Tutorial 5: JavaScript (3)

Objectives

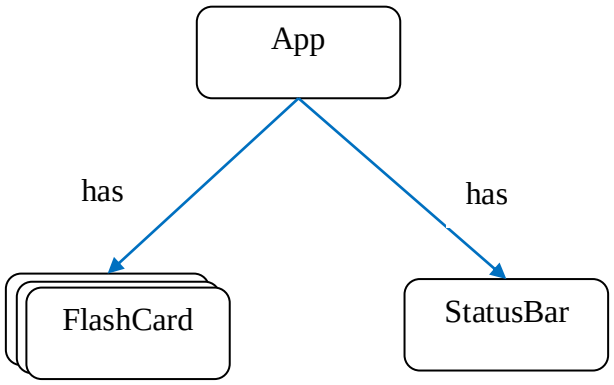
- Practice to communicate between classes in JavaScript using **custom events & callback functions**
- Practice to consume the server APIs
    - Practice `fetch()` APIs with JSON (promise style)
    - Using form to get user input & send params to the server APIs

Tutorial Exercises

In this tutorial, you will **work in pairs** to solve these exercises.

Exercise 1: Flash cards – Communication between StatusBar & App (30 mins)

Download the solution from the previous tutorial `tut04/flash-cards-oop/`, and have a look.



When user click on *`StatusBar`*, it need to notify the `App` to display the correct *`FlashCard`*. In the below tasks, you will facilitate this using 2 different ways:

(1) custom events &

(2) call back functions

Task 1: Using Custom events (15 mins)

In this task, you will use ***custom events*** to facilitate communication between the two classes `StatusBar` & `App`.

Duplicate folder `tut04/flash-cards-oop/` & named it under `tut05/flash-cards-oop-custom-events/` and refactor code so that:

- Class `StatusBar`: when user click events happen, this class has to
  - (1) update (increase/ decrease) the current index & display
  - (2) also, disable buttons (if needed).
  - (3) dispatch custom events ('prev-clicked', 'next-clicked') with the corresponding events.

```
document.dispatchEvent(new CustomEvent('event-name', {data object}));

document.addEventListener('event-name', eventHandlerFunction);
```

- Class `App`: listen for the custom events & display the correct flashcard at the notified index

Task 2: Communication between App & StatusBar (15 mins)

In this task, you will use ***callback functions*** to facilitate communication between the two classes `StatusBar` & `App`.

Duplicate folder `tut05/flash-cards-oop-custom-events/` & named it under `tut05/flash-cards-oop-callbacks/` and refactor code so that:

- Class `App`: passes into the `StatusBar` the callback functions for prev/ next button click events,
- Class `StatusBar`: invokes the appropriate callback function in each suitable case.

Exercise 2: Flash cards – Dynamic data

You can see that the words are now stored as an object in the *data.js* file. Let's make it dynamic by fetching them from the API:

GET https://wpr-quiz-api.herokuapp.com/words/

Request:

- **Data**: none

Response:

- **Status**: 200 (OK)
- **JSON data**: an object of words in format of {word: definition, …}

In file main.js, fetch words before rendering the app.

```
function onResponse(response) {
    return response.json();
}

function onJson(json) {
    console.log(json);
}

fetch('url').then(onResponse).then(onJson);
```

Note: In this case, since we process JSON data (access link API directly on your browser & have a look), the `.json()` is used.

Exercise 3: Flash cards – Add a new word

You will see another file existed in the starterpack, '***add.html***'. A form was defined to add a new word into the list of words for our flashcards app.

- Have a look on the HTML code with form (action, method, input name)
- Enable the line `event.preventDefault()` to observe what this line of code does
- Use fetch to send data to server for adding a new card

POST https://wpr-quiz-api.herokuapp.com/words/

Request:

- **Data**: JSON body contain word & definition

  eg. {"word": "네", "definition": "Yes"}

Response:

- **Status**: 201 (CREATED)
- **JSON data**: an object of just added word in format of {word: definition}

```
function onResponse(response) {
    return response.json();
}

function onJson(json) {
```

```
    console.log(json);
}

fetch('url', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(data)
}).then(onResponse).then(onJson);
```

Note: different from the Exercise 2, this API requires some more information

- `method: 'POST'` (GET by default, that is the reason why it is omitted in Ex. 2)
- `headers: {'Content-Type': 'application/json'}`: since you can send data using normal form (form data) or JSON, so you need to tell server that you are sending JSON data (This is omitted in Ex. 2 since you send nothing)
- `body: JSON.stringify(data)` is utilized to transform data from JavaScript object into string

Exercise 4: (Optional) Update/ Delete word

Similarly, you can extend your Flashcards app with use of these APIs to update & delete a word.

Note: word is unique (distinct), in this case, it is used as the primary key

PUT https://wpr-quiz-api.herokuapp.com/words/:word

Request:

- **Data**:
    - Route param:
        - :word : the word to update
    - JSON body contain new definition

  eg. https://wpr-quiz-api.herokuapp.com/words/너
  { "definition": "Yes"}

Response:

- **Status**: 200 (SUCCESS)

- **JSON data**: an object of just updated word in format of {word: definition}

DELETE https://wpr-quiz-api.herokuapp.com/words/:word

Request:

- **Data**:
    - Route param:
        - :word : the word to update

    eg. https://wpr-quiz-api.herokuapp.com/words/넦

Response:

- **Status**: 204 (NO CONTENT)
- **JSON data**: an object of just deleted word in format of {word: definition}