



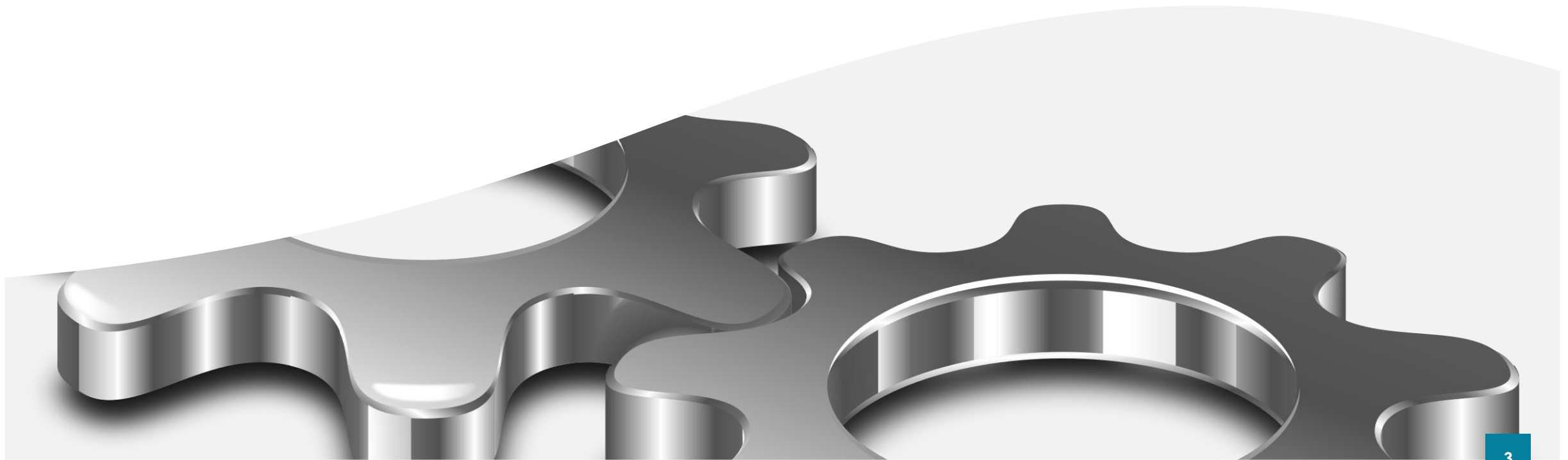
Golang Programming

Building GraphQL Web Services with Go

Where to Find The Code and Materials?

<https://github.com/iproduct/coursego>

GraphQL



GraphQL – Brief History

- GraphQL is an open-source data [query](#) and [manipulation](#) language for APIs, and a runtime for fulfilling queries with existing data.
- GraphQL was developed internally by [Facebook](#) in [2012](#) before being publicly released in [2015](#)
- On [7 November 2018](#), the GraphQL project was moved from Facebook to the newly-established [GraphQL Foundation](#), hosted by the non-profit Linux Foundation.
- Since [2012](#), GraphQL's rise has followed the [adoption timeline](#) as set out by [Lee Byron](#), GraphQL's creator, with accuracy. Byron's goal is to make GraphQL omnipresent across web platforms.

GraphQL

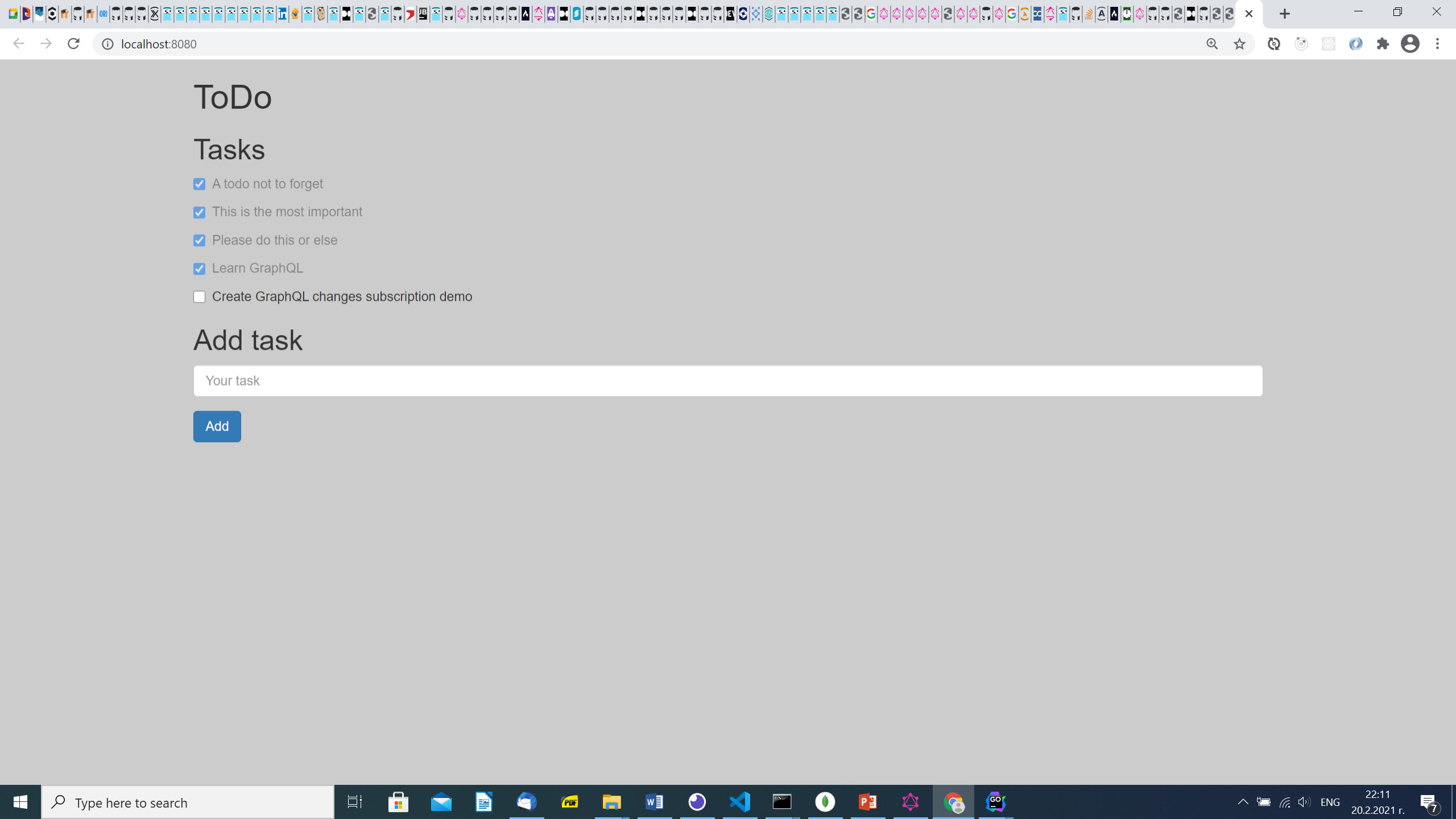
- Similarly to [REST](#) and [gRPC](#) it allows development of [web service APIs](#)
- [Clients](#) can define [the structure of the data required](#), and the same structure of the data is returned from the server, therefore [preventing excessively large amounts of data from being returned](#), but this has implications for how effective [web caching](#) of query results can be.
- It consists of a [type system](#), [query language](#) and [execution semantics](#), [static validation](#), and [type introspection](#).
- GraphQL supports [reading](#), [writing \(mutating\)](#), and [subscribing to changes](#) to data (realtime updates)
- GraphQL servers are available for multiple languages, including [Haskell](#), [JavaScript](#), [Perl](#) [Python](#), [Ruby](#), [Java](#), [C++](#), [C#](#), [Scala](#), [Go](#), [Rust](#), [Elixir](#), [Erlang](#), [PHP](#), [R](#), and [Clojure](#).

GraphQL Client

- Download GraphQL Client at:
<https://github.com/skevy/graphql-app/releases>
- Examples - query and mutation:

```
query all_todos{  
  todoList {  
    id  
    text  
    done  
  }  
}
```

```
mutation create_todo {  
  createTodo(text: "My new todo") {  
    id  
    text  
    done  
  }  
}
```



ToDo

Tasks

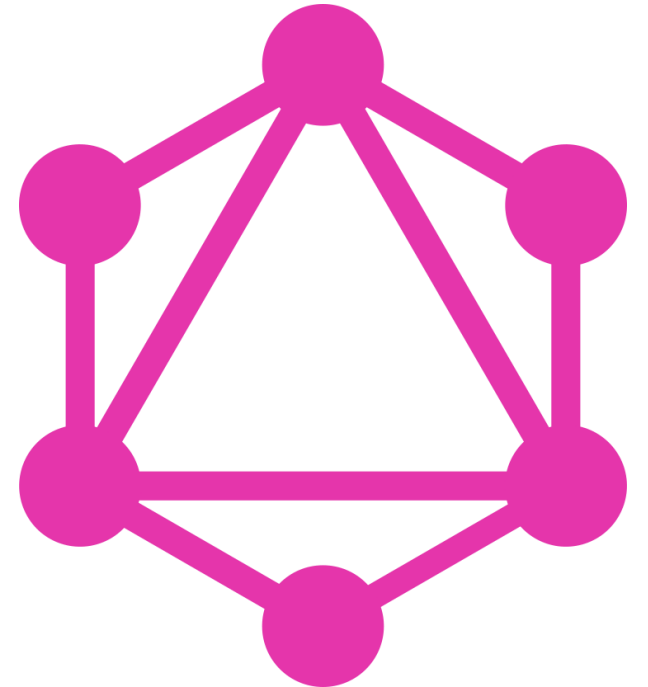
- ☒ A todo not to forget
- ☒ This is the most important
- ☒ Please do this or else
- ☒ Learn GraphQL
- ☐ Create GraphQL changes subscription demo

Add task

Add

Learning GraphQL - <https://graphql.org/learn/>

- **Queries and Mutations** – Fields, Arguments, Aliases, Fragments, Operation Name, Variables, Directives, Mutations, Inline Fragments
- **Schemas and Types** - Type System, Type Language, Object Types and Fields, Arguments, The Query and Mutation Types, Scalar Types, Enumeration Types, Lists and Non-Null, Interfaces, Union Types, Input Types
- **Validation**
- **Execution**
- **Introspection**



Untitled Query 1

GraphQL Endpoint	http://localhost:8080/graphql
------------------	-------------------------------

Method GET ▼

Edit HTTP Headers

GraphiQL



Prettify

History

```

52 ▾ query Luke {
53 ▾   human(id: "1000") {
54     name
55     # Queries can have comments!
56 ▾   friends {
57     id
58     name
59     appearsIn
60   }
61 }
62 }
63
64 ▾ {
65   leftComparison: hero(episode: EMPIRE) {
66     ...comparisonFields
67   }
68   rightComparison: hero(episode: JEDI) {
69     ...comparisonFields
70   }
71 }
72 ▾ fragment comparisonFields on Character {
73   name
74   appearsIn
75   friends {
76     id
77     name
78   }
79 }
80 ▾ {
81   empireHero: hero(episode: EMPIRE) {
82     name
83   }
84   jediHero: hero(episode: JEDI) {
85     name
86   }
87 }
88
89 ▾ query Hero($episode: Episode!, $withFriends: Boolean = false) {
90 ▾   hero(episode: $episode) {
91     name
92     friends @include(if: $withFriends) {
93       name
94     }
95   }
96 }

```

```
{
  "data": {
    "leftComparison": {
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "id": "1002",
          "name": "Han Solo"
        },
        {
          "id": "1003",
          "name": "Leia Organa"
        },
        {
          "id": "2000",
          "name": "C-3PO"
        },
        {
          "id": "2001",
          "name": "R2-D2"
        }
      ],
      "name": "Luke Skywalker"
    },
    "rightComparison": {
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "id": "1000",
          "name": "Luke Skywalker"
        },
        {
          "id": "1002",
          "name": "Han Solo"
        },
        {
          "id": "1003",
```

[← Schema](#)

Query



🔍 Search Query...

No Description

FIELDS

```
droid(id: String!): Droid
```

```
hero(episode: Episode): Character
```

```
human(id: String!): Human
```

QUERY VARIABLES

GraphiQL



Prettify

History

```

1 query GetAllBooks{
2   list{
3     id
4     name
5     description
6     price
7   }
8 }
9
10 query GetBookByID($id: String!){
11   bookById(id: $id){
12     id
13     name
14     description
15     price
16   }
17 }
18
19 mutation Create($name: String!, $description: String, $price: Float!){
20   create(name: $name, description: $description, price: $price) {
21     id
22     name
23     description
24     price
25   }
26 }
27
28 mutation Update($id: String!, $name: String!, $description: String, $price:
29   update(id:$id, name: $name, description: $description, price: $price) {
30     id
31     name
32     description
33   }

```

QUERY VARIABLES

```
1 {
2   "id": "2e9c0af6-1be6-4a31-a717-408290d36c42",
3   "name": "GraphQL with Golang is Easy",
4   "description": "Implementing GraphQL endpoints in Go ...",
5   "price": 125.99
6 }
```

```
{
  "data": {
    "list": [
      {
        "description": "Good Book for GraphQL",
        "id": "2ff788f6-b82c-4b9e-b6a9-871b96b1ad11",
        "name": "GraphQL Basics",
        "price": 35
      },
      {
        "description": "Implementing GraphQL endpoints in Go ...",
        "id": "c3e83d60-642b-4f5f-8825-eca635343021",
        "name": "GraphQL with Golang 1",
        "price": 55.5
      },
      {
        "description": "Implementing GraphQL endpoints in Go ...",
        "id": "51f21351-4066-475f-ad3e-51b40f5d7131",
        "name": "GraphQL with Golang is Easy",
        "price": 125.99
      }
    ]
  }
}
```

[← Schema](#)

Mutation



🔍 Search Mutation...

No Description

FIELDS

```
create(  
    price: Float!  
    description: String  
    name: String!  
): Book
```

Create new book

```
delete(id: String!): Book
```

Delete book by name

```
update(  
    id: String!  
    name: String!  
    price: Float  
    description: String  
): Book
```

Update book by name

GraphiQL



Prettify

History

```

11  bookByID($id: String!){
12      id
13      name
14      description
15      price
16  }
17  }
18
19  mutation Create($name: String!, $description: String, $price: Float!){
20      create(name: $name, description: $description, price: $price) {
21          id
22          name
23          description
24          price
25      }
26  }
27
28  mutation Update($id: String!, $name: String!, $description: String, $price:
29      Float!){
30      update(id: $id, name: $name, description: $description, price: $price) {
31          id
32          name
33          description
34          price
35      }
36  }
37
38  mutation DeleteBookByID($id: String!){
39      delete(id: $id) {
40          id
41          name
42          description
43          price
44      }
45  }

```

```
{
  "data": {
    "create": {
      "description": "Implementing GraphQL endpoints in Go ...",
      "id": "",
      "name": "GraphQL with Golang is Easy",
      "price": 75.99
    }
  }
}
```

QUERY VARIABLES

```
1 {
2   "id": "2e9c0af6-1be6-4a31-a717-408290d36c42",
3   "name": "GraphQL with Golang is Easy",
4   "description": "Implementing GraphQL endpoints in Go ...",
5   "price": 75.99
6 }
```

[← Schema](#)

Mutation



🔍 Search Mutation...

No Description

FIELDS

```
create(  
    price: Float!  
    description: String  
    name: String!  
): Book
```

Create new book

```
delete(id: String!): Book
```

Delete book by name

```
update(  
    id: String!  
    name: String!  
    price: Float  
    description: String  
): Book
```

Update book by name

History

```

query GetSchema {
  __schema {
    types {
      name
      fields {
        name
        description
      }
    }
  }
}

query QueryAndMutationTypes {
  __schema {
    queryType {
      name
      fields {
        name
      }
    }
    mutationType {
      name
      fields {
        name
      }
    }
  }
}

query GetType {
  __type(name: "Todo") {
    fields {
      name
      description
    }
  }
}

```

```
{
  "data": {
    "__schema": {
      "mutationType": {
        "fields": [
          {
            "name": "create"
          },
          {
            "name": "delete"
          },
          {
            "name": "update"
          }
        ],
        "name": "Mutation"
      },
      "queryType": {
        "fields": [
          {
            "name": "bookById"
          },
          {
            "name": "bookByName"
          },
          {
            "name": "list"
          }
        ],
        "name": "Query"
      }
    }
  }
}
```

Mutation

>

🔍 Search Mutation...

No Description

FIELDS

```
create(  
    price: Float!  
    description: String  
    name: String!  
): Book
```

Create new book

```
delete(id: String!): Book
```

Delete book by name

```
update(  
    id: String!  
    name: String!  
    price: Float  
    description: String  
): Book
```

Update book by name

QUERY VARIABLES

GraphQL in Go



GraphQL Starter Example

```
import ("net/http"; "github.com/graphql-go/graphql"; gqhandler "github.com/graphql-go/graphql-go-handler")
var queryType = graphql.NewObject(graphql.ObjectConfig { // ... })
var Schema, _ = graphql.NewSchema(graphql.SchemaConfig{ // ... })

func main() {
    // create a graphl-go HTTP handler with our previously defined schema
    // and we also set it to return pretty JSON output
    h := gqhandler.New(&gqhandler.Config{
        Schema: &Schema,
        Pretty: true,
    })

    // serve a GraphQL endpoint at `/graphql`
    http.Handle("/graphql", h)

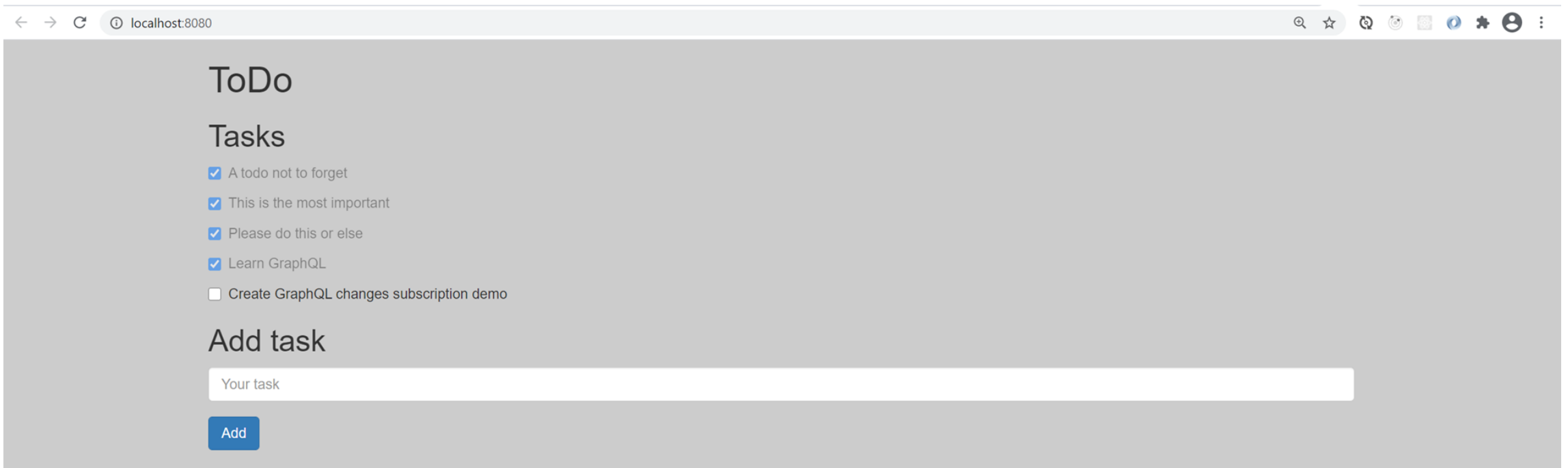
    // and serve!
    http.ListenAndServe(":8080", nil)
}
```

GraphQL Schema Configuration – Code First

```
var queryType = graphql.NewObject(graphql.ObjectConfig{
    Name: "RootQuery",
    Fields: graphql.Fields{
        "latestPost": &graphql.Field{
            Type: graphql.String,
            Resolve: func(p graphql.ResolveParams) (interface{}, error) {
                return "Hello World!", nil
            },
        },
        "postsCount": &graphql.Field {
            Type: graphql.Int,
            Resolve: func(p graphql.ResolveParams) (interface{}, error) {
                return rand.Intn(100), nil
            },
        },
    },
})

var Schema, _ = graphql.NewSchema(graphql.SchemaConfig{
    Query: queryType,
})
```


GraphQL TODOs Example



← → ↻ ⓘ localhost:8080 🔍 ☆ 🔄 🤖 📱 🌐 ⚙️ 👤 ⋮

ToDo

Tasks

- ☒ A todo not to forget
- ☒ This is the most important
- ☒ Please do this or else
- ☒ Learn GraphQL
- ☐ Create GraphQL changes subscription demo

Add task

GraphQL Todos: Model

```
package model
```

```
var TodoList []Todo
```

```
type Todo struct {  
    ID   string `json:id`  
    Text string `json:text`  
    Done bool   `json:done`  
}
```

GraphQL Todos: Schema Objects

```
var todoType = graphql.NewObject(graphql.ObjectConfig{
    Name: "Todo",
    Fields: graphql.Fields{
        "id": &graphql.Field{
            Type: graphql.String,
        },
        "text": &graphql.Field{
            Type: graphql.String,
        },
        "done": &graphql.Field{
            Type: graphql.Boolean,
        },
    },
})
```

GraphQL Todos: RootQuery

```
var rootQuery = graphql.NewObject(graphql.ObjectConfig{
    Name: "RootQuery",
    Fields: graphql.Fields{
        "list": &graphql.Field{
            Type:      graphql.NewList(todoType),
            Description: "list of all todos",
            Resolve: func(params graphql.ResolveParams) (interface{}, error) {
                return model.TODOList, nil
            },
        },
    },
})
```

GraphQL Todos: RootQuery + Field Arguments

```
"todo": &graphql.Field{
    Type:      todoType,
    Args: graphql.FieldConfigArgument{
        "id": &graphql.ArgumentConfig{
            Type: graphql.NewNonNull(graphql.String),
        },
    },
    Resolve: func(params graphql.ResolveParams) (interface{}, error) {
        idQuery, ok := params.Args["id"].(string)
        if ok {
            for _, todo := range model.TODOList {
                if todo.ID == idQuery {
                    return todo, nil
                }
            }
        }
        return nil, fmt.Errorf("TODO with id='%s' not found", params.Args["id"])
    },
},
```

GraphQL Todos: RootQuery + Field Arguments

```
"todo": &graphql.Field{
    Type:      todoType,
    Args: graphql.FieldConfigArgument{
        "id": &graphql.ArgumentConfig{
            Type: graphql.NewNonNull(graphql.String),
        },
    },
    Resolve: func(params graphql.ResolveParams) (interface{}, error) {
        idQuery, ok := params.Args["id"].(string)
        if ok {
            for _, todo := range model.TODOList {
                if todo.ID == idQuery {
                    return todo, nil
                }
            }
        }
        return nil, fmt.Errorf("TODO with id='%s' not found", params.Args["id"])
    },
},
```

GraphQL Todos: RootMutation – Create TODO – I

```
var rootMutation = graphql.NewObject(graphql.ObjectConfig{
    Name: "RootMutation",
    Fields: graphql.Fields{
        "create": &graphql.Field{
            Type:      todoType,
            Description: "Create new todo.",
            Args: graphql.FieldConfigArgument{
                "text": &graphql.ArgumentConfig{
                    Type: graphql.NewNonNull(graphql.String),
                },
            },
        },
    },
})
```

GraphQL Todos: RootMutation – Create TODO – II

```
Resolve: func(params graphql.ResolveParams) (interface{}, error) {  
    text, ok := params.Args["text"].(string)  
    if ok {  
        newTodo := model.TODO{  
            ID:    uuid.New().String(),  
            Text: text,  
            Done: false,  
        }  
        model.TODOList = append(model.TODOList, newTodo)  
        return newTodo, nil  
    }  
    return nil, fmt.Errorf("error updating todo: %v", params.Args["text"])  
},  
,
```

GraphQL Todos: RootMutation – Update TODO – I

```
"update": &graphql.Field{
    Type:      todoType,
    Description: "Update todo done status.",
    Args: graphql.FieldConfigArgument{
        "id": &graphql.ArgumentConfig{
            Type: graphql.NewNonNull(graphql.String),
        },
        "done": &graphql.ArgumentConfig{
            Type: graphql.NewNonNull(graphql.Boolean),
        },
    },
},
```

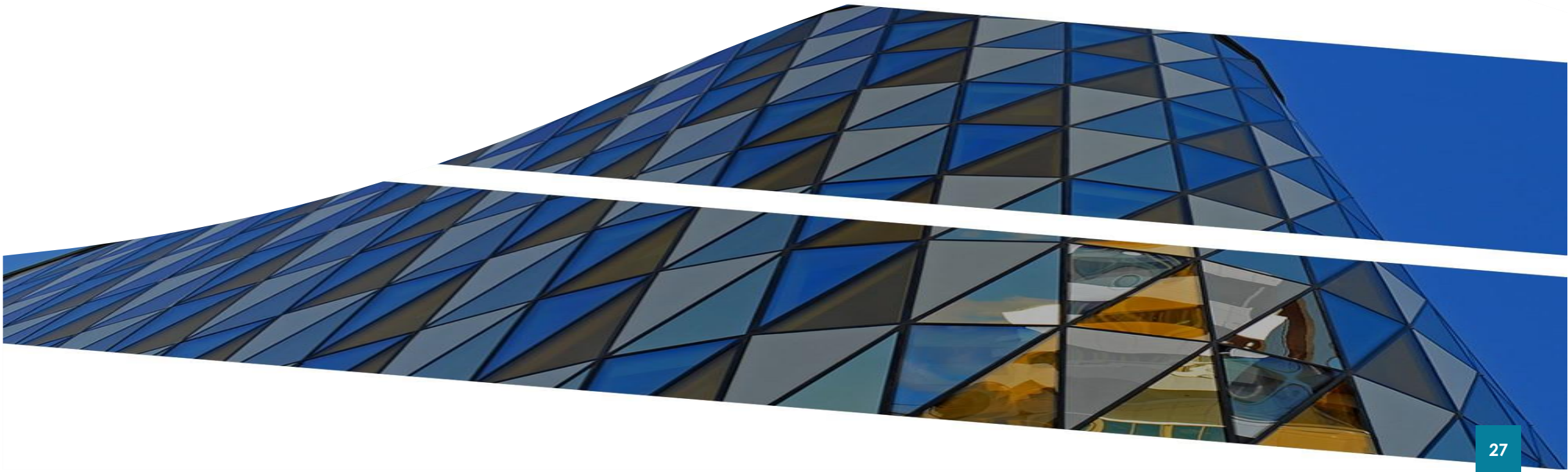

GraphQL Todos: RootMutation – Update TODO – II

```
Resolve: func(params graphql.ResolveParams) (interface{}, error) {
    id, ok1 := params.Args["id"].(string)
    done, ok2 := params.Args["done"].(bool)
    if ok1 && ok2 {
        // Search list for todo with id and change the done status
        for i := 0; i < len(model.TODOList); i++ {
            if model.TODOList[i].ID == id {
                model.TODOList[i].Done = done
                return model.TODOList[i], nil
            }
        }
        return nil, fmt.Errorf("error updating todo: id='%s' not found", id)
    }
    return nil, fmt.Errorf("error updating todo with id='%v'", params.Args["id"])
},
},
})
```

GraphQL Todos: Schema

```
var TodoSchema, _ = graphql.NewSchema(graphql.SchemaConfig{  
  
    Query:    rootQuery,  
  
    Mutation: rootMutation,  
  
    })
```

More examples



There is More About GraphQL ...

- GraphQL subscriptions with Go, GQLgen and MongoDB –
<https://www.freecodecamp.org/news/https-medium-com-anshap1719-graphql-subscriptions-with-go-gqlgen-and-mongodb-5e008fc46451/>,
<https://www.inovex.de/blog/graphql-application-golang-tutorial/>
- Generating Golang code with GQLgen (schema-first approach) –
<https://github.com/99designs/gqlgen>, <https://gqlgen.com/getting-started/>,
<https://dev.to/stevensunflash/using-graphql-in-golang-3gg0>
- Using Golang Relay GraphQL server implementation –
<https://relay.dev/>, <https://github.com/graphql-go/relay>,
<https://github.com/sogko/golang-relay-starter-kit>

Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>