Class 20 - Kanban board - 3

Agenda of this Lecture:
Locking Mechanism
Changing the Priority color of the Task
Filtering out Task with using the priority color filter
Showing All Tasks on db click

Locking Mechanism

1. Now we will be adding a lock so that a task can be edited of a ticket if it needs to be updated and we can lock it again

```
```

2. Now if we observe the lock icons
    a. Actually it is a different icon altogether
    b. Check the fa-lock and fa-lock-open
3. So we want to toggle between these two icons

```
let addTaskFlag = false;
let removeTaskFlag = false;
const lockClose = "fa-lock";
const lockOpen = "fa-lock-open";
```

4. Now we add an event listener and if the icon is lock, add lock-open and vice versa

```
function handleLock(ticket) {
 const ticketLockElem = ticket.querySelector(".ticket-lock");
```

```
const ticketLockIcon = ticketLockElem.children[0];

ticketLockIcon.addEventListener("click", function () {
  console.log("Lock Selected");
  if (ticketLockIcon.classList.contains(lockClose)) {
    ticketLockIcon.classList.remove(lockClose);
    ticketLockIcon.classList.add(lockOpen);
  } else {
    ticketLockIcon.classList.remove(lockOpen);
    ticketLockIcon.classList.add(lockClose);
  }
});
}
```

5. const ticketLockIcon = ticketLockElem.children[0]; accesses the first child element of ticketLockElem and assigns it to ticketLockIcon. This child the icon that visually represents the lock's current status (locked or unlocked).
6. The if statement checks if ticketLockIcon currently has the class corresponding to a closed lock (lockClose):
    a. If true (ticketLockIcon.classList.contains(lockClose)), it means the lock is currently shown as closed. The code then removes the lockClose class and adds the lockOpen class, visually changing the icon to an open lock.
    b. If false, it implies the lock is shown as open. The code removes the lockOpen class and adds the lockClose class, changing the icon back to a closed lock.
7. Call the function when ticket is being created

```
handleRemoval(ticketCont);
handleLock(ticketCont);
```

Making the content editable when lock is open

1. Using Contenteditable attribute

```
function handleLock(ticket) {
 const ticketLockElem = ticket.querySelector(".ticket-lock");
 const ticketLockIcon = ticketLockElem.children[0];
 const ticketTaskArea = ticket.querySelector(".task-area");

 ticketLockIcon.addEventListener("click", function () {
   console.log("Lock Selected");
   if (ticketLockIcon.classList.contains(lockClose)) {
     ticketLockIcon.classList.remove(lockClose);
     ticketLockIcon.classList.add(lockOpen);
     ticketTaskArea.setAttribute("contenteditable", "true"); //
Changed 'contenteditable', 'true'
   } else {
     ticketLockIcon.classList.remove(lockOpen);
     ticketLockIcon.classList.add(lockClose);
     ticketTaskArea.setAttribute("contenteditable", "false"); //
Changed 'contenteditable', 'true'
   }
 });
}
```

   a. ticketTaskArea.setAttribute('contenteditable', 'true'); sets the contenteditable attribute of the ticketTaskArea to 'true'. This makes the task area editable, allowing users to modify its content directly in the browser. This change is applied when the lock icon is clicked and the lock is transitioning from closed to open.

Changing the Priority color of the Task

1. We can click on the color band of the ticket and change the colors to denote transitioning of ticket from todo to in progress to review and then completed
2. We will create a function like others to handle color change
3. Now this is also a pattern where if you need to select something from a set of values and it is like a moving selection

```
const colors = ["lightpink", "lightgreen", "lightblue", "black"];
```

```
function handleColor(ticket) {
 const ticketColorBand = ticket.querySelector(".ticket-color");
 ticketColorBand.addEventListener("click", function () {
   let currentColor = ticketColorBand.style.backgroundColor;
   let currentColorIdx = colors.findIndex(function (color) {
     return currentColor === color;
   });

   currentColorIdx++; // Increment the index

   const newTicketColorIdx = currentColorIdx % colors.length;
   const newTicketColor = colors[newTicketColorIdx];

   ticketColorBand.style.backgroundColor = newTicketColor;

 });
}
```

4. Finding the Index of the Current Color: The index of currentColor within the colors array is found using colors.findIndex(). This

method takes a function that returns true for the element that matches the current color. The index of this color in the colors array is stored in currentColorIdx.

5. Incrementing the Color Index: The color index (currentColorIdx) is incremented by 1 to move to the next color in the colors array. If this increment makes the index equal to the length of the colors array, it will wrap around to 0 in the next step due to the modulo operation
6. Calculating the New Color Index: The new color index is calculated using currentColorIdx % colors.length. This ensures that if the incremented index is beyond the last index of the colors array, it wraps around to the beginning (cyclic behavior).
7. In summary, this function allows a user to cycle through a predefined list of color classes for an element within a ticket by clicking on it, changing its visual appearance each time it's clicked.

Filtering out Task with using the priority color filter

```
const toolboxColors = document.querySelectorAll(".color");
```

```
toolboxColors.forEach(function(colorElem) {
 colorElem.addEventListener("click", function() {
   const selectedColor = colorElem.classList[0];
   const allTickets = document.querySelectorAll(".ticket-cont");
   allTickets.forEach(function(ticket) {
     const ticketColorBand = ticket.querySelector(".ticket-color");
```

```
    if (ticketColorBand.style.backgroundColor === selectedColor) {
      ticket.style.display = "block";
    } else {
      ticket.style.display = "none";
    }
  });
});
colorElem.addEventListener("dblclick", function() {
  const allTickets = document.querySelectorAll(".ticket-cont");
  allTickets.forEach(function(ticket) {
    ticket.style.display = "block";
  });
});
})
```

What is local storage?

1. Local Storage is a **web storage mechanism** that allows websites and applications to store and access data right in the user's browser with no expiration date.
2. This means the data stored in Local Storage persists even after the browser window is closed, making it a convenient place to store data that needs to be accessed across sessions.
3. It offers a more secure and faster way of handling data than cookies, as the data is not sent with every server request.
4. There are two main types of web storage:
    a. Local Storage: Stores data with no expiration date.
    b. Session Storage: Stores data for one session and is cleared when the browser tab is closed.

5. Local Storage is part of the Web Storage API, which provides a simple key-value store.
6. Each piece of data stored is a string, and it is associated with a unique key. You can think of it as a dictionary or a map, where each key points to a specific piece of data.

How to Use Local Storage

1. Using Local Storage involves three basic operations: setting items, getting items, and removing items.
2. Setting Items
    a. To store data in Local Storage, you use the setItem method.
    b. This method requires two arguments: the key and the value. Both the key and the value must be strings.
    c. localStorage.setItem('key', 'value');
    d. If you need to store objects or arrays, you can serialize them to a string using JSON.stringify().

```
localStorage.setItem('user', JSON.stringify({ name:
'Alice', age: 30 }));
```

3. Getting Items
    a. To retrieve data from Local Storage, use the getItem method with the key as the argument.
    b. const value = localStorage.getItem('key');
    c. For objects or arrays, you can convert them back to their original form using JSON.parse().

```
const user = JSON.parse(localStorage.getItem('user'));
```

4. Removing Items
    a. To remove a specific item from Local Storage, use the removeItem method with the key.
    b. localStorage.removeItem('key');
    c. To clear all data stored in Local Storage, use the clear method.
    d. localStorage.clear();

Limitations and Considerations

1. Storage Limits: Local Storage is limited to about 5-10 MB per domain. This limit can vary between different browsers.

2. Synchronous API: Local Storage operations are synchronous, which means they can block the main thread and potentially affect the performance of your application.

3. Security: While Local Storage data is not transmitted to the server with every request like cookies, it’s still accessible through JavaScript. This means it's vulnerable to cross-site scripting (XSS) attacks if your site is susceptible to such vulnerabilities.

4. No Data Expiration: Data in Local Storage doesn’t expire. You need to manually manage when to clear it.

title: Implementing Local Storage Functionality

1. Creating a ticket involves storing them within local storage.
2. Updating a ticket, whether altering its color or task, entails updating these modified values in local storage.
3. Deleting a ticket results in its removal from local storage.
4. Whenever a ticket creation occurs, we have the option to store it in local storage using the "setItem" method. This involves providing the name of the array as a parameter. Here, during the code insertion within the createTicket method, we will employ the setItem function of localStorage

Code

```
const ticketsArr =
JSON.parse(localStorage.getItem('tickets')) || [];

// in the keydown event
ticketsArr.push({ ticketID, taskContent,
ticketColor:modalPriorityColor });
 localStorage.setItem('tickets',JSON.stringify(ticketsArr))
```

Retrieving data from localstorage

1. As we initiate the application, our first step is to retrieve all the tickets stored within local storage.

2. We create an init function to load the content from local storage

```
function init() {
 if (localStorage.getItem("tickets")) {
   ticketsArr.forEach(function (ticket) {
     createTicket(ticket.ticketColor, ticket.taskContent,
ticket.ticketID);
   });
 }
}

init();
```

3.

   a. Create updateLocalStorage method

```
function updateLocalStorage() {
 localStorage.setItem("tickets",
JSON.stringify(ticketsArr));
}
```

```

```

   b. Update the keydown function where tickets are generated

```
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
    ticketsArr.push({ ticketID, taskContent, ticketColor:
modalPriorityColor });
    updateLocalStorage();
  }
});
```

c. Code so far

```
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");
let modalPriorityColor = "black"; // Default to black
const removeBtn = document.querySelector(".remove-btn");
const allTickets = document.querySelectorAll(".ticket-cont");
const colors = ["lightpink", "lightgreen", "lightblue", "black"];
const toolboxColors = document.querySelectorAll(".color");
let ticketsArr = JSON.parse(localStorage.getItem("tickets")) || [];

let addTaskFlag = false;
let removeTaskFlag = false;
const lockClose = "fa-lock";
const lockOpen = "fa-lock-open";

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
   // 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
   ticketsArr.push({ ticketID, taskContent, ticketColor:
modalPriorityColor });
   updateLocalStorage();
 }
});

function init() {
 if (localStorage.getItem("tickets")) {
   ticketsArr.forEach(function (ticket) {
     createTicket(ticket.ticketColor, ticket.taskContent,
ticket.ticketID);
   });
 }
}

init();

function handleRemoval(ticket) {
 ticket.addEventListener("click", function () {
```

```
    if (!removeTaskFlag) return;
    else {
      ticket.remove();
    }
  });
}

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>
      <div class="ticket-lock">
        <i class="fa-solid fa-lock"></i>
      </div>
    `;
  // Append the ticket container to the main container
  mainCont.appendChild(ticketCont);
  handleRemoval(ticketCont);
  handleLock(ticketCont);
  handleColor(ticketCont);
  // ticketsArr.push({ ticketID, ticketTask, ticketColor });
  // localStorage.setItem("tickets", JSON.stringify(ticketsArr));
}

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];
  });
```

```
});

function handleLock(ticket) {
 const ticketLockElem = ticket.querySelector(".ticket-lock");
 const ticketLockIcon = ticketLockElem.children[0];
 const ticketTaskArea = ticket.querySelector(".task-area");

 ticketLockIcon.addEventListener("click", function () {
   console.log("Lock Selected");
   if (ticketLockIcon.classList.contains(lockClose)) {
     ticketLockIcon.classList.remove(lockClose);
     ticketLockIcon.classList.add(lockOpen);
     ticketTaskArea.setAttribute("contenteditable", "true"); // Changed
'contenteditable', 'true'
   } else {
     ticketLockIcon.classList.remove(lockOpen);
     ticketLockIcon.classList.add(lockClose);
     ticketTaskArea.setAttribute("contenteditable", "false"); // Changed
'contenteditable', 'true'
   }
 });
}

function handleColor(ticket) {
 const ticketColorBand = ticket.querySelector(".ticket-color");
 ticketColorBand.addEventListener("click", function () {
   // let currentColor = ticketColorBand.classList[0];
   const currentColor = ticketColorBand.style.backgroundColor;

   let currentColorIdx = colors.findIndex(function (color) {
     return currentColor === color;
   });

   currentColorIdx++; // Increment the index

   const newTicketColorIdx = currentColorIdx % colors.length;
   const newTicketColor = colors[newTicketColorIdx];
   ticketColorBand.style.backgroundColor = newTicketColor;
 });
}

toolboxColors.forEach(function (colorElem) {
```

```
 colorElem.addEventListener("click", function () {
   const selectedColor = colorElem.classList[0];
   const allTickets = document.querySelectorAll(".ticket-cont");
   allTickets.forEach(function (ticket) {
     const ticketColorBand = ticket.querySelector(".ticket-color");
     if (ticketColorBand.style.backgroundColor === selectedColor) {
       ticket.style.display = "block";
     } else {
       ticket.style.display = "none";
     }
   });
 });
 colorElem.addEventListener("dblclick", function () {
   const allTickets = document.querySelectorAll(".ticket-cont");
   allTickets.forEach(function (ticket) {
     ticket.style.display = "block";
   });
 });
});

function updateLocalStorage() {
 localStorage.setItem("tickets", JSON.stringify(ticketsArr));
}
```

Updating ticket details like task details and color in Local Storage

1. Create a function that returns the ticket index in the ticketArr based on ticketID

```
function getTicketIdx(id) {
 let ticketIdx = ticketsArr.findIndex(function(ticketObj) {
     return ticketObj.ticketID === id;
 });
 return ticketIdx;
```

```
}
```

2. Within the handleLock function, whenever a lock is unlocked through a click event, we will retrieve the id of the ticjet so we can make changes
3. Code

```
function handleLock(ticket) {
 const ticketLockElem = ticket.querySelector(".ticket-lock");
 const ticketLockIcon = ticketLockElem.children[0];
 const ticketTaskArea = ticket.querySelector(".task-area");
 const id = ticket.querySelector(".ticket-id").innerText;

 ticketLockIcon.addEventListener("click", function () {
   console.log("Lock Selected");
   const ticketIdx = getTicketIdx(id);
   if (ticketLockIcon.classList.contains(lockClose)) {
     ticketLockIcon.classList.remove(lockClose);
     ticketLockIcon.classList.add(lockOpen);
     ticketTaskArea.setAttribute("contenteditable", "true"); // Changed
'contenteditable', 'true'
   } else {
     ticketLockIcon.classList.remove(lockOpen);
     ticketLockIcon.classList.add(lockClose);
     ticketTaskArea.setAttribute("contenteditable", "false"); // Changed
'contenteditable', 'true'
   }
   ticketsArr[ticketIdx].taskContent = ticketTaskArea.innerText;
   updateLocalStorage();

 });
}
```

Updating color change in localstorage

```
function handleColor(ticket) {
 const ticketColorBand = ticket.querySelector(".ticket-color");
 const id = ticket.querySelector(".ticket-id").innerText;
 ticketColorBand.addEventListener("click", function () {
   // let currentColor = ticketColorBand.classList[0];
   const ticketIdx = getTicketIdx(id);
   const currentColor = ticketColorBand.style.backgroundColor;

   let currentColorIdx = colors.findIndex(function (color) {
     return currentColor === color;
   });

   currentColorIdx++; // Increment the index

   const newTicketColorIdx = currentColorIdx % colors.length;
   const newTicketColor = colors[newTicketColorIdx];
   ticketColorBand.style.backgroundColor = newTicketColor;
   ticketsArr[ticketIdx].ticketColor = newTicketColor;
   updateLocalStorage();
 });
}
```

Removing ticket from local storage

```
function handleRemoval(ticket) {
 const id = ticket.querySelector(".ticket-id").innerText;

 ticket.addEventListener("click", function () {
   if (!removeTaskFlag) return;
   else {
     ticket.remove();
     const ticketIdx = getTicketIdx(id);
      ticketsArr.splice(ticketIdx, 1);
      updateLocalStorage();
   }
```

```
 });
}
```

Deploying the project to Giithub Pages

1. Create a GitHub Repository: Go to GitHub, create a new repository for your project.
2. Prepare Your Project: Make sure your project has an index.html at the root.
3. Push Your Project to GitHub:
    a. Initialize your project folder as a Git repository (if not already) using git init.
    b. Add the GitHub repository as a remote with git remote add origin https://github.com/<username>/<repository-name>.git.
    c. Add your project files to the staging area with git add ..
    d. Commit the changes with git commit -m "Initial commit".
    e. Push the project to GitHub with git push -u origin master (use main instead of master if your default branch is main).
4. Enable GitHub Pages:
    a. Go to your repository on GitHub.
    b. Navigate to "Settings" > "Pages".
    c. Select the branch you want to deploy from, usually master or main.
    d. Click "Save".

5. Your project is now live, and you can access it via https://<username>.github.io/<repository-name>/.
6. https://ayushrajsd.github.io/kanban/