Deployment Guide

NestJs + ReactJS + TypeOrm

MyDevil Hosting

1. Introduction.

This guide is built on my own experience. I created it to help less experienced people (Here I am :) deploy applications on a real server. Remember that even a different version of the same package might cause errors.
Another thing worth mentioning is that I'm going to focus here on something that caused issues in my deployment process, which have never shown on localhost.
If you want to check the entire repository, everything is available on my GitHub
There are branches that You can easily go through and find more details that were made compared to the localhost app.
Frontend branches: <production> and <version_2.0>
Backend branches : <main> and <deploy>

https://github.com/PatrykKeska/company_management_app_backend_nest

https://github.com/PatrykKeska/company_management_app_front

Before the application deployment process, there will be a few chapters about the changes I made earlier.
Here is a breakdown :

1. Introduction.
2. Environment variables.
3. Database config file.
4. Paths.
5. React Router on the nginx server.
6. JWT Cookies.
7. Cors settings.
8. MySQL.
9. Build your projects.
10. Deployment Process.
11. Debugging process.

2. Environment variables.

There are a few different ways to manage environment variables, and it is up to you which method you choose. Here I'm going to show just a .ts file that we will put into the gitignore file. ATTENTION! Don't push this kind of file with data on GitHub.
Hostings like MyDevil, don't have a GUI to add environment variables. In that case, you need to use ssh. Read more: https://wiki.mydevil.net/Node.js
In my case, I've moved everything to a JS file.

You can't just drop a TypeScript file on your server.
Add it before the npm build command, and everything will be ready for you. All you need to do later directly from the server is edit variable values.
If you have all these values, you can do it now.

NestJS => src/secrectFile.ts

```
14
15  export const dbConnection = {
16      DB_CONNECTION: 'mysql',
17      DB_HOST: 'localhost',
18      DB_USERNAME: 'Here your MySQL user name',
19      DB_PASSWORD: 'Here your MySQL user password',
20      DB_DATABASE: 'Here your database name',
21  };
22
23  export const SALT = 'Your Salt for hash pwd function';
24
25  export const jwtConstants = {
26      secret: 'Your Secret',
27  };
28
```

NestJS => dist/secrectFile.js

```
secretFile.js ×
~/Library/Caches/com.binarynights.ForkLift2/#22/secretFile.js
1   Object.defineProperty(exports, "__esModule", { value: true });
2   exports.jwtConstants = exports.dbConnection = void 0;
3   exports.dbConnection = {
4       DB_CONNECTION: 'mysql',
5       DB_HOST: 'localhost',
6       DB_USERNAME: 'User',
7       DB_PASSWORD: 'Password',
8       DB_DATABASE: 'Db name',
9   };
10
11  export const SALT: 'SALT',
12
13  exports.jwtConstants = {
14      secret: 'Secret,
15  };
16  //# sourceMappingURL=secretFile.js.map
17
```

That is how the file looks after the compilation

3. Database config file.

It is worth mentioning at the beginning entities import looks like that :

```
entities: ['dist//**/**.entity{.ts,.js}'],
```

And this causes TypeOrm errors, but what is most interesting, TypeOrm was able to build the whole structure of the database but did not work on it.
After changing entities to importing each one separately, it starts working.

NestJS => src/config/dbConfig.ts

product-in-places.module.ts × product-in-places.service.ts × NotFoundExceptionFilter.filter.ts × dbConfig.ts ×

```
1  import { TypeOrmModule } from '@nestjs/typeorm';
2  import { dbConnection } from '../secretFIle';
3  import { User } from '../entities/user.entity';
4  import { Places } from '../entities/places.entity';
5  import { ProductInPlaces } from '../entities/product_in_places.entity';
6  import { Products } from '../entities/products.entity';
7  export default () =>
8    ({
9      type: dbConnection.DB_CONNECTION,
10     host: dbConnection.DB_HOST,
11     username: dbConnection.DB_USERNAME,
12     password: dbConnection.DB_PASSWORD,
13     database: dbConnection.DB_DATABASE,
14     entities: [User, Places, ProductInPlaces, Products],
15     bigNumberStrings: false,
16     logging: true,
17     synchronize: true,
18   } as TypeOrmModule);
19
```

all entities imported separately

In the final Production, version consider changing these options

NestJS => src/app.module.ts

```
@Module( metadata: {
  imports: [
    ConfigModule.forRoot( options: { isGlobal: true }),
    TypeOrmModule.forRoot(dbConfig()),
    AuthModule,
    UserModule,
    PlacesModule,
    ProductsModule,   PatrykKeska, 10/09/2022, 17:20 • * product module import
    ProductInPlacesModule,
    FileTransferModule,
    ServeStaticModule.forRoot( options: {
      rootPath: join(__dirname, 'public/product-photos/'),
      serveRoot: '/',
    }),
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Here is config file for db

Module to serve static files.
More about it in another chapter.

4. Paths.

My API endpoints locally look like this:
-localhost:3000/storage
-localhost:3000/storage/add-new
etc.
For the hosting server, I did add prefixes before each API endpoint.

```
https://yourdomain.com/api/storage
```

```
https://yourdomain.com/api/storage/add-new
```

You can use NestJs global prefix

```
const app = NestFactory.create(ApplicationModule);
app.setGlobalPrefix('api');
```

Or do it manually

```
NestJS =>  src/products/products.controller.ts
```

```
@Controller( prefix: 'api/products')
export class ProductsController {
  constructor(
    @Inject(forwardRef( fn: () => ProductsService))
    private productsService: ProductsService,
  ) {}

  @UseGuards(AuthGuard( type: 'jwt'))
  @Get( path: '/')
  async getAllAvailableProducts(): Promise<Products[]> {
    return await this.productsService.getAllAvailableProducts();
  }
```

After That, It is very important all your HTTP requests from React have to be changed.
The easiest way to achieve this is to create an api.ts file that contains variables with URLs.

Make sure if your site uses HTTPS instead of HTTP to include it in your URL other way you will get errors from cors

React => src/utils/api.ts

```
api.ts ×
export const fileApi = 'https://kendziior4.usermd.net/'

export const apiURL = 'https://kendziior4.usermd.net/api'
```

Now easily you can change a "basic" URL in every single request.

```
api.ts ×   getAllProducts.ts ×
import { apiURL } from '../../../utils/api'
import { SinglePlacesProductsTypes } from '../../../types/places_products.types'

export const getAllProducts = async () => {
  const response = await fetch( input: `${apiURL}/products`,  init: {
    credentials: 'include',
    headers: { 'Content-Type': 'application/json' },
  })
  return (await response.json()) as SinglePlacesProductsTypes[]
}
```

5. React Router on the nginx server.

MyDevil Hosting for nodeJs uses nginx server.
Because of that .htaccess file won't work like on apache to handle the problem with React Router refresh.
However, there is a way to create a config file for nginx read more:

https://stackoverflow.com/questions/43951720/react-router-and-nginx

I spoke with support, and unfortunately, For MyDevil hosting, it won't work. They say "Please handle this problem in NodeJS"

In NestJs, we can handle this problem with middleware or global filter, for example

src/filters/NotFoundExceptionFilter.filter.ts

```
duct-in-places.module.ts ×   product-in-places.service.ts ×   NotFoundExceptionFilter.filter.ts ×
import {
  ArgumentsHost,
  Catch,
  ExceptionFilter,
  HttpException,
  NotFoundException,
} from '@nestjs/common';
import * as path from 'path';
@Catch(NotFoundException)
export class NotFoundExceptionFilter implements ExceptionFilter {
  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse();
    response.sendFile(path.resolve( paths: '../public_nodejs/public/index.html'));
  }
}
```

NestJS =>. /src/main.ts

```
app.use(cookieParser());
app.useGlobalFilters(new NotFoundExceptionFilter());
await app.listen( port: 3001);
}
bootstrap();
```

That's it all. Right now when someone tries to refresh a page, NestJS will send an HTML file

6. JWT Cookies

When the app was running on localhost, the cookie was set a little bit different

src/auth/auth.service.ts

```
res.clearCookie( name: 'jwt',  options: {
  secure: false,
  domain: 'localhost',
  httpOnly: true,
});
return res.json( body: { logged: false, status: 200 });
```

If you are using a JWT auth combined with secure cookies, this is how it should look on production server.

```
53          return res
54            .cookie( name: 'jwt', token.accessToken,  options: {
55              secure: true,
56              domain: 'kendziior4.usermd.net',
57              httpOnly: true,
58            })
59            .json( body: { logged: true, status: 200 });
60        } catch (e) {
61          return res.json( body: { error: e.message, message: 'this is error' });
62        }
63      }
64
65      async logout(user: User, res: Response) {
66        try {
67          user.currentTokenId = null;
68          await user.save();
69          res.clearCookie( name: 'jwt',  options: {
70            secure: true,
71            domain: 'kendziior4.usermd.net',
72            httpOnly: true,
73          });
74          return res.json( body: { logged: false, status: 200 });
75        } catch (e) {
76          return res.json( body: { error: e.message });
77        }
78      }  PatrykKeska, 09/09/2022, 12:37 • * auth module added
79    }
```

7. Cors settings

src/main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as cookieParser from 'cookie-parser';
import { NotFoundExceptionFilter } from './filters/NotFoundExceptionFilter.filter';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors( options: {
    credentials: true,
    origin: 'kendziior4.usermd.net',
  });
  app.use(cookieParser());
  app.useGlobalFilters(new NotFoundExceptionFilter());
  await app.listen( port: 3001);
}
bootstrap();
```

8. MySQL

The website panel in the MySQL tab simply adds a new database and a new user.
All your data copy and paste to your environment variables.

In My case, I'm going to paste it to the

```
NestJS =>.   src/secrectFile.ts
```

before the compilation.

Because we are using TypeOrm, you don't have to import your database with the whole structure.
Only created database is necessary.
If your TypeOrm settings are different, then you have to read more about database migrations :

https://typeorm.io/migrations

9. Build your projects

In the directories of your local project, run these commands:

```
NestJs =>    npm run build
```

after that, your compiled code is in /dist

```
React =>     npm run build
```

after that, your compiled code is in /build

10. Deployment Process

This is very important, to get read all documentation for each server you want to use. Different servers might need a special folder structure, file names, etc.
For MyDevil hosting here are the docs

https://wiki.mydevil.net/Node.js

https://wiki.mydevil.net/React

Another thing worth mentioning that caused me errors is the folder structure after the build command. Remember, I'm describing my case so that It can be different from yours.
After building a project, folder and file structure are changed, and our folder structure on the production server differs from localhost. I had to update the paths (especially when you are handling static files)
These of course are examples, but it might be one of your issues too.
MyDevil Hosting :

```
export function storageDir() {
    return path.join(__dirname, '../public/product-photos');
}
```

Hosting

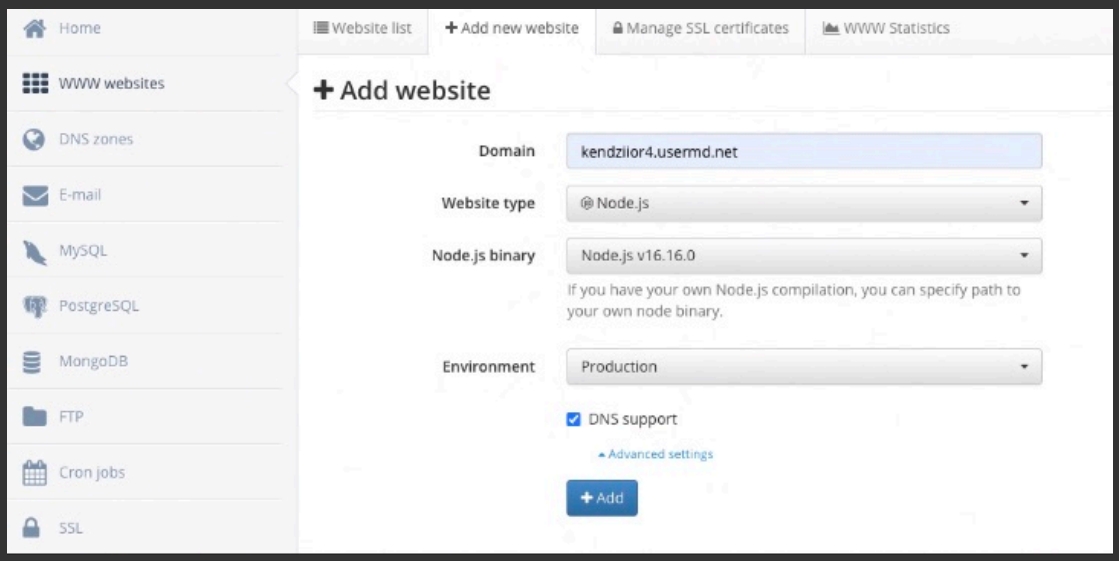
```
export function storageDir() {
    return path.join(__dirname, '../../storage');
}   PatrykKeska, 15/09/2022, 20:16 • * multer storage options
```

Localhost

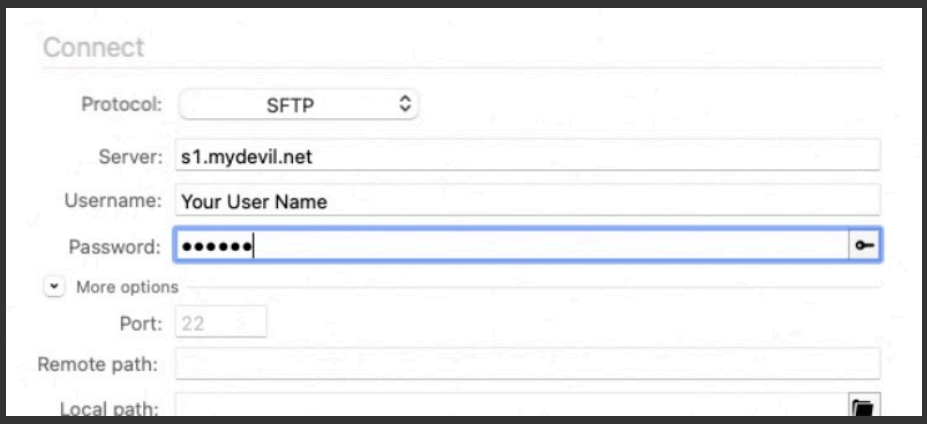
Let's start the deployment process.
Things which you need.
- SFTP application to transfer files (MyDevil offers a build-in file manager on their website after login)
- terminal (we are going to use ssh)

I'm using :
- macOS system
- iTerm
- ForkLift

When you log in to your hosting account, you can add a new website.
Then choose the NodeJS version and environment (Production/Development etc.)
Changes are possible later on the www panel or in ssh.



This is how to connect SFTP application :

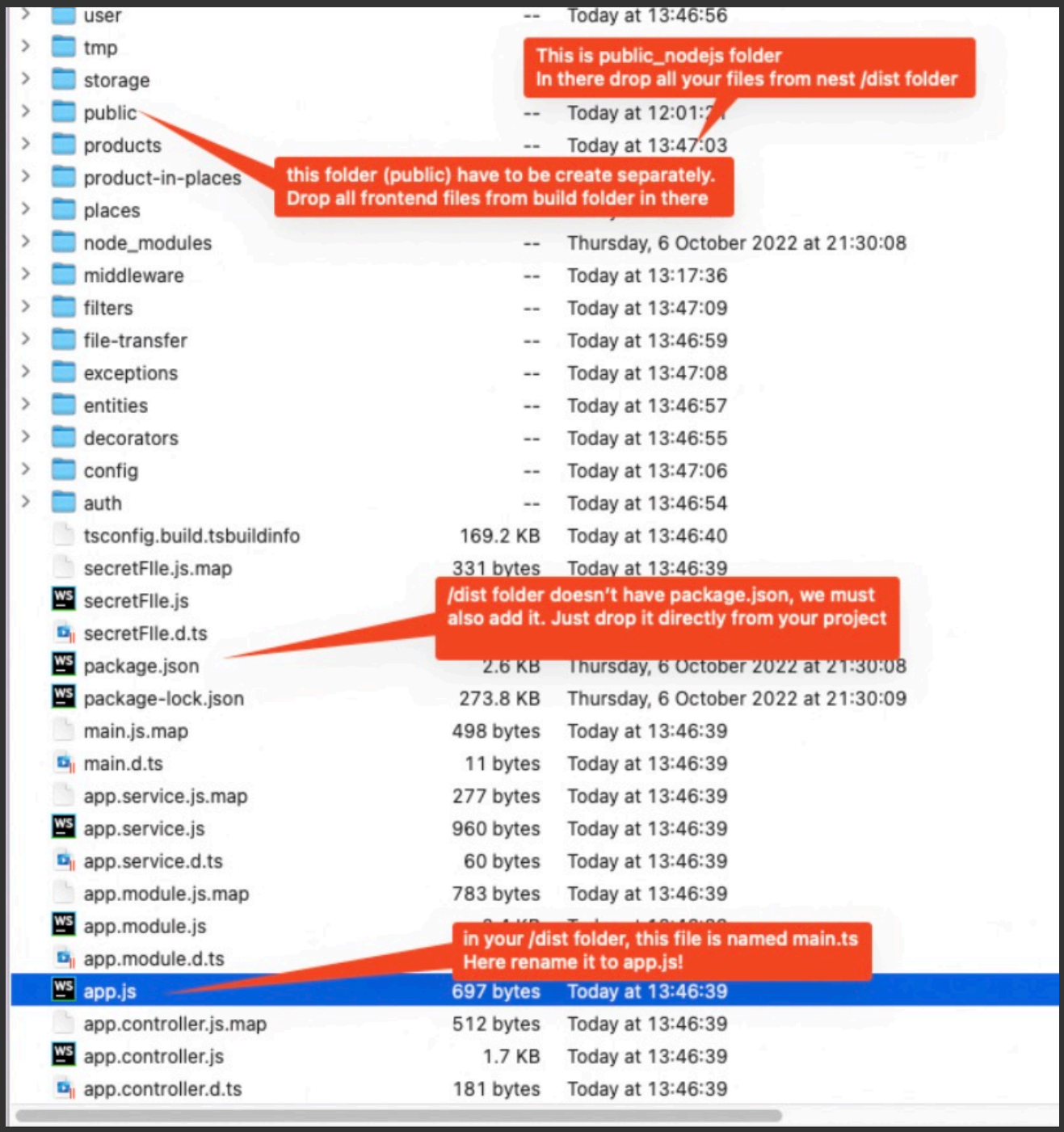


MyDevil hosting requires a specific folder structure to handle the NodeJS applications.
This is how it should look like :

```
domains/YourDomainName/public_nodejs/all files from Nest project /dist folder.
```

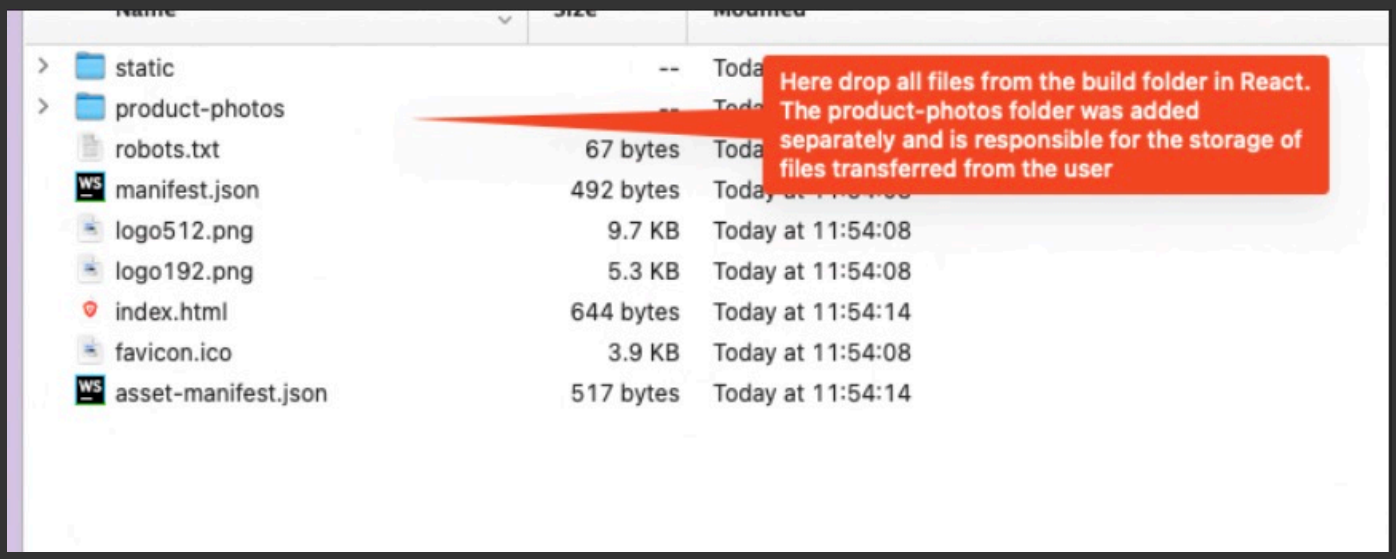
After the build command, NestJs /dist folder doesn't have package.json.
Don't forget to upload it in your public_nodejs/ folder.

After the build command in the /dist folder, there is a main.js file.
MyDevil hosting requires an app.js file to run your application!
Make Sure you rename the file name.



FrontEnd Files

```
domains/YourDomainName/public_nodejs/public/ your frontend files
```



CheckPoints

```
- [ ] Required Folder Structure Backend + Fronend
- [ ] Compiled files transferred => for NestJS dist/ all from here
- [ ] Compiled files transferred => for React build/ all from here
- [ ]  MyDevil hosting node app requires an app.js file to run ( NestJs created a main.js file, so don't forget to rename it, another way it won't work)
- [ ]   After the build command, NestJs /dist folder doesn't have package.json. Don't forget to include it in your public_nodejs/ folder.
```

If you did transfer all the files right, we would move to ssh.

Open a terminal and run this command :

```
ssh UserLogin@s20.mydevil.net
```

Your user login

Your server

than paste or type your password

Node app initial configuration is required by hosting.

Run these commands :

```
mkdir ~/.npm-global
```

```
npm config set prefix '~/.npm-global'
```

```
echo 'export PATH=~/.npm-global/bin:~/bin:$PATH ' >> $HOME/.bash_profile && source $HOME/.bash_profile
```

You can also change the NodeJS version and a couple more things. Read more about it here :

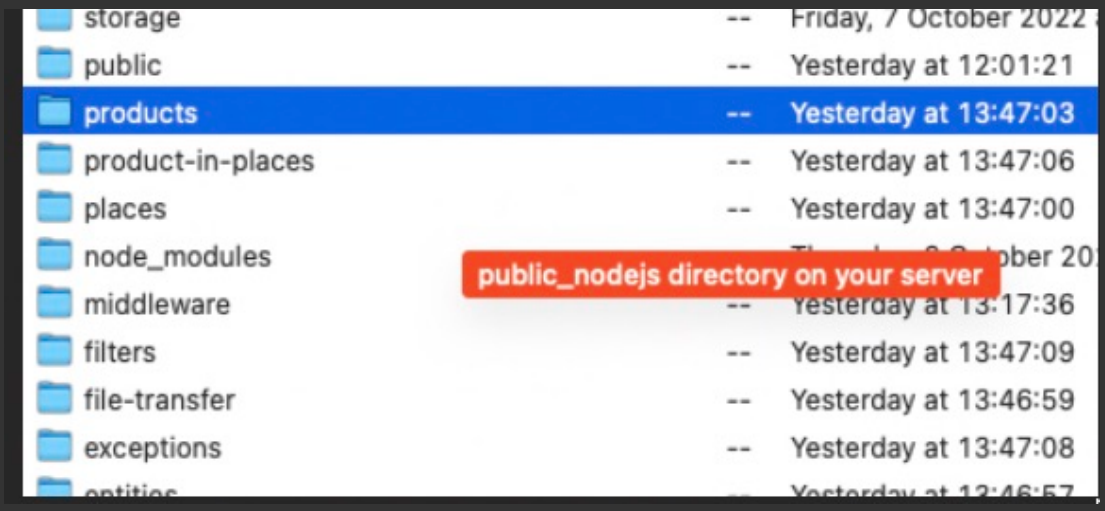
https://wiki.mydevil.net/Node.js

In the terminal, we can move like on any other UNIX system

```
cd/domains/yourDomain/public_nodejs/
```

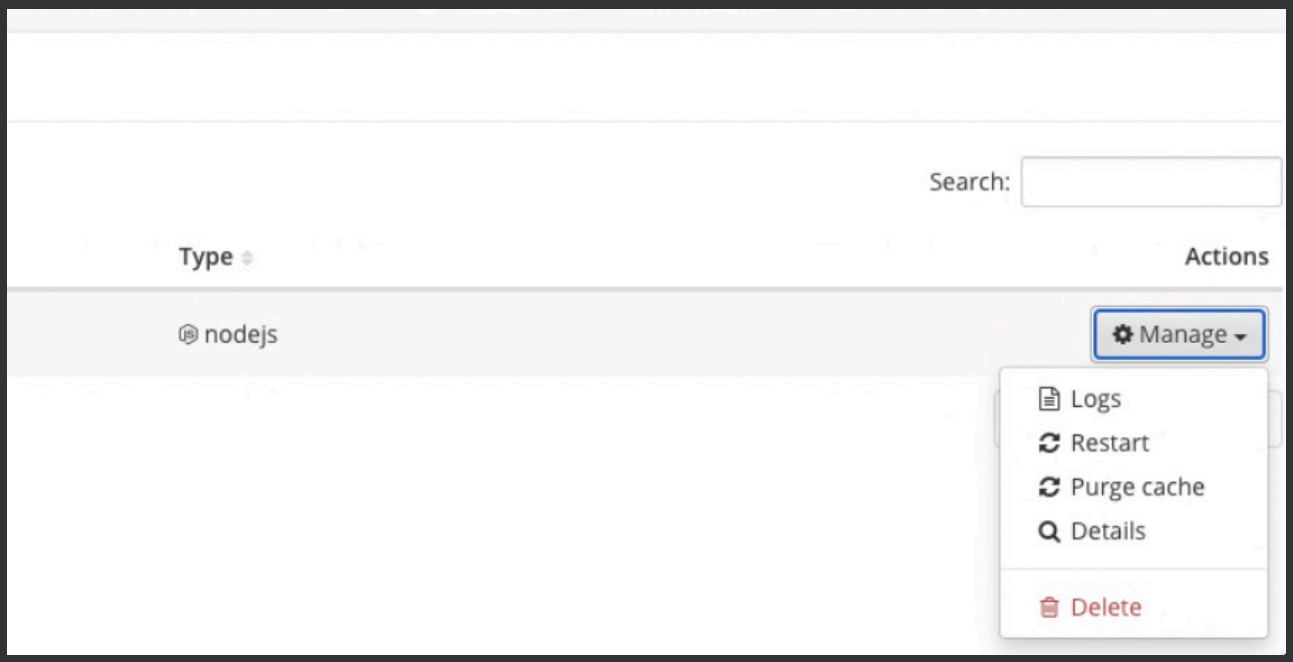
When we are in this directory run: `npm install`

It might take a while so be patient
When everything is ready node_modules folder will show up.
Depending on the SFTP client, some of them have to be refreshed after action to see the results



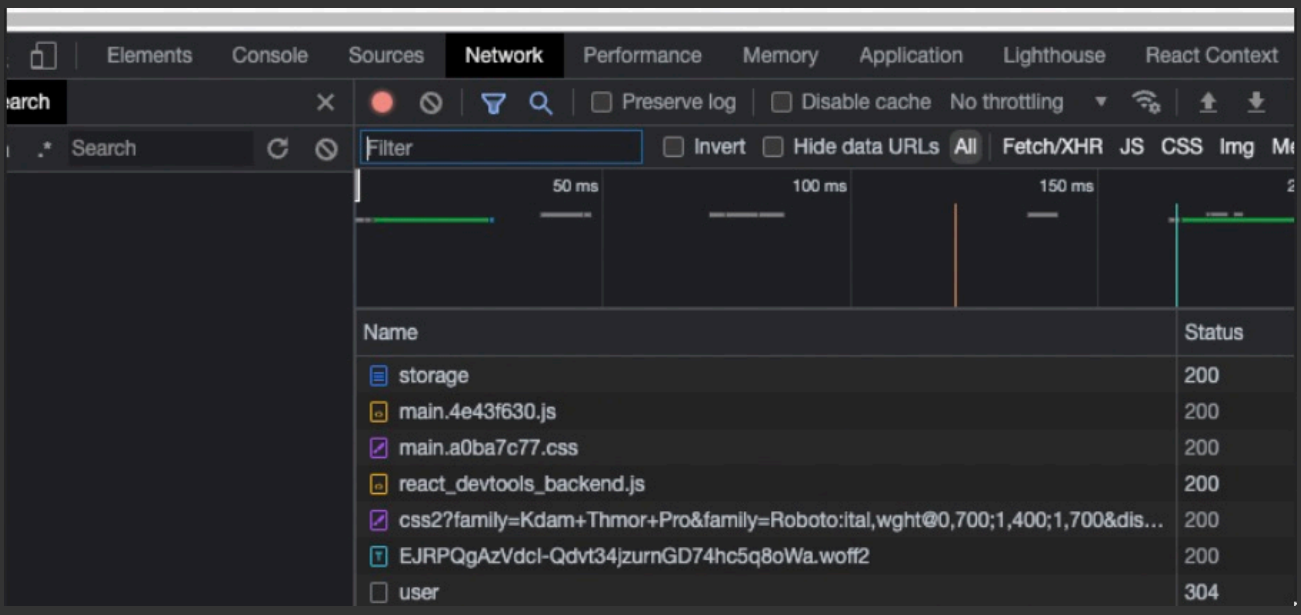
Make sure the environment which you chose(in the www panel) contains the right packages. If the environment is a production and some required packages are in dev dependencies, you might need to install them separately

After that, just move to the website panel => "www websites" tab and click manage => restart



```
each time you changed/transfer files just do a restart.

Your app should run. After restart, an HTTP request may take a few
seconds so be patient.
The good idea is to open the network tab on your browser to check what
is going on :)
```



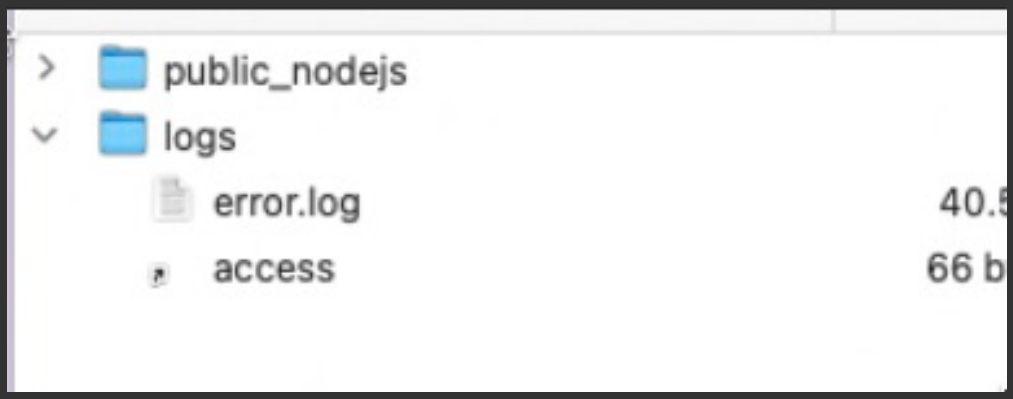
11. Debugging process

All errors which going to show are stored on your server in :

`domains/yourDomain/logs/error.log`

Don't ignore this point. This is your weapon to deal with errors and find out what went wrong.

Even when my app was working locally after moving in on the server, some npm packages had to be installed separately, like rxjs, mime, or multer. It happened, but I was able to find and fix it thanks to error.log
When something doesn't work, just go in there and fix it!



Maybe you accidentally made space while moving in the file? Thanks to logs, I was able to find it

```
7  [31m[Nest] 9870  - [39m09.10.2022, 13:48:45 [31m  ERROR [39m [38;5;3m[ExceptionsHand
8  TypeORMError: "product InPlaces" alias was not found. Maybe you forgot to join it?
```

```
return await this.dataSource
  .getRepository(ProductInPlaces) Repository<ProductInPlaces>    PatrykKeska, 14/09/2022, 16:24 • * new methods added :
  .createQueryBuilder( alias: 'productInPlaces') SelectQueryBuilder<ProductInPlaces>
  .leftJoinAndSelect( property: 'product InPlaces.places', alias: 'places') SelectQueryBuilder<ProductInPlaces>
  .leftJoinAndSelect( property: 'productInPlaces.products', alias: 'products') SelectQueryBuilder<ProductInPlaces>
  .where( where: 'places.id = :placeId', parameters: { placeId }) SelectQueryBuilder<ProductInPlaces>
  .andWhere( where: 'products.id = :productId', parameters: { productId }) SelectQueryBuilder<ProductInPlaces>
  .getOne();                                  Here is our Problem
}
```