Citec UBB practicas comunes al escribir codigo

#Programacion #TypeScript #NestJS #Express #NodeJS #MySQL #Sequelize

Programación

Declaración de Variables

- Se utiliza "const" antes que "let". Exceptuando que se requiera que la variable cambie en el futuro, aun asi, en ese caso se recomienda cambiar la logica.

```
✅ const miVariable = 12; //(Camel case)

❌ let miVariable = 12; //(Snake case)
```

- Se utiliza camel case para la declaración de variables

```
✅ const miVariable = 12; //(Camel case)

❌ const mi_variable = 12; //(Snake case)

❌ const MiVariable = 12; //(Pascal case)

❌ const mi-variable = 12; //(Kebab case)
```

- Se utilizan comillas simples para declarar variables

```
✅ const miVariable = 'hola mundo';

❌ const miVariable = "hola mundo"
```

El tipado de TypeScript

- Es recomendable utilizar la declaracion explicita solo en puntos criticos o donde no sea claro el retorno. Por ejemplo, en funciones.

```
function miFuncion (parametro: string): string {
    return parametro;
}
```

Declaracion de Funciones

- Se prefiere la sintaxis de funcion flecha y utilizacion de variables como funciones.
- Puede haber excepciones, especialmente en Backend

```
//Generalmente en Frontend se utiliza esta sintaxis
const miFuncion = (parametro: string): string =>{
    return parametro
}

//Generalmente en Backend por cuestiones del framework se utiliza esta
sintaxis
async crear(usuario: CrearUsuariosDto): Promise<Usuarios> {
        return usuarioCreado;
}
```

Uso de asincronia

- Para manejo de peticiones asincronas usamos async/await, no promesas

```
✅ const miFuncion = async (parametro: Usuario): Promise<Usuario> =>{
        const miUsuario = await
axios.get('https://localhost:4000/api/usuarios/crear');

    return miUsuario;
}

❌ miFuncion
  .then((valor) => {
    console.log(`Resultado positivo: (Resolve) ${valor}`);
  })
  .catch((error) => {
    console.error(`Resultado negativo (Reject): ${error}`);
  });
```

Base de datos

- Las nombres de las tablas se declaran en plural, en minusculas y en snake case.

```
✅ "tipos_de_usuarios"

❌ "Tipos de usuarios"
```

Recomendaciones Generales

- (Frontend) Usar las variables de entorno con precaución, a pesar de ser variables de entorno, igualmente se compilan publicamente al cliente.

```
FRONTEND_URL=http://localhost:3000
```

- Antes de ejecutar acciones correctas, la lógica debe validar primero los posibles errores. Por ello, se recomienda pensar en los casos negativos o aquellos que podrían provocar un error primero.

✅
```
if (!usuario) {
        throw new NotFoundException([
                `Usuario con email ${clavePrimaria.email} no encontrado`,
        ]);
}

return usuario;
```

❌
```
if (usuario) {
        return usuario;
}else{
        throw new NotFoundException([
                `Usuario con email ${clavePrimaria.email} no encontrado`,
        ]);
}
```

Recordar ejecutar en frontend y en backend

- Al inicio de proyecto se les entregaran archivos .zip, recordar instalar las librerias necesarias

```
npm run install
```

Frontend y Backend

- Ajusten las variables de entorno tanto del frontend como del backend segun los datos reales que usen.

Backend

Inicializar el Proyecto

```
$ npm install
```

```
# Es necesario este paso para ejecutar
npm install -g @nestjs/cli
```

Compilar y correr el proyecto

```
# Desarrollo

$ npm run start

# El mas usado. El que deben usar para desarrollo

$ npm run start:dev

# Para produccion

$ npm run start:prod
```

Correr Los Test (No es necesario si no los usan)

```
# Test unitarios

$ npm run test

# Test end to end (punto a punto)

$ npm run test:e2e

# Test de cobertura

$ npm run test:cov
```

Comandos basicos de nest

Documentacion cli de NestJS

- "--skip-git" Se salta git al momento de crear un proyecto

- "--flat" Para que no cree una carpeta adicional solo la que indicamos
- "--no-spec" Para no crear los test

```
# Crear proyecto (Si el proyecto ya esta creado no es necesario)

$ nest new . --skip-git

# Crear controladores

$ nest g co nombre_del_controlador carpeta/subcarpeta --flat --no-spec

# Crear servicios

$ nest g s nombre_del_servicio carpeta/subcarpeta --flat --no-spec

# Crear modulos

$ nest g mo nombre_del_modulo carpeta/subcarpeta --flat --no-spec
```