Welcome! My name is Evan

- 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!

Evan Huus, Developer Lead, Shopify
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) ->

```json
{
    "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 graph in GraphQL

A partial data graph

GraphQL schema

type Hero {
  name: String
  height: Float
}

Node → Edge

Query → Hero → Name → Height
GraphQL execution

A partial data graph

Resolver

```graphql
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 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.
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

```graphql
query {
  user(id: "abc123") {
    name
    email
    birthday
  }
}
```
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

```json
{  
  "user": {
    "name": "MidDevCon",
    "email": "admin@middevcon.com",
    "birthday": "July 13, 2018"
  }
}
```
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

Windows

```bash
$ cd ~/Downloads

$ Expand-Archive mlh-localhost-shopify-graphql-master.zip .
```

Mac

```bash
$ cd ~/Downloads

$ unzip mlh-localhost-shopify-graphql-master.zip
```

Do not forget the "." in this command.
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”
Register for a Shopify developer account

6. Fill in the rest of the form and click “See Partner dashboard” at the bottom.
Register for a Shopify developer account

7. Click “Development stores.”
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.”
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.
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."
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.

```javascript
var storeFrontAPI = "https://name-of-your-store.myshopify.com/api/graphql";
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.

```javascript
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”
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."
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."
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

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

```javascript
function makeRequest(query) {
  var headers = {
    "X-Shopify-Storefront-Access-Token": storeFrontAccessToken,
    "Content-Type": "application/json"
  };
  return $.ajax({
    url: storeFrontAPI,
    type: "POST",
    data: JSON.stringify({ query: query }),
    headers: headers
  });
}
```
Write your first GraphQL call

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

```graphql
{ 
  shop {
    name
  }
}
```
Query to fetch products

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

6. Click "products."
Query to fetch products

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

8. Click the Play symbol.
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."
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.

**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.
Query to fetch products

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

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

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

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

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

Answer: images and title
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.
Query to fetch products

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

18. Click the Play symbol.
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.
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."
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.
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.
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 GraphiQL explorer into `queries.js`.
Write your first GraphQL call: queries.js

```
// Queries for product information
function fetchProducts() {
  var query = `query { 
    shop {
      products(first: 4) {
        edges {
          node {
            title
            images(first: 1) {
              edges {
                node {
                  src
                }
              }
            }
          }
        }
     variants(first: 1) {
        // 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

```javascript
function buyPowerUp(variantId) {
  var query = `mutation {
    checkoutCreate(input: {
      lineItems: [{
        }
      }) {
        checkout {
          }
        }
    }
  }
  return makeRequest(query);
}
```
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`.

```javascript
function buyPowerUp(variantId) {
  var query = `...

  return makeRequest(query);
}
```
Code review

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

```javascript
function buyPowerUp(variantId) {
    var query = `mutation {
        checkoutCreate(input: {
            lineItems: [
                {}
            ]
        }) {
            checkout {
            };
        }
    }
    return makeRequest(query);
}
```
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).

```
  mutation {
    checkoutCreate(input: {
      lineItems: [{
        lineItems: []
      ]
    }) {
      checkout {
      }
    }
  }
```
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.

```graphql
mutation {
  checkoutCreate(input: {
    lineItems: [{},
      lineItems: [{},
        checkout {
        }]
      ]
  }) {
    checkout
  }
}
```
Challenge

Navigate to


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

`lineItems` requires two arguments:

- `quantity`
- `variantId`
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 it in the `query` variable.
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!
The three fields are `webUrl`, `completedAt`, and `id`.
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!

MLH & Shopify Present...

Snake 2000

INSTRUCTIONS

Use the arrow keys to change direction.
Press SPACE to start and pause the game.

PRESS [SPACE] TO BEGIN

Score: 0

Game Store

Extra Life -100pts

Speed Boost -100pts
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.
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
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
- 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. ♡


http://developers.shopify.com
@ShopifyDevs
Thank you! Don’t be a stranger!
And don’t forget your socks 🧦

http://developers.shopify.com
@ShopifyDevs