Szakmai gyakorlat

Drótos Zoltán

Eötvös Loránd Tudományegyetem Informatikai Kar

ABSTRACT

A szakmai gyakorlat során állami vállalatok adatait tartalmazó adatbázist hoztunk létre SAP Hana architektúrában, amelyhez hozzáférést biztosítottunk egy OData szolgáltatás segítségével. Ezt felhasználva Angular web alkalmazás készült, amely frontendként működve felhasználóbarát frontendet biztosít az adatbázis számára. A web alkalmazás lehetővé teszi cégek területi hierarchia szerinti megjelenítését, illetve az ezek között történő szabadszavas keresést. A keresés több különböző kategória szerint történik.

2019. Szeptember 4.

Tartalomjegyzék

1. Bevezetés

A nyári gyakorlat első két hete pilot projekt készítésével telt, ahol SAP Hana backendet, illetve egy minimálisan működő Angular frontendet hoztunk létre. Állami vállalatok adatait kaptuk meg évekre lebontva, CSV formátumban. Ezt némi adattisztítás után importáltuk, majd normalizáltuk. Ezek után Odata szolgáltatást hoztunk létre, amely segítségével a megfelelő URL-en keresztül lekérdezhetővé vált az adatbázis külső programok számára is. Ezek után készítettünk egy Angular alkalmazást, aminek egyetlen célja a backenddel történő kommunikáció demonstrálása volt. A pilot projekt után kisebb csapatok alakultak, amik már specifikusabban foglalkoztak külön feladatokkal. Az én feladatom frontend készítése volt a pilot projektben készített backend számára.

2. Feladat

Az általunk használt adathalmaz az Országleltár által megjelenített adatok egy változata. Mintaként is ez szolgált a készülő webes alkalmazásnak, és a követelmények is ennek tükrében fogalmazódtak meg.

2.1. Követelmények

Az országleltár interfésze lényegében három részből áll: területi hierarchia szerint rendezett táblázatok csoportjából, a kiválasztott terület adatait megjelenítő grafikon(ok)ból, illetve a kiválasztott területet megjelenítő térképből. Az általam elkészített alkalmazásnak is meg kellett jelenítenie ezeket az információkat, illetve biztosítania kellett egy kereső funkciót is, amellyel képesek vagyunk kategória szerint keresni, illetve szabadszavas, azaz substringekre is kereshetünk vele. Elvárás volt még, hogy az alkalmazás összehangoltan jelenítse meg a releváns információkat, illetve hogy maga az alkalmazás cross-platform legyen.

Összegezve tehát:

- Területi hierarchia szerint megjelenő táblázatok
- Diagram
- Térkép
- Kereső
- Az információ összehangolt megjelenése
- Cross-platform

3. Felhasznált technológiák

A követelmények ismeretében ideje volt kiválasztani a megvalósításhoz szükséges technológiát, framework-öket, és könyvtárakat.

3.1. Technológiai lehetőségek

Az alkalmazás cross-platform jellege miatt előnyt élveznek a webes technológiák, mint az Angular, React, Vue JavaScript framework-ök. Felmerült még a Java lehetősége is, mivel a JVM miatt nagyon sok platformon futtatható minimális kódváltoztatás mellett. Szóba került még az SAP saját készítésű JavaScript framework-je, az SAPUI5 is. Mivel a pilot alkalmazás Angularban készült, és így azzal volt már kis tapasztalatom, a többi technológiával viszont nem, így az Angular használata mellett döntöttem.

3.2. Angular

3.2.1. Architektúra

Az Angular egy olyan framework, aminek segítségével könnyedén készíthetők kliens alkalmazásokat HTML-ben és TypeScriptben. Maga az Angular TypeScriptben íródott. Az Angular alkalmazások úgynevezett *NgModule*-okból állnak, amik még tovább bonthatóak komponensekre. Egy Angular alkalmazás mindig áll legalább egy root modulból. A komponensek definiálhatnak nézeteket, illetve használhatnak szolgáltatásokat is. A szolgáltatásokat függőségként hozzáadhatjuk különböző komponensekhez, ezzel biztosítva az alkalmazás modularitását. A különböző nézetek között egy *Router* szolgáltatás segítségével

navigálhatunk.

3.2.2. Komponensek

Minden Angular alkalmazás áll legalább egy gyökér (*root*) komponensből, illetve tetszőleges egyéb komponensből is. A komponenseket általában funkciók szerint bontjuk szét. A komponensek egy hierarchiába vannak rendezve, amiket a root komponens köt össze. A komponensek lényegében **osztályokat** és a hozzájuk tartozó **HTML sablonokat** (*template*), illetve **CSS fájlokat** jelentik. Ahhoz, hogy egy osztályt komponensként jelöljünk meg, a *@Component()* dekorátort kell használnunk. A dekorátorokkal metaadatokat is megadhatunk, amik az Angular számára jelzésértékűek:

```
@Component({
  selector: 'app-dummy',                // HTML selector
  templateUrl: 'dummy.component.html',  // sablonok
  styleUrls: ['dummy.component.scss']   // stíluslapok
})
export class DummyComponent implements OnInit {
  ...
}
```

3.2.3. Sablonok

Egy komponens megjelenítéséhez szükség van valamilyen sablonra. Ez alapértelmezetten valamilyen HTML fájlt jelent, de lehet sablon például PUG vagy SVG fájl is. A sablonok segítségével olyan kifejezéseket helyezhetünk el a szövegben, melyeket az Angular futás közben kiértékel, és ezáltal dinamikusan generálhatunk tartalmakat, illetve kezelhetünk eseményeket.

3.2.3.1. Eseménykezelés

Különböző HTML elemek eseményeit lekezelhetjük az alkalmazásban. Például a felhasználó kiválaszt egy legördülő listából egy évszámot, és az alkalmazás az az évi adatokat fogja lekérni, majd megjeleníteni.

```
<!-- dummy.component.html -->
<select (change)="yearSelected($event.target.value)">
    <option value="2017">2017</option>
    <option value="2018">2018</option>
    <option value="2019">2019</option>
</select>
```

```
/* dummy.component.ts */
yearSelected(n: number): void {
    this._year = n;
}
```

3.2.3.2. Direktívák

Direktívák segítségével az Angular képes feltételek szerint tartalmat renderelni, vagy dinamikusan adatokat megjeleníteni. Két ilyen gyakrabban használt direktíva az **ngIf* és **ngFor* direktívák. Az alábbi példában szereplő táblázat csak akkor jelenik meg az oldalon, amennyiben `showTable: boolean` logikai változó értéke igaz. Amennyiben a változó értéke igaz, úgy ezután az **ngFor* direktíva hatására az Angular végigiterál a `data: any[]` tömbön, és minden elemre kiírja a táblázat megfelelő oszlopába a megfelelő értékeket. A generálás így az összes kapott adatra képes a táblázatban egy sort generálni, így kikerülve a HTML táblázatok statikusságát.

```
<div *ngIf="showTable">
  <table>
    <tr class="table-heading">
      <th>Név</th>
      <th>Cím</th>
      <th>Vagyon</th>
    </tr>
    <tr *ngFor="let e of data">
      <td>{{e.MEGYE}}</td>
      <td>{{e.CIM}}</td>
      <td>{{e.VAGYON}}</td>
    </tr>
  </table>
</div>
```

3.2.3.3. Interpoláció

Az interpolációs szintaxis teszi lehetővé alkalmazásbeli adatok megjelenítését. Szintaxisa:

```
{{expression}}
```

Például a fenti példában generált táblázat elemeit is interpoláció segítségével jelenítjük meg.

3.2.3.4. Pipes

A pipe-ok felhasználásával adatok megjelenését formatálhatjuk az interpolációs szintaxis alkalmazásánál. Külön formázási lehetőségek léteznek például pénznemek, dátumok, nagy számok megjelenítésére. Nem csak beépített pipe-ok léteznek, mi is definiálhatunk sajátot.

```
/* dummy.component.ts */
let wealth: number = 18446744;
```

```
<!-- dummy.component.html -->
{{ wealth | number }}
```

```
output: 18,446,744
```

3.2.4. Szolgáltatások

A szolgáltatások olyan osztályok, amelyek segítségével különböző adatokat oszthatunk meg komponensek között. Egy osztály szolgáltatáskénti megjelölésére az *@Injectable()* dekorátor szolgál. Itt ugyanúgy megadhatunk metaadatokat mint a komponensekre vonatkozó dekorátoroknál.

```
@Injectable({
  providedIn: 'root'
})
export class DummyService {
  ...
}
```

Ahhoz, hogy egy komponens elkezdhessen használni egy szolgáltatást, át kell adnunk a szolgáltatást a komponens konstruktorában.

```
@Component({
  selector: 'app-dummy',
  templateUrl: 'dummy.component.html',
  styleUrls: ['dummy.component.scss']
})
export class DummyComponent implements OnInit {
  constructor(private dummyService: DummyService) {}
  ...
}
```

4. Felhasználói leírás

Az alkalmazást webes felületen tudjuk elérni tetszőleges böngészőprogram segítségével. Az oldal helyes működéséhez fontos, hogy a böngészőnk támogassa JavaScript futtatását. Az oldal négy részből áll: bal oldalt táblázatok kaptak helyet, ahol területi hierarchia szerint böngészhetünk, jobb oldalt pedig egy kereső helyezkedik el, illetve alatta egy diagram és egy térkép.

4.1. Táblázat

A táblázat területi hierarchia szerint jelenít meg adatokat. A komponsens három táblázatból áll, melyek egymás alatt helyezkednek el. Mindhárom táblázat három oszlopot tartalmaz: a terület nevét, az ott található vállalatok összvagyonát, illetve a bejegyzett vállalatok számát. Alapértelmezetten csak a régió tábla jelenik meg. Ha ebben rákattintunk egy sorra, megjelenik egy új táblázat az adott régión belüli megyékről. Ha itt kiválasztunk egy megyét, akkor megjelenik egy új táblázat a megyén belüli településekről. Ha bárhol új bejegyzést választunk, akkor csak a hierarchia szerint releváns táblázatok maradnak láthatóak. Ha a kiválasztott régióra mégegyszer rákattintunk, akkor a régió kijelölését megszüntetjük, és újból az egész országot vizsgáljuk. A táblázatban kiválasztott elemeket a weboldal többi része is tükrözi: a térképen kékre szineződik a vizsgált terület, és a diagrammon is megjelennek a területre vonatkozó adatok. A keresőben található lista csak a területen belüli cégeket mutatja, ilyenkor keresni is csak ezek között tudunk. A táblázat ilyen módon lehetővé teszi a területi hierarchia szerinti böngészést. A táblázat felett helyet kapott egy évválasztó is. Évválasztásnál minden komponens frissíti önmagát, hogy a megfelelő adatokat mutassa.

4.2. Kereső

A kereső a kiválasztott területen található cégek közötti szabadszavas keresést teszi lehetővé. Ha nincs kiválasztva külön terület, akkor az egész ország területén keresünk. Ilyenkor a térképen sem szerepel színezett elem. Keresés indításához elegendő elkezdenünk beírni a keresett szöveget; amint szünetet tartunk a gépelésben, a keresés automatikusan lefut. Lehetőség van kategória szerinti keresésre is. A keresés kategóriáját a keresőmező alatti legördülő menüből tudjuk kiválasztani; az alapértelmezett viselkedés a név szerinti keresés.

5. Fejlesztői leírás

Az országleltárt alapul véve amellett döntöttem, hogy egyetlen nézetet fogok készíteni a frontendhez. Ebben a nézetben kap helyet a négy megvalósítandó funkció:

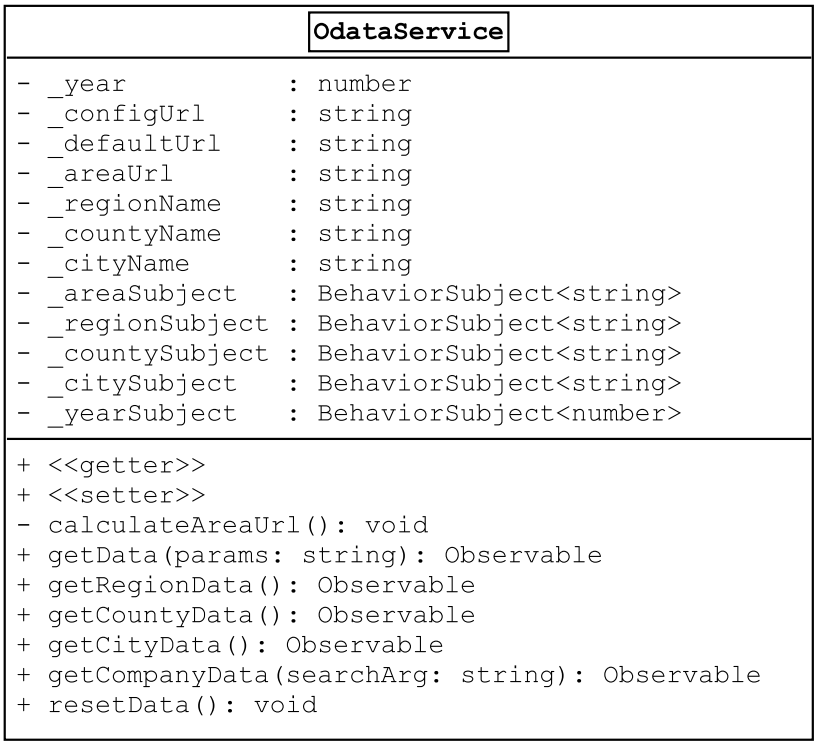
- Diagram
- Táblázatok
- Térkép
- Kereső

A funkciók szétválasztásának elvét követve az alkalmazás négy külön komponensből áll. A komponensek által megjelenített adatokat egy szolgáltatás biztosítja (*OdataService*), ami a szerverről történő lekérdezésekért, illetve közös adatok tárolásáért felelős. A szolgáltatás lényegében összeköti a különböző komponenseket, és ez lehetőséget is biztosít az oldal összehangolt működésére, így a követelmények minden pontja teljesül.

5.1. Szolgáltatás

Az alkalmazás egy szolgáltatás segítségével kommunikál az SAP szerverrel. A szolgáltatás HTTP lekérdezéseket küld a szervernek, a szerver pedig XML, vagy JSON formátumban küldi el válaszként a lekérdezett adatokat. Mindezt a szerver által biztosított OData szolgáltatás teszi lehetővé.

Az alkalmazás komponensei a szolgáltatáson keresztül mindig lekérdezhetik a számukra releváns adatokat egyszerű metódusokon keresztül. A szolgáltatás a lekérdezéseken kívül olyan adatokat is tárol, melyek több komponens számára is fontosak. Ezeket az adatokat úgynevezett **BehaviourSubjectek**-ben tárolódnak. Amikor egy komponens létrejön, feliratkozik a számára releváns subjectekre. Ezután amikor egy adat változik, a hozzá tartozó subject next metódusát meghívva a szolgáltatás lépteti azt, és a rá feliratkozott komponensek frissítik a saját adataikat, illetve a megjelenítés is frissül.

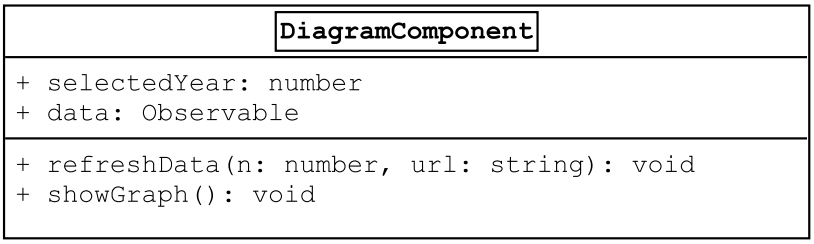


- **calculateAreaUrl():** megvizsgálja, mely területnevek üres stringek, és ez alapján kiszámolja a legkisebb kiválasztott területet, és ha ez a terület változott, lépteti _areaSubjectet a frissült értékkel.

- **getData(params: string):** A paraméterül kapott stringet lekérdezésként továbbítja a webszerver felé. Minden lekérdezés rajta keresztül történik.
- **getRegionData(): getDatát** hívja meg, a régiótábla lekérdezéséhez szükséges stringgel. A tábla komponens hívja meg.
- **getCountyData(): getDatát** hívja meg, a megyetábla lekérdezéséhez szükséges stringgel. A tábla komponens hívja meg.
- **getCityData(): getDatát** hívja meg, a településtábla lekérdezéséhez szükséges stringgel. A tábla komponens hívja meg.
- **getCompanyData(searchArg: string):** Ez a metódus megvizsgálja, mely régió, megye, település van kiválasztva (ha van kiválasztva), és ezek alapján készít egy stringet, amelyet paraméterként adva **getDatának** elvégez egy lekérdezést. A lekérdezés eredményét ezután visszaadja. Ezt a metódust a kereső komponens hívja meg.
- **resetData():** Alaphelyzetbe állítja a területneveket, illetve a kijelölt területet. Akkor hívódik meg, amikor a kiválasztott régió deaktiválódik.

5.2. Diagram

A diagram komponens lényegében egy CanvasJS kördiagram, amely a kiválasztott év és terület szerint ábrázolja az állami vállalatok jegyzett tőkéjét. A komponens az OData szolgáltatás **_yearSubject** és **_areaSubject** változóira iratkozik fel, és így ha azok az alkalmazás egyéb komponenseiben változnak, úgy a diagram is követi a változásokat a **refreshData()** metódus meghívásával, és frissül.



5.3. Táblázat

A táblázatban egy terület kiválasztása az OData szolgáltatásban beállítja az adott régió nevét a setterek segítségével. Az OData szolgáltatásban található setterek nem csak az adott változót állítják be a paraméterül kapott értékre, hanem a **calculateAreaUrl()** metódussal újra is számolják a legkisebb kiválasztott területet, illetve léptetik a megfelelő subjecteket. Ekkor minden komponens, amely feliratkozott az adott subjectekre, elvégzi a kívánt műveletet (például a térkép frissíti a kiválasztott terület háttérszínét, a diagram frissül, stb.)

A táblázatok feltételes megjelenése az Angular ***ngIf** direktívájának köszönhető: amennyiben valamelyik táblázatban megjelenítendő adat **null**, úgy a táblázat nem fog megjelenni. Ha például az egész országot szemléljünk, elég régiókat listáznunk, nincs értelme megyéket vagy településeket listáznunk; ilyenkor a megfelelő táblák adatait tároló változóknak elég null értéket adnunk, hogy ne jelenjenek meg.

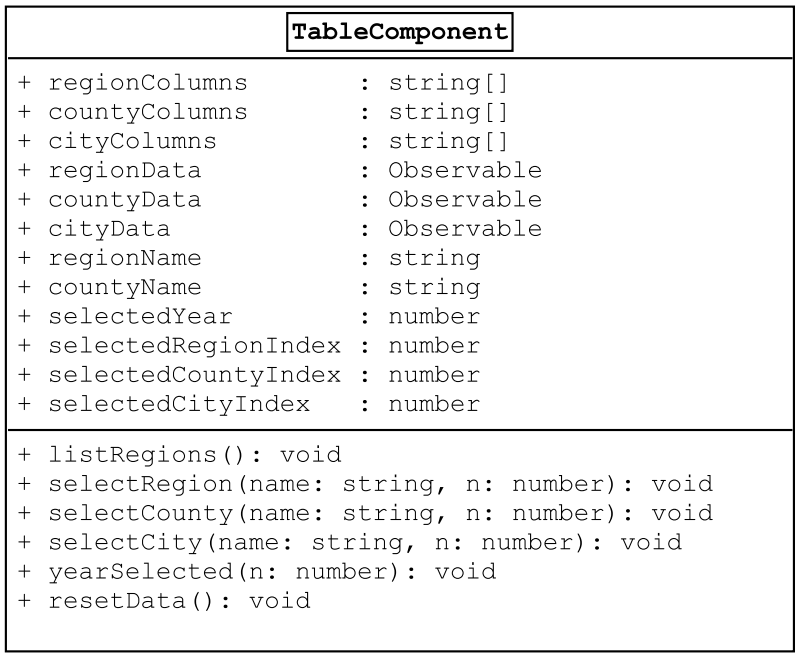
A táblázatok sorainak dinamikus generálásáért az ***ngFor** direktíva felel. Ez lehetővé teszi hogy a táblázat adatait tároló változókon (amelyek objektumokat tároló tömbök) végig tudjunk iterálni, és minden egyes bejegyzéshez külön sort tudjunk generálni a HTML táblázaton belül.

Maguk a táblázatok által megjelenített adatok aggregációs lekérdezések eredményei, így az eredeti backendet használva a szervernek mindig újra el kellett volna végeznie ezeket a számításokat. Ez az adatbázis gyakori változása mellett nem lett volna rossz megoldás, azonban a projekt által használt adatbázis értékei nincsenek kitéve ilyen változásoknak, így célszerűbb volt létrehozni három új táblát az aggregált értékekkel, amelyeket a táblázatok az OData szolgáltatáson keresztül lekérhetnek, elkerülve így a felesleges számításokat.

REGIONS		
NAME	**SQL Data Type**	**Column Store Data Type**
REGIO	NVARCHAR(255)	STRING
ERTEK	DOUBLE	DOUBLE
MENNYISEG	BIGINT	FIXED
EV	INTEGER	INT

COUNTIES		
NAME	**SQL Data Type**	**Column Store Data Type**
REGIO	NVARCHAR(255)	STRING
MEGYE	NVARCHAR(255)	STRING
ERTEK	DOUBLE	DOUBLE
MENNYISEG	BIGINT	FIXED
EV	INTEGER	INT

CITIES		
NAME	**SQL Data Type**	**Column Store Data Type**
REGIO	NVARCHAR(255)	STRING
MEGYE	NVARCHAR(255)	STRING
TELEPULES	NVARCHAR(255)	STRING
ERTEK	DOUBLE	DOUBLE
MENNYISEG	BIGINT	FIXED
EV	INTEGER	INT



A **listRegions()** metódus akkor hívódik meg, amikor az egész országot vizsgáljuk, tehát az alkalmazás legelső betöltésekor, illetve amikor egy kiválasztott régió kijelölését megszüntetjük.

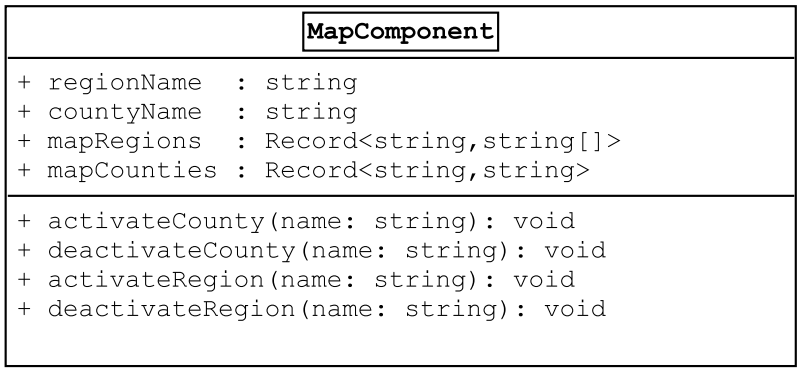
A **select*** metódusokat a HTML táblázat **click** eseménye váltja ki. A **select*** metódusok meghívják az OData szolgáltatás megfelelő **setter** metódusait, ezzel frissítve az aktuális területnevet illetve a kijelölt legszűkebb területet. A **yearSelected()** metódus az évválasztás eseményt lekezelve frissíti az OData szolgáltatás év adatát. A **selectRegion()** metódus ezen kívűl még vizsgálja, hogy a kattintott régió aktív-e már. Ha igen, akkor meghívja a **resetData()** metódust, amely az év kivételével alapállapotba állítja minden szűrőt, az egész ország kijelölését eredményezve.

5.4. Térkép

A térkép feladata a kiválasztott régiók és megyék vizuális megjelenítése. Az oldal betöltésekor, amikor még az egész ország ki van jelölve, a térképen csak Magyarország megyéinek körvonalait láthatjuk. Amikor elkezdünk a táblázat segítségével régiókat, illetve megyéket kiválasztani, úgy fog a térképen is kék háttérszínt kapni a kiválasztott régió, vagy megye.

Ennek megvalósításához egy SVG fájlt használtam a térkép komponens sablonaként. Az SVG fájl poligonokat határoz meg, amelyek megyéket határolnak, és minden poligonhoz rendel egy HTML osztályt (*class*) illetve azonosítót (*id*). A komponenshez tartozó TypeScript állományban minden megye nevéhez hozzárendeljük az őt reprezentáló poligon azonosítóját, a **mapCounties** változóban. Így amikor a felhasználó a táblázat segítségével kiválaszt valamilyen területet, a térkép az ahhoz tartozó azonosító alapján aktívra állítja a poligon objektumot, ami így a stíluslapban meghatározott háttérszínt fog kapni.

A régiók és az őket alkotó megyék között is történik egy hozzárendelés, ezt a **mapRegions** változó tárolja el. Régiók kiválasztásánál így a térkép a fenti módszerrel végig tud iterálni a régiót alkotó megyéken, aktívra állítva őket.

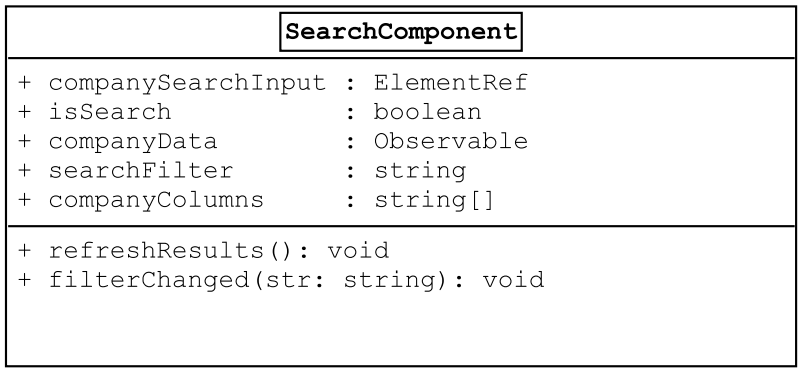


Az **activateCounty(name: string)** metódus paraméterül egy megye nevét kapja, és a **mapCounties-ban** a megyéhez rendelt poligon objektumot aktívra állítja. A **deactivateCounty(name: string)** metódus ugyanezen az elven működik, de ő az aktív állapot megszüntetéséért felel. Az **activateRegion(name: string)** metódus régiónevet kap paraméterül, a **mapRegions** által azonsított megyéket pedig aktívra állítja a **deactivateCounty(name: string)** metódus többszöri meghívásával. **deactivateRegion(name: string)** a paraméterül kapott régiót ugyanezzel a módszerrel deaktiválja, **deactivateCounty(name: string)** többszöri meghívásával.

5.5. Kereső

A komponens egy searchbarból, egy legördülő menüből, illetve egy táblázatból áll. A táblázat az ***ngFor** direktívával jön létre, dinamikusan generált. A legördülő menü három kategória szerinti keresést enged meg:

- Név szerint
- Cím szerint
- Adószám szerint



A komponens konstruktorában helyet kapott egy függvényhívás, amely a keresőmező változásait figyeli. Ehhez **rxjs** operátorokat használtam. A függvényhívásban paraméterezhető, hogy hány karaktert kell minimum beírnia a felhasználónak a keresés elindításához, illetve hogy mennyi idő teljen el a keresés indítása és az utolsó billentyűleütés között. A komponens a konstruktorában feliratkozik a régió, megye, illetve településnevek változásaira; amint ezek változnak, frissíti a keresés reedményeit tartalmazó táblázatot, a **refreshResults()** metódus segítségével. Mivel ez a metódus az OData szolgáltatás **getCompanyData(searchArg: string)** metódusát használja, a keresőmezőbe beírt keresőszöveg továbbra is szűri a lekérdezés eredményeit. A **filterChanged(str: string)** metódus a szűrés kategóriáját állítja be; ez a metódus lényegében a sablonban található selectbox változásainak eseményét kezeli le.

6. Fejlesztési tapasztalatok

6.1. SAP HANA

A HANA backend az elkészülését követően megbízhatóan és gyorsan működött. A kezdeti beállítások azonban nem történtek zökkenőmentesen. Véleményem szerint az SAP dokumentációja sok esetben nem fogalmaz elég világosan vagy lényegretörően. A tutorialok magas színvonalúak szerintem, de sok esetben nem naprakészek. Például a frontend készítésére vonatkozó tutorial is még az Angular régi verzióját használta, miközben a framework számos változáson esett át, és a használata mára teljesen átalakult. Sok probléma csak a HANA verziói közti váltással oldódott meg, ilyen eset volt a kalkulációs nézet létrehozása is. Azonban a kezdeti nehézségek leküzdése után a HANA működésére, illetve teljesítményére nem volt panasz.

6.2. Angular

Angularban dolgozni számomra kellemes meglepetés volt. A framework okos absztrakciók használatával teszi lehetővé webes felületek elkészítését, és a sablonok használatával nem csak HTML állományokat használhatunk fel az alkalmazásunk megjelenítésére, hanem sok egyéb formátumot is, mint például az SVG, vagy PUG formátumok. A sablonok ezen rugalmassága jelentősen megkönnyítette a térkép SVG fájllal történő megvalósítását. Külön tetszett, hogy natív JavaScript helyett statikusan típusozott TypeScriptet használhattam. A framework teljesítményét is elég gyorsnak találtam. Az Angularnak is vannak hátrányai. Véleményem szerint a legegyszerűbb projektekhez is nagyon nagy méretű projektmappákat generál, több tízezer fájllal, illetve sok időbe telik elsajátítani; ez részben az Angular saját absztrakcióinak köszönhető, amik függetlenítik a statikus HTML oldalaktól.

7. Továbbfejlesztési lehetőségek

Bár a projekt végeredményeként kapott szoftver kielégíti a kezdetben megfogalmazott elvárásokat, számos módon továbbfejleszthető még. Az egyik ilyen lehetőség a térkép komponensében található SVG fájl lecserélése egy Bing Maps API, vagy Google Maps API által biztosított térképre. Ez ugyanúgy lehetővé tenné régiók, illetve megyék kiszínezését, vagy egyéb módon történő megkülönböztetésüket, azonban a kereső komponens által listázott cégek is meg tudnának jelenni a térképen a címeik alapján. A felhasználó

így pontosan tudná, helyileg hol helyezkednek el az általa keresett vállalatok székhelyei. Az efféle API-k által biztosított térképekben egyéb lehetőségek is rejtőznek, amiket az SVG fájl limitációi miatt a jelenlegi verzióban nem lehetne megvalósítani.