TP3 : Système de gestion de tâches avec GraphQL	**A.U. : 2024/2025**
Matière : SoA et Microservices **Enseignant :** Dr. Salah Gontara **Classe :** 4Info	

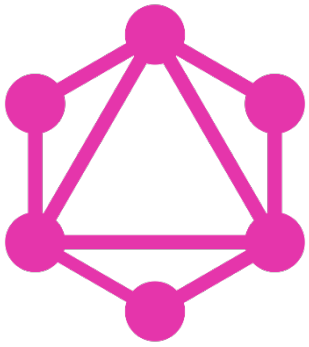
OBJECTIF(S):

- Comprendre comment configurer et utiliser GraphQL avec Node.js et Express.
- Apprendre à créer un schéma GraphQL et des résolveurs pour gérer les requêtes et les mutations pour une API simple de gestion de tâches.

OUTILS UTILISÉS :

node, graphql, express, appollo

Utilisation de GraphQL



GraphQL est un langage de requête pour les API et un runtime pour répondre à ces requêtes avec vos données existantes. GraphQL fournit une description complète et compréhensible des données de votre API, donne aux clients le pouvoir de demander exactement ce dont ils ont besoin et rien de plus, facilite l'évolution des API au fil du temps et active de puissants outils de développement.

1. Installer NodeJS:
    - NodeJS : Installez Nodejs depuis le site officiel
      https://nodejs.org/en/download
      Sur ubuntu : **sudo snap install node --classic**
2. Créez un répertoire pour l'exemple de projet **tp-graphql** et accédez-y :
   mkdir tp-graphql
   cd tp-graphql
   npm init -y
3. Installez les dépendances de javascript pour graphql et le projet:
   npm install express @apollo/server body-parser @graphql-tools/schema graphql

4. Créez un fichier « **taskSchema.gql** » contenant le schéma de données pour GraphQL :

```
type Task {
  id: ID!
  title: String!
  description: String!
  completed: Boolean!
}

type Query {
  task(id: ID!): Task
  tasks: [Task]
}

type Mutation {
  addTask(title: String!, description: String!, completed: Boolean!): Task
  completeTask(id: ID!): Task
}
```

5. Créez un fichier « **taskSchema.js** » qui va importer le schéma de données pour GraphQL.

```
const fs = require('fs');
const path = require('path');
const { buildSchema } = require('graphql');
const { promisify } = require('util');
const readFileAsync = promisify(fs.readFile);

async function getTaskSchema() {
  const schemaPath = path.join(__dirname, 'taskSchema.gql');
  try {
    const schemaString = await readFileAsync(schemaPath, { encoding: 'utf8'
});
    return buildSchema(schemaString);
  } catch (error) {
    console.error("Error reading the schema file:", error);
    throw error;
  }
}

module.exports = getTaskSchema();
```

6. Créez un fichier « **taskResolver.js** » contenant le résolveur de données pour GraphQL :

```
// taskResolver.js

let tasks = [
  {
    id: '1',
    title: 'Développement Front-end pour Site E-commerce',
    description: 'Créer une interface utilisateur réactive en utilisant React et Redux
pour un site e-commerce.',
    completed: false,
  },
  {
    id: '2',
    title: 'Développement Back-end pour Authentification Utilisateur',
    description: "Implémenter un système d'authentification et d'autorisation pour une
application web en utilisant Node.js, Express, et Passport.js",
    completed: false,
  },
  {
    id: '3',
    title: 'Tests et Assurance Qualité pour Application Web',
    description: 'Développer et exécuter des plans de test et des cas de test
complets.',
    completed: false,
  },
];

const taskResolver = {
  Query: {
    task: (_, { id }) => tasks.find(task => task.id === id),
    tasks: () => tasks,
  },
  Mutation: {
    addTask: (_, { title, description, completed }) => {
      const task = {
        id: String(tasks.length + 1),
        title,
        description,
        completed,
      };
      tasks.push(task);
      return task;
    },
    completeTask: (_, { id }) => {
      const taskIndex = tasks.findIndex(task => task.id === id);
      if (taskIndex !== -1) {
        tasks[taskIndex].completed = true;
        return tasks[taskIndex];
      }
      return null;
    },
  },
};

module.exports = taskResolver;
```

7. Créer un fichier **index.js** et y configurer une instance d'express avec une route pour gérer les requêtes GraphQL :

```
const express = require('express');
const { ApolloServer } = require('@apollo/server');
const { expressMiddleware } = require('@apollo/server/express4');
const { json } = require('body-parser');
const { addResolversToSchema } = require('@graphql-tools/schema');
const taskSchemaPromise = require('./taskSchema');
const taskResolver = require('./taskResolver');

const app = express();

async function setupServer() {
  try {
    const taskSchema = await taskSchemaPromise;
    const schemaWithResolvers = addResolversToSchema({
      schema: taskSchema,
      resolvers: taskResolver,
    });

    const server = new ApolloServer({
      schema: schemaWithResolvers,
    });

    await server.start();

    app.use(
      '/graphql',
      json(),
      expressMiddleware(server)
    );

    const PORT = process.env.PORT || 5000;
    app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
  } catch (error) {
    console.error('Failed to start the Apollo server:', error);
  }
}

setupServer();
```

8. Démarrer le serveur graphql avec :

   node index.js

9. Naviguer vers pour ouvrir l'interface graphique **Studio Sandobox** d'Appollo : http://localhost:5000/graphql

10. Créer des requêtes GraphQL pour
    - Récupérer toutes les tâches
    - Ajouter une nouvelle tâche
    - Marquer une tâche comme terminée

11. Ajouter au schéma de données une variable **duration** de type entier et apporter les changements nécessaires.

12. Ajouter une mutation **changeDescription** pour changer la description de la tâche au schéma et au résolveur.

13. Ajouter également une mutation **deleteTask** pour effacer une tâche par son id.