Eseményszervező alkalmazás

Kerezsi Zoltán – M0W4LW – 2023.12.14

Backend

Adattárolás

A backend kidolgozása viszonylag egyszerű lett. Adatbázisnak a nyílt forráskódú PostgreSQL-t használtam. Három entitást hoztam létre. Ezek a Poll, Option és a Vote. A Poll és az Option illetve a Poll és a Vote objektumok között egy-a-többhöz kapcsolat van. Az Option és a Vote között szintén egy-többes. A Vote objektumon a szavazó nevét tároljuk, ez van hozzárendelve a megfelelő Option és Pollhoz. A gyerek objektumok mind törlődnek, ha a szülő törlődne, így elkerüljuk az inkonzistens állapotokat, és nem lesz teleszemetelve az adatbázis.

Controller

Az egyszerűség kedvéért nem hoztam létre service osztályokat, mivel egységesíteni akartam a DTO-kat. Utólag kiderült, hogy ettől csak bonyolultabb lett a feladat, mert a megjelenítéshez szükséges adatmanipulációt és párosítást, így kizárólag a frontend végzi.

PollController

- [GET]/polls: Visszaküldi az összes poll objektumot. Nem dob kivételt. Ha az adatbázis üres, akkor egy üres array-el tér vissza.
- [GET]/polls/{id}: Visszaadja a poll objektumot az adatbázisból az id alapján, ha létezik. 404-es hibát dob ha nem található.
- [PUT]/polls/: Validálja a bejövő poll objektum adatait, majd frissíti a poll objektumot, ha ugyanolyan id-val már létezik az adatbázisban, különben beilleszt egy újat. 400-as hibakóddal tér vissza, ha a poll objektum bármely mezője a validációnak nem megfelelő formátumú. Visszaküldi a hibás mezőket és a hibákat egy JSON objektumban.
- [DELETE]/polls/{id}: Kitöröl egy poll objektumot id alapján, ha létezik, vagy visszatér sikeresen ha nem létezett. A hivatkozó objektumokból is kitörli a rá mutató referenciát.

OptionController

- [GET]/options/{id}: Visszaadja az option objektumot az adatbázisból az id alapján, ha létezik. 404-es hibát dob ha nem található.
- [PUT]/options/: Validálja a bejövő option objektum adatait, majd frissíti az option objektumot, ha ugyanolyan id-val már létezik az adatbázisban, különben beilleszt egy újat. 400-as hibakóddal tér vissza, ha az option objektum bármely mezője a validációnak nem megfelelő formátumú. Visszaküldi a hibás mezőket és a hibákat egy JSON objektumban.
- [DELETE]/options/{id}: Kitöröl egy option objektumot id alapján, ha létezik, vagy visszatér sikeresen ha nem létezett. A hivatkozó objektumokból is kitörli a rá mutató referenciát.

VoteController

- [GET]/votes/{id}: Visszaadja a vote objektumot az adatbázisból az id alapján, ha létezik. 404-es hibát dob ha nem található.
- [PUT]/votes/: Validálja a bejövő vote objektum adatait, majd frissíti a vote objektumot, ha ugyanolyan id-val már létezik az adatbázisban, különben beilleszt egy újat. 400-as hibakóddal tér

vissza, ha az vote objektum bármely mezője a validációnak nem megfelelő formátumú. Visszaküldi a hibás mezőket és a hibákat egy JSON objektumban.

- [DELETE]/votes/{id}: Kitöröl egy vote objektumot id alapján, ha létezik, vagy visszatér sikeresen ha nem létezett. A hivatkozó objektumokból is kitörli a rá mutató referenciát.
-

DTO

Entitás	*Bemeneti DTO*	*Kimeneti DTO*

Option	{
	optionId: string
	title: string
	description: string
	price: integer
	poll: string
	votes?: string[]
	}
		{
		optionId: string
		title: string
		description: string
		price: integer
		poll: string
		votes?: string[]
		}

Poll	{
	pollId: string
	title: string
	description: string
	multipleResult: boolean
	options?: string[]
	votes?: string[]
	}
		{
		pollId: string
		title: string
		description: string
		multipleResult: boolean
		options?: string[]
		votes?: string[]
		}

Vote	{
	voteId: string
	username: string
	poll: string
	option: string
	}
		{
		voteId: string
		username: string
		poll: string
		option: string
		}

Validáció

A kimeneti és a bemeneti DTO objektumokra validációs dekorátorokat tettem, hogy garantáljam a be és kiérkező objektumok formájának és tartalmának helyességét. A frontendre generált típusok formáját a backend-en lévő validációs dekorátorok is befolyásolják. Az “@Valid”-dal jelzett paramétereken mielőtt átadásra kerülnek lefut a hozzá való validáció amelyet a “spring-boot-starter-validation” package végez.

Hibakezelés

A “MethodArgumentNotValidException” kivételeket az ExceptionController “handleValidationExceptions” metódusa kapja el. A validációnak nem megfelelt objektumok hibái ide kerülnek, majd egy JSON objektumként kerülnek továbbításra a frontendre. A többi REST controller

“ResponseStatusException” típusú kivételt emelek, ha valamilyen logikai probléma megakadályozza a normál folyamot. Ekkor a Spring Boot framework a megadott HTTP státuszkóddal ellátott hibát küldi a kliensnek.

Frontend

A frontend nagyobb bonyolultságú lett mint eredetileg terveztem. A design-hoz Tailwind CSS-t használtam. Az állapot kezelést az új Angular signal-ok segítségével oldottam meg, ami lehetővé teszi, hogy reaktív komponenseket hozzunk létre. A specifikációban leírt módon az “ng-openapi-gen” nevű openapi generátorral hoztam létre a backendel való kommunikációért felelős komponenseket, amelyek az “src/app/api” mappában találhatóak meg.

Storage Service

A kliens központi része az angular service-ként implementált “StorageService”. Ez egy injektálható komponens, amelyet a többi komponens használ adatelérésre és a backendről való adat lekérésre. Az eredmények signal-okon keresztül válnak elérhetővé. Figyeltem a felelősségek elkülönítésére, így a generált “OptionControllerService”, “VoteControllerService”, “PollControllerService” egyedül ebben a komponensben kerül használatra.

Signals

A feladat megoldásában kulcsszerepet játszott az Angular-ba nemrégiben implementált signal és computed függvények, amelyek segítségével funkcionális paradigmában írhatjuk a frontend kódunkat. Lényegében egy olyan rendszer, amely granulárisan követi, hogyan és hol használják az állapotokat az alkalmazáson belül, lehetővé téve a keretrendszer számára a renderelési frissítések optimalizálását. Ezekre egy-egy példa látható az alábbi kódrészleten. A “voteCountByOptionId” signalban leírjuk, hogy az “allOptions” signal, a “voteCountByOptionId” és az “optionsByPollIds” signal segítségével hogyan jutunk el a kívánt eredményig. Nekünk nem kell meghívni, mivel a függőségek változásakor automatikusan meghivódik.

```
polls = signal<PollOutput[]>([]);
optionsByPollIds = signal<Record<string, OptionOutput[]>>({});
votesByOptionIds = signal<Record<string, VoteOutput[]>>({});

allVotes = computed(() => Object.values(this.votesByOptionIds()).flat());
allOptions = computed(() => Object.values(this.optionsByPollIds()).flat());

voteCountByOptionId = computed<Record<string, number>>(() =>
  this.allOptions().reduce(
    (acc, { optionId }) => ({
      ...acc,
      [optionId]: this.votesByOptionIds()[optionId]?.length ?? 0,
    }),
    {}
  )
);

maxVoteCountsByPollId = computed<Record<string, number>>(() =>
  this.polls().reduce((acc, poll) => {
    return {
      ...acc,
      [poll.pollId]: this.optionsByPollIds()[poll.pollId].reduce(
        (a, { optionId }) => {
          const voteCount = this.voteCountByOptionId()[optionId];
          return a > voteCount ? a : voteCount;
        },
        0
      ),
    };
  }, {})
);
```

Deployment

A teljes projekt konténerizálva van, a forráskód és a compose.yaml file teljesen leírja az alkalmazást, így mindenféle egyéb beállítás nélkül futtatható a “docker compose up db -d” majd “docker compose up frontend backend –build” paranccsal.

Az API_TEST.md fájl tartalmaz néhány manuális tesztelést megkönnyítő parancsot. Mivel az idő hiányában nem csináltam automatikus e2e teszteket, így ezekre hagyatkoztam.