Lab 7 - Vue CLI & Ionic Vue

`COMP7980 - Web and Mobile Programming - HKBU - Fall2022`

We will continue to work on the Vue CLI project. Please download the **Express backend application** and also **clone your Vue CLI frontend project** with `VSCode`.

Run the two projects with commands:

```
npm start
```

and

```
npm run serve
```

Vue CLI - Displaying a Single Booking

To display a single booking in a Vue CLI app, we should first set up a new route in `router/index.js`:

```
{
    path: '/booking/:id/',
    name: 'booking',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */
'../views/BookingView.vue')
},
```

Similar to an Express route handler, we set up the **dynamic path parameter** with `:id`. Next, we are going to develop the new file `BookingView.vue` under `/views`. We can copy the codes from `HomeView.vue` and start with it.

Under `export default`, rename this component to `BookingView`. To obtain the **dynamic path parameter**, we have to import `useRoute` from `vue-router`:

```
import { useRoute } from "vue-router";
```

Then, develop the `setup()` function as follows:

```
setup() {
    const booking = ref({});
    const route = useRoute();

    onMounted(async () => {

        var response = await fetch("/api/bookings/" + route.params.id);

        if (response.ok) {
            booking.value = await response.json();
        } else {
            alert(response.statusText);
        }
    })

    return {
        booking
    }
}
```

This page is expected to display a single booking, thus `booking` is set up to hold a single JavaScript object: `ref({})`. We obtain a `route` object via `useRoute()`, and the **path parameters** are under `route.params`. In the `onMounted` lifecycle hook, we access the incoming `id` via `route.params.id`.

Bootstrap List Group

We will display the content of this booking with a **Bootstrap List Group** (https://getbootstrap.com/docs/5.0/components/list-group/). Simply use the first template and develop some data bindings:

```
<div class="container">
    <ul class="list-group">
        <li class="list-group-item">{{ booking._id }}</li>
        <li class="list-group-item">{{ booking.email }}</li>
        <li class="list-group-item">A third item</li>
        <li class="list-group-item">A fourth item</li>
        <li class="list-group-item">And a fifth one</li>
    </ul>
</div>
```

Router-Link

Go back to our `PaginatedBookings.vue` and develop the a new table cell:

```
<td>
    <router-link :to="`/booking/${booking._id}`">Details</router-link>
</td>
```

Inside a Vue CLI app, the links are developed with `router-link` elements. Notice the use the **template literal** in the `v-bind:to` directive. Now, the link should direct us to the booking detail page.

We haven't imported the **Bootstrap CSS** to our Vue CLI app. Head to the **terminal** and run

```
npm install bootstrap --save
```

Then, visit `main.js` and develop the following above the `createApp()`:

```
import "bootstrap/dist/css/bootstrap.min.css"
```

Reload the page, and you should see the Bootstrap list group.

Interactive Charts with ApexChart

We will cover quite a number of chart libraries in this course. We will start with *ApexChart*, which has the best integration with modern frontend frameworks like Vue (https://apexcharts.com/docs/vue-charts/).

Make sure we run the following lines in the terminal to install the ApexChart for Vue3.

```
npm install --save apexcharts
npm install --save vue3-apexcharts
```

Then, import *ApexCharts* globally in `main.js`:

```
import VueApexCharts from "vue3-apexcharts";

createApp(App).use(router).use(VueApexCharts).mount('#app')
```

We are going to develop a new page to display a donut chart. This page should be reached via the `/chart` route. Let's develop this route `/chart` in `router/index.js` as follows:

```
{
    path: '/chart',
    name: 'chart',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */
'../views/ChartView.vue')
}
```

Create a new file `views/ChartView.vue`, and we can again copy the codes from `HomeView.vue`.

We will follow the **Donut chart** example on https://apexcharts.com/docs/vue-charts/. However, the code is written in **Options API**, and we want to stick with **Composition API**, so some slight changes are needed.

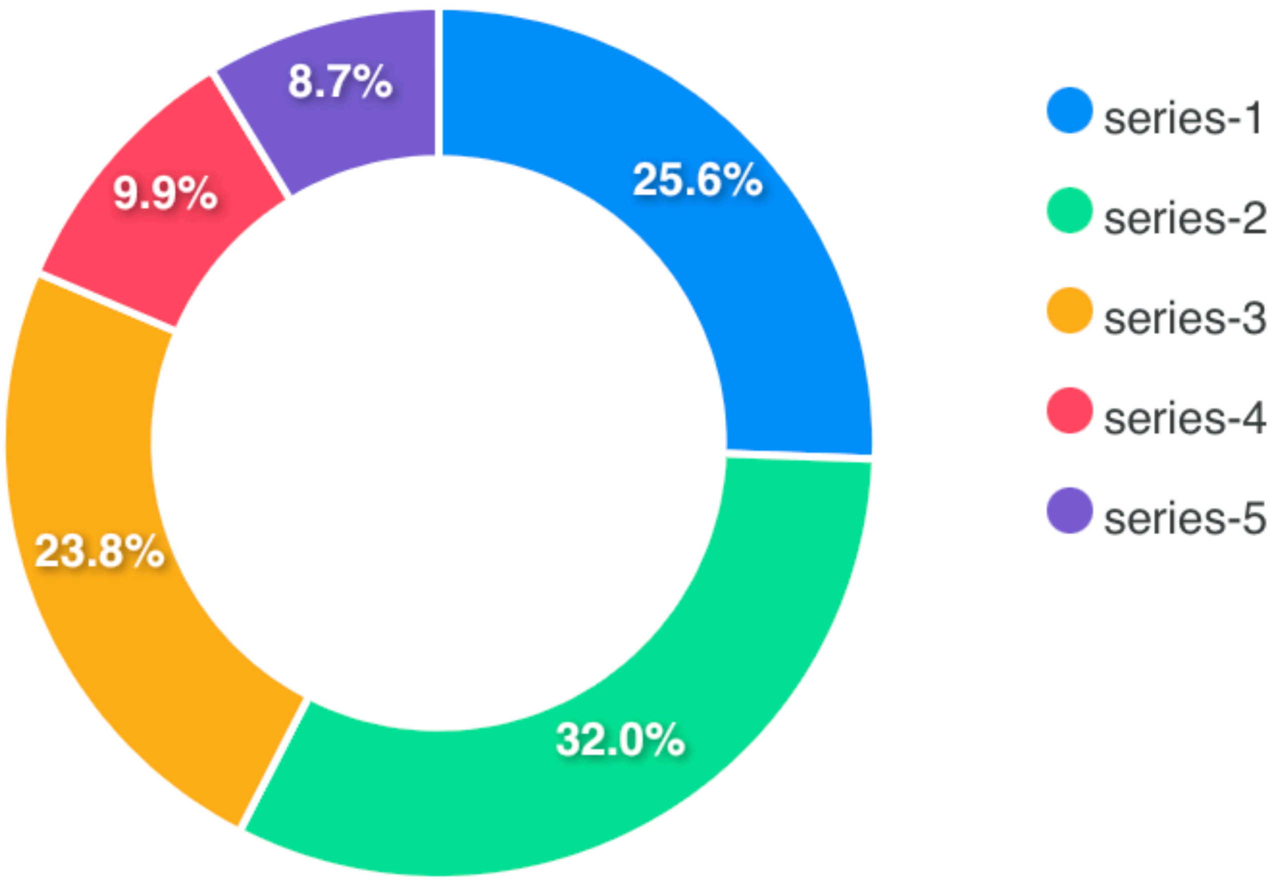
First, the vue template could be simplified as follows:

```
<template>
    <div>
        <apexchart width="380" type="donut" :options="options"
:series="series"></apexchart>
    </div>
</template>
```

This implied that instances properties `options` and `series` should be provided.

```
<script>
// @ is an alias to /src
import { ref } from "vue";

export default {
    name: 'ChartView',
    setup() {
        const options = ref({});
        const series = ref([44, 55, 41, 17, 15]);

        return {
            options, series
        }
    }
}
</script>
```

Visit http://localhost:8080/chart and you should see the following:



Aggregate in MongoDB

We want to show the **superhero popularities** with this chart, and this information could be easily provided by a MongoDB operation.

Let's visit our express application and develop a new route handler:

```
// GroupBy
router.get('/api/bookings/aggregate/groupby', async function (req, res) {

  const pipeline = [
    { $match: { payment: "Paypal" } },
    { $group: { _id: "$superhero", count: { $sum: 1 } } }
  ];

  const results = await
db.collection("bookings").aggregate(pipeline).toArray();

  return res.json(results);

});
```

MongoDB provides the `aggregate()` method to perform some more complicated operations. The `aggregate()` method takes a **pipeline** of a number of stages.

Here, `$match` stage filters documents based `payment`. The second stage makes use of `$group` to group the remaining documents based on `superhero`. The use of `$sum` would calculate the total popularity of each `superhero`.

The `aggregate()` method returns an `AggregationCursor`, which accesses each `superhero` as a time. We chain it with `toArray()` to generate **an array of superheroes with the counts**. This is a sample output:

```
[
  {
    "_id": "Captain America",
    "count": 137
  },
  {
    "_id": "superman",
    "count": 134
  },
  {
    "_id": "Ironman",
    "count": 112
  },
  {
    "_id": "batman",
    "count": 128
  },
  {
    "_id": "others",
    "count": 7
  }
]
```

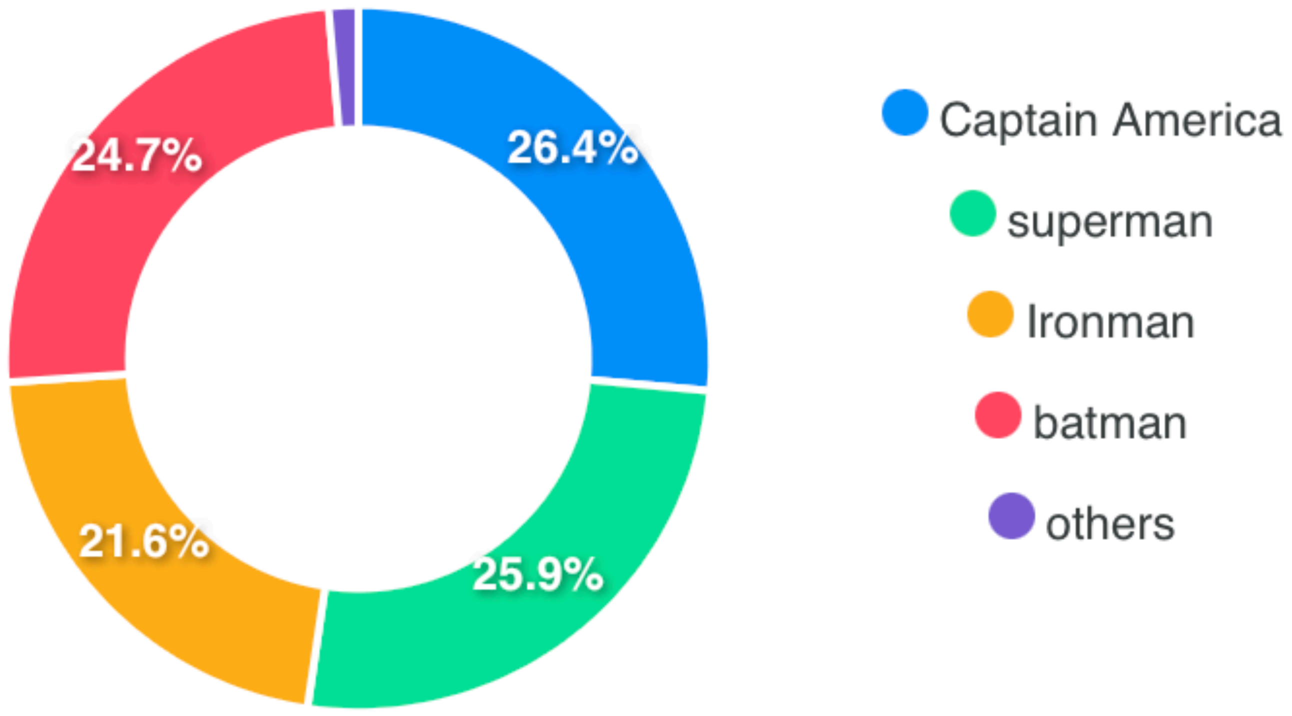
Restart the Express application, this data could be reached via
http://localhost:3000/api/bookings/aggregate/groupby

Consuming this API

Back to the `ChartView.vue`, develop the following `onMounted` to consume this API.

```
onMounted(async () => {

    var response = await fetch("/api/bookings/aggregate/groupby");

    if (response.ok) {
        var heroes = await response.json();

        series.value = heroes.map(a => a.count);
        options.value = { labels: heroes.map(a => a._id) };
    }
});
```

Here, the `map()` array method can help extract a particular property from each hero object.



Clear up

Currently, the **filter** and **groupBy** values are hardcoded. In assignment 2, you are required to make them dynamic.

Push your work to the remote GitHub repo.

To **deploy** this Vue CLI project, we should run the following command in the terminal

```
npm run build
```

This will generate a `/dist` folder which holds a single HTML file. Most of our work have been moved to inside the `/js` and `/css` folders.

Rapid App Development with Ionic Vue.

There're many popular **hybrid mobile development frameworks**, like Ionic, ReactNative, NativeScript etc. These hybrid frameworks usually allow developers to make use of **web technologies (HTML5, JavaScript)** in app development. Usually, the same codebase could be published on **both iOS** and **Android** platforms.

Today, we will use the **Ionic** framework, which becomes quite popular nowadays, due to its simplicity and flat learning curve. Ionic framework can work with different frontend technologies like **Angular** and **Vue**. As compared to Angular, Vue is more lightweight and got better support from the Ionic community.

The official website of the Ionic-Vue framework could be found on https://ionicframework.com/docs/vue/overview.

We are going to develop a simple **shopping app** using this **Ionic Vue** framework.

Let's Get Started

In the terminal, run the following command to set up a new **ionic-vue** project:

```
npx @ionic/cli start shoppingApp tabs --type vue

npx @ionic/cli serve
```

You should see an application with **3 tabs created** on a browser. Then, open the `shoppingApp` folder with `VSCode`. Notice that the project structure is really similar to a standard Vue CLI app with router installed. Let's develop the correct tab names `Ladies`, `Gents`, `My Cart` in `TabsPage.vue`. We can also pick other tab icons from https://ionic.io/ionicons.

Component

All tabs now includes the `ExploreContainer.vue` component. Let's develop an **instance property** to store the fashion items:

```
setup() {
    const items = ref([]);

    return {
      items
    };
}
```

Remove `lang="ts"` from the `<script>` tag, as we are writing JavaScript instead of TypeScript.

The fashion items are available online at https://api.npoint.io/5529943ab6c290922ca9. Thus, let's develop the `onMounted()` lifecycle hook:

```
onMounted(async() => {

  var response = await
fetch("https://api.npoint.io/5529943ab6c290922ca9");

  if (response.ok) {
    items.value = await response.json()
  }
});
```

To display the `items`, let's first remove `<div>` element, and bring in the following an **Ionic Card** as shown on https://ionicframework.com/docs/api/card:

```
<template>
  <!-- <div id="container">
    <strong>{{ name }}</strong>
    <p>Explore <a target="_blank" rel="noopener noreferrer"
href="https://ionicframework.com/docs/components">UI Components</a></p>
  </div> -->
  <ion-card>
    <img alt="Silhouette of mountains"
src="https://ionicframework.com/docs/img/demos/card-media.png" />
    <ion-card-header>
      <ion-card-title>Card Title</ion-card-title>
      <ion-card-subtitle>Card Subtitle</ion-card-subtitle>
    </ion-card-header>

    <ion-card-content>
      Here's a small text description for the card content. Nothing more,
nothing less.
    </ion-card-content>
  </ion-card>
</template>
```

Then, further modify the card with the introduction of the `v-for` directive and some **data bindings**.

```
<ion-card v-for="item in items" :key="item.id">
    <img alt="Silhouette of mountains" :src="item.image" />
    <ion-card-header>
      <ion-card-title>{{item.name}}</ion-card-title>
      <!-- <ion-card-subtitle>Card Subtitle</ion-card-subtitle> -->
    </ion-card-header>

    <ion-card-content>
      {{item.desc}}
    </ion-card-content>
</ion-card>
```

Finally, modify the `import` statement as follows:

```
import { defineComponent, ref, onMounted } from 'vue';
```

Four cards will then be shown.

Ladies and Gentlemen.

We want to show the **ladies** items on the first tab, and the **gents** ones on the second. To achieve this, we have to perform **filtering** on the `items` array. Please also note that the `ExploreContainer.vue` contains a `props`. `Props` are **custom attributes** you can **register on a component**. When a value is passed to a `prop` attribute, **it becomes a property on that component instance**. We can make use of this prop to perform the filtering.

In the `onMounted()` lifecycle hook, modify data handling as follows:

```
onMounted(async () => {

  var response = await
fetch("https://api.npoint.io/5529943ab6c290922ca9");

  if (response.ok) {
    let fashionItems = await response.json();

    items.value = fashionItems.filter(function (item) {
      return item.department == props.name;
    })
  }

});
```

Lines 8 - 10 demonstrate a **simple filtering block**. Those fashion items that satisfy the given condition (for example, `department == "Ladies"`) will be returned and populated to `items`.

We have to provide `props` as the first argument of `setup()`, so `props.name` could be accessible:

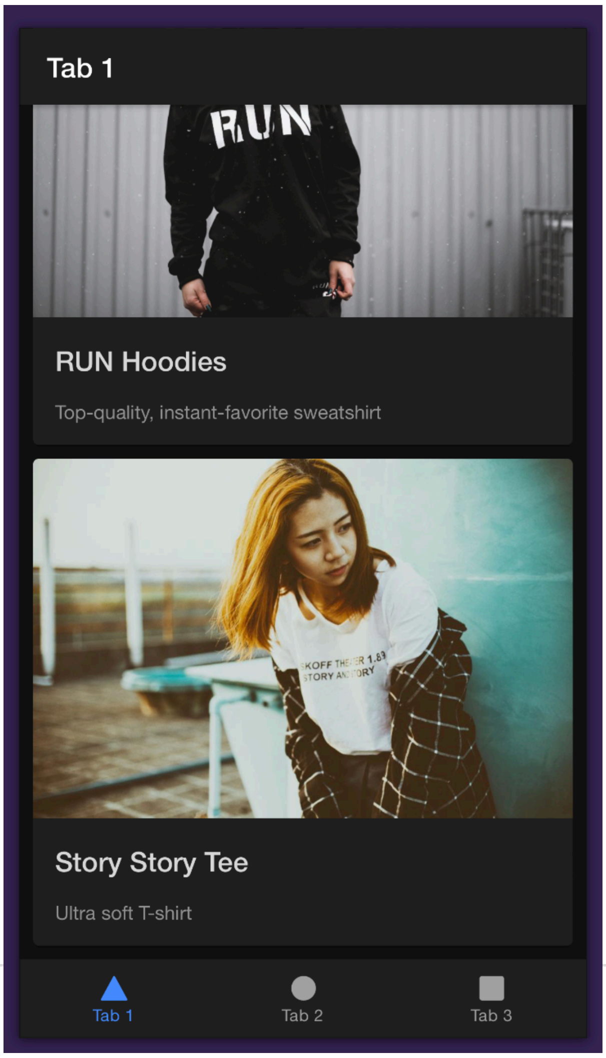
```
setup(props)
```

Finally, go to `Tab1Page.vue` and `Tab2Page.vue` to provide the correct department names:

```
<ExploreContainer name="Ladies" />
```

and

```
<ExploreContainer name="Gents" />
```



Item Detail Page

To develop a new page, a new `vue` file should be created. Now, let's create a new file `ItemPage.vue`, which is expected to show the **item details**.

Let's copy the code from `Tab3Page.vue` and rename the component name to `ItemPage`. Then, develop the `setup()` as follows:

```
setup() {
    const item = ref({});

    onMounted(async () => {

        var response = await
fetch("https://api.npoint.io/5529943ab6c290922ca9");

```

```
        if (response.ok) {
            let fashionItems = await response.json();

            item.value = fashionItems.filter(function (item) {
                return item.id == "1";
            })[0];
        }
    });

    return {
        item
    }
}
```

Display an Ionic card to show the details:

```
<!-- <ExploreContainer name="Tab 3 page" /> -->
<ion-card>
    <img alt="Silhouette of mountains" :src="item.image" />
    <ion-card-header>
        <ion-card-title>{{ item.name }}</ion-card-title>
        <ion-card-subtitle>{{ item.price }}</ion-card-subtitle>
    </ion-card-header>

    <ion-card-content>
        {{item.desc}}
    </ion-card-content>

    <ion-button @click="addToCart()">Add To Cart/ion-button>
    <ion-button @click="removeFromCart()">Remove From Cart</ion-button>
</ion-card>
```

We will come back to further work on this page. Before that, let's first make sure this page could be reached when an item is being tapped in the first two tabs.

Routing and Navigation

Let's visit `router/index.ts` and develop one more route:

```
{
    path: '/item/:id',
    component: () => import('@/views/ItemPage.vue')
}
```

In `ExploreContainer.vue`, import `useRouter`:

```
import { useRouter } from 'vue-router';
```

Then, develop an instance method `navigateWithId`:

```
const router = useRouter();

const navigateWithId = function (id) {
  router.push('/item/' + id)
};
```

To **navigate to** the item detail page, let's develop an `@click()` directive to the `ionic-card`:

```
@click="navigateWithId(item.id)"
```

Page navigation is triggered by `router.push()`, the `id` of the selected item is passed along.

Test this work, make sure the item page could be reached.

Receiver End

In `ItemPage.vue`, also set up the `router`:

```
import { useRoute } from 'vue-router';
```

Inside `setup()`, obtain the incoming `id` via:

```
const route = useRoute();
const { id } = route.params;
```

Finally, use this `id` in the filter function:

```
item.value = fashionItems.filter((item) => {
    return item.id == id;
})[0];
```

Now, the correct item should be displayed whenever we selected an item from the first two tabs.

Shopping Cart

We want to set up a shopping cart to hold the selected items. Head to `App.vue` and develop the following as an option of the `defineComponent()`

```
setup() {
    provide('cart', ref([]))
}
```

Modify the `import` statement to bring in `ref` and `provide`:

```
import { defineComponent, ref, provide } from 'vue';
```

In `ItemPage.vue`, access this shopping cart via:

```
const cartItems = ref(inject('cart').value);
```

Then, set up a **computed property** which tells whether the fashion item has been selected:

```
const selected = computed(() => {

    return cartItems.value.includes(id);
});
```

Then, the following two instance methods will flip the selection:

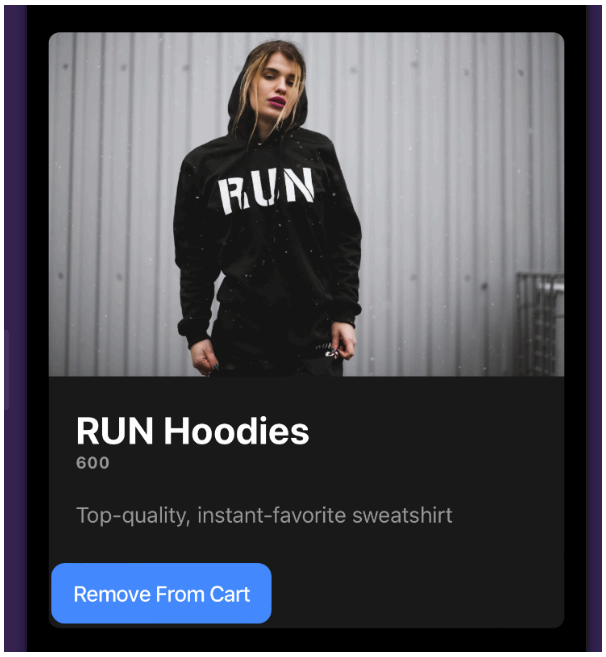
```
const addToCart = function () {
    cartItems.value.push(id);
}

const removeFromCart = function () {

    const index = cartItems.value.indexOf(id);
    if (index > -1) { // only splice array when item is found
        cartItems.value.splice(index, 1); // 2nd parameter means remove
one item only
    }
}
```

The import statement is now as follows:

```
import { defineComponent, onMounted, ref, inject, computed } from 'vue';
```

Finally, use `v-if` to control which button to be displayed:

```
<ion-button v-if="!selected" @click="addToCart()">Add To Cart</ion-button>
<ion-button v-if="selected" @click="removeFromCart()">Remove From
Cart</ion-button>
```


RUN Hoodies
600
Top-quality, instant-favorite sweatshirt

My Shopping Cart

Time to work on the third tab, which shows the current items in the **shopping cart**. We will use the list template on https://ionicframework.com/docs/api/card#list-card. Keep one `ion-item` element and delete the others. Also, remove all references of `ExploreContainer` from this file.

We have to download the fashion items again with the following block of codes:

```
const items = ref([]);

onMounted(async () => {

  var response = await
fetch("https://api.npoint.io/5529943ab6c290922ca9");

```

```
  if (response.ok) {
        items.value = await response.json();
  }
});
```

We then obtain the id of the shopping cart items via `cartItems`. Then, we develop a computed property `selectedItems` which holds the full details of the fashion items that have been selected.

```
const cartItems = ref(inject('cart').value);

const selectedItems = computed(() => {

  return items.value.filter((item) => {
        return cartItems.value.includes(item.id)
  })

});
```

The following computed property calculates the the total price of the selected items.

```
const totalPrice = computed(() => {

  var price = 0;

  items.value.forEach((item) => {
        if (cartItems.value.includes(item.id)) {
          price += item.price
        }
  })

  return price;

});
```

Our `setup()` function should return the two computed properties:

```
return {
  selectedItems, totalPrice
}
```

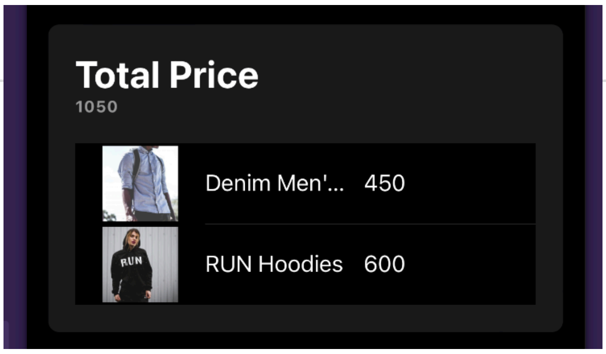
These values are displayed with the following template:

```
<ion-card>
    <ion-card-header>
      <ion-card-title>Total Price</ion-card-title>
      <ion-card-subtitle>{{totalPrice}}</ion-card-subtitle>
    </ion-card-header>
    <ion-card-content>
      <ion-list>
        <ion-item v-for="item in selectedItems" :key="item.id">
          <ion-thumbnail slot="start">
            <img alt="Silhouette of mountains" :src="item.image" />
          </ion-thumbnail>
          <ion-label>{{item.name}}</ion-label>
          <ion-label>{{item.price}}</ion-label>
        </ion-item>
      </ion-list>
    </ion-card-content>
</ion-card>
```

We have to import the following from `vue`:

```
import { defineComponent, ref, inject, computed, onMounted } from 'vue';
```

Submission



Total Price
1050

Denim Men'... 450
RUN Hoodies 600

There's another **invitation link** on our course room. Please claim it and set up a new **GitHub repo**. Push the **Ionic-Vue app** to this repo.

Please **zip your express project** and submit it to BUmoodle.

20/10/2022 08:22