Kanban-tavla

Av Mellissa Qholizadeh, student ID: meqh4654

Sammanfattning

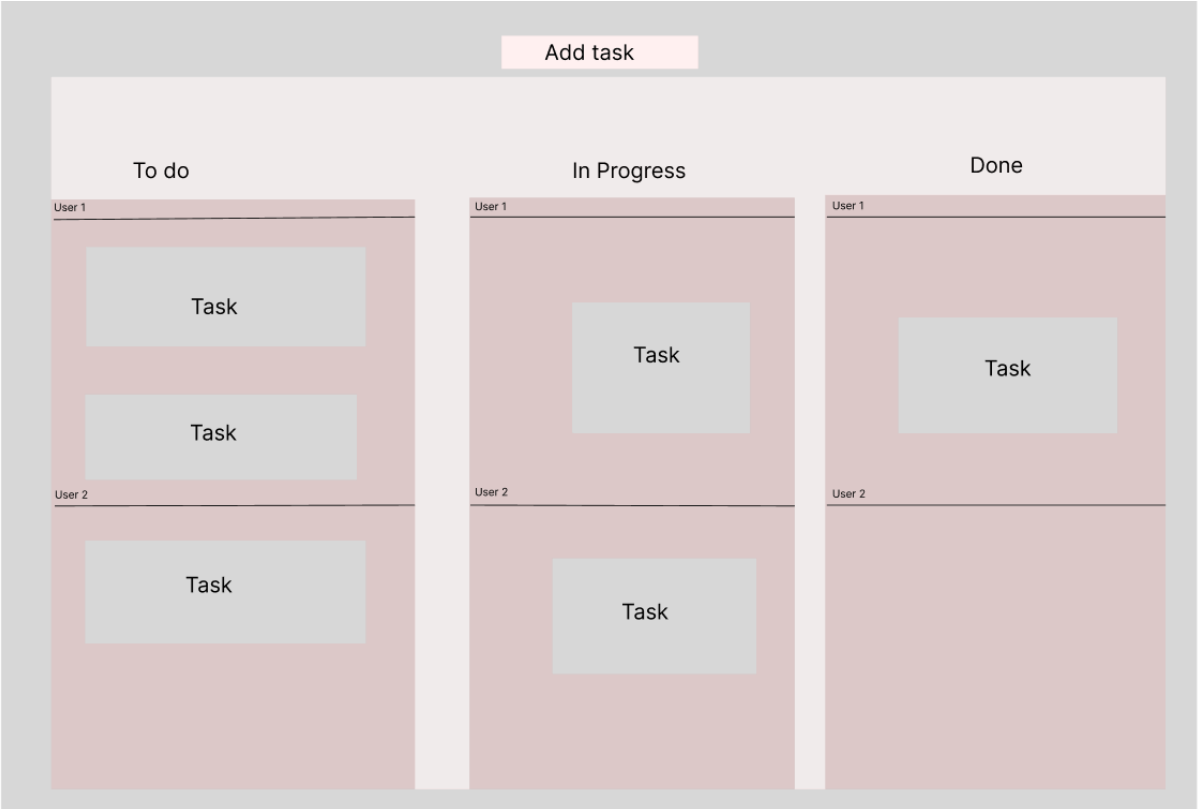
Den här hemsidan som jag har skapat ska föreställa en form av kanban-tavla. Det fungerar så att användaren kan skapa olika “tasks”, på svenska; uppgifter. Det finns tre olika stadier där en uppgift kan vara : To do, In progress och done. Användaren kan skapa nya uppgifter genom att trycka på en knapp, sedan ska de fylla i titel, ägare och beskrivning av uppgiften. Sedan är det möjligt att flytta uppgiften mellan kolumnerna med hjälp av två pilar på varje uppgift.

Framtagande

Idé samling och skiss

Idén till att göra denna webbsida hade sin grund i den frivilliga uppgiften nummer 3.2.7, där jag använde mig av olika jQuery UI:s. Då fick jag idén att göra något med drag & drop-funktion. Jag ville även göra något användbart för mig själv, inte göra något bara för att. Till en början tänkte jag att det skulle bli en hemsida där man kan skapa ett schema, men jag använder mig redan av Google Kalender flitigt. Då fick jag idén att göra en form av kanban-tavla.

Jag googlade på hur en kanban-tavla kan se ut och utifrån det skapade jag en skiss i figma:



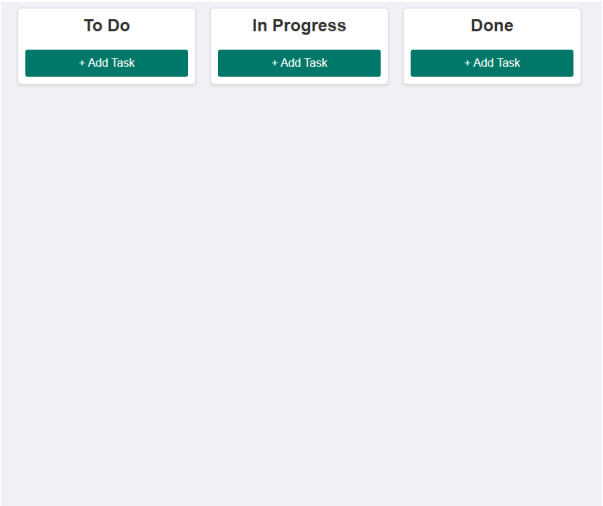
Tanken var alltså från början att användaren skulle ha tre olika kolumner: To do, In Progress och Done. Därefter var tanken att varje task/ uppgift skulle kunna förflyttas med drag & drop funktionalitet och att användaren skulle kunna skapa rader som tillhörde olika användare.

För att få upp en plan på vad som bör göras skapade jag en form av överblickande planering på vad som ska vara med, den såg ut ungefär följande:
1: Skapa tre olika kolumner
2: Skapa ett kort som ska fungera som en task/ uppgift
3: Gör en knapp som kan skapa denna uppgift
4: När de skapas ska de hamna i To do kolumnen
5: Skapa ett formulär där användaren kan mata in vad som ska visas på kortet
6: Flytta task mellan olika kolumner (med pilar till en början)
7: Ta bort uppgifter
8: Spara data i local-storage så att användaren kan ladda om sidan utan att alla uppgifter försvinner
9: Skapa rader till användaren
10: Implementera drag & drop

Under denna plan påbörjades arbetet och det var en konstant vägledning under arbetets gång. Slutprodukten blev olik denna plan, men på det stora hela fungerade detta som en bra indikator på vad som skulle göras.

Start av projekt

Det första som jag ville göra var att skapa en grundstruktur med hjälp av HTML och CSS. Tanken var, som tidigare nämnt, att det skulle finnas tre pelare och det var relativt simpelt att implementera. Det första resultatet blev detta:



Men jag märkte att om användaren ska kunna ha användning för denna webbsida på exempelvis en mobiltelefon eller surfplatta, behöver denna layout kunna ändras responsivt. Därav implementerade jag en layout för 1024px och under i bredd, som innebar att användaren bara skulle se en kolumn i taget över hela skärmen. Med hjälp av lite javascript och CSS gick det att implementera. I Javascript implementerade jag två olika funktioner, en som visade alla kolumner och en som visade en kolumn. Därefter kunde jag med hjälp av en

"resize"-event lyssnare känna av när bredden på skärmen ändras, för att då anropa de två olika funktionerna. Denna funktionalitet kräver även att användaren ska kunna navigera mellan kolumnerna, det gjorde jag med hjälp av två pilar och sen två button event som adderar eller subtraherar värde till ett index som håller koll på vilken kolumn det är.

Lägg till kort

Nästa steg var att skapa ett kort som ska användas som uppgifter senare. Men till en början ville jag bara få till funktionaliteten av att ett kort ska skapas i en kolumn. Det var enkelt att göra med bara HTML och CSS, jag ville fixa hur kortet skulle se ut innan det i ett senare skede ska skapas dynamiskt. Här märkte jag också att all text i beskrivningen i kortet inte får plats på skärmen när det skapas, så jag implementerade en toggle text som heter show more eller show less, för att kunna expandera och minimera beskrivningstexten. När jag fick det att fungera implementerade jag även en knapp på kortet som hette “delete” och kan ta bort just det kortet.
Till nu hade jag en knapp i varje kolumn för att lägga till uppgifter, men jag bestämde mig för att endast ha en knapp centralt där alla nya uppgifter ska komma till To Do kolumnen.

Lägg till kort dynamiskt

Fortsättningen på kort/ uppgift funktionaliteten är att användaren ska kunna skapa flera kort och fylla i de med information som ska visas. Det går att uppfylla med hjälp av ett formulär som kommer upp när användaren trycker på knappen “add task”.
För att göra detta skapade jag en funktion som kallas för createTaskCard, som använder sig av innerHTML för att skapa användarinput och visa det nya kortet. När användaren fyller i uppgifterna i formuläret så skapas det i To Do kolumnen, sedan går det att skapa flera kort, som alla läggs till där.

I detta skede implementerade jag även ett sätt att kunna flytta korten mellan kolumnerna, till en början tänkte jag att det skulle bli lättast att implementera med hjälp av pilar på korten. Det var relativt simpelt att implementera med hjälp av klass och id-element från HTML, för att hålla koll på vilka kolumner som är vad. Sedan kopplade jag ett knapptryck-event som flyttade kortet till nästa kolumn eller den förra kolumnen, beroende på vilken pil användaren klickar på.

Jag märkte även här att jag ville ha ett sätt att kunna ta bort ett kort, då jag tills nu behövde ladda om sidan för att ta bort ett kort. Jag löste detta genom att implementera en delete-knapp som tar bort kortet från newTaskCard- listan.

Bonus ändringar & problem

I detta skede kände jag mig relativt nöjd, jag kunde skapa kort, mata in information i dem, flytta runt korten mellan olika kolumner som ska representera olika stadier av en uppgift. Det

nästa steget var att implementera någon form av lokalt sparande av data, så att jag faktiskt kan använda denna webbapplikation, som mitt mål med detta projekt var. Så jag implementerade en saveTasksToLocalStorage metod. Den går ut på att lägga till alla uppgifternas data och var de befinner sig i en array, för att sedan lagra data i webbläsarens Local Storage. Arrayen konverteras till en JSON-sträng, eftersom Local Storage bara kan lagra text. Denna funktion anropas vid olika tillfällen under mitt programs körning, såsom när användaren skapar en ny uppgift, eller flyttar en uppgift till en annan kolumn. Det behövdes även att skapa en Load-funktion som hämtar den nedsparade data som uppstod i save-funktionen.

Därefter ville jag ge mig på drag & drop funktionaliteten som var anledningen till att jag kom på denna idé till projekt. Då använde jag mig av jQuery och dess event draggable och droppable. Själva funktionaliteten var simpel, men att få in detta med all annan kod gick inte så bra. Jag märkte att när jag skulle flytta korten mellan kolumnerna så kunde de “flyga iväg” eller försvinna helt, det var svårt att få dem att fastna i kolumnerna.
Så tillslut efter mycket om och men så valde jag att slumpa hela drag & drop funktionaliteten, jag insåg att det tog upp för mycket tid. Som jag tidigare nämnde tyckte jag vid det här laget att allt fungerade, det skulle bara vara en bonus med drag & drop.

I och med att jag inte fick drag & drop att fungera så behövde jag även slumpa mitt tänkta sätt att kunna dela in uppgifter i olika rader, som skulle tillhöra olika användare. Därav behövde jag anpassa denna funktionalitet och jag höll det väldigt simpelt, med att lägga till ett till textfält i mina kort, owner textfältet. Tanken med det textfältet är att flera användare ska kunna använda sidan och för att förtydliga vem som ska göra vad så kan man skriva en användares namn på kortet.

Utökad responsiv layout

När allting nu kändes klart ville jag förfina min responsiva layout, vid det här laget hade jag bara två olika vyer, en för datorskärmar och en för mobiler/ plattor. Men jag märkte att det skulle vara bra om jag implementerade lite mer layouter som i den första CSS uppgiften. Det jag gjorde var att kolla på hur jag tacklade uppgiften, jämförde med vad som kunde göras med mitt projekt. Resultatet blev att jag har en layout för skärmbredd 206px och under, mobilskärmar upp till 600 px, sulfplattor/ mindre datorskärmar mellan 600 och 1024px och till sist för datorskärmar. Det var det sista som jag gjorde i detta projekt!

Form

Column hantering (Javascript)

För att hantera den responsiva layouten och på det sättet jag ville ha det på (Större skärm 3 kolumner, mindre skärm en kolumn i taget) behövde jag ta hjälp av lite javascript. Som jag

tidigare nämnde använde jag mig av eventet resize på detta sätt:

```
window.addEventListener('resize', () => {
    if (window.innerWidth >= 1024) {
        showAllColumns();
    } else {
        showColumn(currentIndex);
    }
});

if (window.innerWidth >= 1024) {
    showAllColumns();
} else {
    showColumn(currentIndex);
}
```

```
function showColumn(index) {
    columns.forEach((col, i) => {
        col.style.display = i === index ? 'block' : 'none';
    });
}
function showAllColumns() {
    columns.forEach(col => {
        col.style.display = 'block';
    });
}
```

När användaren kommer till vyn med en kolumn dyker det även upp två pilar för att navigera mellan de olika kolumnerna, den koden är också i denna fil och består endast av ett index som adderas för next och subtraheras för prev.

Uppgiftshantering (Javascript)

Den resterande javascript-koden befinner sig i en fil som heter board.js och den består främst av kort/ uppgifts - hanteringen i projektet. Den stora funktionen i denna fil är skapandet av en ny task, funktionen heter createTaskCard och skapar en ny uppgift med hjälp av innerHTML:

```
newTaskCard.innerHTML = `
    <div class="task-controls">
    <p><strong>Owner: ${ownerText}</strong></p>
        <button class="move-left">←</button>
        <button class="move-right">→</button>
    </div>

    <h3>${title}</h3>
    <div class="description-container">
        <p class="description">${descriptionText}</p>
        <button class="toggle-description">Show more</button>
    </div>

    <button class="delete-task">Delete</button>
`;

const column = document.getElementById(columnId).querySelector('
column.appendChild(newTaskCard);
```

Användaren fyller dessa element med hjälp av detta “submit” event:

```
taskForm.addEventListener('submit', (event) => {
    event.preventDefault();
    const title = taskTitleInput.value;
    const description = taskDescriptionInput.value;
    const owner = taskOwnerInput.value;
    createTaskCard(title, description, owner);
    taskTitleInput.value = '';
    taskDescriptionInput.value = '';
    taskOwnerInput.value = '';
    modal.style.display = 'none';
    saveTasksToLocalStorage();
});
```

Efter data har skickats till createTaskCard med de inmatade parametrarna så nollställs värdet i varje input och sedan försvinner formuläret igen, tills användaren trycker på knappen add task.

createTaskCard funktionen är även ansvarig för att skapa funktionaliteten för exempelvis toggle-knappen som visar mer text beroende på hur mycket text det finns i “description” rutan, och pilarna för att flytta korten mellan kolumnerna:

```
const newToggleButton = newTaskCard.querySelector('.toggle-description');
const newDescription = newTaskCard.querySelector('.description');

if (newDescription.scrollHeight > 70) {
    newToggleButton.style.display = 'block';
} else {
    newToggleButton.style.display = 'none';
}
newToggleButton.addEventListener('click', () => {
    newDescription.classList.toggle('expanded');
    newToggleButton.textContent = newDescription.classList.contains('expanded') ? 'Show less' : 'Show more';
});
```

```
const moveLeftButton = newTaskCard.querySelector('.move-left');
const moveRightButton = newTaskCard.querySelector('.move-right');

moveLeftButton.addEventListener('click', () => {
    const currentColumn = newTaskCard.parentElement;
    const prevColumn = currentColumn.closest('.column').previousElementSibling?.querySelector('.task-list');
    if (prevColumn) {
        prevColumn.appendChild(newTaskCard)
        saveTasksToLocalStorage();
    };
});
moveRightButton.addEventListener('click', () => {
    const currentColumn = newTaskCard.parentElement;
    const nextColumn = currentColumn.closest('.column').nextElementSibling?.querySelector('.task-list');
    if (nextColumn) {
        nextColumn.appendChild(newTaskCard)
        saveTasksToLocalStorage();
    };
});
```

Denna Javascript fil innehåller koden för att kunna spara och ladda den data som användaren har skapat localt i form av en JSON-fil. Den koden fungerar genom att skapa en array som tar emot alla befintliga korts data och sedan sparas ned localt med hjälp av att convertera det till en string, för att spara det som en JSON. Sedan för att ladda det som har sparats hämtas det sparade tillståndet med hjälp av att analysera (parse) den redan skapta JSON - filen. Dessa funktioner ser ut så här:

```
function saveTasksToLocalStorage() {
const tasks = [];
columns.forEach(column => {
    const tasksInColumn = Array.from(column.querySelectorAll('.task-card')).map(task => ({
        title: task.querySelector('h3').textContent,
        description: task.querySelector('.description').textContent,
        owner: task.querySelector('p strong').nextSibling.textContent.trim(),
        column: column.closest('.column').id
    }));
    tasks.push(...tasksInColumn);
});
localStorage.setItem('kanbanTasks', JSON.stringify(tasks));
}
```

```
function loadTasksFromLocalStorage() {
const savedTasks = JSON.parse(localStorage.getItem('kanbanTasks')) || [];
savedTasks.forEach(task => {
    createTaskCard(task.title, task.description, task.owner, task.column);
});
}
```

HTML & CSS

HTML och CSS:n är relativ simpel men här är några exempel på hur det ser ut:

```
<main class="kanban-container">
    <button class="nav-button" id="prev">
        ←
        <span class="nav-label" id="prev-label">prev</span>
    </button>

    <section class="column" id="todo">
        <h2>To Do</h2>
        <article class="task-list">
        </article>
    </section>
    <section class="column" id="in-progress">
        <h2>In Progress</h2>
        <article class="task-list">
        </article>
    </section>
    <section class="column" id="done">
        <h2>Done</h2>
        <article class="task-list">
        </article>
    </section>

    <button class="nav-button" id="next">
        →
        <span class="nav-label" id="next-label">next</span>
    </button>
</main>
ody>
```

Det här är HTML:en för kolumnerna på webbsidan, nästan varje element är indelade i klasser och id för att lätt kunna hämta rätt element i Javascript och CSS.

Ett exempel på en del av CSS:en:

```
.add-task {
    display: flex;
    margin-top: 10px;
    padding: 10px;
    background-color: #00796b;
    color: white;
    text-align: center;
    justify-content: center;
    border-radius: 3px;
    cursor: pointer;
    font-size: 0.6em;
    transition: background-color 0.3s ease;
}

.add-task:hover {
    background-color: #004d40;
}
```

Display flex används i många element och är till för att kunna använda funktioner som text-align och justify-content, för att exempelvis kunna centrera ett element på sidan. Border Radius används för att runda kanterna. Transition är till för att få en smidig animering med hover effekten på knapped.

Pseudo-kod

Pseudo-kod från next_previous_column.js:

```
FUNKTION showColumn(index)
      För varje kolumn col med index [i] i columns:
         Om i == index
              Sätt col synlig (display = ‘block’)
         Annars
              Sätt col osynlig (display = ‘none’)

FUNKTION showAllColumns()
     För varje kolumn col i columns:
        Sätt col synlig (display = ‘block’)

EVENT Fönsterstorlek ändras
    Om fönsterbredd >= 1024
        showAllColumns()
    Annars
       showColumn(currentIndex)




VID SIDLADDNING:
     Om fönsterbredd >= 1024
          showAllColumns()
     Annars
          showColumn(currentIndex)
```

Pseudo-kod från board.js:

```
FUNKTION createTaskCard(title, desciptionText, ownerText, columnId = ‘todo’)
     Skapa nytt element newTaskCard med klassen ‘task-card’
     Sätt position för newTaskCard till ‘relative’
     Skapa kortets HTML-innehåll med:
              - Ägare (ownerText)
              - Knapp: Flytta vänster
              - Knapp: Flytta höger
              - Titel (title)
              - Beskrivning (descriptionText)
              - Knapp: Visa mer/mindre
              - Knapp: Ta bort
    Hämta kolumn baserat på columnId
    Lägg till newTaskCard i den kolumnen
    Kör: saveTasksToLocalStorage()
```

Om beskrivningen är längre än 70 pixlar:
Visa knappen ‘show more’
Annars:
Dölj knappen ‘show more’

EVENT När ‘show more’ klickas
Växla om beskrivningen är utökad eller ej
Uppdatera knapptitel baserat på om beskrivningen är utökad

EVENT När ‘move-right/ move-left’ klickas
Hämta nuvarande kolumn
Hämta föregående/ nästa kolumn
Om föregående/ nästa kolumn existerar
Flytta kortet till föregående/ nästa kolumn
Kör: saveTasksToLocalStorage()

EVENT När ‘delete-task’ klickas
Ta bort kortet från DOM
Kör: saveTasksToLocalStorage()

FUNKTION saveTasksToLocalStorage()
Skapa en tom lista: tasks
För varje kolumn i columns:
Hämta alla uppgifter i kolumnen
Skapa en lista av uppgifter med:
-Titel
- beskrivning
- ägare
- kolumn-ID
Lägg till dessa uppgifter i tasks
Spara tasks i localStorage som JSON-sträng

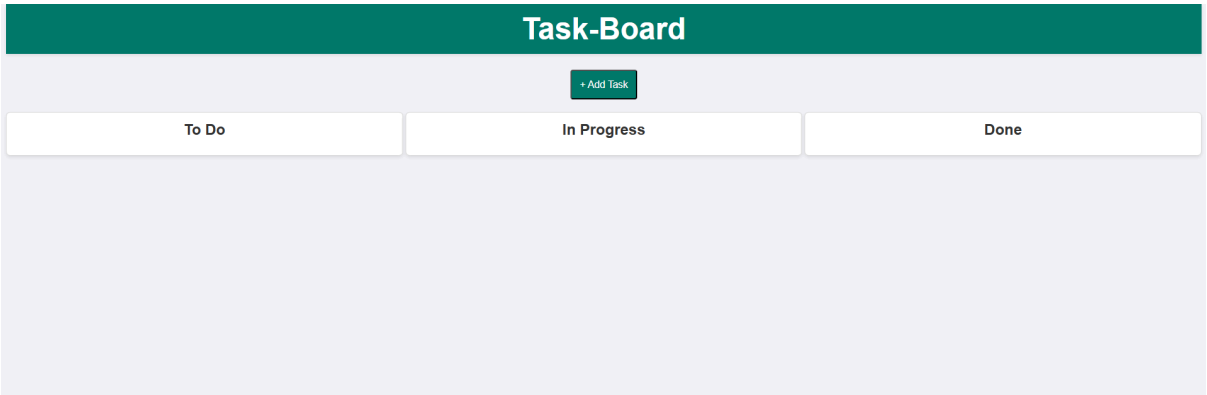
FUNKTION loadTasksFromLocalStorage()
Hämta sparade uppgifter från localStorage
Om det finns sparade uppgifter:
För varje sparad uppgift:
Kör: createTaskCard() med uppgiftens data

EVENT När formuläret skickas
Förhindra standardformulärhantering
Hämta titel, beskrivning och ägare från formuläret
Kör: createTaskCard() med inmatad data
Töm formulärfält

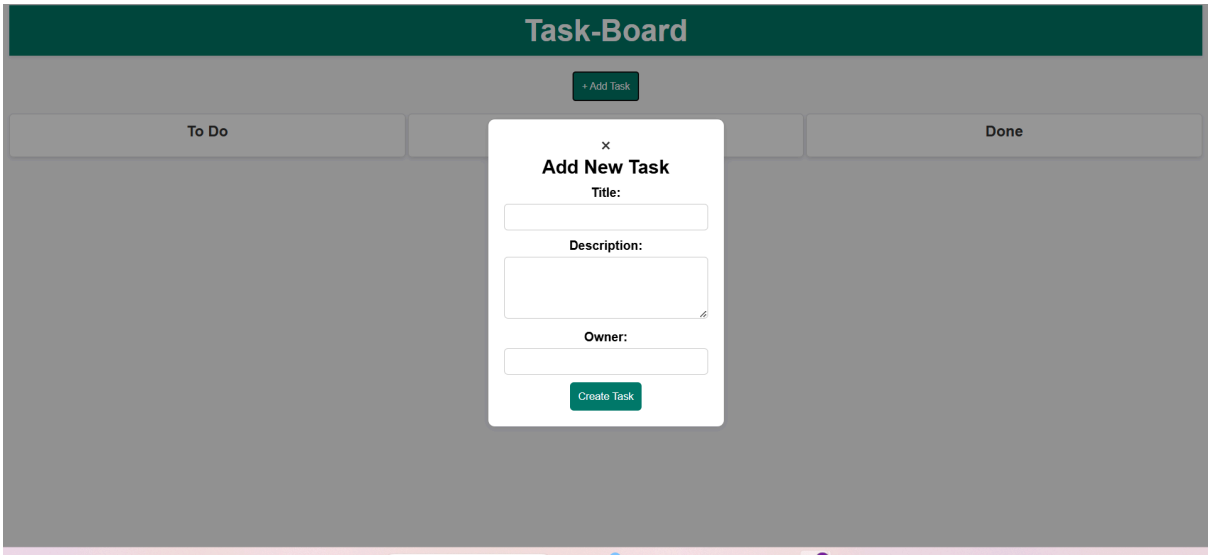
Dölj modal-fönstret
Kör: saveTasksToLocalStorage()

Funktion

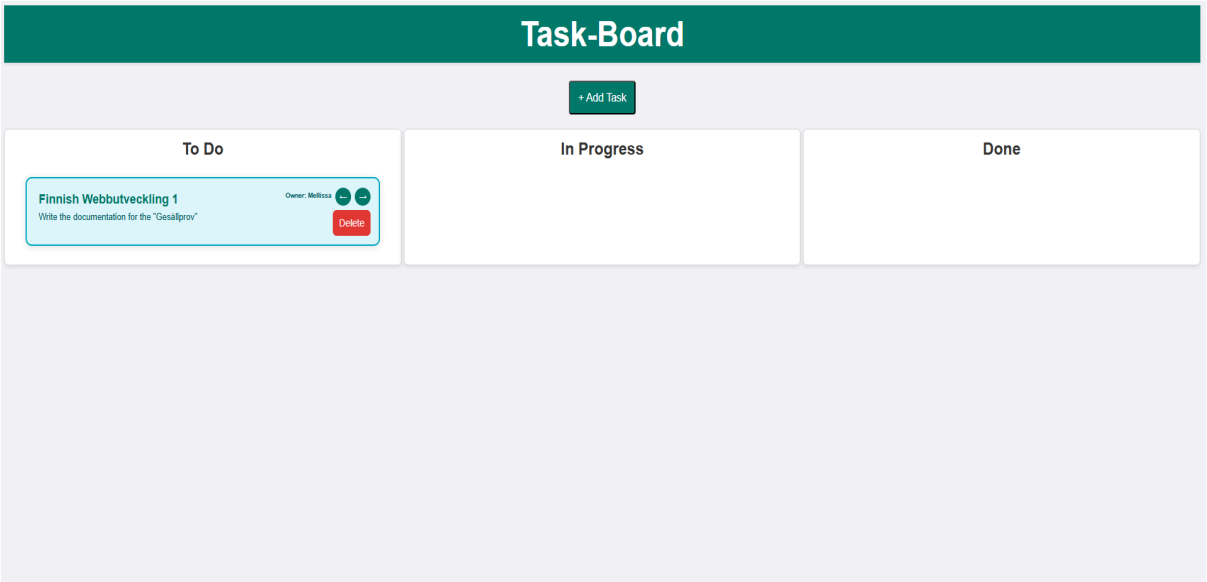
När användaren startar webbsidan är det detta som kommer upp först:



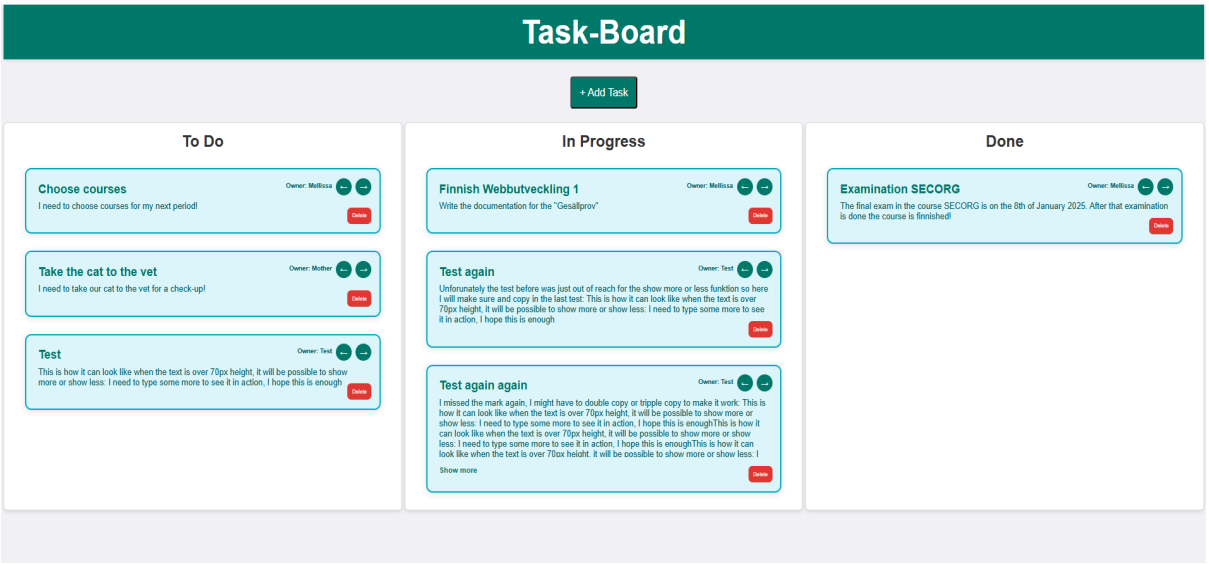
Det användare kan göra är att trycka på knappen Add Task, då kommer det upp ett formulär som ser ut så här:



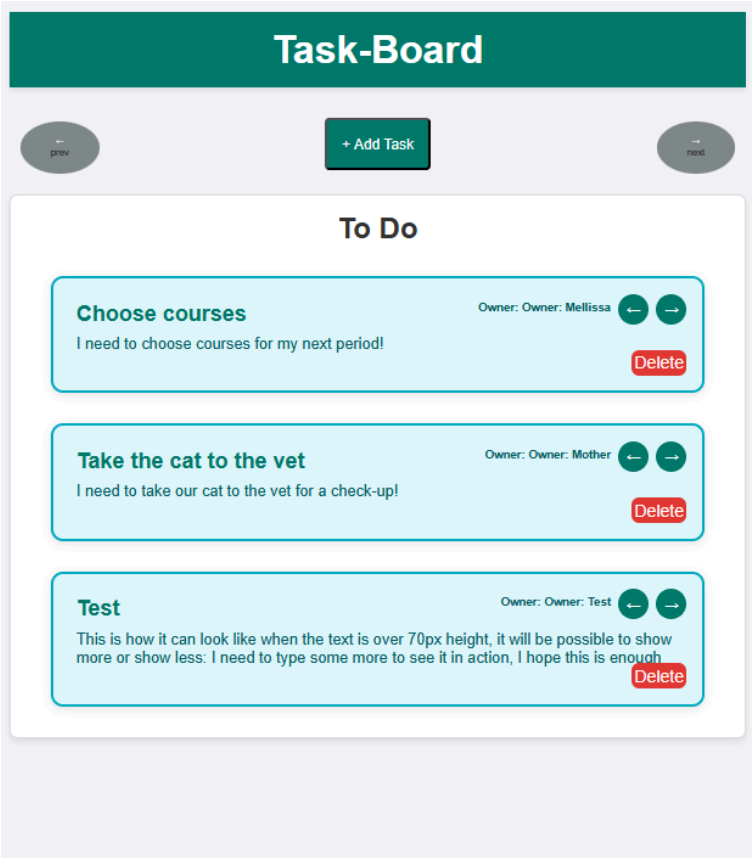
Där kan användaren mata in titeln, beskrivningen och ägaren som sedan ska fylla ut ett kort och användas som en “task”. När användaren trycker på Create Task knappen kommer det att se ut så här:



Korten som användare skapar hamnar i kolumnen To Do. Innehållet i varje kort består av titel, beskrivning, ägare, navigeringspilar och en delete-knapp. Med de två pilarna som finns i ett kort kan man flytta runt uppgiften mellan de olika kolumnerna. Användaren kan även välja att ta bort uppgiften om den är färdig eller felaktig med delete-knappen. När fler uppgifter har skapats kan det se ut på följande sätt:



Det är det stora hela, tanken är som sagt att denna webbsida ska fungera som en kanban tavla, där användaren kan hålla koll på olika uppgifter som behöver göras. En uppgift kan befinna sig i tre olika stadier: Att göra, pågående och klar. Så här ser hemsidan ut i mobil anpassad layout:



För en surfplatta är det väldigt likt, det är bara lite små ändringar som storleken på texten på knapparna och liknande:

Task-Board

← prev

+ Add Task

→ next

In Progress

Finnish Webbutveckling 1

Owner: Owner: Mellissa ← →

Write the documentation for the "Gesällprov"

Delete

Test again

Owner: Owner: Test ← →

Unforunately the test before was just out of reach for the show more or less funktion so here I will make sure and copy in the last test: This is how it can look like when the text is over 70px height, it will be possible to show more or show less: I need to type some more to see it in action, I hope this is enough

Delete

Test again again

Owner: Owner: Test ← →

I missed the mark again, I might have to double copy or tripple copy to make it work: This is how it can look like when the text is over 70px height, it will be possible to show more or show less: I need to type some more to see it in action, I hope this is enoughThis is how it can look like when the text is over 70px height, it will be possible to show more or show less: I

Delete