ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

Закарпатський угорський інститут ім. Ференца Ракоці ІІ

Кафедра математики та інформатики

Реєстраційний №_____

Кваліфікаційна робота

Застосування веб-фреймворку Angular для створення фронтеду для українсько-угорського електронного словника.

Керек Ванесса Шандорівна

Студентка IV-го курсу

Освітня програма 014 «Середня освіта (Математика)»

Ступінь вищої освіти: бакалавр

Тема затверджена Вченою радою ЗУІ

Протокол № 7 /27 жовтня 2020 року

Науковий керівник: **Берегсасі Степан Степанович**

старший викладач

Завідувач кафедрою математики та інформатики: **Кучінка Каталін Йожефівна**

к. ф.-м. н

Робота захищена на оцінку _____, «___» _____ 202_ року

Протокол № _____ / 202_

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

Закарпатський угорський інститут ім. Ференца Ракоці II

Кафедра математики та інформатики

Кваліфікаційна робота

Застосування веб-фреймворку Angular для створення фронтеду для українсько-угорського електронного словника.

Ступінь вищої освіти: бакалавр

Виконав: студентка IV-го курсу

Керек Ванесса Шандорівна

Освітня програма 014 «Середня освіта (Математика)»

Науковий керівник: **Берегсасі Степан Степанович**

старший викладач

Рецензент: **Полоі Федір Федорович**

старший викладач

Берегове
2021

Зміст

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola

Matematika és Informatika Tanszék

ANGULAR WEBES KERETRENDSZER ALKALMAZÁSA AZ UKRÁN-MAGYAR ELEKTRONIKUS SZÓTÁR FRONTEND RÉSZÉNEK A KIALAKÍTÁSÁRA

Szakdolgozat

Képzési szint: alapképzés

Készítette: Kerek Vanessza

IV. évfolyamos hallgató

Képzési program: 014 „Középiskolai oktatás (Matematika)”

Témavezető: Beregszászi István

adjunktus

Recenzens: Pallay Ferenc

adjunktus

Beregszász – 2021

Tartalomjegyzék

Bevezetés

Az elektronikus szótárak nagyon népszerűek, mivel használatuk könnyű, gyors és egyszerű. Munkám során éppen ezért foglalkoztam egy ukrán – magyar elektronikus szótár kialakításával, amely elkészítéséhez Angular webes keretrendszert alkalmaztam.

Az Angular egy fejlesztési platform, amely a TypeScript-re épül. Az Angular a Google által kifejlesztett, nyílt forráskódú rendszer, amely dinamikus webes alkalmazásokhoz használható. Ez által leegyszerűsödik a webes alkalmazások front end fejlesztése. Használatával a HTML eszköztára kibővül. Az alkalmazások komponensei még egyértelműbben elkülönülnek egymástól. Az Angular-nak köszönhetően rengeteg felesleges kód elhagyható, így a program sokkal rövidebb és gyorsabb lesz.

Az Angular egy nagyon új és fejlődő nyelv. A fejlesztők jelenleg is a keretrendszer tökéletesítésén dolgoznak. Ezért is választottuk ezt az alkalmazás elkészítéséhez, bármilyen más, már elavultnak minősülő nyelv helyett.

1. fejezet

Az Angular-ról

1.1. Az Angular története

Az AngularJS átírását "Angular 2" -nek hívták, de ez meglehetősen össze zavarta a fejlesztőket. Ezért a csapat bejelentette, hogy minden keretrendszerhez külön kifejezéseket kell használni, az "AngularJS" -nél az 1.X verzióra, valamint az "Angular" -ra, „JS” nélkül, az egy újabb verzióra utal.

Az Angular 2.0 verzióját az ng-Európa konferencián jelentették be 2014. október. 22-23-án. A 2.0 verzió nagy változásai jelentős vitát váltottak ki a fejlesztők körében. Ezért 2015. április 30-án az Angular fejlesztők bejelentették, hogy az Angular 2 átkerült az Alfa alól a Developer Preview oldalra. Az Angular 2 2015 decemberében költözött a Beta-ba, és az első kiadását 2016 májusában tették meg. A végleges verzió viszont 2016. szeptember 14-én jelent meg.

2016. december 13-án jelentették be az Angular 4-et. Kihagyva a 3-at, hogy elkerüljék a zavarokat az útválasztó csomag változata miatt, amelyet már v3.3.0-ként terjesztettek. A végleges verzió 2017. március 23-án jelent meg. Az Angular 4 visszafelé kompatibilis az Angular 2-vel is. Az Angular 4.3 verzió kisebb jelentőségű, ami azt jelenti, hogy nem tartalmaz olyan nagy változásokat. A 4.3-as verzió jellemzői:

- Bemutatják a HttpClient szoftvert. Amely egy kisebb, könnyebben használható és hatékonyabb könyvtár a HTTP kérések készítéséhez.

- Új útválasztó életciklus események a resolvers számára. Négy új esemény: GuardsCheckStart , GuardsCheckEnd , ResolveStart , ResolveEnd csatlakozik a meglévő életciklus-eseményekhez, mint például a NavigationStart.

- Az animációk feltételes letiltása.

Az Angular 5 megjelenése 2017. november 1. Az Angular 5 legfontosabb fejlesztései között szerepel a progresszív webalkalmazások támogatása. A build optimalizáló és az anyagtervezéssel kapcsolatos fejlesztések.

Az Angular 6 2018. május 4-én jelent meg. Ez egy jelentős kiadás, amely kevésbé alapul a szolgáló keretrendszerre. Inkább az eszközláncra összpontosít, és arra, hogy a jövőben könnyebbé tegye az Angular gyors mozgatását, például: ng update, ng add, CLI-munkaterületek, Könyvtári támogatás, Fa meghatározó szolgáltatók, Animációk teljesítményjavításai és RxJS v6.

Az Angular 7 megjelenése 2018. október 18 volt. Frissítések jöttek ki az alkalmazás teljesítményére, a szögletes anyagra és a CDK-ra, a virtuális görgetésre, a kiválasztottak jobb hozzáférhetőségére. Mostantól támogatják a Tartalom-vetítést webes szabvány használatával az egyedi elemeknél, és a függőségi frissítéseket a Typescript 3.1, RxJS 6.3, Node 10.

Az Angular 8 2019. május 28-án jelent meg. Az összes alkalmazáskód differenciális betöltésével. A lassú útvonalak dinamikus importálásával, a webmunkásokkal, a TypeScript 3.4 támogatással és az Angular Ivy opcióval rendelkezik. Az Angular Ivy a következőket tartalmazza:

- Generált kód, amely futás közben könnyebben olvasható és hibakereshető

- Gyorsabb újjáépítési idő

- Jobb teherbírás

- Továbbfejlesztett sablontípus-ellenőrzés

- Fordított kompatibilitás

Az Angular 9 2020. február 6-án tették közzé. Ez a verzió az összes alkalmazást az Ivy fordító és a futásidő használatára alapértelmezés szerint áthelyezi. Az Angular

frissítésre került a TypeScript 3.6 és 3.7 használatához. A több száz hibajavítás mellett az Ivy fordítója és futási ideje számos előnyt kínál:

- Kisebb kötegméretek
- Gyorsabb tesztelés
- Jobb hibakeresés
- Továbbfejlesztett CSS osztály- és stíluskötés
- Továbbfejlesztett típusellenőrzés
- Javított összeállítási hibák
- Javított építési idő, alapértelmezés szerint engedélyezve van az AOT
- Javult nemzetközivé válás

Az Angular 10 2020. június 24-én jelent meg. Melyben megjelennek:

- Új dátumtartomány-választó (Material UI library)
- Figyelmeztetések a CommonJS behozatalával kapcsolatban
- Opcionális szigorúbb beállítások
- Az ökoszisztéma naprakészen tartása
- Új alapértelmezett böngésző konfiguráció
- Értékcsökkenések és eltávolítások

Az Angular 11 2020. november 11-én jelent meg.

A v9 óta az Angular csapat minden új alkalmazást áthelyezett az Ivy fordító. Az Ivy-n dolgoznak a kimeneti csomagméreteken és a fejlesztési sebesség javításán. Minden verzió várhatóan kompatibilis lesz az előző kiadással. Az Angular fejlesztőcsapat ígéretet tett arra, hogy évente kétszer frissít. Jelenleg az Angular 11-et használjuk.

1.2. Mi az Angular?

Az Angular egy platform és keretrendszer alkalmazások HTML és TypeScript használatával történő felépítéséhez. Az alapvető és opcionális funkciókat TypeScript könyvtárak készleteként valósítja meg, amelyeket importálunk az alkalmazásokba.

Az Angular alkalmazás architektúrája bizonyos alapfogalmakra támaszkodik. Az Angular keret alapvető építőkövei olyan alkatrészek, amelyek NgModulákba vannak rendezve. Az NgModules a kapcsolódó kódokat funkcionális halmazokba gyűjti; egy alkalmazást egy NgModule készlet határoz meg. A dokumentumban mindig van legalább egy gyökér modul, amely lehetővé teszi betöltő, és jellemzően sokkal több funkció használatát. [7]

Az alkatrészek megadják a nézeteket, amelyek a képernyő elemei. Ezek közül az Angular választhat és módosíthat a program logikája és adatai szerint.

Az összetevők olyan szolgáltatásokat használnak, melyek olyan speciális funkciókat biztosítanak, amelyek nem közvetlenül kapcsolódnak a nézetekhez. A szolgáltatókat komponensekbe injektálhatjuk függőségként, így a kód moduláris lesz, újra felhasználható és hatékony.

A modulok, alkatrészek és szolgáltatások olyan osztályok, amelyek úgynevezett dekoratőröket használnak. Ezek a dekorátorok megjelölik a típusukat, és olyan meta adatokat adnak meg, amelyek megadják az Angular használatuk módját.

Egy összetevő osztály meta adatai társítják a nézetet meghatározó sablonnal. A sablon a hétköznapi HTML-t ötvözi az Angular direktívákkal és az összerendelési jelölésekkel, amelyek lehetővé teszik, hogy az Angular módosítsa a HTML-t, mielőtt megjelenítené.

A szolgáltatási osztály meta adatai megadják azokat az információkat, amelyekre szüksége van, hogy az összetevők számára elérhetővé tegye a függőségi injekciót. Az alkalmazás összetevői általában sok nézetet határoznak meg hierarchikusan rendezve. Az Angular szolgáltatást Router nyújt, hogy segítsen meghatározni a nézetek közötti navigációs útvonalakat. Az útválasztó kifinomult böngészőbeli navigációs képességeket biztosít. [6]

Anguler alkalmazás felépítését tekintve a manapság elterjedt keretrendszerekhez hasonlóan itt is az MVC (Model View Controller) modellel találkozhatunk. A nézetek a HTML fájlok lesznek, míg a kontrollerek valamint a modellek JavaScript nyelven íródnak majd. [7]

Nézetét tekintve az Angular követi a HTM szabványt, de kibővíti azt direktívákkal. Ezek a direktívák nagyon megkönnyítik az életünket, és ezeknek köszönhetően nem is lesz szükségünk közvetlen DOM manipulációra. Néhány fontosabb direktíva:

- ng-app: - meghatározhatjuk vele az alkalmazásunk gyökérelemét, ezzel hozzuk a keretrendszer tudtára, hogy az elemen belül Angular kifejezéseket fogunk használni.
- ng-model: - segítségével összekapcsolhatunk egy modellbeli elemet egy felületivel, így azok mindig konzisztensek maradnak.
- ng-class: -dinamikusan CSS osztályok betöltését teszi lehetővé.
- ng-controller: -meghatározhatjuk vele, hogy melyik kontrollert szeretnénk használni az adott elemen belül.
- ng-repet: -ez valójában egy foreach ciklus.
- ng-show/ng-hide: - logikai kifejezés alapján megjelenít, vagy elrejt egy elemet.
- ng-if: -a klasszikus elágazás.
- ng-switch: -switch elágazás. [7]

A keretrendszer főbb célkitűzései:

- Felület definiálására ideális egy deklaratív leírás (HTML), míg az imperatív programozás kiváló, hogy kifejezze az üzleti logikát.
- Válasszuk le a DOM (Dokumentum Objektum Modell) manipulációt az alkalmazás logikáról.
- A program tesztelése legalább olyan kritikus, mint annak írása.
- Az alkalmazás kliens, illetve szerveroldal teljes szétválasztása. [7]

1.3. Anguler kifejezések

Az Angular kifejezések JavaScript szerűek, amiket kapcsos zárójelek közé helyezhetünk el. Viszont van a JavaScript és Anguler kifejezések között néhány hasonlósági kivétel:

- Kontextus: JavaScript esetén a kiértékelés kontextusa a window, míg Angular esetén a scope objektum.
- Engedékenység: JavaScript-ben, ha ki szeretnénk értékelni egy nem definiált tulajdonságot, akkor ReferenceError-t, vagy TypeError-t kapunk. Ezzel ellentétben Angulárban nem kapunk hibát undefined, illetve null kifejezések kiértékelésekor.
- Nincsenek vezérlési szerkezetek: A következők nem használhatók Angular kifejezésekben: elágazás, ciklus, kivétel.
- Szűrők: Használhatunk különféle szűrűket (pl.: dátum formázása, listák szűrése). [7]

Ha olyan kifejezést szeretnénk írni, ami a fenti megszorítások miatt nem megvalósítható, akkor valószínűleg annak már egy kontroller metódusban lesz a helye. Ha pedig ezt kontrollerben szeretnénk kiértékelni, akkor azt az eval() segítségével tehetjük meg.

1.4. Modulok

Az Angg NgModules eltér a JavaScript moduloktól, és kiegészíti azokat. Az NgModule deklarál egy olyan összeállítási kontextust, amely egy alkalmazás tartománynak, egy munkafolyamatnak vagy egy szorosan kapcsolódó képességkészletnek van szentelve. Az NgModule funkcionális egységek létrehozásához társíthatja összetevőit a kapcsolódó kódokkal, például szolgáltatásokkal. [6]

Minden Angularnak van egy gyökér modulja, konvencionális nevük AppModule, amely egy bootstrap mechanizmus. Ez indítja el az alkalmazást. Egy alkalmazás általában sok funkcionális modult tartalmaz.

A JavaScript modulokhoz hasonlóan az NgModules képes importálni a funkcionalitást más NgModule modulokról, és lehetővé teszi saját funkcionalitásuk exportálását és más NgModule használatát. Például az útválasztó szolgáltatás használatához az alkalmazásban importálja az RouterNgModule modult. [6]

A kód különálló funkcionális modulokba rendezése segít a komplex alkalmazások fejlesztésében és az újrafelhasználhatóság megtervezésében. Ez a technika lehetővé teszi a lassú betöltést - vagyis a modulok igény szerinti betöltés - előnyeit, hogy minimalizálja az indításkor betöltendő kód mennyiségét.

1.5. Komponensek

Minden szögletes alkalmazásnak van legalább egy összetevője, a gyökérösszetevő, amely összeköti az összetevő hierarchiát az oldal dokumentum objektum modelljével (DOM). Minden összetevő meghatároz egy osztályt, amely alkalmazás adatokat és logikát tartalmaz, és egy HTML- sablonnal van társítva, amely meghatározza a célkörnyezetben megjelenítendő nézetet.[6]

A dekorátorok olyan funkciók, amelyek módosítják a JavaScript osztályokat. Az Angular számos dekorátort határoz meg, amelyek meghatározott típusú meta adatokat csatolnak az osztályokhoz, így a rendszer tudja, mit jelentenek ezek az osztályok és hogyan kell működniük.

Minden egyes összetevőhöz megadhatunk nemcsak HTML sablont, hanem az ehhez a sablonhoz tartozó CSS stílusokat is, megadva a szükséges választókat, szabályokat és média lekérdezéseket.

1.6. Sablonok

A sablon a HTML-t ötvözi az Anguler jelöléssel, amely módosíthatja a HTML-elemeket, mielőtt azok megjelennek. A sablon irányelvek programlogikát nyújtanak, és a kötelező jelölés összeköti az alkalmazás adatait és a DOM-ot. Kétféle adatkötés létezik:

- Az esemény-összerendelés lehetővé teszi az alkalmazásának, hogy az alkalmazás adatainak frissítésével válaszoljon a célkörnyezet felhasználói beviteleire.
- A tulajdonságkötés lehetővé teszi az alkalmazásadatokból kiszámított értékek interpolálását a HTML-be.[6]

Mielőtt egy nézet megjelenik, az Angular kiértékeli az irányelveket. Megadja a sablonban található kötőszintaxist a HTML-elemek és a DOM módosítására a program adatai és logikája szerint. Az Angular támogatja a kétirányú adatkötést, tehát a DOM-ban végrehajtott változások, például a felhasználói döntések, a programadatokban is tükröződnek.

Sablonjai csövek segítségével javíthatják a felhasználói élményt a megjelenített értékek átalakításával. Például pipák segítségével jelenítse meg a felhasználó területi

beállításainak megfelelő dátumot és pénznem értéket. Az Angular előre definiált sablonokat biztosít a közös átalakításokhoz, és saját sablont is meghatározhat.

Az Angular RouterNgModule olyan szolgáltatást nyújt, amely lehetővé teszi a navigáció elérési útjának meghatározását a különböző alkalmazásállapotok között és az alkalmazás hierarchiáinak megtekintését. A már ismert böngésző navigációs konvenciók alapján készült:

- Írjon be egy URL-címet a címsorba, és a böngésző navigál a megfelelő oldalra.
- Kattintson az oldalon található linkekre, és a böngésző új oldalra navigál.
- Kattintson a böngésző Vissza és előre gombjaira, és a böngésző előre és hátra navigál a látott oldalak előzményei között.[6]

Az útválasztó az URL-hez hasonló utakat oldalak helyett nézetekhez térképezi fel. Amikor a felhasználó olyan műveletet hajt végre, hogy például rákattint egy linkre, amely új oldalt tölt be a böngészőbe, az útválasztó elfogadja a böngésző viselkedését, és megjeleníti vagy elrejti a nézet hierarchiáit.

Ha az útválasztó megállapítja, hogy az alkalmazás aktuális állapota különleges funkciókat igényel, és az azt meghatározó modul nincs betöltve, az útválasztó igény szerint lassan töltheti be a modult.

Az útválasztó értelmezi a hivatkozás URL-jét az alkalmazás nézetének navigációs szabályainak és adatállapotának megfelelően. Navigálhat új nézetekhez, amikor a felhasználó egy gombra kattint, vagy kiválaszt egy legördülő mezőből, vagy válaszul bármilyen más ösztönzésre bármely forrásból. Az útválasztó naplózza a tevékenységeket a böngésző előzményeiben, így a vissza és az előre gomb is működik.[6]

A navigációs szabályok meghatározásához társítja a navigációs útvonalakat az összetevőkhöz. Az elérési út egy URL-szerű szintaxist használ, amely integrálja a program adatait, ugyanúgy, ahogy a sablon szintaxisa integrálja a nézeteit a program adataival. Ezután a program logikáját alkalmazva kiválaszthatja a megjelenítendő vagy elrejtendő nézeteket, válaszul a felhasználói bevitelre és a saját hozzáférési szabályaira.

Az alkatrészsablonok nem mindig vannak rögzítve. Előfordulhat, hogy egy alkalmazás futás közben új komponenseket tölt be.

Az Angular Material Angular CLI sémákkal van ellátva, hogy megkönnyítse az Material alkalmazások létrehozását. Ezek telepíthetőek @angular/cdk és @angular/mate-

rial. Az npm csomagok telepítése után az Angular CLI-n keresztül elérhetők lesznek. Az alábbi paranccsal telepíthető a projektbe az Angular Material: ng add @angular/material. Ez alapján számos új, hasznos és különleges beállítást kapunk az oldalunkhoz. [8]

A telepítés mellett az Angular Material több vázlattal is rendelkezik, amelyekkel könnyen létrehozhatók az Material Design alkatrészek:

- address-form Komponens egy űrlapcsoporttal, amely a Material Design űrlapvezérlőket használja a szállítási cím megadására
- navigation Létrehoz egy összetevőt egy adaptív Material Design sidenav és egy eszköztár segítségével az alkalmazás nevének megjelenítéséhez
- dashboard Komponens több Material Design kártyával és menüvel, amelyek rács elrendezésben vannak igazítva
- table Készít egy összetevőt egy Material Design adattáblával, amely támogatja a rendezést és a lapozást
- tree Komponens, amely interaktív módon vizualizálja a beágyazott mappastruktúrát az <mat-tree>összetevő használatával [6]

1.7. Angular CLI

Az Angular CLI egy parancssori interfész eszköz, amelyet a szögletes alkalmazások inicializálásához, fejlesztéséhez, állványozásához és karbantartásához használható közvetlenül a parancssorból.

Az Angular CLI a leggyorsabb, legegyszerűbb és ajánlott módszer az Angular alkalmazások fejlesztésére. A CLI számos feladatot megkönnyít. Például egy Anguler alkalmazást fordít egy kimeneti könyvtárba. Felépíti és kiszolgálja az alkalmazását, újjáépítve a fájlmódosításokat. Fájlokat generál vagy módosít egy vázlat alapján. Futtatja az egység teszteket egy adott projekten. Épít és kiszolgál egy Angular alkalmazást, majd végpontok közötti teszteket futtat. Az Angular CLI értékes eszközt jelent az alkalmazások kiépítéséhez. [6]

Az Angular CLI főbb verziói követik az Angular támogatott főbb verzióit. Telepítése a parancssorból a legegyszerűbb. Ehhez csak ki kell adnunk a parancsot: *npm install -g @angular/cli*

Egy új Angular projektet létrehozni pedig az ng new név paranccsal lehet. A létrejött fájlt megnyitni pedig a cd név paranccsal kell. Az új alkalmazás futtatásához a böngészőben meg kell nyitni a http: // localhost: 4200 címet. Ha a forrásfájlban bármit is megváltoztatunk, akkor az ng serve parancsot kell használni, és a kiszolgáló automatikusan újjáépíti az alkalmazást és újratölti az oldalt.

Az new parancs létrehoz egy Angular munkaterület mappát, és új alkalmazásvázlatot hoz létre. Egy munkaterület több alkalmazást és könyvtárat tartalmazhat. Az ng új parancs által létrehozott kezdeti alkalmazás a munkaterület legfelső szintjén található. Amikor további alkalmazást vagy könyvtárat hozunk létre a munkaterületen, az egy projects/ almappába kerül.

Egy újonnan létrehozott alkalmazás tartalmazza a gyökérmodul forrásfájljait, gyökér összetevővel és sablonnal. Minden alkalmazáshoz tartozik egy src mappa, amely tartalmazza a logikát, az adatokat és az eszközöket. Közvetlenül szerkeszthetjük a létrehozott fájlokat, vagy hozzáadhatjuk és módosíthatjuk őket a CLI parancsokkal. Az ng generálás paranccsal új fájlokat adhatunk hozzá további összetevőkhöz és szolgáltatásokhoz, valamint kódolhatjuk az új csöveket, irányelveket stb. Az alkalmazásokat és könyvtárakat létrehozó vagy azokon működő parancsokat, mint például a hozzáadás és a létrehozás, egy munkaterületen vagy projektmappán belül kell végrehajtani.[6]

Egyetlen munkaterület-konfigurációs fájl angular.json jön létre a munkaterület legfelső szintjén. Itt állítható be projektenként alapértelmezéseket a CLI parancsopcióihoz, és meghatározhatjuk azokat a konfigurációkat, amelyeket akkor kell használni, amikor a CLI különböző célokhoz épít projektet.

Az ng config parancs segítségével beállíthatjuk és lekérhetjük a konfigurációs értékeket a parancssorból, vagy angular.json közvetlenül szerkeszthető a fájlt. Ne feledjük, hogy a konfigurációs fájlban szereplő opcióneveknek CamelCase-t kell használniuk, míg a parancsokhoz adott opciónevek CamelCase-t vagy Dash-case-t is használhatnak. [6]

A parancs szintaxisa a következőképpen jelenik meg: *ng commandNameOrAlias requiredArg [opcionálisArg][options]* . A legtöbb parancsnak és néhány opciónak vannak álnevei. Az álnevek az egyes parancsok szintaxis utasításában láthatók.

Az opciók nevét dupla kötőjel jelöli. Az opciós álneveket egyetlen kötőjel jelöli. Az érvek nem szerepelnek előtagban. Például: *ng build my-app -c production.* Jellemzően a

létrehozott műtermék neve argumentumként megadható a parancs számára, vagy megadható a name opcióval.

Az argumentum és az opció neve megadható akár camelCase vagy kötőjelben is *–myOptionName*egyenértékű *–my-option-name.* A logikai opcióknak két formája van: *–this-optiona* jelzőt állítja true, és *–no-this-option* állítja false. Ha egyik opció sem szerepel, akkor a jelző az alapértelmezett állapotban marad, amint az a referencia dokumentációban szerepel.[6]

A fájlokat meghatározó opciók abszolút elérési útként adhatók meg az aktuális munkakönyvtárhoz képest, amely általában a munkaterület vagy a projekt gyökere. Az ng generálás és az ng parancsok argumentumként veszik fel az aktuális projekthez létrehozandó vagy hozzáadandó tárgyat vagy könyvtárat. Az általános lehetőségek mellett minden műtárgy vagy könyvtár sematikusan meghatározza a saját lehetőségeit. A sematikus opciókat a parancs ugyanabban a formátumban szállítja, mint az azonnali parancsopciók. [6]

1.8. Angular schematics

A sematikus sablon alapú kódgenerátor, amely támogatja az összetett logikát. Ez egy utasításkészlet, egy szoftverprojekt átalakításához kód generálásával vagy módosításával. A sémákat gyűjteményekbe csomagolják és npm-el telepítik.

A sematikus gyűjtemény hatékony eszköz lehet bármilyen szoftverprojekt létrehozásához, módosításához és karbantartásához, de különösen hasznos az Angular projektek testreszabásához, hogy megfeleljen a saját szervezetének egyedi igényeinek. Sémákat használhat például a gyakran használt felhasználói felület minták vagy meghatározott összetevők előállításához előre definiált sablonok vagy elrendezések segítségével. A vázlatokkal kikény-szerítheti az építészeti szabályokat és egyezményeket, következetesé és interoperatívvá téve a projekteket. [6]

A sémák a szögletes ökoszisztéma részét képezik. Az Angular CLI sematikusan alkalmazza a transzformációkat egy webalkalmazás-projektbe. Módosíthatja ezeket a sémákat, és definiálhat újakat. Például frissítheti a kódot a függőségben bekövetkező törésváltozások kijavításához, vagy új konfigurációs opciót, keretet adhat hozzá egy meglévő projekthez.

A hozzáadási sémát általában egy könyvtárhoz szállítják, így a könyvtár hozzáadható egy meglévő projekthez ng add. A addparancs a csomagkezelővel tölti le az új függőségeket, és meghív egy vázlatosan megvalósított telepítési parancsfájlt. Egy sematikus hozzáadással a projekt konfigurációs változtatásokkal frissülhet, további függőségeket vagy állványcsomag-specifikus inicializáló kódot adhat hozzá. Például a *angular/pwasematic* alkalmazás PWA-vá alakítja az alkalmazást az alkalmazás-nyilvántartás és a szolgáltatás munkatársának hozzáadásával, a angular/elements sematikus pedig *document-register-element.jsa* szögletes elemek kitöltését és függőségeit adja hozzá. [6]

A generációs sémák az *ng generate* parancs utasításai. A dokumentált részparancsok az alapértelmezett Angular generációs sémákat használják, de a könyvtárában definiált termék előállításához megadhat egy másik sémát is.

1.9. Kontroller

Az Angular kontrollerek szabványos JavaScript nyelven íródnak. Amikor a kontrollert hozzácsatoljuk a DOM-hoz az ng-controller segítségével, akkor létre fog jönni egy új példány. Ilyenkor létrejön egy új gyerek scope, és a kontroller konstruktorán keresztül beinjektálhatjuk a paramétereket.

A kontrollereket használhatjuk objektumok inicializálására, vagy meghatározhatjuk egy objektum viselkedését vele. A kontrollereket nem ajánlatos használni:

- DOM manipulációra - A kontrollerek csak üzleti logikát tartalmazhatnak.
- Kimenet formázása – erre használjuk inkább az erre létrehozott szűrőket.
- Függvények, vagy állapotok megosztása más kontrollerekkel.
- Más komponensek életciklusának felügyelete. [6]

A komplexitása és objektum orientált programozás meg-közelítése miatt nem bemutatkozó oldalak készítésére alkották, habár biztosít lehetőséget dom manipulálásra, animációkra és minden egyébre. Segítségével egyszerűen, gyorsan lehet single-page adminisztrációs felületeket létrehozni. A szerverrel történő kommunikációt támogatja JSON, XML, HTML formában is, erre külön proxy osztályokat hoztak létre. Az Ext JS támogatást nyújt mind MVC, mind MVVM architektúrájú alkalmazások fejlesztéséhez. Mindkét megközelítésnek az a lényege, hogy logikailag szétválasszuk a kódot az újra felhasználhatóság

és átláthatóság érdekében. Természetesen nem kötelező ezeket a mintákat követni, de az osztály szerkezet és a Sencha Cmd ezekhez lett tervezve, így a leghatékonyabb fejlesztés az előző két architektúra követésével biztosított. [6]

1.10. Űrlapok

A felhasználói adatok űrlapokkal történő kezelése számos elterjedt alkalmazás alapja. Az alkalmazások űrlapokat tesznek lehetővé a felhasználók számára a bejelentkezéshez, a profil frissítéséhez, a bizalmas információk megadásához és sok más adatbeviteli feladat elvégzéséhez.

Az Angular két különböző megközelítést kínál a felhasználói adatok űrlapokon keresztüli kezeléséhez: reaktív és sablon-vezérelt. Mindkettő rögzíti a felhasználói beviteli eseményeket a nézetből, érvényesíti a felhasználói adatbevitelt, létrehoz egy űrlapmodellt és adatmodellt a frissítéshez, és lehetőséget nyújt a változások nyomon követésére. A reaktív űrlapok és a sablon által vezérelt űrlapok különböző módon dolgozzák fel és kezelik az űrlapadatokat. Minden megközelítés különböző előnyöket kínál. [6]

A reaktív űrlapok közvetlen, explicit hozzáférést biztosítanak az alapul szolgáló űrlapok objektummodelljéhez. A sablon által vezérelt űrlapokhoz képest robusztusabbak: méretezhetőbbek, újrafelhasználhatóbbak és tesztelhetőbbek. Ha az űrlapok az alkalmazás kulcsfontosságú részét képezik, vagy ha már reaktív mintákat használ az alkalmazás felépítéséhez, használjon reaktív űrlapokat.[6]

A sablon által vezérelt űrlapok a sablon direktíváira támaszkodva hozzák létre és manipulálják az alapul szolgáló objektum modellt. Hasznosak egy egyszerű űrlap hozzáadásához egy alkalmazáshoz, például egy e-mail lista regisztrációs űrlaphoz. Könnyen hozzáadhatók egy alkalmazáshoz, de nem méreteznek olyan jól, mint reaktív formák. Ha nagyon alapvető űrlapkövetelményei és logikája van, amelyek kizárólag a sablonban kezelhetők, akkor a sablon által vezérelt űrlapok megfelelőek lehetnek. Ha az űrlapok az alkalmazás központi részét képezik, a méretezhetőség nagyon fontos. Az űrlapmodellek újrafelhasználása az összetevők között kritikus fontosságú. A reaktív űrlapok jobban skálázhatók, mint a sablon által vezérelt űrlapok. Közvetlen hozzáférést biztosítanak az alapul szolgáló űrlap API-hoz, és szinkron hozzáférést biztosítanak az űrlap adatmodelljéhez, megkönnyítve a nagyméretű űrlapok létrehozását. A reaktív űrlapok ke-

vesebb telepítést igényelnek a teszteléshez, és a teszteléshez nincs szükség a változások észlelésének mély megértésére az űrlapfrissítések és -ellenőrzés megfelelő teszteléséhez. A sablon által vezérelt űrlapok egyszerű forgatókönyvekre összpontosítanak, és nem annyira újrafelhasználhatóak. Elvonják az alapul szolgáló űrlap API-t, és csak aszinkron hozzáférést biztosítanak az űrlap adatmodelljéhez. A sablon alapú űrlapok absztrakciója is befolyásolja a tesztelést. A tesztek mélyen a kézi változás észlelés végrehajtására támaszkodnak a megfelelő futtatáshoz, és további beállítást igényelnek. [6]

A reaktív és a sablon által vezérelt űrlapok egyaránt követik az értékváltozásokat az űrlapbeviteli elemek. A két megközelítés megegyezik az alapul szolgáló építőelemekkel, de különböznek a közös űrlapvezérlő példányok létrehozásának és kezelésének módjában.

A reaktív űrlapokkal az űrlapmodellt közvetlenül az összetevő osztályban definiálja. Az *[formControl]* irányelv a kifejezetten létrehozott *FormControl* példányt a nézet egy adott űrlapeleméhez kapcsolja, egy belső érték-hozzáféréssel.

A sablon alapú formákban az űrlapmodell inkább implicit, mint explicit. Az irányelv *NgModel* létrehoz és kezel egy FormControl példányt egy adott űrlapelemhez. Ha egy alkalmazás űrlapot tartalmaz, akkor az Angularnak szinkronban kell tartania a nézetet az összetevő modellel, és az összetevő modellt szinkronban a nézettel. Amint a felhasználók megváltoztatják az értékeket és a nézeten keresztül választanak, az új értékeknek tükröződniük kell az adatmodellben. Hasonlóképpen, amikor a programlogika megváltoztatja az értékeket az adatmodellben, ezeket az értékeket tükrözni kell a nézetben. [6]

A reaktív és a sablon által vezérelt űrlapok abban különböznek egymástól, hogy miként kezelik a felhasználóból származó adatokat vagy az automatizált változtatásokat. Az alábbi ábrák az egyes nyomtatványtípusok mindkét típusú adatfolyamát szemléltetik a fent meghatározott kedvenc-szín beviteli mező használatával.

A reaktív formák az adatmodellt tisztán tartják azzal, hogy megváltoztathatatlan adatstruktúrának biztosítják. Minden alkalommal, amikor az adatmodellben változás történik, a FormControlpéldány új adatmodellt ad vissza, nem pedig a meglévő adatmodellt. Ez lehetővé teszi az adatmodell egyedi változásainak nyomon követését a vezérlés megfigyelhetősége révén. A változás észlelése hatékonyabb, mert csak egyedi változások esetén kell frissíteni. Mivel az adatfrissítések reaktív mintákat követnek, integrálhatók megfigyelhető operátorokkal az adatok átalakításához. A reaktív űrlapok az egyéni validátorokat olyan funkciókként definiálják, amelyek ellenőrzést kapnak az ellenőrzéshez. [6]

A sablonvezérelt űrlapok a kétirányú adatkötéssel végzett mutabilitásra támaszkodnak az összetevő adatmodelljének frissítéséhez, amikor a sablonban módosítások történnek. Mivel a kétirányú adatmegkötés használata esetén nincs egyedi nyomon követhető adatmodell, a változások észlelése kevésbé hatékony a frissítések szükségességének meghatározásakor. A sablonvezérelt űrlapok sablon direktívákhoz vannak kötve, és egyedi ellenőrző irányelveket kell tartalmazniuk, amelyek beburkolják az ellenőrzési funkciókat. A tesztelés nagy szerepet játszik a komplex alkalmazásokban. Egyszerűbb tesztelési stratégia hasznos, ha ellenőrzi, hogy az űrlapok megfelelően működnek-e. A reaktív űrlapok és a sablonvezérelt űrlapok különböző mértékben támaszkodnak arra, hogy az UI-t az űrlapvezérlésen és az űrlapmező változásain alapuló állítások végrehajtására fordítsák. Az alábbi példák bemutatják az űrlapok tesztelésének folyamatát reaktív és sablon-vezérelt űrlapokkal. [6]

2. fejezet

Alkalmazás készítése

2.1. Környezet kialakítása

Az Angular keretrendszer használatához szükséges ismerni a következő nyelveket: HTML, CSS, JavaScript, TypeScript. Egy programot ezekre építünk rá. A kódok szerkesztéséhez a Visual Studio Code programot használtuk, amely egy ingyenes, nyílt forráskódú kódszerkesztő, melyet a Microsoft fejleszt ki. Támogatja a hibakeresőket, képes az intelligens kódkezelésre az IntelliSense segítségével.

Szükségünk van a node.js alkalmazás legfrissebb verziójára. A Node.js egy szoftverrendszer, melyet internetes alkalmazások, webszerverek készítésére hoztak létre. Az Angular CLI és az Angular alkalmazások az npm csomagoktól függenek számos szolgáltatás és funkció esetében. Az npm csomagok letöltéséhez és telepítéséhez npm csomagkezelőre van szükség. Az npm a JavaScript programozási nyelv csomagkezelője is. Az npm az alapértelmezett csomagkezelő a Node.js JavaScript futási környezethez. Telepítenünk kell az Angular CLI a parancssorból, az npm install -g @angular/cli parancs kiadásával. Az Angular CLI telepíti a szükséges Angular npm csomagokat és egyéb függőségeket. Ez eltarthat néhány percig. A CLI új munkaterületet és egy egyszerű üdvözlő alkalmazást hoz létre, futtatásra készen. Az Angular CLI tartalmaz egy kiszolgálót, így helyben kiépíthető és kiszolgálható az alkalmazás.

Mivel a szótár adatbázisból fog dolgozni, ezért egy ehhez szükséges adatbáziskezelő webszerver is szükséges, ez a XAMPP. A XAMPP – egy szabad és nyílt forrású platformfüggetlen webszerver-szoftvercsomag, amelynek legfőbb alkotóelemei az Apache webszerver, a MariaDB adatbázis-kezelő, valamint a PHP programozási nyelv értelmezői.

Amint megvan minden szükséges program elkezdhető az alkalmazás készítése.

2.2. Az alkalmazás fájlstruktúrája

Az Angular CLI ng new parancs létrehoz egy munkaterületet. A parancs futtatásakor a parancssori felület telepíti a szükséges Angular npm csomagokat és egyéb függőségeket egy új munkaterületre. A munkaterület gyökérmappája különféle támogatási és konfigurációs fájlokat, valamint egy testreszabható generált leíró szöveget tartalmazó README fájlt tartalmaz. Alapértelmezés szerint ng new létrehoz egy kezdeti csontvázalkalmazást a munkaterület gyökérszintjén, a végpontok közötti tesztekkel együtt.

Az egyik legnagyobb dolog a CLI-al kapcsolatban, hogy gondoskodik mindenről, amit meg kell tennie, egy új Angular projekt létrehozásától vagy szolgáltatások, összetevők vagy bármi mástól, amire a projektnek szüksége van. Egy dolog, amit sokan szeretnek, az az, hogy a CLI minden új alkatrészhez, szolgáltatáshoz, csőhöz stb. Társított tesztet is létrehoz. A CLI emellett szabványosított módon futtatja az összes tesztet is. Ez további rugalmasságot biztosít annak tudatában, hogy minden új fájlnak van tesztje, és frissítheti a teszteket a fejlesztés bármely szakaszában.

A munkaterületen belüli összes projekt osztozik egy CLI konfigurációs környezetben . A munkaterület legfelső szintje munkaterületre kiterjedő konfigurációs fájlokat, a gyökérszintű alkalmazás konfigurációs fájljait, valamint a gyökérszintű alkalmazásforrás és tesztfájlok almappáit tartalmazza. Ezek a konfigurációs fájlok a következők:

- .editorconfig – Konfiguráció a kódszerkesztőkhöz.
- .gitignore - Szándékosan nem követett fájlokat határoz meg.
- README.md - Bevezető dokumentáció a gyökéralkalmazáshoz.
- angular.json - A CLI konfigurációja alapértelmezés szerint a munkaterület összes projektjéhez, beleértve a CLI által használt build, kiszolgálás és teszteszközök konfigurációs beállításait tartalmazza.
- package.json - Konfigurálja a munkaterület összes projektjéhez elérhető npm csomagfüggőségeket .

- package-lock.json - Megadja node modules az npm kliens által telepített összes csomag verzióinformációját .
- src - Forrásfájlok a gyökérszintű alkalmazásprojekthez.
- node modules - Biztosítja NPM csomagokat az egész munkaterületre. A munkaterület egészére kiterjedő node modules függőségek minden projekt számára láthatók.
- tsconfig.json - Az alap konfiguráció a munkaterületen lévő projektekhez. Az összes többi konfigurációs fájl öröklődik erről az alapfájlról.
- tslint.json – Alapértelmezett konfiguráció a munkaterületen lévő projektekhez. [6]

Az src mappa tartalmazza a gyökéralkalmazás forrásfájljait. Ebben a mappában lévő fájlokkal dolgozunk főként az alkalmazás elkészítésekor. Az itt található almappák tartalmazzák az alkalmazás forrását és az alkalmazás-specifikus konfigurációt. Ezek:

- app - Tartalmazza azokat az összetevő fájlokat, amelyekben meghatározódik az alkalmazás logikája és adatai.
- assets - Kép- és egyéb eszközfájlokat tartalmaz, amelyeket az alkalmazás készítésekor a jelenlegi állapotban kell másolni.
- environments - Konfigurációs beállításokat tartalmaz az adott célkörnyezethez. Alapértelmezés szerint van egy meg nem nevezett szabványos fejlesztői környezet és egy termelési ("prod") környezet.
- favicon.ico - Az alkalmazáshoz az ikon, amely a könyvjelzősávon található.
- index.html - A fő HTML-oldal, amely akkor jelenik meg, amikor valaki felkeresi a webhelyet. A CLI automatikusan hozzáadja az összes JavaScript- és CSS-fájlt az alkalmazás felépítésekor, így általában nem kell semmilyen más <script>vagy <link>címkét hozzáadnia ide manuálisan.
- main.ts - Az alkalmazás fő belépési pontja. Fordítja az alkalmazást a JIT fordítóval, és az alkalmazás gyökérmodulját (AppModule) a böngészőben történő futtatáshoz indítja.
- polyfills.ts - Polifill szkripteket biztosít a böngésző támogatásához.

- styles.sass - Felsorolja azokat a CSS-fájlokat, amelyek stílusokat kínálnak egy projekthez. A kiterjesztés tükrözi azt a stílusprocesszort, amelyet a projekthez konfigurálunk.
- test.ts - Az egység fő belépési pontja, bizonyos szögspecifikus konfigurációval. [6]

A src mappában a app mappa tartalmazza a projekt logikáját és adatait. Az app mappa tartalma:

- app.component.ts - Meghatározza az alkalmazás gyökér összetevőjének logikáját.
- app.component.html - Meghatározza a gyökérhez társított HTML sablont.
- app.component.css - Meghatározza a gyökér CSS alapstílusát.
- app.component.spec.ts - Meghatározza az egység tesztjét a gyökér számára.
- app.module.ts - Meghatározza a gyökérmodult, amely megmondja az Angularnak az alkalmazás összeállítását. [6]

Az app mappán belül találhatóak még más különböző mappák, amelyek az alkalmazás beállítására szolgálnak. A components mappa tartalmazza a weboldal egyes elemeit. Elemek szerint külön mappákra bontva, a könnyebb átláthatóság érdekében. Minden elemnek saját HTML, CSS és TypeScrint fájlja van.

A fő mappán belül található még az e2e mappa. A e2e legfelső szintű mappa tartalmazza a gyökérszintű alkalmazásnak megfelelő végpontok közötti tesztkészlet forrásfájljait, valamint a tesztspecifikus konfigurációs fájlokat.

A db.config.js fájl exportálja a MySQL kapcsolat és a Sequelize konfigurációs paramétereit. A server.js fájl szolgál a REST API-k inicializálására és futtatására. Az összes CRUD művelet kezeléséért a routes.js fájl felelős.

2.3. Az alkalmazás programkódja

Számos fájl tartozik az alkalmazás megfelelő működéséhez. A legfontosabbakat emelem ki.

A szótár a következő API-kat biztosítja:

POST - új szótári bejegyzés készítése

GET - az összes vagy adott azonosítójú szótárbejegyzés kilistázása

PUT - a szótári bejegyzés módosítása

DELETE - szótári bejegyzés törlése

Ehhez először is a szükséges sablon alapú űrlapokat: *FormsModule, HttpClientModule* importáltuk az app.moduls.ts fájlban (2.1. melléklet). Beállításra kerültek a app-routing.module.ts (2.2. melléklet) fájlban a fő útvonalak. Valamint ide is számos komponenst importáltunk.

A app.component.html (2.3. melléklet) fájlban kerültek beállításra a fő menüpont elemei, *<nav>* - tagek, blokkok és felsorolások között. A fő modellosztályt a model.ts (2.4. melléklet) fájl tartalmazza,amelyek megfelelően típusokat is kaptak: *id: any, title: string, description: string; published: boolean;* A service.ts (2.5. melléklet) fájl tartalmazza a főbb adatszolgáltatásokat. Ez az Angular HTTPClienta szolgáltatást használja a kérések elküldéséhez. A funkciói CRUD műveleteket és kereső módszereket tartalmaznak.

Egy új szó hozzáadásáért a component.ts (2.6. melléklet) fájl felelős. Ennek az összetevőnek van egy űrlapja két mezővel: *title és description*. Amelyet a *TutorialService.create()* metódus hív meg. Begyűjti az adatokat, az állapotát pedig alapértelmezetten false-ra állítja. Ezeket az adatokat pedig a component.html (2.7. melléklet) fájlból gyűjt be.

A list.component.ts (2.8. melléklet) fájl szolgál az adatok kilistázásáért. Ehhez három TutorialService módszert hív meg: *getAll(); deleteAll(); findByTitle()*. Ez a kezdeti menüpont.

A details.component.ts (2.9. melléklet) fájl üzemelteti a szótárban a mentést, törlést és beillesztést. Ehhez szintén három összetevőt használ: *get(), update(), delete()*. Amikor elkészült a webszerver alapja, hozzáadjuk a MySQL adatbázis konfigurációját, modellt készítünk a Sequelize alkalmazással. Ezt követően pedig meghatározzuk az összes CRUD művelet kezelésének útvonalait.

Az alkalmazáshoz a következő modulok voltak szükségesek: *express, sequelize, mysql2 és body-parser*. Amelyet npm-en keresztül installáltunk. A Sequelize egy promise-alapu node server, amely szilárd tranzakciós támogatást, kapcsolatokat, gyors betöltést, olvasási replikációt és egyebeket tartalmaz.

A gyökérmappában található a server.js (2.10. melléklet) fájl. Ide három fő elem importálódott: az *express*, amely a Rest api építésére szolgál. A *body-parser* amely segít a

kérelem elemzésében és az *req.body* objektum létrehozásában. Valamint a cors, amely az a köztes szoftvert biztosítja a CORS számára különféle opciókkal. Az *app.use()* beállításánál történik meg az oldal 8081 portra való kapcsolása. Ez határozza meg a GET útvonalat.

Az alkalmazás mappában külön konfigurációs mappát hozunk létre a konfigurációhoz a db.config.js (2.11. melléklet) fájllal – ez tartalmazza az adatbázishoz szükséges adatokat, a hostot, felhasználónevet és jelszót, az adatbázis tábla nevét. A pool tartalmazza a Sequelize kapcsolatkészletét: a maximum és minumum csatlakozást és az időt.

A következő lépésben inicializáljuk a Sequelize alkalmazást a modellek mappában, az index.js (2.12. melléklet) fájlban. A model.js (2.13. melléklet) fájl beállításai szolgálnak a MySQL adatbázis tábla bemutatására. Ezeket az oszlopokat automatikusan generálja. A Sequelize inicializálása után nem kell CRUD függvényeket írni, a Sequelize mindegyiket támogatja: létrehozás *(create(object))*, keresés *(findByPk(id))*, összes klistázása *(findAll())*, frissítés *(update(data, where: id: id))*, eltávolítás *(destroy(where:))*.

A controller.js (2.14. melléklet) fájl tartalmazza az alapvető CRUD műveleteket: *create, findAll, findOne, update, delete, deleteAll, findAllPublished.*

Ha a böngésző HTTP-kérés (GET, POST, PUT, DELETE) segítségével végpontra vonatkozó kérést küld, akkor az útvonalak beállításával meg kell határoznunk, hogyan reagál a szerver, ezt tartalmazza a routes.js (2.15. melléklet) fájl a controller használatával.

A fájlok nem tartalmaznak hosszas CCS beállításokat, mivel az oldal kinézetéért a Bootstrap felelős.

Minden elem tartalma megtalálható a Mellékletekben.

2.4. A weboldal működése

Az elkészült weboldal csak helyi számítógépen fut, a http://localhost:8081 címen.

Az alkalmazás megnyitásakor felül három menüpont jelenik meg előttünk ez a : Szótár, Szerkesztés és Hozzáadás. A legelső menüpontban láthatjuk az összes szó listáját, valamint felül egy keresőt – ezt használva tudunk keresni a szavak között. A jelentéséhez pedig rá kell kattintanunk a kívánt szóra, ekkor jelenik meg ez nekünk oldalt. Amely alatt találunk egy „Szerkeszt ” gombot. Ez a gomb szolgál arra, ha valamelyik szót meg szeretnénk változtatni. Ezzel a gombbal átmegyünk a „Szerkesztés ” fülre. Itt megtaláljuk az adott szót, a jelentését, valamint három gombot: a Mentés, Törlés és Beillesztés. Itt

tudjuk kitörölni az adott szót vagy átírni, megváltoztatni, ha szeretnénk. Ekkor a szó Státusza is meg fog változni, szerkesztettre. A harmadik menüpont a „Hozzáadás” – itt tudunk új szót bevinni az adatbázisba.

Резюме

Під час своєї роботи я мала справу з веб-фреймворком Angular у створенні українсько-угорського словника. Оскільки сьогодні словники використовуються дуже часто, вони полегшують нам роботу в галузі мов.

Я вибрав веб-фреймворк Angular для створення словника, оскільки він має просту архітектуру, призначену для створення сучасних веб-додатків. Це нова структура, яка розвивається сьогодні і постійно вдосконалюється Google.

Для створення динамічного додатка не потрібно покладатися на зовнішній каталог. Забезпечує швидке та просте створення та вкладені файли, необхідні для програми. Angular використовує HTML для створення користувальницького інтерфейсу програми. А для створення зовнішнього вигляду можна використовувати CSS або іншу мову. Angular використовує TypeScript, що стало для мене величезною перевагою. TypeScript забезпечує кращі функції навігації та автозаповнення, його також можна налагодити безпосередньо у браузері або редакторі.

Тестування в Angular також надзвичайно просто. Він дотримується принципу стабільної структури файлів. Дозволяє ефективно підтримувати та оновлювати коди. З усіх доступних інтерфейсних фреймворків, Angular, ймовірно, займе найменше часу для прискорення. Існує багато факторів, які роблять Angular таким, щоб він мав короткий час навчання. Тож було гарною ідеєю вибрати цей веб-фреймворк для створення словника.

Összefoglalás

Munkám során az Angular webes keretrendszerrel foglalkoztam egy ukrán-magyar szótár elkészítése során. Mivel a szótárak napjainkban nagyon gyakran használatosak, megkönnyítik munkánkat a nyelvek terén.

A szótár elkészítéséhez azért választottam az Angular webes keretrendszert, mert egyszerű architektúrával rendelkezik, modern webalkalmazások létrehozására tervezték. Ez egy új napjainkban is fejlődő és a Google által folyamatosan tökéletesített keretrendszer.

Egy dinamikus alkalmazás készítéséhez nem kell külső könyvtárra támaszkodnunk. Biztosítja az alkalmazáshoz szükséges fájlok gyors és egyszerű létrehozását, egymásba ágyazását.

Az Angular HTML-t használ az alkalmazás felhasználói felületének elkészítéséhez. A kinézet kialakításához pedig használhatunk CSS-t, vagy egyéb nyelvet is. Az Angular natív módon használja a TypeScript-et , ami hatalmas előnyt jelentett számomra. A TypeScript lehetőséget nyújt a jobb navigációs és automatikus kiegészítési szolgáltatásokhoz. Valamint közvetlenül hibakereshető a böngészőben vagy a szerkesztőben.

Az Angularban a tesztelés is rendkívül egyszerű. Stabil fájlfelépítési elvet követ. Lehetővé tesz a kódok hatékony karbantartását és frissítését.

Az összes rendelkezésre álló front-end keretrendszer közül az Angular valószínűleg a legkevesebb időt veszi igénybe a felgyorsuláshoz. Nagyon sok olyan tényezője van, amelyek miatt az Angularnak rövid a tanulási ideje. Tehát jó ötlet volt ezt a webes keretrendszert választani a szótár elkészítéséhez.

Felhasznált irodalom

1. Адам Фримен. Pro Angular. Питер Пресс, Москва, 2020 - 800 с.

2. Nate Murray, Felipe Coury, Ari Lerner, and Carlos Taborda: *The Complete Guide to Angular.* California by Fullstack.io, San Francisco, 2020 - 88 o.

3. Yakov Fain, Anton Moiseev: *Angular Development with TypeScript.* Manning Publications, New York, 2019 - 560 o.

4. Sakshi Arora: *Angular Development Project.*

   Interneten:https://www.grazitti.com [hozzáférés dátuma: 2020.11.10.]

5. Bezkoder: *Angular 11 CRUD Application example with Web API.*

   Interneten: https://bezkoder.com [hozzáférés dátuma: 2021.01.20.]

6. Super-powered by Google: *Angular.*

   Interneten: https://angular.io [hozzáférés dátuma: 2020.12.10.]

7. ELTE: *Frotend keretrendszerek.*

   Interneten: http://nyelvek.inf.elte.hu [hozzáférés dátuma: 2021.01.20.]

8. Powered by Google: *Angular Material.*

   Interneten:https://material.angularjs.org [hozzáférés dátuma: 2021.03.01.]

9. Mark Techson: *Version 11 of Angular Now Available.*

   Interneten:https://blog.angular.io [hozzáférés dátuma:2021.04.10.]

10. Nwose Lotanna Victor: *What's New in Angular 11?*

    Interneten: https://www.telerik.com [hozzáférés dátuma: 2021.02.12.]

Mellékletek

2.1. melléklet

Az **app.moduls.ts** dokumentum tartalma

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';


import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AddTutorialComponent } from './components/add-tutorial/add-tutorial.
component';
import { TutorialDetailsComponent } from './components/tutorial-details/
tutorial-details.component';
import { TutorialsListComponent } from './components/tutorials-list/
tutorials-list.component';

@NgModule({
  declarations: [
    AppComponent,
    AddTutorialComponent,
    TutorialDetailsComponent,
    TutorialsListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2.2. melléklet

Az **app-routing.module.ts** dokumentum tartalma

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { TutorialsListComponent } from './components/tutorials-list/
tutorials-list.component';
import { TutorialDetailsComponent } from './components/tutorial-details/
tutorial-details.component';
import { AddTutorialComponent } from './components/add-tutorial/
add-tutorial.component';

const routes: Routes = [
  { path: '', redirectTo: 'tutorials', pathMatch: 'full' },
  { path: 'tutorials', component: TutorialsListComponent },
  { path: 'tutorials/:id', component: TutorialDetailsComponent },
```

```
14    { path: 'add', component: AddTutorialComponent }
15  ];
16
17  @NgModule({
18    imports: [RouterModule.forRoot(routes)],
19    exports: [RouterModule]
20  })
21  export class AppRoutingModule { }
```

2.3. melléklet

Az `app.component.html` dokumentum tartalma

```
1   <div>
2     <nav class="navbar navbar-expand navbar-dark bg-dark">
3       <a href="#" class="navbar-brand">Szotar</a>
4       <div class="navbar-nav mr-auto">
5         <li class="nav-item">
6           <a routerLink="tutorials" class="nav-link">Szerkesztes</a>
7         </li>
8         <li class="nav-item">
9           <a routerLink="add" class="nav-link">Hozzaadas</a>
10        </li>
11      </div>
12    </nav>
13
14    <div class="container mt-3">
15      <router-outlet></router-outlet>
16    </div>
17  </div>
```

2.4. melléklet

A `model.ts` dokumentum tartalma

```
1   export class Tutorial {
2     id?: any;
3     title?: string;
4     description?: string;
5     published?: boolean;
6   }
```

2.5. melléklet

A `service.ts` dokumentum tartalma

```
1   import { Injectable } from '@angular/core';
2   import { HttpClient } from '@angular/common/http';
3   import { Observable } from 'rxjs';
4   import { Tutorial } from '../models/tutorial.model';
5
6   const baseUrl = 'http://localhost:8080/api/tutorials';
7
8   @Injectable({
```

```
  providedIn: 'root'
})
export class TutorialService {
  constructor(private http: HttpClient) { }
  getAll(): Observable<Tutorial[]> {
    return this.http.get<Tutorial[]>(baseUrl);
  }
  get(id: any): Observable<Tutorial> {
    return this.http.get(`${baseUrl}/${id}`);
  }
  create(data: any): Observable<any> {
    return this.http.post(baseUrl, data);
  }
  update(id: any, data: any): Observable<any> {
    return this.http.put(`${baseUrl}/${id}`, data);
  }
  delete(id: any): Observable<any> {
    return this.http.delete(`${baseUrl}/${id}`);
  }
  deleteAll(): Observable<any> {
    return this.http.delete(baseUrl);
  }
  findByTitle(title: any): Observable<Tutorial[]> {
    return this.http.get<Tutorial[]>(`${baseUrl}?title=${title}`);
  }
}
```

2.6. melléklet

A `component.ts` dokumentum tartalma

```
import { Component, OnInit } from '@angular/core';
import { Tutorial } from 'src/app/models/tutorial.model';
import { TutorialService } from 'src/app/services/tutorial.service';

@Component({
  selector: 'app-add-tutorial',
  templateUrl: './add-tutorial.component.html',
  styleUrls: ['./add-tutorial.component.css']
})
export class AddTutorialComponent implements OnInit {
  tutorial: Tutorial = {
    title: '',
    description: '',
    published: false
  };
  submitted = false;
  constructor(private tutorialService: TutorialService) { }
  ngOnInit(): void {
  }
  saveTutorial(): void {
    const data = {
      title: this.tutorial.title,
      description: this.tutorial.description
    };
    this.tutorialService.create(data)
      .subscribe(
```

```
27          response => {
28            console.log(response);
29            this.submitted = true;
30          },
31          error => {
32            console.log(error);
33          });
34    }
35    newTutorial(): void {
36      this.submitted = false;
37      this.tutorial = {
38        title: '',
39        description: '',
40        published: false
41      };
42    }
43  }
```

2.7. melléklet

A `component.html` dokumentum tartalma

```
1  <div>
2    <div class="submit-form">
3      <div *ngIf="!submitted">
4        <div class="form-group">
5          <label for="title">Szo hozzaadasa</label>
6          <input
7            type="text"
8            class="form-control"
9            id="title"
10           required
11           [(ngModel)]="tutorial.title"
12           name="title"
13         />
14       </div>
15       <div class="form-group">
16         <label for="description">Jelentese</label>
17         <input
18           class="form-control"
19           id="description"
20           required
21           [(ngModel)]="tutorial.description"
22           name="description"
23         />
24       </div>
25       <button (click)="saveTutorial()" class="btn␣btn-success">Beilleszt</button>
26     </div>
27     <div *ngIf="submitted">
28       <h4>Sikeresen elk ldve!</h4>
29       <button class="btn␣btn-success" (click)="newTutorial()">Uj</button>
30     </div>
31   </div>
32 </div>
```

2.8. melléklet

A `list.component.ts` dokumentum tartalma

```
import { Component, OnInit } from '@angular/core';
import { Tutorial } from 'src/app/models/tutorial.model';
import { TutorialService } from 'src/app/services/tutorial.service';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-tutorials-list',
  templateUrl: './tutorials-list.component.html',
  styleUrls: ['./tutorials-list.component.css']
})
export class TutorialsListComponent implements OnInit {
  tutorials?: Tutorial[];
  currentTutorial?: Tutorial;
  currentIndex = -1;
  title = '';
  constructor(private tutorialService: TutorialService) { }
  ngOnInit(): void {
    this.retrieveTutorials();
  }
  retrieveTutorials(): void {
    this.tutorialService.getAll()
      .subscribe(
        data => {
          this.tutorials = data;
          console.log(data);
        },
        error => {
          console.log(error);
        });
  }
  refreshList(): void {
    this.retrieveTutorials();
    this.currentTutorial = undefined;
    this.currentIndex = -1;
  }
  setActiveTutorial(tutorial: Tutorial, index: number): void {
    this.currentTutorial = tutorial;
    this.currentIndex = index;
  }
  removeAllTutorials(): void {
    this.tutorialService.deleteAll()
      .subscribe(
        response => {
          console.log(response);
          this.refreshList();
        },
        error => {
          console.log(error);
        });
  }
  searchTitle(): void {
    this.currentTutorial = undefined;
    this.currentIndex = -1;
    this.tutorialService.findByTitle(this.title)
      .subscribe(
        data => {
```

```
57            this.tutorials = data;
58            console.log(data);
59          },
60          error => {
61            console.log(error);
62          });
63    }
64  }
```

2.9. melléklet

A `details.component.ts` dokumentum tartalma

```
1  import { Component, OnInit } from '@angular/core';
2  import { TutorialService } from 'src/app/services/tutorial.service';
3  import { ActivatedRoute, Router } from '@angular/router';
4  import { Tutorial } from 'src/app/models/tutorial.model';
5
6  @Component({
7    selector: 'app-tutorial-details',
8    templateUrl: './tutorial-details.component.html',
9    styleUrls: ['./tutorial-details.component.css']
10 })
11 export class TutorialDetailsComponent implements OnInit {
12   currentTutorial: Tutorial = {
13     title: '',
14     description: '',
15     published: false
16   };
17   message = '';
18   constructor(
19     private tutorialService: TutorialService,
20     private route: ActivatedRoute,
21     private router: Router) { }
22   ngOnInit(): void {
23     this.message = '';
24     this.getTutorial(this.route.snapshot.params.id);
25   }
26   getTutorial(id: string): void {
27     this.tutorialService.get(id)
28       .subscribe(
29         data => {
30           this.currentTutorial = data;
31           console.log(data);
32         },
33         error => {
34           console.log(error);
35         });
36   }
37   updatePublished(status: boolean): void {
38     const data = {
39       title: this.currentTutorial.title,
40       description: this.currentTutorial.description,
41       published: status
42     };
43     this.message = '';
44     this.tutorialService.update(this.currentTutorial.id, data)
```

```
45        .subscribe(
46          response => {
47            this.currentTutorial.published = status;
48            console.log(response);
49            this.message = response.message ? response.message :'Feltoltve';
50          },
51          error => {
52            console.log(error);
53          });
54    }
55    updateTutorial(): void {
56      this.message = '';
57      this.tutorialService.update(this.currentTutorial.id, this.currentTutorial)
58        .subscribe(
59          response => {
60            console.log(response);
61            this.message = response.message ? response.message : 'Sikeresen mentve';
62          },
63          error => {
64            console.log(error);
65          });
66    }
67    deleteTutorial(): void {
68      this.tutorialService.delete(this.currentTutorial.id)
69        .subscribe(
70          response => {
71            console.log(response);
72            this.router.navigate(['/tutorials']);
73          },
74          error => {
75            console.log(error);
76          });
77    }
78  }
```

2.10. melléklet

A `server.js` dokumentum tartalma

```
1  const express = require("express");
2  const bodyParser = require("body-parser");
3  const cors = require("cors");
4  const app = express();
5  var corsOptions = {
6    origin: "http://localhost:8081"
7  };
8  app.use(cors(corsOptions));
9  app.use(bodyParser.json());
10 app.use(bodyParser.urlencoded({ extended: true }));
11 const db = require("./app/models");
12 db.sequelize.sync();
13 app.get("/", (req, res) => {
14   res.json({ message: "Welcome␣to␣bezkoder␣application." });
15 });
16 require("./app/routes/turorial.routes")(app);
17 const PORT = process.env.PORT || 8080;
18 app.listen(PORT, () => {
```

```
19    console.log(`Server is running on port {PORT}.`);
20  });
```

2.11. melléklet

A `db.config.js` dokumentum tartalma

```
1  module.exports = {
2    HOST: "localhost",
3    USER: "root",
4    PASSWORD: "",
5    DB: "testdb",
6    dialect: "mysql",
7    pool: {
8      max: 5,
9      min: 0,
10     acquire: 30000,
11     idle: 10000
12   }
13 };
```

2.11. melléklet

Az `index.js` dokumentum tartalma

```
1  const dbConfig = require("../config/db.config.js");
2  const Sequelize = require("sequelize");
3  const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
4    host: dbConfig.HOST,
5    dialect: dbConfig.dialect,
6    operatorsAliases: false,
7
8    pool: {
9      max: dbConfig.pool.max,
10     min: dbConfig.pool.min,
11     acquire: dbConfig.pool.acquire,
12     idle: dbConfig.pool.idle
13   }
14 });
15 const db = {};
16 db.Sequelize = Sequelize;
17 db.sequelize = sequelize;
18 db.tutorials = require("./tutorial.model.js")(sequelize, Sequelize);
19 module.exports = db;
```

2.13. melléklet

A `model.js` dokumentum tartalma

```
1  module.exports = (sequelize, Sequelize) => {
2    const Tutorial = sequelize.define("tutorial", {
3      title: {
4        type: Sequelize.STRING
5      },
```

```
    description: {
      type: Sequelize.STRING
    },
    published: {
      type: Sequelize.BOOLEAN
    }
  });
  return Tutorial;
};
```

2.14. melléklet

A `controller.js` dokumentum tartalma

```
const db = require("../models");
const Tutorial = db.tutorials;
const Op = db.Sequelize.Op;
exports.create = (req, res) => {
  if (!req.body.title) {
    res.status(400).send({
      message: "Content can not be empty!"
    });
    return;
  }
  const tutorial = {
    title: req.body.title,
    description: req.body.description,
    published: req.body.published ? req.body.published : false
  };
  Tutorial.create(tutorial)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Hiba tortent."
      });
    });
};
exports.findAll = (req, res) => {
  const title = req.query.title;
  var condition = title ? { title: { [Op.like]:

  Tutorial.findAll({ where: condition })
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Hiba tortent."
      });
    });
};
exports.findOne = (req, res) => {
  const id = req.params.id;
```

```
44
45    Tutorial.findByPk(id)
46      .then(data => {
47        res.send(data);
48      })
49      .catch(err => {
50        res.status(500).send({
51          message: "Error␣retrieving␣Tutorial␣with␣id=" + id
52        });
53      });
54  };
55  exports.update = (req, res) => {
56    const id = req.params.id;
57    Tutorial.update(req.body, {
58      where: { id: id }
59    })
60      .then(num => {
61        if (num == 1) {
62          res.send({
63            message: "Feltoltve."
64          });
65        } else {
66          res.send({
67            message: `Nem talalhato id={id}.`
68          });
69        }
70      })
71      .catch(err => {
72        res.status(500).send({
73          message: "Hiba␣tortent" + id
74        });
75      });
76  };
77  exports.delete = (req, res) => {
78    const id = req.params.id;
79
80    Tutorial.destroy({
81      where: { id: id }
82    })
83      .then(num => {
84        if (num == 1) {
85          res.send({
86            message: "Torolve"
87          });
88        } else {
89          res.send({
90            message: `Nem lehet torolni={id}.`
91          });
92        }
93      })
94      .catch(err => {
95        res.status(500).send({
96          message: "Nincs␣torolve=" + id
97        });
98      });
99  };
100
101 exports.deleteAll = (req, res) => {
```

```
102   Tutorial.destroy({
103     where: {},
104     truncate: false
105   })
106     .then(nums => {
107       res.send({ message: `{nums} Torlese` });
108     })
109     .catch(err => {
110       res.status(500).send({
111         message:
112           err.message || "Hiba␣tortent."
113       });
114     });
115 };
116 exports.findAllPublished = (req, res) => {
117   Tutorial.findAll({ where: { published: true } })
118     .then(data => {
119       res.send(data);
120     })
121     .catch(err => {
122       res.status(500).send({
123         message:
124           err.message || "Hiba␣tortent."
125       });
126     });
127 };
```

2.15. melléklet

A `routes.js` dokumentum tartalma

```
1  module.exports = app => {
2    const tutorials = require("../controllers/tutorial.controller.js");
3    var router = require("express").Router();
4    router.post("/", tutorials.create);
5    router.get("/", tutorials.findAll);
6    router.get("/published", tutorials.findAllPublished);
7    router.get("/:id", tutorials.findOne);
8    router.put("/:id", tutorials.update);
9    router.delete("/:id", tutorials.delete);
10   router.delete("/", tutorials.deleteAll);
11   app.use('/api/tutorials', router);
12 };
```

UNICHECK

Ім'я користувача:
Моца Андрій Андрійович

ID перевірки:
1007785637

Дата перевірки:
09.05.2021 00:28:17 EEST

Тип перевірки:
Doc vs Internet

Дата звіту:
09.05.2021 00:58:11 EEST

ID користувача:
100006701

Назва документа: Kerek_Vanessza_Angular_Webes_Keretrendszer_alkalmazása_ukrán-magyar_elektronikus_szótár_front_end.

Кількість сторінок: 32 Кількість слів: 9934 Кількість символів: 54452 Розмір файлу: 633.04 KB ID файлу: 1007884564

0.1%
Схожість

Найбільша схожість: 0.1% з Інтернет-джерелом (https://docplayer.hu/109119126-Limes-a-ii-rakoczi-ferenc-karpataljai-m...

0.1% Джерела з Інтернету | 7 ..... Сторінка 34

Пошук збігів з Бібліотекою не проводився

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 1

Nyilatkozat

Alulírott, Kerek Vanessza 014. Középiskolai oktatás (Matematika) képzési program hallgatója, kijelentem, hogy a dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskolán, a Matematikai és Informatika Tanszéken készítettem, 014. Középiskolai oktatás (Matematika) BSc diploma megszerzése végett.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola könyvtárában a kölcsönözhető könyvek között helyezik el.