SER421: Web-based Programming   Vue3 Lab

The goal of this lab is to get you working in the Vue framework. Keep in mind that we are using the Vue framework version 3 with the Options API ONLY. PLEASE FOLLOW SUBMISSION INSTRUCTIONS!

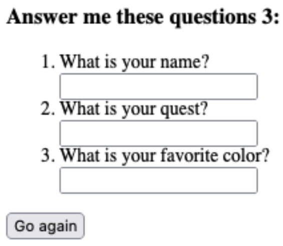
Activity 1: Create a simple survey using the in-DOM (HTML, no build tools approach)

Create a simple Vue app to answer these 3 very important questions:

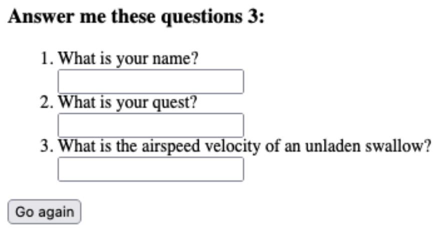
1. What is your name?
2. What is your quest?
3. What is your favorite color? -- OR -- What is the airspeed velocity of an unladen swallow?

After each question create a textual input to capture the user response, and put a "Go again" button at the bottom that resets the questions and input widgets.

It should look like this:



OR


Answer me these questions 3:
1. What is your name?
2. What is your quest?
3. What is the airspeed velocity of an unladen swallow?
Go again

Functional requirements:

R1. If the user enters "idk" (for 'I don't know') into any of the text inputs, then a line of text should appear just above the button that says "Knight: AAAAAAAAHHHHHHHHHHH!!!

R2. If the user provides 3 answers (> length 0) none of which are "idk") then a line of text should appear just above the button that says "Bridgekeeper: Alright, off you go then"

R3. When the "Go again" button is clicked, the 3rd question switches between the 2 options shown.

Constraints:

C1. You need to complete this as an HTML file using the no build tools method as shown in the code walkthroughs.

C2. No CSS should be included for this activity.

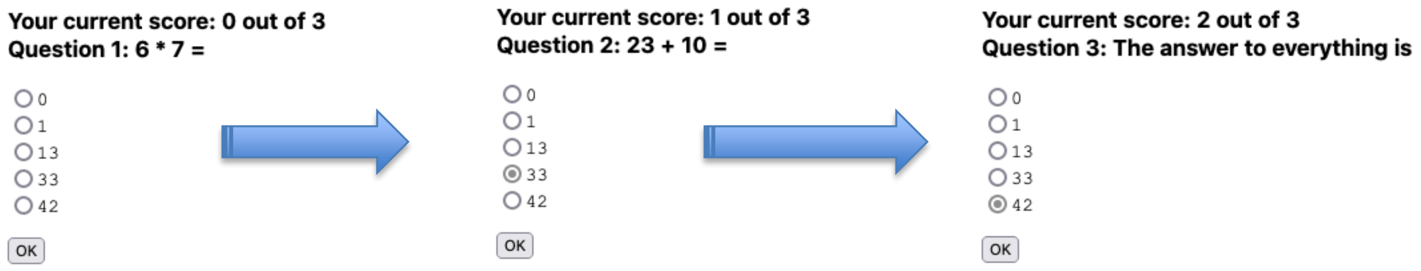
Hints:

- To complete this activity, you will need to use the v-if/v-else-if/v-else directives in some combination.
- You will also need the v-on directive.
- There is more than one way to handle answer processing for the 3 input textboxes. I suggest looking at v-model.

Submission: Include a file labvue_act1_<asurite>.html in your submission zipfile, where asurite is your ASURite (login) id.

Activity 2: Create a SFC for a simple survey

For this activity we will extend the concepts in Activity 1 by creating a dynamic survey. The survey is dynamic in the sense that it is not hardcoded to a specific set of questions, question length, or question choices (the questions are rendered as multiple-choice single-answer questions). For example, suppose I have the following 3 questions:


Your current score: 0 out of 3
Question 1: 6 * 7 =
0
1
13
33
42
OK
Your current score: 1 out of 3
Question 2: 23 + 10 =
0
1
13
33
42
OK
Your current score: 2 out of 3
Question 3: The answer to everything is
0
1
13
33
42
OK

I could use a Javascript part of SFC like this:

```
<script>
export default {
  data() {
    return {
       qno: 0,
       questions: ['6 * 7 =', '23 + 10 =', 'The answer to everything is '],
       qanswers: ['42','33', '42'],
       choices: ['0','1','13','33','42'],
       // you will add more to complete the SFC as you design
```

Where `qno` is the current question, `questions` is an array of the question stems, `qanswers` is the corresponding answers, and `choices` is the answer options for *all* questions. Note that this is not the complete Javascript part, and does not include the `<template>` and `<style>` parts of the SFC either; you will need to add to it to complete the functional requirements:

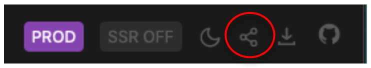
R1. When a user is not on the last question, then clicking "OK" takes them to the next question.
R2. When the component is rendered while the user is on a survey question, the first line will state the user's current score as shown in the images.
R3. Continuing on R2, the second line will say "Question #" where # is the current question number, and follow it with the question text.
R4. The available choices for the user's answer will be rendered as shown: a vertical line of radio buttons.
R5. Styling:
  a. The 1st 2 lines are bold and 20% larger than the default browser font size
  b. The choices are in normal size, `Courier New` font.
R6. When the user completes the questions, the last screen should look at follows:

You have completed the quiz! *Your score was 0 out of 3*

The questions are replaced with this statement, in **green**, Times New-Roman font, 50% larger than normal. The last part is italicized.

Constraints:
C1. You must do styling in a <style> block of an SFC
C2. You must use the 4 properties I give you above to start. Beyond that it is entirely up to you what other parts of the SFC `<script>` and `<template>` you have, though you must have all 3 SFC parts (`<script>`, `<template>`, `<style>`)
C3. You must implement this functionality within a single SFC

Submission:
Use the SFC Playground to implement your solution. When you are done, copy-paste your code in the left pane into a file named labvue_act2_<asurite>.vue. In addition, grab the share link in the upper right and copy it into the .vue file as a comment (//) at the top.



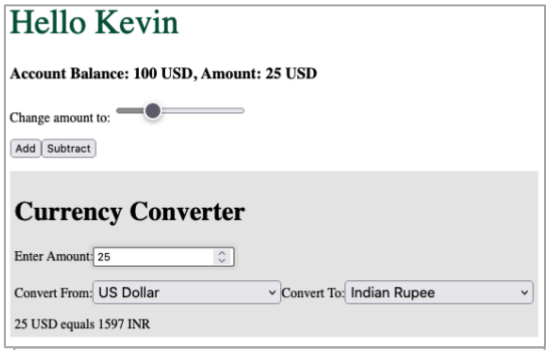
Activity 3: Create a multi-component Vue application with build tools
For this activity you will integrate multiple SFCs in a single application. Specifically, you will compose the HelloWorld, Balance, and Currency SFCs, all already available to you as sample code, into a Vue multi-SFC application.

A few changes will be required to individual components *HelloWorld* and *Balance*:

R1. HelloWorld: "Hello World" becomes "Hello <name>". However, if this revised app, you will need to put the name on the root component (App.vue) and use the Vue props feature to display it in the child component (HelloWorld.vue)
R2. Balance SFC: add a slider that is bound to the amount, and can change the amount value to anything between 5 and 100, in increments of 5.
R3. Balance SFC: Subtract is no longer allowed if the amount > balance.

The integration requirements are:
R4. The balance denomination in the Balance component must match the denomination chosen in the "Convert From:" field of the Currency component.
R5. The "Enter Amount" field on Currency must show a default value equal to the Balance amount

In order to do this, you will have to find a way to communicate across the hierarchy of Vue components. How you do this is up to you! The sample app has a single root component and 2 child components (HelloWorld and Balance), but these siblings do not "talk" to each other. You will need to add in Currency.vue (which is in the sample repo too as a separate file), and decide how the tree should be structured and what mechanism to use to keep this integration dynamic.


Hello Kevin

Account Balance: 100 USD, Amount: 25 USD

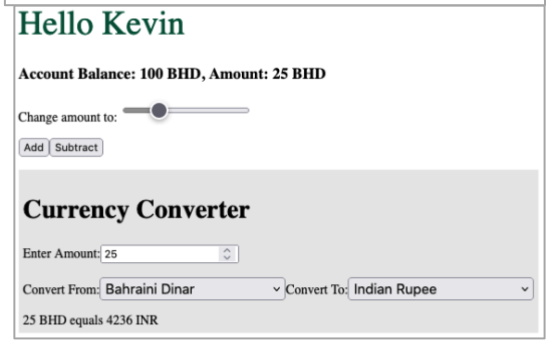
Change amount to:

Currency Converter

Enter Amount: 25

Convert From: US Dollar Convert To: Indian Rupee

25 USD equals 1597 INR


Hello Kevin

Account Balance: 100 BHD, Amount: 25 BHD

Change amount to:

Currency Converter

Enter Amount: 25

Convert From: Bahraini Dinar Convert To: Indian Rupee

25 BHD equals 4236 INR

Constraints:
1. You must use separate SFCs for HelloWorld, Balance, and Currency, not roll them together to solve the integration.
2. ~~"Global" values you can use anywhere are possible as a hack but are not allowed. Look at props, events, and ways of integrating!~~ *I think my statement here was misleading, so I removed it. You may use "Global State Management" as defined in the Vue 3 documentation (https://vuejs.org/guide/scaling-up/state-management.html#simple-state-management-with-reactivity-api), though there are multiple ways to handle this!*
3. You must use "npm init vue@latest" to create your project

Submission:
Please create a zipfile named lab3vue_act3_<asurite>.zip with your entire project directory, ***except*** the node_modules directory and package-lock.json file. Please remove these before you submit to reduce the size of your overall deliverable.

Submission Instructions over all for this lab:
Submit your lab as a single zipfile named <asurite>_ser421labVue.zip with the 3 respective files for each of activity 1, 2, and 3.