Class - 19 Kanban board 2

Agenda of this Lecture:

1. Modal Popup with Toggle
2. Ticket Generation
3. Adding Task, colour, ID to generated ticket
4. Ticket Removal

Pop up modal Generation

1. Whenever the (+) button is clicked we need to open the modal pop up box so that we can enter our Task and whenever we click on the button (+) again the pop up closes.
2. How we will Implement this?
    a. So to implement this we can use a very simple technique i.e we can set up a flag ,
    b. by default we will keep it to false, so nothing will happen but as soon as we click the (+) button we will change the flag to true , and when it is true we will open the modal pop up
    c. Again when you click on this now the flag will again turn back to false and the pop up will close
3. Implementation
    a. Create a script.js and link it in the html

```
const addBtn = document.querySelector(".add-btn");
```

```
const modalCont = document.querySelector(".modal-cont");
let addTaskFlag = false;

addBtn.addEventListener("click", function () {
 // Display the model
 addTaskFlag = !addTaskFlag;

 if (addTaskFlag == true) {
   modalCont.style.display = "flex";
 } else {
   modalCont.style.display = "none";
 }
});
```

b. let addTaskFlag = false: This line initializes a boolean flag named addTaskFlag to false. This flag will be used to keep track of the modal's visibility state—whether it is currently shown (true) or hidden (false).
c. Change the display as none for modal-cont in css

Ticket Generation

1. Now , let's generate a task ticket, which involves creating a function to dynamically generate new task tickets.

```
function createTicket() {
 // Create a new ticket container element
 const ticketCont = document.createElement("div");
}
```

2. Now we will add class to this particular div using the setAttribute

```
function createTicket() {
 // Create a new ticket container element
```

```
 const ticketCont = document.createElement("div");

 // Set the class attribute of the ticket container
 ticketCont.setAttribute("class", "ticket-cont"); //
}
```

3. Whenever this function is called, a new ticket will be created with class ticket-cont.
4. As ticketCont contains 3 more divs inside, we will create them inside this function using innerHTML
5. We can create new divs and append but this approach can be shorter

```
const mainCont = document.querySelector(".main-cont");

function createTicket() {
 // Create a new ticket container element
 const ticketCont = document.createElement("div");
 // Set the class attribute of the ticket container
 ticketCont.setAttribute("class", "ticket-cont"); //
 ticketCont.innerHTML = `
 <div class="ticket-color"></div>
 <div class="ticket-id">12345/div>
 <div class="task-area">Random Task</div>
<div class="ticket-lock">
   <i class="fa-solid fa-lock"></i>
</div>
`;
 mainCont.appendChild(ticketCont);
}
```

6. When we set the innerHTML property, **the string is treated as HTML**, meaning any HTML tags within the string will be parsed and rendered as DOM elements. This allows us to construct complex elements with nested tags directly from strings.

Adding Event Listener to Generate Ticket

1. We add an event listener to the modalCont element for the 'keydown' event. This event occurs when a key on the keyboard is pressed and then released.

```
const textArea = document.querySelector(".textArea-cont");

modalCont.addEventListener("keydown", function (e) {
 const key = e.key;
 if (key === "Shift") {
   createTicket(); // Call the createTicket function to create
a new ticket
   modalCont.style.display = "none"; // Hide the modal
   textArea.value = ""; // Clear the textarea's content
 }
});
```

2. The if (key === 'Shift') { ... } block checks if the pressed key is the Shift key. If it is, the code block inside the if statement is executed, which involves three actions:
    a. createTicket();

        i. This line calls the createTicket() function, which, as described previously, creates a new ticket element and adds it to the page.
    b. modalCont.style.display = 'none';
        i. This line hides the modalCont element by setting its display style property to 'none'. This is typically used to hide modal dialogs or similar components after completing an action.
3. textArea.value = '';
    a. This line clears the content of an element referred to by textArea

Adding Task, Color, ID to the generated ticket

1. As of now only the static ticket is getting generated on press of the shift
2. everything like Task, color and ID of the created task is static and we are not able to set the values by ourselves.
3. Requirements
    a. we need to choose color so a specific color band will be applied to the ticket, and the ticket will also have a unique ID for which we will generate IDs for them
    b. To identify and select these priority color divs, we will use querySelectorAll method
    c. We want to select all elements with the class name 'priority-color' using querySelectorAll and then iterate

through each of these elements using the forEach method. Here's how to do that:

```
const allPriorityColors =
document.querySelectorAll(".priority-color");

allPriorityColors.forEach(function (colorElem) {
 colorElem.addEventListener("click", function () {
   // Remove 'active' class from all priority colors
   allPriorityColors.forEach(function (priorityColorElem) {
     priorityColorElem.classList.remove("active");
   });

   // Add 'active' class to the clicked colorElem
   colorElem.classList.add("active");

   // Implement additional logic to assign the selected color
to a task
   // For example, you can use this space to perform your task
color assignment
 });
});
```

d. In this code, when a color element with the class 'priority-color' is clicked, the event listener:
e. Iterates through all allPriorityColors and removes the 'active' class from each element.
f. Adds the 'active' class to the clicked colorElem.
g. Implements additional logic to assign the selected color to a task
h. To get a particular color, we create a variable to track the selected color

i.

```
let modalPriorityColor = 'black'

allPriorityColors.forEach(function (colorElem) {
 colorElem.addEventListener("click", function () {
   // Remove 'active' class from all priority colors
   allPriorityColors.forEach(function (priorityColorElem) {
     priorityColorElem.classList.remove("active");
   });

   // Add 'active' class to the clicked colorElem
   colorElem.classList.add("active");

   modalPriorityColor = colorElem.classList[0];
 });
});
```

Pass the selected color to the ticket

1. Our createTicket function will start accepting a color parameter
2. Updated createTicket function

```
function createTicket(ticketColor) {
 // Create a new ticket container element
 let ticketCont = document.createElement("div");
 ticketCont.setAttribute("class", "ticket-cont");

 // Create the HTML content for the ticket container
 ticketCont.innerHTML = `
     <div class="ticket-color" style="background-color:
${ticketColor};"></div>
     <div class="ticket-id">12345</div>
     <div class="task-area">Random Task</div>
```

```
    `;

 // Append the ticket container to the main container
 mainCont.appendChild(ticketCont);
}
```

3. Update the function invocation in the shift key event listener

```
const textArea = document.querySelector(".textArea-cont");

modalCont.addEventListener("keydown", function (e) {
 const key = e.key;
 if (key === "Shift") {
   createTicket(modalPriorityColor); // Call the createTicket
function to create a new ticket
   modalCont.style.display = "none"; // Hide the modal
   addTaskFlag = false; // Set the addTaskFlag to false
   textArea.value = ""; // Clear the textarea's content
 }
});
```

Updating Task details

```
function createTicket(ticketColor, ticketTask) {
 // Create a new ticket container element
 let ticketCont = document.createElement("div");
 ticketCont.setAttribute("class", "ticket-cont");

 // Create the HTML content for the ticket container
 ticketCont.innerHTML = `
    <div class="ticket-color" style="background-color: ${ticketColor};"></div>
    <div class="ticket-id">12345</div>
    <div class="task-area">${ticketTask}</div>
   `;

 // Append the ticket container to the main container
```

```
 mainCont.appendChild(ticketCont);
}
```

1. Passing content in the createTicket

```
modalCont.addEventListener("keydown", function (e) {
 const key = e.key;
 if (key === "Shift") {
   const taskContent = textArea.value; // Get the content from the
textarea
   createTicket(modalPriorityColor, taskContent); // Create a new
ticket with the selected color and task content
   modalCont.style.display = "none"; // Hide the modal
   addTaskFlag = false; // Set the addTaskFlag to false
   textArea.value = ""; // Clear the textarea's content
 }
});
```

Generating unique id for the ticket

1. We will be using an external library shortID for this
2. Copy the script tag
3. Put it at the end

```
<script
src="https://unpkg.com/shortid-dist@1.0.5/dist/shortid-2.2.13.min.js"></script>
   <script src="./script.js"></script>
 </body>
```

4. Generate the id
    a. We will use inbuilt function to do this for now. Can use library as well for this

```
modalCont.addEventListener("keydown", function (e) {
```

```
 const key = e.key;
 if (key === "Shift") {
   const taskContent = textArea.value; // Get the content from the
textarea
   // Generates a 6-character ID

   const ticketID = Math.random().toString(36).substring(2,
8);
//    let ticketID = shortid()
    createTicket(modalPriorityColor, taskContent,
ticketID); // Create a new ticket with the selected color
and task content
   modalCont.style.display = "none"; // Hide the modal
   addTaskFlag = false; // Set the addTaskFlag to false
   textArea.value = ""; // Clear the textarea's content
 }
});
```

5. Updating createTicket function

```
function createTicket(ticketColor, ticketTask, ticketID) {
 // Create a new ticket container element
 let ticketCont = document.createElement("div");
 ticketCont.setAttribute("class", "ticket-cont");
 // Create the HTML content for the ticket container
 ticketCont.innerHTML = `
     <div class="ticket-color" style="background-color: ${ticketColor};"></div>
     <div class="ticket-id">${ticketID}</div>
     <div class="task-area">${ticketTask}</div>
   `;
 // Append the ticket container to the main container
 mainCont.appendChild(ticketCont);
}
```

Ticket Removal from UI

1. Same approach as for add ticket

```
const removeBtn = document.querySelector(".remove-btn");
```

```
let addTaskFlag = false;
let removeTaskFlag = false;
```

```
removeBtn.addEventListener("click", function () {
 removeTaskFlag = !removeTaskFlag; // Toggle the removeTaskFlag
when the button is clicked

 if (removeTaskFlag) {
   alert("Delete button is activated.");
   removeBtn.style.color = "red";
 } else {
   removeBtn.style.color = "white";
 }
});
```

2. To remove the tickets, we will loop through tickets and add an event for click

```
const allTickets = document.querySelectorAll(".ticket-cont");

function handleRemoval(ticket) {
 ticket.addEventListener("click", function () {
   if (!removeTaskFlag) return;
   else {
     ticket.remove();
   }
 });
}

allTickets.forEach(function (ticket) {
 handleRemoval(ticket);
})
```

Complete code for reference:

1. html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Document</title>
   <script
     src="https://kit.fontawesome.com/589957875e.js"
     crossorigin="anonymous"
   ></script>
   <link rel="stylesheet" href="style.css" />
 </head>
 <body>
   <div class="toolbox-cont">
     <div class="toolbox-priority-cont">
       <div class="lightpink color"></div>
       <div class="lightgreen color"></div>
       <div class="lightblue color"></div>
       <div class="black color"></div>
     </div>
     <div class="action-btn-cont">
       <div class="add-btn">
         <i class="fa-solid fa-plus"></i>
       </div>
       <div class="remove-btn">
         <i class="fa-solid fa-xmark"></i>
       </div>
     </div>
   </div>
   <div class="main-cont">
     <div class="ticket-cont">
       <div class="ticket-color"></div>
       <div class="ticket-id">12345</div>
       <div class="task-area">Random Task</div>
       <div class="ticket-lock">
         <i class="fa-solid fa-lock"></i>
       </div>
```

```
    </div>
  </div>
  <div class="modal-cont">
    <textarea class="textArea-cont" placeholder="Enter Your Task"></textarea>
    <div class="priority-colors-container">
      <div class="lightpink priority-color"></div>
      <div class="lightgreen priority-color"></div>
      <div class="lightblue priority-color"></div>
      <div class="black priority-color active"></div>
    </div>
  </div>
  <script src="./script.js"></script>
 </body>
</html>
```

2. css

```
* {
 box-sizing: border-box;
}
body {
 margin: 0;
 padding: 0;
}
.toolbox-cont {
 height: 5rem;
 background-color: #4b4b4b;
 display: flex;
 align-items: center;
}
.toolbox-cont > * {
 margin-left: 4rem;
}

.toolbox-priority-cont {
 height: 3.5rem;
 width: 18rem;
 background-color: #3d3d3d;
 display: flex;
 align-items: center;
 justify-content: space-evenly;
```

```
}
.toolbox-priority-cont > *:hover {
 background-color: #485460;
}

.color {
 height: 1.5rem;
 width: 3rem;
}
.lightpink {
 background-color: lightpink;
}
.lightgreen {
 background-color: lightgreen;
}
.lightblue {
 background-color: lightblue;
}
.black {
 background-color: black;
}

.action-btn-cont {
 height: 3.5rem;
 width: 8rem;
 background-color: #3d3d3d;
 display: flex;
}
.action-btn-cont > * {
 display: flex;
 width: 50%;
 font-size: 2rem;
 color: white;
 justify-content: center;
 align-items: center;
}

.add-btn:hover,
.remove-btn:hover {
 background-color: #4bb543;
}
```

```css
/* ticket css */
.main-cont {
 display: flex;
 gap: 2rem;
 justify-content: center;
 padding: 2rem;
 flex-wrap: wrap;

 /* As many tickets can be added here so this will make sure that the design is neat to accomodate multiple tickets */
}

.ticket-cont {
 height: 12rem;
 width: 15rem;
 background-color: coral;
}
.ticket-color {
 height: 1rem;
}

.ticket-id {
 background-color: yellow;
 height: 2rem;
}

.ticket-lock {
 display: flex;
 justify-content: flex-end;
 margin-top: 90px;
 margin-right: 5px;
 font-size: 1.5rem;
}
.textArea-cont {
 height: 100%;
 width: 75%;
 resize: none;
 outline: none;
 border: none;
 background-color: #dfe4ea;
 font-size: 2rem;
 color: black;
```

```
}

.modal-cont {
 height: 50vh;
 width: 45vw;
 display: none;
 background-color: lightsalmon;
 position: absolute;
 top: 30%;
 left: 27%;
}

.textArea-cont {
 height: 100%;
 width: 75%;
 resize: none;
 outline: none;
 border: none;
 background-color: #dfe4ea;
 font-size: 2rem;
 color: black;
}
.priority-colors-container {
 height: 100%;
 width: 25%;
 display: flex;
 flex-direction: column;
 background-color: #4b4b4b;
 align-items: center;
 justify-content: space-around;
}

.priority-color {
 height: 2rem;
 width: 3rem;
}

.active {
 border: 5px solid lightsalmon;
}
```

3. js

```
const addBtn = document.querySelector(".add-btn");
const modalCont = document.querySelector(".modal-cont");
const mainCont = document.querySelector(".main-cont");
const allPriorityColors = document.querySelectorAll(".priority-color");
const textArea = document.querySelector(".textArea-cont");
const colors = ["lightpink", "lightgreen", "lightblue", "black"];
let modalPriorityColor = colors[colors.length - 1]; // Default to black
const removeBtn = document.querySelector(".remove-btn");

let addTaskFlag = false;
let removeTaskFlag = false;

addBtn.addEventListener("click", function () {
 // Display the model
 addTaskFlag = !addTaskFlag;

 if (addTaskFlag == true) {
   modalCont.style.display = "flex";
 } else {
   modalCont.style.display = "none";
 }
});

removeBtn.addEventListener("click", function () {
 removeTaskFlag = !removeTaskFlag; // Toggle the removeTaskFlag when the button
is clicked

 if (removeTaskFlag) {
   alert("Delete button is activated.");
   removeBtn.style.color = "red";
 } else {
   removeBtn.style.color = "white";
 }
});

modalCont.addEventListener("keydown", function (e) {
 const key = e.key;
 if (key === "Shift") {
   const taskContent = textArea.value; // Get the content from the textarea
   // const ticketID = nanoid.nanoid(6);
   // Generates a 6-character ID
```

```
   createTicket(modalPriorityColor, taskContent); // Create a new ticket with
the selected color and task content
   modalCont.style.display = "none"; // Hide the modal
   addTaskFlag = false; // Set the addTaskFlag to false
   textArea.value = ""; // Clear the textarea's content
 }
});

function createTicket(ticketColor, ticketTask, ticketID) {
 // Create a new ticket container element
 let ticketCont = document.createElement("div");
 ticketCont.setAttribute("class", "ticket-cont");
 // Create the HTML content for the ticket container
 ticketCont.innerHTML = `
     <div class="ticket-color" style="background-color: ${ticketColor};"></div>
     <div class="ticket-id">${ticketID}</div>
     <div class="task-area">${ticketTask}</div>
   `;
 // Append the ticket container to the main container
 mainCont.appendChild(ticketCont);
}

allPriorityColors.forEach(function (colorElem) {
 colorElem.addEventListener("click", function () {
   // Remove 'active' class from all priority colors
   allPriorityColors.forEach(function (priorityColorElem) {
     priorityColorElem.classList.remove("active");
   });

   // Add 'active' class to the clicked colorElem
   colorElem.classList.add("active");

   modalPriorityColor = colorElem.classList[0];
 });
});
```