



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.

GraphiQL Client

- Download GraphiQL Client at:
<https://github.com/skevy/graphiql-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

Your task

Add



22:11

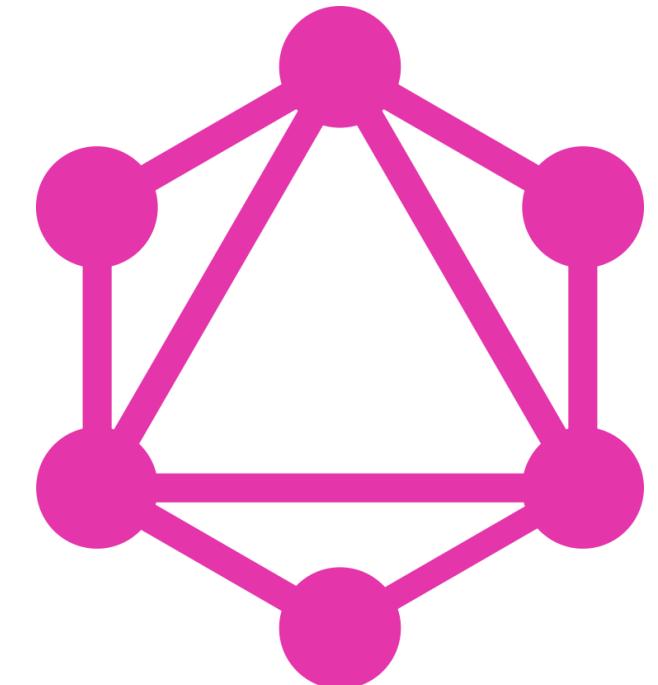
ENG

20.2.2021 r.



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



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   leftComparison: hero(episode: EMPIRE) {
65     ...comparisonFields
66   }
67   rightComparison: hero(episode: JEDI) {
68     ...comparisonFields
69   }
70 }
71 fragment comparisonFields on Character {
72   name
73   appearsIn
74   friends {
75     id
76     name
77   }
78 }
79 {
80   empireHero: hero(episode: EMPIRE) {
81     name
82   }
83   jediHero: hero(episode: JEDI) {
84     name
85   }
86 }
87
88 query Hero($episode: Episode!, $withFriends: Boolean = false) {
89   hero(episode: $episode) {
90     name
91     friends @include(if: $withFriends) {
92       name
93     }
94   }
95 }

```

QUERY VARIABLES

```

{
  "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",
          "name": "Leia Organa"
        }
      ]
    }
  }
}

```

Schema

Query



Search Query...

No Description

FIELDS

droid(id: String!): Droid

hero(episode: Episode): Character

human(id: String!): Human



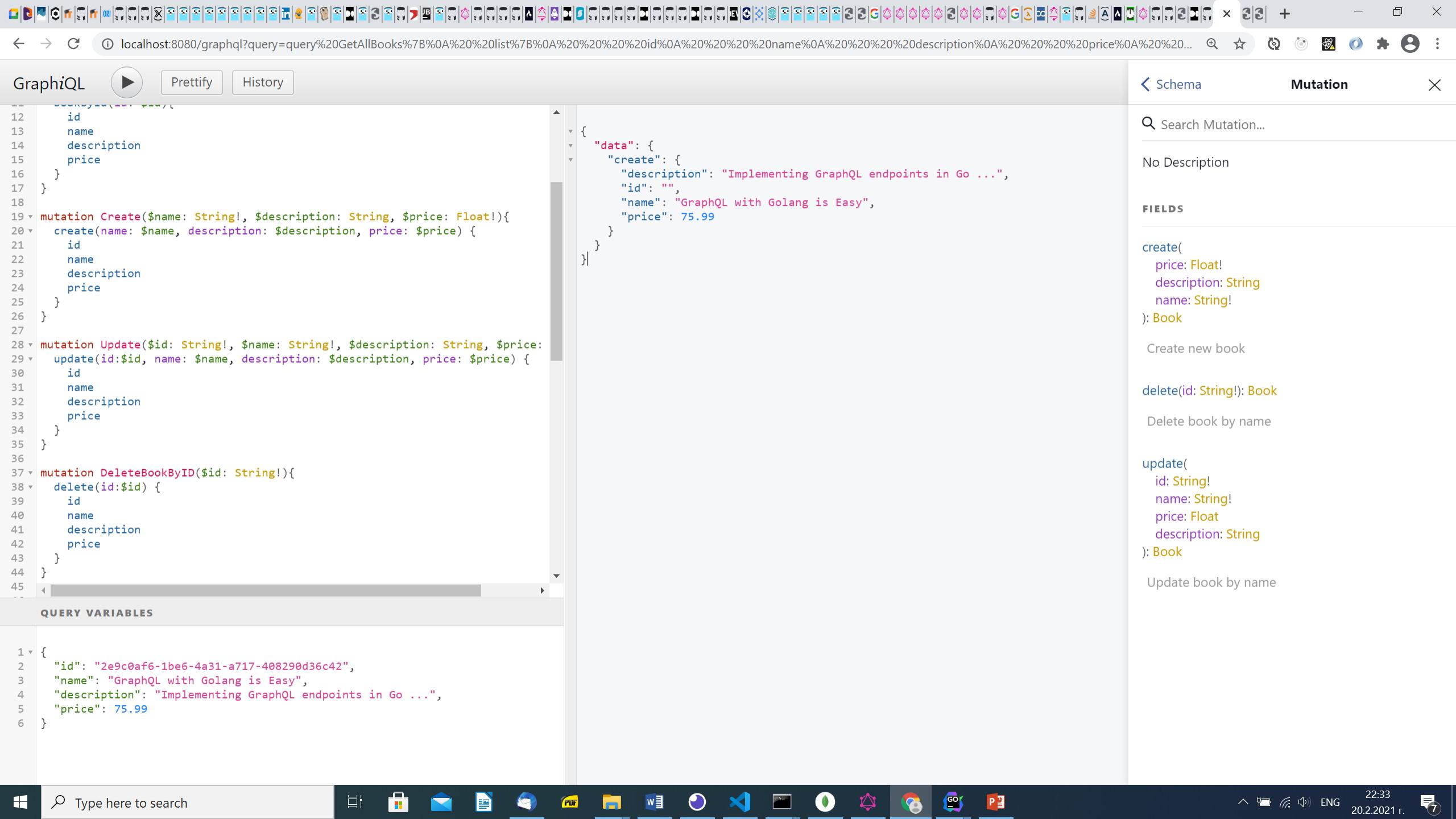
Type here to search



23:05

20.2.2021 r.





The screenshot shows a GraphQL playground interface with the URL `localhost:8080/graphql?query=query%20GetAllBooks%7B%0A%20%20list%7B%0A%20%20%20%20%20id%0A%20%20%20name%0A%20%20%20description%0A%20%20%20price%0A%20%20%20%7D%7D`. The left panel displays the schema code, and the right panel shows the generated mutation types.

GraphQL Schema:

```
39 id
40   name
41   description
42   price
43 }
44 }
45
46 query GetSchema {
47   __schema {
48     types {
49       name
50       fields {
51         name
52         description
53       }
54     }
55   }
56 }
57
58 query QueryAndMutationTypes{
59   __schema {
60     queryType {
61       name
62       fields {
63         name
64       }
65     }
66     mutationType {
67       name
68       fields {
69         name
70       }
71     }
72   }
73 }
74
75 query GetType{
76   __type(name:"Todo"){
77     fields {
78       name
79       description
80     }
81   }
82 }
```

Mutation Types:

- 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

Type here to search

22:32 20.2.2021 r. ENG

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

The screenshot shows a web browser window with the URL `localhost:8080` in the address bar. The page title is "ToDo". Below the title, there is a section titled "Tasks" containing a list of checkboxes:

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

Below the tasks, there is a section titled "Add task" with a text input field containing the placeholder "Your task" and a blue "Add" button.

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

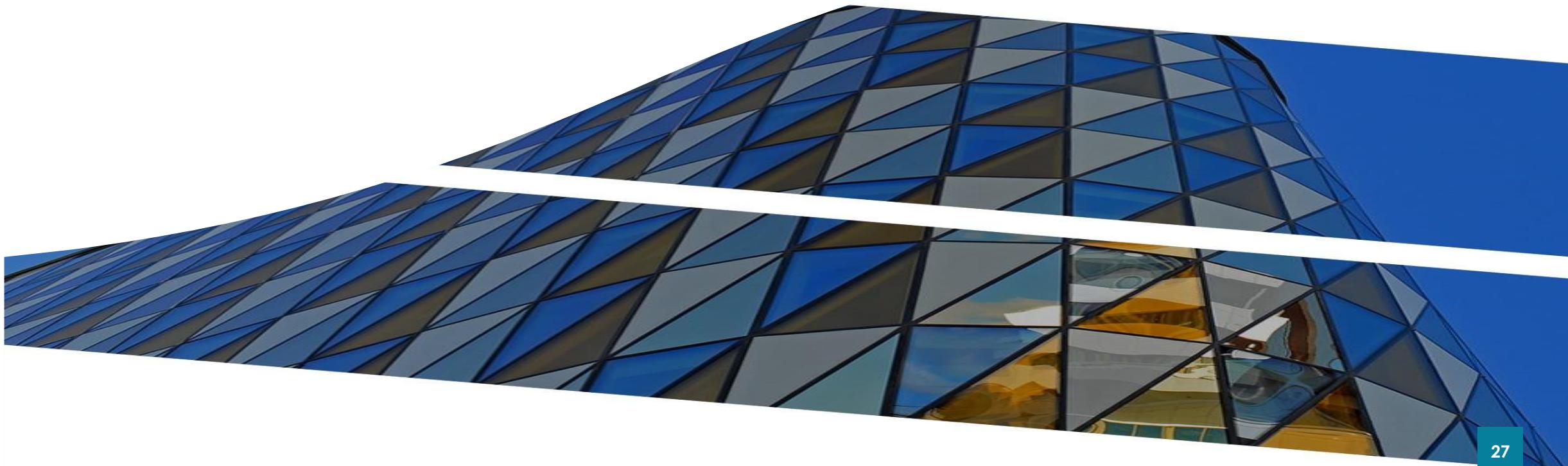
```
var TodoSchema, _ = graphql.NewSchema(graphql.SchemaConfig{
```

```
    Query: rootQuery,
```

```
    Mutation: rootMutation,
```

```
})
```

More examples



There is More About GraphQL ...

- GraphQL subscriptions with Go, GQLgen and MongoDB –
<https://github.com/sgloutnikov/gqlgen-react-chatapp>
<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 GrpahQL 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>