Tartalom

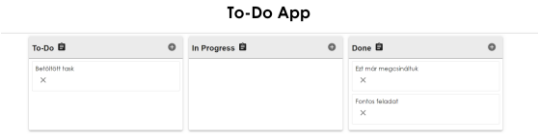
Bevezető

A webalkalmazás felhasználói leírása

Egy hétköznapi teendőket kezelő weboldal, ahol vezethethetjük a teendőinket, és ezeket kategóriákba úgynevezett állapotokba tehetjük. Minden teendő egy önmagában értődő feladat, amit 3 állapotba vehetünk fel: To-Do, In Progress, Done. Ezek az állapotok határozzák meg egy-egy teendővel hogyan is állunk. A teendők priorizálási sorrendje fentről lefele haladva értelmezendő. Alapértelmezetten a teendők felvételének sorrendje. Ha egy feladatot elvégeztünk, egyszerűen kitörölhetjük az állapotunkból.

A webalkalmazás programozói leírása

Egy teendőket kezelő weboldal, aminek elkészült a teljes frontendje és backendje. Általános háromrétegű architektúrára épülő alkalmazás, amelynek a frontend dizájnja Reactban egy single page applicationban (SPA), a backendje pedig ASP .NET Core-t használ. A háttérben futó adatbázist pedig egy MSSQL szerver biztosítja.



A program felépítése

Frontend (React) ismertetése

A frontend a korábban ismertetett React által készült el. A React előnye az egyszerű komponensekre bonthatóság, éppen ezért itt is ezt használtam fel. A teljes oldal Typescript segítségével íródott.

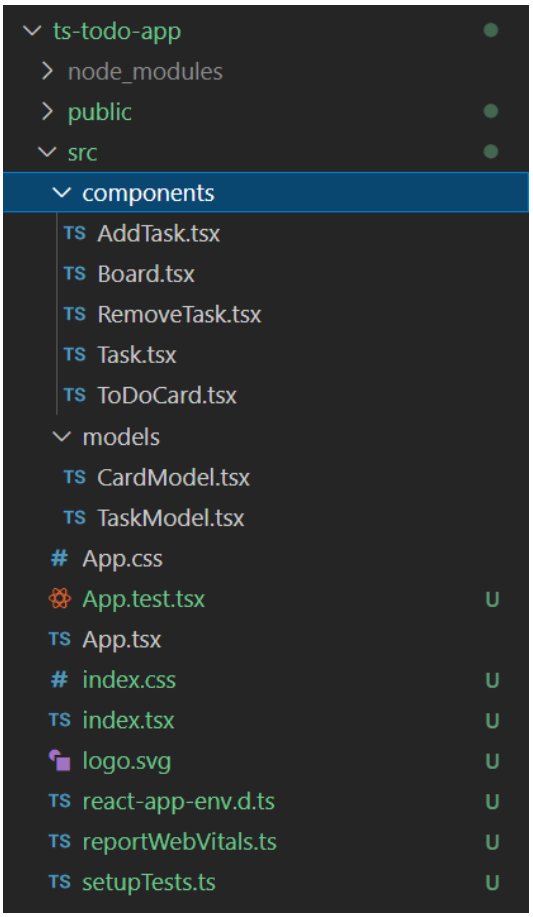
Importált könyvtárak

A frontend megfelelő elkészítéséhez szükséges volt néhány könyvtár amit külső forrásból kellett letölteni, ezzel is egyszerűsítve a saját egyéni munkám. Ezek a következő könyvtárak:

Material-UI	https://mui.com/	Segítségével tudtam megfelelő dizájnt alkotni, és egyszerűen létrehozni a komponenseket. Alkönyvtáarakat is importáltam hozzá, amik tartalmazták például az ikonokat.
Axios	https://axios-http.com/docs/intro	Segítségével tudtam kapcsolatot létesíteni a backenddel.
React-Beautiful-Dnd	https://github.com/atlassian/react-beautiful-dnd	Segítségével tudtam mozgathatóra csinálni az teendőket az adott állapotok között.

Felépítés

A frontend felépítését a következő mappaszerkezet építette fel:



A components mappában találhatóak az általam létrehozott komponensek, amikből a frontend felépül. Ezeknek a rövid felépítése úgy néz ki hogy a Board hívódik meg az App-ban. Az adja a teljes felületet, amin meghívódnak a ToDoCard komponensek. A ToDoCard komponensek az állapotok, amiből 3 létezik. Ezek később bővíthetőek úgy is hogy dinamikuson nőjön a száma, komplexitástól függően. De alapvetően ezek hívják meg a Task komponenseket, ami maga egy feladat amit nyílván szeretnénk tartani. Az AddTask komponens a ToDoCard komponensen hívódik meg, célja hogy ott vehetünk fel új taskokat. A RemoveTask komponens pedig a Task komponensen hívódik meg. Szerepe azért van hogyha a taskot nem kívánjuk tovább nyílvántartani ezzel kitörölhetjük a rendszerünkből. A models mappában tartom a két entitást, amivel az egész oldal dolgozik. A TaskModel egy Task-nak az adatait veszi fel, A CardModel pedig egy teljes állapotnak taskokkal együtt.

Komponensek

AddTask

```tsx
interface Props{
    card: CardModel;

}

const AddTask: FC<Props>=({card:card})=>{

        const [open, setOpen] = React.useState(false);
        const [value,setValue]=React.useState('');



        const handleClickOpen = () => {
          setOpen(true);
        };

        const handleClose = () => {
          setOpen(false);
        };

        const handleChange = (event:any) => {
            setValue(event.target.value);
            //console.log(value);
        }

        const handleSubmit=(event:any) => {
            handleChange(event)
            event.preventDefault();
            setOpen(false);
            //console.log(value);
            //console.log(card.todos)

            const newTask={
                todoId:card.todos.length+1,
                columnId:card.columnId,
                content:value,
                orderId:0,

            }

            const headers = {
                "Access-Control-Allow-Origin": "*"
            };
            console.log("New Task: " + newTask.todoId, newTask.columnId, newTask.content, newTask.orderId);
            axios.post('http://localhost:5240/api/Todos', newTask,{headers})
            .then(res =>{
                console.log("Sikerült!");
                console.log(res);
                console.log(res.data);
                window.location.reload();
            })
            .catch(error => {
                console.error('There was an error!', error);
                console.log(error.response);
                console.log(error.request);
                console.log(error.message);
            });


    return (
        <Box>
            <IconButton onClick={handleClickOpen}>
        <AddCircleRoundedIcon/>
      </IconButton>
      <Dialog open={open} onClose={handleClose}>
        <DialogTitle>{card.title} - Add Task - {card.columnId}</DialogTitle>
        <DialogContent>
          <TextField
            autoFocus
            margin="normal"
            id=""
            fullWidth
            variant="standard"
            value={value}
            onChange={handleChange}
          />
        </DialogContent>
        <DialogActions>
          <Button onClick={handleClose}>Cancel</Button>
          <Button onClick={handleSubmit}>Add</Button>
        </DialogActions>
      </Dialog>
        </Box>
    );
}

export default AddTask;
```

Egy gombot hoz létre a ToDoCard komponens oldalán. Aktiválásával egy POST lekérést küld a megadott endpointra.

RemoveTask

```
const RemoveTask: FC<Props> =({todos:todos})=>{

        const [open, setOpen] = React.useState(false);
        const [value,setValue]=React.useState('');



        const handleClickOpen = () => {
          setOpen(true);
        };

        const handleClose = () => {
          setOpen(false);
        };

        const handleSubmit=() => {

            const headers = {
                "Access-Control-Allow-Origin": "*"
            };
            //console.log("New Task: " + newTask.todoId, newTask.columnId, newTask.content, newTask.orderId);
            axios.delete('http://localhost:5240/api/Todos/'+todos.todoId,{headers})
            .then(res =>{
                console.log("Sikerült!");
                console.log(res);
                console.log(res.data);
                window.location.reload();
            })
            .catch(error => {
                console.error('There was an error!', error);
                console.log(error.response);
                console.log(error.request);
                console.log(error.message);
            });


        };


    return(
        <Box>
            <IconButton onClick={handleClickOpen}>
        <ClearIcon/>
      </IconButton>
      <Dialog open={open} onClose={handleClose}>
        <DialogTitle>Remove Task</DialogTitle>
        <DialogContent>
            Do you really want to remove this: {todos.content}?
        </DialogContent>
        <DialogActions>
          <Button onClick={handleClose}>No</Button>
          <Button onClick={handleSubmit}>Yes</Button>
        </DialogActions>
      </Dialog>
        </Box>
    )
}
```

A RemoveTask a Task komponensben hívódik meg, egy gomb formájában. Aktiválásával az adott taszkot törölhetjük az oldalról ha már nincs rá szükségünk. Ez egy DELETE lekérést küld a megadott endpointra.

Board

```tsx
import { Box, Button } from "@mui/material";
import axios from "axios";
import { FC, useEffect, useState } from "react";
import { CardModel } from "../models/CardModel";
import { TaskModel } from "../models/TaskModel";
import TodoCard from "./ToDoCard";

const Board: FC = () => {
  const [cardList, setCardList] = useState<CardModel[]>([]);

  //const [coutner, setCounter] = useState<number>(0);
  //const [todosList, setTodosList] = useState<TaskModel[]>([]);

  useEffect(() => {
    axios.get(`http://localhost:5240/api/Columns`, {
        headers: { "Access-Control-Allow-Origin": "*" },
      })
      .then((res) => {
        console.log("most fut a use effect");
        const columns = res.data;
        setCardList(columns);
        //console.log("GET Columns: "+columns);
      });
  }, []);

  return (
    <>
      <Box sx={{display:'flex'}}>
        {cardList &&
          cardList.map((card) => {
              console.log(card)
              return (<TodoCard key={card.columnId} card={card} ></TodoCard>)
              })}
      </Box>
    </>
  );
};

export default Board;

```

Ez a komponens az egyik fő összefogója a webalkalmazásnak. Itt kerülnek meghívásra a ToDoCard komponensek. A megadott komponenseket az adatbázisból kéri le, GET metódussal a megadott endpointról.

ToDoCard

```tsx
interface Props {
  card: CardModel;
}

const TodoCard: FC<Props> = ({ card: card}) => {

  return (
    <Card
      sx={{
        minWidth: 400,
        maxWidth: 500,
        marginRight: 2,
        boxShadow: 3,
        borderRadius: 1,
      }}
    >
      {" "}
      <Box
        sx={{
          backgroundColor: "#ebebeb",
          border: 0,
          borderBottom: 1,
          borderBlockColor: "black",
          padding: 1,
          display: "flex",
        }}
      >
        <Typography
          sx={{ fontSize: 20, fontWeight: 600, flexGrow: 0, paddingTop: 1 }}
          color="text.primary"
        >
      {card.title}

        </Typography>
        <Badge
          badgeContent={0}
          color="primary"
          sx={{ paddingLeft: 1, flexGrow: 1, paddingTop: 1 }}
        >
          <AssignmentIcon />
        </Badge>
        <AddTask key={card.columnId} card={card}></AddTask>
      </Box>
      <Box sx={{padding:'8px'}}>
          {card.todos && card.todos.map((todo)=> (
              <Task key={todo.todoId} todos={todo}></Task>
          ))}
      </Box>
    </Card>
  );
};
export default TodoCard
```

Ebben a komponensben jelennek meg az állapotok, amiből egyenlőre 3 van és nem lehet módosítani de ez később bővíthető. A Boardból érkező adatokat veszi át és abból generál tovább megfelelő mennyiségű Taskot. Ezen kívűl meghívja az AddTask komponenst is.

Task

```tsx

interface Props{
    todos?:TaskModel;

}

const Task: FC<Props> =({todos:todos})=> {
    if(todos?.content==undefined){
        return null;
    }
    return(
        <Box sx={
            {border: '1px solid lightgrey', borderRadius:'2px', padding:'8px',marginBottom:'8px',fontSize:'16px', backgroundColor:'white',}} >

        {todos.content}
        <RemoveTask todos={todos}></RemoveTask>
        </Box>
    );
}

export default Task;
```

Ez a komponens hozza létre a feladatokat amiket vezetünk az egész oldalon, és tároljuk őket tematikák szerint. Itt hívódik meg még továbbá a RemoveTask komponens is ahol törölni tudjuk a taskokat.

Modellek

CardModel

```
import { TaskModel } from "./TaskModel";

export interface  CardModel{
    columnId:number,
    title:string,
    todos: TaskModel[]
}
```

Itt jön létre maga a ToDoCard adatmodellje, ahogy az az adatbázisban is tárolva van. Ezzel könnyebben tudjuk feldolgozni az adatokat.

TaskModel

```
export interface TaskModel{
    columnId:number,
    content:string,
    orderId:number,
    todoId:number,
}
```

Itt jön létre maga a Task adatmodellje, ahogy az az adatbázisban is tárolva van. Ezzel könnyebben tudjuk feldolgozni az adatokat.

Backend ismertetése

A backendhez modern platformfüggetlen keretrendszer készült ASP .NET Core-ban. Ezt egy MSSQL adatbázis szolgálja ki.

Modellek

Column (Card)

```
namespace todoservice.Models
{
    12 references
    public class Column
    {
        3 references
        public Column()
        {
            Todos = new HashSet<Todo>();
        }
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        3 references
        public int ColumnId { get; set; }
        [StringLength(20)]
        3 references
        public string Title { get; set; }
        2 references
        public ICollection<Todo> Todos { get; set; }
    }
}
```

A ColumnID automatikusan generálódik. Meghívja a Todo modellt is collectionként.

Todo

```
namespace todoservice.Models
{
    10 references
    public class Todo
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        1 reference
        public int TodoId { get; set; }
        3 references
        public int ColumnId { get; set; }

        [StringLength(255)]
        3 references
        public string Content { get; set; }

        2 references
        public int OrderId { get; set; }

        [JsonIgnore]
        0 references
        public Column Column { get; set; }
    }
}
```

Adatbázis

Az adatbázis C# kódból lett generálva, és ehhez készült egy inicializálás is DbSeed néven. Ennek a tartalma a következő:

```
namespace todoservice.Data
{
    1 reference
    public class DbSeed
    {
        1 reference
        public static void Initialize(TodoContext context)
        {
            if (context.Columns.Any())
            {
                return;
            }

            var columns = new List<Column>();
            columns.Add(new Column { Title = "To-Do" });
            columns.Add(new Column { Title = "In Progress" });
            columns.Add(new Column { Title = "Done" });
            context.Columns.AddRange(columns);
            context.SaveChanges();

            if (context.TodoItems.Any())
            {
                return;
            }

            var todos=new List<Todo>();
            todos.Add(new Todo { Content = "Betöltött task", ColumnId=1});
            context.TodoItems.AddRange(todos);
            context.SaveChanges();
        }

    }
}
```

DbContext ahol az entitásokat felvesszük:

```
namespace todoservice.Models
{
    9 references
    public class TodoContext : DbContext
    {
        0 references
        public TodoContext(DbContextOptions<TodoContext> options)
            : base(options)
        {
        }

        12 references
        public DbSet<Todo> TodoItems { get; set; } = null!;
        12 references
        public DbSet<Column> Columns { get; set; } = null!;
    }
}
```

TodoDTO:

```
namespace todoservice.Data
{
    2 references
    public class ToDODto
    {
        2 references
        public int TodoId { get; set; }
        2 references
        public int ColumnId { get; set; }

        2 references
        public string Content { get; set; }

        2 references
        public int OrderId { get; set; }
    }
}
```

Kontrollerek

Két kontroller lett kigenerálva, a ColumnsController és a TodosController. Ezeknek célja az endpointok létrehozása, ezáltal kapcsolódási felületet biztosítva a frontendhez. Mindkét kontroller alapvető GET, POST, PUT, DELETE metódusokkal van ellátva így az egyszerűség kedvéért most erről nem készült képernyőfotó.