

Introduction to GraphQL

July 13, 2018

MidDevCon

Baltimore, MD

@ShopifyDevs

<http://developers.shopify.com>



Welcome! My name is Evan



Evan Huus, Developer Lead, Shopify

- I'm here to lead this session and help you learn all about GraphQL!
- I'm a Developer Lead at Shopify, in Canada
- My favorite programming language/tool is Golang
- Join us on the MidDevCon Slack in #introtographql
- Tweet along with the workshop and tag @ShopifyDevs!





MID
ATLANTIC
— DEV CON —



Cloud - Mobile - Web - Dev

Thanks to
Our Sponsors



Shout out: MLH Localhost



- Special thanks to Major League Hacking, who created these workshop materials
- Major League Hacking (MLH) powers over 200 weekend-long invention competitions that inspire innovation, cultivate communities and teach computer science skills to more than 65,000 students around the world.
- Localhost is their “between hackathon” workshop offering

<https://mlh.io/>
@MLHacks

What will you learn today?

1. Why are APIs important?
2. Explain the difference between RESTful and GraphQL APIs.
3. Write your first API calls using GraphQL.

Why does this matter?

1. APIs are a large part of how applications communicate with each other.
2. GraphQL is a new way to interact with APIs.
3. GraphQL solves certain problems caused by RESTful APIs.

Table of contents

- 1. Introduction to APIs and GraphQL**
2. Preview the app
3. Set up Shopify storefront
4. Write your own GraphQL calls
5. Review
6. Next steps

Why are APIs important?

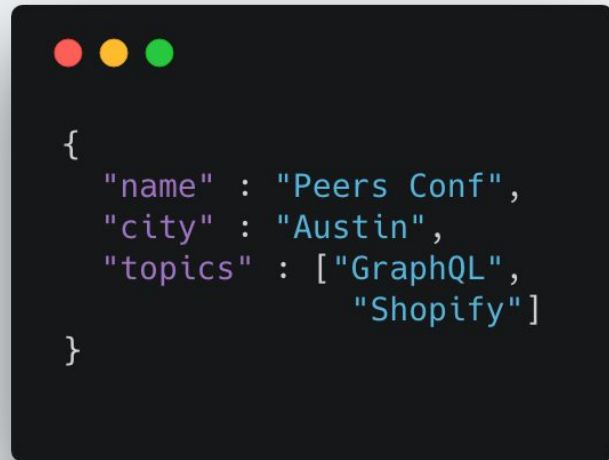
Example: Google APIs

- Google offers a number of APIs that developers can use in their own applications:
- Google Maps API
- Google Fonts API
- Google URL Shortener API
- and many more!

For example, an application that you use to find a restaurant might use the Google Maps API to show you restaurants near your current location.

What is REST?

- REST stands for **Representational State Transfer**.
- Most modern APIs are RESTful APIs.
- Every resource has its own URL.
- Data is generally returned in JSON
(JavaScript Object Notation) ->

A dark-themed code editor window with three colored window control buttons (red, yellow, green) at the top left. The editor contains a JSON object with the following structure: an opening curly brace, a "name" property with value "Peers Conf", a "city" property with value "Austin", a "topics" property with an array of values ["GraphQL", "Shopify"], and a closing curly brace. The text is color-coded: strings are in light blue, the array is in light purple, and the opening and closing braces are in light grey.

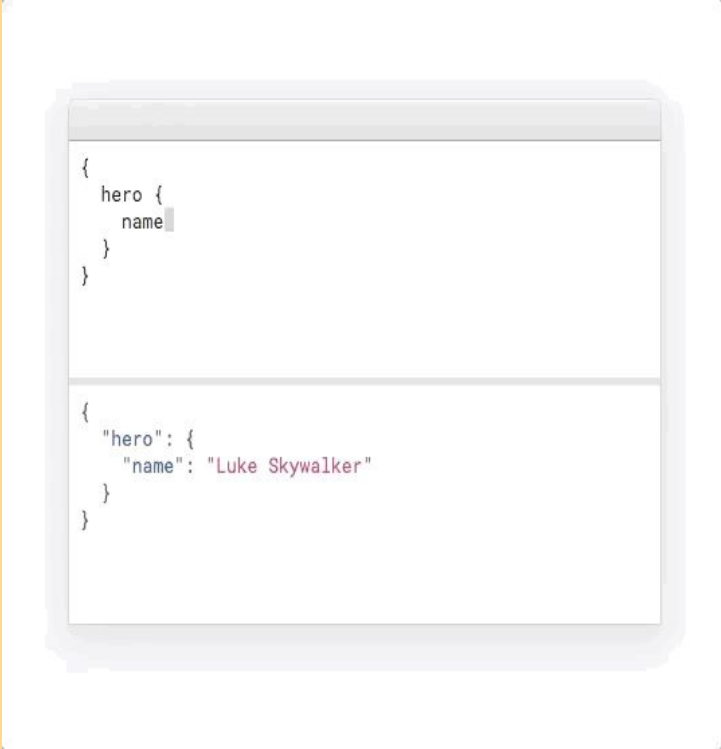
```
{  
  "name" : "Peers Conf",  
  "city" : "Austin",  
  "topics" : ["GraphQL",  
              "Shopify"]  
}
```

What is GraphQL?

GraphQL stands for **Graph Query Language**.

APIs written using GraphQL schemas have only one endpoint that return a data graph

When using a GraphQL API, an application can request multiple resources at a time and only receive what it needs. The information returned looks like this:

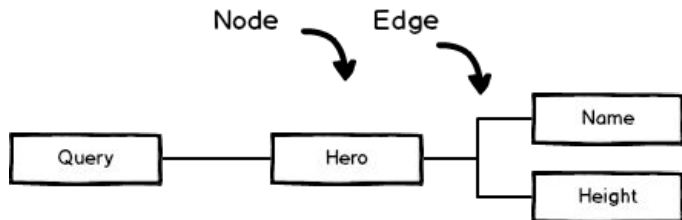


```
{
  hero {
    name
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

The graph in GraphQL

A partial data graph

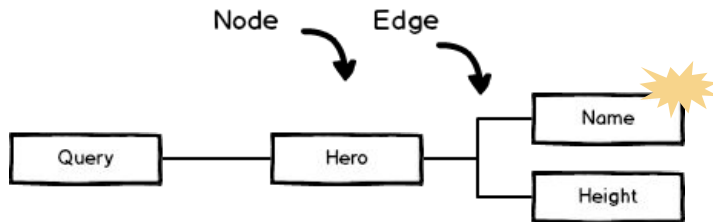


GraphQL schema

```
type Hero {  
  name: String  
  height: Float  
}
```

GraphQL execution

A partial data graph

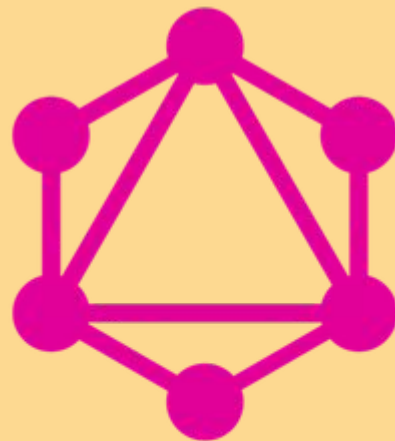


Resolver

```
name(hero) {  
  return hero.name  
}
```

Benefits of GraphQL

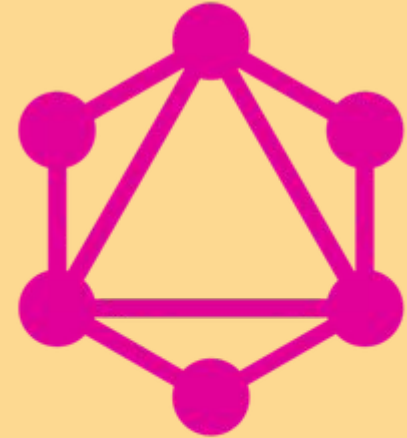
- GraphQL allows you to specify the information you want to retrieve.
- GraphQL has a feature called "introspection" where developers can use commands to ask the server about what queries are allowed.
- GraphQL allows you to retrieve information from multiple data sources in a single request, speeding up your web application.
- GraphQL can be implemented in many languages and there's a strong community to help!



GraphQL

GraphQL basics

- There are three types of GraphQL calls - queries, subscriptions, and mutations. In this workshop, you will explore one query and one mutation.
- **Query:** a GraphQL call that retrieves information from an application through an API.
- **Mutation:** A GraphQL call that updates information in an application's database through an API.



GraphQL

GraphQL query structure

A GraphQL query:

- begins with the keyword "query" followed by a set of curly braces
- has a Query Root as its first requested field
- In the query below, the Query Root user accepts an argument id that has a value of the user's id number
- This query is requesting the user's name, email, and birthday which are fields

```
01 query {  
02   user(id: "abc123") {  
03     name  
04     email  
06     birthday  
07   }  
08 }
```

GraphQL response structure

A GraphQL response:

- returns information in the same way that it was requested
- can return error messages if the queries are written incorrectly

```
01  {
02    "user": {
03      "name": "MidDevCon",
04      "email": "admin@middevcon.com",
06      "birthday": "July 13, 2018"
07    }
08  }
```


GraphQL users

Facebook created GraphQL. Coursera's engineering team was in the process of designing their own replacement for RESTful APIs when they found GraphQL and decided to use it instead.

Now, it's being used by Github, Pinterest, Coursera, and of course Shopify!



GraphQL at Shopify

- Shopify has been using GraphQL internally for several years.
- Shopify now has two public GraphQL APIs.
- Today we'll be using the Storefront API which is designed for building customer-facing shopping flows.
- The other one is the Admin API which is for building merchant-facing applications.



Table of contents

1. Introduction to APIs and GraphQL
- 2. Preview the app**
3. Set up Shopify storefront
4. Write your own GraphQL calls
5. Review
6. Next steps

Try the demo application:

<https://nl-localhost-shopify.herokuapp.com/>

Goal:

Purchase power- in a game using the Shopify Storefront API

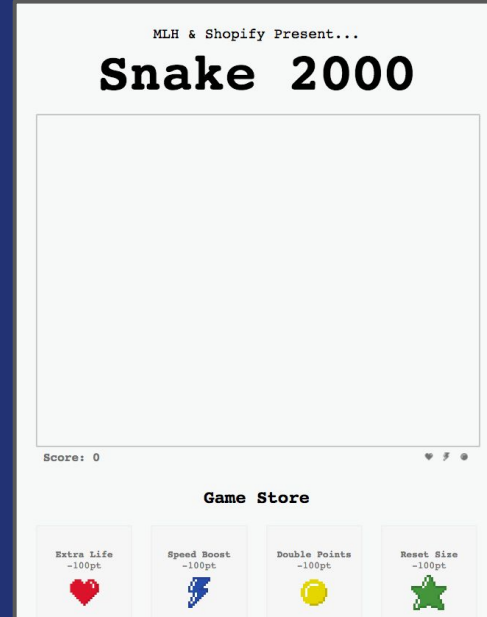
Technologies:

Node.js

JavaScript

HTML / CSS

GraphQL



Step1: install Node



Follow the installation instructions for the type of computer you have at the following URL:

<https://nodejs.org/en/download/>

Let me know if you have any trouble!

Step 2: download the sample code

To get the sample code, enter this URL in your browser:

<https://bit.ly/GraphQLIntro>

Let me know if you have any trouble!

Step 3: unzip files

```
$ cd ~/Downloads  
$ Expand-Archive mlh-localhost-shopify-graphql-master.zip .
```

Windows

Do not forget
the "." in this
command

```
$ cd ~/Downloads  
$ unzip mlh-localhost-shopify-graphql-master.zip
```

Mac

Step 4: run the Node server

Mac and Windows



```
$ cd mlh-localhost-shopify-graphql-master
```

```
$ ls
```

```
README.md  node_modules/  package.json  public/  server.js
```

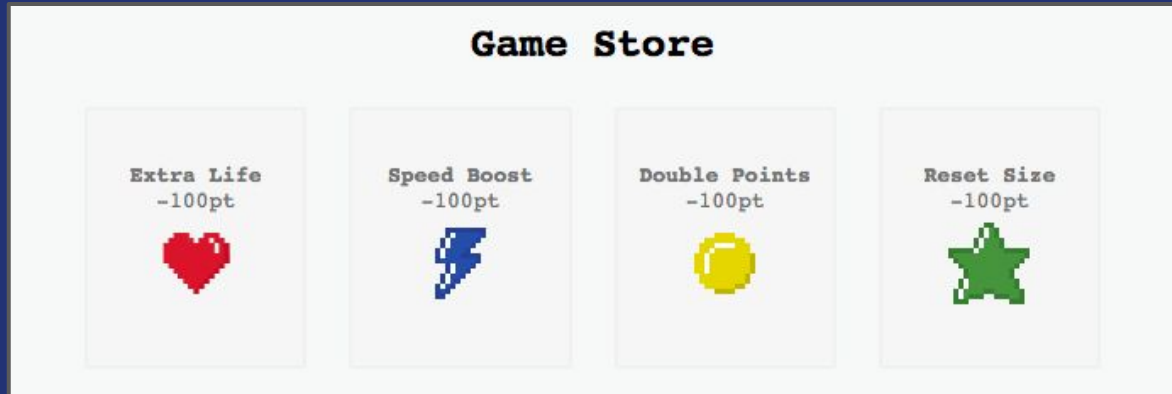
```
$ node server.js
```

```
Listening on http://localhost:5000/
```


Step 5: navigate to the URL below

localhost:5000

Notice something missing?



Set up Shopify for development

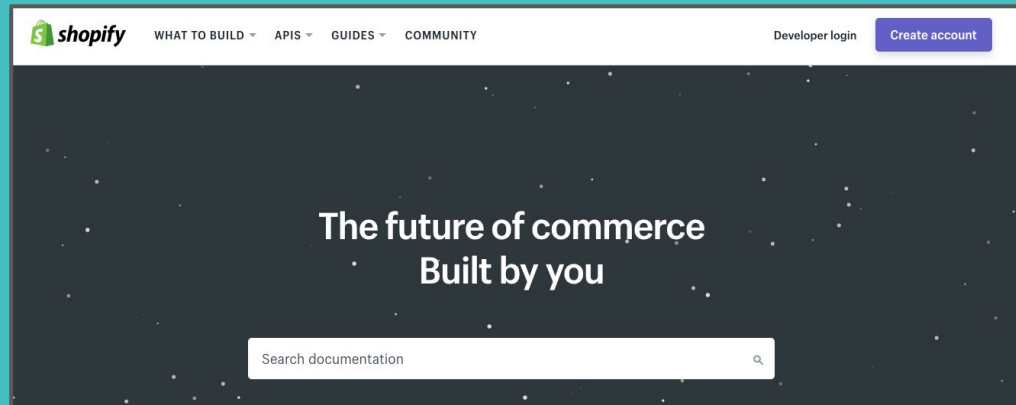
<https://developers.shopify.com>

1. Creating a developer account.
2. Creating a development store on your account.
3. Creating an app on your store.
4. Setting up free payments on your store.

Register for a Shopify developer account

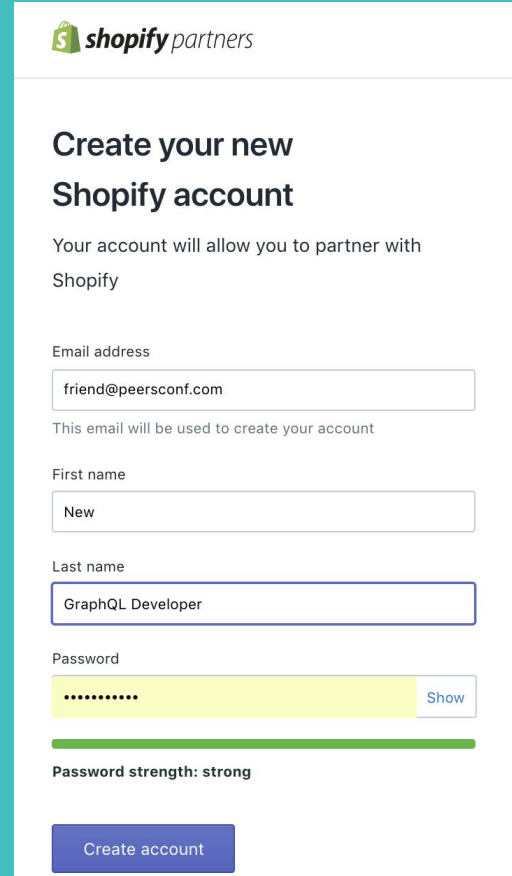
<https://developers.shopify.com>

1. Navigate to the address above.
2. Click "Create account."



Register for a Shopify developer account

3. Enter your name and email address
4. Choose a password
5. Click “Create account”



The screenshot shows the 'Create your new Shopify account' page on the Shopify Partners website. The page has a white background with a teal header bar containing the 'shopify partners' logo. The main heading is 'Create your new Shopify account'. Below this, a sub-heading reads 'Your account will allow you to partner with Shopify'. The form contains several input fields: 'Email address' with the value 'friend@peersconf.com', 'First name' with the value 'New', and 'Last name' with the value 'GraphQL Developer'. The 'Password' field is highlighted in yellow and contains a series of dots, with a 'Show' button to its right. Below the password field is a green progress bar and the text 'Password strength: strong'. At the bottom of the form is a blue 'Create account' button.

shopify partners

Create your new Shopify account

Your account will allow you to partner with Shopify

Email address

This email will be used to create your account

First name

Last name

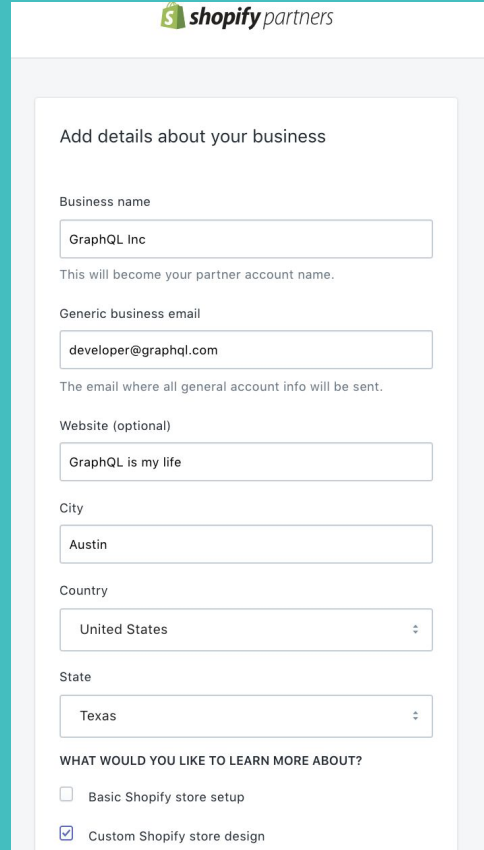
Password

 Show

Create account

Register for a Shopify developer account

6. Fill in the rest of the form and click “See Partner dashboard” at the bottom.



The screenshot shows the 'Add details about your business' section of the Shopify Partners registration form. The form is titled 'shopify partners' at the top. It contains several input fields and a list of checkboxes.

Add details about your business

Business name
GraphQL Inc
This will become your partner account name.

Generic business email
developer@graphql.com
The email where all general account info will be sent.

Website (optional)
GraphQL is my life

City
Austin

Country
United States

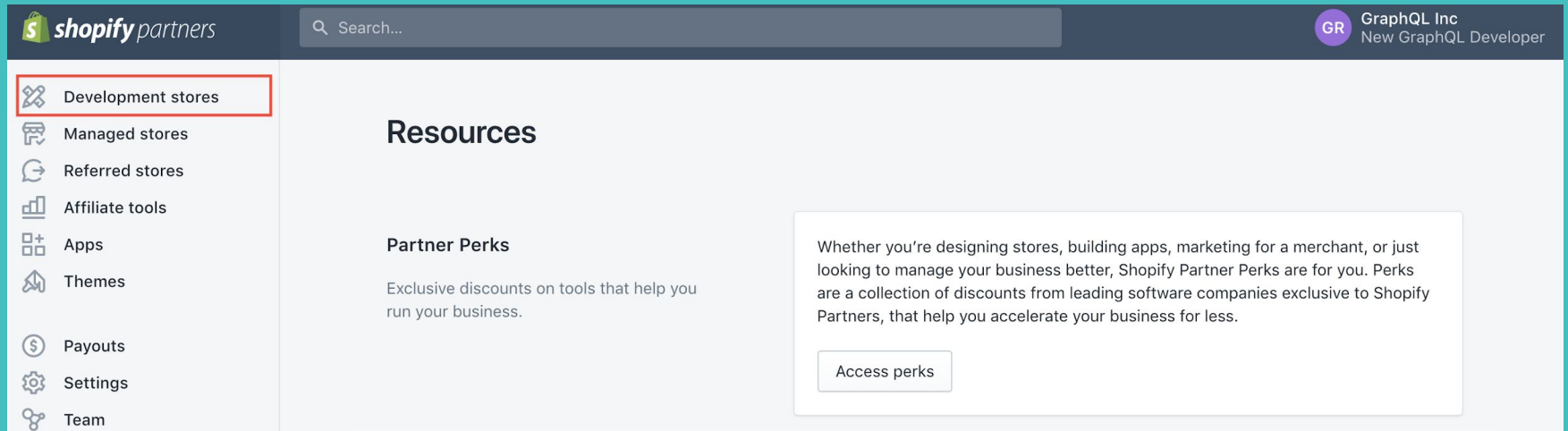
State
Texas

WHAT WOULD YOU LIKE TO LEARN MORE ABOUT?

- Basic Shopify store setup
- Custom Shopify store design

Register for a Shopify developer account

7. Click “Development stores.”



The screenshot shows the Shopify Partners dashboard. The top navigation bar includes the 'shopify partners' logo, a search bar, and a user profile for 'GraphQL Inc' with the role 'New GraphQL Developer'. The left sidebar contains a list of menu items: 'Development stores' (highlighted with a red box), 'Managed stores', 'Referred stores', 'Affiliate tools', 'Apps', 'Themes', 'Payouts', 'Settings', and 'Team'. The main content area is titled 'Resources' and features a 'Partner Perks' section with a description and an 'Access perks' button.

shopify partners Search... GraphQL Inc
New GraphQL Developer

- Development stores**
- Managed stores
- Referred stores
- Affiliate tools
- Apps
- Themes
- Payouts
- Settings
- Team

Resources

Partner Perks

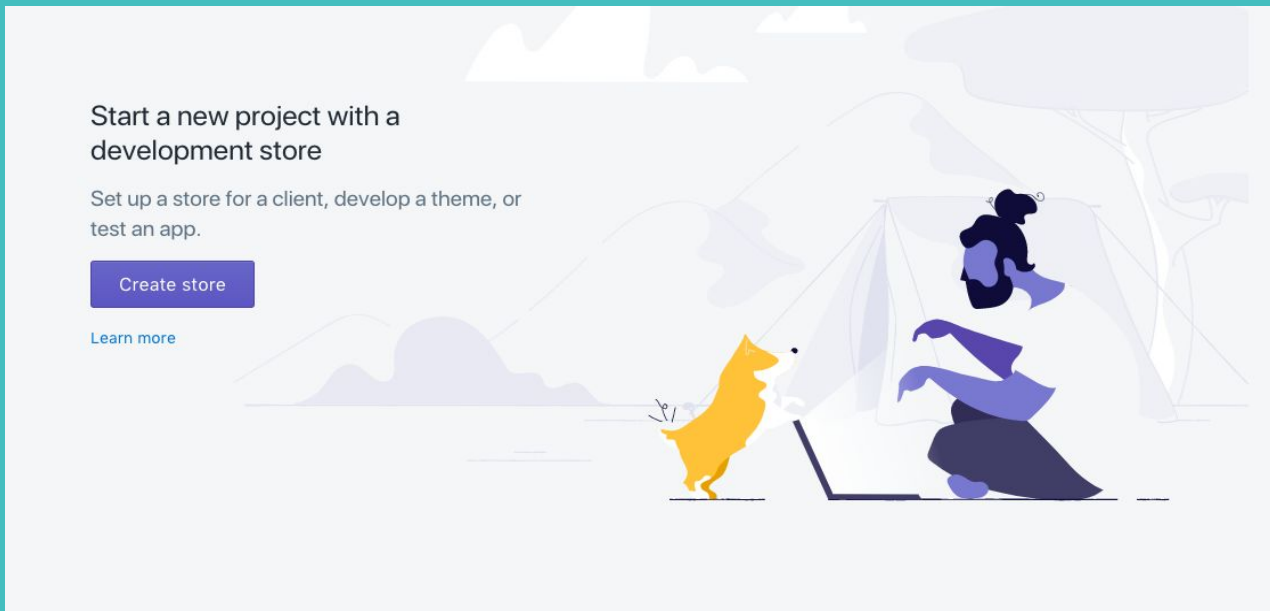
Exclusive discounts on tools that help you run your business.

Whether you're designing stores, building apps, marketing for a merchant, or just looking to manage your business better, Shopify Partner Perks are for you. Perks are a collection of discounts from leading software companies exclusive to Shopify Partners, that help you accelerate your business for less.

[Access perks](#)

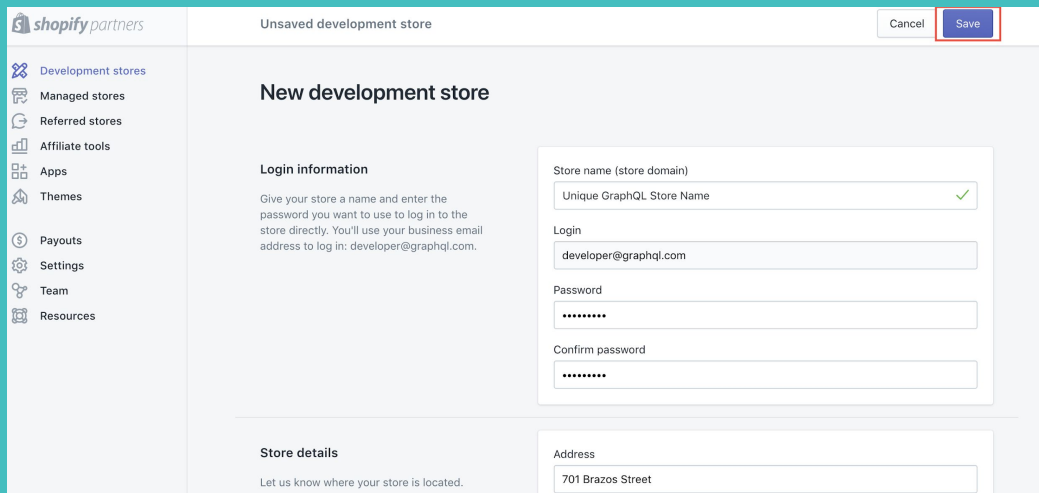
Register for a Shopify developer account

8. Click “Create store”



Register for a Shopify developer account

9. Give your store a unique name, don't use "snake game" or anything similar because the name needs to be unique to your store.
10. Fill in the rest of the form and then click "Save."



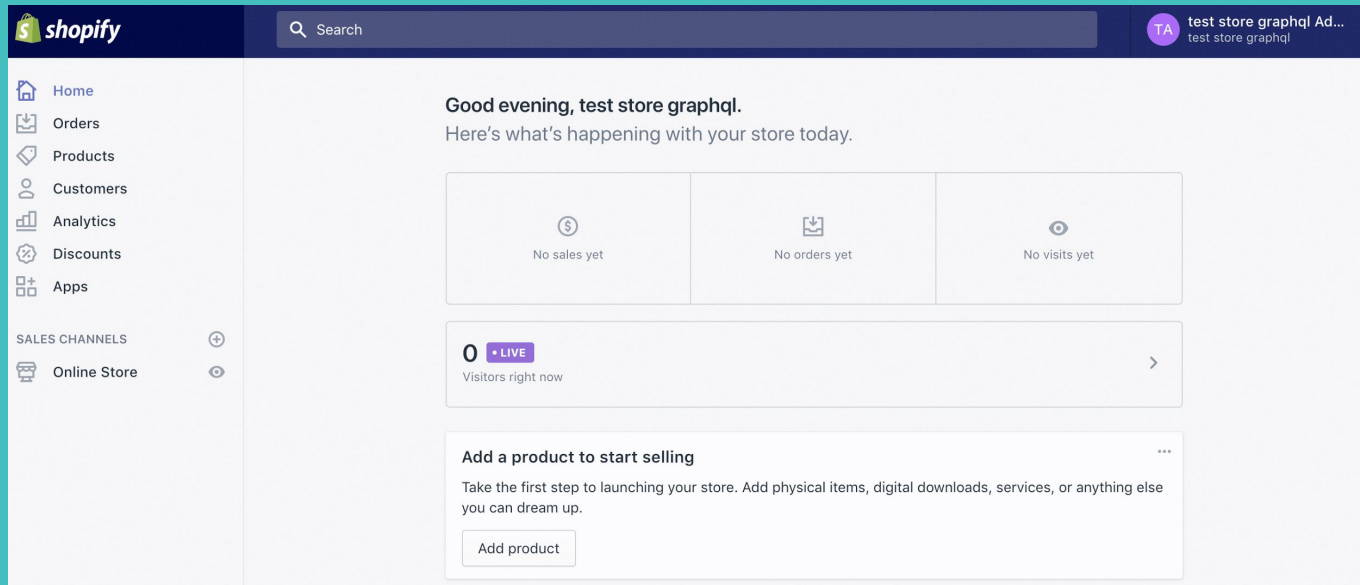
The screenshot shows the 'New development store' form in the Shopify Partners interface. The form is titled 'New development store' and is located under the 'Development stores' section. It contains several input fields and sections:

- Store name (store domain):** A text input field containing 'Unique GraphQL Store Name' with a green checkmark icon to its right.
- Login:** A text input field containing 'developer@graphql.com'.
- Password:** A text input field with masked characters (dots).
- Confirm password:** A text input field with masked characters (dots).
- Store details:** A section with the heading 'Store details' and the text 'Let us know where your store is located.' Below this is a text input field containing '701 Brazos Street'.

At the top right of the form, there are two buttons: 'Cancel' and 'Save'. The 'Save' button is highlighted with a red border.

Get credentials

11. On the left side of the home screen, click "Apps".
12. Click "Manage private apps" on the bottom of the next screen.



The screenshot shows the Shopify admin dashboard. The top navigation bar includes the Shopify logo, a search bar, and the user profile 'test store graphql Ad...'. The left sidebar contains a menu with 'Home', 'Orders', 'Products', 'Customers', 'Analytics', 'Discounts', 'Apps', and 'SALES CHANNELS' (with a plus icon). Under 'SALES CHANNELS', 'Online Store' is listed with an eye icon. The main content area displays a greeting: 'Good evening, test store graphql. Here's what's happening with your store today.' Below this are three summary cards: 'No sales yet' (with a dollar sign icon), 'No orders yet' (with a shopping bag icon), and 'No visits yet' (with an eye icon). A 'LIVE' status indicator shows '0 Visitors right now' with a right arrow. At the bottom, there is a section titled 'Add a product to start selling' with a sub-header 'Take the first step to launching your store. Add physical items, digital downloads, services, or anything else you can dream up.' and an 'Add product' button.

Get credentials

13. Click “Create a new private app.”

Get credentials

14. Name your private app.

15. Enter an email.

16. Click "Allow this app to access your storefront data using the Storefront API."

17. Click "Save."

The screenshot shows the Shopify Admin interface for creating a private app. The top navigation bar includes the Shopify logo, a search bar, and a user profile for 'test store graphQL Ad...'. The left sidebar contains navigation links for Home, Orders, Products, Customers, Analytics, Discounts, Apps, and Online Store. The main content area is titled 'Create private app' and is divided into two sections: 'Description' and 'Admin API'. The 'Description' section includes a text input for 'Private app name' and an optional text input for 'Contact email'. The 'Admin API' section lists permissions for 'Store content like articles, blogs, comments, pages, and redirects' and 'Customer details and customer groups', each with a 'Read access' dropdown menu. A note at the top of the Admin API section states: 'Your API credentials will be generated when you Save.'

Connect your store to application

19. In your preferred text editor, open the project folder you downloaded. In `public/js/queries.js`, paste your storefront access token on Line 2.

```
01 var storefrontAPI = "https://name-of-your-store.myshopify.com/api/graphql";  
02 var storefrontAccessToken = "12345acbde";
```



Connect your store to application

20. On Line 1, change the words "name-of-your-store" to the name you gave your store when you created your account, which can be found in the URL of your Shopify account homepage.

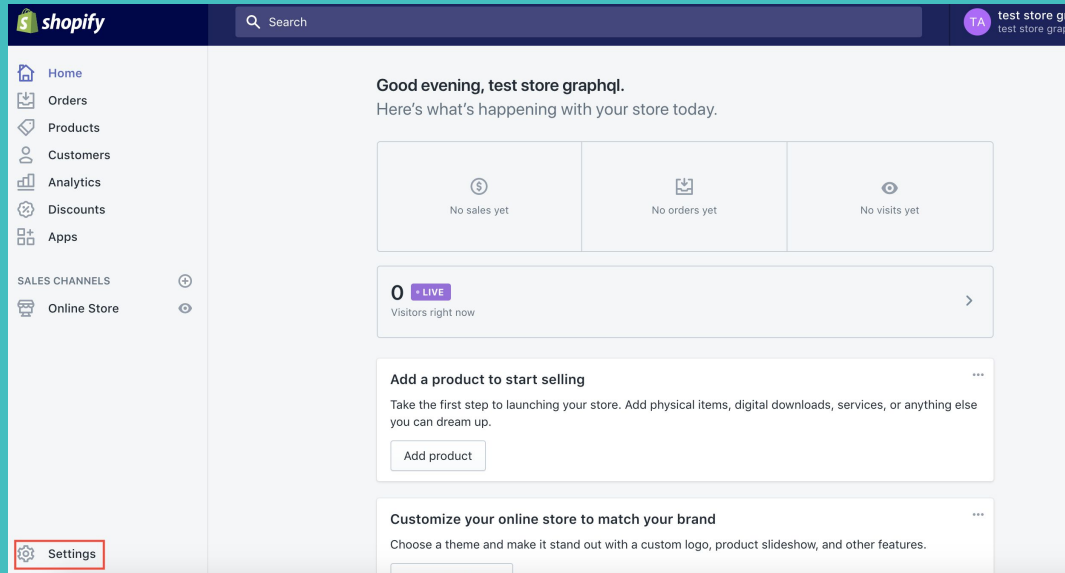
```
01 var storefrontAPI = "https://name-of-your-store.myshopify.com/api/graphql";  
02 var storefrontAccessToken = "12345acbde";
```



Set up payments

Now that you have connected your store to your application, return to your Storefront home page in your browser so that you can set up payments.

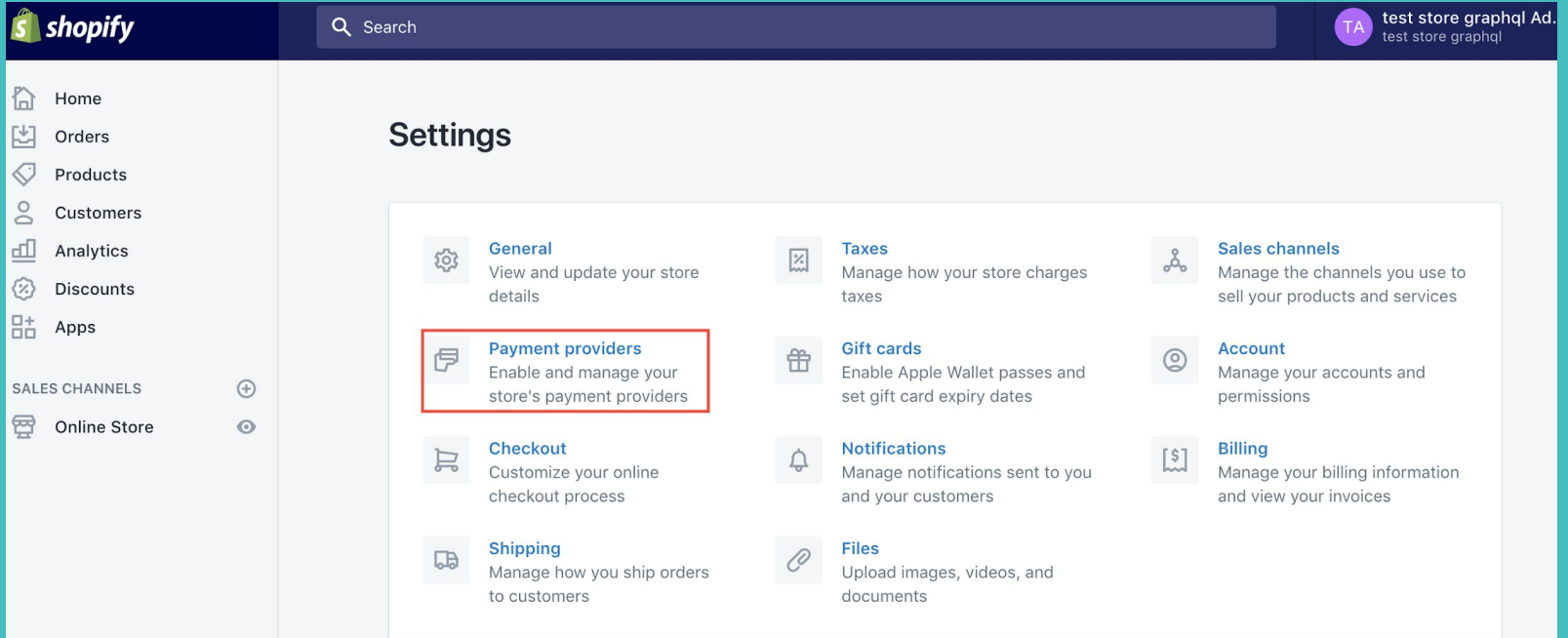
1. Click "Settings" at the bottom of your screen.



The screenshot displays the Shopify admin interface. On the left is a navigation sidebar with icons for Home, Orders, Products, Customers, Analytics, Discounts, Apps, and SALES CHANNELS (Online Store). The main content area shows a greeting "Good evening, test store graphql." followed by three summary cards: "No sales yet", "No orders yet", and "No visits yet". Below these is a "0 LIVE" visitor counter. At the bottom of the sidebar, the "Settings" button is highlighted with a red rectangle.

Set up payments

2. Click “Payment providers”



The screenshot displays the Shopify admin dashboard. The top navigation bar includes the Shopify logo, a search bar, and a user profile icon labeled 'TA' with the text 'test store graphql Ad. test store graphql'. The left sidebar contains a menu with icons and labels for 'Home', 'Orders', 'Products', 'Customers', 'Analytics', 'Discounts', 'Apps', 'SALES CHANNELS', and 'Online Store'. The main content area is titled 'Settings' and features a grid of settings categories. The 'Payment providers' category is highlighted with a red border. The categories and their descriptions are as follows:

- General**: View and update your store details
- Taxes**: Manage how your store charges taxes
- Sales channels**: Manage the channels you use to sell your products and services
- Payment providers**: Enable and manage your store's payment providers
- Gift cards**: Enable Apple Wallet passes and set gift card expiry dates
- Account**: Manage your accounts and permissions
- Checkout**: Customize your online checkout process
- Notifications**: Manage notifications sent to you and your customers
- Billing**: Manage your billing information and view your invoices
- Shipping**: Manage how you ship orders to customers
- Files**: Upload images, videos, and documents

Set up payments

3. Scroll to "Manual payments" and select "Create custom payment method."

Manual payments

Provide customers with instructions to pay outside of your online store. Choose from cash on delivery (COD), money order, bank deposit, or create a custom solution.

Create custom payment method ▾

Set up payments

4. Give your payment method a name and click "Activate."

Manual payments

Provide customers with instructions to pay outside of your online store. Choose from cash on delivery (COD), money order, bank deposit, or create a custom solution.

Create custom payment method ▾

Name of the custom payment method

Additional details

Displayed on the Payment method page, while the customer is choosing how to pay.

Payment instructions

Displayed on the Thank you page, after the customer has placed their order.

Cancel **Activate**

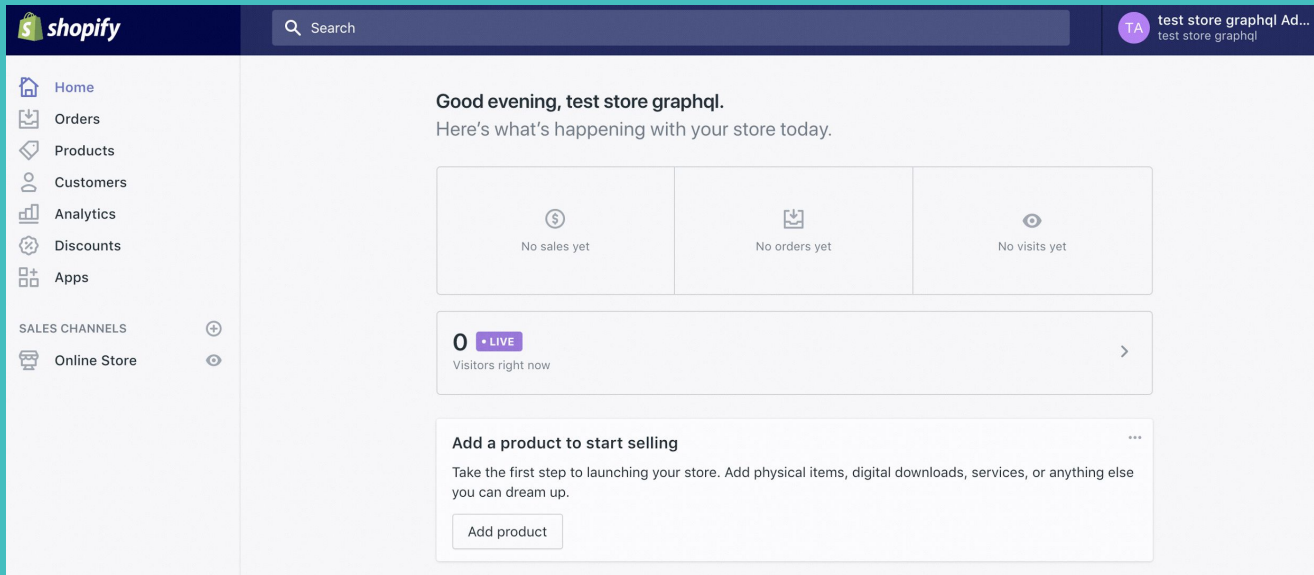
Add power-ups to the storefront

Now that you have created your Storefront, we're going to add the power-ups to the store!

Let us know if you're still setting up! 

Add power-ups to store

1. Return to your account home page on Shopify.
2. On the upper left-hand side of the screen, click "Products."
3. Then, click "Add Product."



The screenshot displays the Shopify admin interface. At the top, there is a dark blue header with the Shopify logo on the left, a search bar in the center, and a user profile on the right labeled 'test store graphql Ad...'. Below the header is a left-hand navigation menu with icons and labels for 'Home', 'Orders', 'Products', 'Customers', 'Analytics', 'Discounts', and 'Apps'. Underneath this menu is a 'SALES CHANNELS' section with a plus icon and 'Online Store' with an eye icon. The main content area features a greeting: 'Good evening, test store graphql. Here's what's happening with your store today.' Below the greeting are three white boxes with icons and text: a dollar sign icon with 'No sales yet', a shopping cart icon with 'No orders yet', and an eye icon with 'No visits yet'. Further down is a 'LIVE' status indicator showing '0' visitors right now. At the bottom, there is a white box titled 'Add a product to start selling' with a three-dot menu icon on the right. The text inside says 'Take the first step to launching your store. Add physical items, digital downloads, services, or anything else you can dream up.' and includes a button labeled 'Add product'.

Add power-ups to store

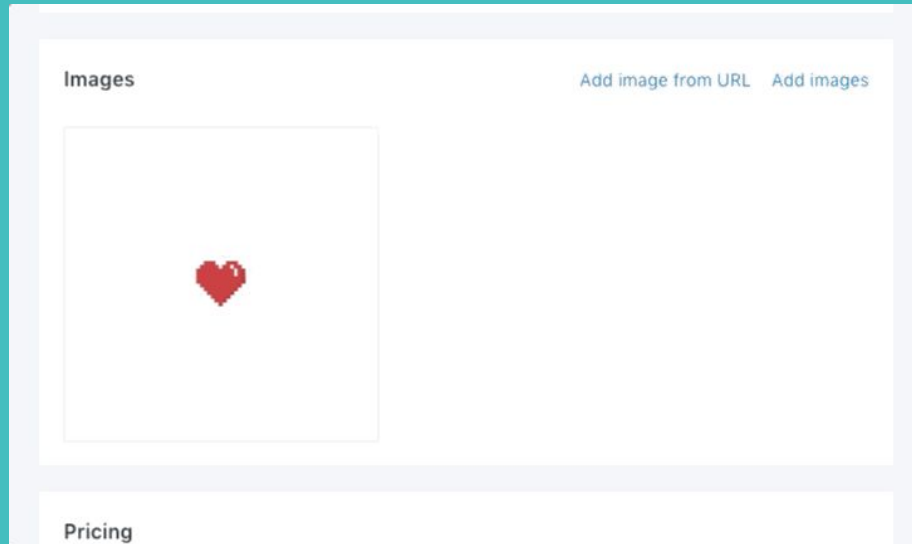
6. Upload public/images/storefront-images/extra-life.png image from the project folder you downloaded.
7. Verify that "Charge taxes on this product" is NOT checked.
8. Verify that "This is a physical product" is NOT checked.

The screenshot shows a product management interface for an 'Unsaved product'. The main content area contains the product name 'Extra Life Power-Up' and an 'Images' section with a placeholder for an image and the text 'Drop files to upload'. The right sidebar contains several sections: 'Organization' with dropdowns for 'Product type' (Shirts) and 'Vendor' (Nike); 'Collections' with a search bar and a note to add the product to a collection; and 'Tags' with a 'View all tags' link and a text input containing 'Vintage, cotton, summer'. At the bottom, a 'Pricing' section is partially visible. The interface includes a rich text editor toolbar at the top and 'Cancel' and 'Save' buttons in the top right corner.

Add power-ups to store

9. At the bottom of the page, click "Edit website SEO."

10. In the URL and handle field, change the name from "extra-life" to "power-up-1." It must be spelled and formatted exactly like this for the game to work.



Add power-ups to store

The game we previewed at the beginning of this workshop had four power-ups. Repeat the process you just completed to add the Speed Boost Power-Up.

Be sure to:

- Upload the "speed-boost.png" image
- Uncheck "Charge taxes on this product" and "This product requires shipping"
- Change the website seo to be "power-up-2"

Table of contents

1. Introduction to APIs and GraphQL
2. Preview the app
3. Set up Shopify storefront
- 4. Write your own GraphQL calls**
5. Review
6. Next steps

What queries do we need?

Now that you have set up your Storefront and connect it to your application, you need to write the GraphQL queries that will retrieve information from your Storefront and purchase items. Three queries are necessary:

- 1. Retrieve the products from your store**
- 2. Create the Checkout**
- 3. Complete the Checkout**

Code review: *queries.js*

In **queries.js**, there is a function called `makeRequest()` that sets up our request with the necessary information.

```
05 function makeRequest(query) {
06   var headers = {
07     "X-Shopify-Storefront-Access-Token": storefrontAccessToken,
08     "Content-Type": "application/json"
09   };
10
11   return $.ajax({
12     url: storefrontAPI
13     type: "POST",
14     data: JSON.stringify({ query: query }),
15     headers: headers
16   });
17 }
```

Write your first GraphQL call

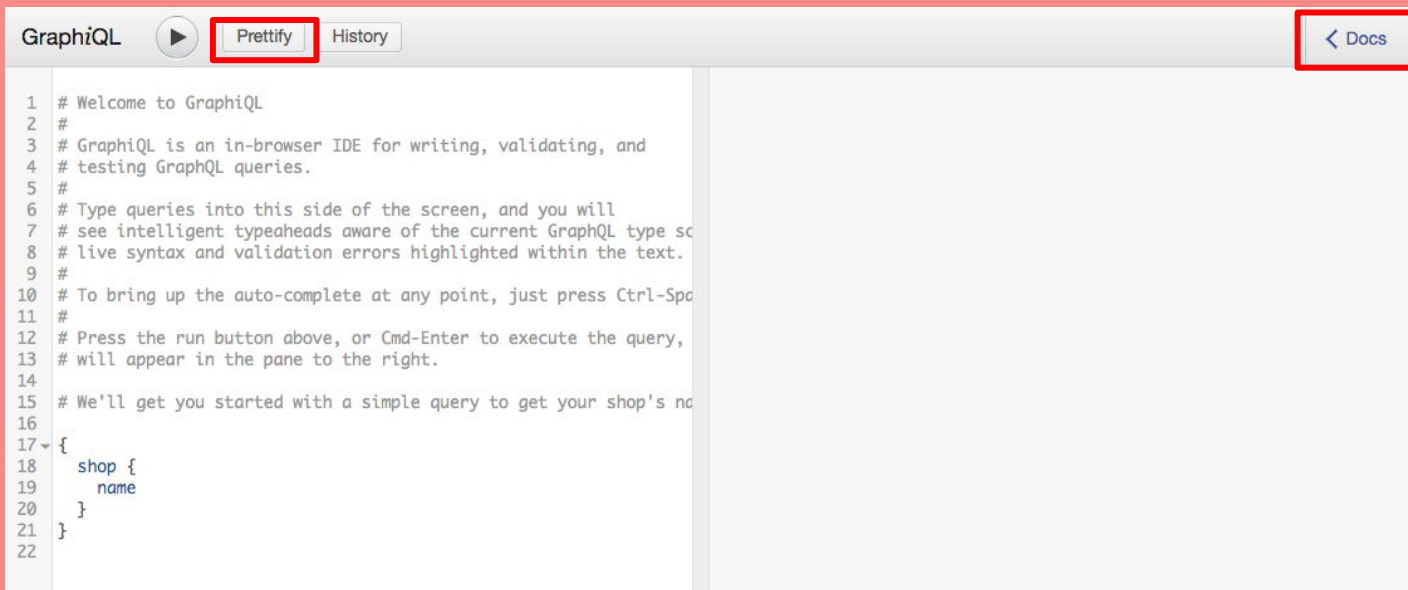
<https://help.shopify.com/en/api/custom-storefronts/storefront-api/graphql-explorer>

We want to retrieve information from our shop. In order to complete our game, we need to retrieve the title of each product and its image.

Navigate to the URL above to access **GraphiQL**, a tool that allows you to test your GraphQL queries.

Query to fetch products

1. Click "Prettify" to remove the notes.
2. Click "Docs" to see the documentation.



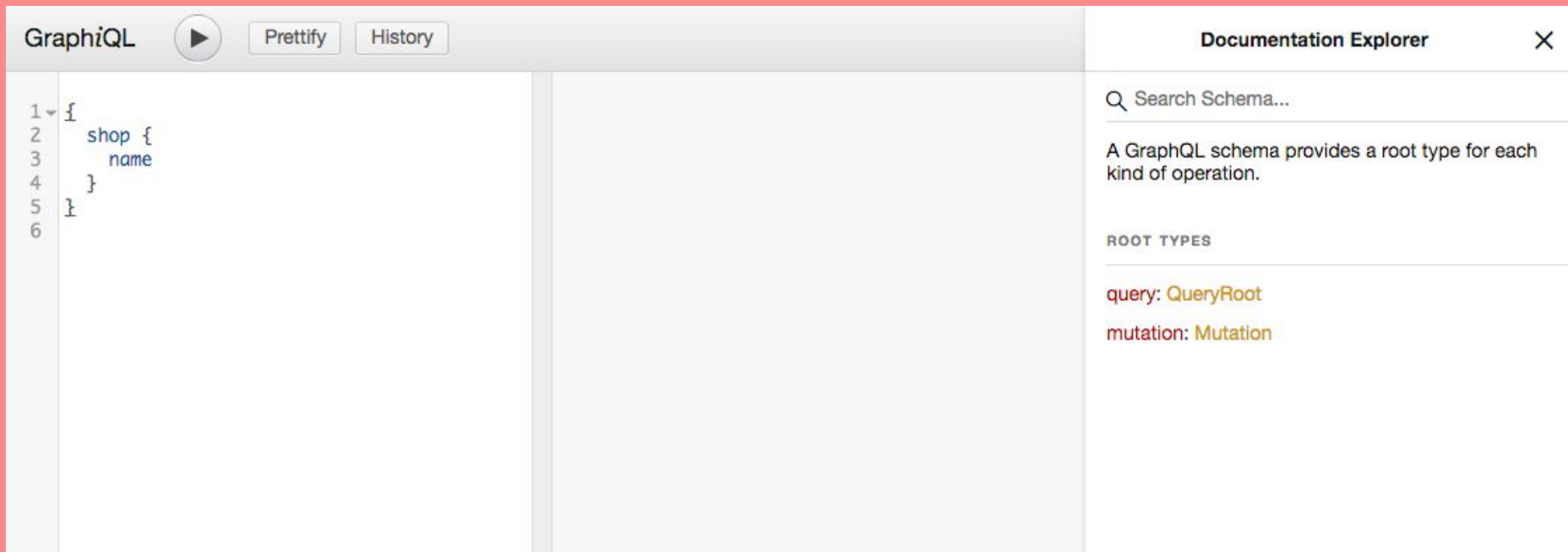
The screenshot shows the GraphQL IDE interface. The top bar contains the text "GraphQL" on the left, a play button icon, a "Prettify" button (highlighted with a red box), and a "History" button. On the right side of the top bar, there is a "< Docs" button (also highlighted with a red box). The main editor area contains the following text:

```
1 # Welcome to GraphQL
2 #
3 # GraphQL is an in-browser IDE for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will
7 # see intelligent typeaheads aware of the current GraphQL type sc
8 # live syntax and validation errors highlighted within the text.
9 #
10 # To bring up the auto-complete at any point, just press Ctrl-Spc
11 #
12 # Press the run button above, or Cmd-Enter to execute the query,
13 # will appear in the pane to the right.
14 #
15 # We'll get you started with a simple query to get your shop's na
16
17 {
18   shop {
19     name
20   }
21 }
22
```

Query to fetch products

3. Click "QueryRoot" to see the entry points for the Shopify Storefront API schema.

Key term: QueryRoot: a "Query Root" is a GraphQL schema's entry-point for queries.



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor:

```
1 {  
2   shop {  
3     name  
4   }  
5 }  
6
```

On the right, the Documentation Explorer is open, displaying the following content:

Documentation Explorer

Q Search Schema...

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

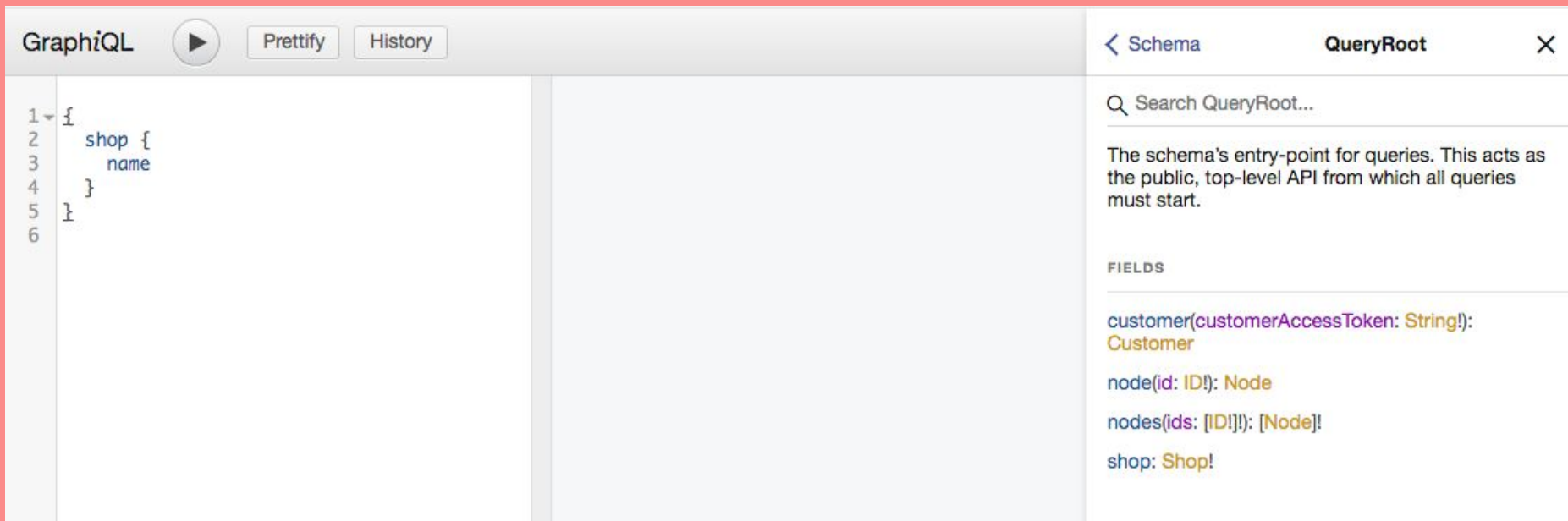
ROOT TYPES

query: **QueryRoot**

mutation: **Mutation**

Query to fetch products

4. We want to retrieve products from our shop. Click "Shop!"



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor:

```
1 {  
2   shop {  
3     name  
4   }  
5 }  
6
```

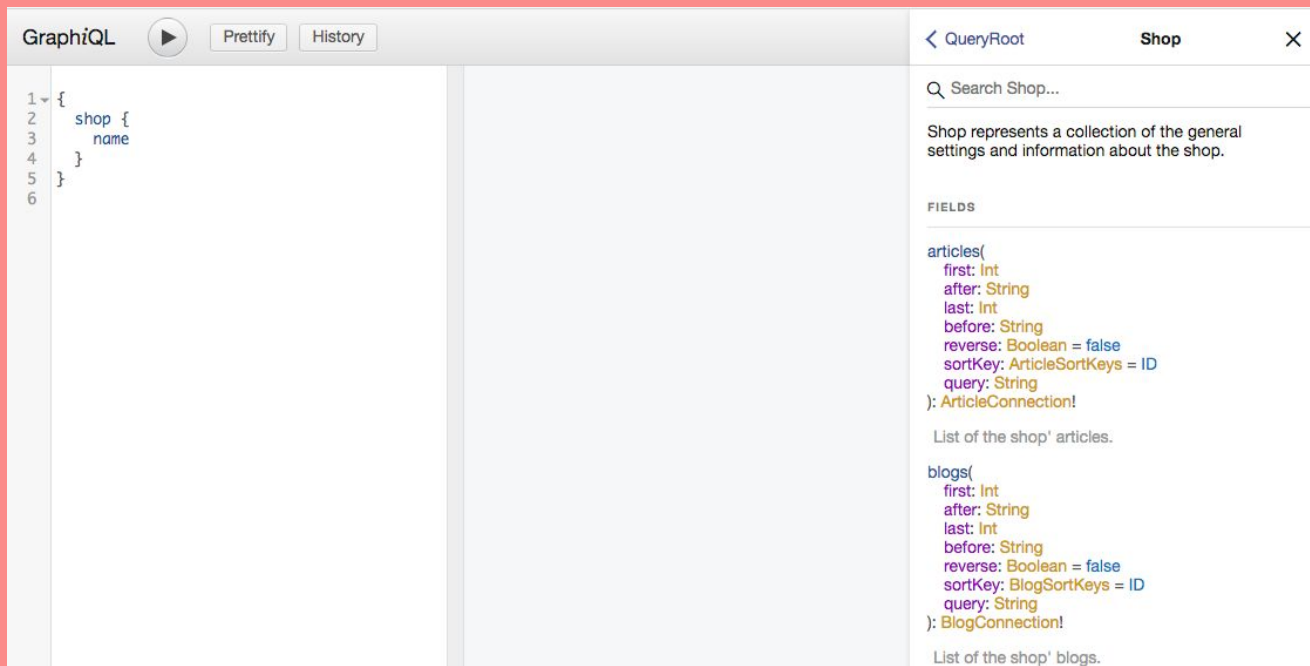
On the right, the 'QueryRoot' schema is displayed. It includes a search bar and a description: 'The schema's entry-point for queries. This acts as the public, top-level API from which all queries must start.' Below this, a list of fields is shown:

- `customer(customerAccessToken: String!): Customer`
- `node(id: ID!): Node`
- `nodes(ids: [ID!]!): [Node]!`
- `shop: Shop!`

Query to fetch products

5. Scroll through the documentation until you find "products."

6. Click "products."



The screenshot shows the GraphQL IDE interface. On the left, the query editor contains the following query:

```
1 {  
2   shop {  
3     name  
4   }  
5 }  
6
```

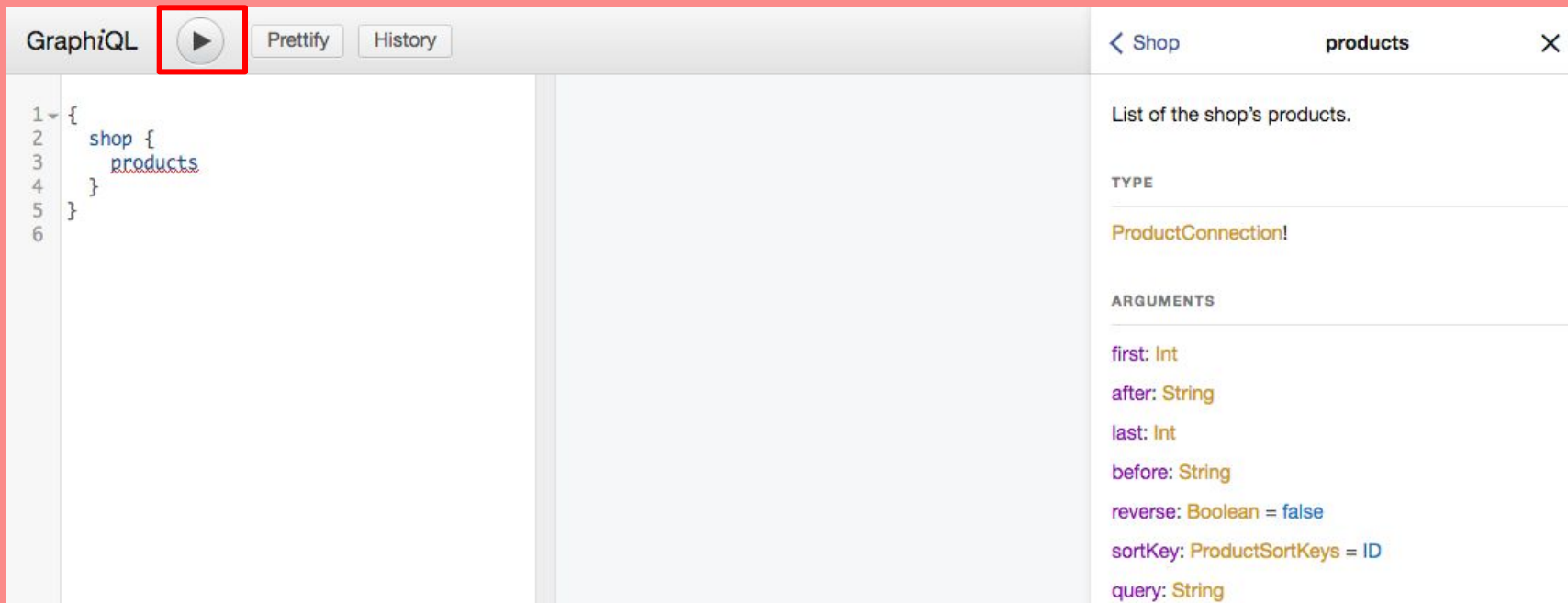
On the right, the results pane shows the schema for the `Shop` type. The schema includes the following fields:

- `articles`:
 - `first`: `int`
 - `after`: `String`
 - `last`: `int`
 - `before`: `String`
 - `reverse`: `Boolean = false`
 - `sortKey`: `ArticleSortKeys = ID`
 - `query`: `String`
- `ArticleConnection!`:
 - List of the shop' articles.
- `blogs`:
 - `first`: `int`
 - `after`: `String`
 - `last`: `int`
 - `before`: `String`
 - `reverse`: `Boolean = false`
 - `sortKey`: `BlogSortKeys = ID`
 - `query`: `String`
- `BlogConnection!`:
 - List of the shop' blogs.

Query to fetch products

7. In the left section of the GraphQL explorer, replace "name" with "products".

8. Click the Play symbol.



The screenshot shows the GraphQL Explorer interface. On the left, the query editor contains the following query:

```
1 {  
2   shop {  
3     products  
4   }  
5 }  
6
```

The query is highlighted with a red box around the play button. The right side of the interface shows the result of the query, which is a list of the shop's products. The result is displayed in a table format with columns for TYPE and ARGUMENTS.

TYPE
ProductConnection!

ARGUMENTS
first: Int
after: String
last: Int
before: String
reverse: Boolean = false
sortKey: ProductSortKeys = ID
query: String

Query to fetch products

Two things happened:

1. Several more required fields were added.
2. We received an error message: "you must provide one of first or last."

The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor with line numbers 1-12. The query is:

```
1 {
2   shop {
3     products {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12 }
```

In the center, the JSON response is displayed, showing an error message: "you must provide one of first or last". The response structure is:

```
{
  "data": null,
  "errors": [
    {
      "message": "you must provide one of first or last",
      "locations": [
        {
          "line": 3,
          "column": 5
        }
      ],
      "path": [
        "shop",
        "products"
      ]
    }
  ]
}
```

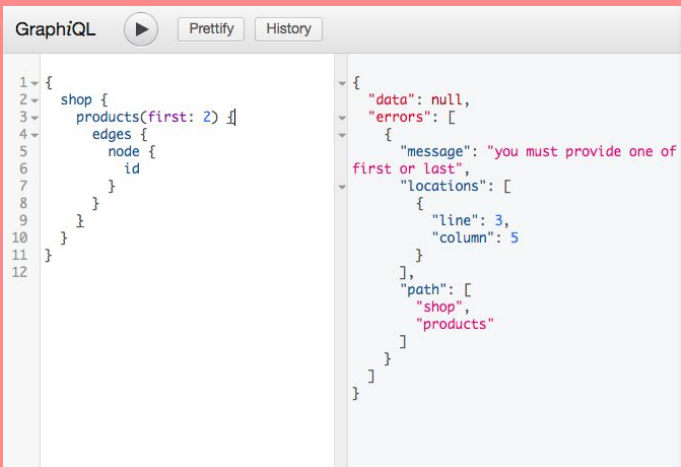
On the right, a sidebar titled "Shop" and "products" is visible. It contains the text "List of the shop's products." and a "TYPE" section showing "ProductConnection!". Below that, an "ARGUMENTS" section lists various options: "first: Int", "after: String", "last: Int", "before: String", "reverse: Boolean = false", and "sortBy: ProductSortKeys = ID".

Query to fetch products

`first` and `last` are arguments that can be passed to "products." We want to retrieve the first 2 products we added to our storefront.

9. Add (`first: 2`) to the end of the word "products."

10. Click the Play symbol.



The screenshot shows the GraphQL IDE interface. On the left, the query editor contains the following code:

```
1 {
2   shop {
3     products(first: 2) {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12 }
```

On the right, the response editor shows an error message:

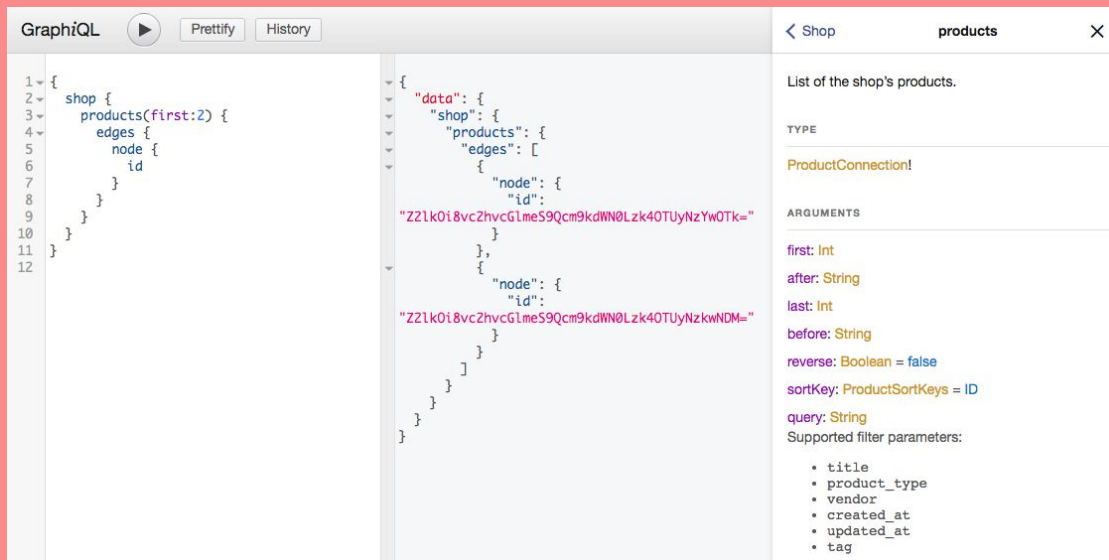
```
{
  "data": null,
  "errors": [
    {
      "message": "you must provide one of
      first or last",
      "locations": [
        {
          "line": 3,
          "column": 5
        }
      ],
      "path": [
        "shop",
        "products"
      ]
    }
  ]
}
```

Key term: Argument: Information passed to a function that is used by the function to produce the desired result.

Query to fetch products

Now, a response has been returned!

11. Click "Product Connection!" to see what other information can be returned.



The screenshot displays the GraphQL Playground interface. On the left, the query is defined as follows:

```
1 {
2   shop {
3     products(first:2) {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12 }
```

The right pane shows the JSON response:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "id": "Z21k0i8vcZhvcG1meS9Qcm9kdWN0Lzk4OTUyNzYwOTk="
            }
          },
          {
            "node": {
              "id": "Z21k0i8vcZhvcG1meS9Qcm9kdWN0Lzk4OTUyNzkwNDM="
            }
          }
        ]
      }
    }
  }
}
```

The right pane also shows a detailed view of the `ProductConnection!` type, including its arguments and supported filter parameters:

ARGUMENTS

- `first`: `int`
- `after`: `String`
- `last`: `int`
- `before`: `String`
- `reverse`: `Boolean = false`
- `sortBy`: `ProductSortKeys = ID`
- `query`: `String`

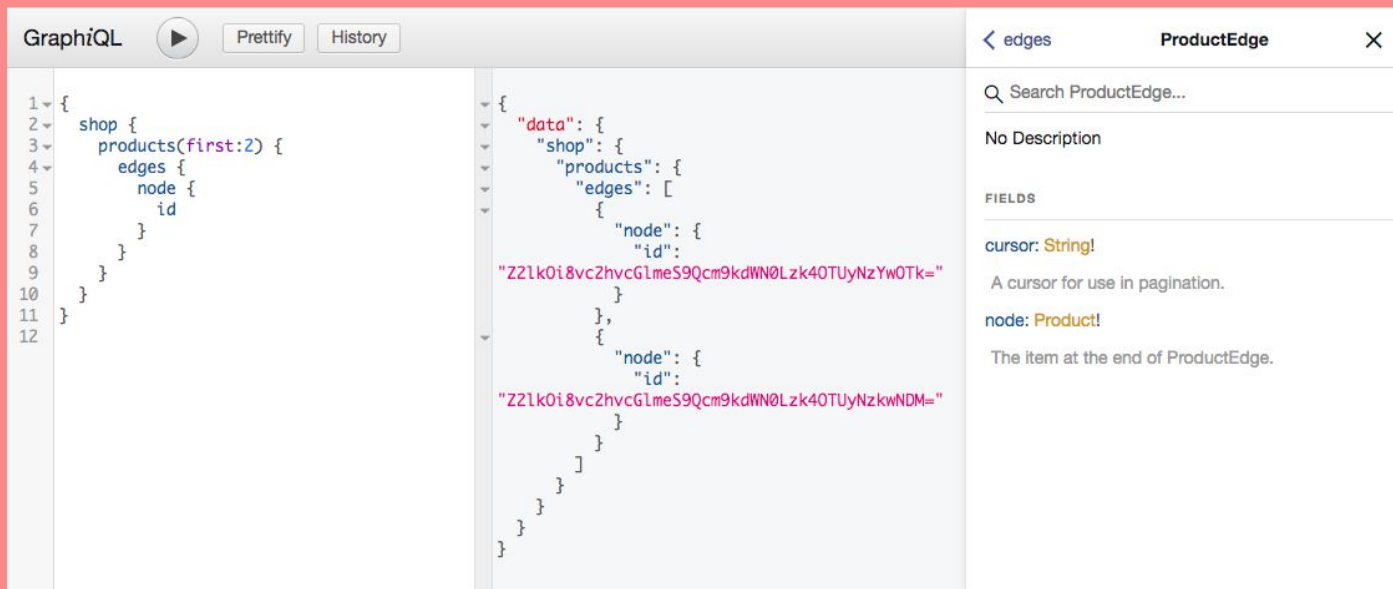
Supported filter parameters:

- `title`
- `product_type`
- `vendor`
- `created_at`
- `updated_at`
- `tag`

Query to fetch products

In the query, inside of "products," the "edges" field is added.

12. Click "ProductEdge!" to see what fields "edges" accepts.



The screenshot shows the GraphQL IDE interface. On the left, the query editor contains the following code:

```
1 {
2   shop {
3     products(first:2) {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12 }
```

On the right, the JSON response is displayed:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzYwOTk="
            }
          },
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzkwNDM="
            }
          }
        ]
      }
    }
  }
}
```

On the far right, the "ProductEdge" type definition is shown:

edges ProductEdge

Search ProductEdge...

No Description

FIELDS

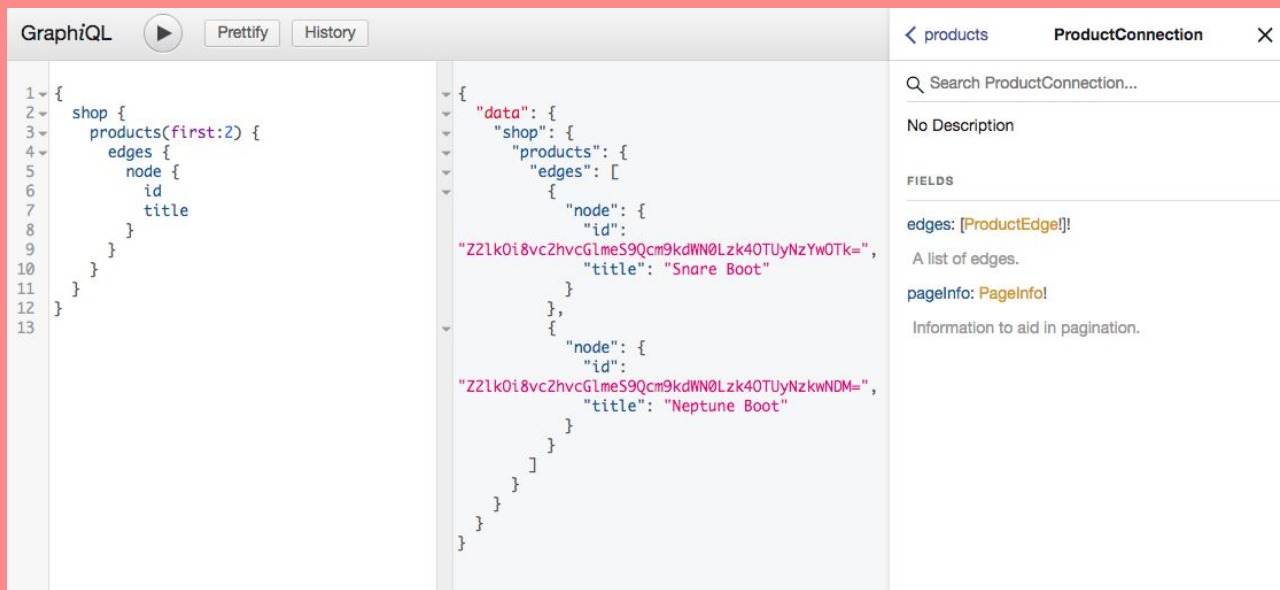
cursor: String!
A cursor for use in pagination.

node: Product!
The item at the end of ProductEdge.

Query to fetch products

Inside of the "edges" field, there is the "node" field.

13. Click "Product !" to see what fields "node" accepts.



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor with line numbers 1 through 13. The query is:

```
1 {
2   shop {
3     products(first:2) {
4       edges {
5         node {
6           id
7           title
8         }
9       }
10    }
11  }
12 }
13 }
```

On the right, the JSON response is displayed, showing the structure of the data returned by the query. The response is:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "id":
              "Z21k0i8vc2hvcGlmeS9Qcm9kdWN0Lzk4OTUyNzYwOTk=",
              "title": "Snare Boot"
            }
          },
          {
            "node": {
              "id":
              "Z21k0i8vc2hvcGlmeS9Qcm9kdWN0Lzk4OTUyNzkwNDM=",
              "title": "Neptune Boot"
            }
          }
        ]
      }
    }
  }
}
```

On the far right, the GraphQL Playground interface shows the query results for the "products" field. It includes a search bar, a description "No Description", and a list of fields: "edges: [ProductEdge!]" and "pageInfo: PageInfo!".

Query to fetch products

Question: What are the two pieces of information that we need to retrieve to include in our game?

The screenshot displays the GraphQL IDE interface. On the left, a query is written in a code editor:

```
1 {
2   shop {
3     products(first:2) {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12 }
```

In the center, the JSON response is shown, with the two product nodes expanded to show their IDs:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzYwOTk="
            }
          },
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzkwNDM="
            }
          }
        ]
      }
    }
  }
}
```

On the right, the **Product** type is defined:

Product

Search Product...

A product represents an individual item for sale in a Shopify store. Products are often physical, but they don't have to be. For example, a digital download (such as a movie, music or ebook file) also qualifies as a product, as do services (such as equipment rental, work for hire, customization of another product or an extended warranty).

IMPLEMENTS

- Node

FIELDS

collections(
 first: Int
 after: String
 last: Int
 before: String
 reverse: Boolean = false
): CollectionConnection!

List of collections a product belongs to.

Query to fetch products

Answer: images and title

The screenshot displays a GraphQL IDE interface with three main sections:

- Query Editor:** Shows a GraphQL query to fetch the first two products from a shop. The query is:

```
1 {
2   shop {
3     products(first:2) {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12
```
- Response Viewer:** Shows the JSON response for the query. The response is:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzYwOTk="
            }
          },
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzkwNDM="
            }
          }
        ]
      }
    }
  }
}
```
- Product Detail Panel:** Shows the details for a selected product. It includes a search bar, a description: "A product represents an individual item for sale in a Shopify store. Products are often physical, but they don't have to be. For example, a digital download (such as a movie, music or ebook file) also qualifies as a product, as do services (such as equipment rental, work for hire, customization of another product or an extended warranty).", and a list of collections it belongs to:

```
collections(
  first: Int
  after: String
  last: Int
  before: String
  reverse: Boolean = false
): CollectionConnection!
```

Query to fetch products

14. In the query, delete "id" because we don't need it.

15. Replace it with "title."

16. Click the Play symbol.

The screenshot shows the GraphQL IDE interface. On the left, the query editor contains the following code:

```
1 {
2   shop {
3     products(first:2) {
4       edges {
5         node {
6           id
7         }
8       }
9     }
10  }
11 }
12
```

In the center, the JSON response is displayed:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzYwOTk="
            }
          },
          {
            "node": {
              "id": "Z21k0i8vc2hvcG1meS9Qcm9kdWN0Lzk4OTUyNzkwNDM="
            }
          }
        ]
      }
    }
  }
}
```

On the right, the type definition for `Product` is shown:

```
Product
  search: String!
  title: String!
  description: String!
  price: Float!
  image: String!
  type: String!
  tags: String!
  collections: CollectionConnection!

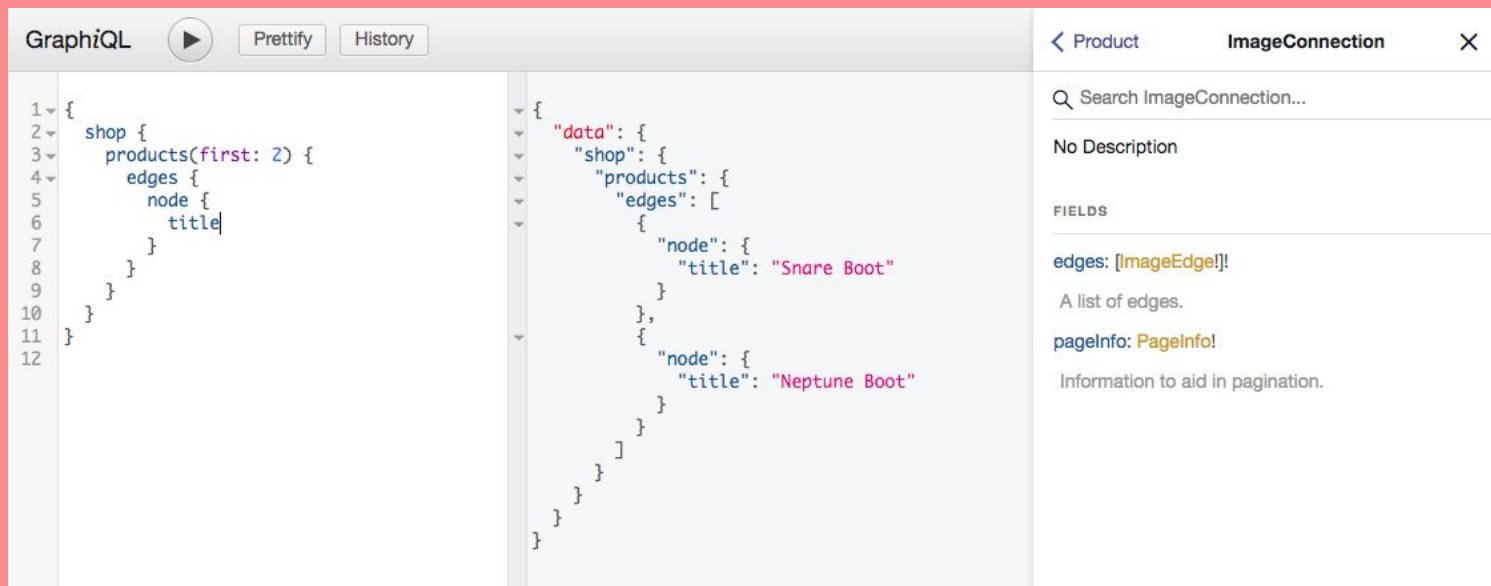
collections(
  first: Int!
  after: String
  last: Int!
  before: String
  reverse: Boolean = false
): CollectionConnection!
```

Below the type definition, it says: "List of collections a product belongs to."

Query to fetch products

17. Below `title`, add the `"images"` field.

18. Click the Play symbol.



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor:

```
1 {
2   shop {
3     products(first: 2) {
4       edges {
5         node {
6           title
7         }
8       }
9     }
10  }
11 }
12
```

On the right, the JSON response is displayed:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "title": "Snare Boot"
            }
          },
          {
            "node": {
              "title": "Neptune Boot"
            }
          }
        ]
      }
    }
  }
}
```

The right panel also shows a breadcrumb navigation: < Product ImageConnection. Below this is a search bar with the text "Search ImageConnection...". Underneath, there is a "No Description" section. A "FIELDS" section lists "edges: [ImageEdge]!" and "pageInfo: PageInfo!".

Query to fetch products

We get the same error message: "you must provide one of first or last" and the location for the error is line 7. The "images" field is on line 7.

19. Add (`first: 1`) to "images." Click Play.

The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor with line numbers 1 through 19. The query is:

```
1 {
2   shop {
3     products(first: 2) {
4       edges {
5         node {
6           title
7           images {
8             edges {
9               node {
10                id
11              }
12            }
13          }
14        }
15      }
16    }
17  }
18 }
19 }
```

In the center, the execution result is displayed as a JSON object:

```
{
  "data": null,
  "errors": [
    {
      "message": "you must provide one of first
or last",
      "locations": [
        {
          "line": 7,
          "column": 11
        }
      ],
      "path": [
        "shop",
        "products",
        "edges",
        0,
        "node",
        "images"
      ]
    }
  ]
}
```

On the right, a sidebar shows the selected field "ImageConnection" with a search bar and a "FIELDS" section containing:

- `edges: [ImageEdge]!` - A list of edges.
- `pageInfo: PageInfo!` - Information to aid in pagination.

Query to fetch products

That worked. However, the "id" field isn't very useful when trying to put the image on our webpage. Let's find out what other fields we can request.

20. Click "ImageConnection!" then "ImageEdge!" then "Image."



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor with line numbers 1 through 19. The query is:

```
1 {
2   shop {
3     products(first: 2) {
4       edges {
5         node {
6           title
7           images(first: 1) {
8             edges {
9               node {
10                id
11              }
12            }
13          }
14        }
15      }
16    }
17  }
18 }
19 }
```

On the right, the JSON response is displayed. The response is a nested object with the following structure:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "title": "Snare Boot",
              "images": {
                "edges": [
                  {
                    "node": {
                      "id":
                        "Z21k0i8vc2hvcGhmeS9Qcm9kdWN0SW1hZ2UvMjEyNzcxNTY5OTU="
                    }
                  }
                ]
              }
            }
          }
        ]
      }
    }
  }
}
```

On the far right, a sidebar shows the selected field "ImageConnection" and its description: "No Description". Below this, the "FIELDS" section lists "edges: [ImageEdge!]" with the description "A list of edges." and "pageInfo: PageInfo!" with the description "Information to aid in pagination."

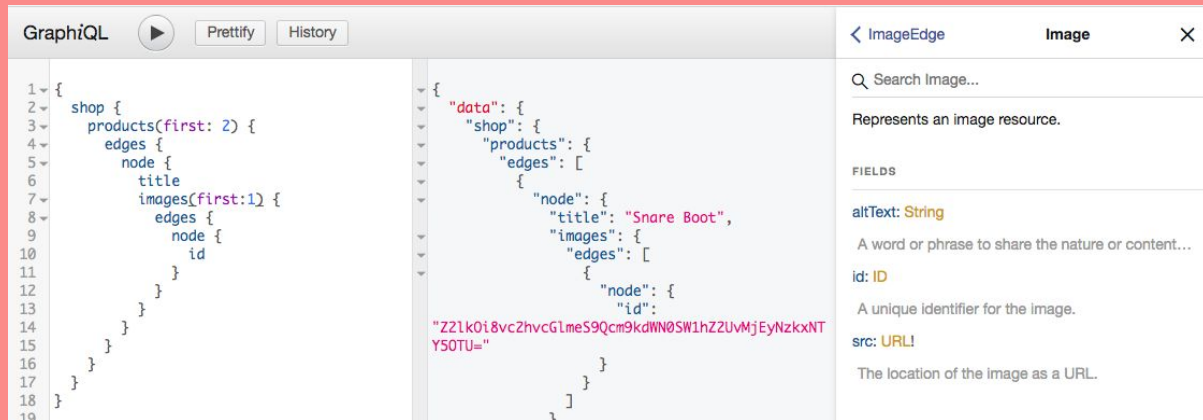
Query to fetch products

Instead of `id`, we can use `originalSrc`, which will return a URL.

That will be more helpful for adding the image to the game

21. Delete `id` and replace it with `originalSrc`.

22. Click the Play symbol.



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor:

```
1 {
2   shop {
3     products(first: 2) {
4       edges {
5         node {
6           title
7           images(first: 1) {
8             edges {
9               node {
10                id
11              }
12            }
13          }
14        }
15      }
16    }
17  }
18 }
19 }
```

In the center, the JSON response is displayed:

```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "title": "Snare Boot",
              "images": {
                "edges": [
                  {
                    "node": {
                      "id":
"Z21k0i8vc2hvcG1meS9Qcm9kdWN0SW1hZ2UvMjEyNzIxNTY5S0U="
                    }
                  }
                ]
              }
            }
          }
        ]
      }
    }
  }
}
```

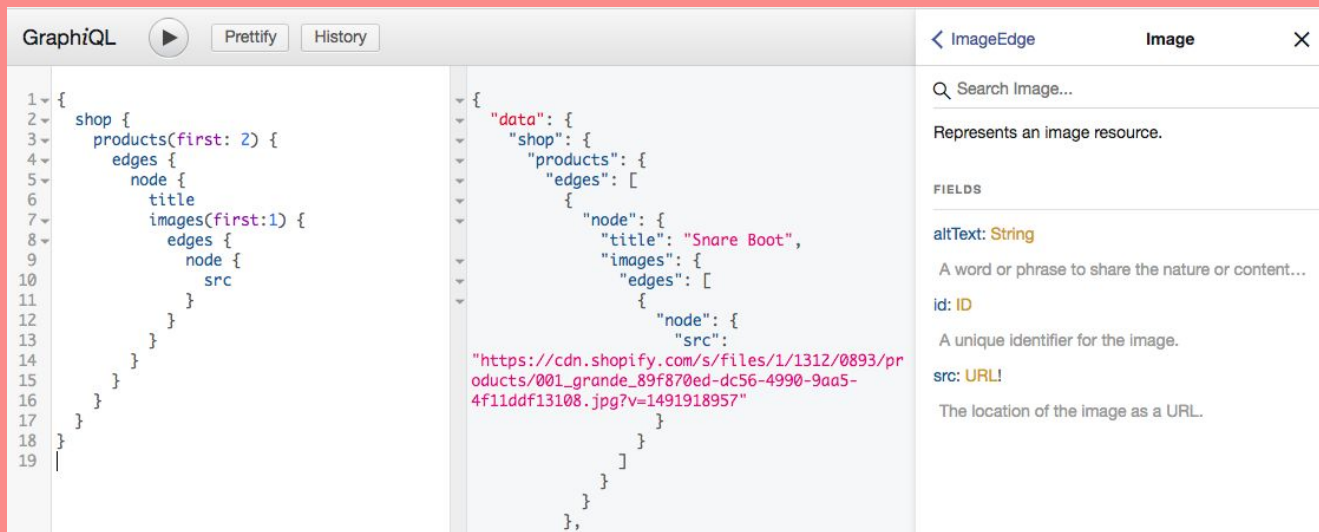
On the right, the schema definition for `ImageEdge` is shown:

```
< ImageEdge Image X
Q Search Image...
Represents an image resource.
FIELDS
altText: String
A word or phrase to share the nature or content...
id: ID
A unique identifier for the image.
src: URL!
The location of the image as a URL.
```

Query to fetch products

In order to create a Checkout instance, we also need the product variant.

23. Under the 3rd closing curly brace below `src`, add `"variants (first: 1)"` and click the Play symbol.



The screenshot shows the GraphQL IDE interface. On the left, a query is written in a code editor with line numbers 1 through 19. The query is:

```
1 {
2   shop {
3     products(first: 2) {
4       edges {
5         node {
6           title
7           images(first: 1) {
8             edges {
9               node {
10                src
11              }
12            }
13          }
14        }
15      }
16    }
17  }
18 }
19 |
```

In the center, the JSON response is displayed, showing the structure of the data returned by the query. The response is:

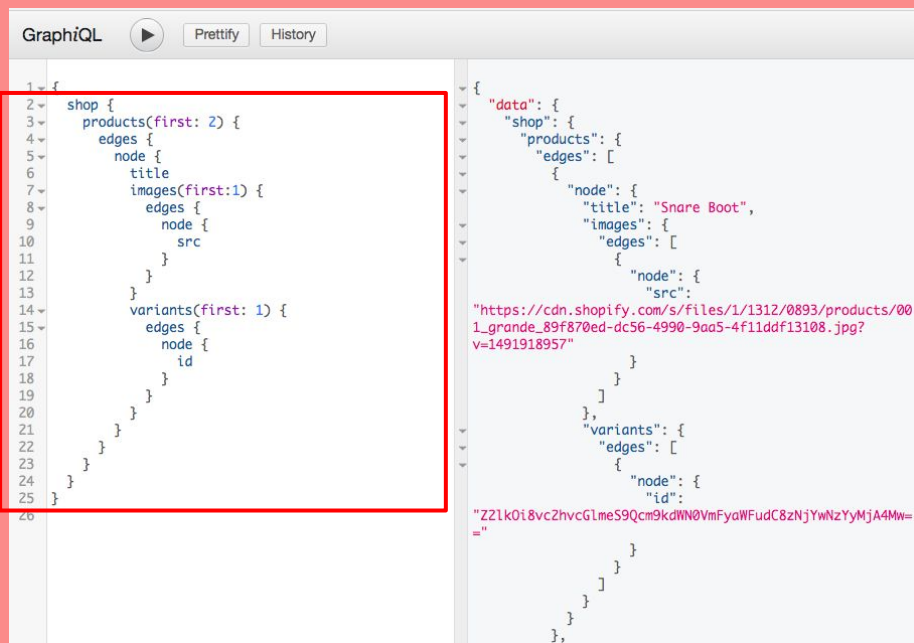
```
{
  "data": {
    "shop": {
      "products": {
        "edges": [
          {
            "node": {
              "title": "Snare Boot",
              "images": {
                "edges": [
                  {
                    "node": {
                      "src":
                        "https://cdn.shopify.com/s/files/1/1312/0893/products/001_grande_89f870ed-dc56-4990-9aa5-4f11ddf13108.jpg?v=1491918957"
                    }
                  }
                ]
              }
            }
          }
        ]
      }
    }
  }
}
```

On the right, a sidebar shows the definition for the `ImageEdge` type. It includes a search bar, a description "Represents an image resource.", and a list of fields:

- `altText`: String - A word or phrase to share the nature or content...
- `id`: ID - A unique identifier for the image.
- `src`: URL! - The location of the image as a URL.

Query to fetch products

Now you have everything you need to write your query! Copy and paste the code from Line 2 to Line 24 of the GraphQL explorer into **queries.js**.



The screenshot shows the GraphQL Explorer interface. On the left, a query is written in a code editor, with lines 2 through 24 highlighted by a red box. The query is:

```
1 f
2 shop {
3   products(first: 2) {
4     edges {
5       node {
6         title
7         images(first: 1) {
8           edges {
9             node {
10              src
11            }
12          }
13        }
14        variants(first: 1) {
15          edges {
16            node {
17              id
18            }
19          }
20        }
21      }
22    }
23  }
24 }
25 }
```

On the right, the JSON response is displayed. The response is a nested object with the following structure:

```
{
  "data": {
    "shop": {
      "products": [
        {
          "edges": [
            {
              "node": {
                "title": "Snare Boot",
                "images": {
                  "edges": [
                    {
                      "node": {
                        "src": "https://cdn.shopify.com/s/files/1/1312/0893/products/001_grande_89f870ed-dc56-4990-9aa5-4f11ddf13108.jpg?v=1491918957"
                      }
                    }
                  ]
                },
                "variants": {
                  "edges": [
                    {
                      "node": {
                        "id": "ZZ1k0i8vc2hvcG1meS9Qcm9kdWN0VmFyaWFudC8zNjYwNzYyMjA4Mw=="
                      }
                    }
                  ]
                }
              }
            }
          ]
        }
      ]
    }
  }
}
```

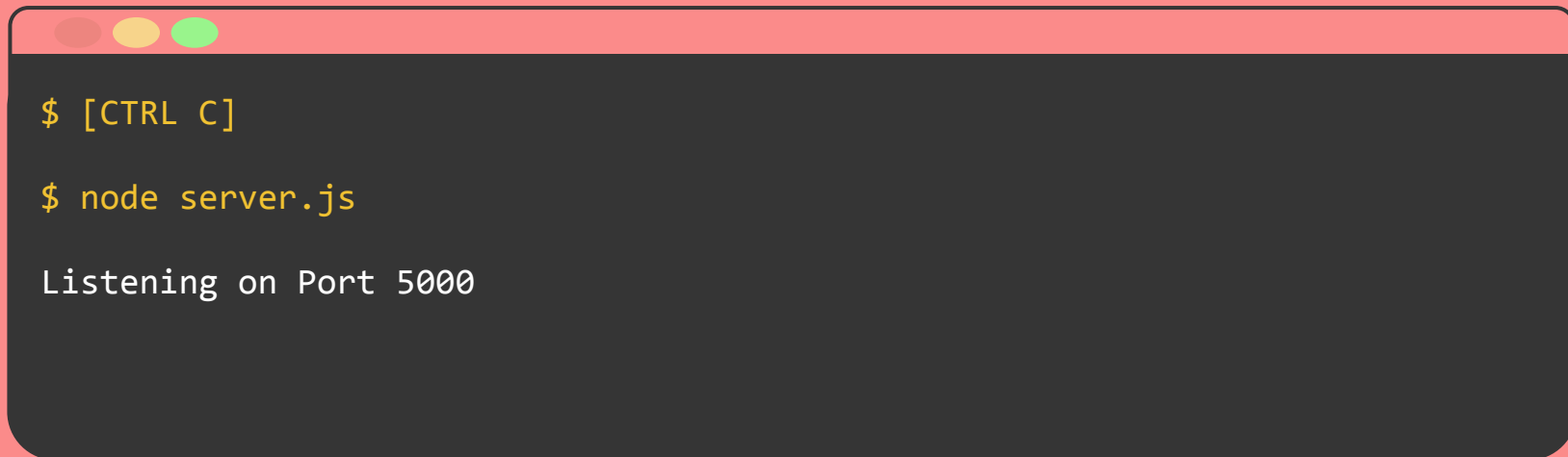
Write your first GraphQL call: *queries.js*

```
19 // Queries for product information
20 function fetchProducts() {
21   var query = `
22     query {
23       shop {
24         products(first: 4) {
25           edges {
26             node {
27               title
28               images(first: 1) {
29                 edges {
30                   node {
31                     src
32                   }
33                 }
34             }
35             variants(first: 1) {
36 // Code Continues Below
```

Let's test the game

1. Type [CTRL] [C] in the command line to kill the server.
2. Type `node server.js` to restart the server.

Mac and Windows

A terminal window with a dark background and a light gray title bar. The title bar contains three colored window control buttons (red, yellow, green). The terminal shows a prompt character '\$' followed by the command '[CTRL C]' in yellow text. Below that, another prompt '\$' is followed by the command 'node server.js' in yellow text. The output of the command is 'Listening on Port 5000' in white text.

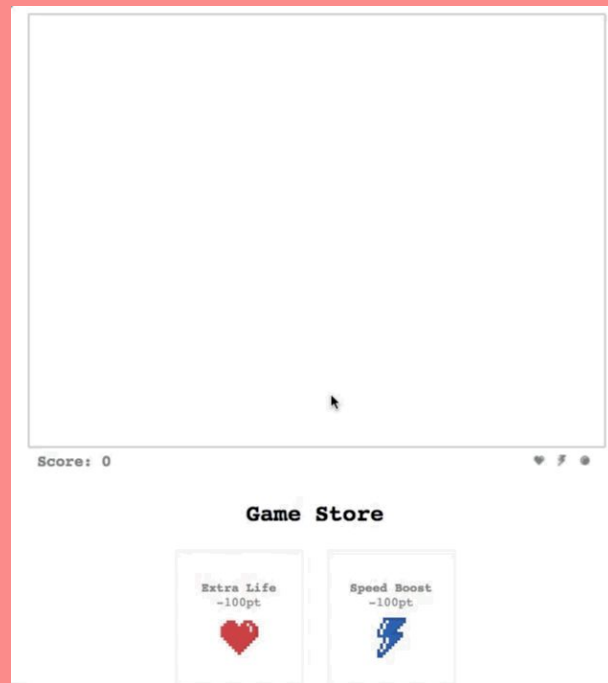
```
$ [CTRL C]

$ node server.js

Listening on Port 5000
```


Let's test the game

1. Refresh **localhost:5000** in the browser.
2. Try to purchase a power-up. What happens?



What queries do we need?

We completed the first query. Now let's do the second!

~~1. Retrieve the products from your store~~

2. Create the Checkout

3. Complete the Checkout

Write a mutation: Delete the double slashes `//` on lines 53 to 64 in *queries.js*

```
55 // function buyPowerUp(variantId) {
56 //   var query = `
57 //     mutation {
58 //       checkoutCreate(input: {
59 //         lineItems: [{
60 //
61 //           }]
62 //       }) {
63 //         checkout {
64 //
65 //           }
66 //         }
67 //       }
68 //     `;
69 //
70 //   return makeRequest(query);
71 // }
```

Code review

- Line 55 declares a function called `buyPowerUp()` that takes a single argument, `variantId`.
- `variantId` is the power-up that the user clicked on the website.
- Line 56 creates a new variable called `query`.

```
55  function buyPowerUp(variantId) {  
56      var query = `  
57  
..  
67  
68  `;  
69  
70  return makeRequest(query);  
71  }
```

Code review

Your code should look like this. Let's break it into parts.

```
55  function buyPowerUp(variantId) {
56      var query = `
57          mutation {
58              checkoutCreate(input: {
59                  lineItems: [{
60
61                      }]
62              }) {
63                  checkout {
64
65                      }
66              }
67          }
68      `;
69
70      return makeRequest(query);
71  }
```

Code review

- Line 55 begins the mutation
- Line 56 specifies which mutation - `checkoutCreate()`
- Because `checkoutCreate()` is a mutation, it takes an argument, `input` (line 56), and returns a value, `checkout` (line 61).

```
57     mutation {
58         checkoutCreate(input: {
59             lineItems: [{
60
61             }]
62         }) {
63             checkout {
64
65             }
66         }
67     }
```

Code review

At <https://help.shopify.com/api/storefront-api/reference/mutation/checkoutcreate>, you can see a list of input fields that the `input` argument accepts.

- On line 59, the `lineItems` input field is added.
- The `lineItems` input field provides information about the items purchased.

```
57     mutation {
58         checkoutCreate(input: {
59             lineItems: [{
60
61             }]
62         }) {
63             checkout {
64
65             }
66         }
67     }
```

Challenge

Navigate to

<https://help.shopify.com/api/storefront-api/reference/mutation/checkoutcreate>

and explore the documentation to find which arguments `lineItems` requires.

```
57     mutation {
58         checkoutCreate(input: {
59             lineItems: [{
60
61             }]
62         }) {
63             checkout {
64
65             }
66         }
67     }
```


Solution

lineItems requires two arguments:

- quantity
- variantId

```
57     mutation {
58         checkoutCreate(input: {
59             lineItems: [{
60
61             }]
62         }) {
63             checkout {
64
65             }
66         }
67     }
```

Update your code

- On line 60, we added the `quantity` argument with a value of 1 which means the default number of power-ups to buy is 1.
- On line 61, we added the `variantId` argument with a value of `"${variantId}"` which takes the value we passed to the function on line 55 and puts into in the `query` variable.

```
57     mutation {
58         checkoutCreate(input: {
59             lineItems: [{
60                 quantity: 1,
61                 variantId: "${variantId}"
62             }]
63         }) {
64             checkout {
65
66             }
67         }
68     }
```

Challenge

- The `checkout` return field (line 62) has no input fields.
- We're going to add three. The first is `webUrl`.
- Look at the `checkCompletedPurchases()` function at the bottom of **queries.js** and compare it to the documentation for `checkout` to see if you can determine the other two input fields!

```
57     mutation {
58         checkoutCreate(input: {
59             lineItems: [{
60                 quantity: 1,
61                 variantId: "${variantId}"
62             }]
63         }) {
64             checkout {
65
66             }
67         }
68     }
```

Solution

The three fields are `webUrl`, `completedAt`, and `id`.

```
63     }) {
64     checkout {
65         webUrl
66         completedAt
67         id
68     }
69 }
```

Let's test the game

- Type [CTRL] [C] in the command line to kill the server.
- Type `node server.js` to restart the server.

Mac and Windows



```
$ [CTRL C]
```

```
$ node server.js
```

```
Listening on Port 5000
```

Test your game!

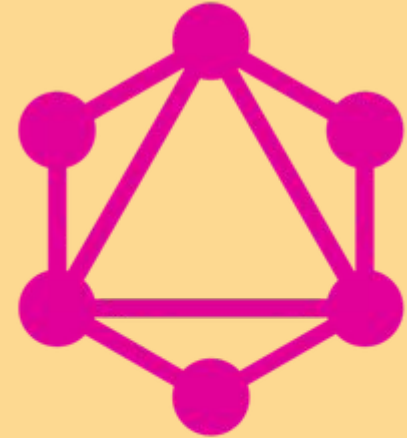


Table of contents

1. Introduction to APIs and GraphQL
2. Preview the app
3. Set up Shopify storefront
4. Write your own GraphQL calls
- 5. Review**
6. Next steps

Let's recap quickly...

1. GraphQL is a new specification for interacting with APIs
2. GraphQL calls fall into two groups - queries and mutations
3. GraphQL allows you to request only the information you need, making it easy to create a Shopify storefront in your app while avoiding data overload.



GraphQL

Table of contents

1. Introduction to APIs and GraphQL
2. Preview the app
3. Set up Shopify storefront
4. Write your own GraphQL calls
5. Review
- 6. Next steps**

Keep learning

Practice problems for later

1. Categories

Challenge: Reorganize your products into categories, which will require you to update the requests you make.

2. Subscriptions

Challenge: Learn about the third type of GraphQL query - a subscription - and try to recreate the 3rd GraphQL call in queries.js from scratch, with help from the documentation

Continue your learning



- Read the GraphQL documentation:
<http://graphql.org/learn/>
- Read the Shopify Storefront documentation:
<https://help.shopify.com/api/storefront-api>
- Read the Shopify Admin documentation:
<https://help.shopify.com/en/api/graphql-admin-api>
- Discover other APIs using GraphQL
<http://graphql.org/users/>

Shopify developer program



- Solve interesting problems for over 600,000 business owners worldwide
- Keep 80% of any app revenue you generate
- Refer stores and generate ongoing income
- Help build the future of commerce!

<http://developers.shopify.com>

@ShopifyDevs

Have a couple minutes?

Please take this super short survey! Your feedback is a gift. 🙌

<https://bit.ly/madevcon>



<http://developers.shopify.com>
@ShopifyDevs

Thank you! Don't be a stranger!

And don't forget your socks 🦶

<http://developers.shopify.com>

@ShopifyDevs

