GraphQL

Для бэкендеров

Всем привет

Меня зовут Александр.

Я программист уже более 10 лет, из них последние 5 лет пишу на Go.

Руковожу практикой Go в компании Usetech.

В этом докладе хочу поделиться, почему бэкендерам стоит обратить внимание на GraphQL.

USETECH

GraphQL

Почему это хорошо?

Какие есть варианты?

- REST

Какие есть варианты?

➔ REST

```
GET /projects/asi_help/issues?set_filter=1&tracker_id=4 HTTP/1.1
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ru;q=0.8
Host: mc.usetech.ru
```

Какие есть варианты?

- REST
- ➔ Плюсы
  - Просто
  - Понятно
  - Привычно
  - Много библиотек заточены под рест
  - Сваггер

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны

      issues?milestone=1&state=open&assignee=AlwxSin&creator=AlwxSin&mentioned=Keanu&labels=great_first_issue,bug,documentation,ui,@high&sort=created&direction=asc&since=2024-01-02T01:33:00Z&page=2&per_page=30

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны

      issues?milestone=1&state=open&assignee=AlwxSin&creator=AlwxSin&mentioned=Keanu&labels=great_first_issue,bug,documentation,ui,@high&sort=created&direction=asc&since=2024-01-02T01:33:00.0Z&page=2&per_page=30

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны

      issues?milestone=1&state=open&assignee=AlwxSin&creator=AlwxSin&mentioned=Keanu&labels=great_first_issue,bug,documentation,ui,@high&sort=created&direction=asc&since=2024-01-02T01:33:00.0Z&page=2&per_page=30

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны

      issues?milestone=1&state=open&assignee=AlwxSin&creator=AlwxSin&mentioned=Keanu&labels=great_first_issue,bug,documentation,ui,@high&sort=created&direction=asc&since=2024-01-02T01:33:00.0Z&page=2&per_page=30

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны
    - Нет связей

/repos

/orgs

/issues

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны
    - Нет связей

      /repos/{owner}/{repo}/issues

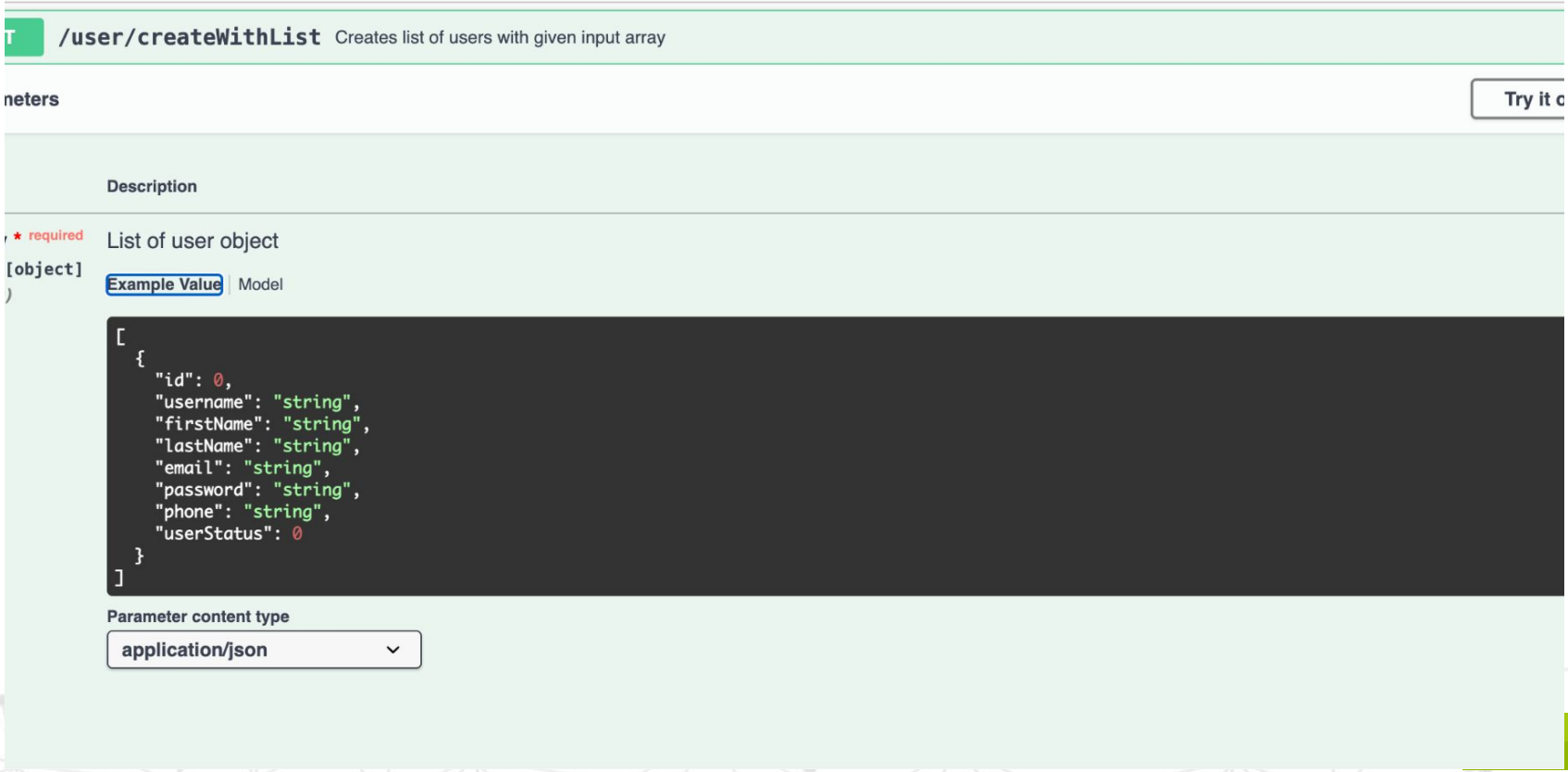
      /orgs/{org}/issues

      /issues

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны
    - Нет связей
    - Нет валидации на уровне схемы данных

Валидация данных



Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны
    - Нет связей
    - Нет валидации на уровне схемы данных
    - Сваггер

Сваггер


```yaml
paths:
/product:
   get:
     tags:
       - manager
       - customer
     summary: 'Посмотреть каталог товаров'
     operationId: viewProductCatalog
     description: 'Параметры фильтрации товаров в каталоге для поиска'
     parameters:
       - in: query
         name: name
         description: 'название товара для поиска'
         required: false
         schema:
           type: string
       - in: query
         name: category
         description: 'категория товара для поиска'
         required: false
         schema:
           type: string
       - in: query
         name: min_price
         description: 'минимальная цена товара для поиска'
         schema:
           type: integer
           format: int32
           minimum: 0
           maximum: 100500
```

Сваггер


```
responses:
    '200':
      description: 'результаты поиска по запросу'
      content:
        application/json:
          schema:
            type: array
            items:
              $ref: '#/components/schemas/Product'
    '400':
      description: 'не верные параметры фильтрации'
```

Какие есть варианты?

- REST
  - Плюсы
  - ➔ Минусы
    - В GET запросе параметры перемешаны
    - Нет связей
    - Нет валидации на уровне схемы данных
    - Сваггер
    - Code-First подход

Какие есть варианты?

- REST
- ➔ gRPC
  - ➔ Плюсы
    - Скорость
    - Размер передаваемых данных
    - Типизация
    - Генерация кода, на выходе вызов функции
    - http 2.0
    - Schema-First подход

Какие есть варианты?

- REST
- ➔ gRPC
  - Плюсы
  - ➔ Минусы
    - Фронтенд не может
    - Нет связей
    - Нужен отдельный репозиторий для протофайлов
    - http 2.0

Какие есть варианты?

- REST
- gRPC
- ➔ GraphQL
  - ➔ Плюсы
    - Типизация
    - Валидация на уровне схемы
    - В клиентах и фронтенд, и бэкенд
    - Независимость от транспорта
    - Связь данных
    - Фронты вас будут любить

Какие есть варианты?

- REST
- gRPC
- ➔ GraphQL
  - Плюсы
  - ➔ Минусы
    - Сложность
    - Размер передаваемых данных
    - Code-first подход

Фронты вас любят


```
input UserInput {
   id: ID!
   username: String!
   firstName: String @assertMaxLen(maxLen: 20)
   lastName: String @assertMaxLen(maxLen: 20)
   email: Email
   password: String!
   phone: Phone
   userStatus: UserStatus!
}

input Email {
   address: String!
}

input Phone {
   code: String!
   number: String!
}

enum UserStatus {
   ACTIVE
   INACTIVE
}
```

Фронты вас любят


```
query {
   me: getMyUser {
       id
       email
       phone
   }
   usersByID: getUsers(filterOptions: {ids: ["user_1", "user_123"]}) {
       results {
           id
           firstName
           lastName
       }
   }
   usersByPhone: getUsers(filterOptions: {phones: [{number: "9162828317", region: "ru"}]}) {
       results {
           id
           region
           isActive
       }
   }
}
```

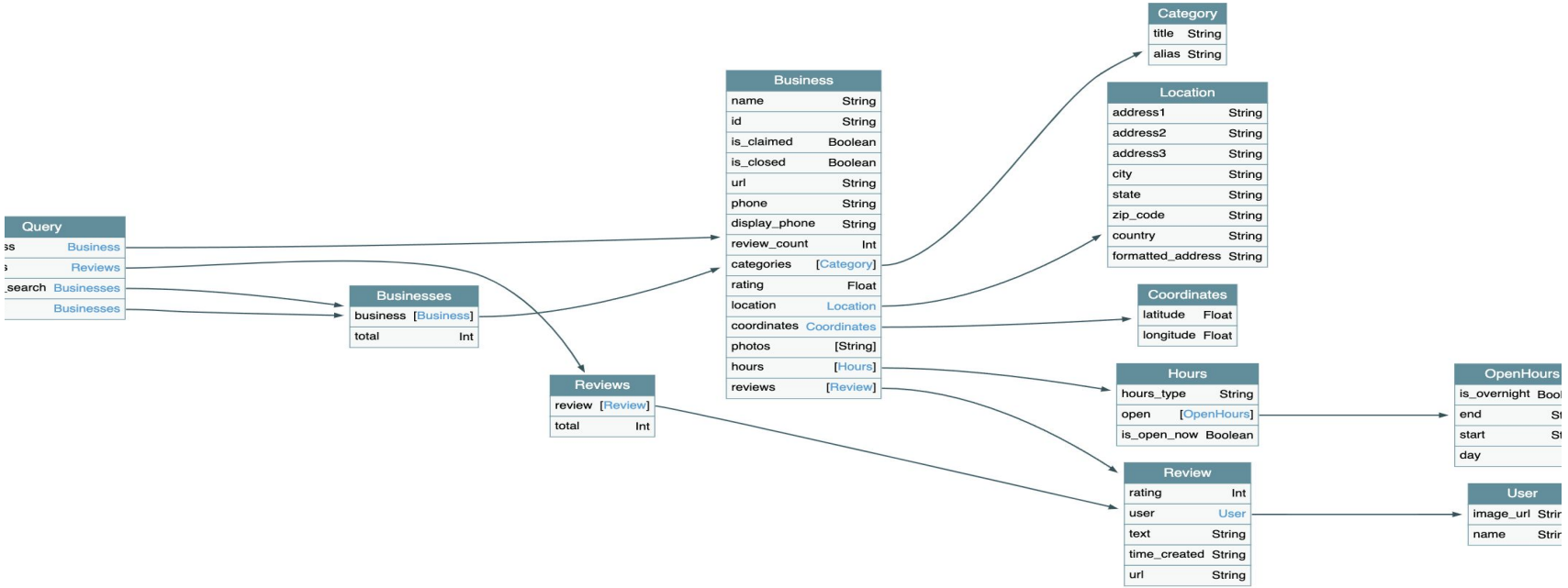
Фронты вас любят


```
query {
    getMyUser { # User
        id
        invitedBy { # User
            firstName
        }
        companies { # Company
            id
            companyUsers { # CompanyUser
                agreementSignedAt
                user { # User
                    id
                }
            }
        }
        ownedExecutors { # Executor
            id
            executiveUser { # User
                id
            }
        }
        projects(paginateOptions: {noPagination: true}) {
            results { # Project
                id
            }
        }
    }
}
```

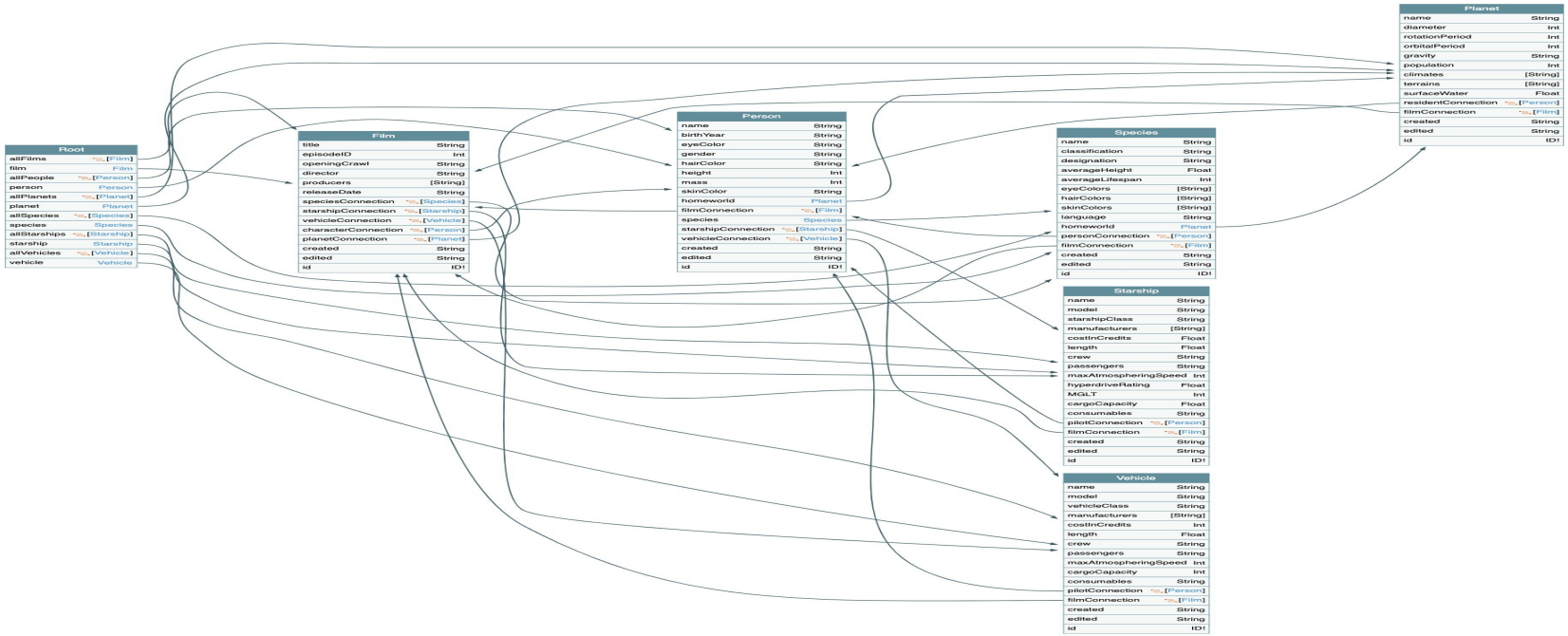
Все это относится и к бэкендерам 😏

Графы

Графы

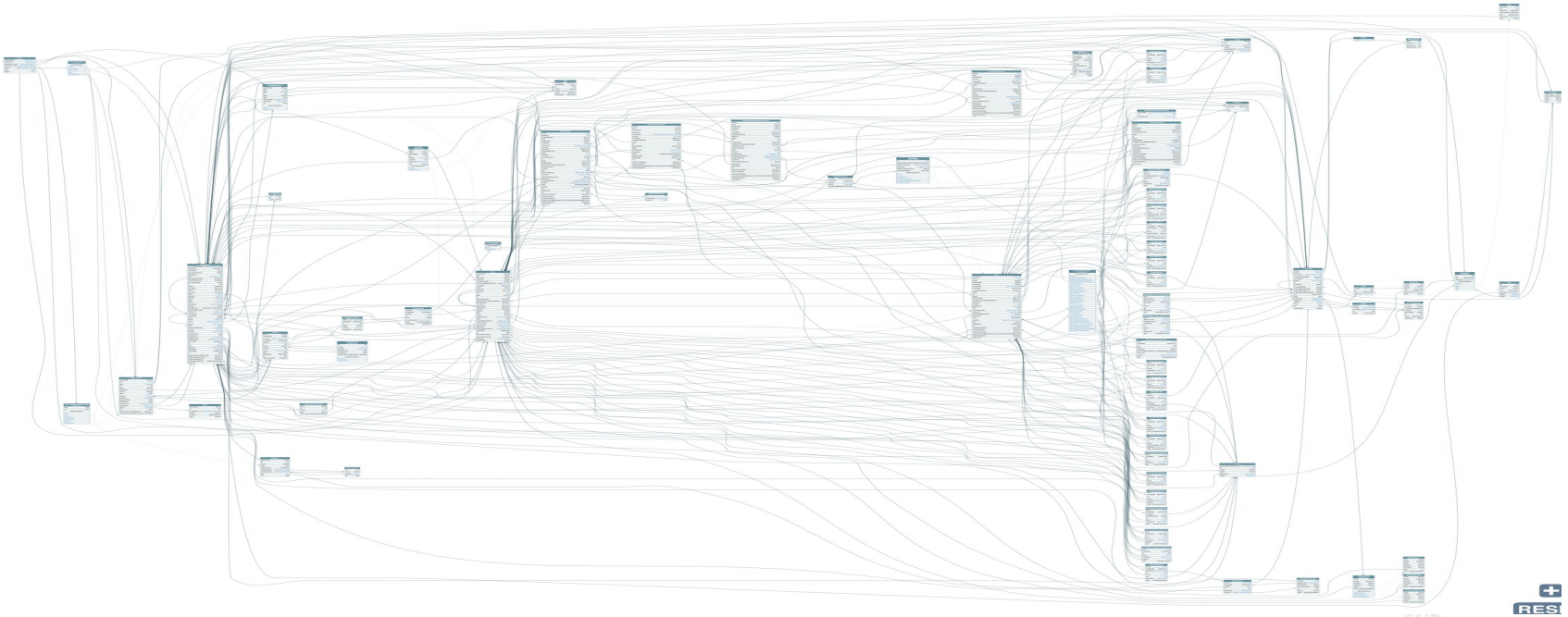

Query
Business
Reviews
search Businesses
Businesses
Businesses
business [Business]
total Int
Reviews
review [Review]
total Int
Business
name String
id String
is_claimed Boolean
is_closed Boolean
url String
phone String
display_phone String
review_count Int
categories [Category]
rating Float
location Location
coordinates Coordinates
photos [String]
hours [Hours]
reviews [Review]
Category
title String
alias String
Location
address1 String
address2 String
address3 String
city String
state String
zip_code String
country String
formatted_address String
Coordinates
latitude Float
longitude Float
Hours
hours_type String
open [OpenHours]
is_open_now Boolean
OpenHours
is_overnight Bool
end St
start St
day
Review
rating Int
user User
text String
time_created String
url String
User
image_url Strin
name Strin

Графы



USETECH

Графы



Типизация

Типизация и генерация


```
type User {
    id: ID!
    email: String!
    firstName: String
    age: Int
    friends: [User!]!
}
```

Типизация и генерация


```
type User {
    id: ID!
    email: String!
    firstName: String
    age: Int
    friends: [User!]!
}
```

```
export type Scalars = {
 ID: { input: string; output: string; }
 String: { input: string; output: string; }
 Boolean: { input: boolean; output: boolean; }
 Int: { input: number; output: number; }
 Float: { input: number; output: number; }
};

export type User = {
 __typename?: 'User';
 id: Scalars['ID']['output'];
 email: Scalars['String']['output'];
 firstName?: Maybe<Scalars['String']['output']>;
 age?: Maybe<Scalars['Int']['output']>;
 friends: Array<User>;
};
```

Типизация и генерация


```
type User {
    id: ID!
    email: String!
    firstName: String
    age: Int
    friends: [User!]!
}
```

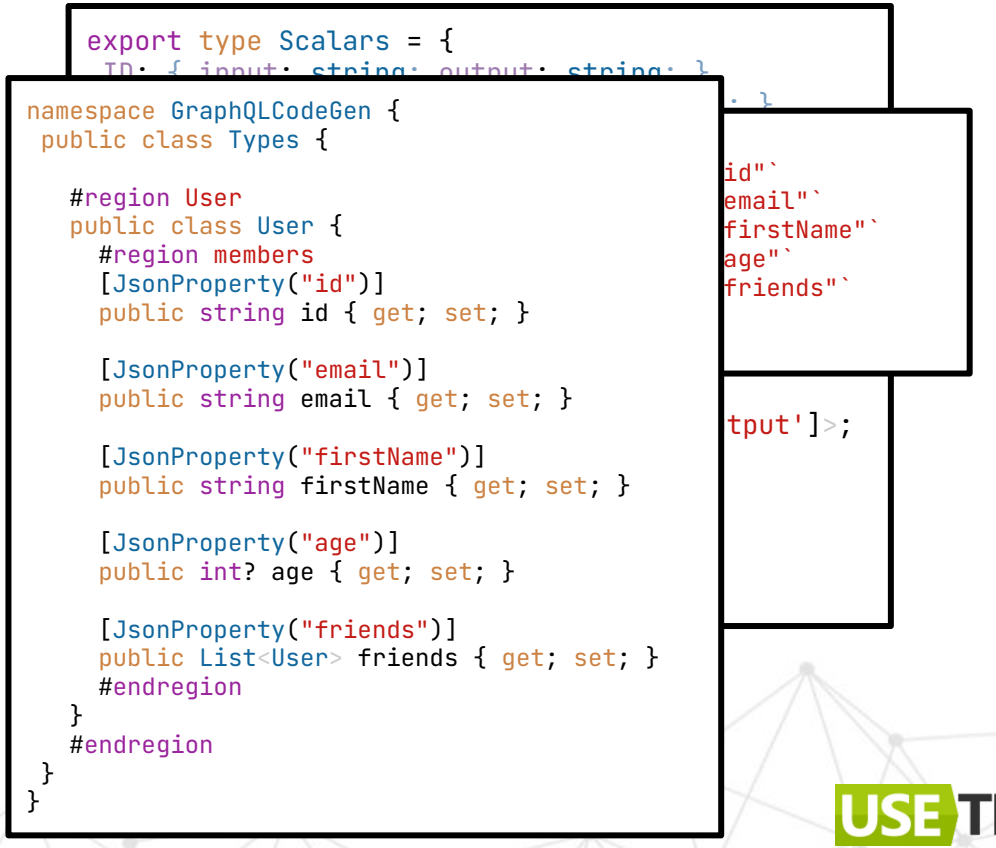
```
export type Scalars = {
 ID: { input: string; output: string; }
 String: { input: string; output: string; }
 Boolean: { inp
 Int: { input:
 Float: { input
};

export type Use
 __typename?: '
 id: Scalars['I
 email: Scalars['String']['output'];
 firstName?: Maybe<Scalars['String']['output']>;
 age?: Maybe<Scalars['Int']['output']>;
 friends: Array<User>;
};
```

```
type User struct {
 ID        uint64   `json:"id"`
 Email     string   `json:"email"`
 FirstName string   `json:"firstName"`
 Age       *uint64  `json:"age"`
 Friends   []User   `json:"friends"`
}
```

USETECH

Типизация и генерация

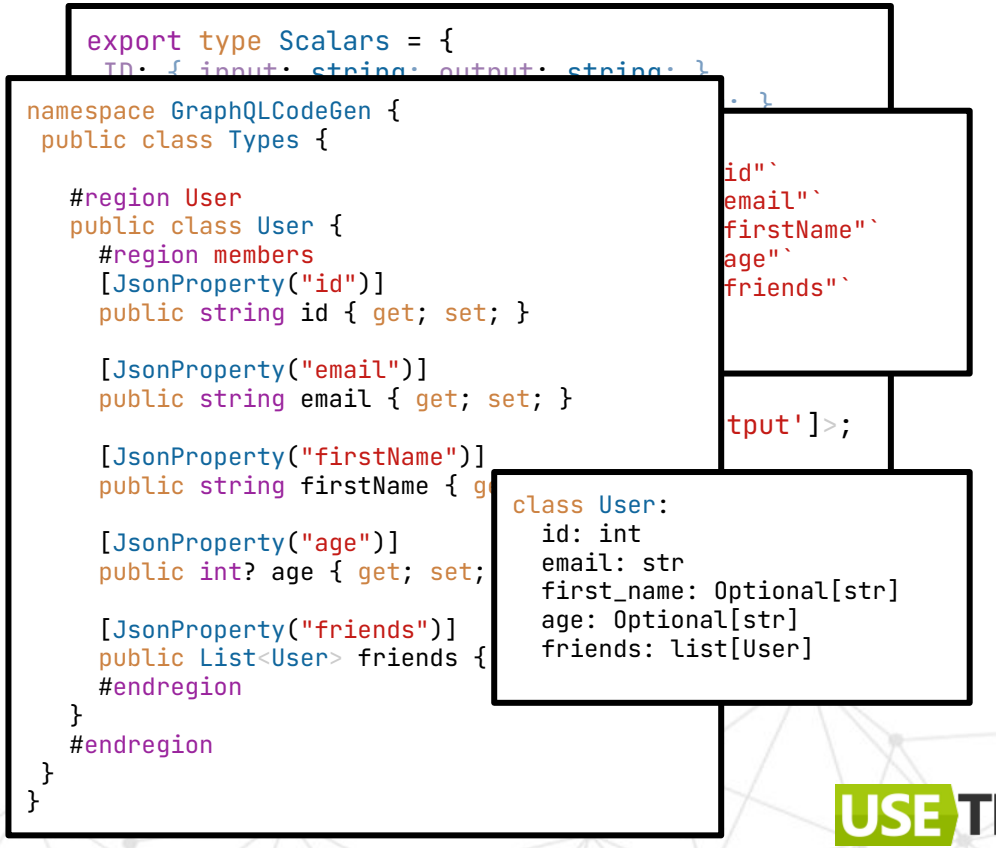

```
type User {
    id: ID!
    email: String!
    firstName: String
    age: Int
    friends: [User!]!
}
```



USETECH

Типизация и генерация


```
type User {
    id: ID!
    email: String!
    firstName: String
    age: Int
    friends: [User!]!
}
```



USETECH

Типизация и генерация


```
type Query {

 getUsers (
   filterOptions: GetUsersFilterOptions!,
   paginateOptions: PaginateOptions!
 ): UserListPaginated!  @hasPermission(permission: USER_LIST)

 getMyUser: User! @isAuthenticated

}
```

Типизация и генерация


```
type Query {

 getUsers (
   filterOptions: Ge
   paginateOptions:
 ): UserListPaginate

 getMyUser: User! @i

}
```

```go
func (r *queryResolver) GetMyUser(ctx context.Context) (*models.User, error) {
  s := scope.GetScope(ctx)
  opts := &models.GetUsersOptions{
     FilterOptions: models.GetUsersFilterOptions{
        Ids: []*gql.GlobalID{{
           ID:     s.User.ID,
           Entity: string(src.UserEntity),
        }},
     },
     PaginateOptions: gql.PaginateOptions{
        NoPagination: true,
     },
  }
  users, err := models.GetUsers(ctx, r.DB, opts)
  if err != nil {
     return nil, err
  }
  return users.Results[0], nil
}
```

USETECH

Типизация и генерация


```
type Query {

 getUsers (
   filterOptions: Ge
   paginateOptions:
 ): UserListPaginate

 getMyUser: User! @i

}
```

```
func (r *queryResolver) GetMyUser(ctx context.Context) (*models.User, error) {
  s := scope.GetScope(ctx)
  opts := &models.GetUsersOptions{
     Filt
        I
        }
     },
     Pagi
        N
     },
  }
  users,
  if err
     retu
  }
  return
}
```

```
func (r *queryResolver) GetUsers(
  ctx context.Context,
  filterOptions models.GetUsersFilterOptions,
  paginateOptions gql.PaginateOptions,
) (*models.UserListPaginated, error) {
  opts := &models.GetUsersOptions{
     FilterOptions:   filterOptions,
     PaginateOptions: paginateOptions,
  }

  users, err := models.GetUsers(ctx, r.DB, opts)
  return users, err
}
```

USETECH

Готовим graphql в go

Готовим graphql в go


```
schema:
 - rootFolder/graph/*.graphqls
exec:
 filename: rootFolder/graph/generated/generated.go
 package: generated
resolver:
 layout: follow-schema
 dir: rootFolder/graph
 package: graph
autobind:
 - "rootFolder/src/models"
model:
 filename: rootFolder/models/models_gen.go
 package: models
models:
 ID:
   model:
     - github.com/99designs/gqlgen/graphql.ID
 Int:
   model:
     - github.com/99designs/gqlgen/graphql.Int
     - github.com/99designs/gqlgen/graphql.Int64
     - github.com/99designs/gqlgen/graphql.Int32
 Map:
   model: example/rootFolder/models.Map
 UUID:
   model: example/rootFolder/models.UUID
```

Готовим graphql в go


```
scalar Email
scalar Phone
scalar Time
scalar Map
scalar UUID


type Query {
   getUsers: [User!]!
}

type Mutation {
   createUser(input: UserInput!): User!
}

type User {
   id: ID!
   name: String
   email: Email!
   phone: Phone!
   birthDate: Time
   analyticsID: UUID!
}
```

Готовим graphql в go


```go
// CreateUser is the resolver for the createUser field.
func (r *mutationResolver) CreateUser(
    ctx context.Context,
    input models.UserInput,
) (*models.User, error) {
    // business logic
}

// GetUsers is the resolver for the getUsers field.
func (r *queryResolver) GetUsers(ctx context.Context) ([]*models.User, error) {
    // business logic
}
```

Готовим graphql в go


```
func Handler(config GqlConfig) http.Handler {
    srv := handler.New(
        generated.NewExecutableSchema(generated.Config{
            Resolvers: &Resolver{},
        }),
    )

    setContextFn := func(ctx context.Context, r *http.Request) context.Context {
        return context.WithValue(ctx, "requestScope", &RequestScope{})
    }

    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        ctx := setContextFn(r.Context(), r)
        srv.ServeHTTP(w, r.WithContext(ctx))
    })
}
```

Готовим graphql в go


```
type Query {
   me: MeQuery!
   public: PublicQuery!
   admin: AdminQuery!
}

type MeQuery {
   me: User!
}

type PublicQuery {
   getUsers: [PublicUser!]!
}

type AdminQuery {
   getUsers(filterOptions: GetUsersFilterOptions!): [User!]!
   deleteUser(userID: ID!): Boolean
}
```

Готовим graphql в go


```go
// GetUsers is the resolver for the getUsers field.
func (r *adminQueryResolver) GetUsers(
      ctx context.Context, obj *models.Query, filterOptions models.GetUsersFilterOptions
) ([]*models.User, error) {
   // business logic
}

// DeleteUser is the resolver for the deleteUser field.
func (r *adminQueryResolver) DeleteUser(ctx context.Context, obj *models.Query, userID string) (*bool, error)
{
   // business logic
}

// Me is the resolver for the me field.
func (r *meQueryResolver) Me(ctx context.Context, obj *models.Query) (*models.User, error) {
   // business logic
}

// CreateUser is the resolver for the createUser field.
func (r *mutationResolver) CreateUser(ctx context.Context, input models.UserInput) (*models.User, error) {
   // business logic
}

// GetUsers is the resolver for the getUsers field.
func (r *publicQueryResolver) GetUsers(ctx context.Context, obj *models.Query) ([]*models.PublicUser, error)
{
   // business logic
}
```

Готовим graphql в go


```
models:
 Me:
   model: example/rootFolder/models.Query
 MeQuery:
   model: example/rootFolder/models.Query
 PublicQuery:
   model: example/rootFolder/models.Query
 AdminQuery:
   model: example/rootFolder/models.Query
```

Готовим graphql в go

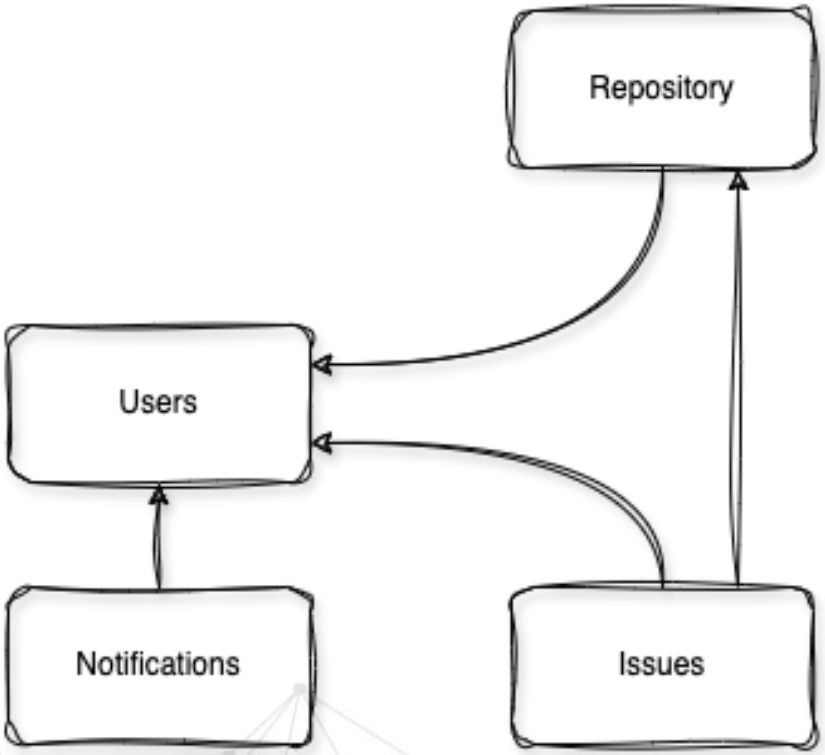

```
type Subscription {
   notifications(userID: ID!): [Notification!]!
}

type Notification {
   user: User!
   message: String!
}
```

Готовим graphql в go

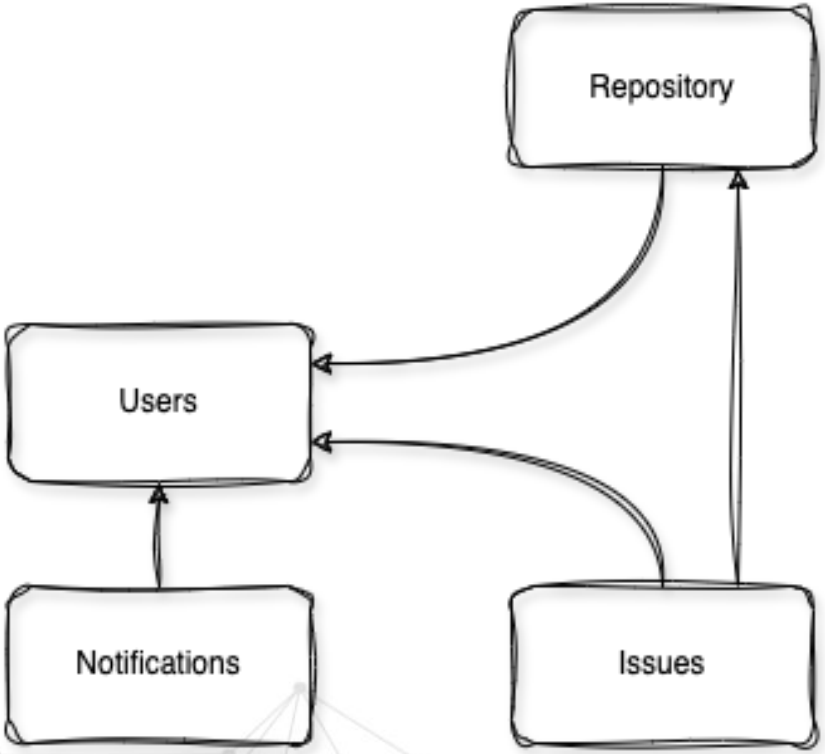

```
func (r *subscriptionResolver) Notifications(
      ctx context.Context, userID string,
) (<-chan []*models.Notification, error) {

   ch := make(chan []*models.Notification)

   go func() {
       for {
           time.Sleep(1 * time.Second)
           t := []*models.Notification{
               {Message: fmt.Sprintf("message with current time %s", time.Now())},
           }
           select {
           case ch <- t: // actual send.
               // Our message went through, do nothing
           }
       }
   }()

   return ch, nil
}
```

Федерация

Федерация

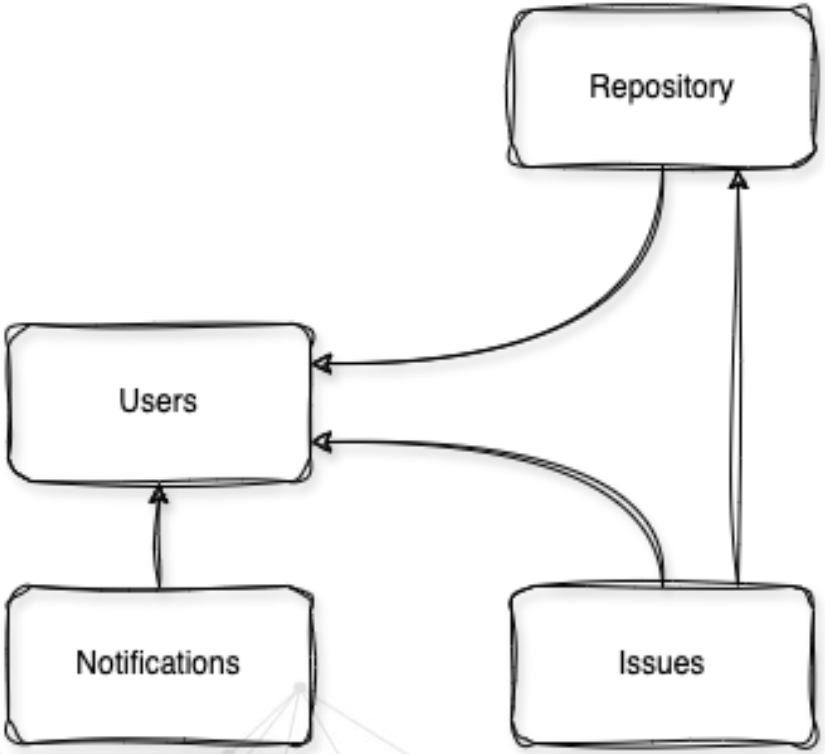


Федерация



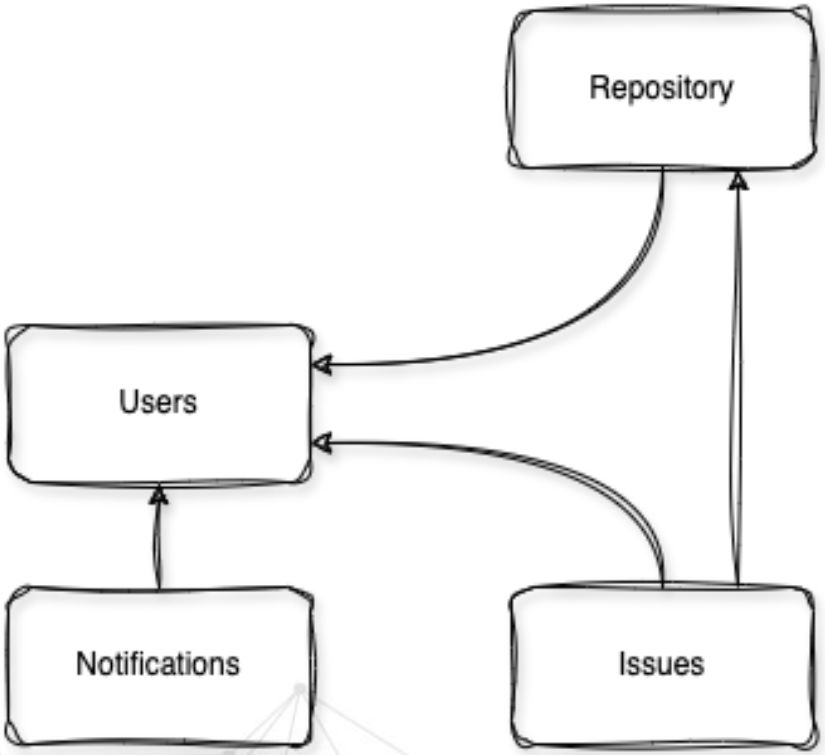

```
users/me
repositories?user_id={user_id}
commits?repository_id={repository_id}
issues?repository_id={repository_id}
notifications?user_id={user_id}
```

Федерация



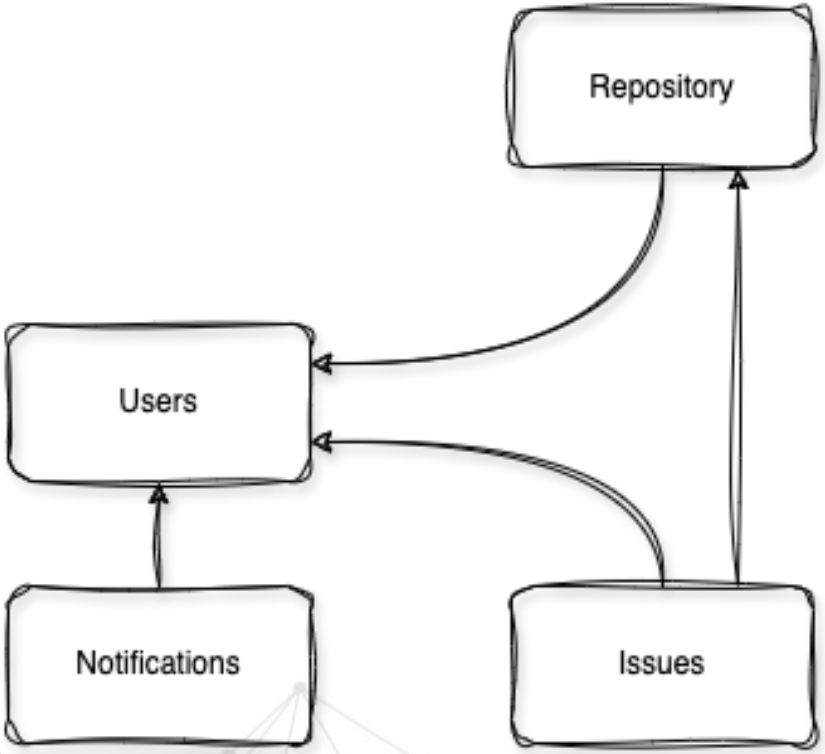

```
users/me
repositories/{user_id}
repositories/{user_id}/commits
repositories/{user_id}/issues
users/me/notifications
```

Федерация




```
query {
 getMyUser {
   id
   username
   repositories {
     name
   }
   lastIssues: issues(limit: 5) {
     message
     creator {
       username
     }
     repositories {
       name
     }
   }
   notifications(limit: 20) {
     message
   }
 }
}
```

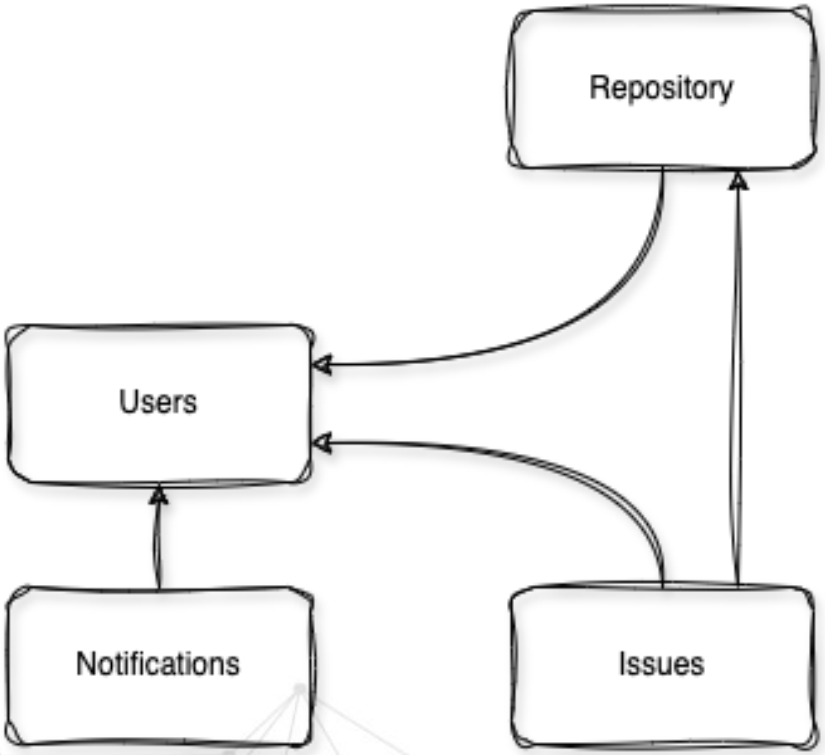
Федерация




```
type User @key(fields: "id") {
 id: ID!
 username: String!
}
```

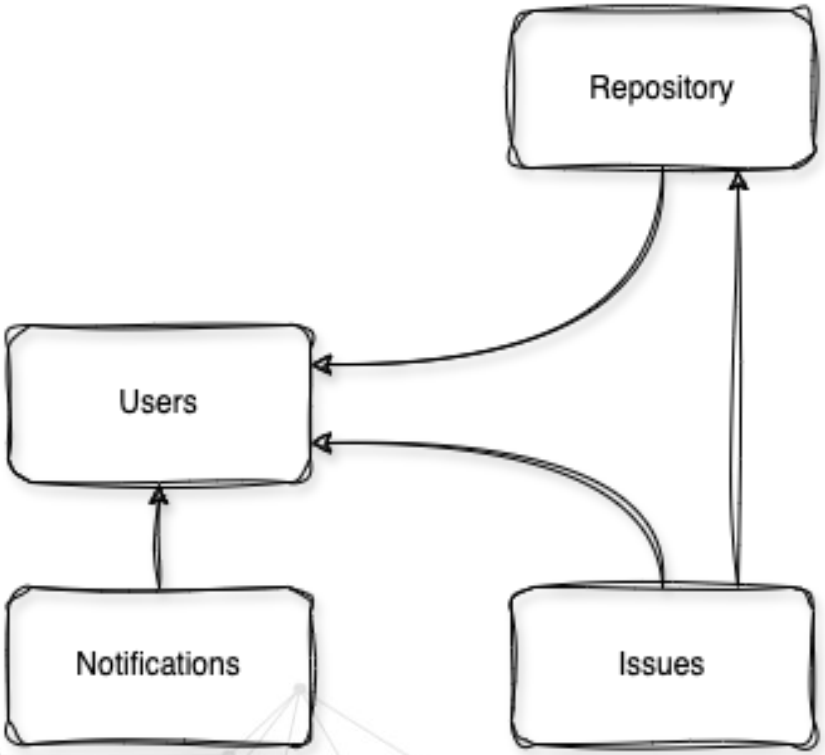
USETECH

Федерация




```
extend type User @key(fields: "id") {
   id: ID! @external
   repositories: [Repository!]!
}

type Repository {
 name: String!
 creator: User!
}
```

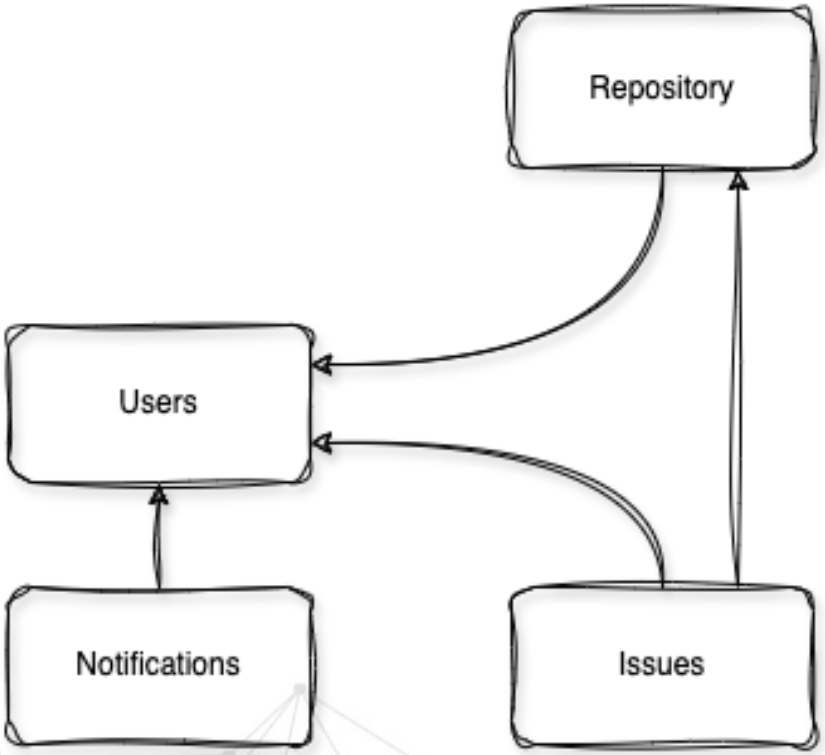
Федерация




```
extend type User @key(fields: "id") {
   id: ID! @external
   notifications: [Notification]!
}

type Notification {
 message: String!
 user: User!
}
```

USETECH

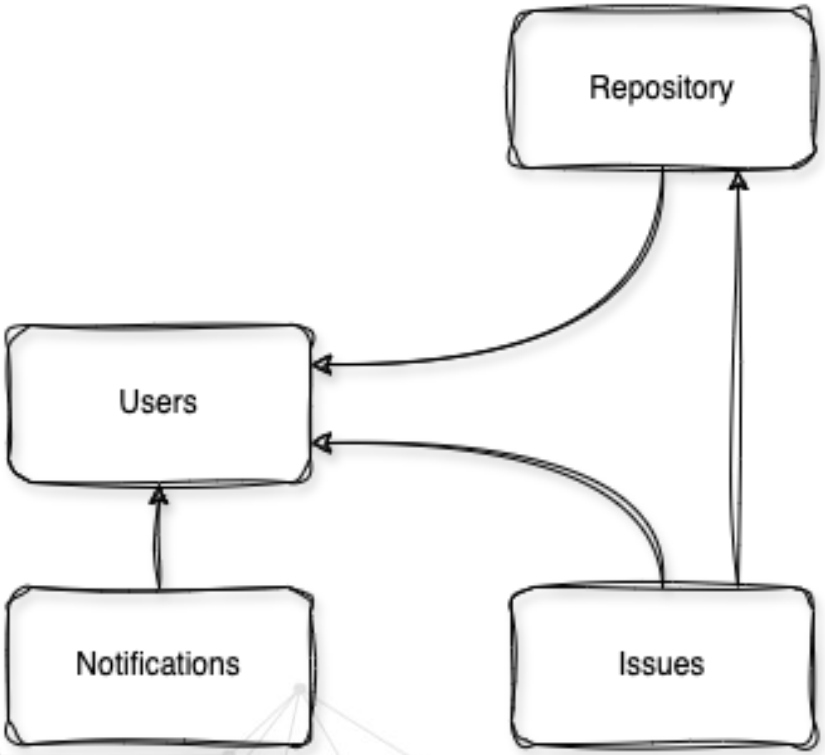
Федерация




```
extend type User @key(fields: "id") {
   id: ID! @external
   issues(limit: Int!): [Issue]!
}

extend type Repository @key(fields: "id") {
   id: ID! @external
   issues(limit: Int!): [Issue]!
}

type Issue {
 message: String!
 creator: User!
 repository: Repository!
}
```

USETECH

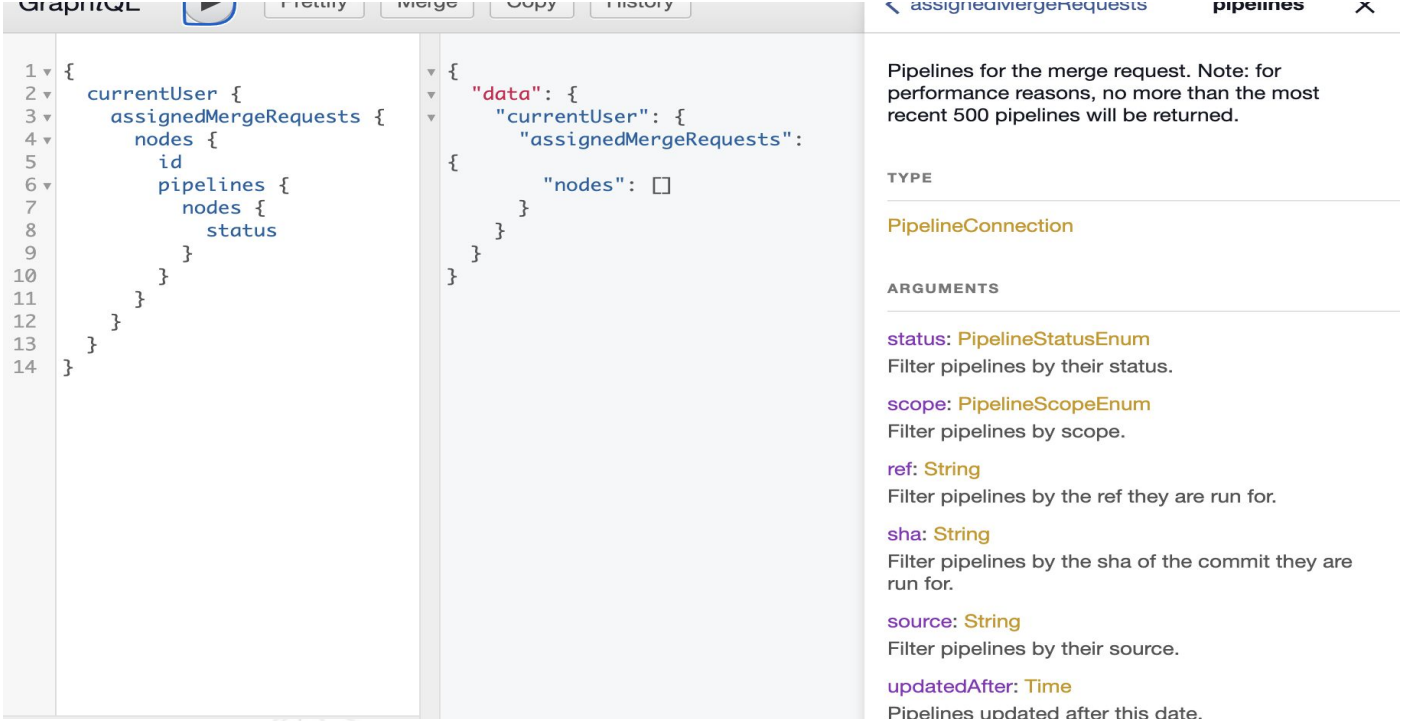
Федерация




```
type User {
 id: ID!
 username: String!
 repositories: [Repository!]!
 notifications(limit: Int!): [Notification]!
 issues(limit: Int!): [Issue]!
}
```

USETECH

GraphQL explorer



USETECH

Dataloaders

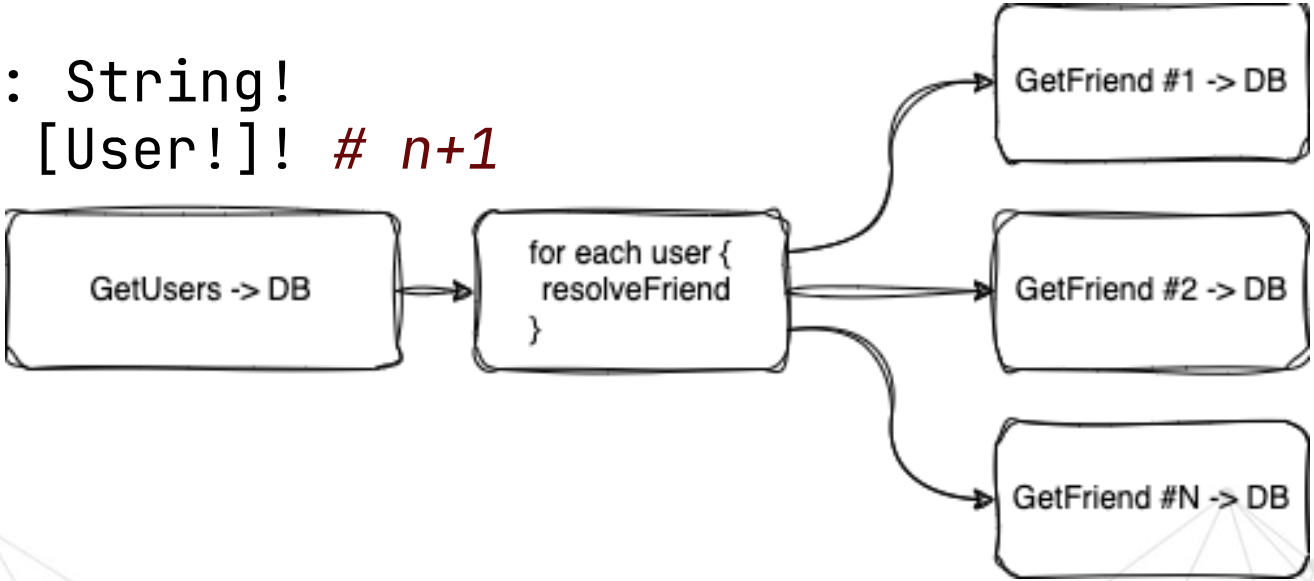

```
type User {
 id: ID!
 username: String!
 friends: [User!]!
}
```

Dataloaders


```
type User {
 id: ID!
 username: String!
 friends: [User!]! # n+1
}
```

Dataloaders


```
type User {
 id: ID!
 username: String!
 friends: [User!]! # n+1
}
```

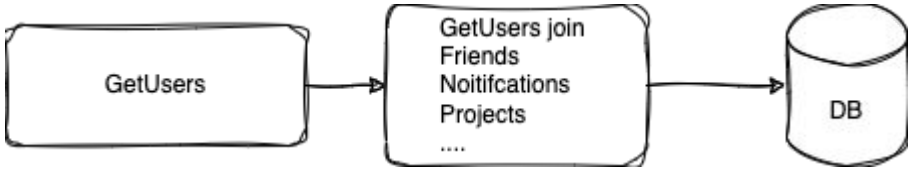


Dataloaders


```
type User {
 id: ID!
 username: String!
 friends: [User!]! # n+1
}
```

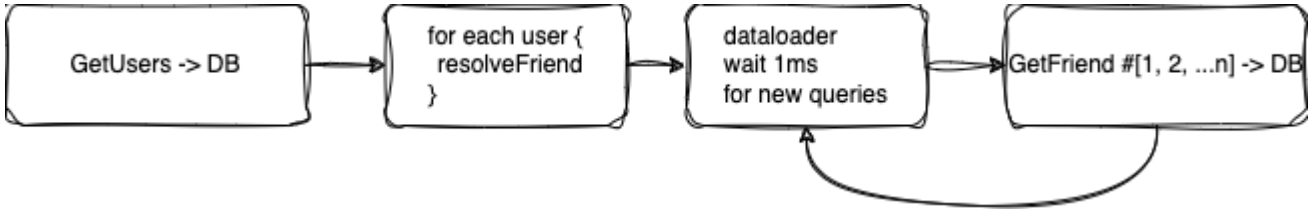
Dataloaders


```
type User {
 id: ID!
 username: String!
 friends: [User!]! # n+1
 notifications: [Notification!]! # n+1
 projects: [Projects!]! # n+1
}
```



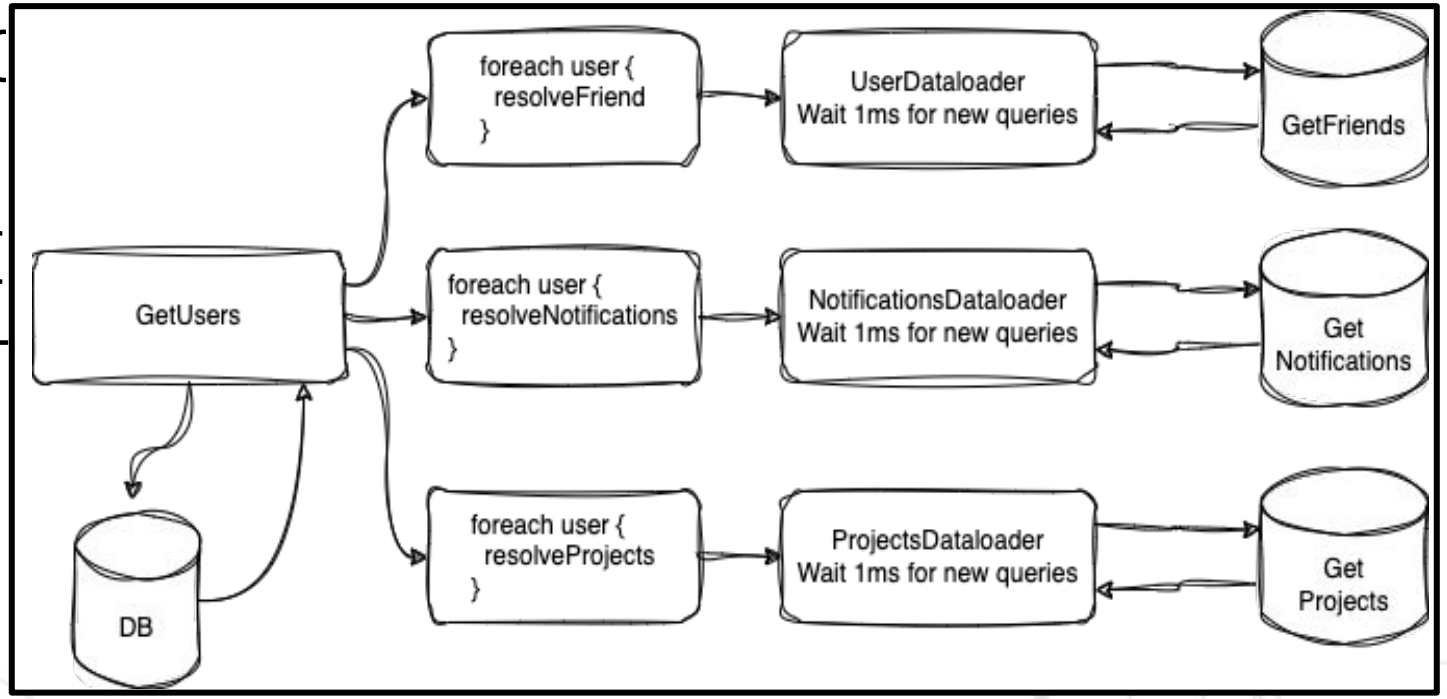
Dataloaders


```
type User {
 id: ID!
 username: String!
 friends: [User!]! # n+1
}
```



Dataloaders


```
type User {
 id: ID!
 username:
 friends: [
 notificati
 projects:
}
```



Dataloaders


```
func NewUserDataLoader(db *goqu.Database) *ldrs.GenericLoader[string, models.User] {
    return ldrs.NewGenericLoader(
        ldrs.GenericLoaderConfig[string, models.User]{
            Wait: time.Millisecond,
            Fetch: func(ctx context.Context, userIDs []string) ([]*models.User, []error) {
                users, err := models.GetUsersByIDs(ctx, db, userIDs)
                if err ≠ nil {
                    return nil, ldrs.SpreadErrorsToKeys(err, userIDs)
                }

                return models.UserListSortByKeys(users, userIDs), nil
            },
        },
    )
}
```

Dataloaders


```go
func NewUserDataLoader(db *goqu.Database) *ldrs.GenericLoader[string, models.User] {
    return ldrs.NewGenericLoader(
        ldrs.GenericLoaderConfig[string, models.User]{
            Wait: time.Millisecond,
            Fetch: func(ctx context.Context, userIDs []string) ([]*models.User, []error) {
                users, err := models.GetUsersByIDs(ctx, db, userIDs)
                if err ≠ nil {
                    return nil, ldrs.SpreadErrorsToKeys(err, userIDs)
                }

                return models.UserListSortByKeys(users, userIDs), nil
            },
        },
    )
}
```

Dataloaders


```
func NewUserDataLoader(db *goqu.Database) *ldrs.GenericLoader[string, models.User] {
    return ldrs.NewGenericLoader(
        ldrs.GenericLoaderConfig[string, models.User]{
            Wait: time.Millisecond,
            Fetch: func(ctx context.Context, userIDs []string) ([]*models.User, []error) {
                users, err := models.GetUsersByIDs(ctx, db, userIDs)
                if err != nil {
                    return nil, ldrs.SpreadErrorsToKeys(err, userIDs)
                }

                return models.UserListSortByKeys(users, userIDs), nil
            },
        },
    )
}
```

Dataloaders


```
func NewUserDataLoader(db *goqu.Database) *ldrs.GenericLoader[string, models.User] {
    return ldrs.NewGenericLoader(
        ldrs.GenericLoaderConfig[string, models.User]{
            Wait: time.Millisecond,
            Fetch: func(ctx context.Context, userIDs []string) ([]*models.User, []error) {
                users, err := models.GetUsersByIDs(ctx, db, userIDs)
                if err != nil {
                    return nil, ldrs.SpreadErrorsToKeys(err, userIDs)
                }

                return models.UserListSortByKeys(users, userIDs), nil
            },
        },
    )
}
```

Dataloaders


```
func NewUserDataLoader(db *goqu.Database) *ldrs.GenericLoader[string, models.User] {
    return ldrs.NewGenericLoader(
        ldrs.GenericLoaderConfig[string, models.User]{
            Wait: time.Millisecond,
            Fetch: func(ctx context.Context, userIDs []string) ([]*models.User, []error) {
                users, err := models.GetUsersByIDs(ctx, db, userIDs)
                if err ≠ nil {
                    return nil, ldrs.SpreadErrorsToKeys(err, userIDs)
                }

                return models.UserListSortByKeys(users, userIDs), nil
            },
        },
    )
}
```

Прочее

- не зависит от транспорта

Прочее

- не зависит от транспорта

➔ получение данных **после** изменения

```
type Mutation {
 updateUser(id: 1): UpdateUserPayload
}

type UpdateUserPayload {
 record: User
 query: Query
}

mutation {
 updateUser(id: 1) {
   record {
     username
   }
   query {
     getUsers { ... }
     getProjects { ... }
   }
 }
}
```

Прочее

- не зависит от транспорта
- получение данных **после** изменения

➔ markdown документация 🥰

Прочее

- не зависит от
- получение да
- ➔ markdown до

```
type Query {
    """
    Translates a string from a given language into a different language.
    It uses translators in these order:
    - google translate
    - yandex translate
    - chatGPT
    """
    translate(
        "The original language that `text` is provided in."
        fromLanguage: Language
        "The translated language to be returned."
        toLanguage: Language
        "The text to be translated."
        text: String
    ): String
}

"""
The set of languages supported by `translate`.
"""
enum Language {
    "English"
    EN
    "French"
    FR
    "Chinese"
    CH
}
```

Прочее


```
type Query {
  """
```

given language into a different language.
e order:

that `text` is provided in."

e to be returned."

ted."

by `translate`.

translate

Translates a string from a given language into a different language.
It uses translators in these order:
- google translate
- yandex translate
- chatGPT

Type

String

Arguments

fromLanguage: Language

The original language that `text` is provided in.

toLanguage: Language

The translated language to be returned.

text: String

The text to be translated.

Прочее


```
type Query {
  """
```

translate

Translates a string from a given language into a different langua
It uses translators in these order:
- google translate
- yandex translate
- chatGPT

Type

String

Arguments

fromLanguage: Language

The original language that `text` is provided in.

toLanguage: Language

The translated language to be returned.

text: String

The text to be translated.

Language

The set of languages supported by `translate`.

Enum Values

EN

English

FR

French

CH

Chinese

Давайте обсудим?

- gqlgen.com
- github.com/nodkz/conf-talks
- apollographql.com/docs/federation