

# 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!





# 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 interface with three colored circular icons at the top (red, yellow, green). It displays a JSON object with three properties: name, city, and topics. The value for topics is an array containing two strings: "GraphQL" and "Shopify".

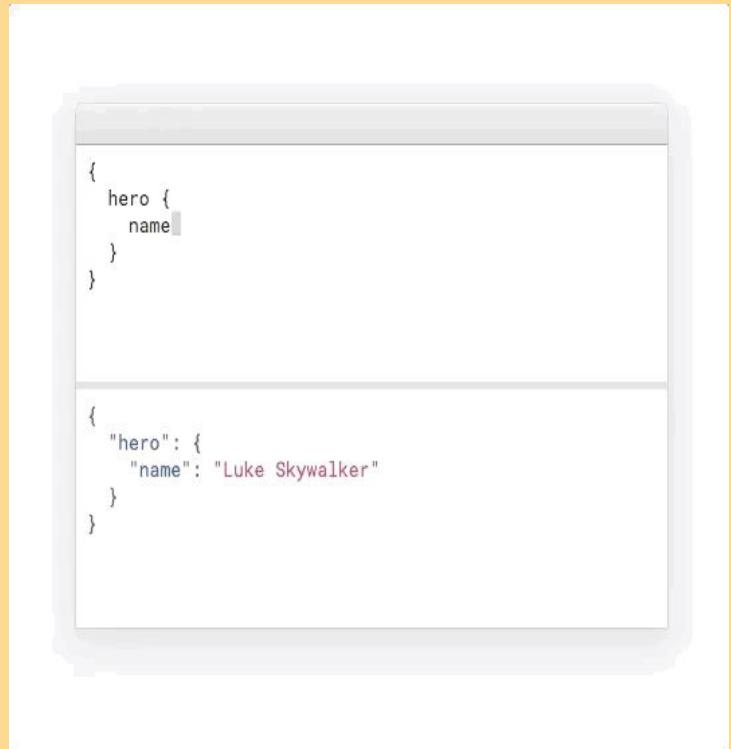
```
{  
  "name" : "Peers Conf",  
  "city" : "Austin",  
  "topics" : ["GraphQL",  
             "Shopify"]  
}
```

# What is GraphQL?

GraphQL stands for **Graph Query Language**.

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

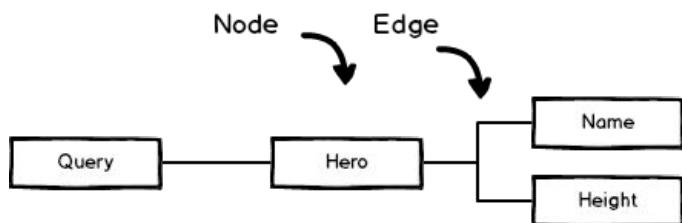
When using a GraphQL API, an application can request multiple resources at a time and only receive what it needs. The information returned looks like this:



The image shows a GraphQL playground interface. At the top, there is a code editor window containing a GraphQL query:{ hero { name }Below this is a results window showing the response from the GraphQL server:{ "hero": { "name": "Luke Skywalker" }}The results window has a light gray background and a white border.

# The graph in GraphQL

A partial data graph

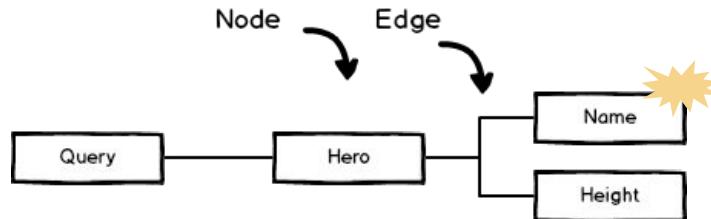


GraphQL schema

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

# GraphQL execution

A partial data graph

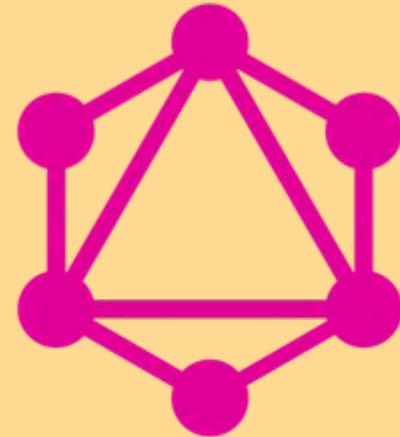


Resolver

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

# Benefits of GraphQL

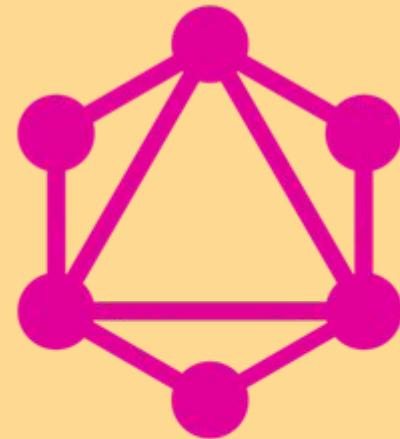
- GraphQL allows you to specify the information you want to retrieve.
- GraphQL has a feature called "introspection" where developers can use commands to ask the server about what queries are allowed.
- GraphQL allows you to retrieve information from multiple data sources in a single request, speeding 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  
05      birthday  
06    }  
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

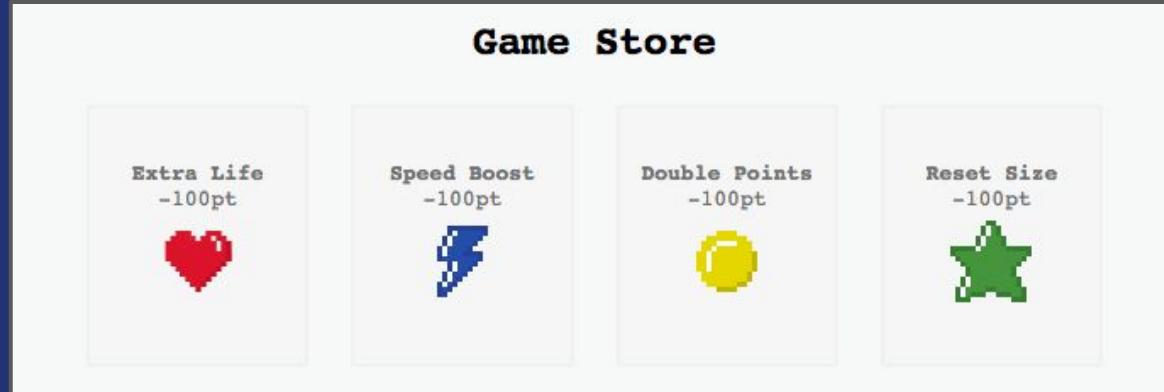


```
$ cd mlh-localhost-shopify-graphql-master  
$ ls  
README.md  node_modules/  package.json  public/  server.js  
$ node server.js  
Listening on http://localhost:5000/
```

# Step 5: navigate to the URL below

localhost:5000

*Notice something missing?*



# Set up Shopify for development

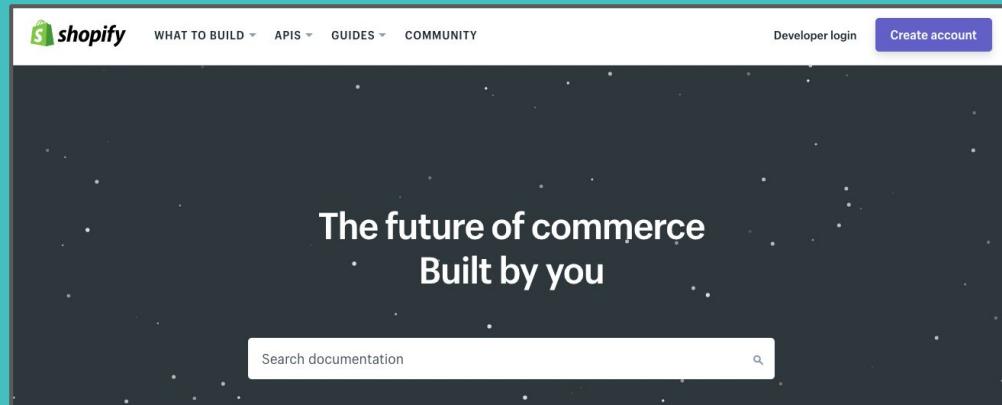
<https://developers.shopify.com>

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

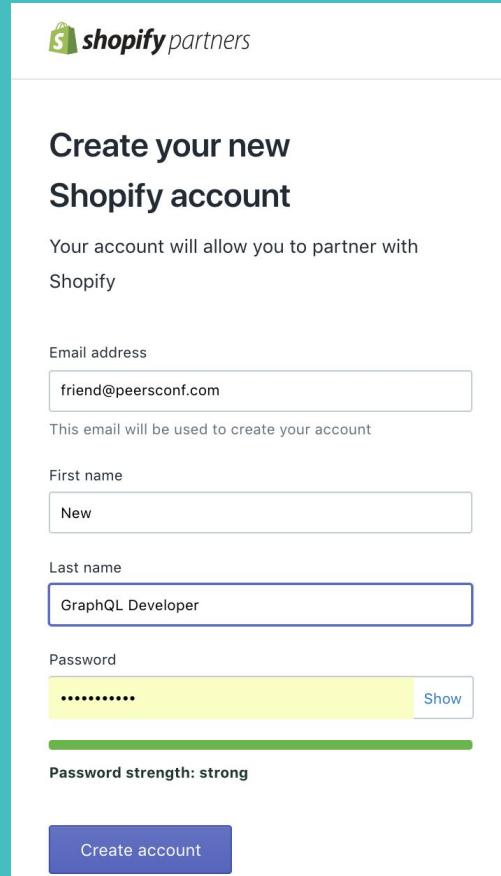
<https://developers.shopify.com>

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 image shows a screenshot of the Shopify Partners registration form. At the top, there's a logo for "shopify partners". Below it, the heading "Create your new Shopify account" is displayed. A subtext explains that the account will allow partnering with Shopify. The form contains several input fields: "Email address" (with the value "friend@peersconf.com"), "First name" (with the value "New"), "Last name" (with the value "GraphQL Developer", which is highlighted with a blue border), and "Password" (with a yellow placeholder bar showing "\*\*\*\*\*" and a "Show" link). Below the password field is a green progress bar indicating "Password strength: strong". At the bottom is a large blue "Create account" button.

shopify partners

Create your new  
Shopify account

Your account will allow you to partner with  
Shopify

Email address

friend@peersconf.com

This email will be used to create your account

First name

New

Last name

GraphQL Developer

Password

\*\*\*\*\* Show

Password strength: strong

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. It includes fields for Business name (GraphQL Inc), Generic business email (developer@graphql.com), Website (optional) (GraphQL is my life), City (Austin), Country (United States), and State (Texas). At the bottom, there's a section titled "WHAT WOULD YOU LIKE TO LEARN MORE ABOUT?" with two options: "Basic Shopify store setup" (unchecked) and "Custom Shopify store design" (checked).

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?	
<input type="checkbox"/>	Basic Shopify store setup
<input checked="" type="checkbox"/>	Custom Shopify store design

# Register for a Shopify developer account

7. Click “Development stores.”

The screenshot shows the Shopify Partners dashboard. At the top left is the Shopify Partners logo. To its right is a search bar with the placeholder "Search...". On the far right, there's a purple circular icon with "GR" and the text "GraphQL Inc New GraphQL Developer". The main navigation menu on the left includes links for "Development stores" (which is highlighted with a red border), "Managed stores", "Referred stores", "Affiliate tools", "Apps", "Themes", "Payouts", "Settings", and "Team". The "Development stores" link has a red box around it. The central area is titled "Resources". Below it is a section titled "Partner Perks" with the subtext "Exclusive discounts on tools that help you run your business." To the right of this is a callout box containing text about Shopify Partner Perks and a "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

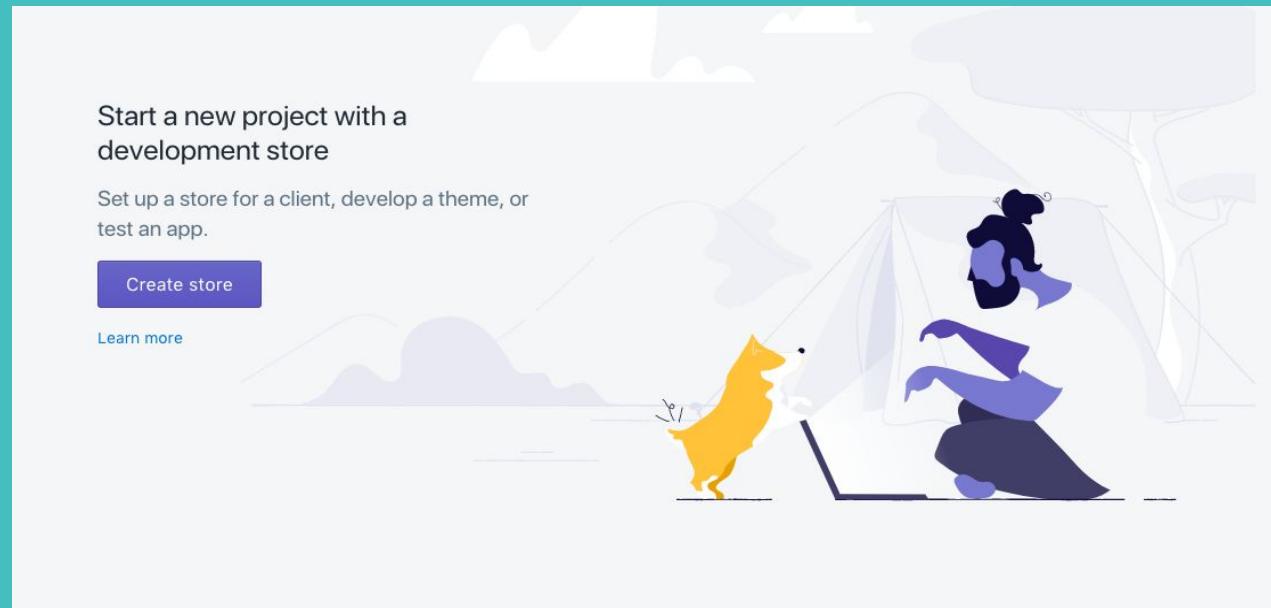
Exclusive discounts on tools that help you run your business.

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”



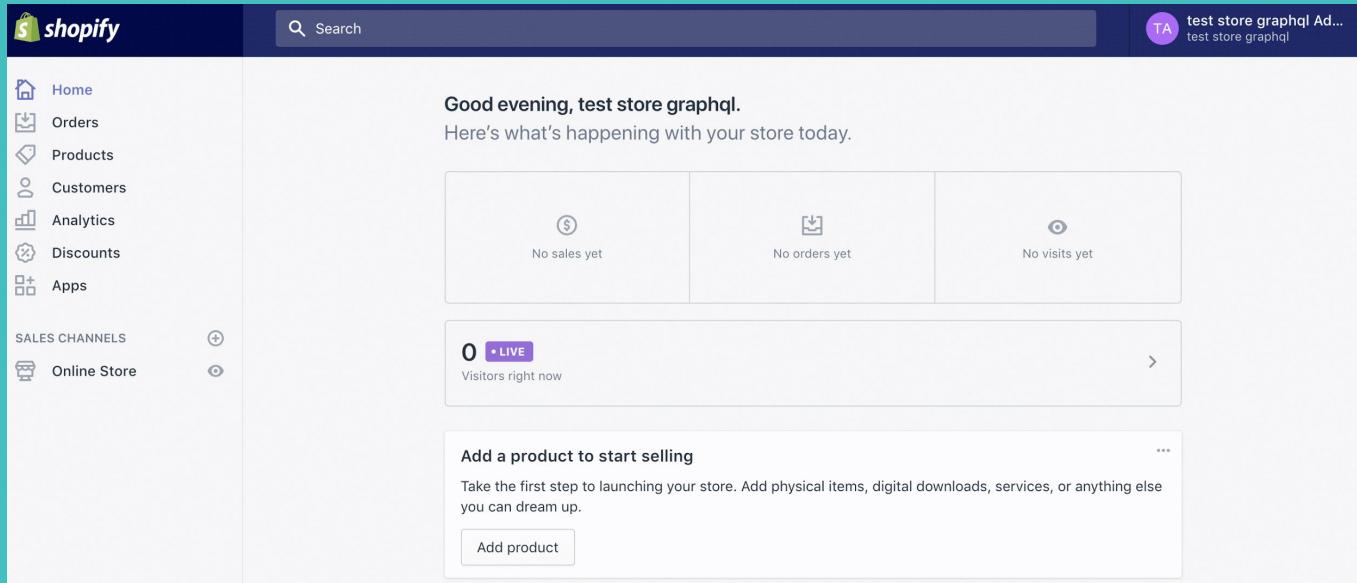
# Register for a Shopify developer account

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 Shopify Partners interface for creating a new development store. On the left is a sidebar with navigation links: Development stores, Managed stores, Referred stores, Affiliate tools, Apps, Themes, Payouts, Settings, Team, and Resources. The main area is titled 'New development store' and contains two sections: 'Login information' and 'Store details'. In the 'Login information' section, there are fields for 'Store name (store domain)' containing 'Unique GraphQL Store Name' with a green checkmark, 'Login' containing 'developer@graphql.com', 'Password' (redacted), and 'Confirm password' (redacted). In the 'Store details' section, there is a field for 'Address' containing '701 Brazos Street'. At the top right of the form are 'Cancel' and 'Save' buttons, with 'Save' being 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.



# Get credentials

13. Click “Create a new private app.”

The screenshot shows a user interface for creating a new private app. At the top left, there is a back arrow labeled 'Apps'. Below it, the title 'Private apps' is displayed. In the center, there is a large icon consisting of four squares arranged in a 2x2 grid, with a plus sign in the middle square. Below this icon, the text 'Create a new private app' is centered. A descriptive paragraph follows: 'Private apps can streamline store processes like accounting, importing products, or even create unique customer-facing storefronts.' At the bottom, a prominent blue button with white text says 'Create a new private app'. At the very bottom, there is a callout bubble containing a question mark icon and the text 'Learn more about private apps.'

>Create a new private app

Private apps can streamline store processes like accounting, importing products, or even create unique customer-facing storefronts.

Create a new private app

?

Learn more about private apps.

# Get credentials

14. Name your private app.

15. Enter an email.

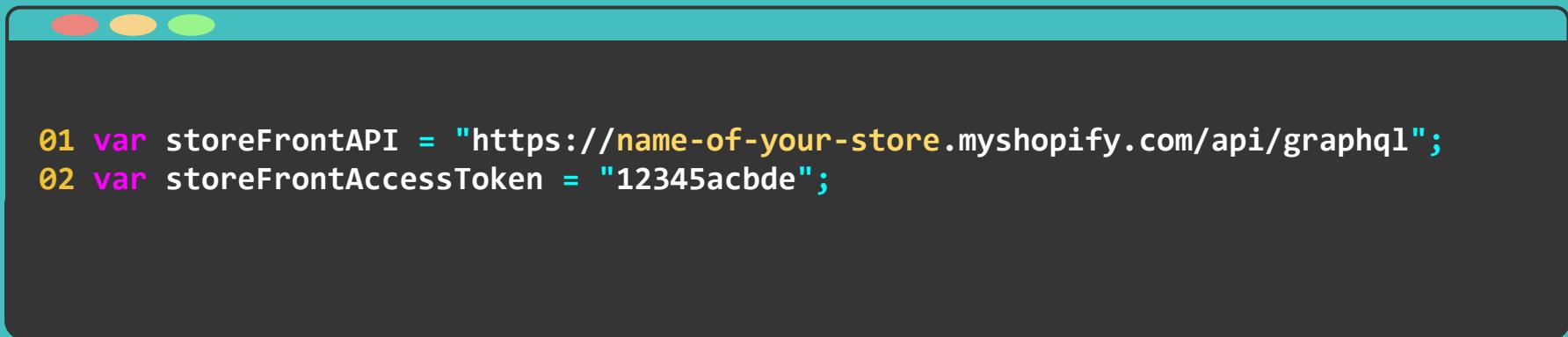
16. Click "Allow this app to access your storefront data using the Storefront API."

17. Click "Save."

The screenshot shows the Shopify Admin interface with a dark blue header. On the left, there's a sidebar with icons for Home, Orders, Products, Customers, Analytics, Discounts, and Apps. Below that is a section for Sales Channels with an Online Store option. The main content area has a title 'Create private app' and a 'Description' section with a note about tracking private apps. It includes fields for 'Private app name' and 'Contact email (optional)'. In the bottom right corner of the main area, there's a note: 'Your API credentials will be generated when you Save.' Below this, under 'Admin API', there are two sections: one for 'Store content like articles, blogs, comments, pages, and redirects' with a 'read\_content, write\_content' permission and a 'Read access' dropdown; and another for 'Customer details and customer groups' with a 'read\_customers, write\_customers' permission and a 'Read access' dropdown. At the very bottom, there are 'Next Step' and 'Cancel' buttons.

# 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.

A screenshot of a Mac OS X desktop environment. At the top, there's a dark grey menu bar with the Apple logo and some icons. Below it is a dark grey dock with several icons. The main window is a terminal or code editor with a black background and light grey text. It shows two lines of code:

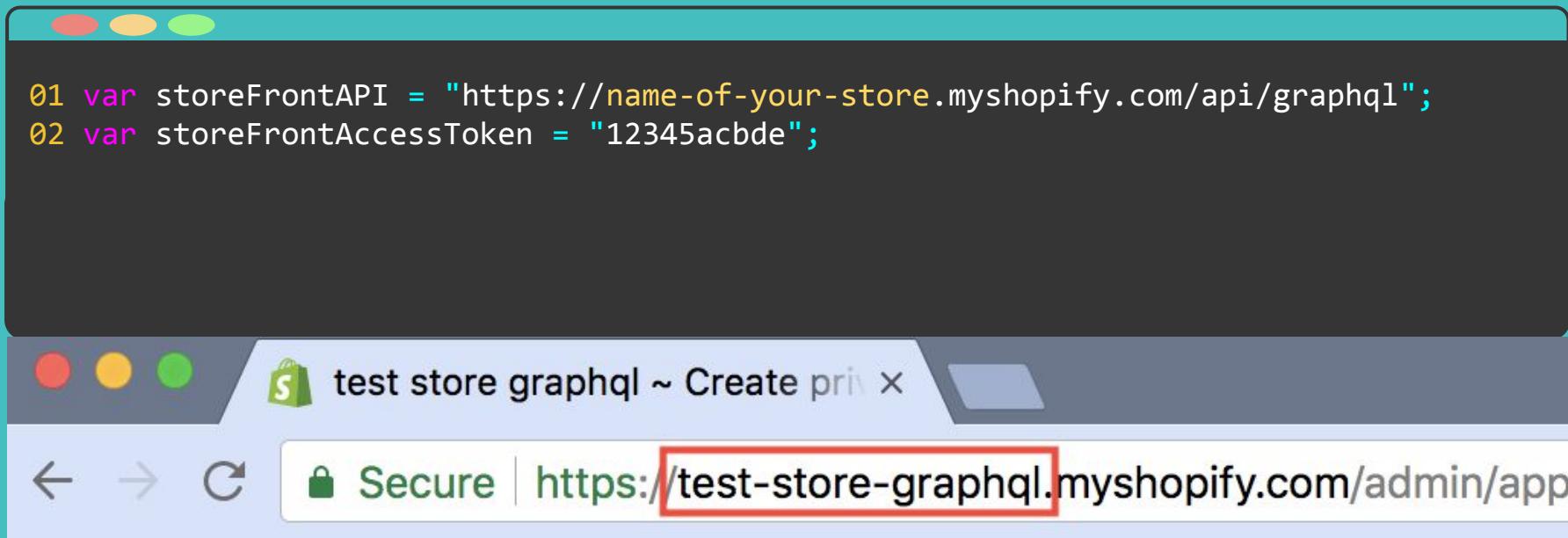
```
01 var storefrontAPI = "https://name-of-your-store.myshopify.com/api/graphql";
02 var storefrontAccessToken = "12345acbde";
```

The window has the standard Mac OS X title bar with red, yellow, and green buttons.

# 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.

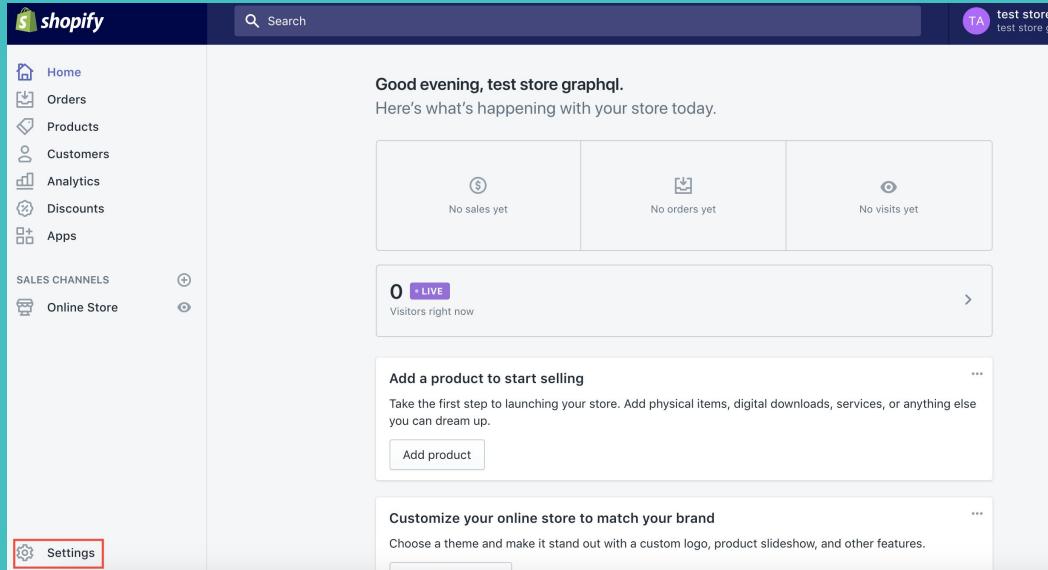
```
01 var storefrontAPI = "https://name-of-your-store.myshopify.com/api/graphql";  
02 var storefrontAccessToken = "12345acbde";
```



# 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.



# Set up payments

2. Click “Payment providers”

The screenshot shows the Shopify admin interface with a dark blue header. The header includes the Shopify logo, a search bar, and a user profile icon labeled "test store graphql Ad. test store graphql". The main navigation bar on the left has links for Home, Orders, Products, Customers, Analytics, Discounts, Apps, Sales Channels (with "Online Store" selected), and a plus sign icon. The main content area is titled "Settings". A grid of settings cards is displayed. The "Payment providers" card, which contains the text "Enable and manage your store's payment providers", is highlighted with a red rectangular border. Other visible cards include 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), Gift cards (Enable Apple Wallet passes and set gift card expiry dates), Account (Manage your accounts and permissions), Notifications (Manage notifications sent to you and your customers), Billing (Manage your billing information and view your invoices), Checkout (Customize your online checkout process), Shipping (Manage how you ship orders to customers), and Files (Upload images, videos, and documents).

Search

TA test store graphql Ad.  
test store graphql

Home

Orders

Products

Customers

Analytics

Discounts

Apps

Sales Channels

Online Store

Settings

General

Taxes

Sales channels

Payment providers

Gift cards

Account

Notifications

Billing

Checkout

Shipping

Files

# 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.

# 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! 

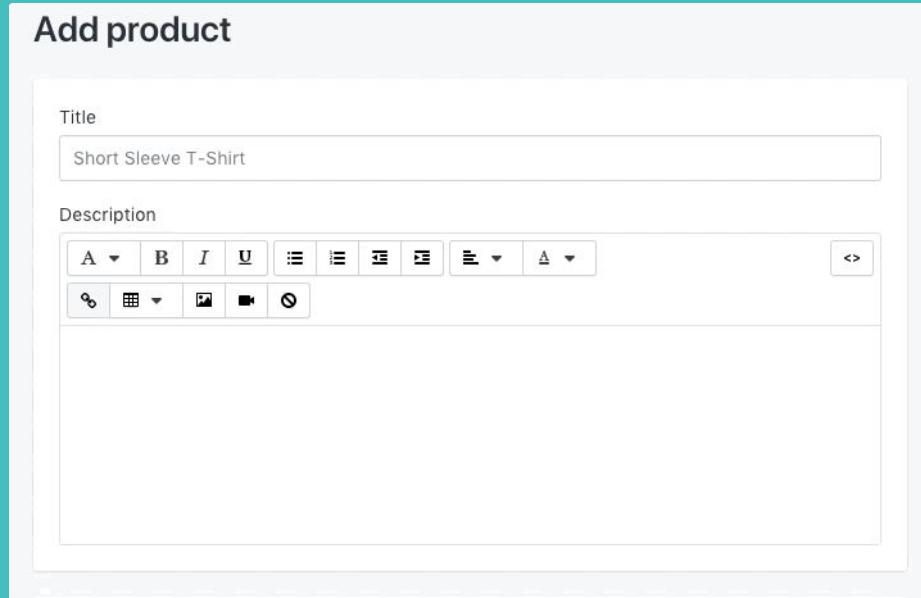
# Add power-ups to store

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 shows the Shopify account home page. At the top, there's a dark header with the Shopify logo, a search bar, and a user profile icon labeled "test store graphql Ad...". The main navigation menu on the left includes "Home", "Orders", "Products" (which is highlighted in blue), "Customers", "Analytics", "Discounts", and "Apps". Below this, under "SALES CHANNELS", is "Online Store". The central content area displays a message: "Good evening, test store graphql. Here's what's happening with your store today." It features three cards: "No sales yet" (with a dollar sign icon), "No orders yet" (with a download icon), and "No visits yet" (with an eye icon). Below this is another card showing "0 + LIVE" visitors right now. At the bottom, there's a call-to-action box with the text "Add a product to start selling" and a "Add product" button.

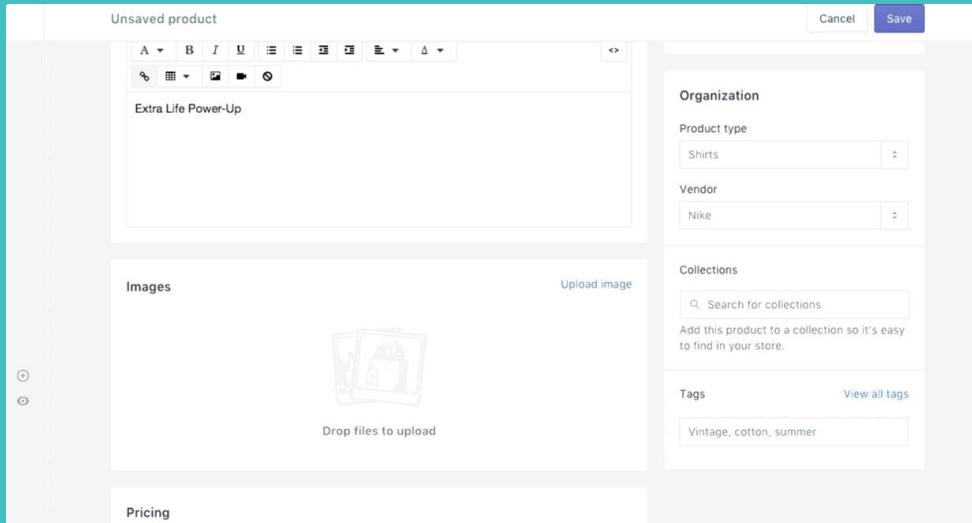
# Add power-ups to store

4. Name the product "Extra Life."
5. Give the product a simple description, like "Extra Life Power-Up."



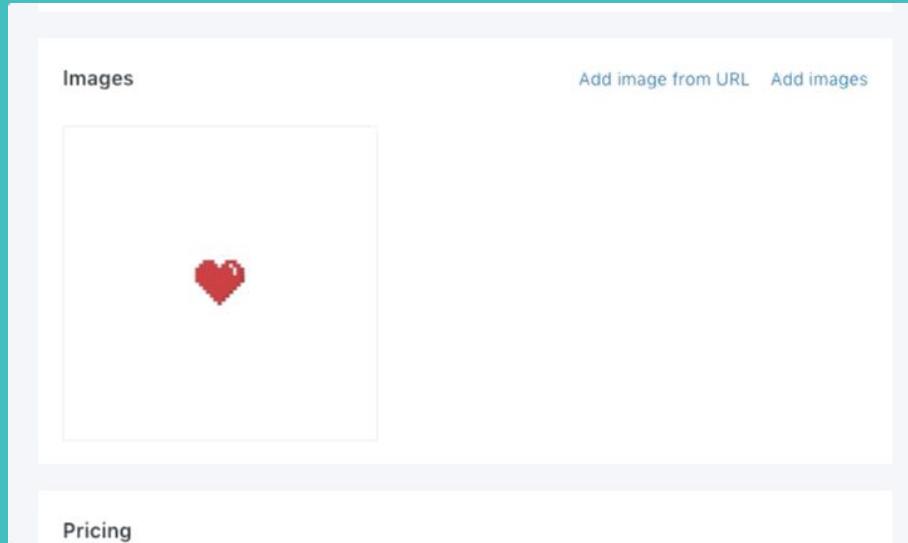
# Add power-ups to store

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.



# 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 a GraphQL IDE interface. At the top, there is a navigation bar with three tabs: "GraphiQL" (disabled), "Prettify" (highlighted with a red box), and "History". To the right of the navigation bar is a "Docs" button, also highlighted with a red box. The main area is a code editor with a light gray background. The code is displayed with line numbers on the left and some lines starting with "#". A vertical scroll bar is visible on the right side of the code editor. The code content is as follows:

```
1 # Welcome to GraphiQL
2 #
3 # GraphiQL 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 typeheads 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-Space
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 GraphiQL interface. On the left, the query editor contains the following GraphQL code:

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

The right side of the interface is the Documentation Explorer. It includes a search bar labeled "Search Schema..." and a description: "A GraphQL schema provides a root type for each kind of operation." Below this, under "ROOT TYPES", it lists "query: [QueryRoot](#)" and "mutation: [Mutation](#)".

# Query to fetch products

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

The screenshot shows a GraphQL playground interface with the following components:

- GraphQL**: The title of the playground.
- Play**: A button to execute the query.
- Prettify**: A button to format the JSON response.
- History**: A button to view previous queries.
- QueryRoot**: The current schema entry-point.
- Schema**: A link to the full schema definition.
- X**: A close button.
- Search QueryRoot...**: A search bar for the schema.
- FIELDS**: A section listing the fields available at the QueryRoot level.
- customer**: Returns a Customer object given a customerAccessToken.
- node**: Returns a Node object given an ID.
- nodes**: Returns a list of Node objects given a list of IDs.
- shop**: Returns a Shop object.

In the main area, the following GraphQL query is displayed:

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

# Query to fetch products

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

The screenshot shows a GraphiQL interface with a query editor and a documentation sidebar.

**GraphiQL:** The left pane contains the following GraphQL query:

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

**Documentation Sidebar:** The right pane displays information about the `Shop` type.

**Shop Type Definition:**

```
QueryRoot Shop X  
Shop represents a collection of the general settings and information about the shop.  
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 a GraphQL explorer with the following interface elements:

- GraphQL** tab (highlighted with a red box).
- Prettify** and **History** tabs.
- Query Editor**: A code editor containing the following GraphQL query:

```
1 * {  
2   shop {  
3     products  
4   }  
5 }  
6
```
- Result Panel**: A table with the following columns:
  - Shop**: Description: "List of the shop's products."
  - products**: Type: "ProductConnection!".
    - TYPE**
    - 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 a GraphiQL interface with a query editor and a results panel.

**GraphQL Query (Editor):**

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

**Result Panel:**

The result panel displays the response from the GraphQL server. The response is a JSON object with a "data" field set to null and an "errors" field containing an array of errors. One error message is highlighted in pink: "you must provide one of first or last".

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

**Details Panel:**

This panel provides metadata about the "products" field. It includes a description, type information, and a list of arguments.

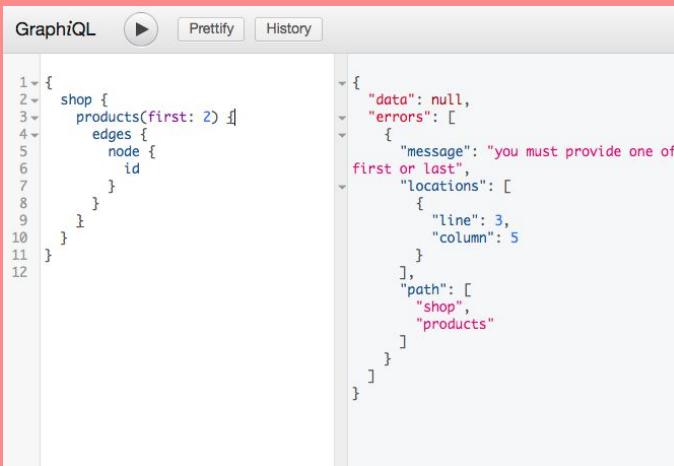
- DESCRIPTION:** List of the shop's products.
- TYPE:** ProductConnection!
- ARGUMENTS:**
  - first: Int
  - after: String
  - last: Int
  - before: String
  - reverse: Boolean = false
  - sortKey: ProductSortKeys = ID

# 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 GraphiQL interface. On the left, there is a code editor window containing a GraphQL query. On the right, there is a results window displaying the JSON response from the query.

```
1 {  
2   shop {  
3     products(first: 2) {  
4       edges {  
5         node {  
6           id  
7         }  
8       }  
9     }  
10   }  
11 }  
  
{  
  "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 shows the GraphiQL interface with a query on the left and a response on the right.

**GraphQL Query:**

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

**GraphiQL Response:**

```
{  
  "data": {  
    "shop": {  
      "products": {  
        "edges": [  
          {  
            "node": {  
              "id": "Z2lkc0i8vc2hvcGlme59Qcm9kdWN0Lzk40TUyNzYwOTk="  
            }  
          },  
          {  
            "node": {  
              "id": "Z2lkc0i8vc2hvcGlme59Qcm9kdWN0Lzk40TUyNzkwNDM="  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

**Shop Product Type Definition:**

List of the shop's products.

**ProductConnection!**

**Arguments:**

- first: Int
- after: String
- last: Int
- before: String
- reverse: Boolean = false
- sortKey: 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 a GraphQL interface with a query editor and a results panel.

**GraphiQL** (top left) contains the following query:

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

The results panel shows the response from the query. On the right, there is a detailed view of the "edges" field for a "ProductEdge".

**edges** (left side of the results panel) shows the structure of the edges field in the query result:

```
{  
  "data": {  
    "shop": {  
      "products": {  
        "edges": [  
          {  
            "node": {  
              "id": "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0Lzk40TUyNzYwOTk="  
            }  
          },  
          {  
            "node": {  
              "id": "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0Lzk40TUyNzkwNDM="  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

**ProductEdge** (right side of the results panel) provides a detailed description of the "edges" field:

- 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 a GraphQL IDE interface with three main sections: a code editor, a results panel, and a detailed view of a specific field.

**Code Editor:**

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

**Results Panel:**

A list of products is shown, each with an ID and title:

- "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0Lzk40TUyNzYwOTk=" (Snare Boot)
- "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0Lzk40TUyNzkwNDM=" (Neptune Boot)

**ProductConnection Detail View:**

**Search:** Search ProductConnection...

**No Description**

**FIELDS**

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

# Query to fetch products

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

The screenshot shows the GraphiQL interface with a query editor and a results panel.

**GraphQL Query (Left):**

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

**GraphQL Results (Right):**

```
{  
  "data": {  
    "shop": {  
      "products": {  
        "edges": [  
          {  
            "node": {  
              "id": "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0Lzk40TUyNzYwOTk="  
            }  
          },  
          {  
            "node": {  
              "id": "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0Lzk40TUyNzkwNDM="  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

**Product Edge Details (Panel on the right):**

- Product Edge:** Shows a search bar and a detailed 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)."
- Product:** Shows the **IMPLEMENTES** relationship to **Node**.
- FIELDS:** Shows the **collections** field with its arguments: `first: Int`, `after: String`, `last: Int`, `before: String`, `reverse: Boolean = false`, and the return type `: CollectionConnection!`. A note below states: "List of collections a product belongs to."

# Query to fetch products

**Answer:** images and title

The screenshot shows the GraphiQL interface with a query editor and a results viewer.

**GraphiQL Editor:**

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

**Results Viewer:**

The results are displayed in a tree-like structure:

- "data": {
  - "shop": {
    - "products": {
      - "edges": [
        - {
          - "node": {
            - "id": "Z2lkc0i8vc2hvcGlme59Qcm9kdWN0Lzk40TUyNzYwOTk="

**Product Edge Details:**

**Product Edge:** 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).

**IMPLEMENTES**

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

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 GraphiQL interface with a query editor on the left and a results panel on the right.

**GraphiQL Query:**

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

**GraphiQL Results:**

```
{  
  "data": {  
    "shop": {  
      "products": {  
        "edges": [  
          {  
            "node": {  
              "id": "Z2lkc018vc2hvcGlmes9Qcm9kdWN0Lzk40TUyNzYwOTk=",  
              "title": "A Product Title"  
            }  
          },  
          {  
            "node": {  
              "id": "Z2lkc018vc2hvcGlmes9Qcm9kdWN0Lzk40TUyNzkwNDM=",  
              "title": "Another Product Title"  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

**Product Edge Details:**

Product Edge

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).

IMPLEMENTES

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

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

18. Click the Play symbol.

The screenshot shows a GraphQL IDE interface. On the left, there is a code editor with a GraphQL query. On the right, there is a tooltip for the 'edges' field of the 'ImageConnection' type.

**GraphQL Query:**

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

**ImageConnection Type Definition (Tooltip):**

FIELDS
edges: [ImageEdge!]!
pageInfo: PageInfo!

The tooltip provides a description for the 'edges' field: "A list of edges." It also mentions "Information to aid in pagination."

# 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 GraphiQL interface with a query editor and a results panel.

**GraphiQL:**

```
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}
```

**Results:**

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

**Product > ImageConnection:**

- Search: Search ImageConnection...
- No Description
- FIELDS**
  - 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 GraphiQL interface with three main sections: the Query Editor, the Results pane, and a sidebar for the "ImageConnection" type.

**Query 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 }
```

**Results pane:**

```
{  
  "data": {  
    "shop": {  
      "products": {  
        "edges": [  
          {  
            "node": {  
              "title": "Share Boot",  
              "images": {  
                "edges": [  
                  {  
                    "node": {  
                      "id": "Z2lk0i8vc2hvcGlmeS9Qcm9kdWN0SW1hZ2UvMjEyNzkxNTY5OTU="  
                    }  
                  ]  
                }  
              }  
            }  
          ]  
        }  
      }  
    }  
  },  
  "pageInfo": {  
    "hasNextPage": false,  
    "hasPreviousPage": false,  
    "startCursor": null,  
    "endCursor": null  
  }  
}
```

**ImageConnection Sidebar:**

- Product**: A back arrow icon.
- ImageConnection**: The current type being viewed.
- X**: A close button.

**Search Bar:** Search ImageConnection...

**No Description**

**FIELDS**

- edges: [ImageEdge!]!**: A list of edges.
- pageInfo: PageInfo!**: 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 GraphiQL interface with two main sections: the Query Editor and the Results pane.

**Query 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}
```

**Results pane:**

The results pane displays the JSON response from the GraphQL query. It includes a detailed description of the `ImageEdge` type and its fields.

```
{  
  "data": {  
    "shop": {  
      "products": {  
        "edges": [  
          {  
            "node": {  
              "title": "Share Boot",  
              "images": {  
                "edges": [  
                  {  
                    "node": {  
                      "id": "Z2lk0i8vc2hvcG1meS9Qcm9kdWN0SW1hZ2UvMjEyNzkxNTY5OTU="  
                    }  
                  }  
                ]  
              }  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

**Image Edge Type Description:**

- 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 brace below `src`, add "`variants(first: 1)`" and click the Play symbol.

The screenshot shows the GraphiQL interface with a query editor and a results panel.

**GraphiQL:**

```
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 }
```

**Results:**

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

**Image Edge Details:**

- ImageEdge:** 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

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**.

GraphiQL

Prettify History

```
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- }
```

1- f
2- shop {
3- "data": {
4- "shop": {
5- "products": {
6- "edges": [
7- {
8- "node": {
9- "title": "Snare Boot",
10- "images": [
11- {
12- "edges": [
13- {
14- "node": {
15- "src": "https://cdn.shopify.com/s/files/1/1312/0893/products/01\_grande\_89f870ed-dc56-4990-9a05-4f11ddf13108.jpg?v=1491918957"
16- }
17- ]
18- }
19- ]
20- }
21- },
22- {
23- "variants": {
24- "edges": [
25- {
26- "node": {
27- "id": "Z2lk0i8vcZhvcGlme59Qcm9kdNN0VmFyaWFudC8znjYwNzYyMjA4Mw=="
28- }
29- }
30- ]
31- }
32- }
33- ]
34- }
35- }
36- }
37- }

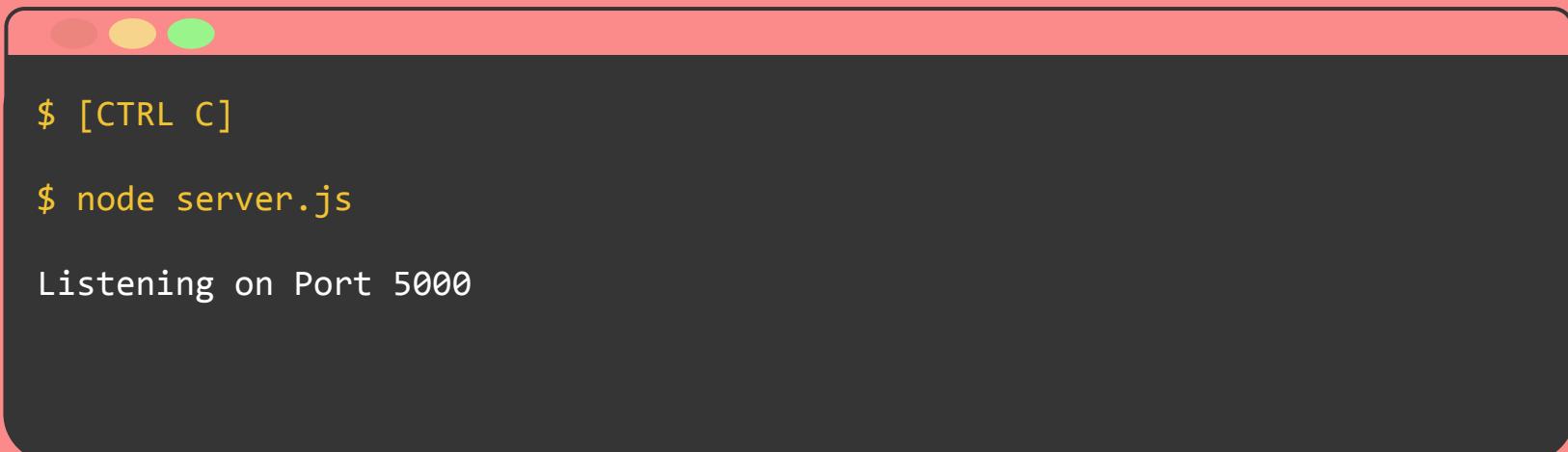
# Write your first GraphQL call: *queries.js*

```
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.

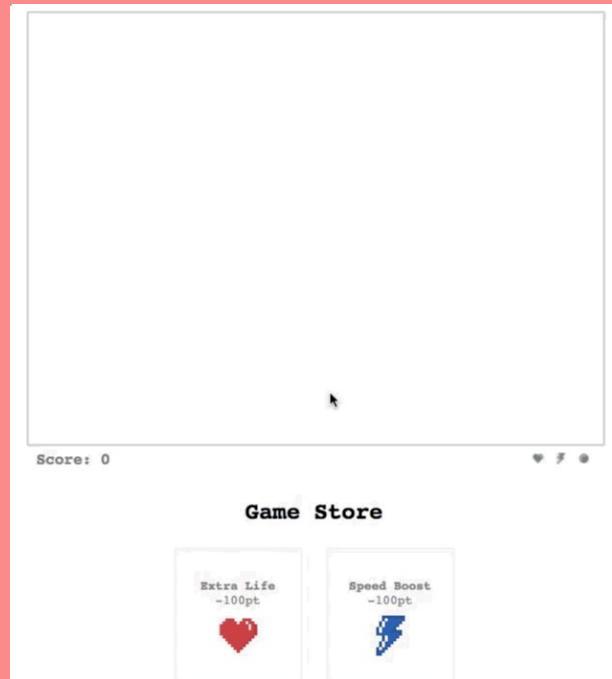
Mac and Windows



```
$ [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

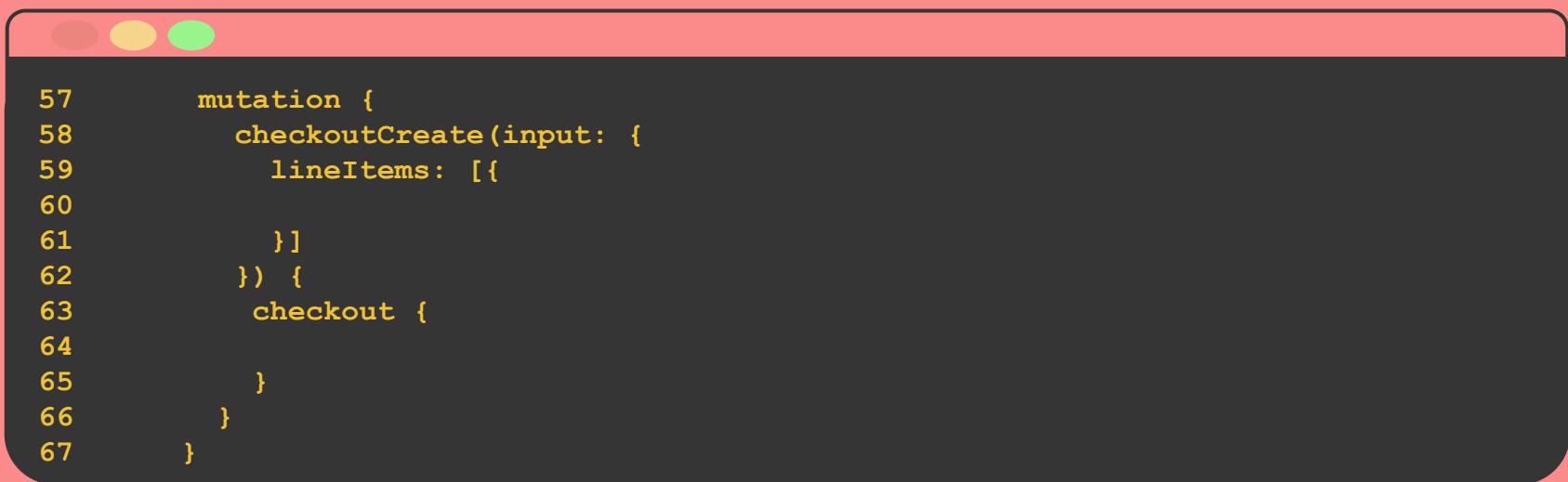
- Line 55 begins the mutation
- Line 56 specifies which mutation - checkoutCreate ()
- Because checkoutCreate () is a mutation, it takes an argument, `input` (line 56), and returns a value, `checkout` (line 61).

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



A screenshot of a code editor window showing a GraphQL mutation script. The window has a dark theme with three colored circular icons in the top-left corner (red, yellow, green). The code is numbered from 57 to 67. Lines 57-66 are part of the mutation block, and line 67 is the closing brace for the mutation.

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

# Solution

lineItems requires two arguments:

- quantity
- variantId

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

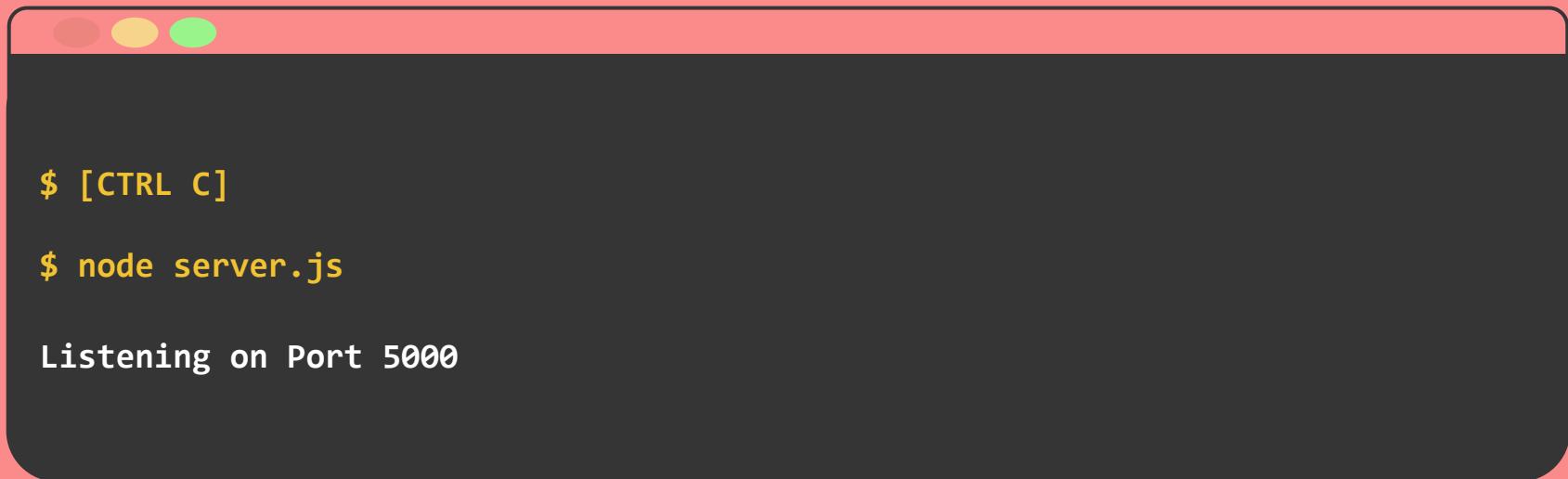
The three fields are `webUrl`, `completedAt`, and `id`.

```
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!

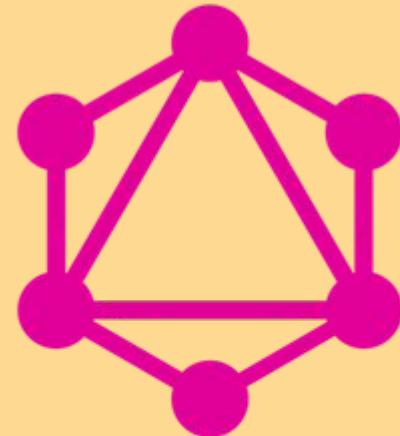


# Table of contents

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

