJson) generateAccountsJson()  
const app = express()  
app.use(cors())  
app.use(express.json())  
app.use(express.urlencoded({ extended: true }))  
app.use(`${RoutesPaths.static}${RoutesPaths.images}`, express.static(PATH\_TO\_PUBLIC\_IMAGES))  
app.use(ApiRouters.api, apiRouter)  
app.use(RoutesPaths.backend, othersRouter)  
app.listen(SERVER\_PORT, () => {  
 console.log(messages.SERVER\_LISTENING(SERVER\_PORT))  
})

import { *useState*, *useCallback* } from 'react'  
import { FetchRequest, RequestMethod } from '../constants/types'  
export const *useFetch* = () => {  
 const [loading, setLoading] = *useState*<boolean>(false)  
 const [error, setError] = *useState*<string>('')  
 const clearError = () => setError('')  
 const request = *useCallback*(async (  
 {  
 url,  
 method = RequestMethod.*get*,  
 body = null,  
 headers = {},  
 }: FetchRequest,  
 ) => {  
 const handleError = (message: string) => {  
 setLoading(false)  
 setError(message)  
 throw new *Error*(message)  
 }  
 setLoading(true)  
 clearError()  
 try {  
 const processedBody = body ? JSON.stringify(body) : null  
 const processedHeaders = body ? { ...headers, 'Content-Type': 'application/json' } : { ...headers }  
 const response = await *fetch*(url, { method, body: processedBody, headers: processedHeaders })  
 const data = await response.json()  
 if (!response.ok) {  
 return handleError(data?.message ?? 'Something went wrong (fetch-hook).')  
 }  
 setLoading(false)  
 return data  
 } catch (e: any) {  
 return handleError(e.message)  
 }  
 }, [])  
 return {  
 loading,  
 request,  
 error,  
 clearError,  
 }  
}