|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №7**

по дисциплине «Интерфейсы прикладного программирования»

**Тема практической работы:**

**Введение в графовые модели интерфейсов прикладного программирования**

**Студент группы** ИКБО-01-21 Маров Герман Андреевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** Зарипов Е.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Москва 2023

**ОГЛАВЛЕНИЕ**

[Цель работы 3](#_Toc115999382)

[Ход работы 3](#_Toc115999383)

[Вывод 9](#_Toc115999384)

# Цель работы

Целью данной практической работы является знакомство обучающихся с набирающим популярность современным подходом к проектированию и реализации API на основе графовых моделей и с реализующей данный подход технологией на основе спецификации GraphQL.

**Постановка задачи**

Используя теоретические сведения из данной практической работы, открытые интернет-источники, официальную документацию по GraphQL необходимо, с использованием SDL создать схему, реализовать сервер и клиента GraphQL для следующей бизнес-задачи:

**Создание приложения для хранения информации об автомобилях**

Схема должна реализовывать возможность хранения краткой информации об автомобиле (поле title), уникальном идентификаторе автомобиля(поле id), информации о бренде автомобиля (поле brand), текущей цены автомобиля (поле price), возраста автомобиля (поле age).

# Ход работы

Для создания Node.js сервера необходимо провести конфигурацию системы. На рисунках 1-2 приведены команды, необходимые для настройки окружения для дальнейшей работы. Далее в файле index.js определяется конфигурация запускаемого сервера (листинг 1), и затем в файле schema.js (листинг 2) необходимо определить схему GraphQL в соответствие с индивидуальным вариантом.

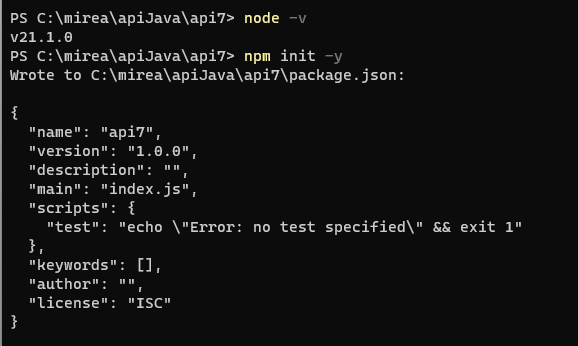


Рисунок 1 – Инициализация node.js проекта

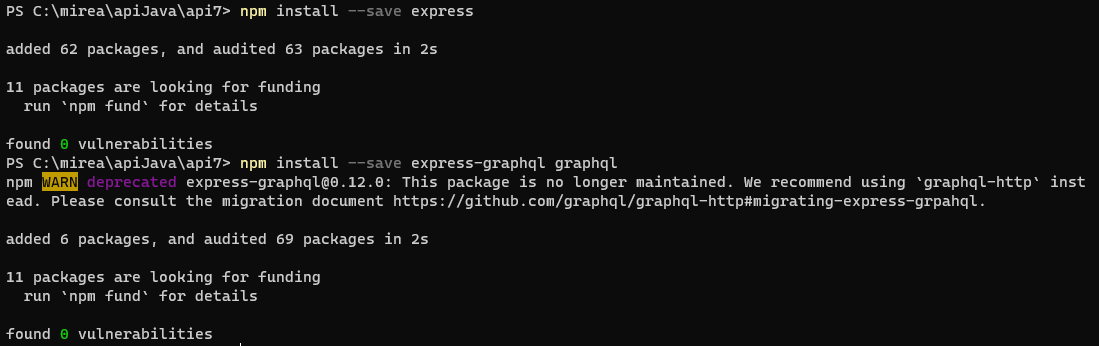


Рисунок 2 – Установка express и GraphQL

Листинг 1 – Код index.js

const express = require('express');

const{ graphqlHTTP } = require('express-graphql');

const schema = require('./schema/schema.js');

const app = express();

const port = 1234;

app.use('/graphql',

    graphqlHTTP({

        schema: schema,

        graphiql: true,

    }))

app.listen(port);

Листинг 2 – Код schema.js

const graphql = require('graphql');

const \_ = require("lodash");

const {

    GraphQLID,

    GraphQLString,

    GraphQLList,

    GraphQLInt,

    GraphQLNonNull,

    GraphQLObjectType,

    GraphQLSchema

} = graphql;

const cars = [

    {

        id: "1",

        title: "Kalina Restyling",

        brand: "LADA",

        price: 500000,

        age: 5

    },

    {

        id: "2",

        title: "C220",

        brand: "Mercedes-Benz",

        price: 5000000,

        age: 1

    },

    {

        id: "3",

        title: "Land Cruiser 200",

        brand: "Toyota",

        price: 3000000,

        age: 3

Продолжение листинга 2

    }];

const CarInfoType = new GraphQLObjectType({

    name:'CarInfo',

    fields: () =>({

        id: {type: GraphQLID},

        title: {type: GraphQLString},

        brand: {type: GraphQLString},

        price: {type: GraphQLInt},

        age: {type: GraphQLInt}

    })

})

const RootQuery = new GraphQLObjectType({

    name: 'RootQueryType',

    fields: {

        info:{

            type: GraphQLString,

            resolve(parent, args){

                return "Server's running"

            }

        },

        car:{

            type: CarInfoType,

            args: {id: {type: GraphQLID}},

            resolve(parent, args) {

                return \_.find(cars, {id: args.id});

            }

        },

        cars:{

            type: new GraphQLList(CarInfoType),

            resolve(parent, args) {

                return cars;

            }

        }

    }

});

const Mutations = new GraphQLObjectType({

    name: 'Mutations',

    fields:{

        addcar: {

            type: CarInfoType,

            args: {

                id: {type: new GraphQLNonNull(GraphQLID)},

                title: {type: new GraphQLNonNull(GraphQLString)},

                brand: {type: GraphQLString},

                price: {type: new GraphQLNonNull(GraphQLInt)},

                age: {type: GraphQLInt}

Продолжение листинга 2

            },

            resolve(parent, args) {

                const arrLength = cars.push(args);

                return cars [arrLength - 1];

            }

        }

    }

});

module.exports = new GraphQLSchema({

    query: RootQuery,

    mutation: Mutations

});

Командой *node index.js* выполним запуск сервера. Перейдя по URL http://localhost:1234/graphql мы попадаем на веб-интерфейс GrapiQL, позволяющий в удобной форме выполнять GraphQL запросы (рисунок 3).

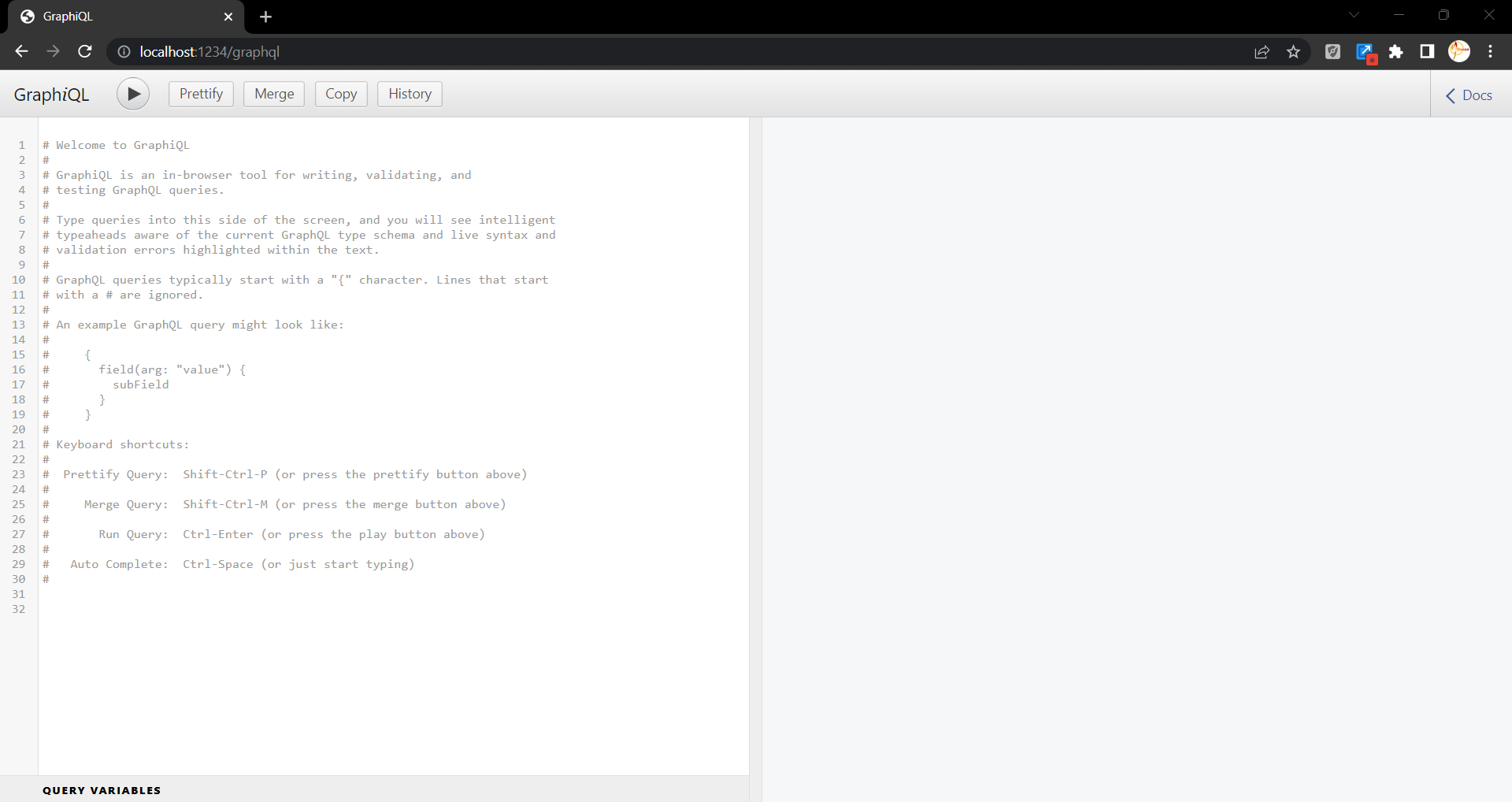


Рисунок 3 – Веб-интерфейс GraphiQL

Далее выполним проверку работы GraphQL сервера с помощью POST-запроса mutation (рисунок 4) и GET-запроса cars (рисунок 5), которые добавляют новый автомобиль и выводят список всех имеющихся машин соответственно.

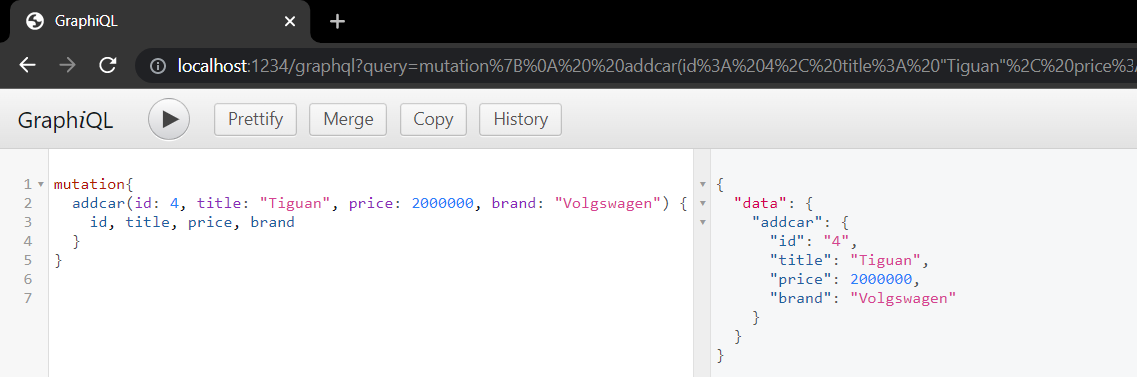


Рисунок 4 – Выполнение запроса mutation

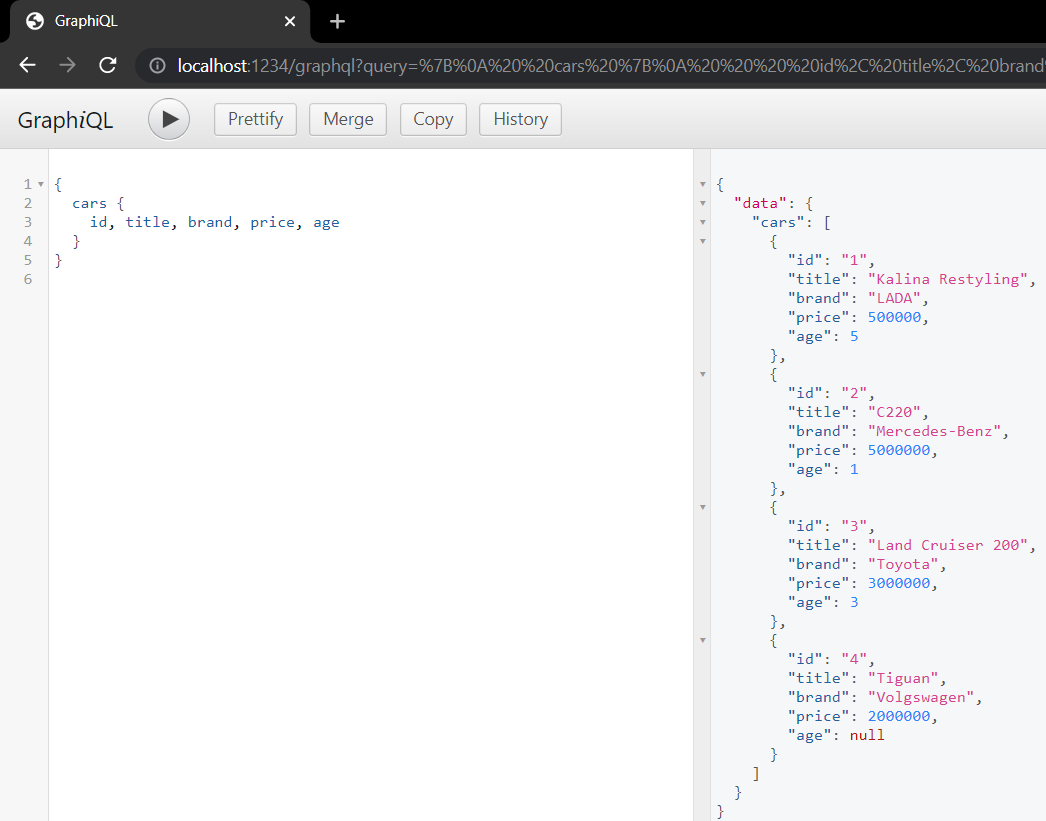


Рисунок 5 – Выполнение запроса cars

# Вывод

Используя теоретические сведения из данной практической работы, открытые интернет-источники, официальную документацию по GraphQL с использованием SDL создал схему, реализовал сервер и клиента GraphQL для приложения для хранения информации об автомобилях.