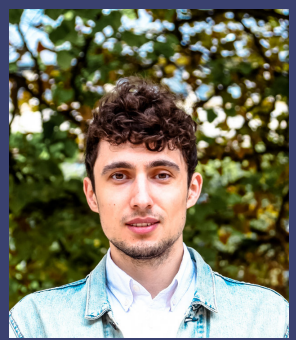
Build React Forms from GraphQL API

#whoami

Charly POLY - Sr Software Engineer at algolia

Past

- ➡️ ex-Dashlane
- ➡️ ex-JobTeaser

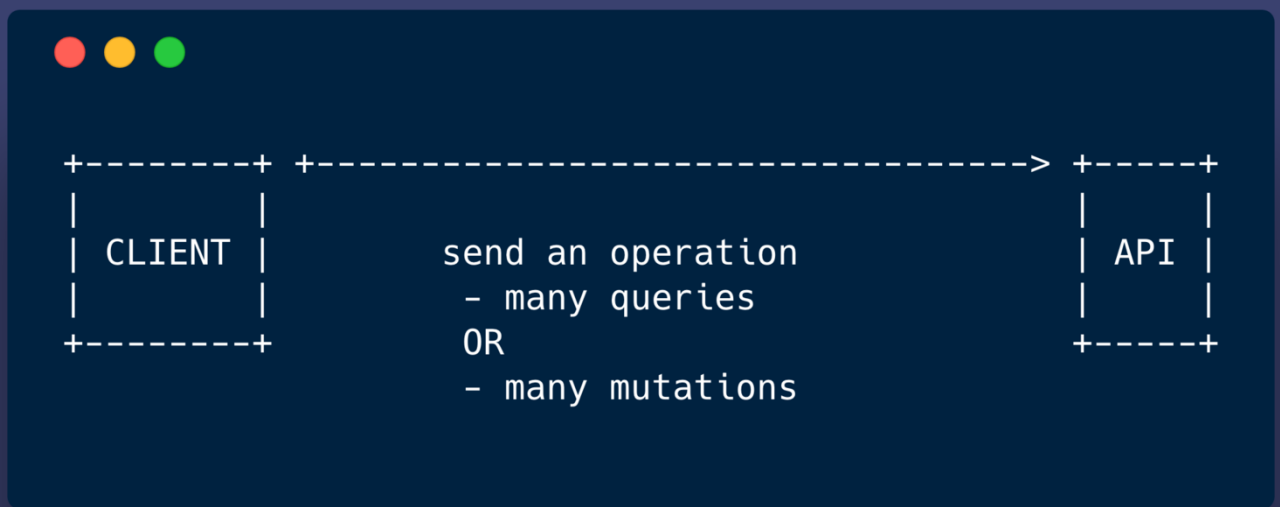


GraphQL in 5 minutes

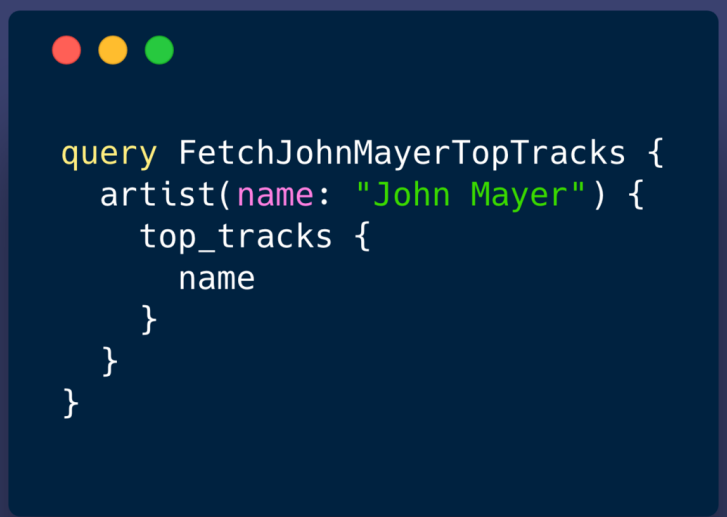
```
type Track {
  id: String
  album(full: Int): Album
  artists(full: Int, throttle: Int): [Artist]
  duration_ms: Int
  explicit: Boolean
  name: String
  popularity: Int
  track_number: Int
  type: String
}

# the schema allows the following query:
type Query {
  artist(id: String, name: String): Artist
}
# the schema allows the following mutation:
type Mutation {
  play_song(id: String!): Boolean
}
# we need to tell the server which types represent the root query
# and root mutation types. We call them RootQuery and RootMutation by convention.
schema {
  query: Query
  mutation: Mutation
}
```

- **Query**
  ➡️ fetch data
- **Mutation**
  ➡️ mutate data
- **Types**
  ➡️ optional by default

GraphQL in 5 minutes



GraphQL in 5 minutes



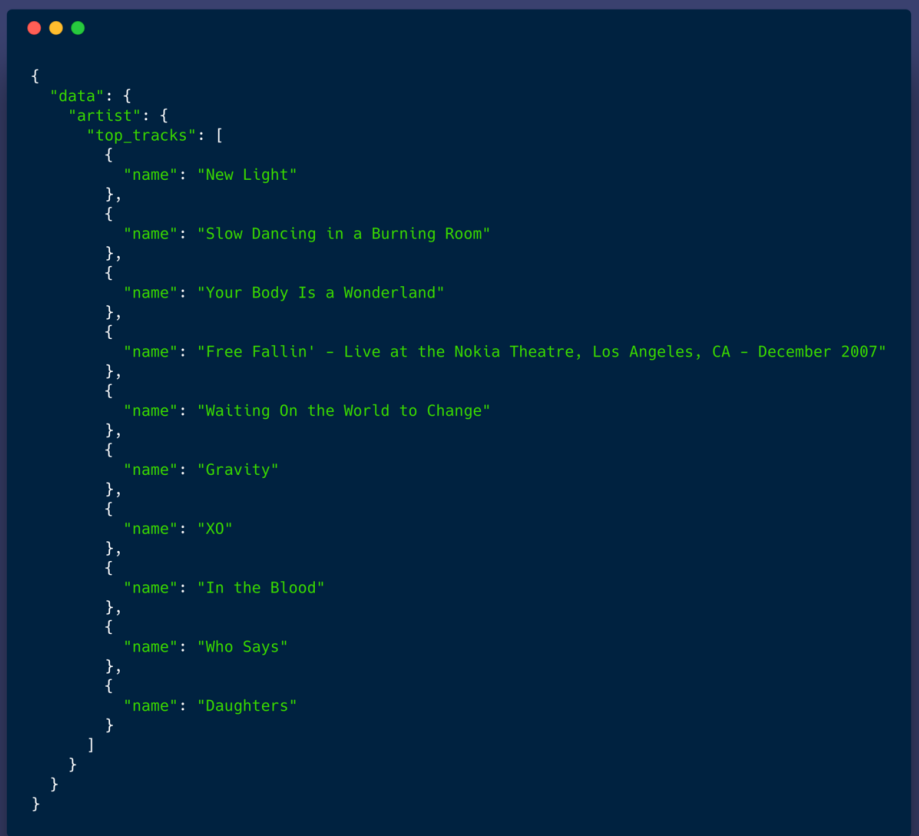
```
query FetchJohnMayerTopTracks {
  artist(name: "John Mayer") {
    top_tracks {
      name
    }
  }
}
```

We describe a query, to **retrieve** data from the API.

We want an artist, named "John Mayer".
And for this artist,
retrieve top tracks names.

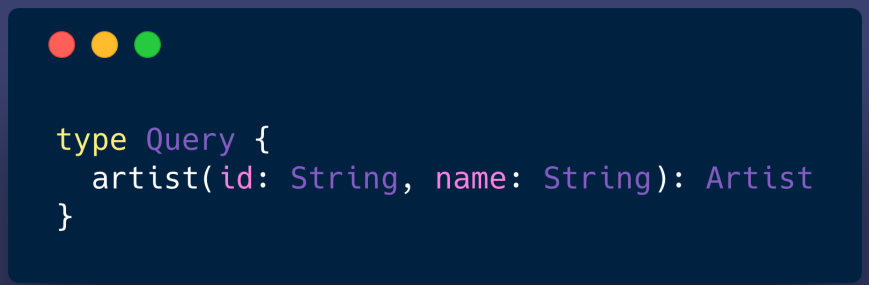
GraphQL in 5 minutes

```
query FetchJohnMayerTopTracks {
  artist(name: "John Mayer") {
    top_tracks {
      name
    }
  }
}
```



Demo: https://spotify-api-graphql-console.herokuapp.com/

GraphQL in 5 minutes



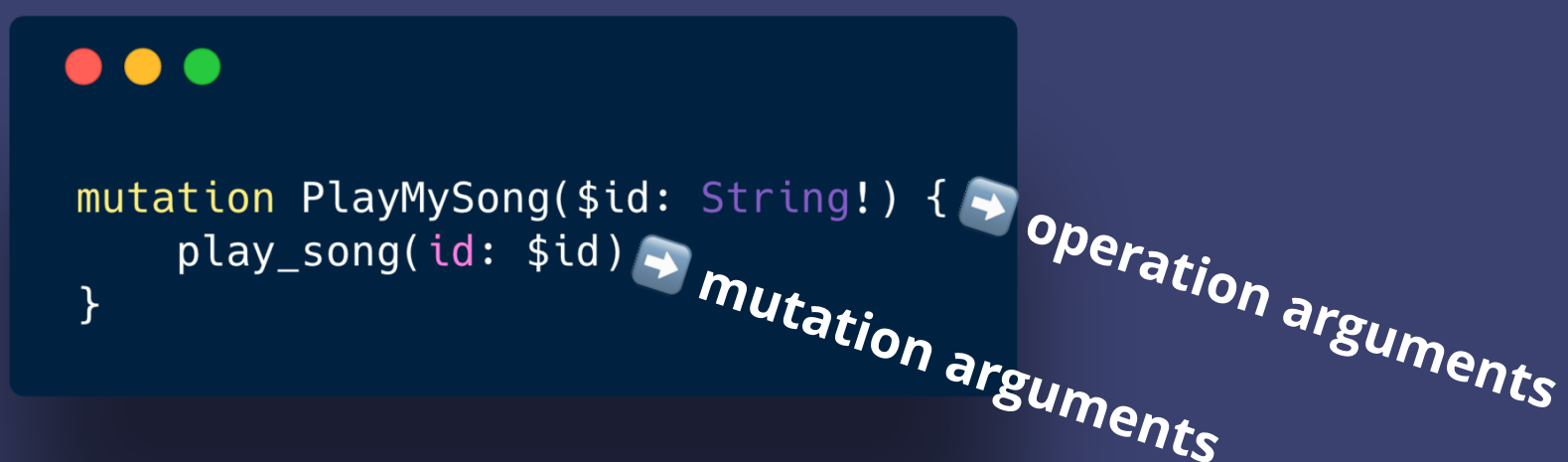
An **Artist** can be retrieved using an **id** or a **name.**

Depending on the arguments, the API resolvers will call different Spotify REST API endpoints.

- from **id** ➡️ searchArtists()
- from **name** ➡️ getArtist()

Here, GraphQL is used as an **abstraction** for the JS clients

The mutation



Observation: React forms nowadays

redux-form

- lot of configuration ✍️
- validators by hand ✍️
- forms state in global state
- Field by Field building workflow ✍️

Formik

- lot of configuration
- validators by hand ✍️
- non-standard validation format
- Field by Field building workflow ✍️

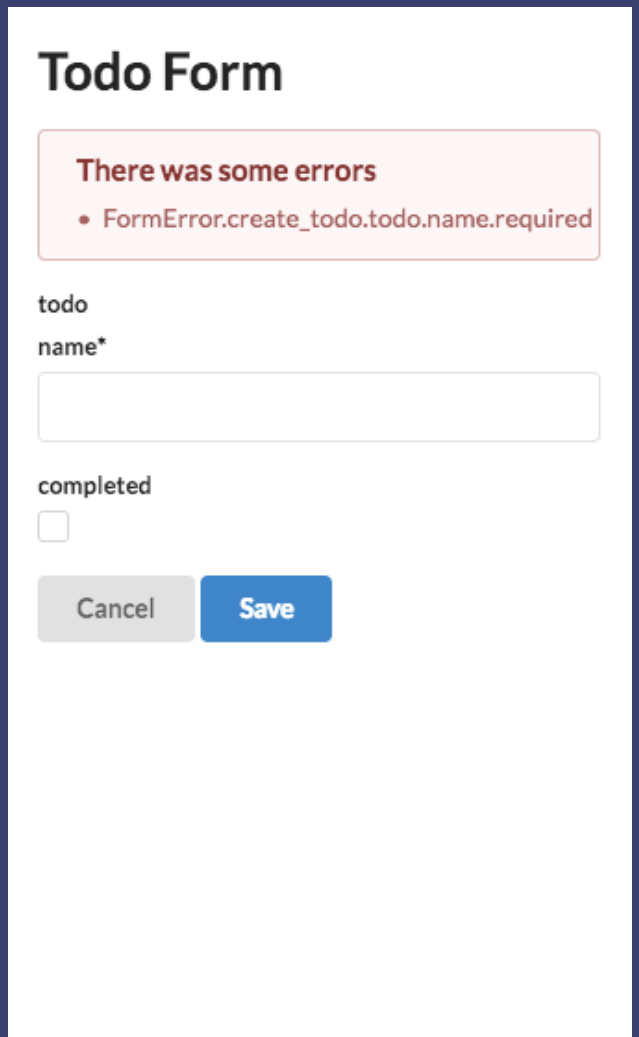
apollo-react-form 

apollo-react-form 

```
import * as React from 'react';
import gql from 'graphql-tag';
import { ApplicationForm } from './forms';

const createTodoMutationDocument = gql`
    mutation createTodo($todo: TodoInputType!) {
        create_todo(todo: $todo) {
            id
        }
    }
`;

const form = p => (
    <ApplicationForm
        title="Todo Form"
        liveValidate={true}
        config={{
            mutation: {
                name: 'create_todo',
                document: createTodoMutationDocument
            }
        }}
        data={{}}
        ui={{}}
     />
);
```

apollo-react-form 

```
import * as React from 'react';
import gql from 'graphql-tag';
import { ApplicationForm } from './forms';

const createTodoMutationDocument = gql`
    mutation createTodo($todo: TodoInputType!) {
        create_todo(todo: $todo) {
            id
        }
    }
`;

const form = p => (
    <ApplicationForm
        title="Todo Form"
        liveValidate={true}
        config={{
            mutation: {
                name: 'create_todo',
                document: createTodoMutationDocument
            }
        }}
        data={{}}
        ui={{}}
     />
);
```



apollo-react-form 👨‍🚀

Compared to redux-form and Formik

- minimal and extendable configuration
- validators imported from GraphQL mutations
- standard validation format using JSON Schema
- Form bootstraping, ability to customise rendering with render props

apollo-react-form 👨‍🚀

How does it works?

- using **react-jsonschema-form** from Mozilla
- translate a **GraphQL Schema** to **JSON Schema** using an **introspection query**
- create a Apollo wrapper around **react-jsonschema-form**

apollo-react-form 👨‍🚀

What is an introspection query?

Spotify GraphQL Console

Docs

```
query IntrospectionQuery {
      __schema {
        queryType { name }
        mutationType { name }
        subscriptionType { name }
        types {
          ...FullType
        }
        directives {
          name
          description
          locations
          args {
            ...InputValue
          }
        }
      }
    }
    fragment FullType on __Type {
      kind
      name
      description
      fields(includeDeprecated: true) {
        name
        description
        args {
          ...InputValue
        }
        type {
          ...TypeRef
        }
        isDeprecated
        deprecationReason
      }
      inputFields {
        ...InputValue
      }
      interfaces {
```

QUERY VARIABLES

```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      },
      "mutationType": null,
      "subscriptionType": null,
      "types": [
        {
          "kind": "OBJECT",
          "name": "Query",
          "description": "the schema allows the following query:",
          "fields": [
            {
              "name": "me",
              "description": "",
              "args": [],
              "type": {
                "kind": "OBJECT",
                "name": "PrivateUser",
                "ofType": null
              },
              "isDeprecated": false,
              "deprecationReason": null
            },
            {
              "name": "user",
              "description": "",
              "args": [
                {
                  "name": "id",
                  "description": "",
                  "type": {
                    "kind": "NON_NULL",
                    "name": null,
                    "ofType": {
```

Powered with Spotify | Open source code | About us

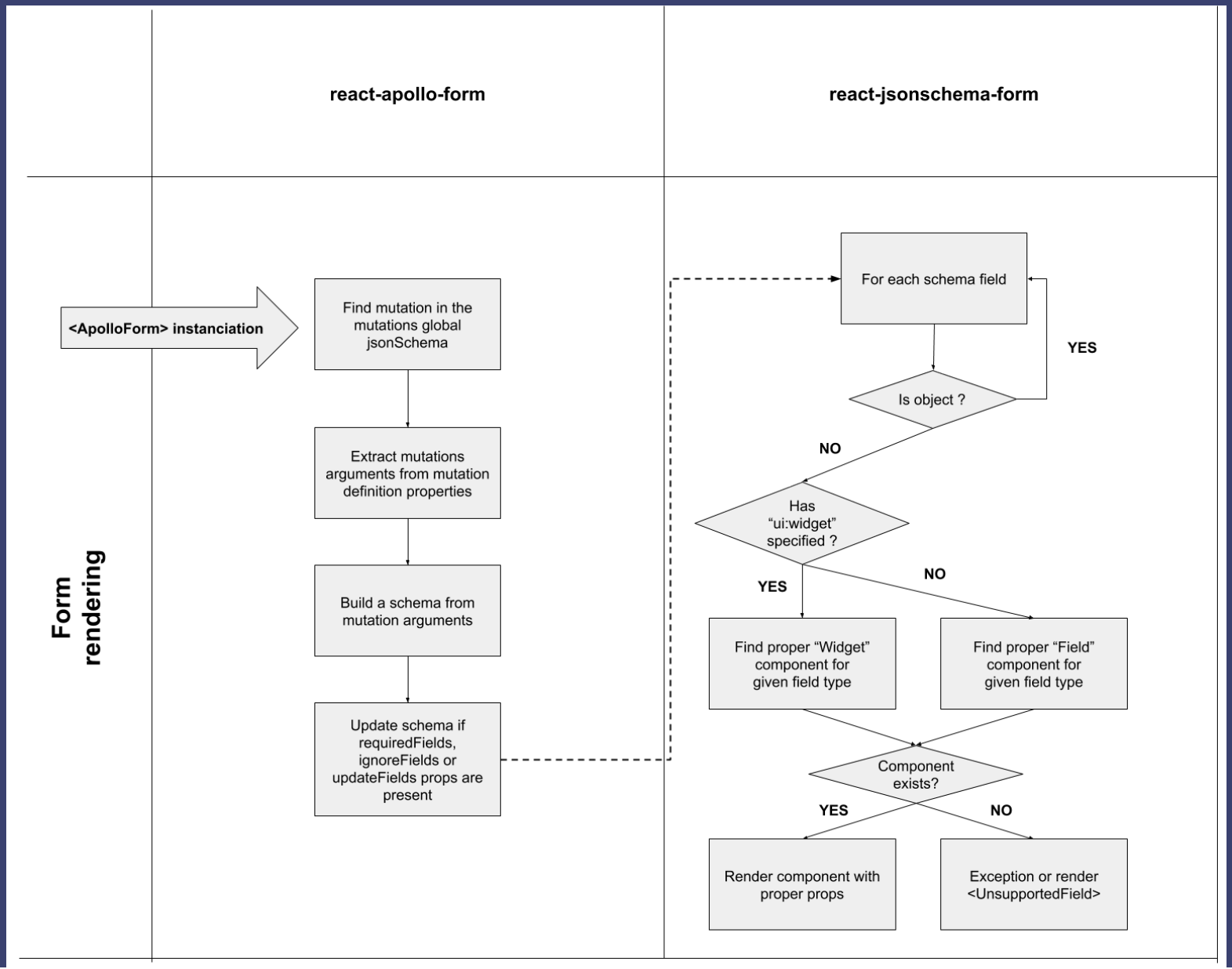
apollo-react-form 

What is a JSON Schema?

```
type Todo {
    id: String!
    name: String!
    completed: Boolean
}

input TodoInputType {
    name: String!
    completed: Boolean
}

type Query {
    todo(id: String!): Todo
    todos: [Todo]
}

type Mutation {
    update_todo(id: String!, todo: TodoInputType!): Todo
    create_todo(todo: TodoInputType!): Todo
}
```

```
{
    $schema: 'http://json-schema.org/draft-06/schema#',
    properties: {
        /* Query ... */
        Mutation: {
            type: 'object',
            properties: {
                update_todo: {
                    type: 'object',
                    properties: {
                        arguments: {
                            type: 'object',
                            properties: {
                                id: { type: 'string' },
                                todo: { $ref: '#/definitions/TodoInputType' }
                            },
                            required: ['id', 'todo']
                        },
                        return: {
                            $ref: '#/definitions/Todo'
                        }
                    },
                    required: []
                },
                create_todo: {
                    type: 'object',
                    properties: {
                        arguments: {
```

apollo-react-form 👨‍🚀



apollo-react-form 👨‍🚀

```
import * as React from 'react';
import gql from 'graphql-tag';
import { ApplicationForm } from './forms';

const createTodoMutationDocument = gql`
    mutation createTodo($todo: TodoInputType!) {
        create_todo(todo: $todo) {
            id
        }
    }
`;

const form = p => (
    <ApplicationForm
        title="Todo Form"
        liveValidate={true}
        config={{
            mutation: {
                name: 'create_todo',
                document: createTodoMutationDocument
            }
        }}
        data={{}}
        ui={{}}
     />
);
```

Benefits

- **API/Client Interoperability**

  map your forms to existing mutations with ease
- **Simplicity**

  only specify what you need, in a functional way
- **Extendable**

  Remove and replace anything if you need
- **State management and validation**

  drop redux state and use form local state

Conclusion

- **GraphQL**, a query language for your API

- **GraphQL** is still new, there is a lot of use-cases to find!
  - Curious?
    ➡️ Look for Apollo Link State, GraphQL SSR, GraphQL on server-side

  - Feel adventurous?
    ➡️ Start playing with GraphQL!

Thanks for listening!

 @whereischarly

 /wittydeveloper

 @wittydeveloper