Subscriptions

Subscription is just another GraphQL operation type like Query and Mutation. It allows creating real-time subscriptions over a bidirectional transport layer, mainly over websockets. Read more about the subscriptions **here**. Below is a `commentAdded` subscription example, copied directly from the official **Apollo** documentation:

```
Subscription: {
  commentAdded: {
    subscribe: () => pubSub.asyncIterator('commentAdded');
  }
}
```

> **NOTICE**
> The `pubsub` is an instance of `PubSub` class. Read more about it **here**.

Schema first

To create an equivalent subscription in Nest, we'll make use of the `@Subscription()` decorator.

```
const pubSub = new PubSub();

@Resolver('Author')
export class AuthorResolver {
  constructor(
    private readonly authorsService: AuthorsService,
    private readonly postsService: PostsService,
  ) {}

  @Query('author')
  async getAuthor(@Args('id') id: number) {
    return await this.authorsService.findOneById(id);
  }

  @ResolveProperty('posts')
  async getPosts(@Parent() author) {
    const { id } = author;
    return await this.postsService.findAll({ authorId: id });
  }
```

```
  @Subscription()
  commentAdded() {
    return pubSub.asyncIterator('commentAdded');
  }
}
```

In order to filter out specific events based on context and arguments, we can set a `filter` property.

```
@Subscription('commentAdded', {
  filter: (payload, variables) =>
    payload.commentAdded.repositoryName === variables.repoFullName,
})
commentAdded() {
  return pubSub.asyncIterator('commentAdded');
}
```

To mutate the published payload, we can use a `resolve` function.

```
@Subscription('commentAdded', {
  resolve: value => value,
})
commentAdded() {
  return pubSub.asyncIterator('commentAdded');
}
```

Type definitions

The last step is to update type definitions file.

```
type Author {
  id: Int!
  firstName: String
  lastName: String
  posts: [Post]
}

type Post {
  id: Int!
  title: String
  votes: Int
```

```
}

type Query {
  author(id: Int!): Author
}

type Comment {
  id: String
  content: String
}

type Subscription {
  commentAdded(repoFullName: String!): Comment
}
```

Well done. We created a single `commentAdded(repoFullName: String!): Comment` subscription. You can find a full sample implementation **here**.

Code first

To create a subscription using the class-first approach, we'll make use of the `@Subscription()` decorator.

```
const pubSub = new PubSub();

@Resolver('Author')
export class AuthorResolver {
  constructor(
    private readonly authorsService: AuthorsService,
    private readonly postsService: PostsService,
  ) {}

  @Query(returns => Author, { name: 'author' })
  async getAuthor(@Args({ name: 'id', type: () => Int }) id: number) {
    return await this.authorsService.findOneById(id);
  }

  @ResolveProperty('posts')
  async getPosts(@Parent() author) {
    const { id } = author;
    return await this.postsService.findAll({ authorId: id });
  }

  @Subscription(returns => Comment)
  commentAdded() {
    return pubSub.asyncIterator('commentAdded');
  }
```

```
}
```

In order to filter out specific events based on context and arguments, we can set a `filter` property.

```
@Subscription(returns => Comment, {
  filter: (payload, variables) =>
    payload.commentAdded.repositoryName === variables.repoFullName,
})
commentAdded() {
  return pubSub.asyncIterator('commentAdded');
}
```

To mutate the published payload, we can use a `resolve` function.

```
@Subscription(returns => Comment, {
  resolve: value => value,
})
commentAdded() {
  return pubSub.asyncIterator('commentAdded');
}
```

PubSub

We used a local `PubSub` instance here. Instead, we should define `PubSub` as a **provider**, inject it through the constructor (using `@Inject()` decorator), and reuse it among the whole application. You can read more about Nest custom providers **here**.

```
{
  provide: 'PUB_SUB',
  useValue: new PubSub(),
}
```

Module

In order to enable subscriptions, we have to set `installSubscriptionHandlers` property to `true` .

```
GraphQLModule.forRoot({
```

```
  typePaths: ['./**/*.graphql'],
  installSubscriptionHandlers: true,
}),
```

To customize the subscriptions server (e.g. change port), you can use `subscriptions` property (read more).

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more here.

Principal Sponsor

Sponsors / Partners

Become a sponsor