MINAB
IT SOLUTIONS

Frontend development Class 05, Series 02

Frontend

Hahu Jobs

CLASS 05

Hahu Jobs

01 Router recap

02 Local storage

03 GraphQL

04 Class Exercise

Routing recap

Vue-Router

The officially-supported router library for **vue**

Simple Routing from Scratch

```
<script setup>
import { ref, computed } from 'vue'
import Home from './Home.vue'
import About from './About.vue'
import NotFound from './NotFound.vue'

const routes = {
  '/': Home,
  '/about': About
}

const currentPath = ref(window.location.hash)

window.addEventListener('hashchange', () => {
  currentPath.value = window.location.hash
})

const currentView = computed(() => {
  return routes[currentPath.value.slice(1) || '/'] || NotFound
})
</script>
```

Vue router

```
// These can be imported from other files
const Home = { template: '<div>Home</div>' }
const About = { template: '<div>About</div>' }

// 2. Define some routes
// Each route should map to a component.
// We'll talk about nested routes later.
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
]

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = VueRouter.createRouter({
  // 4. Provide the history implementation to use. We are using the hash
  history: VueRouter.createWebHashHistory(),
  routes, // short for `routes: routes`
})

// 5. Create and mount the root instance.
const app = Vue.createApp({})
// Make sure to _use_ the router instance to make the
// whole app router-aware.
app.use(router)

app.mount('#app')

// Now the app has started!
```

MINAB IT SOLUTIONS

HahuJobs

Routing recap

router-link

Note how instead of using regular a tags, we use a custom component router-link to create links. This allows Vue Router to change the URL without reloading the page, handle URL generation as well as its encoding. We will see later how to benefit from these features.

router-view

router-view will display the component that corresponds to the url. You can put it anywhere to adapt it to your layout.

Routing recap

Route matching

Matching the current URL with defined routes

```
const routes = [
    // home page
    { path: '/' },
    // about page
    { path: '/about' },
    // matches /o/3549
    { path: '/o/:orderId' },
    // matches /p/books
    { path: '/p/:productName' },
]
```

MINAB IT SOLUTIONS

HahuJobs

Routing recap

Nested Routes

Matching the current URL with defined routes

```
const routes = [
  {
    path: '/user/:id',
    component: User,
    children: [
      {
        // UserProfile will be rendered inside User's <router-view>
        // when /user/:id/profile is matched
        path: 'profile',
        component: UserProfile,
      },
      {
        // UserPosts will be rendered inside User's <router-view>
        // when /user/:id/posts is matched
        path: 'posts',
        component: UserPosts,
      },
    ],
  },
]
```

Routing recap

Nested Routes

The <router-view> here is a top-level router-view. It renders the component matched by a top level route. Similarly, a rendered component can also contain its own, nested <router-view>. For example, if we add one inside the User component's template:

```html
<div id="app">
  <router-view></router-view>
</div>
```

```js
const User = {
  template: `
    <div class="user">
      <h2>User {{ $route.params.id }}</h2>
      <router-view></router-view>
    </div>
  `,
}
```

```js
const routes = [
  {
    path: '/user/:id',
    component: User,
    children: [
      {
        // UserProfile will be rendered inside User's <router-view>
        // when /user/:id/profile is matched
        path: 'profile',
        component: UserProfile,
      },
      {
        // UserPosts will be rendered inside User's <router-view>
        // when /user/:id/posts is matched
        path: 'posts',
        component: UserPosts,
      },
    ],
  },
]
```

Local storage & Session storage

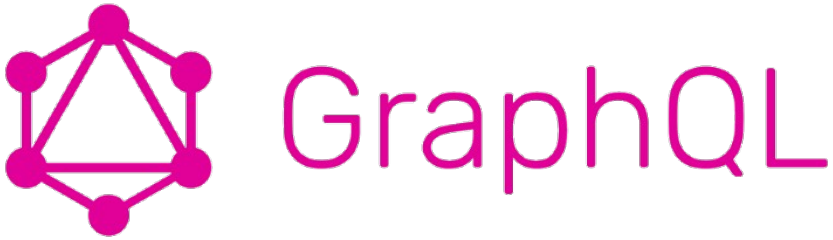
Web storage objects **localStorage** and **sessionStorage** allow to save key/value pairs in the browser.

What's interesting about them is that the data survives a page refresh (for sessionStorage) and even a full browser restart (for localStorage).

Both storage objects provide same methods and properties:

- setItem(key, value) – store key/value pair.
- getItem(key) – get the value by key.
- removeItem(key) – remove the key with its value.
- clear() – delete everything.
- key(index) – get the key on a given position.
- length – the number of stored items.

GraphQL



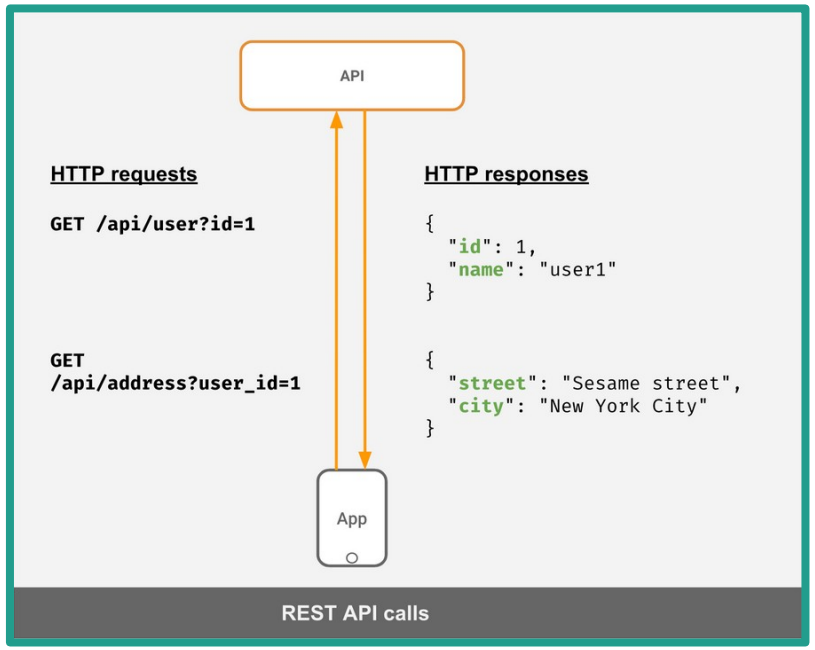
GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data. **GraphQL** isn't tied to any specific database or storage engine and is instead backed by your existing code and data.

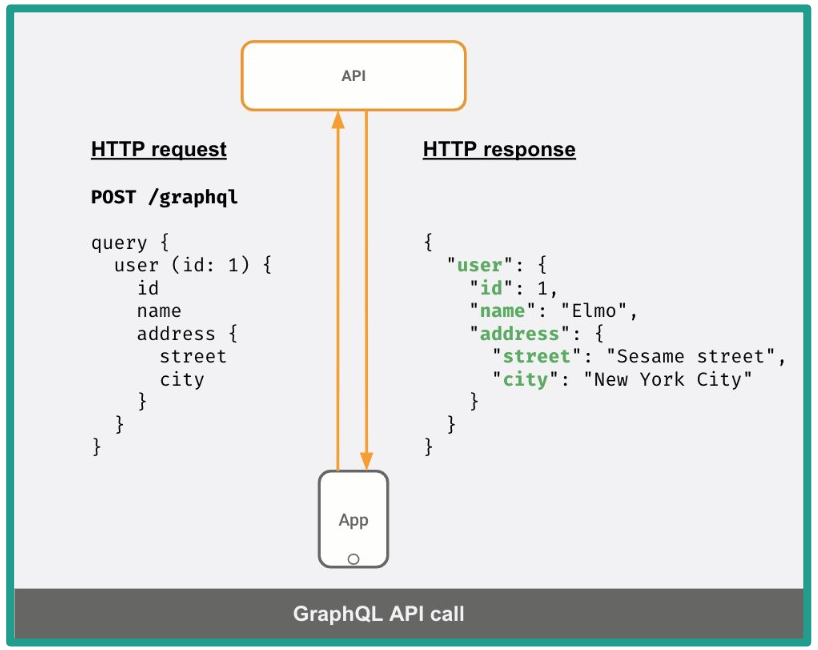
MINAB IT SOLUTIONS

HahuJobs

GraphQL vs REST

GraphQL is often touted as an alternative to REST APIs. In this section, we will look at the key differences between GraphQL and REST with an example and also look at how they both can co-exist and complement each other.





GraphQL operation

A GraphQL operation can be of type

- **query** (a read-only fetch)
- **mutation** (a write followed by fetch)
- **subscription** (a long-lived request that fetches data in response to source events.)

GraphQL Anatomy

Fields

A GraphQL field describes a discrete piece of information. This information could be simple or complex with relationships between data.

GraphQL Anatomy

Arguments

Imagine fields as functions that return values. Now let's assume the function also accepts arguments that behave differently.

```
query {
  author(limit: 5) {
    id
    name
    articles {
      id
      title
      content
    }
  }
}
```

MINAB IT SOLUTIONS

Hahu Jobs

GraphQL Anatomy

Variables

GraphQL queries can be parameterized with variables for reuse and easy construction of queries on the client-side.

```
query ($limit: Int) {
  author(limit: $limit) {
    id
    name
  }
}
```

```
{
  limit: 5
}
```

MINAB IT SOLUTIONS

Hahu Jobs

GraphQL Anatomy

Operation Name

When the document contains multiple operations, the server has to know which ones to execute and map the results back in the same order. For example:

```
query fetchAuthor {
  author(id: 1) {
    name
    profile_pic
  }
}
query fetchAuthors {
  author(limit: 5, order_by: { name: asc }) {
    id
    name
    profile_pic
  }
}
```

GraphQL Anatomy

Aliases

Consider the following example:

```
query fetchAuthor {
  author(id: 1) {
    name
    profile_pic_large: profile_pic(size: "large")
    profile_pic_small: profile_pic(size: "small")
  }
}
```

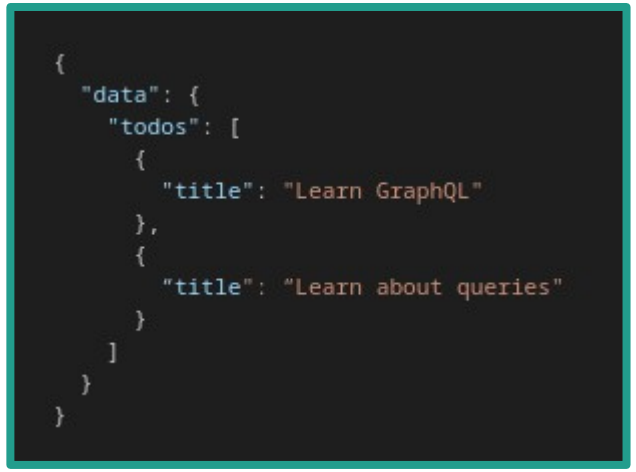
MINAB IT SOLUTIONS

HahuJobs

GraphQL Queries

Fetching data

. A query is a GraphQL Operation that allows you to retrieve specific data from the server.



MINAB IT SOLUTIONS

HahuJobs

GraphQL Mutations

Writing data

A Mutation is a GraphQL Operation that allows you to insert new data or modify the existing data on the server-side. You can think of GraphQL Mutations as the equivalent of POST, PUT, PATCH and DELETE requests in REST.

```
mutation {
  insert_todos(objects: [{ title: "New Todo" }]) {
    returning {
      id
      title
      is_completed
      is_public
      created_at
    }
  }
}
```

```
{
  "data": {
    "insert_todos": {
      "returning": [
        {
          "id": 55386,
          "title": "New Todo",
          "is_completed": false,
          "is_public": false,
          "created_at": "2022-02-24T10:26:30.718681+00:00"
        }
      ]
    }
  }
}
```

MINAB IT SOLUTIONS

HahuJobs

GraphQL Clients

A GraphQL request can be made using native JavaScript Fetch API. For example, to fetch a list of authors, we can make the query using the following code:

```
const limit = 5;
const query = `query author($limit: Int!) {
    author(limit: $limit) {
        id
        name
    }
}`;

fetch('/graphql', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
  body: JSON.stringify({
    query,
    variables: { limit },
  })
})
  .then(r => r.json())
  .then(data => console.log('data returned:', data));
```

GraphQL Clients

Why do I need a GraphQL Client?

Constructing query, processing response

A GraphQL client can help in constructing the full query with just the GraphQL document as input with relevant headers and context information. So instead of you writing the fetch API call everytime, the client will handle it for you giving the response data and error after parsing.

Managing UI State

GraphQL client is also useful to manage UI state and sync data across multiple UI components.

Updating cache

GraphQL client can also be used to manage cached entries of data fetched from queries or mutation. Reactive updates to UI as mentioned above is achieved using a cache.

GraphQL Clients

Set up a GraphQL client with Apollo

Vue Apollo Installation

Let's get started by installing apollo client & peer graphql dependencies:

```
$ npm install --save @apollo/client @vue/apollo-composable @vue/apollo-util graphql graphql-tag
```

Copy

MINAB IT SOLUTIONS

HahuJobs

GraphQL Clients

Set up a GraphQL client with Apollo

Create Apollo Client Instance

```
import { ApolloClient, InMemoryCache, HttpLink } from "@apollo/client/core"
import { getMainDefinition } from "@apollo/client/utilities"
import { onError } from "@apollo/client/link/error"
import { logErrorMessages } from "@vue/apollo-util"

function getHeaders() {
    const headers = {}
    const token = window.localStorage.getItem("apollo-token")
    if (token) {
        headers["Authorization"] = `Bearer ${token}`
    }
    return headers
}

// Create an http link:
const httpLink = new HttpLink({
    uri: "https://hasura.io/learn/graphql",
    fetch: (uri: RequestInfo, options: RequestInit) => {
        options.headers = getHeaders()
        return fetch(uri, options)
    },
})

const errorLink = onError((error) => {
    if (process.env.NODE_ENV !== "production") {
        logErrorMessages(error)
    }
})

// Create the apollo client
export const apolloClient = new ApolloClient({
    cache: new InMemoryCache(),
    link: errorLink.concat(httpLink),
})
```

Copy

MINAB IT SOLUTIONS

HahuJobs

GraphQL Clients

Set up a GraphQL client with Apollo

Install Vue Apollo Plugin and Provider

Now let's install the VueApollo plugin into Vue. Then, let's provide an ApolloClient instance. This is the instance that can then be used by all the child components.

```
import App from "./App.vue"
import { router } from "./router"
- import { createApp } from "vue"
+ import { createApp, provide, h } from "vue"

+ import { DefaultApolloClient } from "@vue/apollo-composable"
+ import { apolloClient } from "./apollo-client"

import authPlugin from "./auth/authPlugin"

- const app = createApp(App)
+ const app = createApp({
+     setup() {
+         provide(DefaultApolloClient, apolloClient)
+     },
+     render: () => h(App),
+ })

app.use(authPlugin)
app.use(router)
app.mount("#app")
```

Class Exercise

Build countries list

- show list of countries
    - displayed information should include
        - name
        - flag
        - continent
        - country code
        - phone code
        - capital
        - currency
        - languages spoken
        - it should include list of states
- design with figma
- GraphQL Server:

https://countries.trevorblades.com/

MINAB IT SOLUTIONS

HahuJobs

MINAB
IT SOLUTIONS

Thank you!

Hahu Jobs

ለሀገር ልጅ በሀገር ልጅ !

Michael Sahlu
michael.sahlu@hahu.jobs