# Graphql Précis et concis

Graphql Client
- Apollo client
- Relay

Graphql Server (schema et resolver functions)
- Apollo Server
- « express-graphql »
- Graphql yoga

Types :

- Int
- Float
- String
- Boolean
- ID (unique identifier)

## Avec express-graphql

https://graphql.org/

- « Query » pour obtenir des données
- « mutation » pour l'ajout/modification de données (post, put, patch, delete data)
- Subscription: real time data

Toutes les reqêtes se font avec la méthode « POST »

### Côté serveur

```
npm i graphql express-graphql
```

Exemple

```
const { buildSchema } = require("graphql");

module.exports = buildSchema(`
    type HelloType {
        text: String!
        views: Int!
    }

    type RootQuery {
        hello: HelloType
    }

    type User {
        _id: ID!
```

Toutes les queries disponibles

```
        name: String!
        email: String!
        password: String
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    type RootMutation {
        createUser(userInput: UserInputData): User!
    }

    schema {
        query: RootQuery
        mutation: RootMutation
    }
`);
```

Toutes les mutations disponibles

resolvers.js toutes les functions

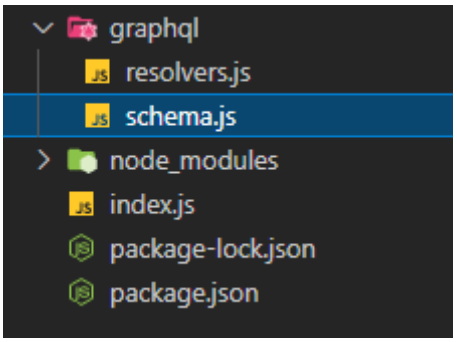
```
const bcrypt = require('bcryptjs');
const User = require('../models/user');

module.exports = {
  hello() {
    return {
      text: "Hello World!",
      views: 12345,
    };
  },
  createUser: async ({ userInput }, req) => {
    // validation ..

    const existingUser = await User.findOne({ email: userInput.email });
    if (existingUser) {
      const error = new Error("User exists already!");
      throw error;
    }
    const hashedPw = await bcrypt.hash(userInput.password, 12);
    const user = new User({
      email: userInput.email,
      name: userInput.name,
      password: hashedPw,
    });
    const createdUser = await user.save();
    return { ...createdUser._doc, _id: createdUser._id.toString() };
```

```
  },
};
```

index.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");

const { graphqlHTTP } = require("express-graphql");
const graphqlSchema = require("./graphql/schema");
const graphqlResolver = require("./graphql/resolvers");

const app = express();

app.use(cors());

app.use(
  "/graphql",
  graphqlHTTP({
    schema: graphqlSchema,
    rootValue: graphqlResolver,
    graphiql: true,
  })
);

mongoose
  .connect("mongodb://jerome:secret@127.0.0.1:27017/sample?authSource=admin")
  .then((result) => {
    app.listen(5000, () => {
      console.log("Server is running");
    });
  })
  .catch((err) => console.log(err));
```

Structure

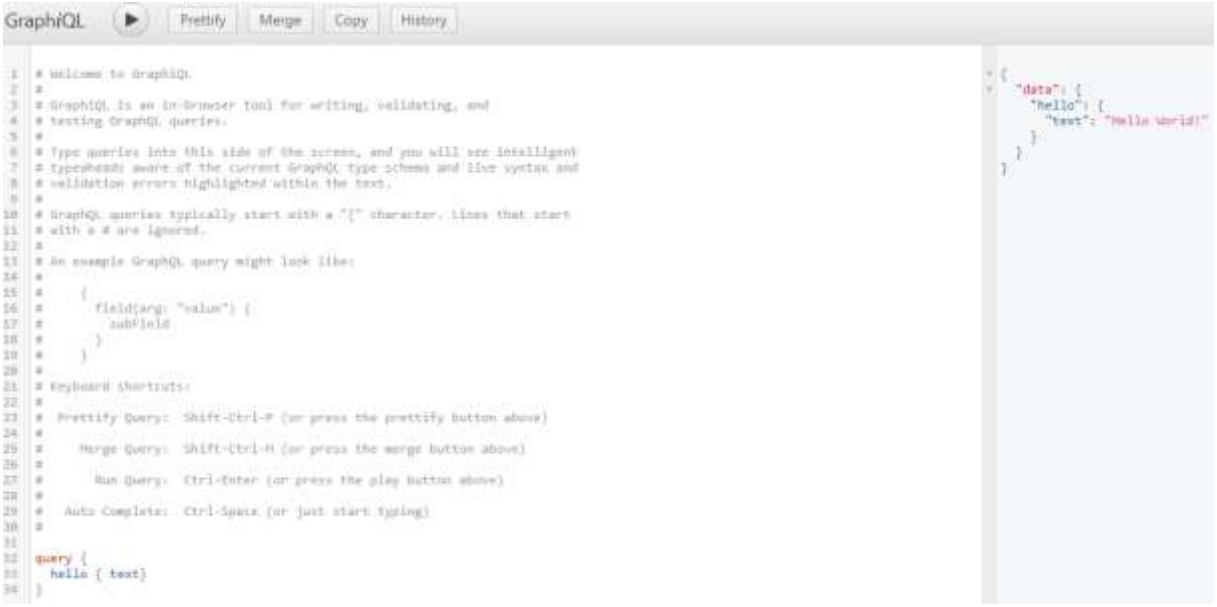



## graphiql

En activant graphiql on a une page dédiée pour les requests en ouvrant « http://localhost:5000/graphql » par exemple

```
app.use(
  "/graphql",
  graphqlHTTP({
    schema: graphqlSchema,
    rootValue: graphqlResolver,
    graphiql: true,
  })
);
```

Permet de saisir ses queries et afficher le résultat




## Coté client (React)

Pour injecter les variables

```
import React from "react";

export default function App() {
  const handleClick = () => {
    const graphqlQuery = {
      query: `
        mutation CreateUser($email: String!, $name: String!, $password: String!) {
          createUser(userInput: {email: $email, name: $name, password: $password}) {
            _id
            email
          }
        }
      `,
      variables: {
        email: "test@email.com",
        name: "test",
        password: "secret"
      }
    };

    fetch("http://localhost:5000/graphql", {
      method: "POST",
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify(graphqlQuery)
    })
```

```
      .then(res => {
        return res.json();
      })
      .then(responseData => {
        console.log(responseData);
      })
      .catch(err => {
        console.log(err);
      });
  };

  return (
    <>
      <h1>Sample</h1>
      <button onClick={handleClick}>Create user</button>
    </>
  );
}
```

# Avec Apopllo Server

## Côté server

Installation

```
npm i apollo-server graphql
```

Le studio permet de :

- Taper plusieurs queries, mutations et de ne sélectionner que les queries à exécuter
- On peut mettre en commentaire une query avec CTRL+ :
- On peut mettre en forme le code avec ALT+MAJ+F

Resolvers : parametres sont parent puis args puis context puis info

### Au plus simple (pas de datasources)

schema.js

```
const { gql } = require("apollo-server");

module.exports = gql`
  type Todo {
    id: ID
    text: String
    completed: Boolean
  }

  input TodoInput {
    id: ID
    text: String!
    completed: Boolean
```

```
  }

  type Query {
    todos: [Todo]
    todoById (id: ID): Todo
  }

  type Mutation {
    addTodo(input: TodoInput): Todo
    updateTodo(input: TodoInput): Todo
    deleteTodo(id: ID!): Todo
    toggleTodo(id: ID!): Todo
  }
`;
```

### resolvers.js

```
const { v4 } = require("uuid");

const todos = [{ id: "12345", text: "Test", completed: false }];

const resolvers = {
  Query: {
    todos: () => {
      return todos;
    },
    todoById: (parent, { id }) => {
      const todo = todos.find((x) => x.id == id);
      return todo;
    },
  },
  Mutation: {
    addTodo: (parent, { input }) => {
      const newTodo = {
        id: v4(),
        text: input.text,
        completed: input.completed || false,
      };
      todos.unshift(newTodo);
      return newTodo;
    },
    updateTodo: (parent, { input }) => {
      const todo = todos.find((x) => x.id === input.id);
      if (todo) {
        todo.text = input.text;
        todo.completed = input.completed;
      }
      return todo;
    },
    deleteTodo: (parent, { id }) => {
      const index = todos.findIndex((x) => x.id === id);
      if (index !== -1) {
        const deletedTodo = todos[index];
        todos.splice(index, 1);
        return deletedTodo;
      }
      return false;
    },
    toggleTodo: (parent, { id }) => {
      const todo = todos.find((x) => x.id === id);
      if (todo) {
        const completed = todo.completed;
        todo.completed = !completed;
      }
      return todo;
    },
  },
};

module.exports = resolvers;
```

index.js

```
const { ApolloServer } = require("apollo-server");

const port = process.env.PORT || 4000;

const typeDefs = require("./schema");

const resolvers = require("./resolvers");

const server = new ApolloServer({ typeDefs, resolvers });

server.listen({ port }).then(({ url }) => {
  console.log(`graphQL running at ${url}`);
});
```

Avec datasources

Queries

```
query TodoById($id: ID) {
  todoById(id: $id) {
    id
    text
    completed
  }
}

mutation AddTodo($input: TodoInput) {
  addTodo(input: $input) {
    id
    text
    completed
  }
}

mutation ToggleTodo($id: ID!) {
  toggleTodo(id: $id) {
    id
    text
    completed
  }
}

mutation UpdateTodo($input: TodoInput) {
  updateTodo(input: $input) {
    id
    text
```

```
    completed
  }
}

mutation DeleteTodo($id: ID!) {
  deleteTodo(id: $id) {
    id
    text
    completed
  }
}

query {
  todos {
    id
    text
    completed
  }
}
```

Variables

```
{
  "input": {
    "id": "1631288858802",
    "text": "updated",
    "completed": false
  }
}
```

Exemple de query

```
query {
  todos {
    id
    text
    completed
  }
}
```

On peut nommer les queries si plusieurs

```
query Todos {
  todos {
    id
    text
    completed
  }
}
```

```
{
  "input": {
    "text": "message 1",
    "completed": false
  }
}
```

Exemple de mutation

## Au plus simple

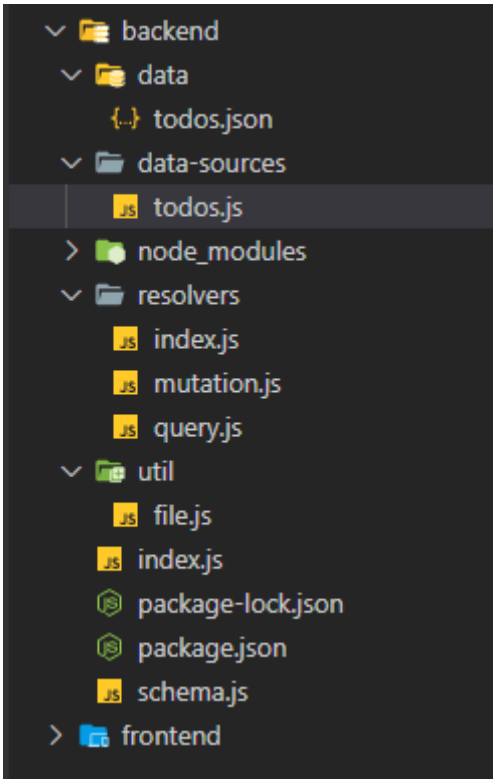
```
const { ApolloServer, gql } = require("apollo-server");
const sessions = require("./data/sessions.json");

// query
const typeDefs = gql`
  type Query {
    sessions: [Session]
  }
  type Session {
    id: ID!
    title: String!
    description: String
    startsAt: String
    endsAt: String
    room: String
    day: String
    format: String
    track: String
    level: String
  }
`;
// resolvers
const resolvers = {
  Query: {
    sessions: () => {
      return sessions;
    },
  },
};

const server = new ApolloServer({ typeDefs, resolvers });
```

```
server.listen({ port: process.env.PORT || 4000 }).then(({ url }) => {
  console.log(`graphQL running at ${url}`);
});
```

### Avec Datasources

https://www.apollographql.com/docs/apollo-server/data/data-sources/

| CLASS | SOURCE | FOR USE WITH |
|---|---|---|
| `RESTDataSource` | Apollo | REST APIs (see below) |
| `HTTPDataSource` | Community | HTTP/REST APIs (newer community alternative to `RESTDataSource`) |
| `SQLDataSource` | Community | SQL databases (via Knex.js) |
| `MongoDataSource` | Community | MongoDB |
| `CosmosDataSource` | Community | Azure Cosmos DB |
| `FirestoreDataSource` | Community | Cloud Firestore |



Mutation.js

```
module.exports = {
  createTodo: (parent, { todo }, { dataSources }, info) => {
    return dataSources.todoAPI.createTodo(todo);
  },
};
```

Query.js

```
module.exports = {
  todos: (parent, args, { dataSources }, info) => {
    return dataSources.todoAPI.getTodos();
  },
};
```

Resolvers/index.js

```
const Query = require("./query");
const Mutation = require("./mutation");

module.exports = {
  Query,
  Mutation,
};
```

Datasources

```
const { DataSource } = require("apollo-datasource");
const todos = require("../data/todos.json");
const { v4 } = require("uuid");
const _ = require("lodash");
const { writeJSON } = require("../util/file");


class TodoAPI extends DataSource {
  constructor() {
    super();
  }

  getTodos() {
    return todos;
  }

  createTodo(todo) {
    const id = v4();
```

```
    todo.id = id;
    todo.completed = false;
    todos.push(todo);

    writeJSON(todos);

    return todo;
  }
}

module.exports = TodoAPI;
```

Index.js

```
const { ApolloServer } = require("apollo-server");
const TodoAPI = require("./data-sources/todos");

const typeDefs = require("./schema");
const resolvers = require("./resolvers/index");
const dataSources = () => ({
  todoAPI: new TodoAPI(),
  // other
});

const server = new ApolloServer({ typeDefs, resolvers, dataSources });

server.listen({ port: process.env.PORT || 4000 }).then(({ url }) => {
  console.log(`graphQL running at ${url}`);
});
```

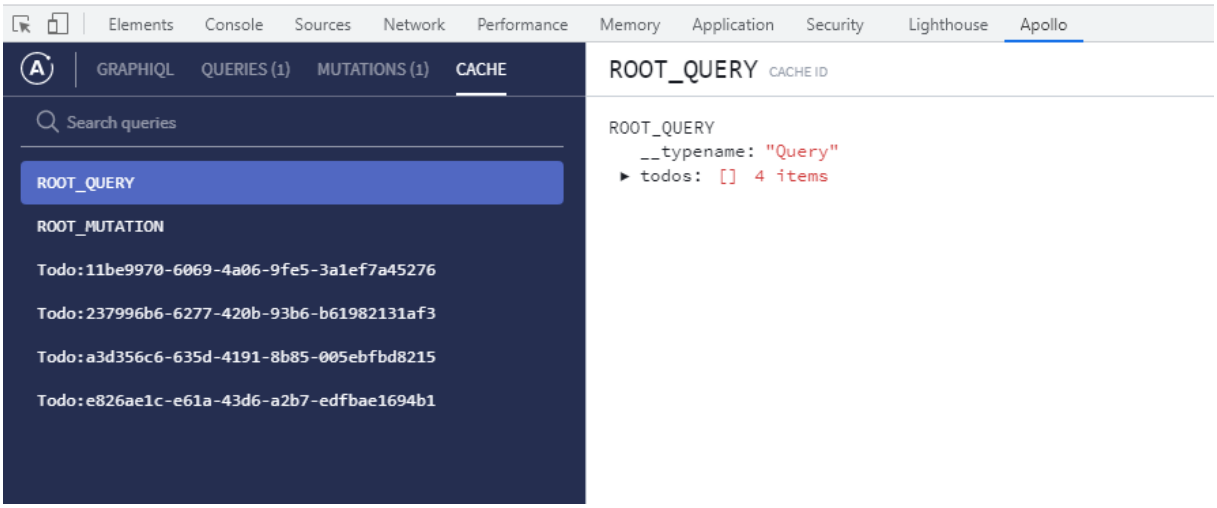
## Côté client

```
npm i @apollo/client graphql
```

## Extension Chrome

Apollo Client Devtools

Ajoute un onglet « Apollo ». On peut observer le cache, les queries, mutations et tester des queries




## Problème de rafraichissement

### *refetchQueries*

On définit les queries à exécuter après la mutation.

Soit en même temps que « useMutation »

```
  const [addTodo] = useMutation(ADD_TODO, {
    refetchQueries: [{ query: ALL_TODOS }]
  });
```

Soit à l'exécution de la méthode

```
const todo = await addTodo({
        variables: {
          input: {
            text: inputRef.current.value,
            completed: false
          }
        },
        refetchQueries: [{ query: ALL_TODOS }]
      });
```

### *Cache*

On peut définir la méthode update afin de modifier le cache et répercuter les changements dans la page :

En même temps que « useMutation »

```
 const [addTodo] = useMutation(ADD_TODO, {
    update: (cache, { data: { addTodo } }) => {
      const { todos } = cache.readQuery({
        query: ALL_TODOS
      });
      cache.writeQuery({
        query: ALL_TODOS,
        data: {
```

```
          todos: [addTodo, ...todos]
        }
      });
    }
  });
```

… Ou à l'exécution de la méthode

```
const todo = await addTodo({
        variables: {
          input: {
            text: inputRef.current.value,
            completed: false
          }
        },
        //  refetchQueries: [{ query: ALL_TODOS }]
        update: (cache, { data: { addTodo } }) => {
          const { todos } = cache.readQuery({
            query: ALL_TODOS
          });
          cache.writeQuery({
            query: ALL_TODOS,
            data: {
              todos: [addTodo, ...todos]
            }
          });
        }
      });
```

App.js

```
import React from "react";
import {
  ApolloClient,
  ApolloProvider,
  HttpLink,
  InMemoryCache
} from "@apollo/client";
import TodoPage from "./pages/Todo";

// Initialize Apollo Client
const client = new ApolloClient({
  cache: new InMemoryCache(),
  link: new HttpLink({
```

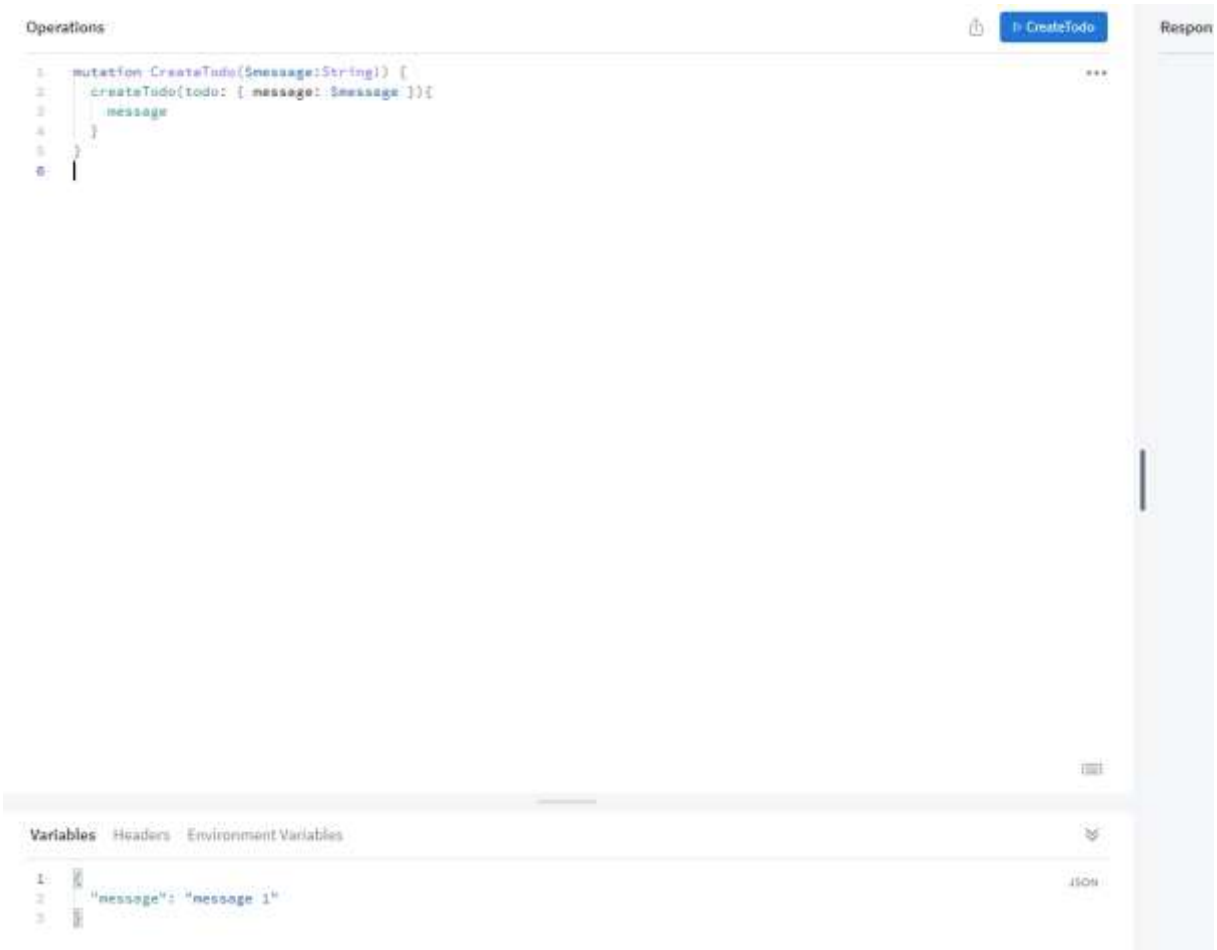
```
    uri: "http://localhost:4000/graphql"
  })
});

export default function App() {
  return (
    <ApolloProvider client={client}>
      <TodoPage />
    </ApolloProvider>
  );
}
```

Todo page

```
import React, { useEffect, useRef, useState } from "react";
import { gql, useQuery, useMutation } from "@apollo/client";

const TODOS_ATTRIBUTES = gql`
  fragment TodoInfo on Todo {
    id
    message
    completed
  }
`;

const ALL_TODOS = gql`
  query todos {
    todos {
      ...TodoInfo
    }
  }
  ${TODOS_ATTRIBUTES}
`;

const TodoList = () => {
  const { loading, error, data } = useQuery(ALL_TODOS);

  if (loading) return <p>Loading...</p>;

  if (error) return <p>Error loading todos!</p>;

  return (
    <ul>
      {data.todos.map((todo, i) => (
        <li key={i}>{todo.message}</li>
      ))}
    </ul>
  );
```

```
};

const CREATE_TODO = gql`
  mutation createTodo($todo: TodoInput!) {
    createTodo(todo: $todo) {
      ...TodoInfo
    }
  }
  ${TODOS_ATTRIBUTES}
`;

const TodoForm = () => {
  const inputRef = useRef();

  const [createTodo, { called, error }] = useMutation(CREATE_TODO, {
    update: (cache, mutationResult) => {
      cache.modify({
        fields: {
          todos(existingTodos = []) {
            const newTodo = mutationResult.data.createTodo;
            cache.writeQuery({
              query: ALL_TODOS,
              data: { newTodo, ...existingTodos }
            });
          }
        }
      });
    }
  });

  const handleSubmit = async event => {
    event.preventDefault();

    await createTodo({
      variables: { todo: { message: inputRef.current.value } }
    });

    inputRef.current.value = "";
  };

  //   if (called) return <p>Session Submitted Successfully!</p>;

  //   if (error) return <p>Failed to submit session</p>;

  return (
    <form onSubmit={handleSubmit}>
      <input ref={inputRef} />
      <input type="submit" />
    </form>
```

```
  );
};

const TodoPage = () => {
  return (
    <>
      <h1>Todos</h1>
      <TodoForm />
      <TodoList />
    </>
  );
};

export default TodoPage;
```




```
mutation CreateTodo($message:String!) {
  createTodo(todo: { message: $message }){
    message
  }
}
```