Background Story

It was a busy Monday morning at a co-working space in Bonifacio Global City. Odessa was sipping her kapeng barako ☕ while watching freelancers shuffle sticky notes on whiteboards. One group had a Kanban board drawn on a roll of kraft paper—"To Do," "In Progress," "Done." But every time someone moved a note, they had to peel and re-stick it, leaving glue marks like EDSA traffic jams.

Odessa thought, "What if I make this digital, drag-and-drop style, like Trello?" 🚀 She imagined tasks you could grab with a mouse, drag from column to column, and drop with a satisfying animation. This feature would delight clients and speed up workflows—perfect for her logistics startup's internal tool.

She sketched the UI: three columns (`#todo`, `#inProgress`, `#done`), each a `<div>` with tasks inside. Each task card would be a `<div draggable="true">`. With JavaScript's `dragstart`, `dragover`, `drop`, and `dragend` events, she could wire up the logic. For a smooth feel, she'd add a CSS class during dragging—cards would scale up slightly and cast a shadow.

By afternoon, her prototype was live. She dragged a "Pack orders" card from "To Do" into "In Progress," watching it glide and snap into place. Fellow devs gathered around her laptop, applauding the smooth UX. Clients started talking about adopting her Kanban board—and Odessa realized this drag-and-drop feature might be the spark for her next startup pitch.

Today, you'll learn how to build that interactive task board: handling drag events, combining CSS animations with JS, and creating reusable code. Let's drag it, drop it, own it! 🎉

Theory & Lecture Content

1. Drag Events Overview

- **dragstart**: Fires when dragging begins.
- **dragover**: Fires when a dragged element is over a drop target—you must `event.preventDefault()` to allow drop.
- **drop**: Fires when the dragged element is released over a target.
- **dragend**: Fires when dragging ends (useful for cleanup).

2. Basic HTML Structure

```
<div class="column" id="todo">
  <h2>To Do</h2>
  <!-- task cards go here -->
</div>
<div class="column" id="inProgress">
  <h2>In Progress</h2>
</div>
<div class="column" id="done">
  <h2>Done</h2>
</div>
```

Each `.column` is a drop zone; each task card is `draggable`.

3. JavaScript APIs

allowDrop(event)

```
function allowDrop(evt) {
  evt.preventDefault(); // enable dropping
}
```

drag(event)

```
function drag(evt) {
  evt.dataTransfer.setData("text/plain", evt.target.id);
  // optional: add CSS class for visual cue
  evt.target.classList.add("dragging");
}
```

drop(event)

```
function drop(evt) {
  evt.preventDefault();
  const id = evt.dataTransfer.getData("text/plain");
  const card = document.getElementById(id);
  evt.target.appendChild(card);
  card.classList.remove("dragging");
}
```

dragend(event)

```
function dragEnd(evt) {
  evt.target.classList.remove("dragging");
}
```

4. CSS/Animation Combo

```
.column {
  border: 2px dashed #ccc;
  padding: 10px;
  min-height: 200px;
  transition: background 0.3s;
}
```

```
.column.over {
  background: #f0f8ff;
}

.task-card {
  padding: 8px;
  margin: 5px 0;
  background: #fff;
  border-radius: 4px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.2);
  transition: transform 0.2s, box-shadow 0.2s;
}

.task-card.dragging {
  opacity: 0.7;
  transform: scale(1.05);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
}
```

5. Wiring It Up

```
<script>
  const cols = document.querySelectorAll(".column");
  cols.forEach((col) => {
    col.addEventListener("dragover", allowDrop);
    col.addEventListener("drop", drop);
    col.addEventListener("dragenter", () => col.classList.add("over"));
    col.addEventListener("dragleave", () => col.classList.remove("over"));
  });

  const cards = document.querySelectorAll(".task-card");
  cards.forEach((card) => {
    card.addEventListener("dragstart", drag);
    card.addEventListener("dragend", dragEnd);
  });
</script>
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API

Exercises

Exercise 1: Basic Drag & Drop

Problem Statement
Implement the core drag-and-drop functions in `dragDrop.js`:

- allowDrop(evt)
- drag(evt)

- drop(evt)
- dragEnd(evt)

Make sure cards can be moved between columns.

TODOs

1. In `dragDrop.js`, export the four functions as described.
2. In `index1.html`, wire up events on `.column` and `.task-card`.

Starter Code (dragDrop.js)

```
// dragDrop.js

export function allowDrop(evt) {
  // TODO
}

export function drag(evt) {
  // TODO
}

export function drop(evt) {
  // TODO
}

export function dragEnd(evt) {
  // TODO
}
```

Full Solution (dragDrop.js)

```
export function allowDrop(evt) {
  evt.preventDefault();
}

export function drag(evt) {
  evt.dataTransfer.setData("text/plain", evt.target.id);
  evt.target.classList.add("dragging");
}

export function drop(evt) {
  evt.preventDefault();
  const id = evt.dataTransfer.getData("text/plain");
  const card = document.getElementById(id);
  // if dropping on inner element, find .column parent
  const col = evt.target.closest(".column");
  col.appendChild(card);
  card.classList.remove("dragging");
```

```
}

export function dragEnd(evt) {
  evt.target.classList.remove("dragging");
}
```

Demonstration HTML (index1.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Drag & Drop</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <div class="column" id="todo">
      <h2>To Do</h2>
      <div class="task-card" id="task1" draggable="true">Task 1</div>
      <div class="task-card" id="task2" draggable="true">Task 2</div>
    </div>
    <div class="column" id="inProgress">
      <h2>In Progress</h2>
    </div>
    <div class="column" id="done">
      <h2>Done</h2>
    </div>

    <script type="module">
      import { allowDrop, drag, drop, dragEnd } from "./dragDrop.js";

      document.querySelectorAll(".column").forEach((col) => {
        col.addEventListener("dragover", allowDrop);
        col.addEventListener("drop", drop);
        col.addEventListener("dragenter", () => col.classList.add("over"));
        col.addEventListener("dragleave", () => col.classList.remove("over"));
      });

      document.querySelectorAll(".task-card").forEach((card) => {
        card.addEventListener("dragstart", drag);
        card.addEventListener("dragend", dragEnd);
      });
    </script>
  </body>
</html>
```

Exercise 2: Dynamic Task Card Creator

Problem Statement
Write `createTaskCard(task)` in `task.js` that:

1. Creates a `<div>` with class `"task-card"` and `draggable="true"`.
2. Sets `id` to `task.id`.
3. Sets its text content to `task.text`.
4. Attaches `dragstart` and `dragend` listeners using the `dragDrop` module.
5. Returns the card element.

TODOs

- Import `drag` and `dragEnd` from `dragDrop.js`.
- Implement `createTaskCard` in `task.js`.

Starter Code (task.js)

```
// task.js
import { drag, dragEnd } from "./dragDrop.js";

/**
 * @param {{id:string, text:string}} task
 * @returns {HTMLElement}
 */
export function createTaskCard(task) {
  // TODO
}
```

Full Solution (task.js)

```
import { drag, dragEnd } from "./dragDrop.js";

export function createTaskCard(task) {
  const card = document.createElement("div");
  card.className = "task-card";
  card.draggable = true;
  card.id = task.id;
  card.textContent = task.text;

  card.addEventListener("dragstart", drag);
  card.addEventListener("dragend", dragEnd);

  return card;
}
```

Demonstration HTML (index2.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Dynamic Cards</title>
  </head>
  <body>
    <div class="column" id="todo">
      <h2>To Do</h2>
    </div>
    <script type="module">
      import { createTaskCard } from "./task.js";
      import { allowDrop, drop } from "./dragDrop.js";

      const todo = document.getElementById("todo");
      todo.addEventListener("dragover", allowDrop);
      todo.addEventListener("drop", drop);

      const tasks = [
        { id: "t1", text: "Design logo" },
        { id: "t2", text: "Write content" },
      ];
      tasks.forEach((t) => todo.appendChild(createTaskCard(t)));
    </script>
  </body>
</html>
```

Exercise 3: Fade-In Animation

Problem Statement
Create `animate.js` with `animateCard(card)` that:

1. Adds class `"fade-in"` to `card`.
2. After 500ms, removes `"fade-in"`.
3. Returns nothing.

TODOs

- Implement `animateCard` using `setTimeout`.
- Include CSS for `.fade-in` animation in HTML.

Starter Code (animate.js)

```
// animate.js

/**
 * Applies fade-in animation to a card.
 * @param {HTMLElement} card
 */
```

```
export function animateCard(card) {
  // TODO
}
```

Full Solution (animate.js)

```
export function animateCard(card) {
  card.classList.add("fade-in");
  setTimeout(() => {
    card.classList.remove("fade-in");
  }, 500);
}
```

Demonstration HTML (index3.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Animate Cards</title>
    <style>
      .fade-in {
        animation: fadeIn 0.5s ease-in;
      }
      @keyframes fadeIn {
        from {
          opacity: 0;
          transform: scale(0.95);
        }
        to {
          opacity: 1;
          transform: scale(1);
        }
      }
      .task-card {
        background: #fff;
        padding: 8px;
        margin: 5px;
      }
    </style>
  </head>
  <body>
    <div id="todo"></div>
    <script type="module">
      import { createTaskCard } from "./task.js";
      import { allowDrop, drop } from "./dragDrop.js";
      import { animateCard } from "./animate.js";
```

```
      const todo = document.getElementById("todo");
      todo.addEventListener("dragover", allowDrop);
      todo.addEventListener("drop", drop);

      const card = createTaskCard({ id: "a1", text: "New Task" });
      todo.appendChild(card);
      animateCard(card);
    </script>
  </body>
</html>
```

Test Cases

Use Jest with `jsdom` and fake timers for animations.

dragDrop.test.js

```
/**
 * @jest-environment jsdom
 */
import { allowDrop, drag, drop, dragEnd } from "./dragDrop.js";

function createEvent(type, target) {
  const ev = {
    type,
    target,
    preventDefault: jest.fn(),
    dataTransfer: {
      storage: {},
      setData(k, v) {
        this.storage[k] = v;
      },
      getData(k) {
        return this.storage[k];
      },
    },
  };
  return ev;
}

describe("dragDrop functions", () => {
  let col, card;
  beforeEach(() => {
    document.body.innerHTML = `
      <div class="column" id="col"></div>
      <div class="task-card" id="card" draggable="true"></div>`;
    col = document.getElementById("col");
    card = document.getElementById("card");
  });
```

```
  test("allowDrop calls preventDefault", () => {
    const ev = createEvent("dragover", col);
    allowDrop(ev);
    expect(ev.preventDefault).toHaveBeenCalled();
  });

  test("drag sets dataTransfer and adds class", () => {
    const ev = createEvent("dragstart", card);
    drag(ev);
    expect(ev.dataTransfer.getData("text/plain")).toBe(card.id);
    expect(card.classList.contains("dragging")).toBe(true);
  });

  test("drop moves card into column and removes class", () => {
    // simulate dragstart
    const dragEv = createEvent("dragstart", card);
    drag(dragEv);

    const dropEv = createEvent("drop", col);
    // ensure dataTransfer has text/plain
    dropEv.dataTransfer = dragEv.dataTransfer;
    drop(dropEv);

    expect(ev.preventDefault).not; // just syntax; check below
    expect(col.contains(card)).toBe(true);
    expect(card.classList.contains("dragging")).toBe(false);
  });

  test("dragEnd removes dragging class", () => {
    card.classList.add("dragging");
    const ev = createEvent("dragend", card);
    dragEnd(ev);
    expect(card.classList.contains("dragging")).toBe(false);
  });
});
```

task.test.js

```
/**
 * @jest-environment jsdom
 */
import { createTaskCard } from "./task.js";

describe("createTaskCard", () => {
  test("returns a draggable card with correct id and text", () => {
    const task = { id: "x1", text: "Test Task" };
    const card = createTaskCard(task);

    expect(card.tagName).toBe("DIV");
    expect(card.classList.contains("task-card")).toBe(true);
    expect(card.draggable).toBe(true);
```

```
    expect(card.id).toBe("x1");
    expect(card.textContent).toBe("Test Task");
  });
});
```

animate.test.js

```
import { animateCard } from "./animate.js";
jest.useFakeTimers();

describe("animateCard", () => {
  let card;
  beforeEach(() => {
    card = document.createElement("div");
  });

  test("adds and then removes fade-in class", () => {
    animateCard(card);
    expect(card.classList.contains("fade-in")).toBe(true);

    // advance time by 500ms
    jest.advanceTimersByTime(500);
    expect(card.classList.contains("fade-in")).toBe(false);
  });
});
```

Closing Story

With drag-and-drop and slick animations, Odessa's task board prototype felt as smooth as sliding sticky notes on glass. Investors at co-working spaces tapped their screens, imagining their own workflows transformed. The first draft of her pitch deck highlighted this interactive board—proof that a simple drag event could spark a whole startup.

Next up, Odessa will wire this board into a real backend: fetching and saving tasks via REST APIs with `fetch` and async/await. She's building end-to-end features now—from front-end drag events to persistent data storage. The journey continues, one draggable card at a time! 🚀