Introduction to GraphQL

shopify

Hello! My name is

Jenna Blumenthal

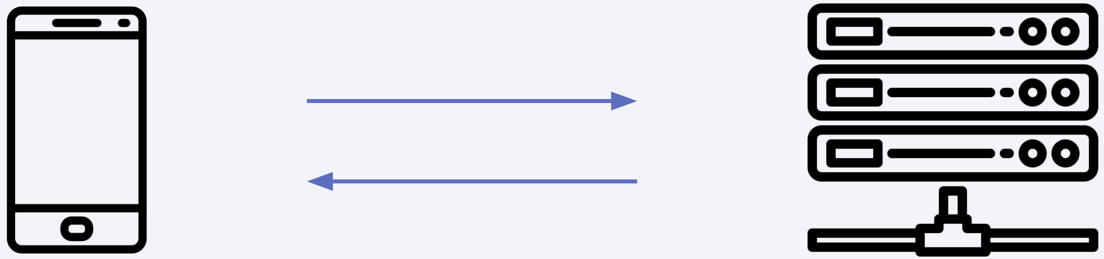
I’m a software developer at Shopify

@jennaleeblume
jenna.blumenthal@shopify.com

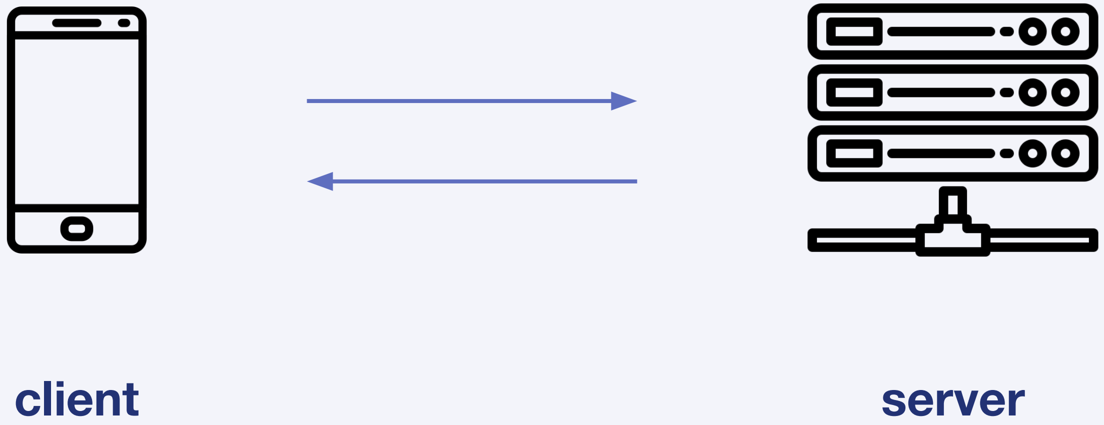
What we’re going to learn today 👩‍💻

- What is GraphQL (and what it’s not)
- Basics of HTTP & client-server architecture
- REST vs GraphQL APIs
- Fetching data from an existing GraphQL API
- Creating our own (!) GraphQL API

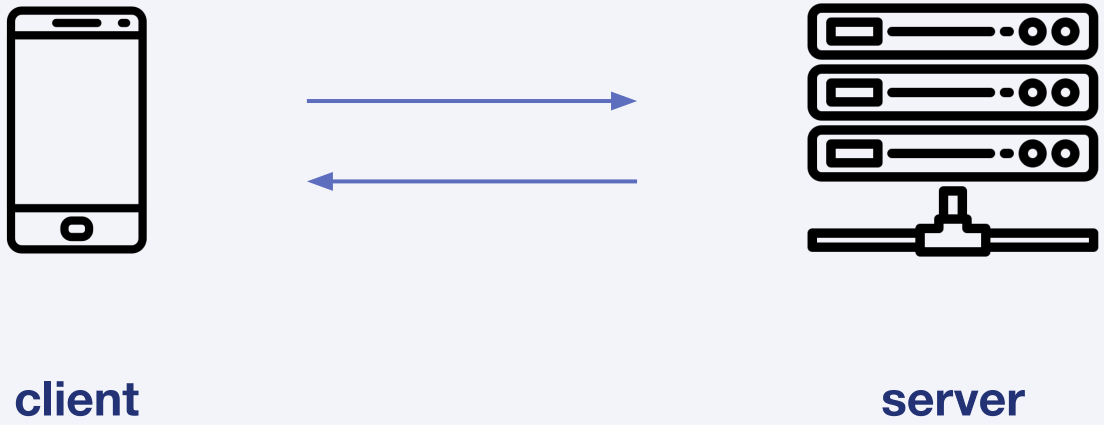
API



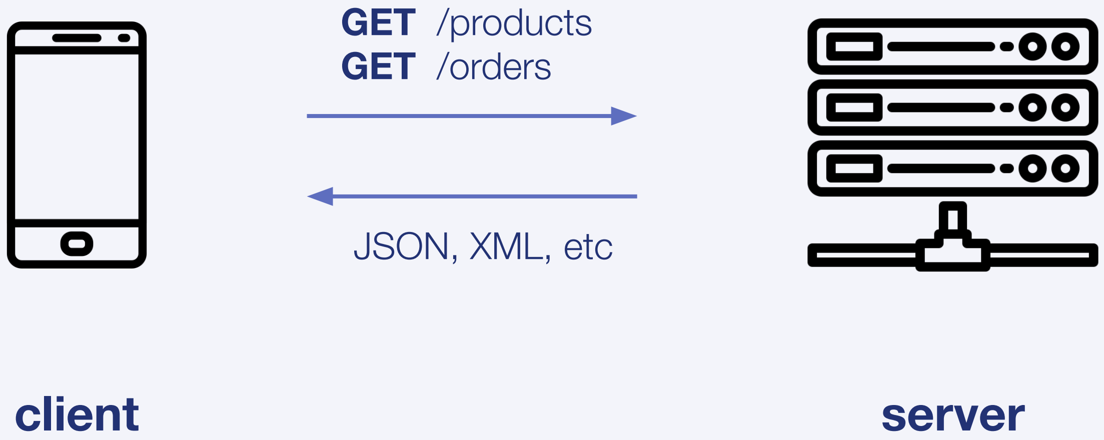
API



REST API



REST API



REST API

Implement **CRUD** actions (create, read, update, delete) via **HTTP** requests (POST, GET, PUT, DELETE)

HTTP

HTTP method	**URI**
POST	/products

HTTP

HTTP method	**URI**	**CRUD operation**
POST	/products	**Create** a new product

HTTP

HTTP method	**URI**	**CRUD operation**
POST	/products	**Create** a new product
GET	/products	**Read** all products

HTTP

HTTP method	**URI**	**CRUD operation**
POST	/products	**Create** a new product
GET	/products	**Read** all products
	/product/:id	**Read** specific product

HTTP

HTTP method	**URI**	**CRUD operation**
POST	/products	**Create** a new product
GET	/products	**Read** all products
	/product/:id	**Read** specific product
PUT	/product/:id	**Update** a product
DELETE	/product/:id	**Delete** a product

GET

shopify.com/api/products

```
{
  data: {
    products: [
      {
        id: 1,
        image_url: "cdn.shopify.com/1",
        title: 'denim jeans',
        price: 100.00,
      },
      {
        id: 2,
        image_url: "cdn.shopify.com/2",
        title: 't-shirt',
        price: 20.00,
      },
      {
        id: 3,
        image_url: "cdn.shopify.com/3",
        title: 'scrunchie',
        price: 5.00,
      }
    ]
  }
}
```

Problems with REST

- ✗ Number of endpoints (lots)
- ✗ Client has no control over what data is sent
- ✗ Overfetching
- ✗ Underfetching (& round-trips)
- ✗ Unknown structure of response data

GraphQL 💕

~~Programming language~~

~~Framework~~

~~Library~~

~~Data storage~~

Specification & query language

GraphQL

Receive exactly what you asked for.

```
{
  product {
    title
  }
}
```

query

```
{
  “product”: {
    “title”: "Jumpsuit"
  }
}
```

response

GraphQL

1. Client receives exactly what it asked for
2. Multiple resources in 1 request

GraphQL

Multiple resources in 1 request

```
{
  product {
    title
    price
    orders {
      total_cost
    }
  }
}
```

query

```
{
  “product”: {
    “title”: "Jumpsuit",
    “price”: “15.00”
    “orders”: [
      {
          “total_cost”: 50.00
      },
      {
          “total_cost”: 110.00
      },
    ]
  }
}
```

response

GraphQL

1. Client receives exactly what it asked for
2. Multiple resources in 1 request
3. Strongly typed

GraphQL

Strongly Typed

```
type Product {
  title: String
  image: Image
  orders: [Order]
}
```

GraphQL

Strongly Typed

```
type Product {
  title: String
  image: Image
  orders: [Order]
}
```

built-in scalars (string, int, float, etc)

GraphQL

Strongly Typed

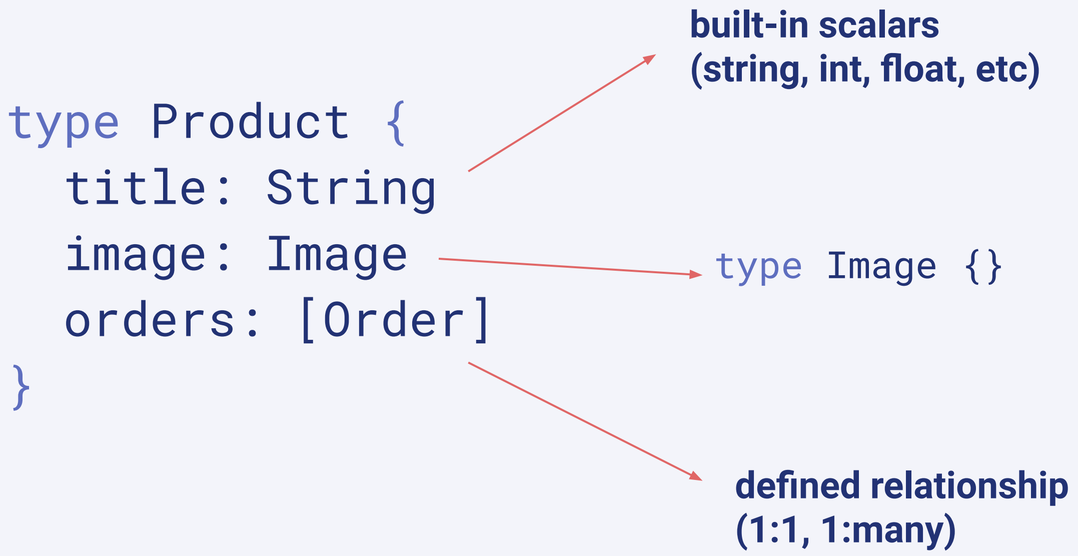
```
type Product {
  title: String
  image: Image
  orders: [Order]
}
```

built-in scalars (string, int, float, etc)

```
type Image {}
```

GraphQL

Strongly Typed



```
type Product {
  title: String
  image: Image
  orders: [Order]
}
```

GraphQL

1. Client receives exactly what it asked for
2. Multiple resources in 1 request
3. Strongly typed
4. Schema inspection & validation

GraphQL

Schema introspection & validation

```
{
  product {
    boop
  }
}
```

query

```
{
 "errors": [
   {
     "message": "Cannot query field
\"boop\" on type \"Product\"."
   }
 ]
}
```

response

GraphQL

1. Client receives exactly what it asked for
2. Multiple resources in 1 request
3. Strongly typed
4. Schema inspection & validation
5. Independent of language, application framework or data storage

GraphQL

Independent of data storage, **YOU** define how a field is **RESOLVED**

```
type Product {
  title: String
  image: Image
  orders: [Order]
}
```

```
class Product
 def title
   self.display_title.humanize
 end

 def image
   self.all_product_images.first
 end

 def orders(created_after)
   Order.where(
     product_id: self.id,
     created_after: created_after
   )
 end
end
```

Queries: fetch data

Mutations: create, update, delete data

```
mutation {
 createProduct(input: {
   title: "An even nicer jumpsuit",
   price: "100.00"
 }) {
   product {
     id
   }
 }
}
```

```
{
  “product”: {
    “id”: 91802
  }
}
```

mutation

response

Hands on: client-side

https://developer.github.com/v4/explorer

Step 1: Authenticate with Github & write our first query

```
query {
  viewer {
    login
  }
}
```

Step 2: Another query - this time, with variables!

```
query {

}
```

Step 2: Another query - this time, with variables!

```
query {
  user(login: "eileencodes") {
    bio
  }
}
```

Step 3: Connection fields & pagination

```
query {

}
```

Step 3: Connection fields & pagination

```
query {
  user(login: "eileencodes") {
    bio
    avatarUrl
    repositoriesContributedTo(first: 5) {
      pageInfo {
        endCursor
        hasNextPage
      }
      edges {
        cursor
        node {
          name
          description
        }
      }
    }
  }
}
```

Step 4: Exercise

```
query {
  repository(name: "intro-to-graphql-exercise", owner: "jennaleeb") {
    id
    description
    url
  }
}
```

Find all the **issues** associated with this repository

Step 4: Exercise

```
query {
  repository(name: "intro-to-graphql-exercise", owner: "jennaleeb") {
    id
    description
    url
    issues(first: 5) {
      nodes {
        title
        author {
          login
        }
      }
    }
  }
}
```

Step 4: Exercise

```
query {
  repository(name: "intro-to-graphql-exercise", owner: "jennaleeb") {
    id
    description
    url
    issues(first: 5) {
      nodes {
        title
        author {
          login
        }
      }
    }
  }
}
```

Find all the issues that are

- **CLOSED**
- have the label: **“bug”**

Find another open source project that interests you
https://github.com/collections

- Find the **REPOSITORY** (by name & owner)
- Find all the **LANGUAGES** in the repository

Step 4: Exercise

```
query {
  repository(name: "intro-to-graphql-exercise", owner: "jennaleeb") {
    id
    description
    url
    issues(first: 5, states: CLOSED, labels: ["bug"]) {
      edges {
        node {
          title
        }
      }
    }
  }
}
```

Step 4: Exercise

```
query {
  repository(name: "data", owner: "fivethirtyeight") {
    id
    languages(first: 10) {
      edges {
        node {
          name
        }
      }
    }
  }
}
```

Step 5: Mutations

```
mutation {
  createIssue(input: {
    repositoryId: "MDEwOlJlcG9zaXRvcnkyMzQ5ODI4MDA=",
    title: "i am an issue",
    body: "the issue, very important."
  }) {
    issue {
      title
    }
  }
}
```

Step 5: Mutations

```
mutation {
  createIssue(input: {
    repositoryId: "MDEwOlJlcG9zaXRvcnkyMzQ5ODI4MDA=",
    title: "i am an issue",
    body: "the issue, very important."
  }) {
    issue {
      title
    }
  }
}
```

Create an **issue** and **assign** it to yourself

Step 5: Mutations

```
query {
  viewer {
    id
  }
}
```

```
mutation {
  createIssue(
    input: {
      repositoryId: "MDEwOlJlcG9zaXRvcnkyMzQ5ODI4MDA=",
      title: "it is yet another issue",
      assigneeIds: ["MDQ6VXNlcjg4MjQ4MjQ="]
    }
  ) {
    issue {
      title
    }
  }
}
```

Hands on: server-side

https://glitch.com/~wise-intro-to-graphql

Step 1: Basic structure of Apollo Server

- Running on a Node app
- `server.js` (HTTP handling)
- `typeDefs.js` (define schema)
- `resolvers.js` (define how data is fetched & returned)

Step 2: Ping/Pong

```
query {
  ping
}
```

Step 3: Fetching data

```
query {
  getRestaurants {
    name
    location {
      city
      country
    }
    styles
  }
}
```

Step 3: Fetching data

```
query {
  getRestaurants {
    name
    location {
      city
      country
    }
    styles
  }
}
```

Add the field **website** to the Restaurant object, and return it!

Step 4: Queries with arguments

```
query {
  getReviews(restaurantName: "the sparrow") {
    numStars
    comment
  }
}
```

Step 4: Queries with arguments

```
query {
  getReviews(restaurantName: "the sparrow") {
    numStars
    comment
  }
}
```

Add a **city** variable to getRestaurants to return only restaurants from that city.

Step 5: Mutations

```
mutation {
  createRestaurant(
    input: {
      name: "fet zun"
    }
  ) {
    name
  }
}
```

Step 5: Mutations

```graphql
mutation {
  createRestaurant(
    input: {
      name: "fet zun"
    }
  ) {
    name
  }
}
```

Add a **mutation** that creates a new **Review**

Thanks!

shopify