Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Звіт до комп’ютерного практикуму №6

З дисципліни «Основи Back-end технологій»

Прийняв:
Зубко Роман Анатолійович

Виконав:
Студент 3 курсу,
Котенко Ярослав
гр. ІМ-13

2024 р.

Лабораторна робота №6. Тема: GraphQL.

Створення Schema GraphQL та Resolvers.

Створення Query та Mutation.

Завдання.

- На своїй БД (розробленої в лаб. роб. #5) за допомогою Schema Definition Language (SDL) створити схему GraphQL.
- Додати Resolvers для виконання операцій GraphQL.
- Створити та виконати Query та Mutation для виконання операцій додавання, редагування та видалення інформації (CRUD) в БД.
- Виконати дослідження роботи створених query та mutation за допомогою Postman.

Хід роботи

На своїй БД (розробленої в лаб. роб. #5) за допомогою Schema Definition Language (SDL) створив схему GraphQL.

schema > schema.graphql

```
type Detail {
  id : ID!
  name : String!
  description : String!
}

type ProcType {
  id : ID!
  name : String!
  description : String!
}

type TechCard {
  id : ID!
  detail_ID : Detail!
  procType_ID : ProcType!
  duration : Int!
  creationDate : String
  description : String!
}

type Query {
  getTechCardById(id: ID): TechCard
  getProcTypeById(id: ID): ProcType
  getDetailById(id: ID): Detail
  getAllTechCards : [TechCard!]
  getAllProcTypes : [ProcType!]
  getAllDetails : [Detail!]
}

```

schema > schema.graphql

```
30
31  input newTechCard {
32    detail_ID : ID
33    procType_ID : ID
34    duration : Int
35    creationDate : String
36    description : String
37  }
38
39  input newDetail {
40    name : String
41    description : String
42  }
43
44  input newProcType {
45    name : String
46    description : String
47  }
48
49  type Mutation {
50    addTechCard(techCard: newTechCard): [TechCard!]!
51    addDetail(detail: newDetail): [Detail!]!
52    addProcType(procType: newProcType): [ProcType!]!
53    updateTechCard(id: ID!, body: newTechCard): TechCard
54    updateProcType(id: ID!, body: newProcType): ProcType
55    updateDetail(id: ID!, body: newDetail): Detail
56    deleteTechCard(id: ID!): [TechCard!]!
57    deleteProcType(id: ID!): [ProcType!]!
58    deleteDetail(id: ID!): [Detail!]!
59  }
```

Додав **Resolvers** для виконання операцій GraphQL. В Resolvers створив та виконав **Query** та **Mutation** щоб описати **CRUD** операції для головної таблиці(**TechCard**) та допоміжних таблиць(**ProcessingTypes**, **Details**).

schema > resolver.js > resolvers > Query > getTechCardById

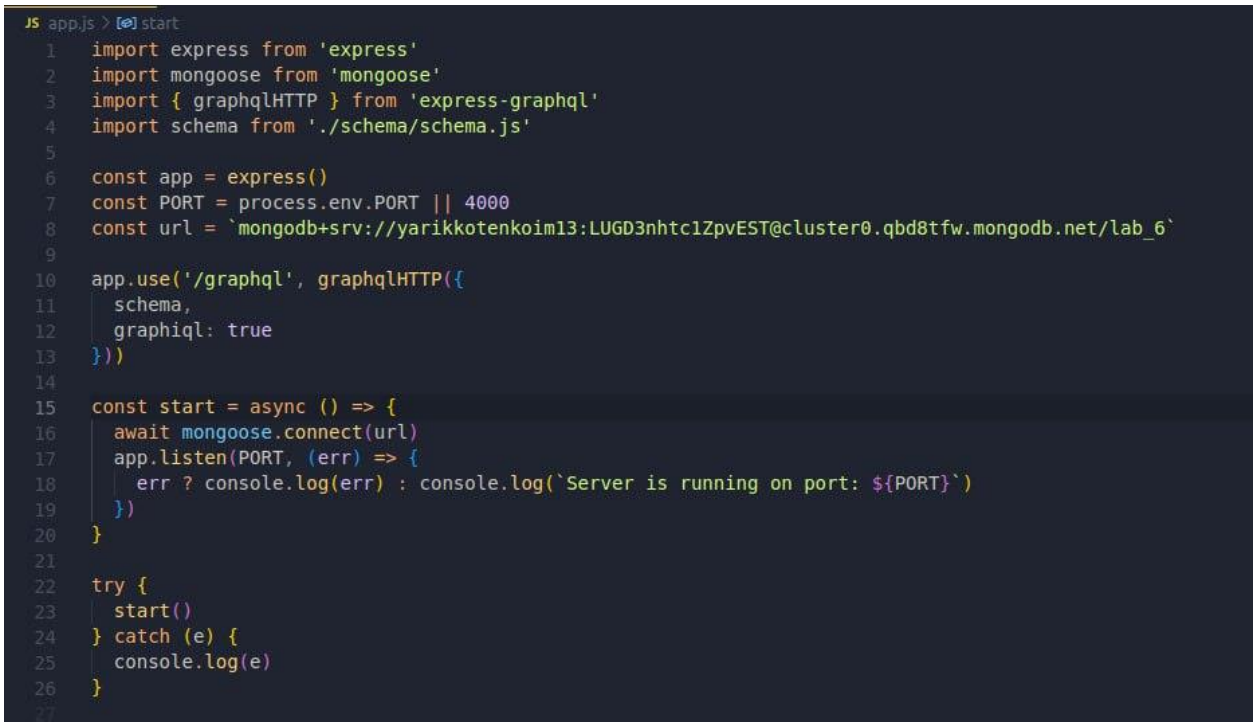
```
1   import techCards from '../models/techCards.js'
2   import processingTypes from '../models/processingTypes.js'
3   import details from '../models/details.js'
4
5   const resolvers = {
6     Query: {
7       getTechCardById: async (_, { id }) => {
8         return await techCards.findById(id)
9       },
10      getProcTypeById: async (_, { id }) => {
11        return await processingTypes.findById(id)
12      },
13      getDetailById: async (_, { id }) => {
14        return await details.findById(id)
15      },
16      getAllTechCards: async () => {
17        return await techCards.find()
18      },
19      getAllProcTypes: async () => {
20        return await processingTypes.find()
21      },
22      getAllDetails: async () => {
23        return await details.find()
24      }
25    },
26    Mutation: {
27      addTechCard: async (_, { techCard: { detail_ID, procType_ID, duration, creationDate, description } }) => {
28        const date = creationDate ? creationDate : new Date(Date.now())
29        try {
30          const newTechCard = new techCards({
31            detail_ID,
32            procType_ID,
33            duration,
34            creationDate: date,
35            description
36          })
37          await newTechCard.save()
38          return await techCards.find()
39        } catch (err) {
40          throw new Error(err)
41        }
42      },
43      addDetail: async (_, { detail: { name, description } }) => {
44        try {
45          const newDetail = new details({
46            name,
47            description
48          })
49          await newDetail.save()
50          return await details.find()
51        } catch (err) {
52          throw new Error(err)
53        }
54      },
```

```
    addProcType: async (_, { procType: { name, description } }) => {
      try {
        const newProcType = new processingTypes({
          name,
          description
        })
        await newProcType.save()
        return await processingTypes.find()
      } catch (err) {
        throw new Error(err)
      }
    },
    updateTechCard: async (_, { id, body }) => {
      try {
        await techCards.findByIdAndUpdate(id, body)
        return techCards.findById(id)
      } catch (err) {
        throw new Error(err)
      }
    },
    updateProcType: async (_, { id, body }) => {
      try {
        await processingTypes.findByIdAndUpdate(id, body)
        return processingTypes.findById(id)
      } catch (err) {
        throw new Error(err)
      }
```

schema > JS resolver.js > resolvers > Mutation > updateDetail

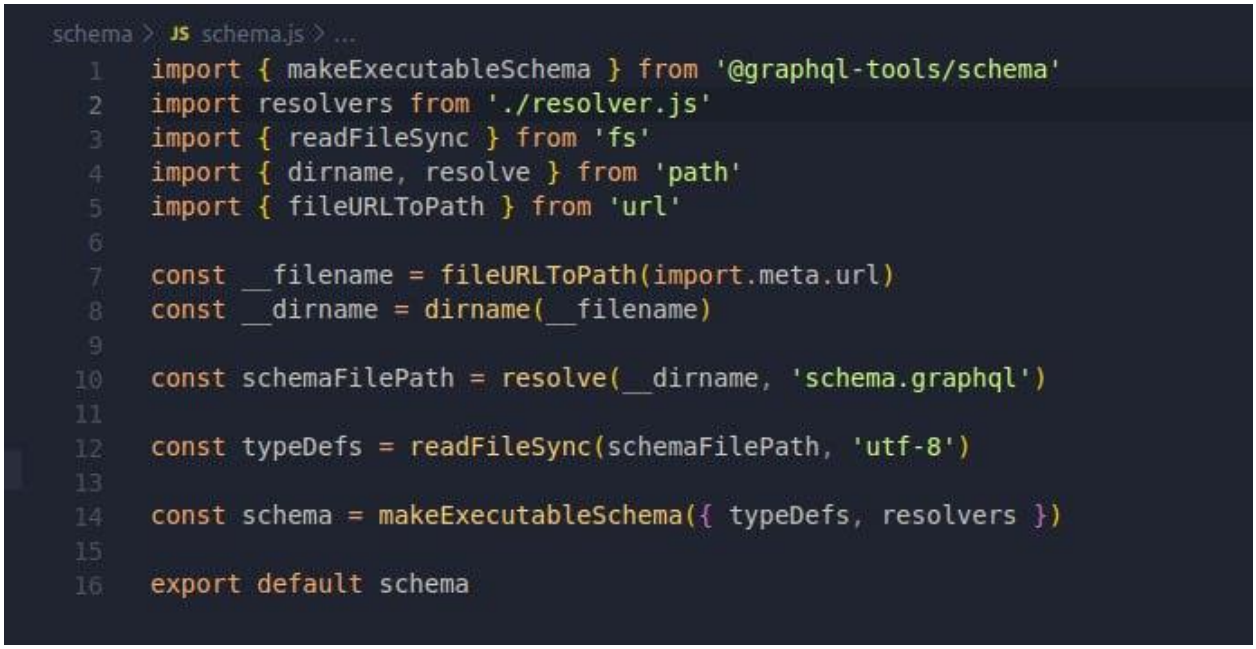
```
const resolvers = {
  Mutation: {
    addDetail: async (_, { detail: { name, description } }) => {
      try {
        const newDetail = new details({
          name,
          description
        })
        await newDetail.save()
        return await details.find()
      } catch (err) {
        throw new Error(err)
      }
    },
    addProcType: async (_, { procType: { name, description } }) => {
      try {
        const newProcType = new processingTypes({
          name,
          description
        })
        await newProcType.save()
        return await processingTypes.find()
      } catch (err) {
        throw new Error(err)
      }
    },
    updateTechCard: async (_, { id, body }) => {
      try {
        await techCards.findByIdAndUpdate(id, body)
        return techCards.findById(id)
      } catch (err) {
        throw new Error(err)
      }
    },
    updateProcType: async (_, { id, body }) => {
      try {
        await processingTypes.findByIdAndUpdate(id, body)
        return processingTypes.findById(id)
      } catch (err) {
        throw new Error(err)
      }
      }
    },
    updateDetail: async (_, { id, body }) => {
      try {
        await details.findByIdAndUpdate(id, body)
        return details.findById(id)
      } catch (err) {
        throw new Error(err)
      }
    },
    deleteTechCard: async (_, { id }) => {
      await techCards.findByIdAndDelete(id)
      return await techCards.find()
    },
    deleteProcType: async (_, { id }) => {
      await processingTypes.findByIdAndDelete(id)
      return await processingTypes.find()
    },
    deleteDetail: async (_, { id }) => {
      await details.findByIdAndDelete(id)
      return await details.find()
    },
  },
  TechCard: {
    detail_ID: async (parent) => await details.findById(parent.detail_ID),
    procType_ID: async (parent) => await processingTypes.findById(parent.procType_ID)
  }
}

export default resolvers
```

Головний файл де я запускаю сервер



```
app.js > start
1  import express from 'express'
2  import mongoose from 'mongoose'
3  import { graphqlHTTP } from 'express-graphql'
4  import schema from './schema/schema.js'
5
6  const app = express()
7  const PORT = process.env.PORT || 4000
8  const url = `mongodb+srv://yarikkotenkoim13:LUGD3nhtc1ZpvEST@cluster0.qbd8tfw.mongodb.net/lab_6`
9
10 app.use('/graphql', graphqlHTTP({
11   schema,
12   graphiql: true
13 }))
14
15 const start = async () => {
16   await mongoose.connect(url)
17   app.listen(PORT, (err) => {
18     err ? console.log(err) : console.log(`Server is running on port: ${PORT}`)
19   })
20 }
21
22 try {
23   start()
24 } catch (e) {
25   console.log(e)
26 }
```

Допоміжний файл для створення схеми

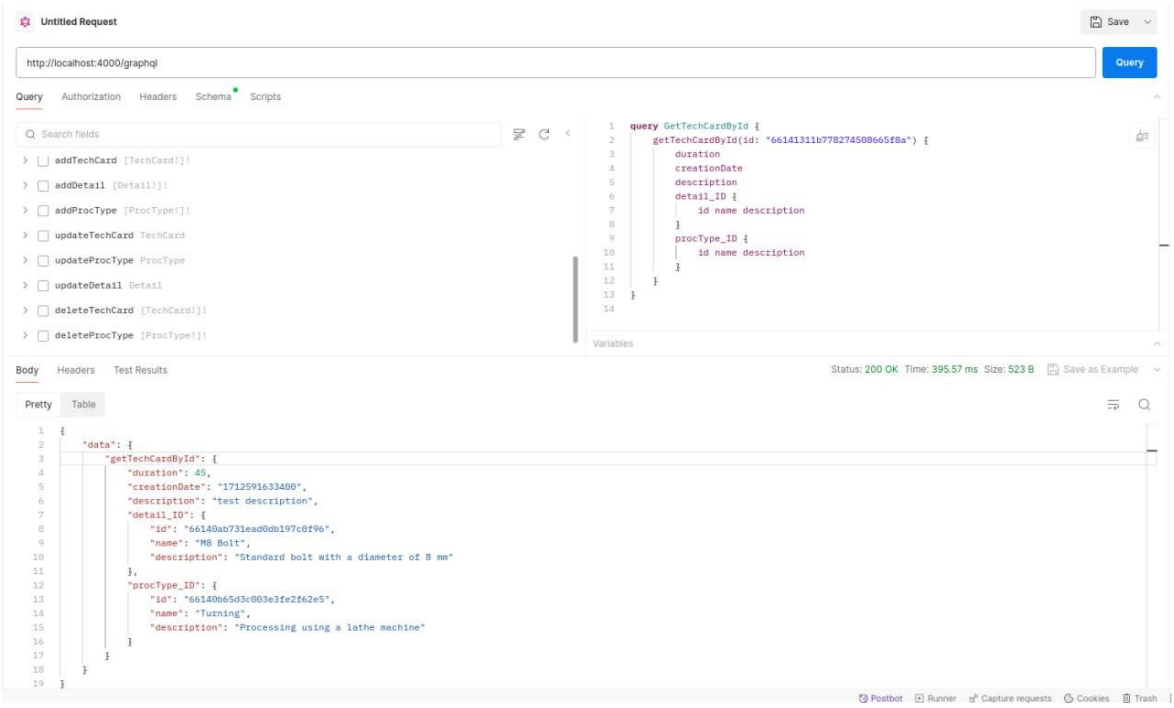


```
schema > schema.js > ...
1  import { makeExecutableSchema } from '@graphql-tools/schema'
2  import resolvers from './resolver.js'
3  import { readFileSync } from 'fs'
4  import { dirname, resolve } from 'path'
5  import { fileURLToPath } from 'url'
6
7  const __filename = fileURLToPath(import.meta.url)
8  const __dirname = dirname(__filename)
9
10 const schemaFilePath = resolve(__dirname, 'schema.graphql')
11
12 const typeDefs = readFileSync(schemaFilePath, 'utf-8')
13
14 const schema = makeExecutableSchema({ typeDefs, resolvers })
15
16 export default schema
```

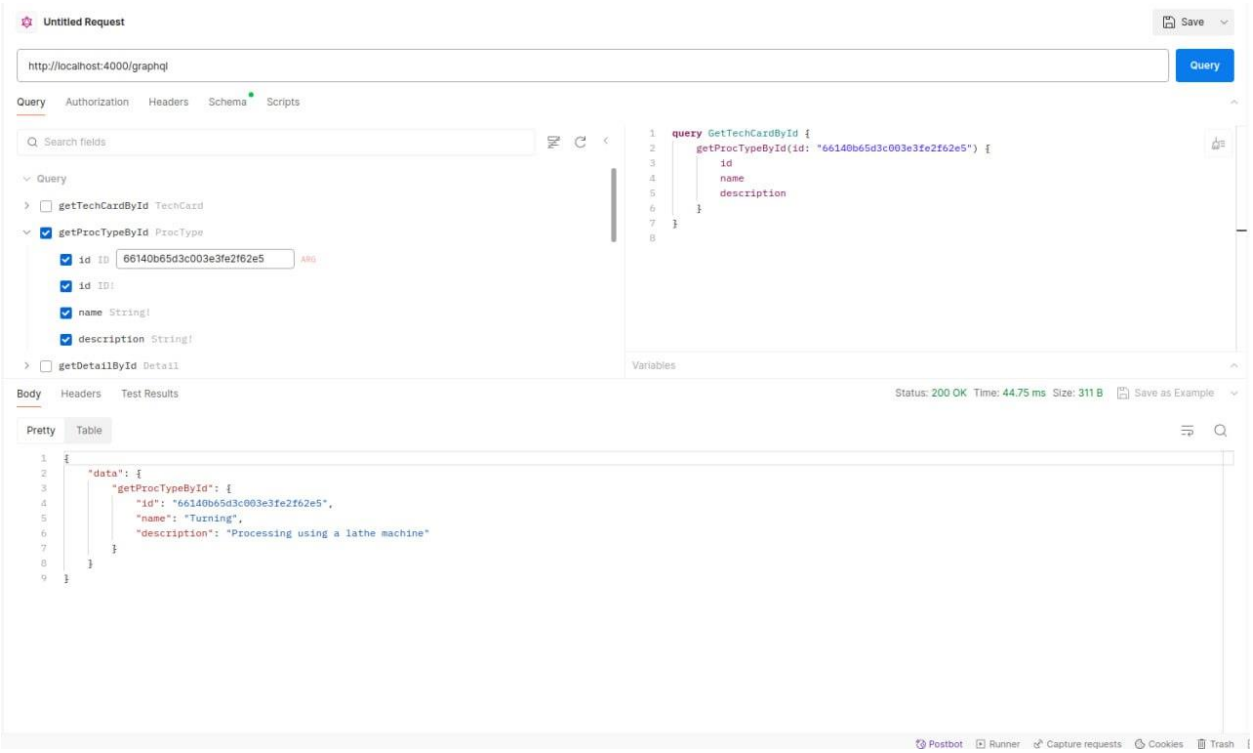
Тут я об’єдную resolvers з описом GraphQL типів та передаю його до головного файлу серверу.

Дослідження

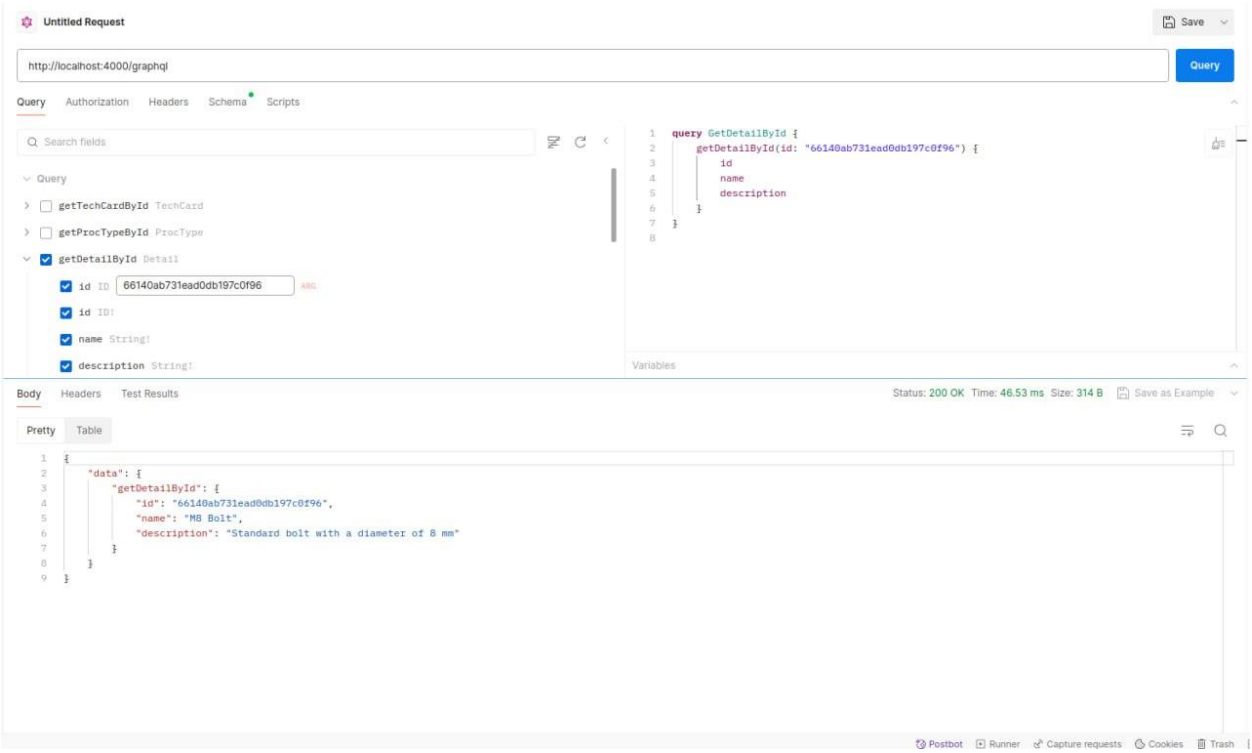
Отримання технологічної карти по id:



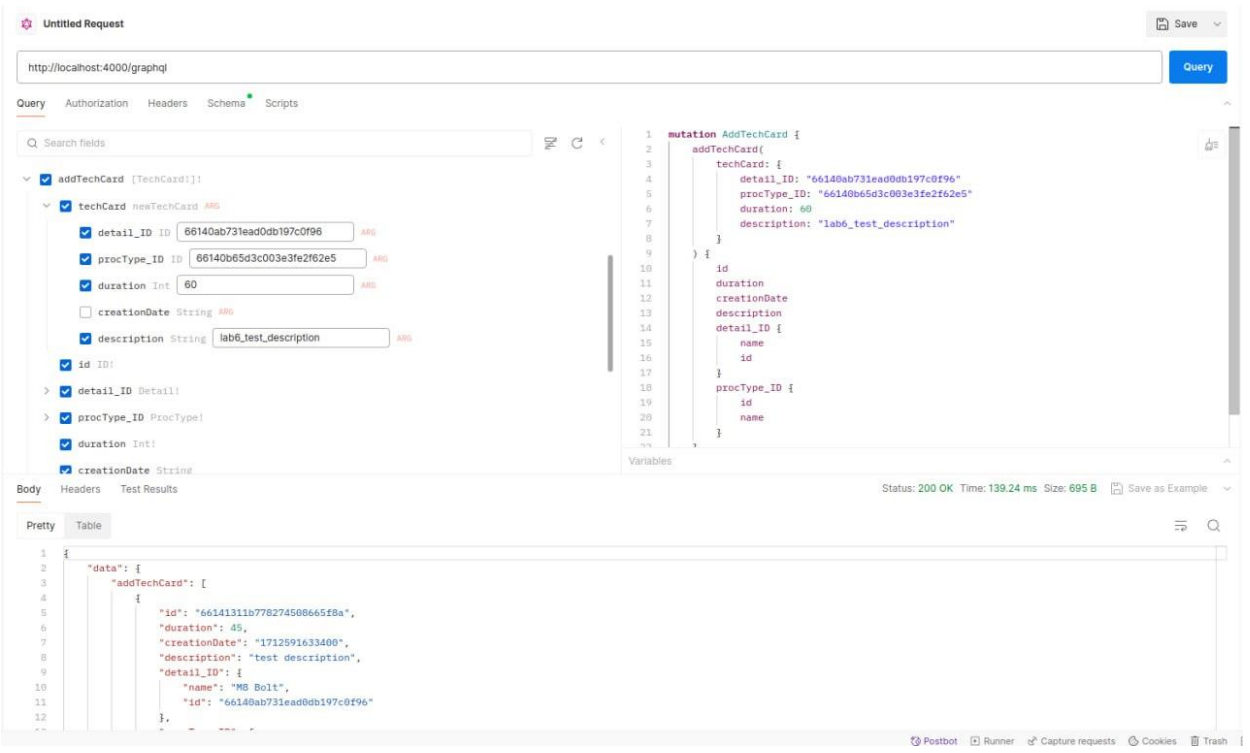
Отримання типу обробки по id:

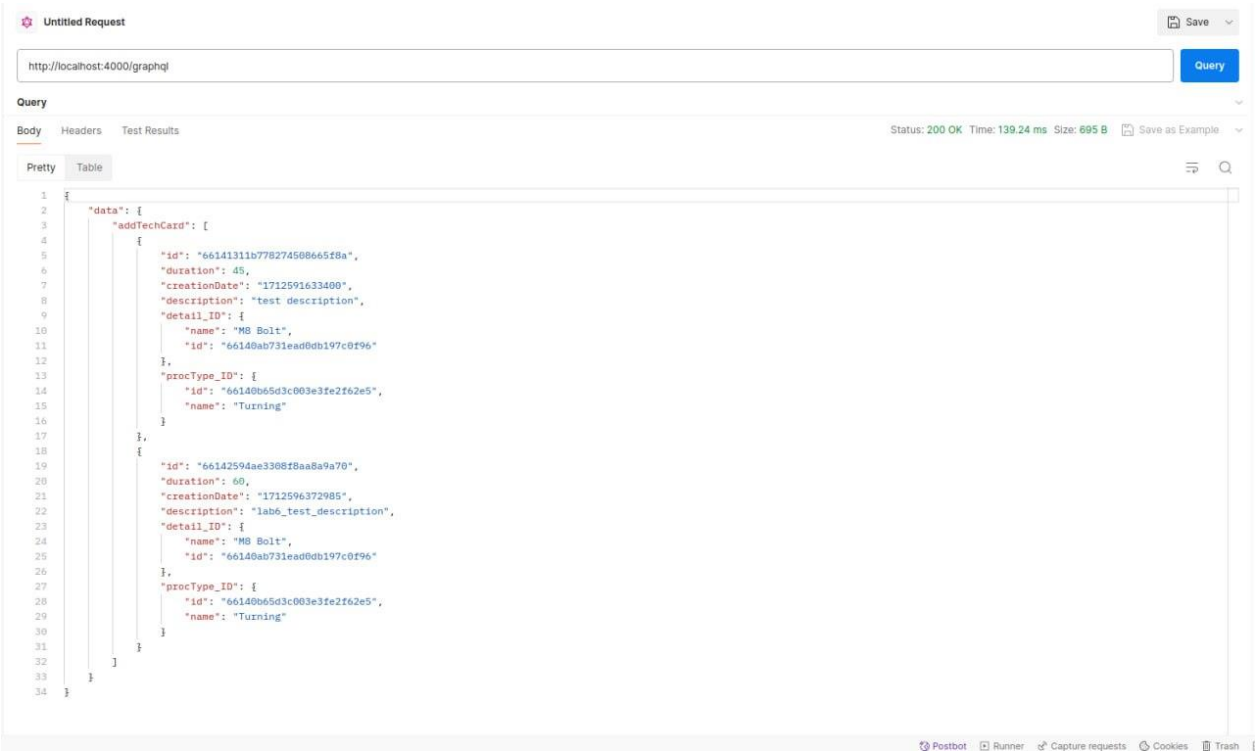


Отримання деталі по id:

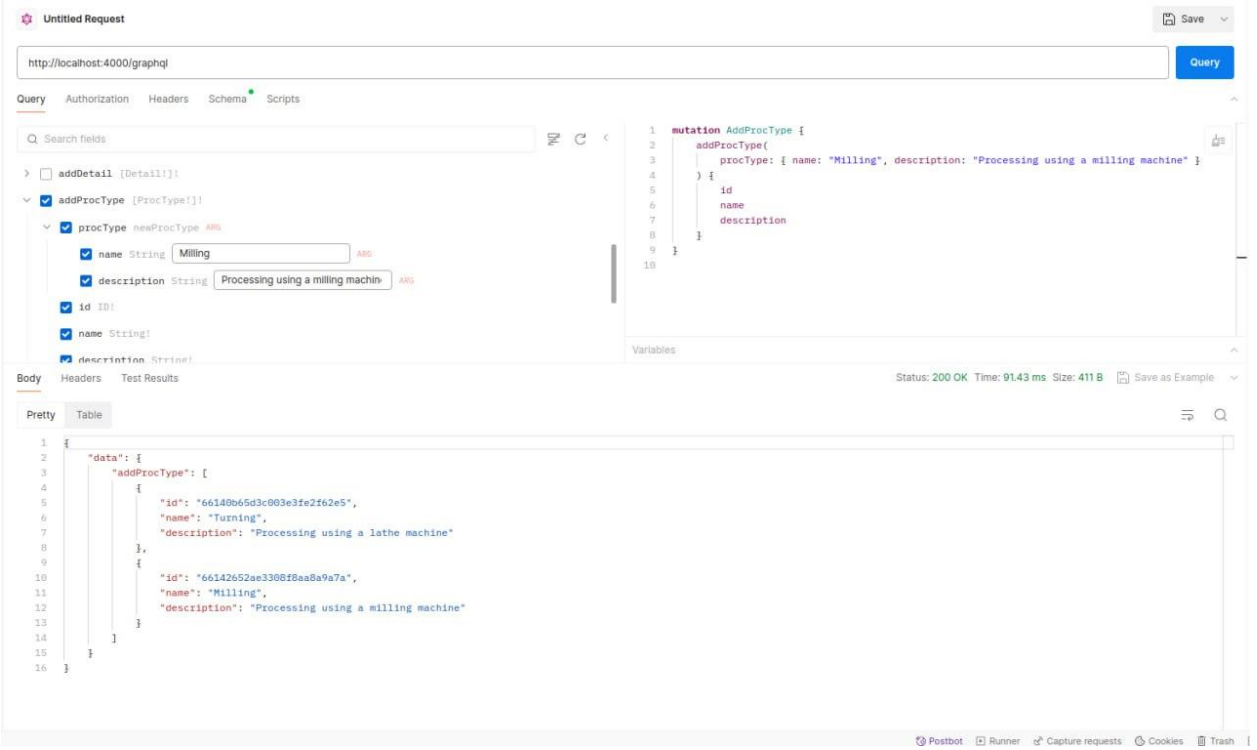


Створення нової технологічної карти:

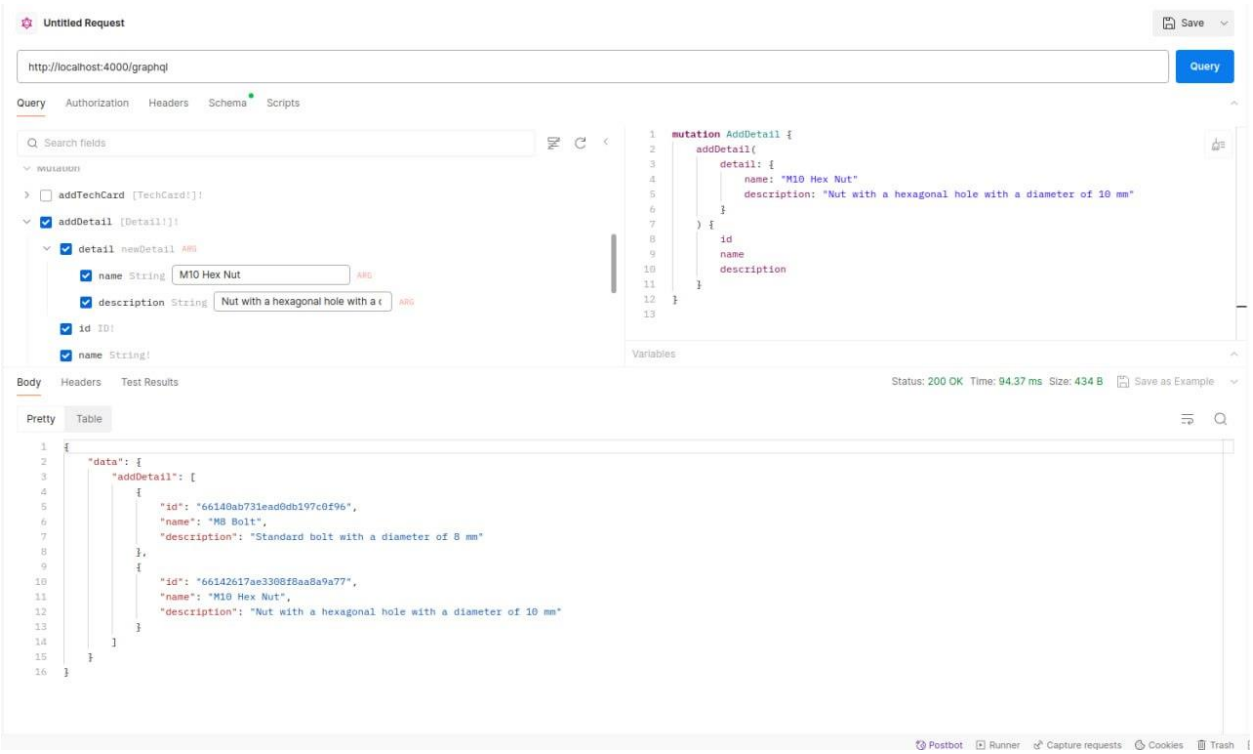




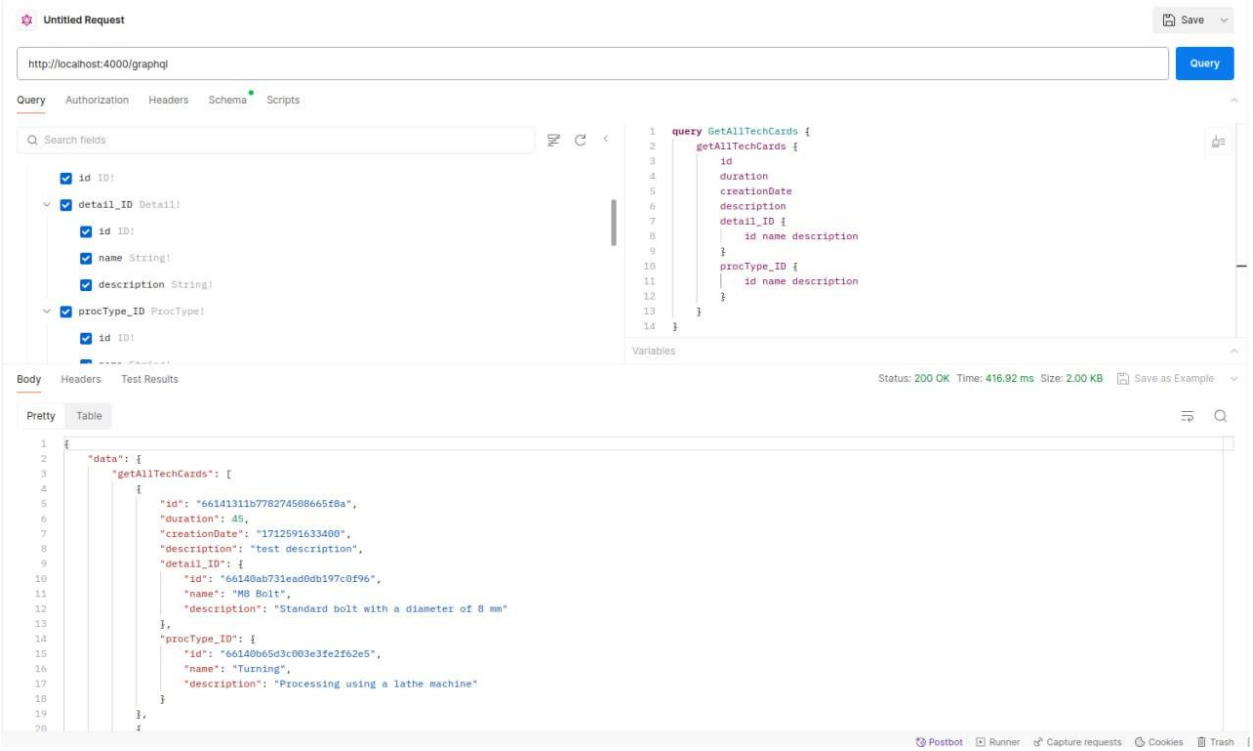
Створення нового типу обробки:

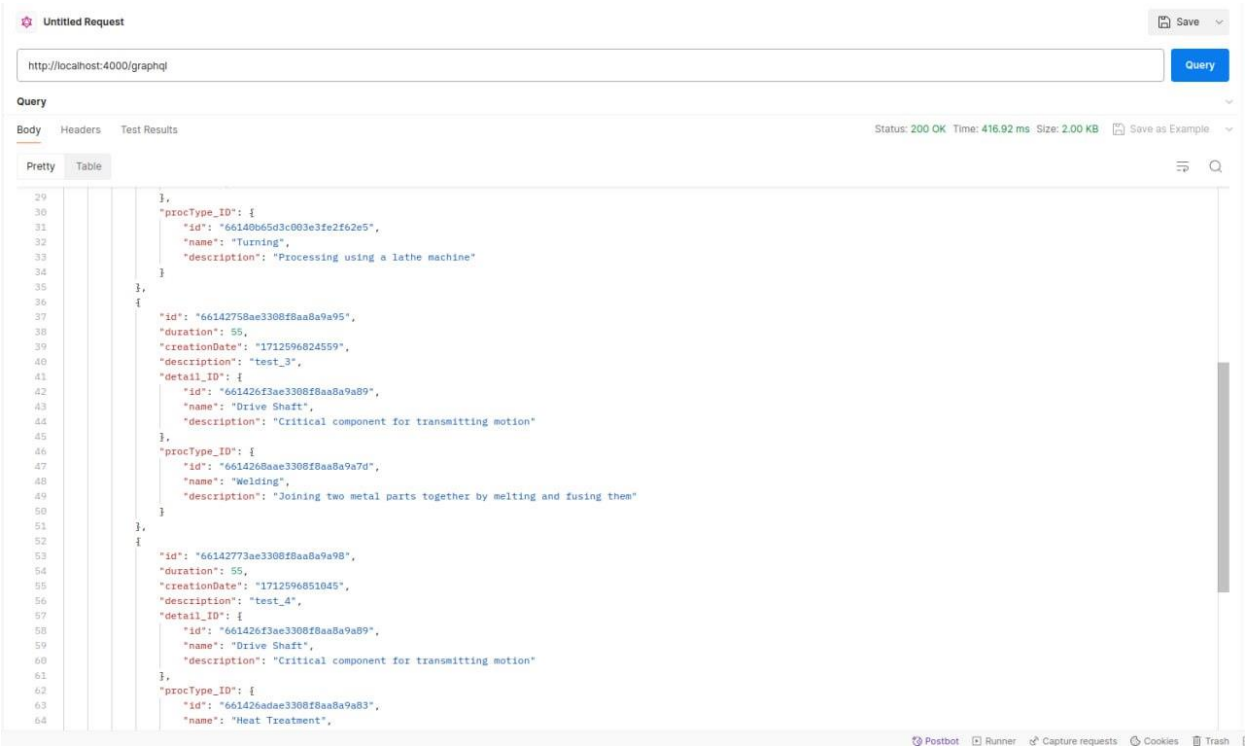


Створення нової деталі:

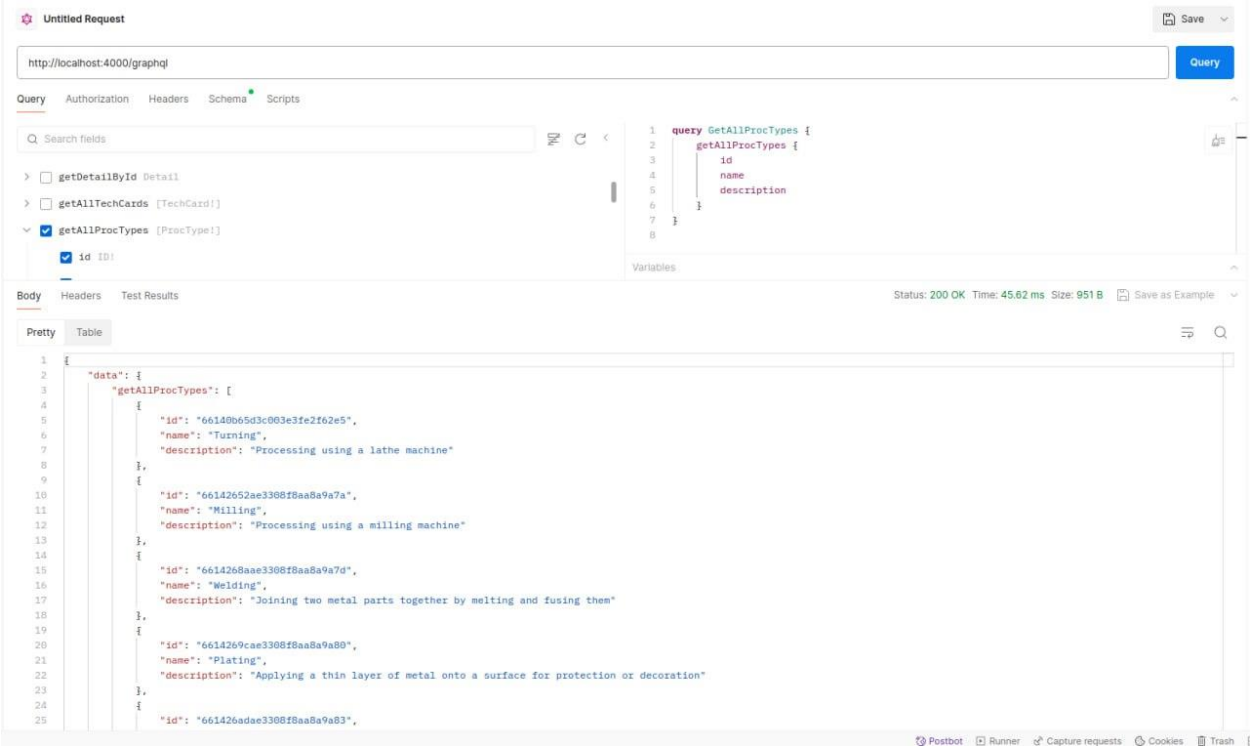


Отримання всіх технологічних карток:

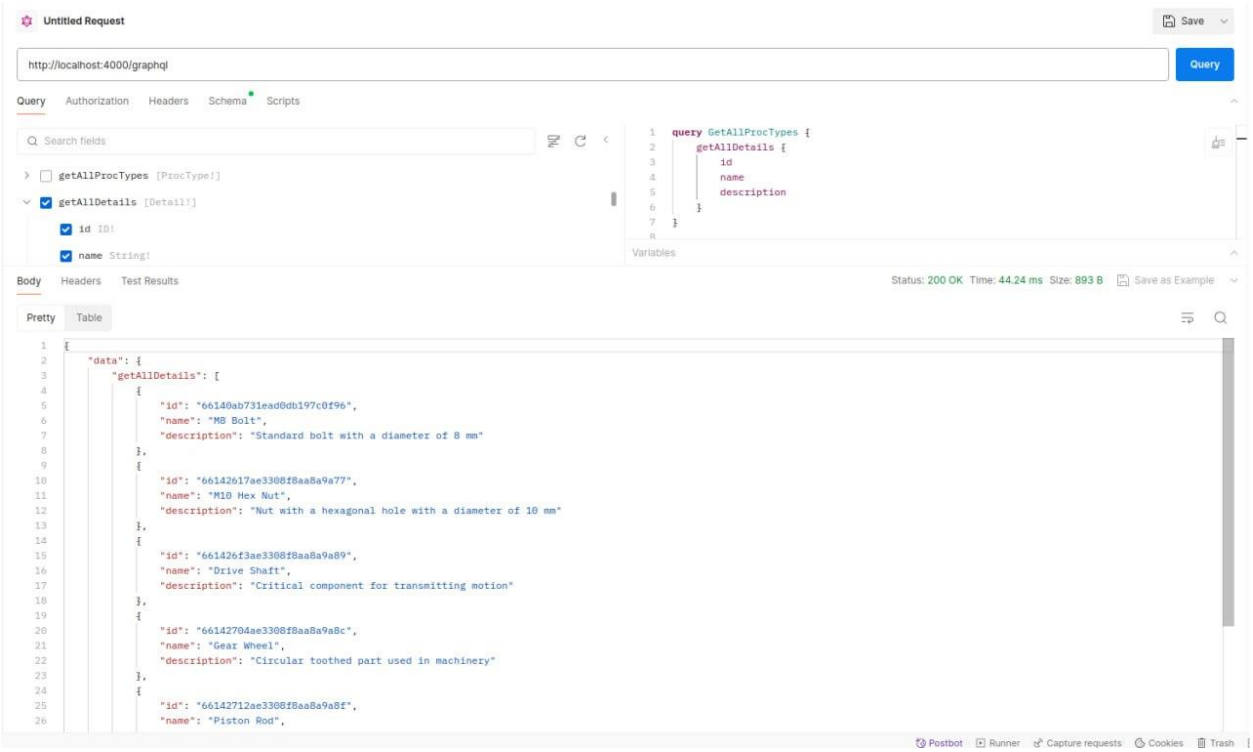




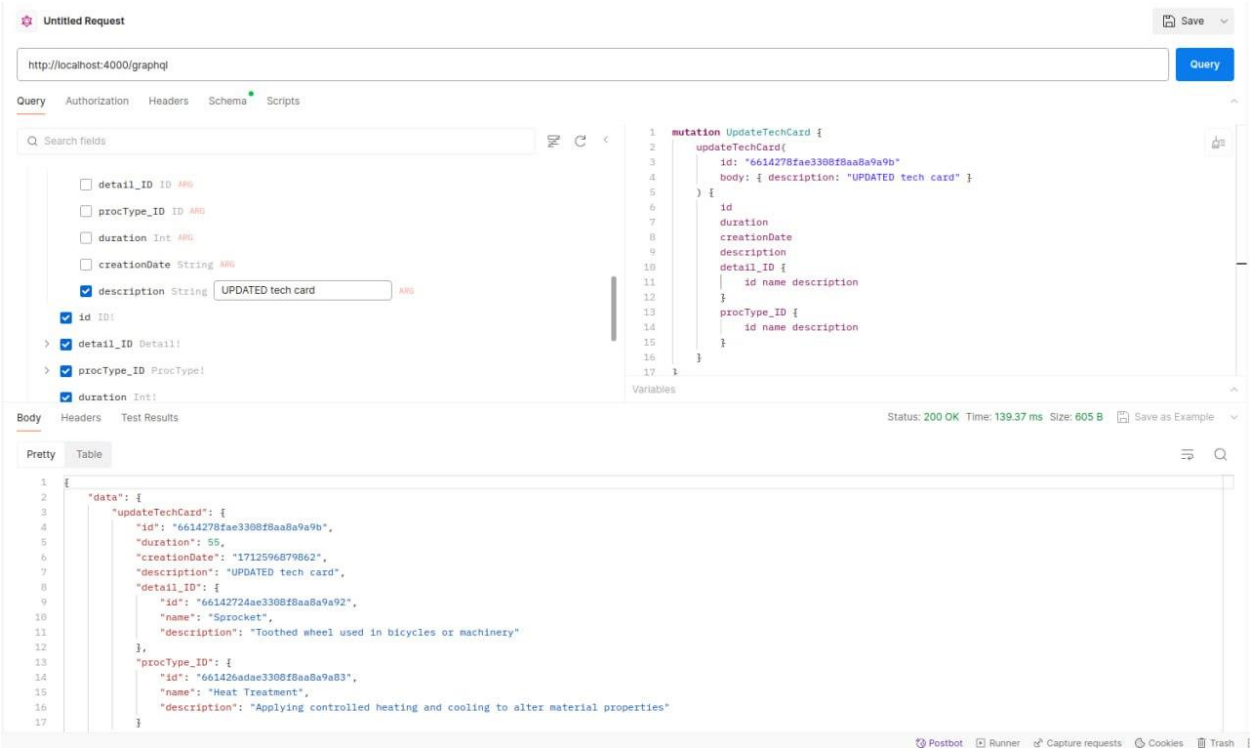
Отримання всіх типів обробки:



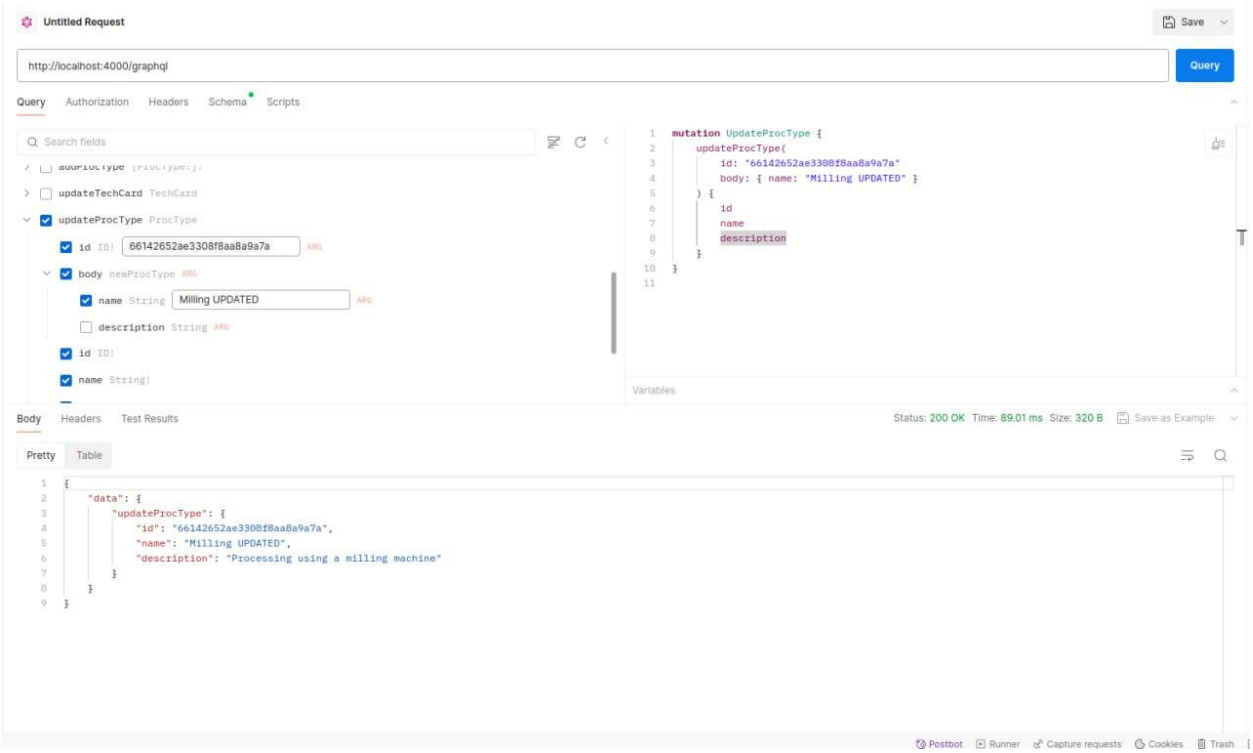
Отримання всіх деталей:



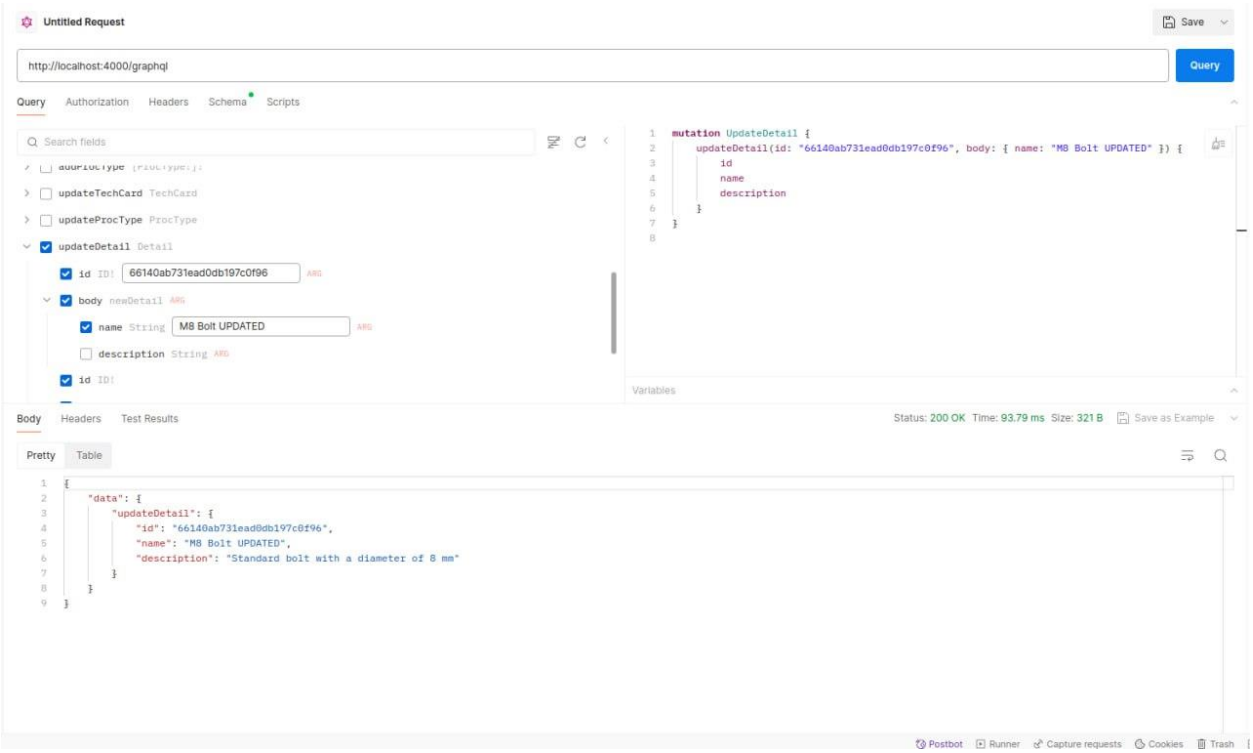
Оновлення техноголічної картки:



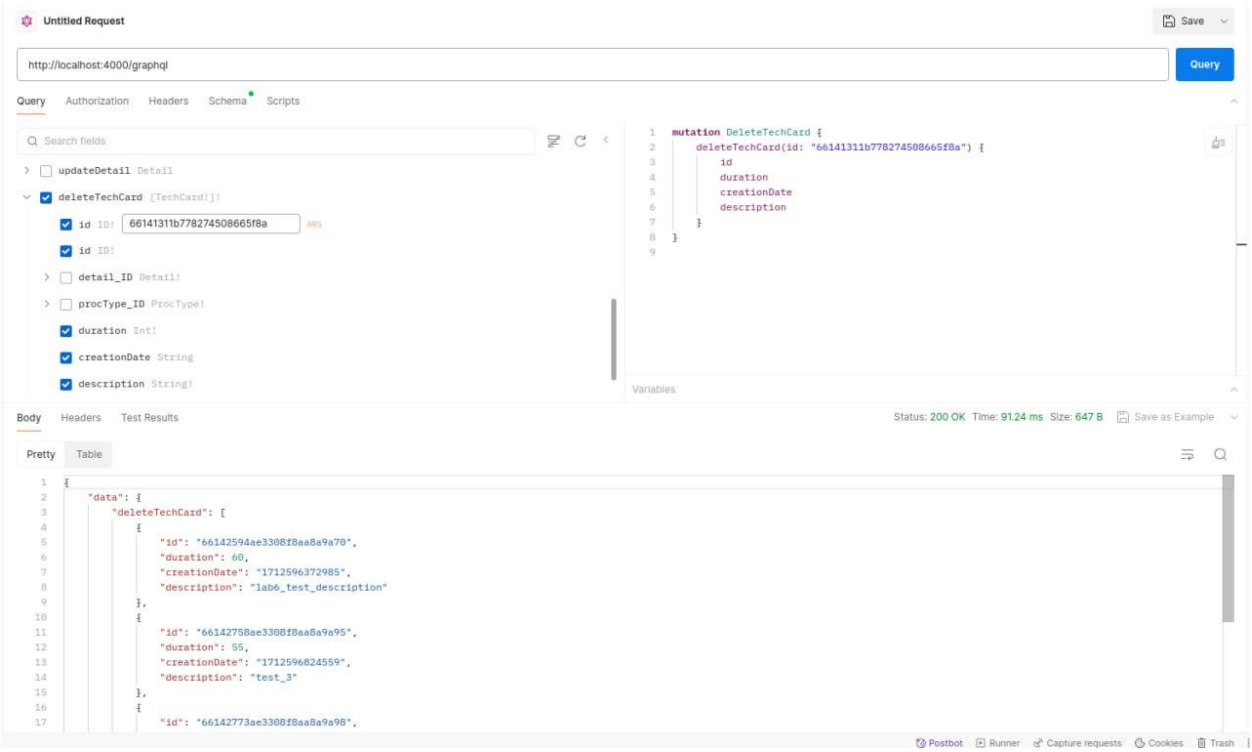
Оновлення типу обробки:



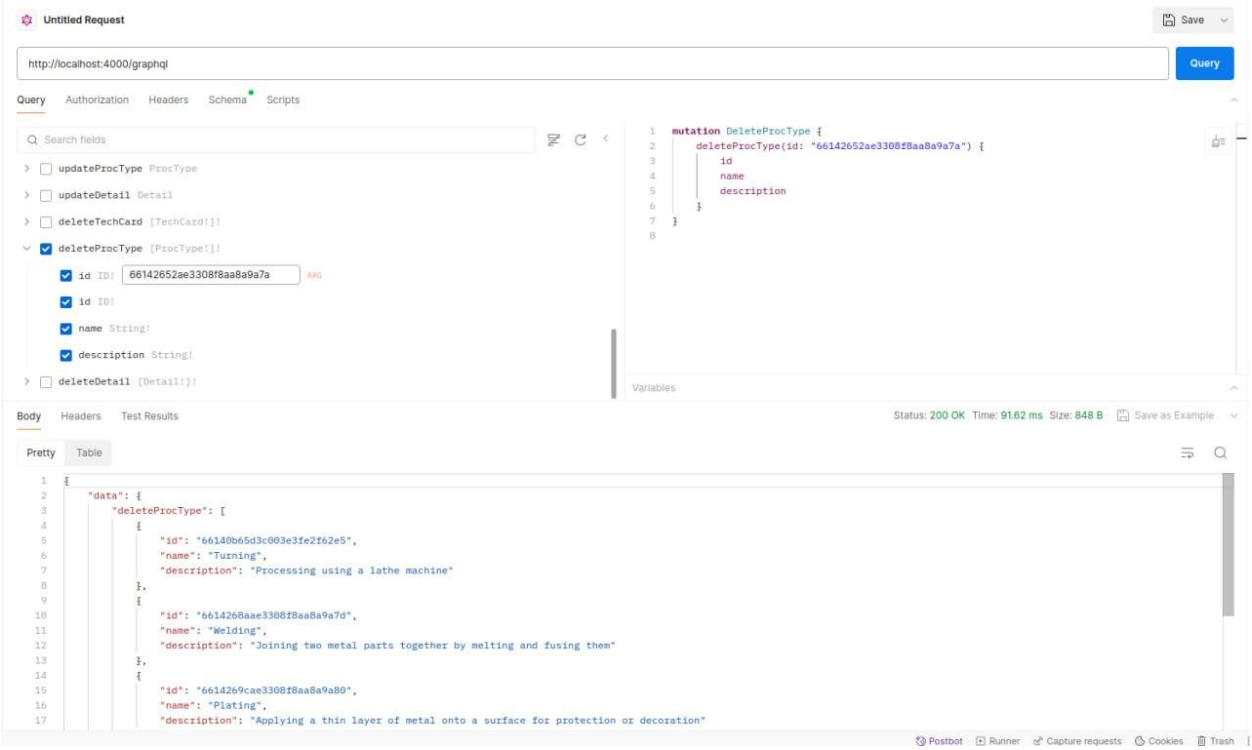
Оновлення деталі:



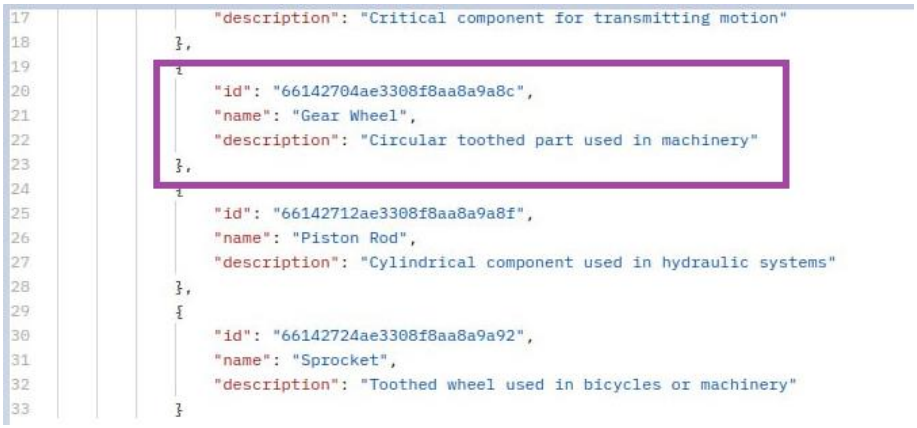
Видалення першої технологічної картки (буде повертатись список тих карток, що залишилось, на скріншотах вище можна буде переконатись, що видалена картка дійсно існувала):

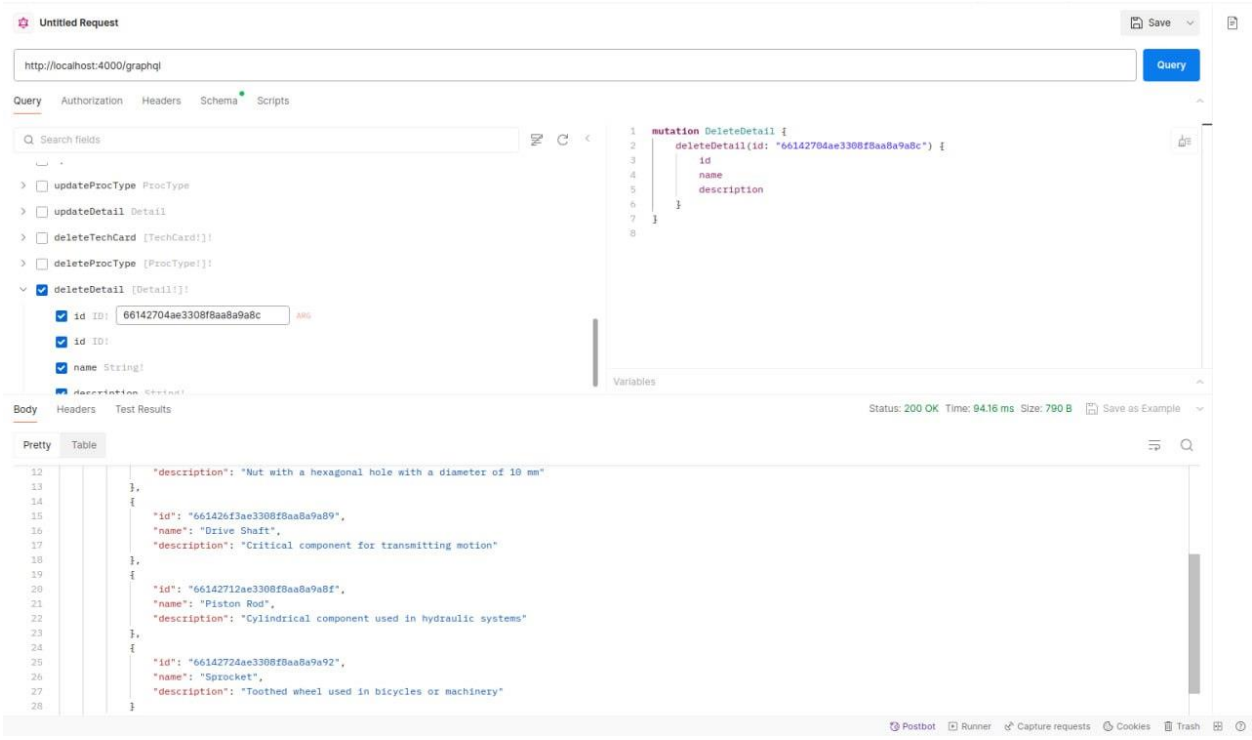


Видалення типу обробки:



Видалення деталі (Gear Wheel):





Висновок:

Під час виконання лабораторної роботи з GraphQL я отримав досвід у створенні схеми GraphQL та реалізації resolvers для операцій CRUD та використання Query та Mutation для взаємодії з базою даних. Це дозволило мені отримати базові знання та навички у роботі з GraphQL.