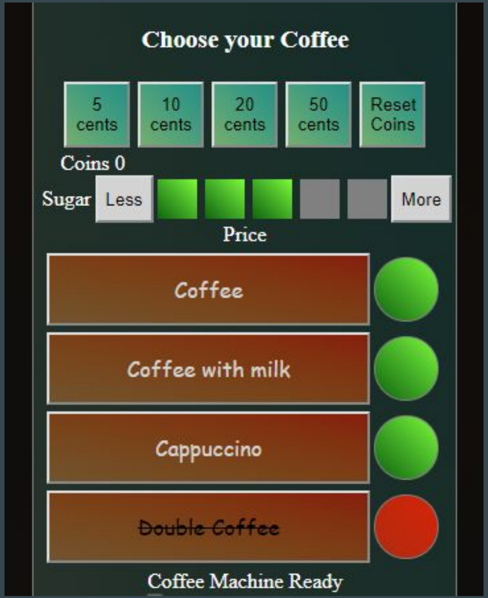
GraphQL for Coffee Machine

Query Language for Coffee Machine APIs

The Coffee Machine - Functions

The Coffee Machine allows you to:

- Show a list of coffees
- Check the price
- Add coins
- Reset coins
- Choose the sugar
- Choose the coffee



Once you put enough coins and push the coffee button, the Coffee Machine gives you the coffee.

The Coffee Machine - Coffee Type

```
type Coffee {
        name: String!
        price: Int!
        sugar: Int
       milk: Int
       coffeePowder: Int
       available: Boolean
}
```

The Coffee Machine queries a subset of data to get a list of coffees:

- name
- price
- available

When the coffee is ready, the Coffee Machine gives you the coffee, querying another subset of data:

- sugar
- milk
- coffeePowder

The type doesn’t change, the query changes.

The Coffee Machine - Server

Query

```
type Query {
        coffees: [Coffee]
        coffee(name: String!, sugar: Int, coins Int!): Coffee
}
```

Resolvers

```
Query: {
      coffees: () => {
        return defaultCoffeeService.getCoffees().concat(customCoffeeService.getCoffees());
      },
      coffee: (name: string) => {
        return
          defaultCoffeeService.exists(name) ?
          defaultCoffeeService.prepareCoffee(name) :
          customCoffeeService.prepareCoffee(name);
      }
}
```

The Coffee Machine - Client

```
query {
        getCoffees {
                coffees {
                        name
                        price
                        available
                }
        }

        prepareCoffee($name: String!, $sugar: Int, $coins: Int!) {
             coffee(name: $name, sugar: $sugar, coins: $coins) {
                coffeePowder
                milk
                sugar
             }
       }
}
```

What can we deduce about GraphQL?

What is it?

- Syntax that describes how to ask for data
- Runtime for fulfilling those queries with your existing data

What it does?

- Provides a complete and understandable description of data
- Gives clients the power to ask for exactly what they need
- Makes easy to aggregate data coming from different sources

What can we deduce about GraphQL?

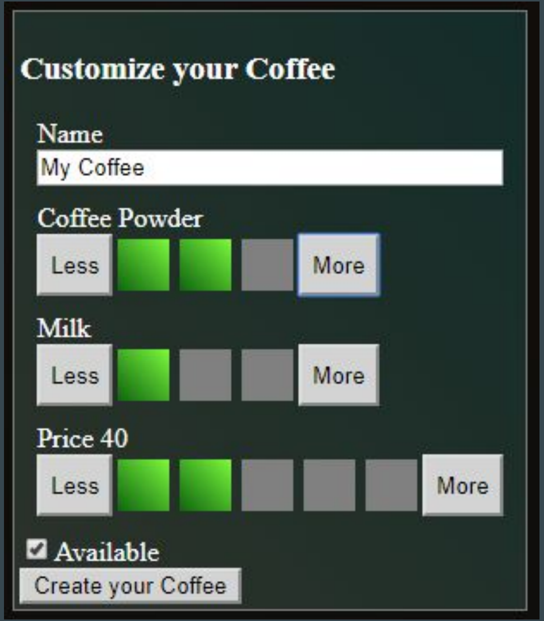
What are its features?

- *Queries mirror their response* => easy to predict the returned data shape
- *Hierarchical* => match better with structured data and with user interface
- *Strongly typed (Schema)* => descriptive error messages before executing queries
- *Introspective* => the GraphQL server can be queried for the types it supports
- *Version free* => the data shape is defined by the client query
- *Client Driven* => the client queries what it needs

The Coffee Dashboard - Functions

The Dashboard allows you to create your coffee:

- Choosing a unique name
- Setting the coffee powder quantity
- Setting the milk quantity
- Fixing the price
- Making it available/unavailable



Customize your Coffee

Name
My Coffee

Coffee Powder
Less 🟩 🟩 ⬜ More

Milk
Less 🟩 ⬜ ⬜ More

Price 40
Less 🟩 🟩 ⬜ ⬜ ⬜ More

☑ Available

Create your Coffee

Once you choose all settings, you can create your custom coffee. It will appear in the list.

The Coffee Dashboard - Server

Mutation and Subscription

```
type Mutation {
            create(name: String!, coffeePowder: Int, milk: Int, sugar: Int, price: Int!, available: Boolean): Boolean
}
type Subscription {
        coffeeCreated: Coffee
```

Resolvers

```
Mutation: {
    create: (root: any, coffee: Coffee) => {
        let created = customCoffeeService.addCoffee(coffee);
        if (created)
            pubsub.publish(COFFEE_CREATED, { coffeeCreated: { name: coffee.name, available: coffee.available } });
        return created;
    }
},
Subscription: {
    coffeeCreated: {
        subscribe: () => pubsub.asyncIterator([COFFEE_CREATED])
    },
}
```

The Coffee Dashboard - Client

```
mutation createCustomCoffee($name: String!, $coffeePowder: Int, $milk: Int, $sugar: Int, $price: Int!, $available: Boolean) {
        create(name: $name, coffeePowder: $coffeePowder, milk: $milk, sugar: $sugar, price: $price, available: $available)
}

subscription coffeeCreated {
        coffeeCreated {
            name
            price
            available
}
```

Let’s put it all together - Definitions

Operation Types

- Query
- Mutation
- Subscription (WebSockets)

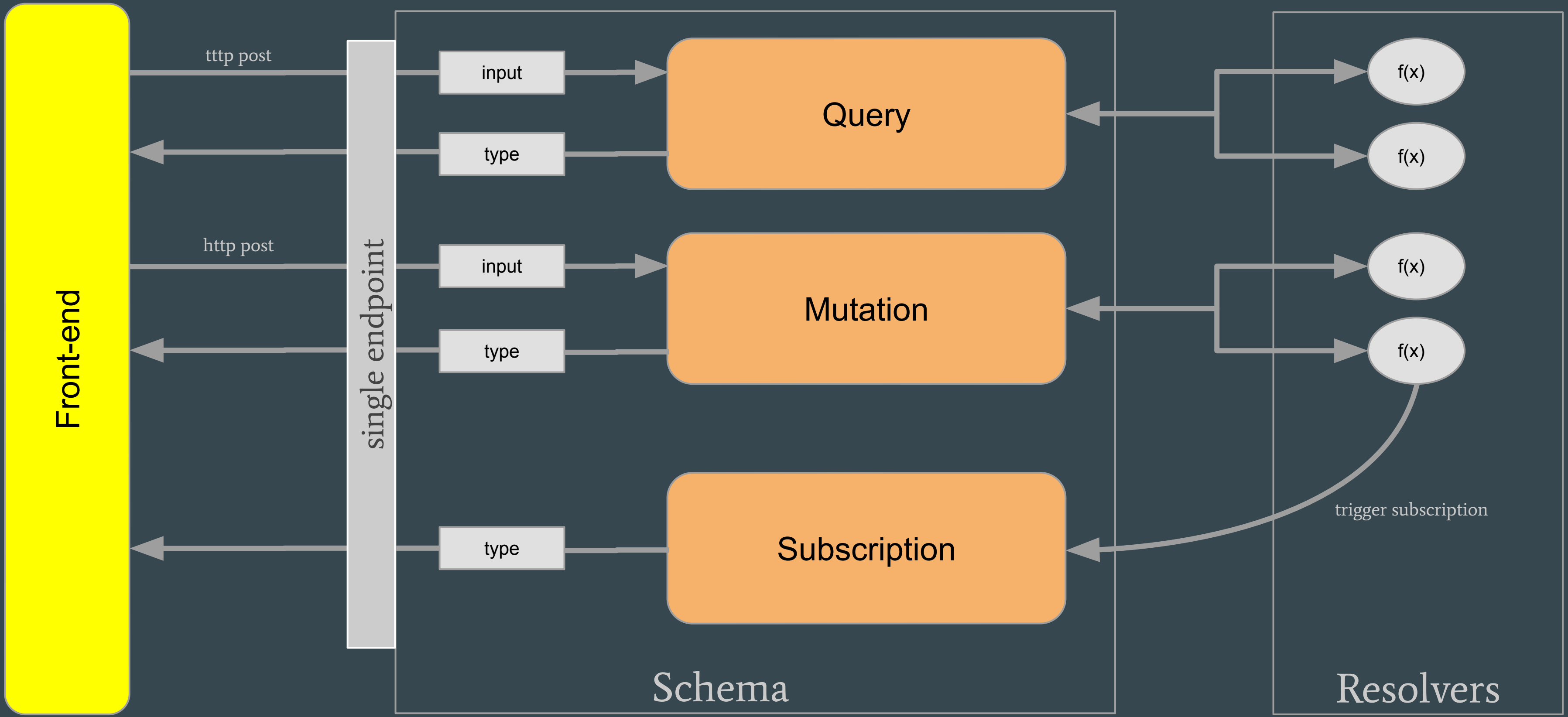
Operation Endpoints

Functions that clients can call

Requested Fields

Subset of fields of a data

Let’s put it all together - Diagram



Let’s put it all together - HTTP Post

Notes on GraphQL calls:

- There is only one endpoint and more operation endpoints.
- The client call the single endpoint, specifying the operation endpoint.
- Both Queries and Mutations operations are Post.
- The Query Expression is put in the Request Body.

References

- https://github.com/antpass79/graphql-for-coffee-machine
- https://medium.com/devgorilla/what-is-graphql-f0902a959e4
- https://honest.engineering/posts/why-use-graphql-good-and-bad-reasons