INFO3139 Lab 10

Rev 1.0

General JavaScript Topic - Are arguments passed ByVal or ByRef?

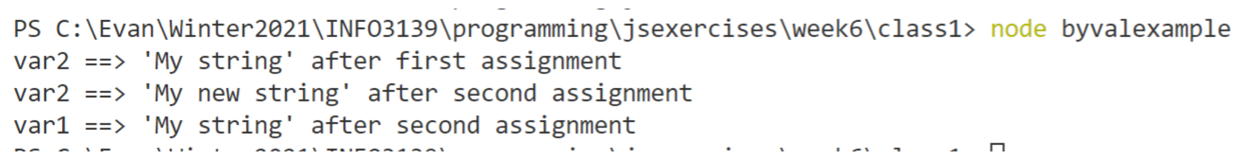
Understanding how JavaScript passes data is important to writing bug-free code. JavaScript passes **variables by value,** when the arguments are one of JavaScript's five primitive types (i.e., Boolean, null, undefined, String, and Number). However, when the variable is an Array, Function, or Object it will be passed **by reference**. Keep these assignments straight or they could cause problems if you're expecting Object assignment to work like a primitive assignment.

Here's a couple of examples to confirm this:

1. Create a week6\class1\byvalexample.js with the following code:

```
// primitives are passed ByVal
let var1 = 'My string';
let var2 = var1;
console.log(`var2 ==> '${var2}' after first assignment`);
var2 = 'My new string';
console.log(`var2 ==> '${var2}' after second assignment`);
console.log(`var1 ==> '${var1}' after second assignment`); // original data doesn't change
```

2. Execute the file:



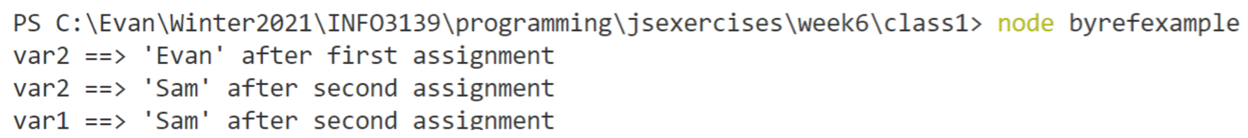
```
PS C:\Evan\Winter2021\INFO3139\programming\jsexercises\week6\class1> node byvalexample
var2 ==> 'My string' after first assignment
var2 ==> 'My new string' after second assignment
var1 ==> 'My string' after second assignment
```

Assigning a new value to the copied string has no effect on the original

3. Create another file called week6\class1\byrefexample.js with the following:

```
// objects are passed ByRef
let var1 = { name: 'Evan' }
let var2 = var1;
console.log(`var2 ==> '${var2.name}' after first assignment`);
var2.name = 'Sam';
console.log(`var2 ==> '${var2.name}' after second assignment`);
console.log(`var1 ==> '${var1.name}' after second assignment`); // original overwritten
```

4. Then execute it:



```
PS C:\Evan\Winter2021\INFO3139\programming\jsexercises\week6\class1> node byrefexample
var2 ==> 'Evan' after first assignment
var2 ==> 'Sam' after second assignment
var1 ==> 'Sam' after second assignment
```

Assigning a new value overwrote the original.

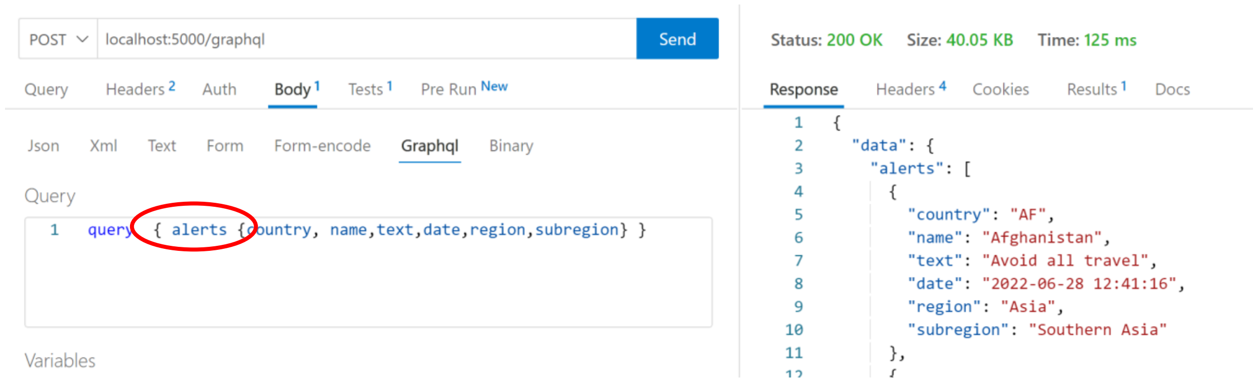
Case Study# 1 Work – cont'd

1. Once you have the setup working from lab 9 - part B, add an **Alert** type to project1\schema.js with these properties (derived from the alerts collection on Atlas):

```
type Alert {
  country: String
  name: String
  text: String
  date: String
  region: String
  subregion: String
}
```

2. Then, add a series of queries to the schema and a set of resolvers to satisfy the following:

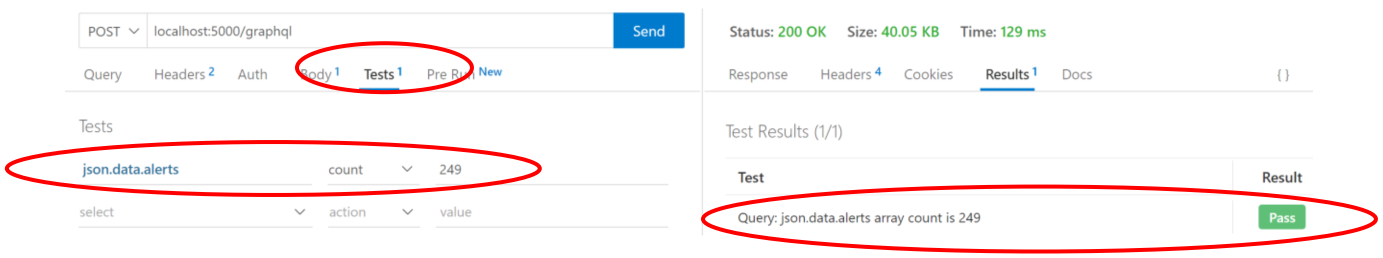
   a. Return **all alerts** (should return **249** alerts):



```
query { alerts {country, name,text,date,region,subregion} }
```

```
{
  "data": {
    "alerts": [
      {
        "country": "AF",
        "name": "Afghanistan",
        "text": "Avoid all travel",
        "date": "2022-06-28 12:41:16",
        "region": "Asia",
        "subregion": "Southern Asia"
      },
      {
```

To confirm there are **249** alerts set up a test like:

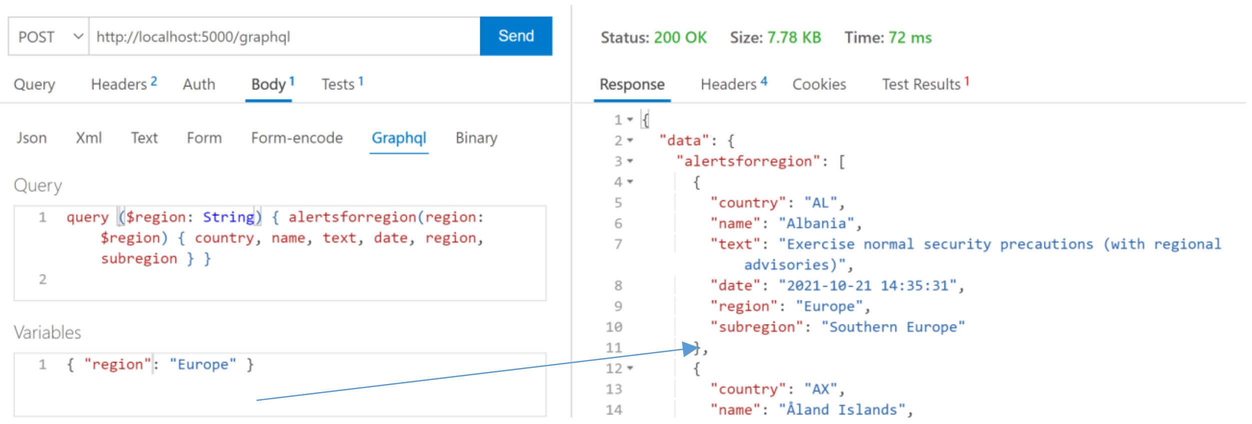


json.data.alerts	count	249

Test Results (1/1)

Test	Result
Query: json.data.alerts array count is 249	Pass

b. Return all **alerts for a region** e.g. (**Europe** would return **51** alerts):



POST http://localhost:5000/graphql Send

Query Headers 2 Auth Body 1 Tests 1

Json Xml Text Form Form-encode Graphql Binary

Query

```
query ($region: String) { alertsforregion(region:
    $region) { country, name, text, date, region,
    subregion } }
```

Variables

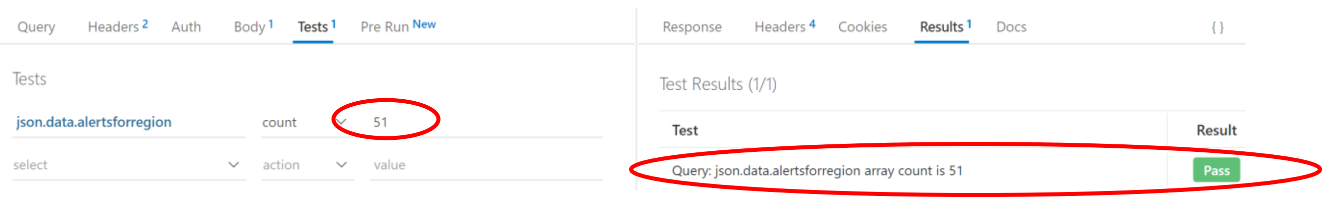
```
{ "region": "Europe" }
```

Status: 200 OK Size: 7.78 KB Time: 72 ms

Response Headers 4 Cookies Test Results 1

```
{
  "data": {
    "alertsforregion": [
      {
        "country": "AL",
        "name": "Albania",
        "text": "Exercise normal security precautions (with regional
            advisories)",
        "date": "2021-10-21 14:35:31",
        "region": "Europe",
        "subregion": "Southern Europe"
      },
      {
        "country": "AX",
        "name": "Åland Islands",
```

Confirm there are indeed **51** alerts by adding another test like the one above.



Query Headers 2 Auth Body 1 Tests 1 Pre Run New

Tests

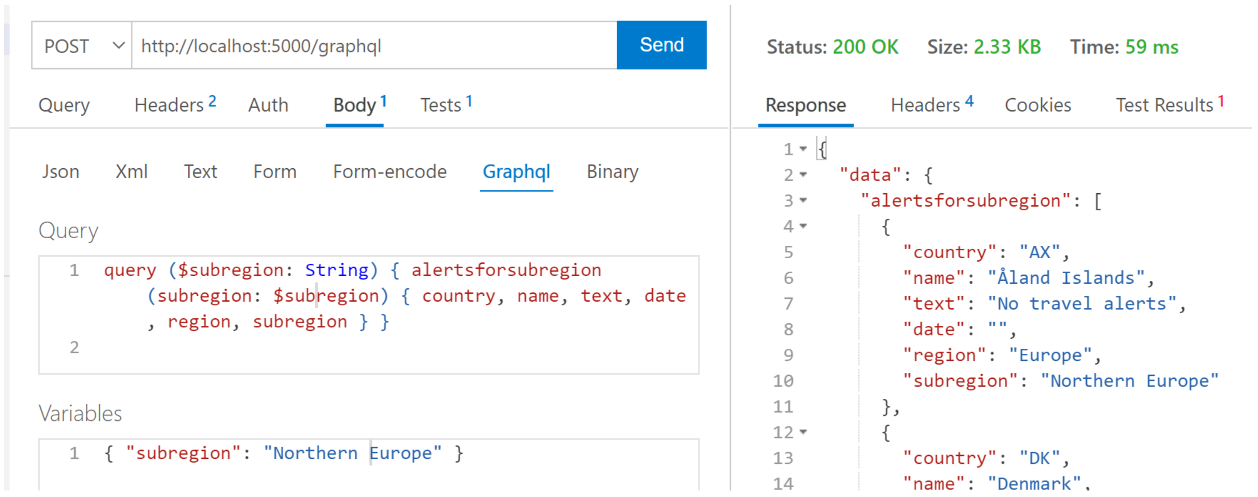
json.data.alertsforregion	count	51
select	action	value

Response Headers 4 Cookies Results 1 Docs { }

Test Results (1/1)

Test	Result
Query: json.data.alertsforregion array count is 51	Pass

c. Return all **alerts for a subregion** (e.g. **Northern Europe** would return **16** alerts)



POST http://localhost:5000/graphql Send

Query Headers 2 Auth Body 1 Tests 1

Json Xml Text Form Form-encode Graphql Binary

Query

```
query ($subregion: String) { alertsforsubregion
    (subregion: $subregion) { country, name, text, date
    , region, subregion } }
```

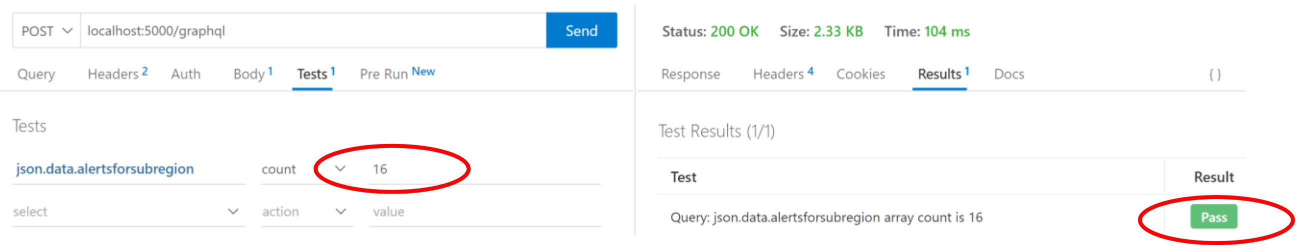
Variables

```
{ "subregion": "Northern Europe" }
```

Status: 200 OK Size: 2.33 KB Time: 59 ms

Response Headers 4 Cookies Test Results 1

```
{
  "data": {
    "alertsforsubregion": [
      {
        "country": "AX",
        "name": "Åland Islands",
        "text": "No travel alerts",
        "date": "",
        "region": "Europe",
        "subregion": "Northern Europe"
      },
      {
        "country": "DK",
        "name": "Denmark",
```


POST localhost:5000/graphql Send

Query Headers 2 Auth Body 1 Tests 1 Pre Run New

Tests

json.data.alertsforsubregion	count	16
select	action	value

Status: 200 OK Size: 2.33 KB Time: 104 ms

Response Headers 4 Cookies Results 1 Docs

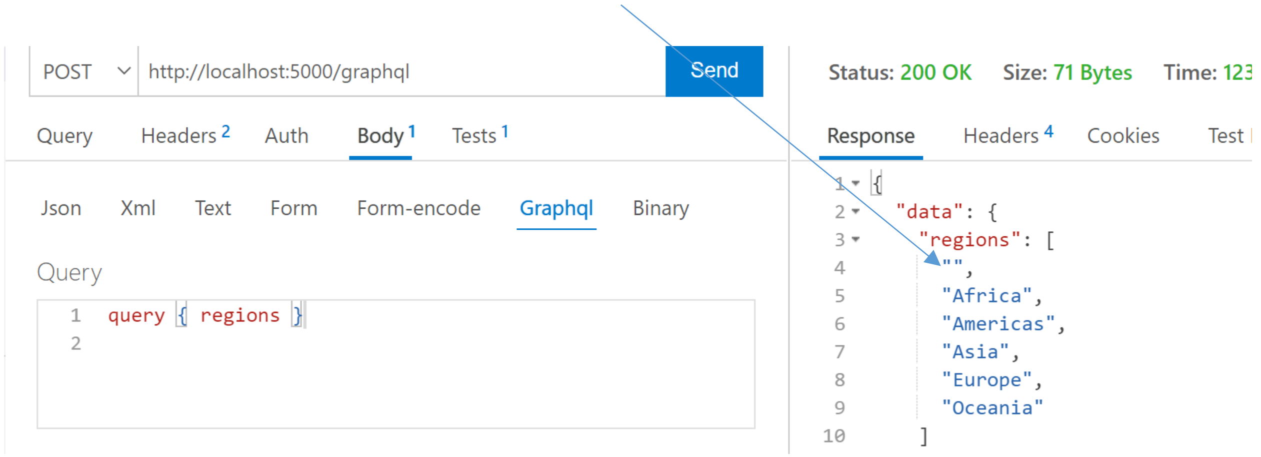
Test Results (1/1)

Test	Result
Query: json.data.alertsforsubregion array count is 16	Pass

For the next 2 queries, we can add a method to our db_routines that will search a collection and find **unique** values. This will find all unique regions and subregions in the collection. Add the following method to your db_routines module and don't forget to update the exports:

```
const findUniqueValues = (db, coll, field) => db.collection(coll).distinct(field);
```

d. Return all **6** regions (FYI - Antarctica does not have a region):


POST http://localhost:5000/graphql Send

Query Headers 2 Auth Body 1 Tests 1

Json Xml Text Form Form-encode Graphql Binary

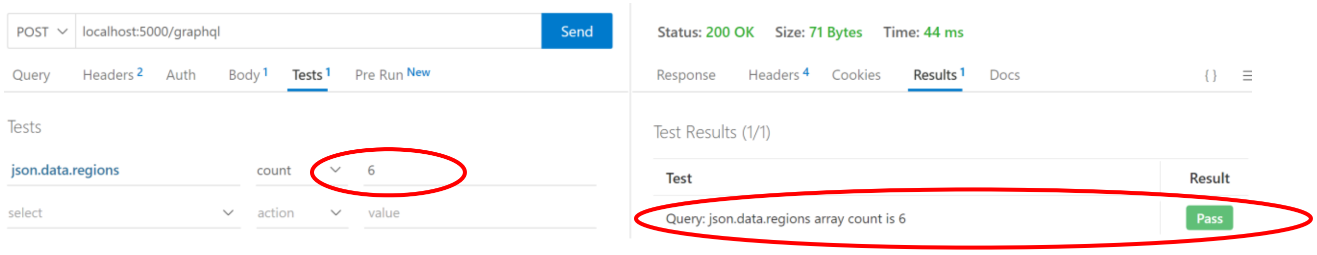
Query

```
query { regions }
```

Status: 200 OK Size: 71 Bytes Time: 123

Response Headers 4 Cookies Test

```
{
  "data": {
    "regions": [
      "",
      "Africa",
      "Americas",
      "Asia",
      "Europe",
      "Oceania"
    ]
```


POST localhost:5000/graphql Send

Query Headers 2 Auth Body 1 Tests 1 Pre Run New

Tests

json.data.regions	count	6
select	action	value

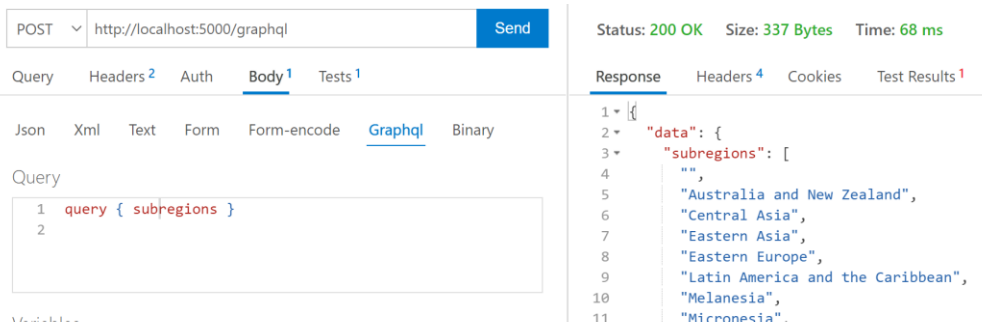
Status: 200 OK Size: 71 Bytes Time: 44 ms

Response Headers 4 Cookies Results 1 Docs

Test Results (1/1)

Test	Result
Query: json.data.regions array count is 6	Pass

e. Return all **18** subregions:


POST http://localhost:5000/graphql Send

Query Headers 2 Auth Body 1 Tests 1

Json Xml Text Form Form-encode Graphql Binary

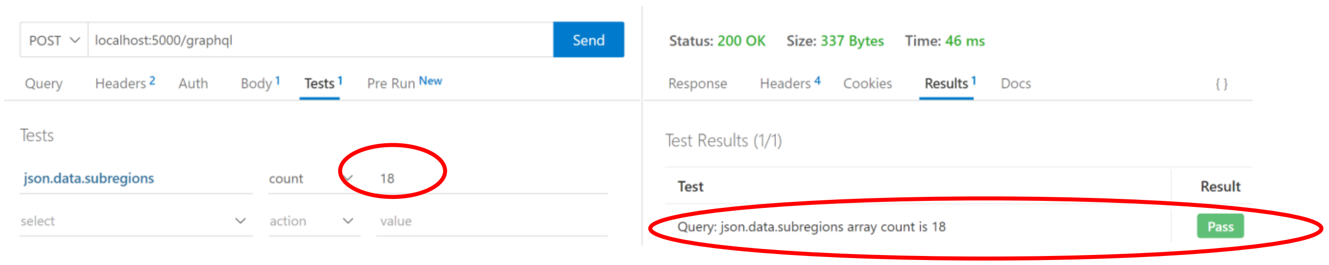
Query

```
query { subregions }
```

Status: 200 OK Size: 337 Bytes Time: 68 ms

Response Headers 4 Cookies Test Results 1

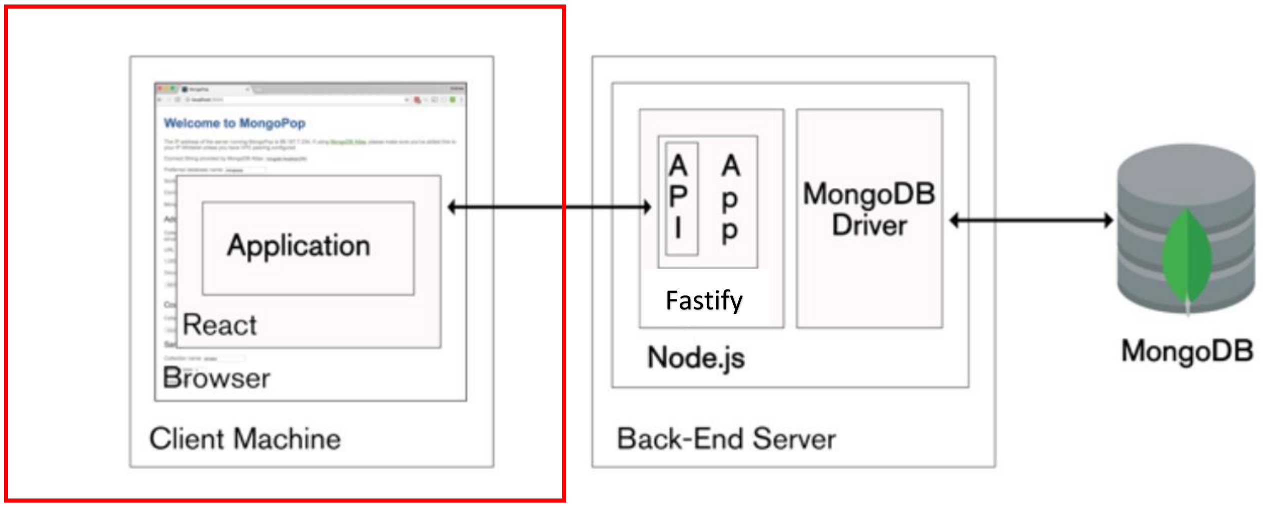
```
{
  "data": {
    "subregions": [
      "",
      "Australia and New Zealand",
      "Central Asia",
      "Eastern Asia",
      "Eastern Europe",
      "Latin America and the Caribbean",
      "Melanesia",
      "Micronesia",
```



```
POST    localhost:5000/graphql    Send
Query   Headers 2   Auth   Body 1   Tests 1   Pre Run New

Tests
json.data.subregions    count    18
select                  action   value

Status: 200 OK   Size: 337 Bytes   Time: 46 ms
Response   Headers 4   Cookies   Results 1   Docs

Test Results (1/1)
Test                                              Result
Query: json.data.subregions array count is 18     Pass
```

Moving Client Side – React.js

We are going to switch gears now, and look at the last part of our ME**R**N stack (technically we're using a MFRN stack where we've replaced Express with Fastify):



Everything we have done up to now has been what is considered server-side code. The main client technology we will utilize today was described in the reading from the homework from last class. We will be using a User Interface library called **React**. Fortunately, we only need to install a builder library called **Vite** that packages everything we need for developing applications in React.

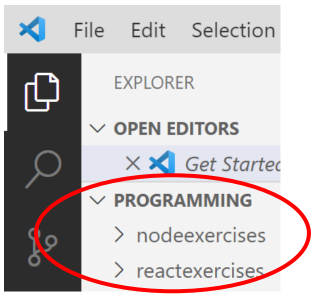
1. Do this in the terminal window, go **one level above** your current **nodeexercises** and issue the following command to **setup Vite and your first react project.**:

```
PS C:\Evan\Winter2023\info3139\programming> npm init vite
Need to install the following packages:
  create-vite@4.0.0
Ok to proceed? (y)
√ Project name: ... reactexercises
√ Select a framework: » React
√ Select a variant: » JavaScript
```

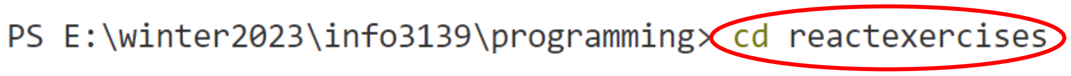
Make sure to enter these options

```
Scaffolding project in C:\Evan\Winter2023\info3139\programming\reactexercises...

Done. Now run:

  cd reactexercises
  npm install
  npm run dev
```

Again, this should result in a **reactexercises** folder being at the same level as the nodeexercises folder. Close the current folder VSCode and re-open the folder above the nodeexercises/reactexercises folders (mine was programming) as we'll be flipping between both projects for the remainder of the course.

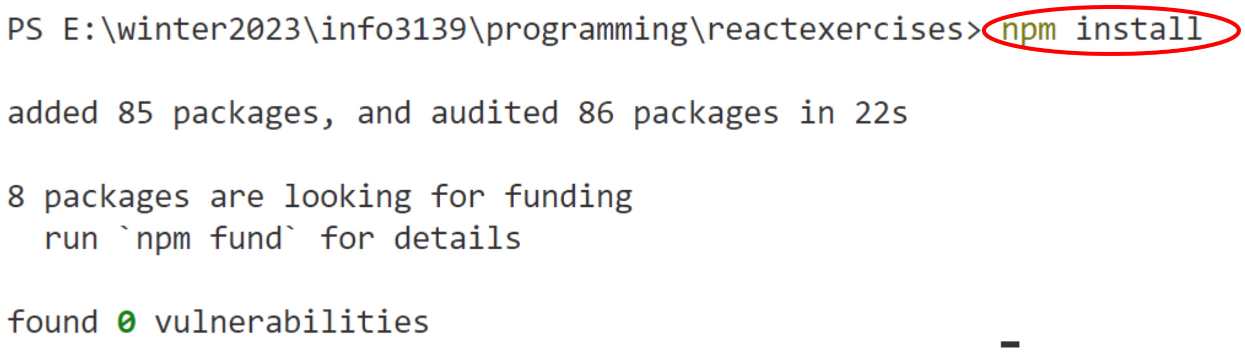


2. Open a new terminal window and change directories into the new reactexercises folder:



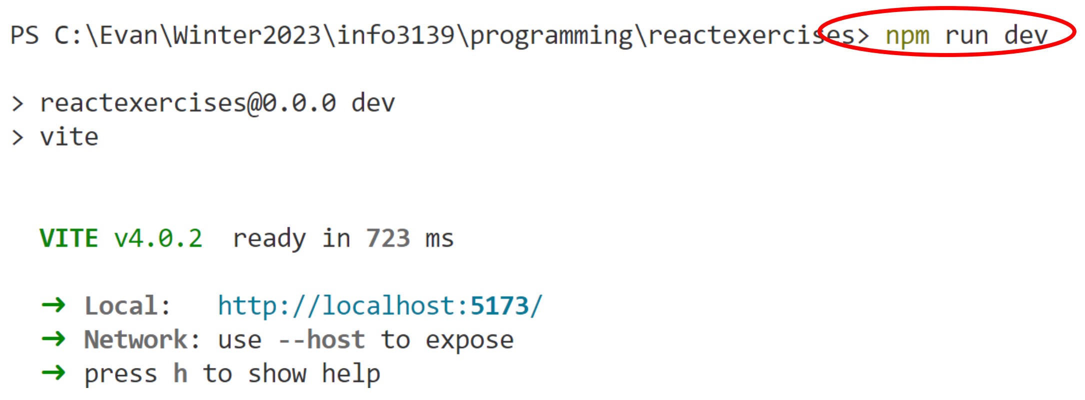
```
PS E:\winter2023\info3139\programming> cd reactexercises
```

3. Run the npm install command in your reactexercises project. Note this creates a brand new 50,000 file node_modules folder (so you'll end up with two, one for node in nodeexercises, and one for react in reactexercisess



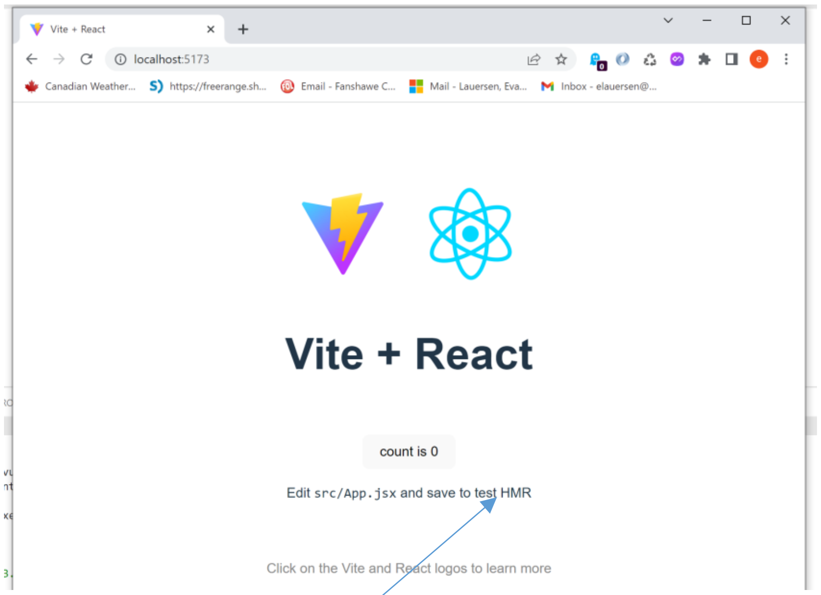
```
PS E:\winter2023\info3139\programming\reactexercises> npm install

added 85 packages, and audited 86 packages in 22s

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

4. Start the development server with **npm run dev**:



```
PS C:\Evan\Winter2023\info3139\programming\reactexercises> npm run dev

> reactexercises@0.0.0 dev
> vite


  VITE v4.0.2  ready in 723 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

5. Open a browser and point it to http://localhost:5173



Vite + React

count is 0

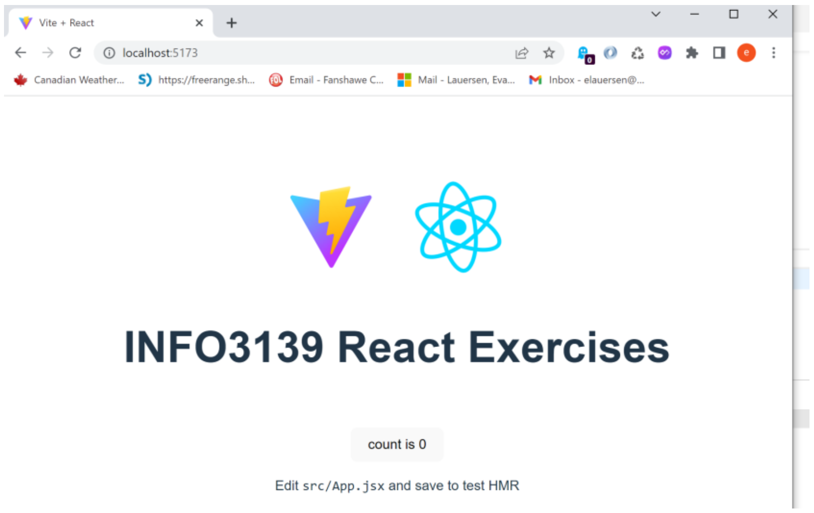
Edit `src/App.jsx` and save to test HMR

Click on the Vite and React logos to learn more

6. Test the **H**ot **M**odule **R**eplacement by making a small change to the **App.jsx** file. Change the heading html to something like the following, then save the change:

Was: `<h1>Vite + React</h1>`

Now: `<h1>INFO3139 React Exercises</h1>`

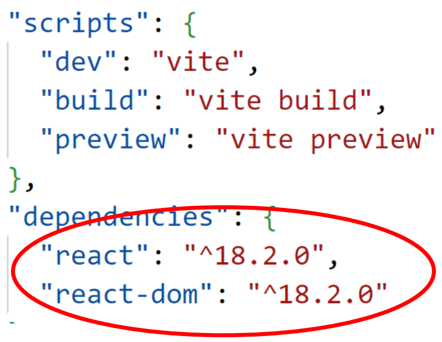


INFO3139 React Exercises

Edit `src/App.jsx` and save to test HMR

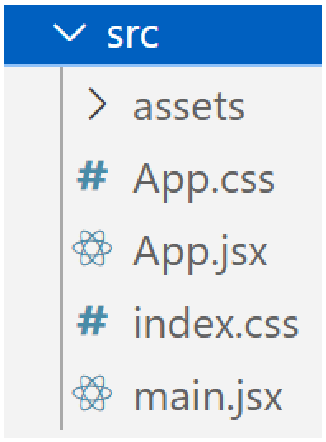
Summary of What Just Happened

- We have installed react **18.2.0** (at the time of this writing), looking in the package.json file we see just the react application's dependencies property:

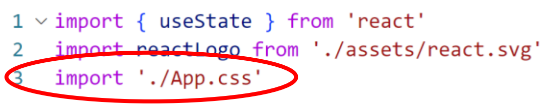


```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
},
"dependencies": {
  "react": "^18.2.0",
  "react-dom": "^18.2.0"
```

- The 2 scripts we are interested in, are **dev** and **build**
    - The **dev** script is what we just ran, it starts a development server on port 5173.
    - The **build** script is what we will run when we want to move the code over to our production environment, we will run this in an upcoming class (week 9)
- The actual application can be found in the **src** folder in the following files. The important files are noted below:



- **main.jsx**, this is the starting point for the, its main job is to render the react root component (<App />). It renders it in the html element `<div id="root"></div>` in the public**/index.html** file
- **App.jsx** – The main component written in a react specific sub language called **JSX**. JSX is a combination of JavaScript and XML and is one of the main building blocks when creating react components. We will be delving into setting up components quite extensively over the next couple of classes.
- **App.css** – this is the style sheet for the App component. Look inside **App.jsx** and see how it is referenced:

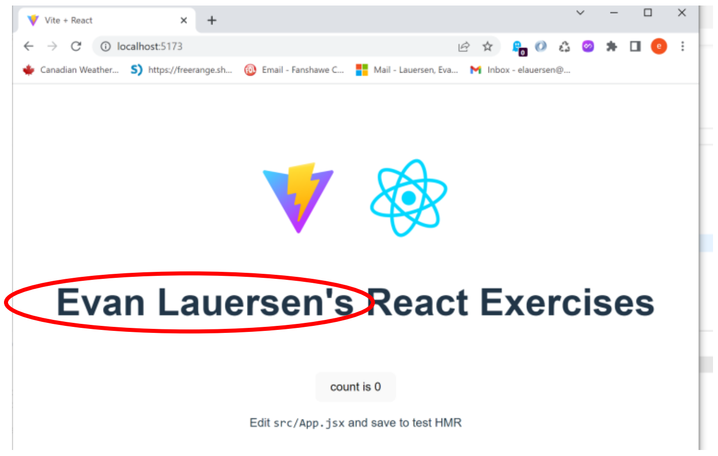


```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import './App.css'
```

Lab 10 - (2%)

Submit a single Word document containing 3 screen shots:

1. From the case study, return all alerts for the Region **Americas** along with proof (from a test) of how many alerts are in that region. Ensure that the Graph QL variables and query are visible along with the start of the results in the screen shot.
2. Return all alerts for the Sub-Region **Eastern Europe** with proof (from a test) of how many alerts are in that sub-region. Ensure that the Graph QL variables and query are visible along with the start of the results
3. Screen shot of the default React/Vite page, replace the <h1>..</h1> with **your name React Exercises** in the heading:



Evan Lauersen's React Exercises

Edit `src/App.jsx` and save to test HMR

Part B - (.5%) - Lab 10 Theory Quiz on FOL - 10 questions

Review

1. Which data types are passed ByRef in JavaScript? Which data types are passed ByVal in JavaScript?
2. If I have a query with a format of: { x {y,z} }, what does the y,z represent?
3. If I have a query with a format of:{ x(y:z) {a,b,c} }, what does the x represent?
4. What code would I enter if I wanted to add **a user** to our users collection in Atlas with GraphiQL?
5. What would be the format for entering the query from Q4 in Thunder Client?
6. If your query returns lots of data in an array, how can you confirm the number of elements returned?
7. How many queries should your case study schema now have?
8. What build tool are we using to facilitate the react framework installation?
9. How is the react development server started?
10. What is the single page of the single page application in the starter package?
11. What is the name of the starting file?
12. What is the relationship between index.html and main.jsx?

Homework

Look over these links to get a feel for the design framework we'll be working with in the client project:

- https://en.wikipedia.org/wiki/Material_Design
- http://www.material-ui.com/#/get-started/installation