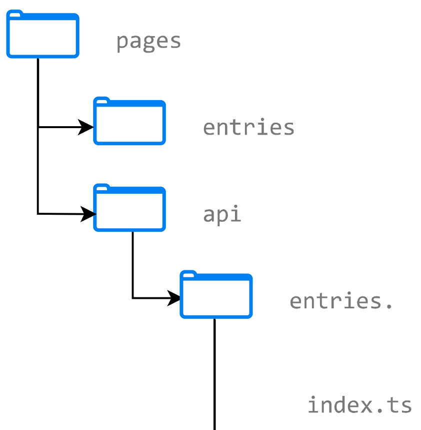

```
import type { NextApiRequest, NextApiResponse } from "next";
import { db } from "../../../database";
import { Entry, IEntry } from "../../../models";


type Data =
    | {message: string}
    | IEntry[]
    | IEntry

export default function handler(
    req: NextApiRequest,
    res: NextApiResponse<Data>
) {

    switch (req.method) {
        case 'GET':
            return getEntries(res);

        case 'POST':
            return postEnty(req, res)

        default:
            res.status(400).json({ message: 'Endpoint no existe' });
    }


}

const getEntries = async (res: NextApiResponse<Data>) => {

    await db.connect();

    const entries = await Entry.find().sort({createdAt: 'ascending'});

    await db.disconnect();

    res.status(200).json(entries)

}

const postEnty = async (req: NextApiRequest, res: NextApiResponse<Data>) => {

    const { description = ''} = req.body

    const newEntry = new Entry({
        description
        description,
        createdAt: Date.now(),
    })

    try {

        await db.connect();
        await newEntry.save();
        await db.disconnect();

        return res.status(201).json( newEntry );

    } catch (error) {
        await db.disconnect();
        console.log(error);

        return res.status(500).json(
            { message: 'Algo salió mal, revisar consola del servidor'});
    }

}
```

[id]

index.ts

```
import mongoose from "mongoose";
import type { NextApiRequest, NextApiResponse } from "next";
import { db } from "../../../../database";
import { Entry, IEntry } from "../../../../models";

type Data =
    | { message: string }
    | IEntry;

export default function handler(req: NextApiRequest,
    res: NextApiResponse<Data>) {

    const { id } = req.query;

    if (!mongoose.isValidObjectId(id)) {
        return res.status(400).json({ message: 'El id no es válido'})
    }

    switch (req.method) {
        case 'PUT':
            return updateEntry(req, res);

        case 'GET':
            return getEntry(req, res);

        default:
            res.status(400).json({ message: 'Método no existe '});

    }


}

const getEntry = async ( req:NextApiRequest, res: NextApiResponse ) => {
    const { id } = req.query;

    await db.connect();

    const entryInDB = await Entry.findById( id );

    await db.disconnect();

    if (!entryInDB) {
        return res.status(400).json(
            { message: 'No hay entrada con ese ID: ' + id })
    }

    return res.status(200).json(entryInDB)
}

const updateEntry = async( req:NextApiRequest,
    res: NextApiResponse<Data> ) => {

    const { id } = req.query;

    await db.connect();

    const entryToUpdate = await Entry.findById( id );

    if (!entryToUpdate) {
        await db.disconnect();
        return res.status(400).json(
            { message: 'No hay entrada con ese ID: ' + id })
    }

    const {
        description = entryToUpdate.description,
        status = entryToUpdate.status,
    } = req.body;

    try {
        const updateEntry = await Entry.findByIdAndUpdate(
            id, {description, status}, {runValidators: true, new:
true});

        // entryToUpdate.description = description;
        // entryToUpdate.status = status;
        // await entryToUpdate.save();

        await db.disconnect();
        res.status(200).json(updateEntry!)

    } catch (error) {
        await db.disconnect();
        res.status(400).json({ message: 'Bad request'})
    }


}
```