


Познай темную сторону web разработки. GraphQL на бэкенде

Джавед Бехлим

Инженер-разработчик клиентских приложений.

 <https://github.com/JayBee007>

 @jaybee007

 jaybee.codes@gmail.com

Содержание

01 Что себя представляет GraphQL

02 Плюсы - Минусы

03 Готовые сервисы и реализации
GraphQL сервер

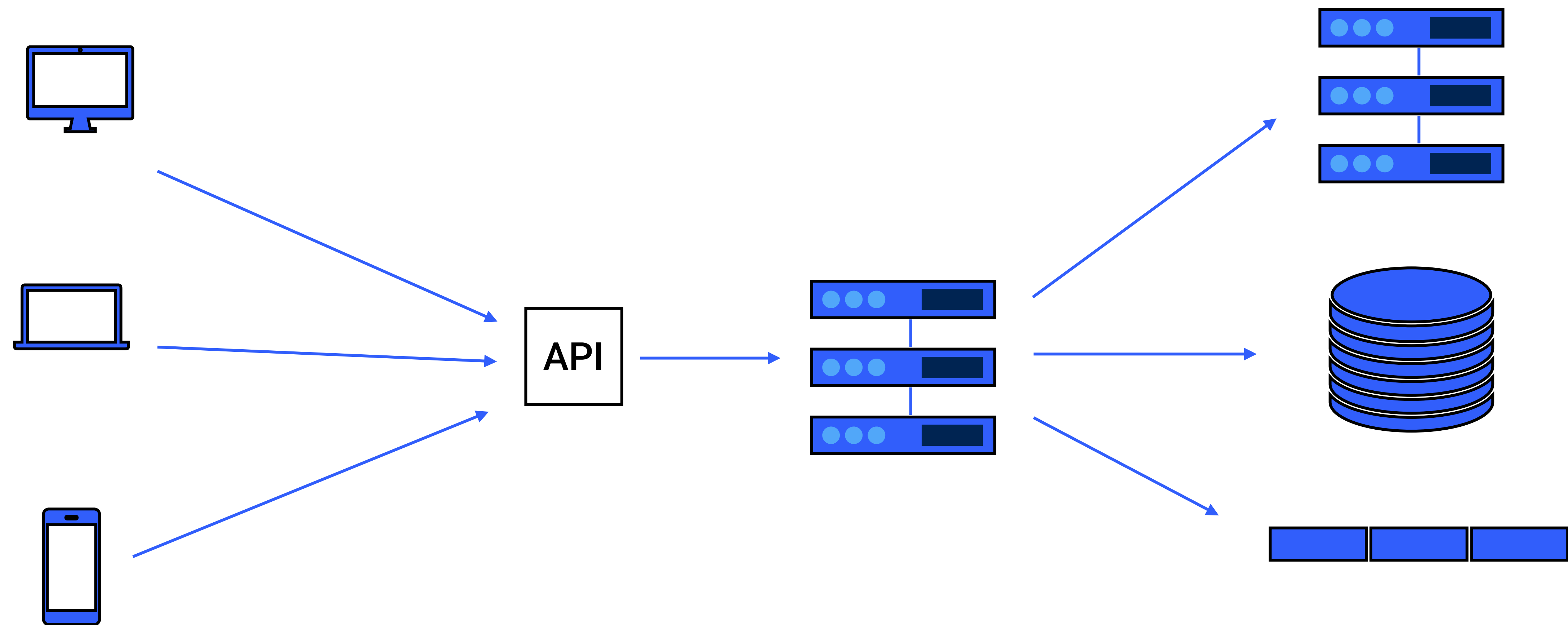
04 Терминология вокруг GraphQL
сервера

05 Демо

06 Что дальше?

Что такое Бекэнд?

Что такое Бекэнд?

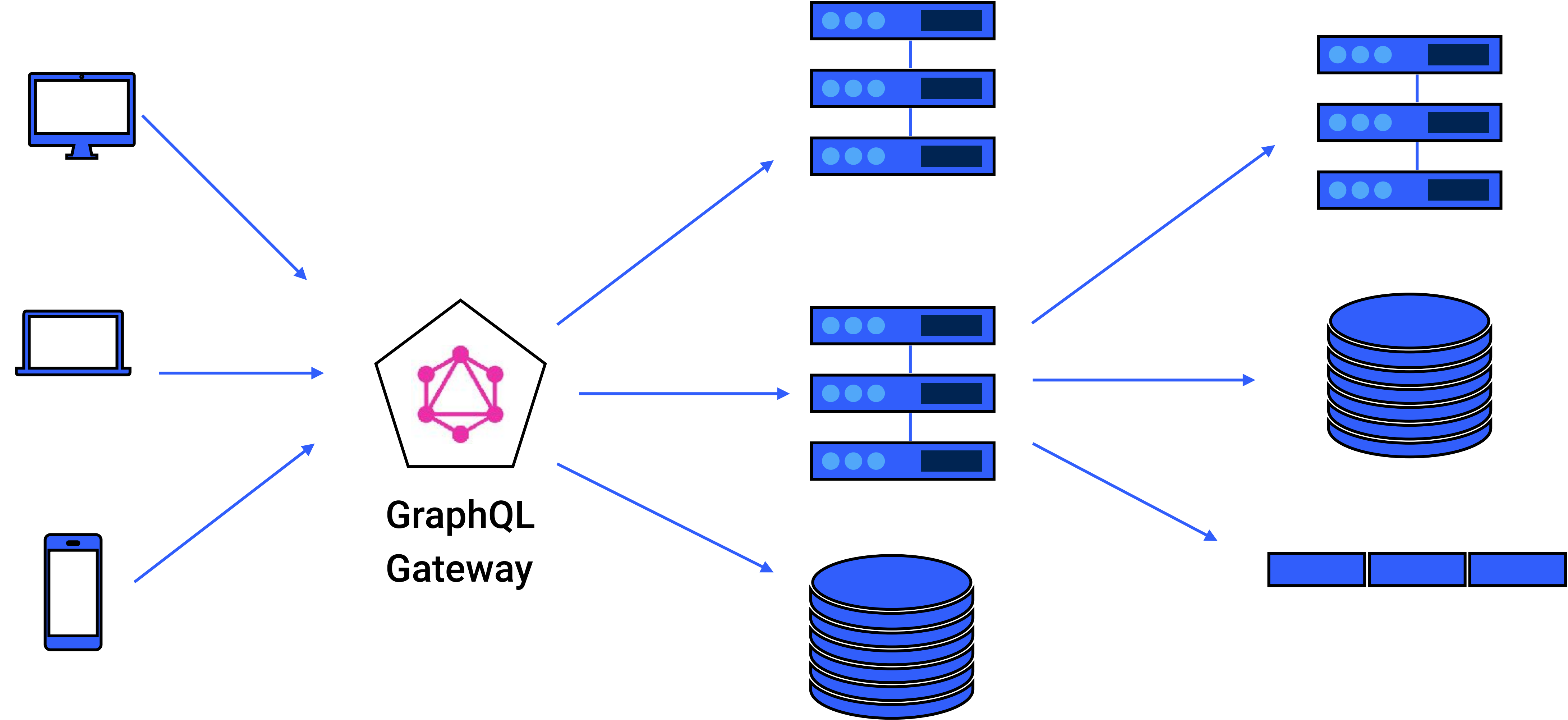


Что себя представляет GraphQL

Основное понятие

GraphQL, это спецификация для языка запросов и движок для выполнения запросов, которую разработала FaceBook и опубликовала в Open Source в 2015г.

Что такое Бекэнд?



Что себя представляет GraphQL

Плюсы

- **Самодокументация**
- Один endpoint
- Управление версиями
- Типизация

Минусы

- Сложность запроса
- Кеширование
- Ограничение количества запросов

[← Message](#)

User



 Search User...

User type, describes the fields for it.

FIELDS

id: Int!

Each user gets an id generated by database.

username: String!

The user's username, should be typed in the login field.

Что себя представляет GraphQL

Плюсы

- Самодокументация
- **Один endpoint**
- Управление версиями
- Типизация

Минусы

- Сложность запроса
- Кеширование
- Ограничение количество запросов

Что себя представляет GraphQL

Плюсы

- Самодокументация
- Один endpoint
- **Управление версиями**
- Типизация

Минусы

- Сложность запроса
- Кеширование
- Ограничение количество запросов

Что себя представляет GraphQL

Плюсы

- Самодокументация
- Один endpoint
- Управление версиями
- **Типизация**

Минусы

- Сложность запроса
- Кеширование
- Ограничение количество запросов

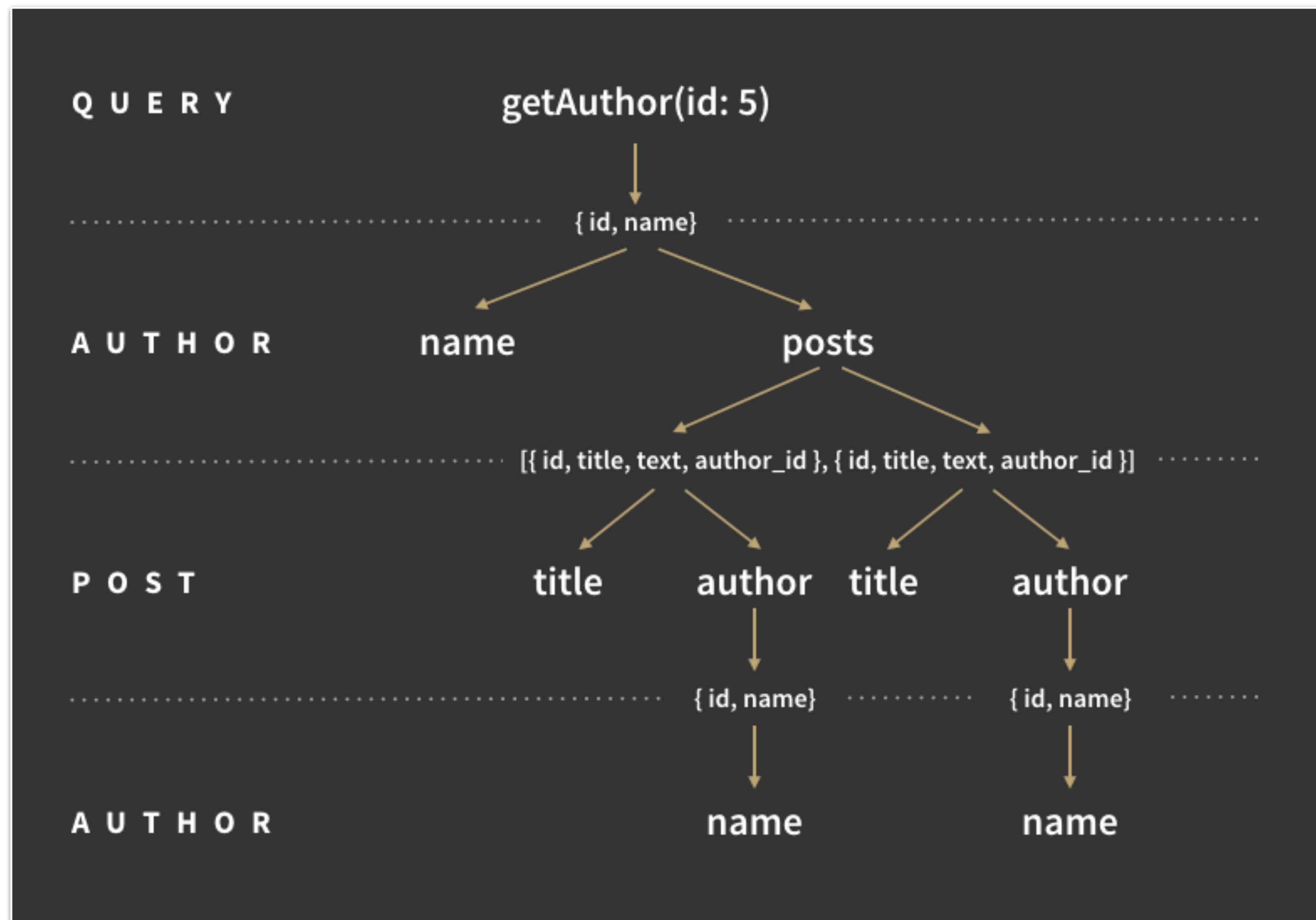
Что себя представляет GraphQL

Плюсы

- Самодокументация
- Один endpoint
- Управление версиями
- Типизация

Минусы

- Сложность запроса
- Кеширование
- Ограничение количество запросов



Что себя представляет GraphQL

Плюсы

- Самодокументация
- Один endpoint
- Управление версиями
- Типизация

Минусы

- Сложность запроса
- **Кеширование**
- Ограничение количество запросов

Что себя представляет GraphQL

Плюсы

- Самодокументация
- Один endpoint
- Управление версиями
- Типизация

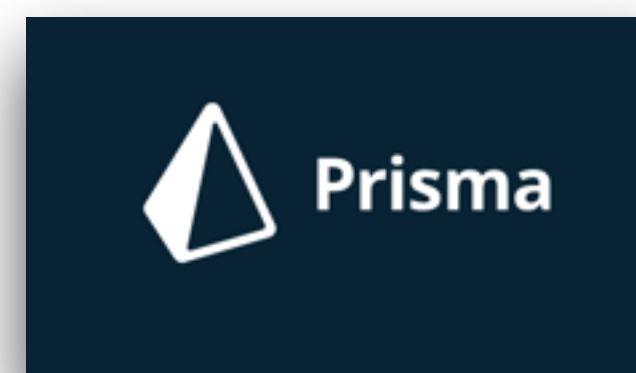
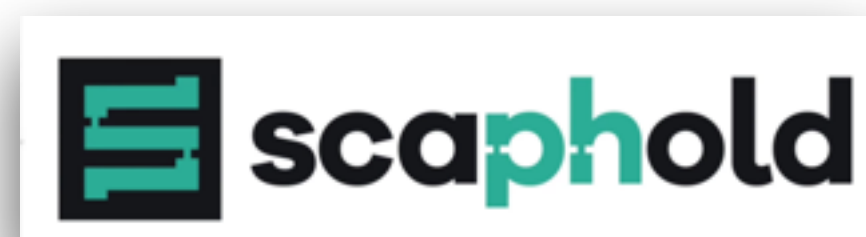
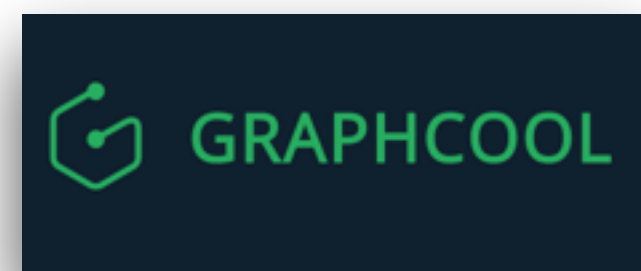
Минусы

- Сложность запроса
- Кеширование
- **Ограничение количество запросов**

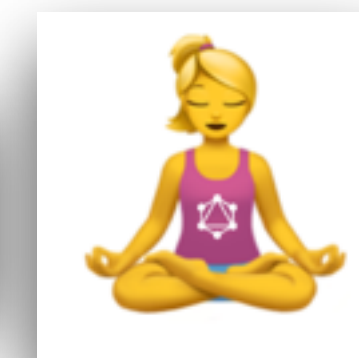
Готовые сервисы и реализации GraphQL сервер

Готовые сервисы и реализации GraphQL сервер*

Сервисы



Реализация на разные языки



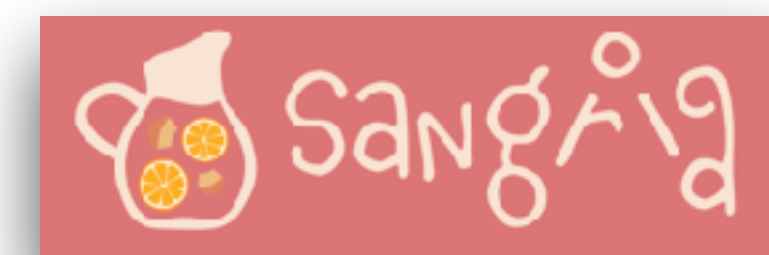
JavaScript



Ruby



php

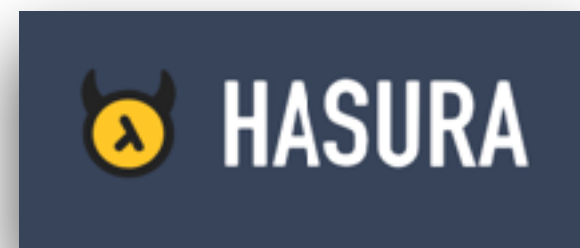
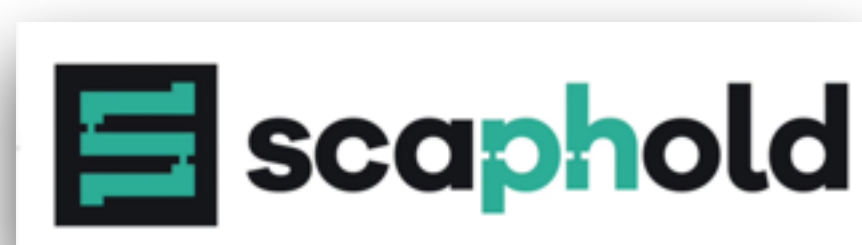
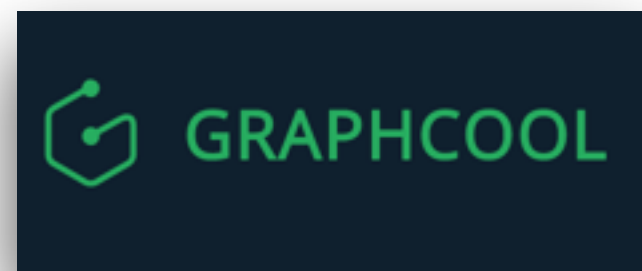


Scala

*<https://github.com/chentsulin/awesome-graphql>

Готовые сервисы и реализации GraphQL сервер*

Сервисы



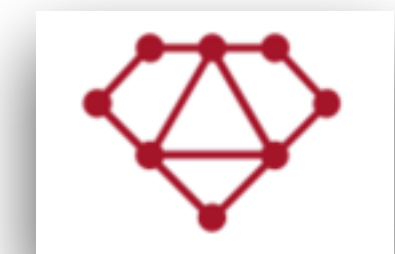
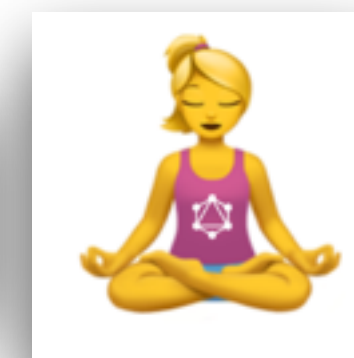
*<https://github.com/chentsulin/awesome-graphql>

Готовые сервисы и реализации GraphQL сервер*

Реализация на разных языках



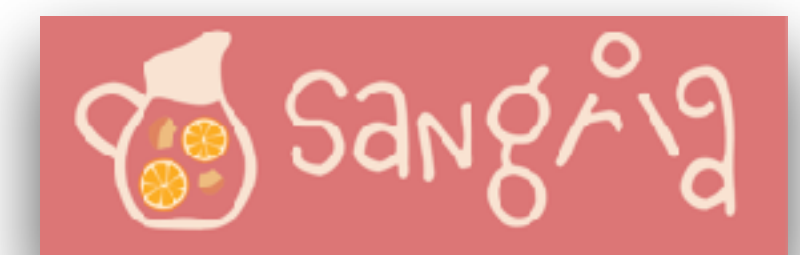
JavaScript



Ruby



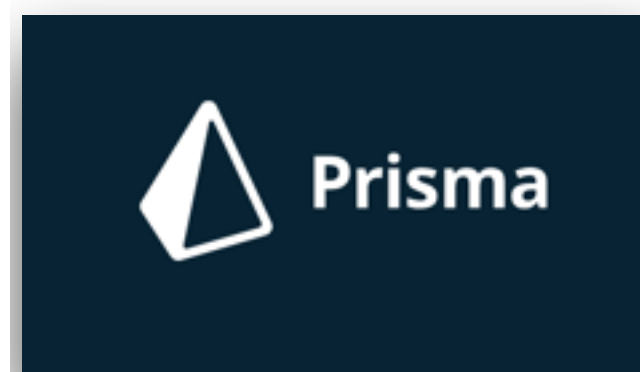
php



Scala

*<https://github.com/chentsulin/awesome-graphql>

Готовые сервисы и реализации GraphQL сервер*



*<https://github.com/chentsulin/awesome-graphql>

Терминология вокруг GraphQL сервера

Терминология вокруг GraphQL сервера

Схемы и типы

Схема описывает все возможные способности GraphQL сервиса. Она определяет типы и директивы, которые GraphQL сервис будет поддерживать.

```
schema {  
  query: Query  
  mutation: Mutation  
  subscription: Subscription  
}
```

Scalar типы

Int: A signed 32-bit integer.

Float: A signed double-precision floating-point value.

String: A UTF-8 character sequence.

Boolean: true or false

ID: The ID scalar type represents a unique identifier, often used to refetch an object or as the key for a cache.

```
schema {  
  query: Query  
  mutation: Mutation  
  subscription: Subscription  
}
```


Схемы и типы

```
schema {  
  query: Query  
  mutation: Mutation  
  
  Type Query {  
    getUser: User!  
  }  
  
  Type User {  
    id: Int!  
    username: String!  
  }  
}
```

```
Query: {  
  getUser: (parent, args, context, info)  
    => ({  
      id: 1,  
      username: 'John'  
    })  
}
```

Резольвер

Каждая схема имеет основные или корневые типы, которые обозначают возможные входы для GraphQL API. Они для запроса вызывают соответствующую функцию - Резольвер по нашему - “решало”.

schema.js

```
schema {  
  query: Query  
  mutation: Mutation  
  
  Type Query {  
    getMessages: [Message]!  
  }  
  
  Type Message {  
    id: Int!  
    text: String!  
    user: User!  
  }  
}
```

client.js

```
query {  
  getMessages {  
    id  
    text  
    user {  
      id  
      username  
    }  
  }  
}
```

server/resolver.js

```
Query: {  
  Message: {  
    user: (parent) => models.User.findOne(  
      { where: { id: parent.userId } }  
    )  
  }  
  
  getMessages: (parent, args, context,  
    info) => [{ id: 1, text: "message1", userId:  
    1},  
    { id: 2, text: "message2", , userId: 2},]  
}
```

client.js

```
query {  
  getUser(id: 34) {  
    id  
    username  
  }  
}
```

server/resolver.js

```
Query: {  
  getUser: (parent, args,  
    {models}, info) =>  
    models.User.findOne(  
      { where: { id: args.id } }  
    )  
}
```

server.js

```
const server = new ApolloServer({  
  schema,  
  resolvers,  
  context: () => {  
    return { models };  
  },  
});
```

server/resolver.js

```
Query: {  
  getUser: (parent, args,  
    context, info) =>  
    context.models.User.findOne(  
      { where: { id: args.id } }  
    )  
}
```

```
Query: {  
  getUser: (parent, args, context, info)  
    => ({  
      id: 1,  
      username: 'John'  
    })  
}
```

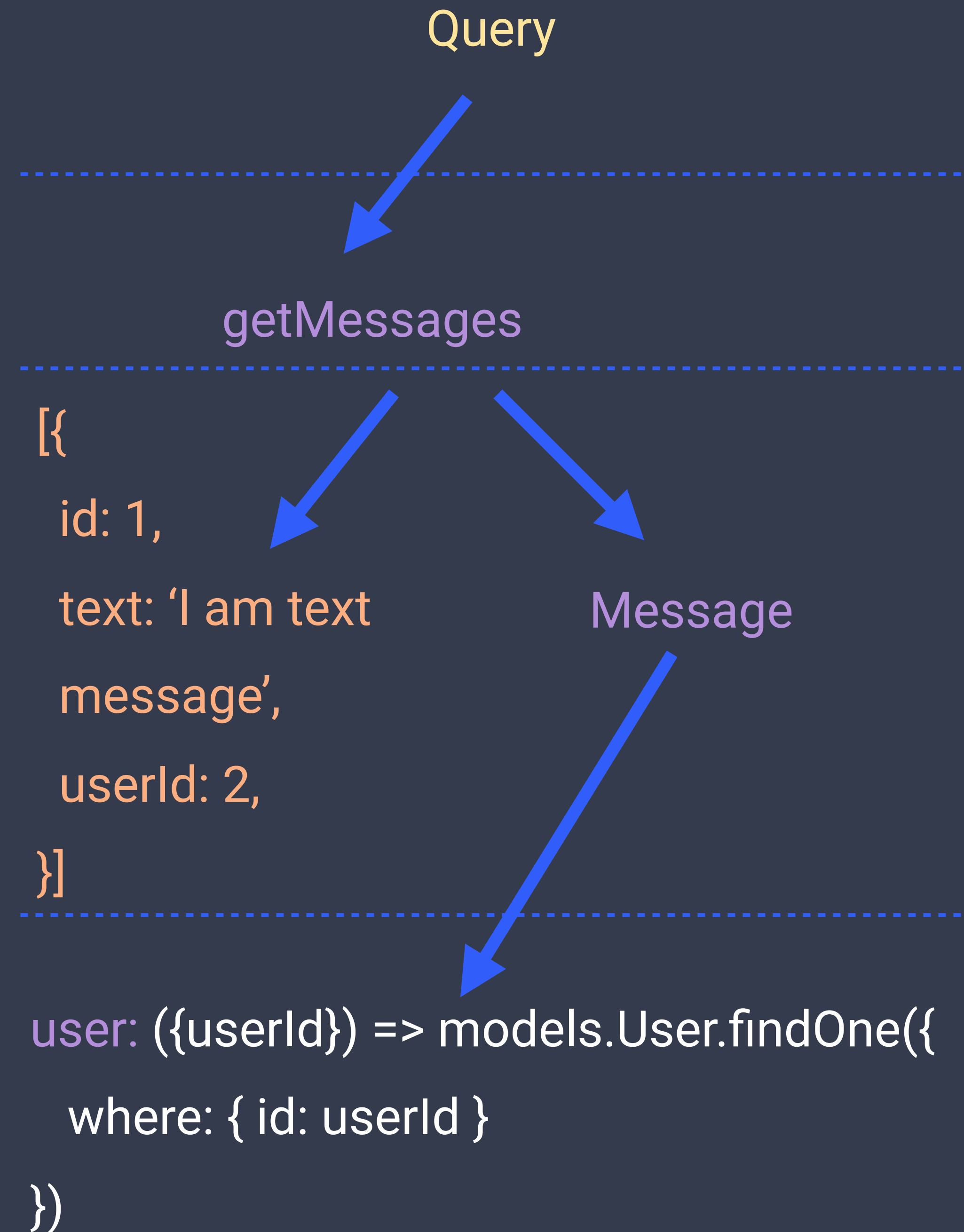
Резольвер

```
schema {  
  query: Query  
  mutation: Mutation  
  
  Type Query {  
    getMessages: [Message]!  
  }  
  
  Type Message {  
    id: Int!  
    text: String!  
    user: User!  
  }  
}
```

```

Query: {
  Message: {
    user: (parent) => models.User.findOne(
      { where: { id: parent.userId } }
    )
  }
  getMessages: (parent, args, context,
    info) => [{ id: 1, text: "message1", userId:
    1},
    { id: 2, text: "message2", , userId: 2},]
}

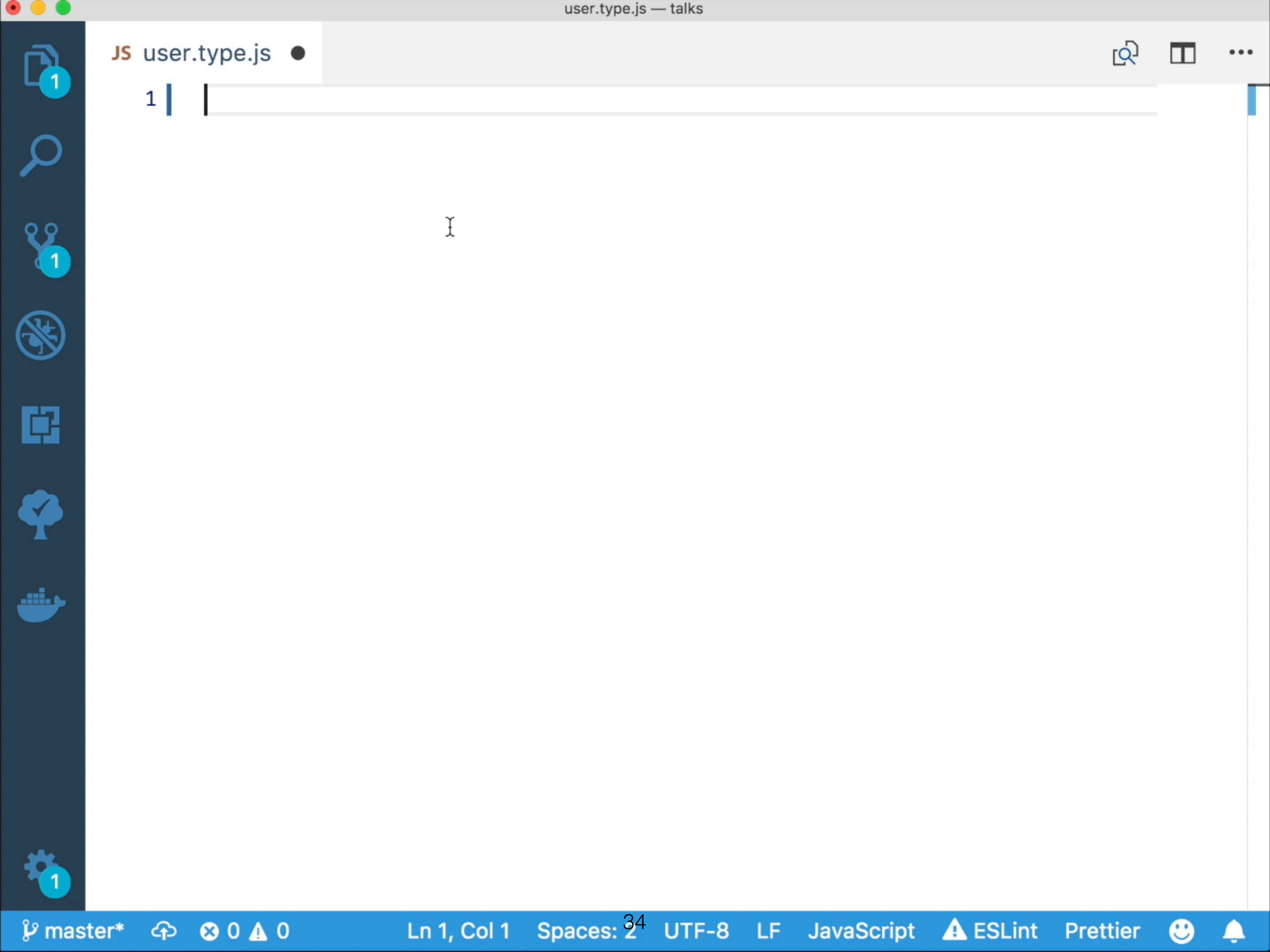
```

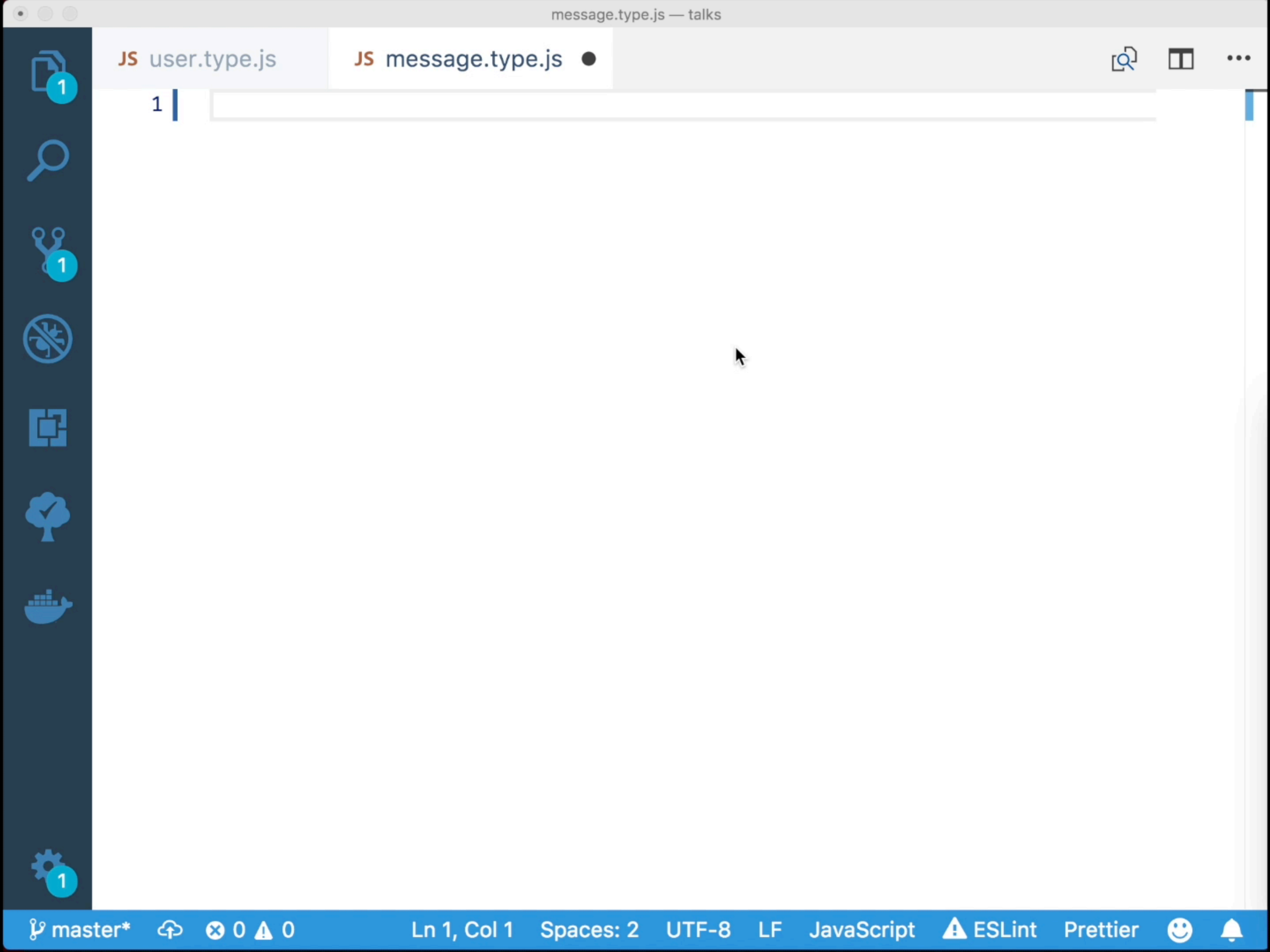


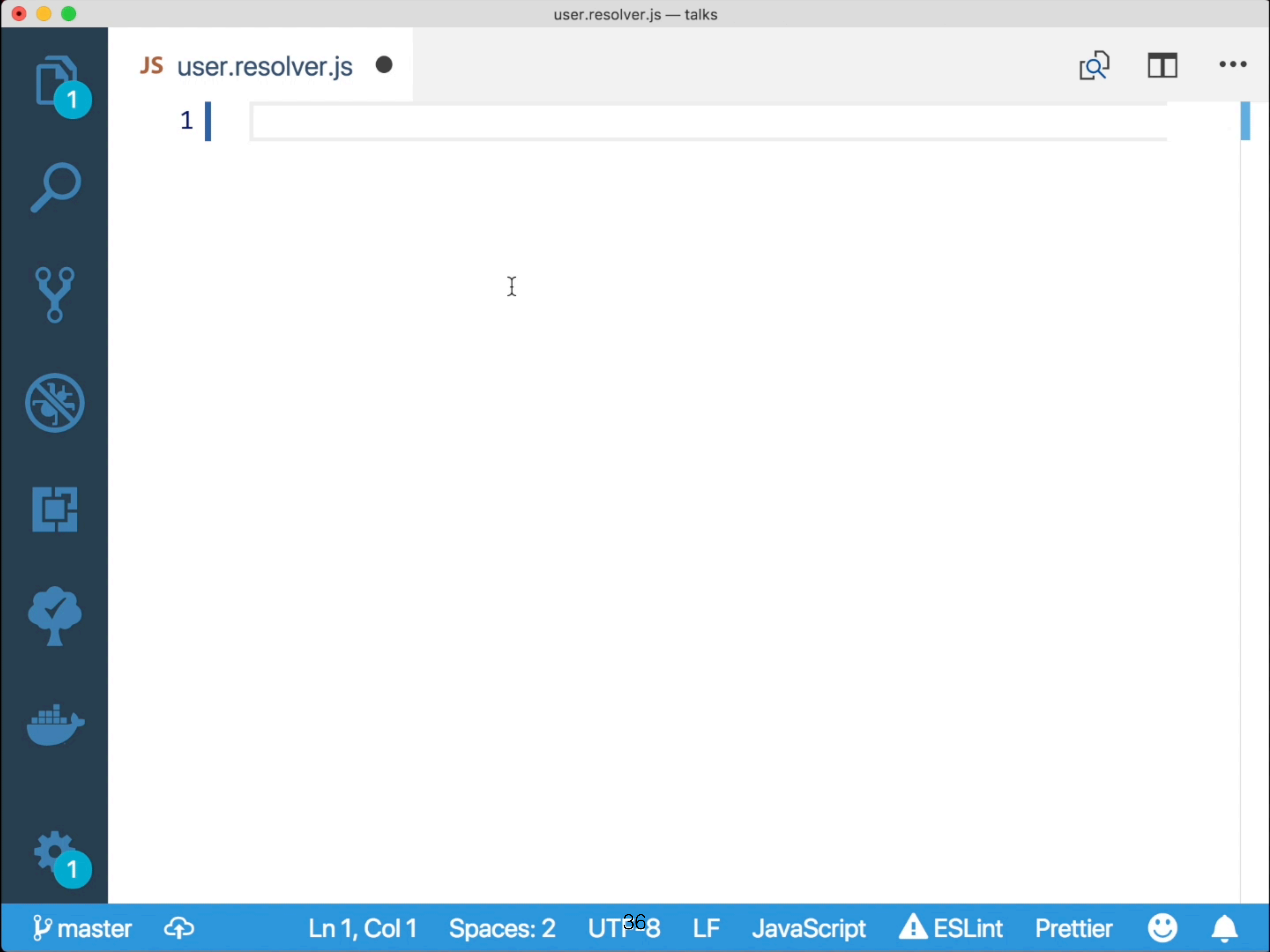
Рамблер/

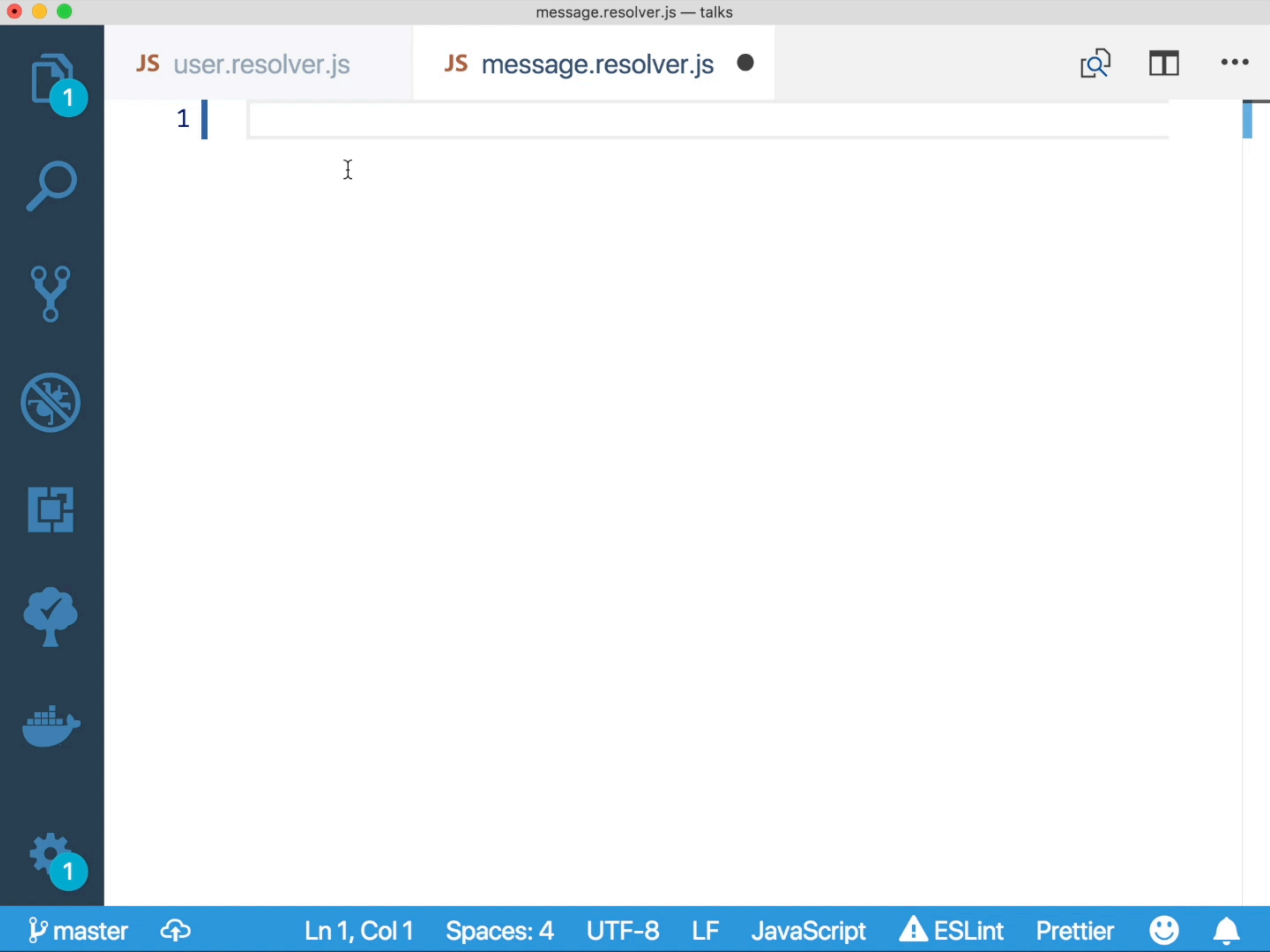
RAMBLER&Co

Демо









localhost

Playground - http://localhost:4000/graphql

New Tab

Settings

+

PRETTIFY

HISTORY

http://localhost:4000/graphql

COPY CURL

SHARE

1

"data": {
 "messageAdded": {
 "id": 3,
 "text": "Vasya
Pupkin",
 "user": {
 "id": 38,
 "username":
 "Vasya"
 }
 }
 }
}

QUERY VARIABLES

HTTP HEADERS

localhost:4000/graphql

New Tab

Settings

+

PRETTIFY

HISTORY

http://localhost:4000/graphql

COPY CURL

SHARE PLAYGROUND

1

"data": {
 "messageAdded": {
 "id": 1,
 "text": "Message
from Jane",
 "user": {
 "id": 1,
 "username":
 "Jane"
 }
 }
 }
}

12 days ago

"data": {
 "messageAdded": {

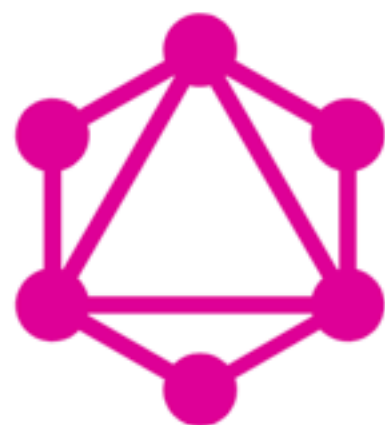
QUERY VARIABLES

HTTP HEADERS (1)

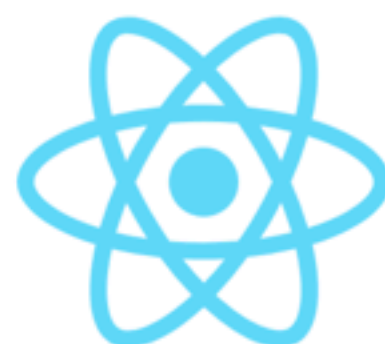
TRACING

Что дальше?

Что дальше?



GraphQL



React

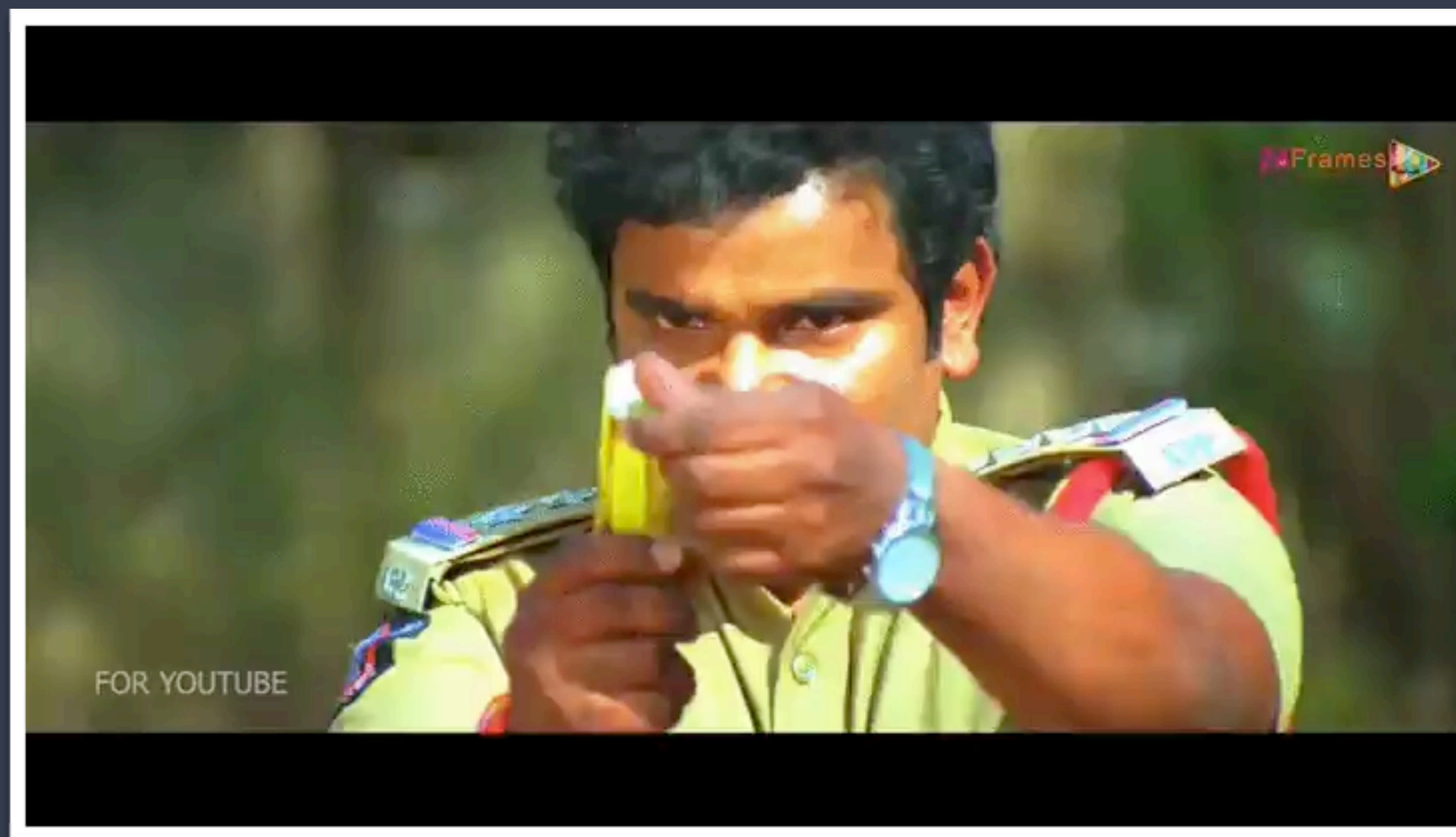


Apollo



Neo4j Database

Заключение



Название раздела

Спасибо за
внимание!

Рамблер/



<http://bit.ly/2yzr0lZ>

RAMBLER&Co