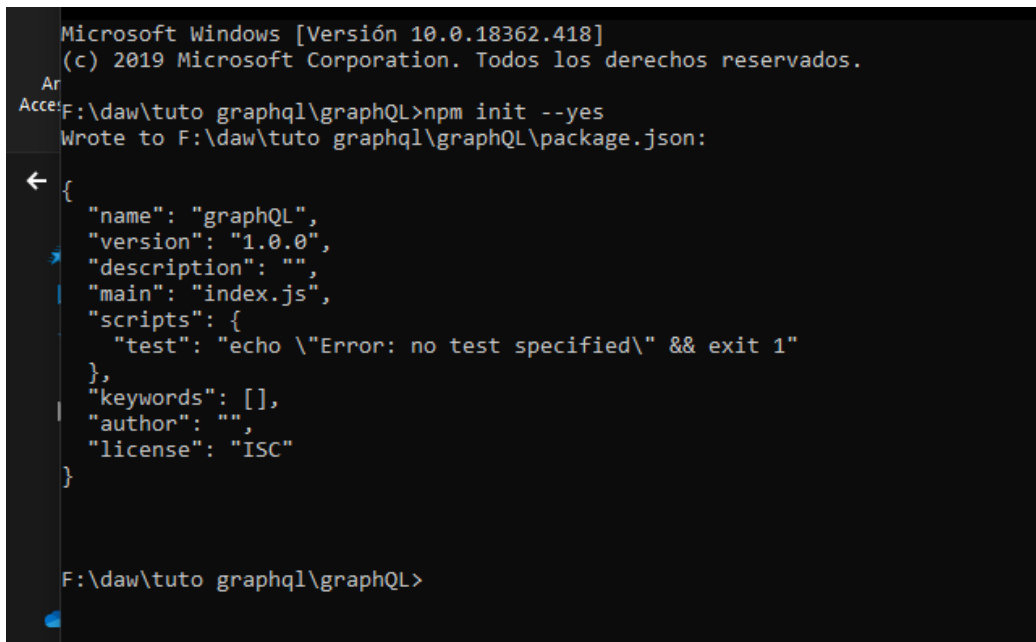


Requisitos para el siguiente tutorial

- ✓ Conocimiento en POO
- ✓ Conocimiento en JavaScript
- ✓ Conocimientos básicos en TypeScript
- ✓ Conocimientos básicos en Node JS
- ✓ Conocimiento en Rest-API o haber seguido el [tutorial](#).

PASOS A SEGUIR:

1. Se crea una carpeta para los archivos necesarios.
2. Se abre una consola en la ubicación del proyecto.
3. Se ejecuta el comando “npm init –yes”.



```
Microsoft Windows [Versión 10.0.18362.418]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

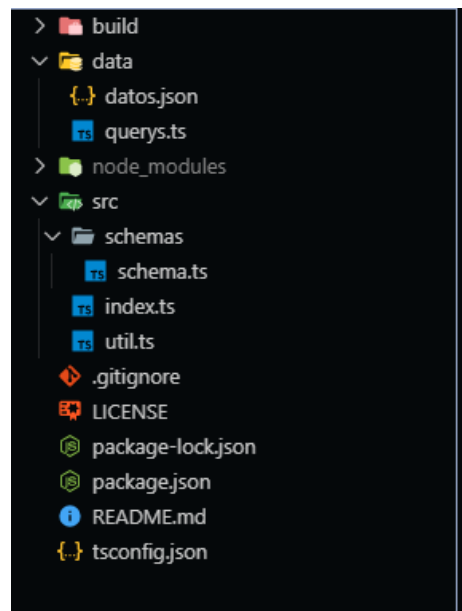
Ar
Acces F:\daw\tuto graphql\graphql>npm init --yes
Wrote to F:\daw\tuto graphql\graphql\package.json:

{
  "name": "graphql",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

F:\daw\tuto graphql\graphql>
```

4. Abrir el proyecto con su editor de código favorito. Para este ejemplo se usará visual studio code.

5. Se procede a crear la siguiente estructura:



- ✓ La carpeta build es donde se situará el código transpilado para que Node Js pueda correr el servidor.
 - ✓ En la carpeta src se procederá a crear toda la estructura del proyecto con código TypeScript.
 - ✓ La carpeta schemas contiene los Schemas del sistema.
 - ✓ La carpeta data es donde estarán los JSON que usara el sistema y los Query de prueba.
 - ✓ El archivo index.ts es el archivo de arranque de nuestro servidor, aquí se ubicará toda la lógica del sistema.
6. Luego, se procede a configurar el archivo tsconfig.json, que es el archivo de configuración del proceso de transpilación. No olvidar que para dar paso a la creación de dicho archivo debemos tener instalado TypeScript 1 y usar el comando “tsc –init”.
7. A continuación, se configura el archivo tsconfig.json los siguiente.

```
8. "target": "es6",  
9. "outDir": "./build",
```

¹ [Instalación de TypeScript](#)

8. Luego de esto, se instala express para crear el servidor con el comando “npm i express” y como dependencia de desarrollo “@types/express”.

```
npm i express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN api-rest@1.0.0 No description
+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 25.737s
found 0 vulnerabilities
```

```
npm i @types/express -D
npm WARN api-rest@1.0.0 No description
+ @types/express@4.17.1
added 8 packages from 57 contributors and audited 139 packages in 6.731s
found 0 vulnerabilities
```

9. Después de esto se procederá a crear el archivo schema.ts dentro de la carpeta schemas para poder configurar el sistema, el schema es la medula de un servidor de graphql aquí es donde se describe cómo se van a mostrar los datos y los tipos de datos que acepta y devuelve el sistema.

```
schema.ts - graphql-DAW - V
schemas > schemas.ts > ...
const { buildSchema } = require('graphql');
// GraphQL Schema
const schema = buildSchema(`
  type Query {
    alumno(id: Int!): Alumno
    alumnos(nota: Int!): [Alumno]
  }

  type Mutation {
    updateAlumnoGrade(id: Int!, nota: Int!): Alumno
  }

  type Alumno {
    Nombre : String
    Universidad : String
    Email : String
    Nota : Int
  }
`);
export default schema;
```

10. Una vez creado el schema es hora de crear las funciones que van a estar asociadas a cada Query o mutación que tenga configurado el schema estas funciones se las crea en el archivo util.ts.



```
Terminal  Ayuda  utils - graphql-DAW - Visual Studio Code
utils.ts
src > utils > ...
1  const { alumnos } = require("../data/datos.json");
2  class Util {
3    getAlumno = (args: any) => {
4      let id = args.id;
5      return alumnos.filter((alumno: any) => {
6        return alumno.ID == id;
7      })[0];
8    }
9
10   getAlumnos = (args: any) => {
11     if (args.nota) {
12       let nota = args.nota;
13       return alumnos.filter((alumno: any) => alumno.Nota === nota);
14     } else {
15       return alumnos;
16     }
17   }
18
19   updateAlumnoGrade = ({ id, nota }: any) => {
20     alumnos.map((alumno: any) => {
21       if (alumno.ID === id) {
22         alumno.Nota = nota;
23         return alumno;
24       }
25     });
26     return this.getAlumno(id);
27   }
28 }
29
30 export default new Util();
```

11. Es hora de crear el servidor, un servidor de graphql requiere de 3 módulos únicamente para poder funcionar de manera correcta: express, graphql y express-graphql para la comunicación del servidor con graphql y opcionalmente Morgan para poder visualizar rutas de acceso si es deseado por el desarrollador.
12. Realización de los imports

```
import express, { Application } from "express";
import morgan from "morgan";
const express_graphql = require('express-graphql');
```

13. luego de esto se importa los schemas que se hayan creado previamente para trabajar con nuestro servidor y el archivo útil que es donde están creadas las funciones a usar por la estructura del schema.

```
//import schemas
import schema from "./schemas/schema";
// import util
import util from "./util";
```

14. se procede a crear el servidor para poder hacer el match con todas las partes del Sistema, se debe definir la clase "server" de la siguiente manera.



```
Terminal  Ayuda  index.ts - graphql-DAW - Visual Studio Code
index.ts x
src > index.ts > ...
1  import express, { Application } from "express";
2  import morgan from "morgan";
3  const express_graphql = require('express-graphql');
4  //import schemas
5  import schema from "./schemas/schema";
6  // import util
7  import util from "./util";
8  class Server {
9      private app: Application;
10     private root = {
11         alumno: util.getAlumno,
12         alumnos: util.getAlumnos,
13         updateAlumnoGrade: util.updateAlumnoGrade
14     };
15     constructor() {
16         this.app = express();
17         this.config();
18     }
19
20     config(): void {
21         this.app.set("port", process.env.PORT || 3000);
22         this.app.use(morgan("dev"));
23         this.app.use('/graphql', express_graphql({
24             schema: schema,
25             rootValue: this.root,
26             graphiql: true
27         }));
28     }
29
30     start(): void {
31         this.app.listen(this.app.get("port"), () => {
32             console.log("server on port: ", this.app.get("port"));
33         });
34     }
35 }
36 const server = new Server();
37 server.start();
```

- ✓ Donde la variable root, representa la configuración de los posibles Query y que funciones responden a esos Query.
 - ✓ Dentro del método config(), la única diferencia con respecto a la configuración realizada para la Rest api es la configuración de graphql, la opción schema es el schema donde tendrá la base para el procesamiento de datos de los clientes, el rootValue es la opción en donde se va asignar la configuración JSON con los métodos que atienden a las peticiones de datos del usuario desde los clientes para graphql y graphiql debe estar en true si se quiere tener una interfaz gráfica para el Testing del servidor de graphql.
15. Una vez configurado el servidor se lo procede a arrancar y hacer las consultas de prueba mediante el uso de la interfaz gráfica accediendo a <https://localhost:300/graphql>