Open-Source Software Practice

Lab 10. Desktop App

TA: **Jiwon Choi** (최지원, jasonchoi3@g.skku.edu)

Interactive Data Computing Lab (IDCLab)

College of Computing and Informatics,

Sungkyunkwan University

Goals

- Develop SKKU-Todo-2
- Learn local storage
- Publish with Electron

SKKU-Todo-2

- **Starter template**
  - https://github.com/e-/skku-todo-2/blob/main/skeleton.html
  - You can use your own code that we developed in the previous lab session.

- **Make sure that no Korean characters are included in the path to your HTML file.**
  - Electron build can fail.

Adding a Task

- Task as an object
  - Easy to maintain and control
  - text: string
  - type: number

```
const Type = {
    Todo: 1,
    Done: 2,
};
```

```
// 1. Read the text in #task-input.
let input = document.querySelector("#task-input");
let text = input.value;

if (!text.length) return;

// 2. Create a new Task object.
let task = {
    text: text,
    type: Type.Todo
};
```

Adding a Task

- We need two buttons for each task item.


```
<div id="todo-list">
    <div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
        <span class="me-auto">Task text</span>
        <button class="btn btn-sm btn-success me-1">
            <i class="bi bi-check"></i>
        </button>
        <button class="btn btn-sm btn-danger">
            <i class="bi bi-x"></i>
        </button>
    </div>
</div>
```

Adding a Task

- Let's write addToList()
- addToList()
  - Create item HTML object.
  - If task.type is Type.Todo append the item to #todo-list.
  - Otherwise, append it to #done-list.

```
let button = document.querySelector("#add");
button.addEventListener("click", () => {
    // 1. Read the text in #task-input.
    let input = document.querySelector("#task-input");
    let text = input.value;

    if (!text.length) return;

    // 2. Create a new Task object.
    let task = {
        text: text,
        type: Type.Todo
    };

    // 3. Create a new task item and attach it to #todo-list.
    addToList(task);

    // 4. Clear #task-input.
    input.value = "";
});
```

Adding a Task

- **`HTMLElementObject.innerHTML`**
  - innerHTML gets or sets the HTML or XML markup contained within the element.
  - Reading the HTML contetns.
  - Replacing the contents of an element.

- **Conditional Operator**
  - `Condition ? exprIfTrue : exprIfFalse`

Adding a Task


```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#todo-list" : "#done-list");
    list.appendChild(div);
}
```

Removing a Task

- **`HTMLElementObject.remove()`**
  - Remove HTML Element from document.

- **Closure**
  - Closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment).
  - In other words, a closure gives you access to an outer function's scope from an inner function.
  - Every (almost) function of JavaScript creates with closure (Every function in JavaScript can access outer function’s scope)

Removing a Task


```
let buttonRemove = document.createElement("button");
buttonRemove.className = "btn btn-sm btn-danger";
buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
div.appendChild(buttonRemove);

buttonRemove.addEventListener("click", () => {
    div.remove();
});

let list = document.querySelector(task.type === Type.Todo ? "#todo-list" : "#done-list");
list.appendChild(div);
```

- Also, the answer of last lab session's summary quiz :P

Saving and Loading the State

- `window.localStorage`
  - localStorage read-only property of the window interface allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions.
  - Document's origin: Protocol(HTTP, HTTPS..) + Host(Domain, IP) + Port(443, 80)

- localStorage methods
  - `localStorage.setItem(key, value)`
  - `localStorage.getItem(key)`
  - `localStorage.removeItem(key)`
  - `localStorage.clear()`

Saving the State


```
let tasks = [];
```

```
function saveTasks() {
    localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

```
// 2. Create a new Task object.
let task = {
    text: text,
    type: Type.Todo
};


// 3. Append the new Task object to tasks
tasks.push(task);
saveTasks();
```

```
// in function addToList(task)

buttonRemove.addEventListener("click", () => {
    div.remove();
    tasks = tasks.filter(t => t !== task);
    saveTasks();
});
```

Loading the State

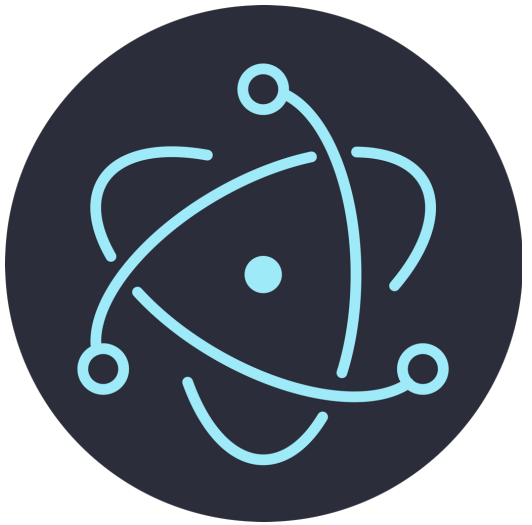

```
window.addEventListener("load", () => {
    loadTasks();
});
```

```
function loadTasks() {
    let lastTasks = localStorage.getItem("tasks");
    if (!lastTasks) return;

    tasks = JSON.parse(lastTasks);
    tasks.forEach(t => {
        addToList(t);
    });
}
```

```
tasks.forEach(addToList);
```

Electron

- **Electron**
  - Electron is a framework that builds cross-platform desktop apps using Web technologies.
  - Slogan: If you can build a website, you can build a desktop app.
  - https://www.electronjs.org/
  - https://www.electronjs.org/docs/tutorial/quick-start



Electron

- `npm init`
- Create three files
  - `main.js`
    - https://www.electronjs.org/docs/tutorial/quick-start
    - https://github.com/e-/skku-todo-2/blob/main/main.js
  - `preload.js`
    - Empty JS file
  - `index.html`
    - The HTML file we coded

Electron

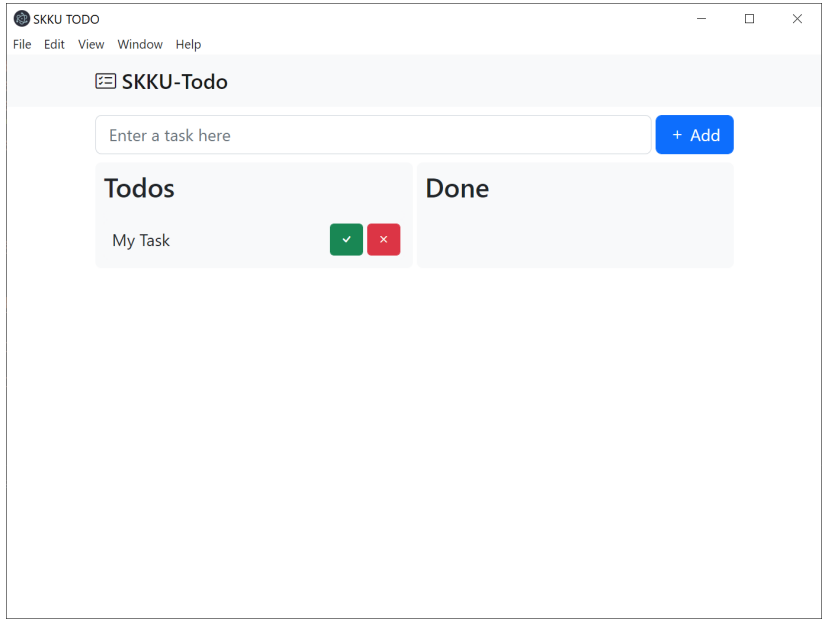
- main.js

```
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow() {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  win.loadFile('index.html')
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow()
    }
  })
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})
```

Electron

- `npm install --save-dev @electron-forge/cli`
- `npx electron-forge import`
- package.json will be converted.

- `npm start`



Remove the Menu Bar

- main.js

```
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow() {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  win.setMenuBarVisibility(false);
  win.loadFile('index.html')
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow()
    }
  })
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})
```

Electron

- Let's make a package for distribution.
- `npm run make`
  - It will take a while.
  - Make sure Hangul is not included in the path to the project.