Web Programming

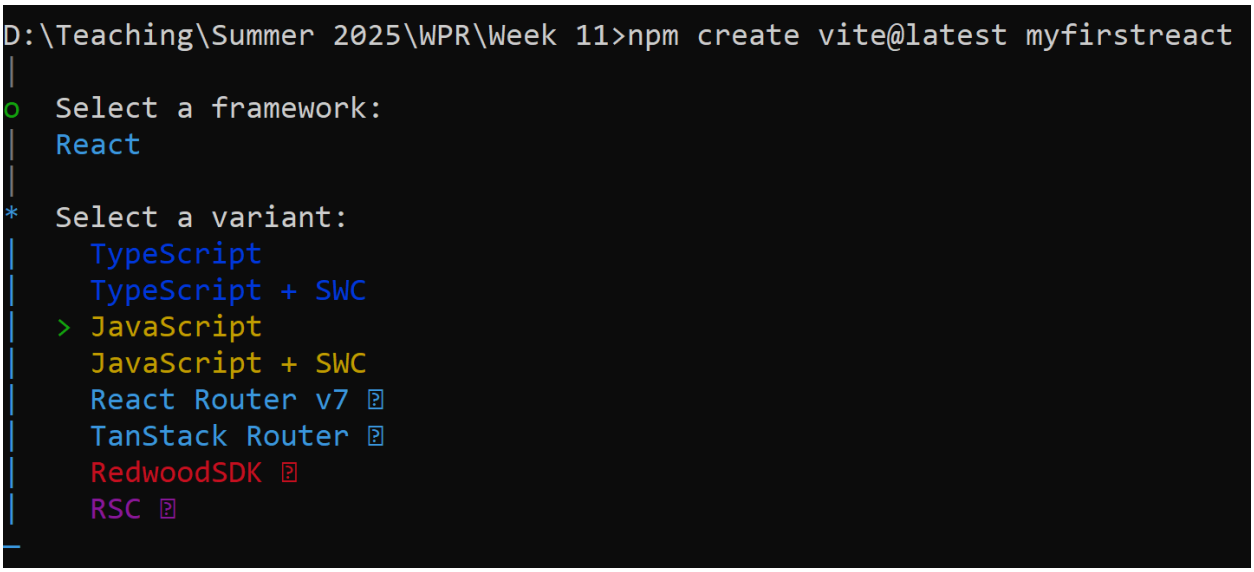
Tutorial 11

To begin this tutorial, please create a React project. When you finish, zip all your source codes (excluding the `node_modules` folder) to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `wpr01, wpr02`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

- Use `vite` to create a React app in a folder named `flashcards`:

`npm create vite@latest flashcards`

(Choose the `React` framework and the `JavaScript` variant)



Activity 1 – Flashcards

Goals

- To convert an HTML page into a React UI by dividing the page into a hierarchy of React class components. Download `tut11-starter.zip` to obtain the provided HTML and CSS files.
- To add simple event handling to this React UI.

Instructions

In the starter file, you'll find the `flashcards` page (shown above) in traditional HTML/CSS, convert it into a React UI. Try to divide the UI into smaller components and give those components suitable names. Here's how it looks like when finished:

Create an object whose property names are English words and values are Vietnamese meanings, such as:

```
const dict = {
    "pretty": "xinh đẹp",
    "car": "xe hơi",
    "study": "học tập",
    "life": "cuộc sống",
    "enormous": "to lớn",
    "computer": "máy tính"
};
```

From this object, create two convenient arrays:

```
const words = Object.keys(dict);
const meanings = Object.values(dict);
```

The idea is to display one of the words in this dictionary on the UI and let the user navigate to the next word using the right arrow button. For this to work, the application must main maintain a state which is the index of the currently displayed word.

Make `App` as a class component, and initialize the `state` object with one property named `current`. Add a method named `nextWord` in the `App` class to update the component's state so that the current value is increased by `1` (but don't let it be greater than the total number of words.

(*) The `nextWord` method must be an arrow function for the `this` keyword to work properly its body.

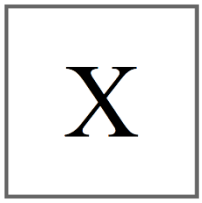
Other components don't need to hold any state so they can all be function components. You need to pass the nextWord method down to the component which contains the right arrow `button` element.

Activity 2

In this activity, you'll try to build a tic-tac-toe game board using React. You should:

- Divide the UI into components
- Use in-line styles for your components

First, create a `Square` component which is a box those width is `50px` and height is `50px`. This component has a `1px solid black` border. Try to position a character at the middle of this box:



Create a `Board` component which is a box of `150 x 150` pixels and put 9 squares in it (use a loop). Let `Board` be a flex box with `flex-wrap: wrap` to let items go to next line when there's no space left. Each item should take exactly `50 x 50` pixels for this board to work. However, a Square now actually takes `52 x 52` pixels (border included). So if you change the width of the squares to `48px`, then the board will look like this:

1	2	3
4	5	6
7	8	9

Please note how the borders are not the same (the inner borders are twice as thick as the outer ones). What you should to is to change the width of the squares into `49px` and set the right and bottom margins to `-1px`. This effectively moves all the squares (except square #1) `1px` to the top and left. The end result should look like this:

1	2	3
4	5	6
7	8	9

We'll continue to add gameplay logic to this TicTacToe game in the next tutorial.