A long time ago in a conference hall

far, far away…

API WARS

The Rise of GraphQL

Personal Programmer

Martin Nachev

What is GraphQL?

- API query language
  - Created by Facebook as an ***<u>alternative</u>*** to REST
  - Open sourced in 2015
- Specification with implementations in many languages

Java node JS Microsoft .NET Scala python

- Used by: GitHub airbnb twitter PayPal Instagram

Schema

- Type system

```
type Query {
 getJedi(id: ID!): Jedi
 getStatuses: [Status!]!
}

type Jedi {
 id: ID!
 name: String!
 tasks(s: Status): [Task]
}

type Task {
 info: String!
 status: Status!
}

enum Status {
 ACTIVE, INACTIVE
}
```

Schema

- Type system

```
type Query {
 getJedi(id: ID!): Jedi
 getStatuses: [Status!]!
}

type Jedi {
 id: ID!
 name: String!
 tasks(s: Status): [Task]
}

type Task {
 info: String!
 status: Status!
}

enum Status {
 ACTIVE, INACTIVE
}
```

Request body

- No wildcard * to get all fields

```
{
 getJedi(id: 666) {
     name
     tasks(s: ACTIVE) {
         info
     }
 }


 getStatuses
}
```

Schema

- Type system

```
type Query {
 getJedi(id: ID!): Jedi
 getStatuses: [Status!]!
}

type Jedi {
 id: ID!
 name: String!
 tasks(s: Status): [Task]
}

type Task {
 info: String!
 status: Status!
}

enum Status {
 ACTIVE, INACTIVE
}
```

Request body

- No wildcard * to get all fields

```
{
 getJedi(id: 666) {
     name
     tasks(s: ACTIVE) {
         info
     }
 }


 getStatuses
}
```

Response

- No over/under-fetch
- Everything you need in 1 call

```
{
 "data": {
  "getJedi": {
    "name": "Rey",
    "tasks": [
      {
        "info": "Ruin Star Wars"
      },
      {
        "info": "Get Corona"
      }
    ]
  },
  "getStatuses": [
    "ACTIVE", "INACTIVE"
  ]
 }
}
```

REST controllers → GraphQL resolvers

https://graphql-java-kickstart.com/

```
dependencies {
   compile 'com.graphql-java-kickstart:graphql-spring-boot-starter:6.0.1'
}
```

```
(@see also 'io.leangen.graphql:graphql-spqr-spring-boot-starter:0.0.4')
```

- REST

```
@Data
class Task {
  Long id;
  String info;
  Status status;
}

enum Status {
  ACTIVE,
  INACTIVE
}
```

```
@RestController
@RequestMapping("/tasks")
class TaskController {

  @Autowired TaskService taskService;

  @GetMapping("/{id}")
  Task getTask(@PathVariable Long id) {
    return taskService.getTask(id);
  }

  @PostMapping
  Task createTask(@RequestBody Task task) {
    return taskService.create(task);
  }

  @DeleteMapping("/{id}")
  void deleteTask(@PathVariable Long id) {
    taskService.delete(id);
  }
}
```

- REST

```
@Data
class Task {
  Long id;
  String info;
  Status status;
}

enum Status {
  ACTIVE,
  INACTIVE
}
```

```
@RestController
@RequestMapping("/tasks")
class TaskController {

  @Autowired TaskService taskService;

  @GetMapping("/{id}")
  Task getTask(@PathVariable Long id) {
    return taskService.getTask(id);
  }

  @PostMapping
  Task createTask(@RequestBody Task task) {
    return taskService.create(task);
  }

  @DeleteMapping("/{id}")
  void deleteTask(@PathVariable Long id) {
    taskService.delete(id);
  }
}
```

- GraphQL

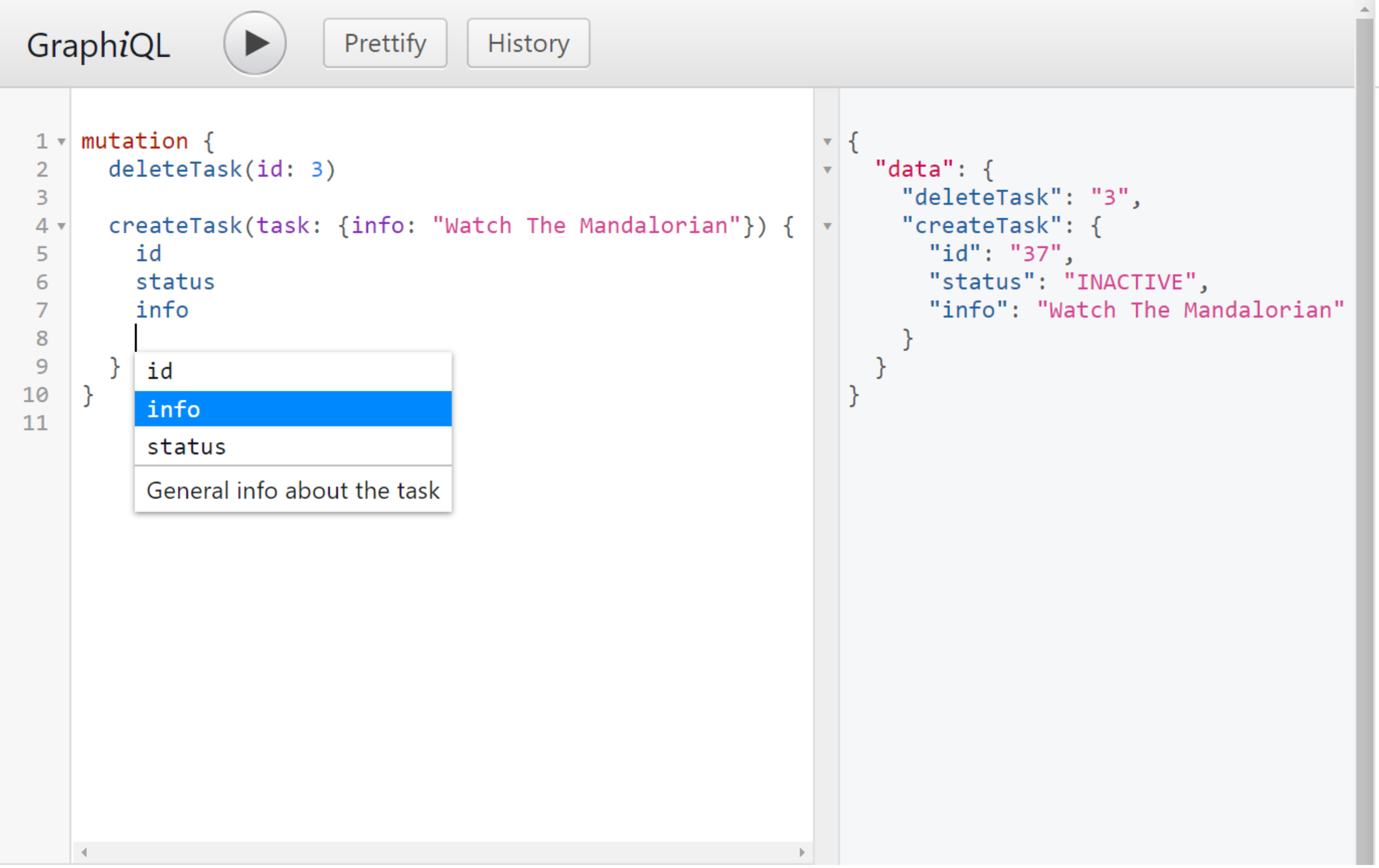
```
@Data
class Task {
  Long id;
  String info;
  Status status;
}

enum Status {
  ACTIVE,
  INACTIVE
}
```

```
@Component
class TaskQuery implements GraphQLQueryResolver {

  @Autowired TaskService taskService;

  Task getTask(Long id) {
    return taskService.getTask(id);
  }
}
@Component
class TaskMutation implements GraphQLMutationResolver {

  @Autowired TaskService taskService;

  Task createTask(Task task) {
    return taskService.create(task);
  }

  Long deleteTask(Long id) {
    taskService.delete(id);
    return id;
  }
```

Define schema → resources/**/*.graphqls

```graphql
type Query {
    getTask(id: ID!): Task
}

type Task {
    id: ID!
    "General info about the task"
    info: String!
    status: Status
}

enum Status {
    ACTIVE, INACTIVE
}
```

```graphql
type Mutation {
    createTask(task: TaskInput!): Task
    deleteTask(id: ID!): ID
}

input TaskInput {
    info: String!
    status: Status = INACTIVE
}
```



```
mutation {
  deleteTask(id: 3)

  createTask(task: {info: "Watch The Mandalorian"}) {
    id
    status
    info

  }
}
```

```
id
info
status
General info about the task
```

```
{
  "data": {
    "deleteTask": "3",
    "createTask": {
      "id": "37",
      "status": "INACTIVE",
      "info": "Watch The Mandalorian"
    }
  }
}
```

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation

subscription: Subscription

- Single endpoint (default: /graphql)
- Full introspection

```
class Task {
  Long id;
  String info;
  Status status;
}
```

```
type Task {
    id: ID!
    info: String!
    externalInfo(criteria: String): String
    status: Status
}
```

```java
class Task {
  Long id;
  String info;
  Status status;
}
```

```graphql
type Task {
    id: ID!
    info: String!
    externalInfo(criteria: String): String
    status: Status
}
```

```java
@Component
class TaskFieldResolver implements GraphQLResolver<Task> {

  @Autowired ExternalService externalService;

  String externalInfo(Task task, String criteria) {
    return externalService.getExternalInfo(task.getId(), criteria);
  }
}
```

Subscriptions

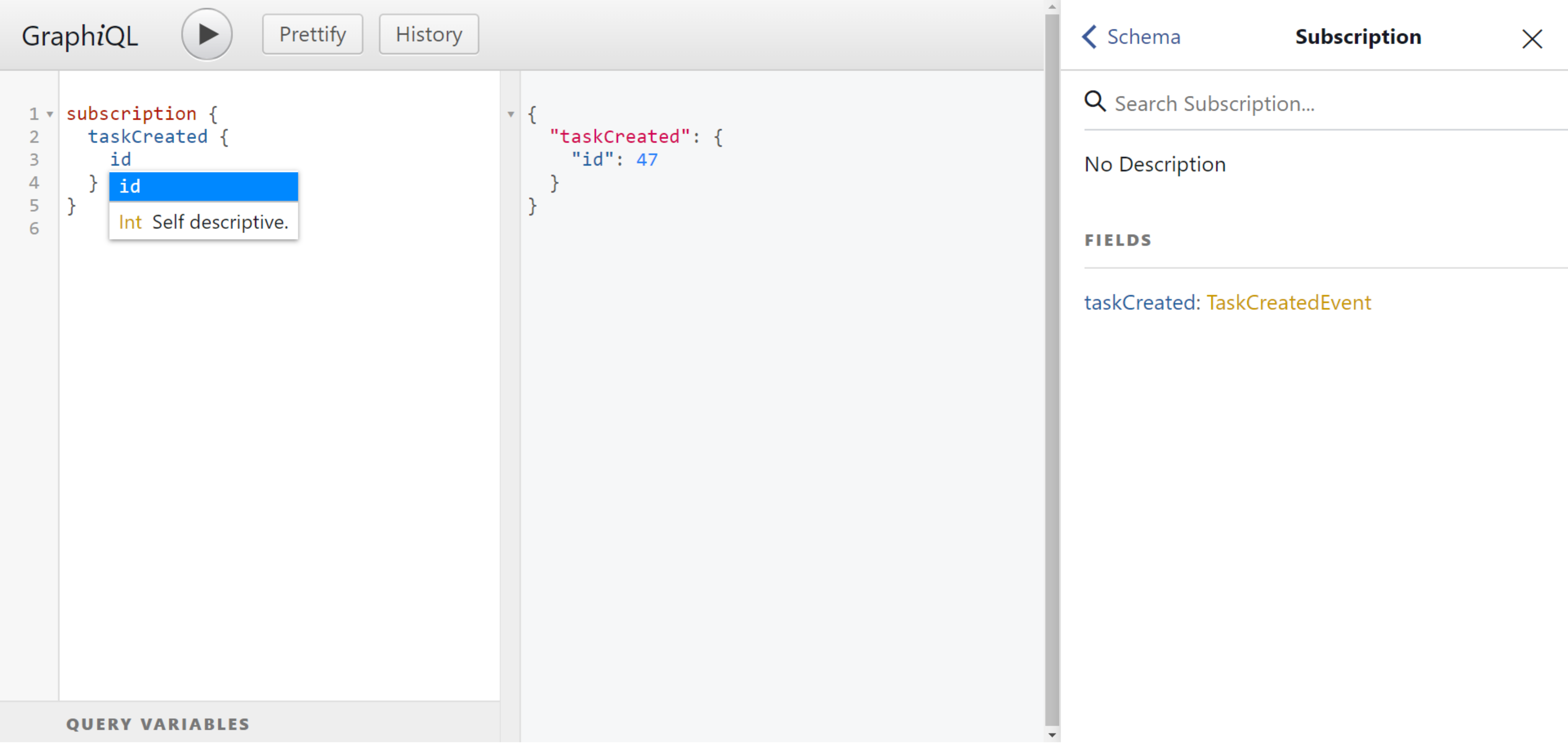
- Real-time push notification events from server to client
  - Based on WebSockets
- Implement with Reactive Streams
  - Reactor Core, RxJava …

```
type Subscription {
    taskCreated: TaskCreatedEvent
}
```

```
type TaskCreatedEvent {
    id: Int
}
```

```java
@Service class TaskService {

  @Autowired ApplicationEventPublisher publisher;

  Task create(Task task) {
    // ...
    publisher.publishEvent(new TaskCreatedEvent(id));
    // ...
  }
}
```

```
                  @Service class TaskService {

                    @Autowired ApplicationEventPublisher publisher;

                    Task create(Task task) {
                      // ...
                      publisher.publishEvent(new TaskCreatedEvent(id));
                      // ...
                    }
                  }

@Component
class TaskCreatedEventSubscription implements GraphQLSubscriptionResolver {

  List<FluxSink<TaskCreatedEvent>> subs = new CopyOnWriteArrayList<>();

  @EventListener
  public void handle(TaskCreatedEvent event) { subs.forEach(sub -> sub.next(event)); }

  Publisher<TaskCreatedEvent> taskCreated() {
    return Flux.create(sub -> subs.add(sub.onDispose(() -> subs.remove(sub))));
  }
```



```
subscription {
  taskCreated {
    id
  }
}
```

id
Int Self descriptive.

```
{
  "taskCreated": {
    "id": 47
  }
}
```

No Description

FIELDS

taskCreated: TaskCreatedEvent

- Default subscription URL: ws://localhost:8080/subscriptions)

Disadvantages?

- HTTP status code is always 200
  - Custom error handling based on message
- No caching at HTTP level
- Complexity?
  - Types, Queries, Mutations, Resolvers…
- No Zuul-like API Gateway on the JVM yet (@see Apollo Federation)

What else?

- Fragments, unions, interfaces, directives… → https://graphql.org
- Clients:
  - For testing: GraphiQL, Altair, GraphQL Playground, Postman 7.2+
  - For front-end → Apollo Client: https://www.apollographql.com
- Star Wars GraphQL API: https://graphql.org/swapi-graphql

Thank you!

Questions?

Personal
Programmer