Hybrid Topic

**Drag and Drop**

* **Door**: Tomas Cop
* **Demo datum**: 12 maart 2025
* **Code url github**:
  + *https://github.com/TomasCop/Tomas-Cop-hybrid-topic*

# Korte uitleg topic

De bedoeling van mijn topic is om een card object te verplaatsen naar een deck door gebruik te maken van een drag en drop systeem. Je kan ook verschillende decks kaarten hebben dus deze moet je ook kunnen verschuiven om er voor te zorgen dat ik deze ook kan verplaatsen.

## Korte beschrijving

Mijn topic bestaat uit het maken van een drag en drop systeem voor makkelijk items naar de juiste locatie te kunnen slepen. Ik maak gebruik van dnd-kit om alles goed te laten verlopen.

## Prerequisites

Wat moet al gekend zijn voor je aan dit topic begint:

* Next.js
* (kan ook met javascript)

Welke tools moeten geinstalleerd zijn voor je aan dit topic kan beginnen:

* React
* Next.js

Info om deze tools te installeren (als het gaat om iets dat we **niet** in de lessen gezien hebben):

* **@dnd-kit/core**
* **@dnd-kit/sortable**
* **(styling: @dnd-kit/utilities)**

## Stappen

1. Maak eerst de objecten die je wilt slepen naar een specifieke zone.
2. Maak een zone waar je het item kunt achterlaten.
3. Voeg een drag-overlay toe aan het item dat je wilt verplaatsen (je kunt eventueel modifiers toevoegen, bijvoorbeeld om te voorkomen dat een kaart buiten de randen van de pagina kan bewegen).
4. "use client"  
     
   import { useSortable } from "@dnd-kit/sortable"  
   import { *CSS* } from "@dnd-kit/utilities"  
   import { Card, CardContent } from "@/src/components/ui/card"  
   import { PokemonCard } from "@/src/components/pokemon-card"  
   import { X } from "lucide-react"  
   import { Button } from "@/src/components/ui/button"  
   import type { SortableCardProps } from "@/types"  
   import type { MouseEvent } from "react"  
     
   export function SortableCard({ id, card, deckId, onRemove }: SortableCardProps) {  
    const { attributes, listeners, setNodeRef, transform, transition, isDragging } = useSortable({  
    id,  
    data: {  
    type: "card",  
    card,  
    deckId,  
    },  
    })  
     
    const style = {  
    transform: *CSS*.Transform.toString(transform),  
    transition,  
    opacity: isDragging ? 0.5 : 1,  
    zIndex: isDragging ? 10 : 1,  
    }  
     
    const handleRemove = (e: MouseEvent<HTMLButtonElement>) => {  
    e.stopPropagation()  
    if (onRemove) {  
    onRemove()  
    }  
    }  
     
    return (  
    <Card  
    ref={setNodeRef}  
    style={style}  
    className="w-[130px] cursor-grab active:cursor-grabbing relative group hover:shadow-xl transition-all duration-200 border-0 shadow-md"  
    >  
    <CardContent className="p-2" {...attributes} {...listeners}>  
    <PokemonCard card={card} />  
     
    {deckId && onRemove && (  
    <Button  
    variant="destructive"  
    size="icon"  
    className="absolute -top-2 -right-2 h-6 w-6 opacity-0 group-hover:opacity-100 transition-opacity shadow-md"  
    onClick={handleRemove}  
    >  
    <X className="h-3 w-3" />  
    </Button>  
    )}  
    </CardContent>  
    </Card>  
    )  
   }
5. "use client"  
     
   import { useState } from "react"  
   import {  
    DndContext,  
    *DragOverlay*,  
    closestCenter,  
    KeyboardSensor,  
    PointerSensor,  
    useSensor,  
    useSensors,  
    useDroppable,  
    type DragStartEvent,  
    type DragEndEvent,  
    type DragOverEvent,  
   } from "@dnd-kit/core"  
   import {  
    arrayMove,  
    SortableContext,  
    sortableKeyboardCoordinates,  
    horizontalListSortingStrategy,  
    verticalListSortingStrategy,  
   } from "@dnd-kit/sortable"  
   import { restrictToWindowEdges } from "@dnd-kit/modifiers"  
   import { Card, CardContent } from "@/src/components/ui/card"  
   import { Button } from "@/src/components/ui/button"  
   import { Plus } from "lucide-react"  
   import { SortableDeck } from "@/src/components/sortable-deck"  
   import { SortableCard } from "@/src/components/sortable-card"  
   import { PokemonCard } from "@/src/components/pokemon-card"  
   import type { DeckType, PokemonCardType } from "@/types"  
     
   // Mock Pokemon card data  
   const pokemonCards: PokemonCardType[] = [  
    {  
    id: "1",  
    name: "Pikachu",  
    type: "Electric",  
    hp: 60,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/25.png",  
    },  
    {  
    id: "2",  
    name: "Charizard",  
    type: "Fire",  
    hp: 120,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/6.png",  
    },  
    {  
    id: "3",  
    name: "Bulbasaur",  
    type: "Grass",  
    hp: 60,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/1.png",  
    },  
    {  
    id: "4",  
    name: "Squirtle",  
    type: "Water",  
    hp: 50,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/7.png",  
    },  
    {  
    id: "5",  
    name: "Jigglypuff",  
    type: "Fairy",  
    hp: 70,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/39.png",  
    },  
    {  
    id: "6",  
    name: "Mewtwo",  
    type: "Psychic",  
    hp: 150,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/150.png",  
    },  
    {  
    id: "7",  
    name: "Gengar",  
    type: "Ghost",  
    hp: 100,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/94.png",  
    },  
    {  
    id: "8",  
    name: "Eevee",  
    type: "Normal",  
    hp: 50,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/133.png",  
    },  
    {  
    id: "9",  
    name: "Snorlax",  
    type: "Normal",  
    hp: 140,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/143.png",  
    },  
    {  
    id: "10",  
    name: "Gyarados",  
    type: "Water",  
    hp: 130,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/130.png",  
    },  
    {  
    id: "11",  
    name: "Dragonite",  
    type: "Dragon",  
    hp: 120,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/149.png",  
    },  
    {  
    id: "12",  
    name: "Machamp",  
    type: "Fighting",  
    hp: 110,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/68.png",  
    },  
    {  
    id: "13",  
    name: "Alakazam",  
    type: "Psychic",  
    hp: 90,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/65.png",  
    },  
    {  
    id: "14",  
    name: "Arcanine",  
    type: "Fire",  
    hp: 100,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/59.png",  
    },  
    {  
    id: "15",  
    name: "Lapras",  
    type: "Water",  
    hp: 120,  
    image: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/131.png",  
    },  
   ]  
     
     
   const COLLECTION\_ID = "available-cards-collection"  
     
   export default function PokemonDeckBuilder() {  
    const [decks, setDecks] = useState<DeckType[]>([  
    { id: "deck-1", name: "Deck 1", cards: [] },  
    { id: "deck-2", name: "Deck 2", cards: [] },  
    ])  
    const [availableCards, setAvailableCards] = useState<PokemonCardType[]>(pokemonCards)  
    const [activeId, setActiveId] = useState<string | null>(null)  
    const [activeCard, setActiveCard] = useState<PokemonCardType | null>(null)  
     
    // Set up the collection area as a droppable zone  
    const { setNodeRef: setCollectionRef, isOver: isOverCollection } = useDroppable({  
    id: COLLECTION\_ID,  
    })  
     
    const sensors = useSensors(  
    useSensor(PointerSensor, {  
    activationConstraint: {  
    distance: 5,  
    },  
    }),  
    useSensor(KeyboardSensor, {  
    coordinateGetter: sortableKeyboardCoordinates,  
    }),  
    )  
     
    const handleDragStart = (event: DragStartEvent) => {  
    const { active } = event  
    setActiveId(active.id.toString())  
     
    // Find if it's a card being dragged  
    const draggedCard = [...availableCards, ...decks.flatMap((deck) => deck.cards)].find(  
    (card) => card.id === active.id,  
    )  
    if (draggedCard) {  
    setActiveCard(draggedCard)  
    }  
    }  
     
    const handleDragEnd = (event: DragEndEvent) => {  
    const { active, over } = event  
    setActiveId(null)  
    setActiveCard(null)  
     
    if (!over) return  
     
    // Handle dropping a card back to the collection  
    if (over.id === COLLECTION\_ID) {  
    const cardId = active.id.toString()  
     
    // Find which deck the card is in  
    const sourceDeckIndex = decks.findIndex((deck) => deck.cards.some((card) => card.id === cardId))  
     
    if (sourceDeckIndex !== -1) {  
    const sourceDeck = decks[sourceDeckIndex]  
    const card = sourceDeck.cards.find((card) => card.id === cardId)  
     
    if (card) {  
    // Remove card from deck  
    setDecks(  
    decks.map((deck, index) => {  
    if (index === sourceDeckIndex) {  
    return {  
    ...deck,  
    cards: deck.cards.filter((c) => c.id !== cardId),  
    }  
    }  
    return deck  
    }),  
    )  
     
    // Add card back to collection  
    setAvailableCards([...availableCards, card])  
    }  
    return  
    }  
    }  
     
    // Handle deck reordering  
    if (active.id.toString().startsWith("deck-") && over.id.toString().startsWith("deck-")) {  
    const oldIndex = decks.findIndex((deck) => deck.id === active.id)  
    const newIndex = decks.findIndex((deck) => deck.id === over.id)  
     
    if (oldIndex !== newIndex) {  
    setDecks(arrayMove(decks, oldIndex, newIndex))  
    }  
    return  
    }  
     
    // Handle card moving between decks or from available to deck  
    const isDeckTarget = over.id.toString().startsWith("deck-")  
     
    if (isDeckTarget) {  
    const targetDeckId = over.id.toString()  
    const cardId = active.id.toString()  
     
    // Check if card is from available cards  
    const cardFromAvailable = availableCards.find((card) => card.id === cardId)  
     
    if (cardFromAvailable) {  
    // Move from available to deck  
    setAvailableCards(availableCards.filter((card) => card.id !== cardId))  
    setDecks(  
    decks.map((deck) =>  
    deck.id === targetDeckId ? { ...deck, cards: [...deck.cards, cardFromAvailable] } : deck,  
    ),  
    )  
    return  
    }  
     
    // Check if card is from another deck  
    const sourceDeck = decks.find((deck) => deck.cards.some((card) => card.id === cardId))  
     
    if (sourceDeck && sourceDeck.id !== targetDeckId) {  
    const card = sourceDeck.cards.find((card) => card.id === cardId)  
     
    if (card) {  
    // Move from one deck to another  
    setDecks(  
    decks.map((deck) => {  
    if (deck.id === sourceDeck.id) {  
    return { ...deck, cards: deck.cards.filter((c) => c.id !== cardId) }  
    }  
    if (deck.id === targetDeckId) {  
    return { ...deck, cards: [...deck.cards, card] }  
    }  
    return deck  
    }),  
    )  
    }  
    }  
    }  
     
     
    if (!isDeckTarget && active.data?.current?.deckId && over.data?.current?.deckId) {  
    const activeDeckId = active.data.current.deckId  
    const overDeckId = over.data.current.deckId  
     
    if (activeDeckId === overDeckId) {  
    const deckIndex = decks.findIndex((deck) => deck.id === activeDeckId)  
    if (deckIndex !== -1) {  
    const activeCardIndex = decks[deckIndex].cards.findIndex((card) => card.id === active.id)  
    const overCardIndex = decks[deckIndex].cards.findIndex((card) => card.id === over.id)  
     
    if (activeCardIndex !== -1 && overCardIndex !== -1) {  
    const newDecks = [...decks]  
    newDecks[deckIndex] = {  
    ...newDecks[deckIndex],  
    cards: arrayMove(newDecks[deckIndex].cards, activeCardIndex, overCardIndex),  
    }  
    setDecks(newDecks)  
    }  
    }  
    }  
    }  
    }  
     
    const handleDragOver = (event: DragOverEvent) => {  
    const { active, over } = event  
     
    if (!over) return  
     
     
    if (active.id !== over.id) {  
    // Get the deck IDs from the data attributes  
    const activeData = active.data.current as { type?: string; deckId?: string } | undefined  
    const overData = over.data.current as { type?: string; deckId?: string } | undefined  
     
     
    if (  
    activeData?.type === "card" &&  
    overData?.type === "card" &&  
    activeData.deckId &&  
    overData.deckId &&  
    activeData.deckId === overData.deckId  
    ) {  
    const deckId = activeData.deckId  
    const deckIndex = decks.findIndex((deck) => deck.id === deckId)  
     
    if (deckIndex !== -1) {  
    const activeIndex = decks[deckIndex].cards.findIndex((card) => card.id === active.id)  
    const overIndex = decks[deckIndex].cards.findIndex((card) => card.id === over.id)  
     
    if (activeIndex !== -1 && overIndex !== -1 && activeIndex !== overIndex) {  
    setDecks(  
    decks.map((deck, index) => {  
    if (index === deckIndex) {  
    return {  
    ...deck,  
    cards: arrayMove(deck.cards, activeIndex, overIndex),  
    }  
    }  
    return deck  
    }),  
    )  
    }  
    }  
    }  
    }  
    }  
     
    const addNewDeck = () => {  
    const newDeckId = `deck-${decks.length + 1}`  
    setDecks([...decks, { id: newDeckId, name: `Deck ${decks.length + 1}`, cards: [] }])  
    }  
     
    const returnCardToAvailable = (cardId: string, deckId: string) => {  
    const deckIndex = decks.findIndex((deck) => deck.id === deckId)  
     
    // Make sure the deck exists  
    if (deckIndex === -1) return  
     
    const card = decks[deckIndex].cards.find((card) => card.id === cardId)  
     
    // Make sure the card exists  
    if (!card) return  
     
    setDecks(  
    decks.map((deck, index) => {  
    if (index === deckIndex) {  
    return {  
    ...deck,  
    cards: deck.cards.filter((c) => c.id !== cardId),  
    }  
    }  
    return deck  
    }),  
    )  
     
    setAvailableCards([...availableCards, card])  
    }  
     
    return (  
    <DndContext  
    sensors={sensors}  
    collisionDetection={closestCenter}  
    onDragStart={handleDragStart}  
    onDragEnd={handleDragEnd}  
    onDragOver={handleDragOver}  
    >  
    <div className="space-y-8">  
    <div className="bg-gradient-to-r from-blue-50 to-indigo-50 p-6 rounded-xl shadow-md">  
    <h2 className="text-2xl font-bold mb-4 text-indigo-800 flex items-center">  
    <span className="bg-indigo-100 p-2 rounded-lg mr-2">🎴</span>  
    Beschikbare Kaarten  
    </h2>  
    <div  
    ref={setCollectionRef}  
    className={`grid grid-cols-2 sm:grid-cols-3 md:grid-cols-4 lg:grid-cols-6 gap-4 p-6 rounded-xl border-2 transition-colors ${  
    isOverCollection ? "border-indigo-500 bg-indigo-50" : "border-dashed border-indigo-300 bg-white"  
    }`}  
    >  
    {availableCards.length === 0 ? (  
    <p className="text-indigo-400 col-span-full text-center py-8 italic">  
    Sleep kaarten hierheen om ze terug in je collectie te plaatsen  
    </p>  
    ) : (  
    <SortableContext items={availableCards.map((card) => card.id)} strategy={horizontalListSortingStrategy}>  
    {availableCards.map((card) => (  
    <SortableCard key={card.id} id={card.id} card={card} />  
    ))}  
    </SortableContext>  
    )}  
    </div>  
    </div>  
     
    <div className="bg-gradient-to-r from-purple-50 to-pink-50 p-6 rounded-xl shadow-md">  
    <div className="flex justify-between items-center mb-4">  
    <h2 className="text-2xl font-bold text-purple-800 flex items-center">  
    <span className="bg-purple-100 p-2 rounded-lg mr-2">🃏</span>  
    Mijn Decks  
    </h2>  
    <Button  
    onClick={addNewDeck}  
    className="bg-gradient-to-r from-purple-500 to-pink-500 hover:from-purple-600 hover:to-pink-600 flex items-center gap-2"  
    >  
    <Plus className="h-4 w-4" />  
    Nieuw Deck  
    </Button>  
    </div>  
     
    <SortableContext items={decks.map((deck) => deck.id)} strategy={verticalListSortingStrategy}>  
    <div className="space-y-6">  
    {decks.map((deck) => (  
    <SortableDeck key={deck.id} deck={deck} returnCardToAvailable={returnCardToAvailable} />  
    ))}  
    </div>  
    </SortableContext>  
    </div>  
    </div>  
     
    <DragOverlay modifiers={[restrictToWindowEdges]}>  
    {activeId && activeCard && (  
    <Card className="w-[150px] shadow-lg">  
    <CardContent className="p-2">  
    <PokemonCard card={activeCard} />  
    </CardContent>  
    </Card>  
    )}  
    </DragOverlay>  
    </DndContext>  
    )  
   }
6. Zet vervolgens een DndContext rond zowel de Dropzone als het item dat je wilt verplaatsen, en voeg een ref={setNodeRef} toe. Vanaf dit punt kunnen de items van
7. "use client"  
     
   import { useSortable } from "@dnd-kit/sortable"  
   import { CSS } from "@dnd-kit/utilities"  
   import { Card, CardContent, CardHeader, CardTitle } from "@/src/components/ui/card"  
   import { SortableContext, horizontalListSortingStrategy } from "@dnd-kit/sortable"  
   import { SortableCard } from "@/src/components/sortable-card"  
   import { GripVertical, ChevronDown, ChevronUp } from "lucide-react"  
   import { useState } from "react"  
   import { Badge } from "@/src/components/ui/badge"  
   import type { SortableDeckProps } from "@/types"  
     
   export function SortableDeck({ deck, returnCardToAvailable }: SortableDeckProps) {  
    const [isCollapsed, setIsCollapsed] = useState(false)  
     
    const { attributes, listeners, setNodeRef, transform, transition } = useSortable({ id: deck.id })  
     
    const style = {  
    transform: CSS.Transform.toString(transform),  
    transition,  
    }  
     
    return (  
    <Card ref={setNodeRef} style={style} className="relative bg-white border-0 shadow-lg overflow-hidden">  
    <div className="absolute left-4 top-4 cursor-grab active:cursor-grabbing" {...attributes} {...listeners}>  
    <GripVertical className="h-5 w-5 text-purple-400" />  
    </div>  
     
    <CardHeader className="pl-12 pb-3 flex flex-row items-center justify-between">  
    <div className="flex items-center">  
    <CardTitle className="text-purple-800">{deck.name}</CardTitle>  
    <Badge variant="outline" className="ml-2 bg-purple-100 text-purple-800 border-purple-200">  
    {deck.cards.length} kaarten  
    </Badge>  
    </div>  
    <button  
    onClick={() => setIsCollapsed(!isCollapsed)}  
    className="p-1 rounded-full hover:bg-purple-100 transition-colors"  
    type="button"  
    >  
    {isCollapsed ? (  
    <ChevronDown className="h-5 w-5 text-purple-500" />  
    ) : (  
    <ChevronUp className="h-5 w-5 text-purple-500" />  
    )}  
    </button>  
    </CardHeader>  
     
    {!isCollapsed && (  
    <CardContent>  
    <div className="p-4 border rounded-xl bg-gradient-to-r from-purple-50 to-pink-50 min-h-[150px]">  
    {deck.cards.length === 0 ? (  
    <p className="text-center text-purple-300 italic">Sleep kaarten naar dit deck</p>  
    ) : (  
    <SortableContext items={deck.cards.map((card) => card.id)} strategy={horizontalListSortingStrategy}>  
    <div className="flex flex-wrap gap-3 overflow-x-auto pb-2">  
    {deck.cards.map((card) => (  
    <SortableCard  
    key={card.id}  
    id={card.id}  
    card={card}  
    deckId={deck.id}  
    onRemove={() => returnCardToAvailable(card.id, deck.id)}  
    />  
    ))}  
    </div>  
    </SortableContext>  
    )}  
    </div>  
    </CardContent>  
    )}  
    </Card>  
    )  
   }
8. plaats naar plaats worden versleept.
9. Als je de items nog van plaats wilt veranderen van verander dan de DndContext naar SortableContext.

## Referenties *.*

* [**https://docs.dndkit.com/**](https://docs.dndkit.com/)
* [**https://v0.dev/**](https://v0.dev/)

# Conclusie

*DND-kit is een handig hulpmiddel om verplaatsbare items op je website te creëren. Ik zie nu in wanneer het nuttig is, vooral als je werkt met items die vaak in een bepaalde volgorde verplaatst moeten worden of wanneer je verschillende items naar een andere tabel moet slepen. Echter, als je een item naar een tabel wilt verplaatsen, zou ik eerder een aankliksysteem gebruiken. Het was interessant om deze technologie te verkennen en kennis te maken met dit onderwerp. Ik zie nu ook het nut ervan en de verschillende toepassingen. Aanvankelijk vond ik het lastig om de code te implementeren, omdat deze gedocumenteerd was in JavaScript en ik geen direct voorbeeld in TypeScript kon vinden. Met wat hulp van Google en AI ben ik erin geslaagd de code te vertalen en in mijn project te integreren.*