SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

Modern JavaScript keretrendszerek alkalmazása: Orvosi egyetem TDK honlapjának megvalósítása

Diplomamunka előkészitése

-beszámoló-

Vezetőtanár : dr. Szabó László Zsolt

Készítette : Dali Szilárd István

Tartalom

Időbeosztás

Az első napokon az egyik legfontosabb kérdést probáltam megválaszólni, azt, hogy melyik JavaScript keretrendszert választom a diplomamunka elvégzéséhez, valamint telepítettem a szükséges programokat, ezt követően felváltva tanulmányoztam a bibliográfiát és tanultam a React alapjait.

Bibliográfia tanulmányozása

A témavezető Tanárral való személyes megbeszélést követően a következő linken levő weboldal: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks elolvasása és ez alapján egy fontos döntés meghozása volt az első feladatom, ugyanis el kellet döntenem, hogy milyen keretrendszert szeretnék választani a diplomamunka elvégzésére. Végülis sikerült leszűkítenem a listát két keretrendszerre, ezek a React és a Svelte. A két keretrendszer közül azért esett a választásom a React-re, mert több a rendelkezésre álló információ, így ha elakadok a munka folyamán könyebben találok rá megoldást, továbbá a döntésemet az is befolyásolta, hogy sokkal több a munkalehetőség belföldön ezzel a keretrendszerrel.

A témavezető ajánlására elkezdtem tanulmányozni *Lucas da Costa "Testing JavaScript Applications"* című könyvét, amely a JavaScript alkalmazások tesztelését és annak fontosságát mutatja be.

A könyv elején arról olvashatunk, hogy ki is a célközönség, itt kiderül, hogy a könyv a fejlesztök szemszögéből mutatja be a továbbiakban a tesztelés folyamatait. A továbbiakban egy valós életből vett példa alapján lesz bemutatva az, hogy pontosan mit is szeretne egy megrendelő az alkalmazástól, és, hogy hogyan is kell megvalósítani azt teszt alapú programozással. Az esettanulmány a következő képpen hangzik: Louis nagybácsinak van egy jól menő péksége, amelynek szeretne egy weboldalt készíteni, amely lehetővé teszi a vásárlóknak, hogy online rendeljenek tőle.

Elsőként egy kosárba betesz fügvényt tárgyal a szerző, itt példának azt veszi, hogy a vásárló megpróbál egyszer egy majd egyszerre több macaroont betenni a kosarába, amit elküld az adatbázisnak. A továbbiakban több tesztelésre szoruló részt is tárgyal, amely a kosárba tesz funkcióhoz

kapcsolodik, például adatbázis tesztelése, elfogyott elem rendelésének kérdése. Már itt is látszik, hogy miért fontosak az automatizált tesztek, ugyanis ha minden funkciónalitás teszteléséhez újra meg újra ki kell választani egyenként a macaroonokat időigényesebb, mint megírni egy tesztet, ahol megadjuk az elvárt kimenetet és az alkalmazás fejlesztése során bármikor meghívható.

Ezután az alkalmazás más funkcionalításának teszteléseiről olvashatunk, továbbá azt is olvashatjuk, hogy mennyivel egyszerübb ha függvényeket tesztelünk kezdetben külön-külön, ugyanis ha 10 sor kódot tesztelünk, akkor nem kell az egész alkalmazásban keresnünk a hibát. És a későbbiekben meghívhatjuk a megírt teszteket, ellenőrizve, hogy még mindig jól működik minden. Mindezt a szerző a küldemény követésével szemlélteti, hiszen ha tesztelni szeretnénk előtte be kell tennünk a kosárba a macaroont, meg kell rendelnünk, és mindezt a folyamatot újra meg újra meg kell ismételni ameddig tesztelni szeretnénk, ellentétben a megírt teszttel, amelyet csupán egyszer kell leprogramozni és utána tőbbször is meg lehet hívni.

Más fejlesztőkkel való közös munka esetében is sokkal könyebb egymás munkáját felhasználni a könyv szerint, mert ha már vannak előre megírt tesztek, akkor, ha egyesítésre kerül a sor nem kell a másik féltől folyton megkérdezni, hogy hogyan is kell használni a függvényeit, ugyanis a tesztekből megtudhatjuk a használatát és már tesztelve is vannak.

A következő rész a tesztelési piramisról szól, ahol a teszteket azok gyakorisága szerint csoportokba sorolja. A teszteket három csoportba soroljuk: Unit tesztek, Service tesztek, UI tesztek.

A Unit tesztek a legyakrabbak, ezekről a tesztekről beszéltünk eddig, ide tartoznak a függvények és más egyedülálló komponensek tesztelései.

A Service tesztekbe azok tartoznak, amelyek több komponens tesztelését is megvalósítják, ezekből kevesebbet használunk.

A UI tesztek a legritkábbak, ugyanis itt már a kész terméket teszteljük.

Ezt követően a teszteket izoláltság szerint rendezi egy piramisba, amely hasonló az elöbbihez, az itt levő csoportok: Unit tesztek, Intagration tesztek, end-to-end tesztek. Ahol a Unit teszt a legizoláltabb, az Intagration tesztek több komponens tesztelését teszik lehetővé, végül az end-to-end tesztek egy visszajelzést adnak nekünk az alkalmazásról egy felhasználó szemszögéből.

Minden típusú tesztről részletes leírást kapunk, React példákkal ellátva. Mindezek után http kérések teszteléséről is olvasni lehet, valamint az APIkról és tesztelésükről információt gyűjthetünk.

Miután arról is információt szerzünk, hogy olvashatóan kell tartanunk a kódot, ugyanis általában sokba kerül egy teszt modósítása, jól el kell különíteni a tesztelést a többi kódtól.

Az olvasottak alapján arra a következtetésre jutottam, hogy a teszt alapú fejlesztés igenis kifizetödő, mivel a teszteket bármikor fel lehet használni a fejlesztés során, bár az elején ezek megírása több időt vesz igénybe, de azután meggyorsítja a fejlesztést. Továbbá ha az alkalmazás kisebb részeire is írunk teszteket, akkor nem kell több száz sorban keresni a hibát, hanem csak azokban, amelyre a teszt vonatkozik. Nagy előnynek látom azt is, hogy ha valakitől segítséget kell kérnem, akkor sokkal átláthatobb a kód számára, és érthetővé válik az is, hogy melyik résznek mi az elvárt kimenete, és a hibája.

Gyakorlati foglalkozás

Elsősorban egy új React alkalmazás létrehozását és futtatását kellet megoldani és letölteni a szükséges fájlokat. Ezután a Codevolution youtube csatorna ReactJs Tutorial című videosorozata alapján kezdtem el megtanulni az alapokat.

Elöszőr megtanultam a React alkalmazások mappáinak felépítését, azután azt, hogy mik a komponensek és, hogy mire használjuk őket, majd az állapot nélküli függvények és állapottal rendelkező osztályok közötti különbségeket. Példa a felépítésükre:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

állapot nélküli függvény

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

állapottal rendelkező osztály

A következő lépés a JSX fogalmának megértése volt, ez a JavaScript szintaxisának kibővitése, amely lehetővé teszi, hogy HTML szintaxist használhassunk JavaScript fájlokban, amely megkönnyíti és érthetővé teszi a kódolást. Ezután a paraméterek átadásával foglalkoztam, itt átadtam adatokat egyaránt függvényeknek és osztályoknak is.

Az állapotok és a változó állapotok megjelenítése következett, majd az eseménykezelő használata, különböző adattípusok feldolgozása, stíluslapok használata végül az űrlapok használata. Néhány példakód:

Adatok feldolgozása

```
const persons  =
[
    {
        id: 1,
        name: 'Személy1',
        age: 18
    },
    {
        id: 2,
        name: 'Személy2',
        age: 24
    },


];
const personList  = persons.map( person => <Person key={person.id} person = {person} />)
```

Adatok átadása paraméterként

```
const Greet = props =>{

    return (
        <div>
            <h1> Hello {props.name}, <small> {props.sname} </small></h1>
            {props.children}
        </div>

    )

}
```

Eseménykezelő használata

```
return (
  <div>
      <h1>
      <button onClick={()=>{console.log("button pressed")}}> Click </button>
      </h1>
  </div>
)
```

Változó állatopú adat megjelenítése

```
constructor() {
    super()
    this.state = {
        message: 'Hello here, please subscribe !'
    }
}


render() {
    return (
        <div>
            <h1> { this.state.message    }</h1>
            <button onClick={ () => {this.setState({message: "Thank you for subscribing ! "})}}> Subscribe </button>
        </div>
    );
}
```

Űrlap használata

```
const [submitting, setSubmitting] = useState(false);
const handleSubmit = event => {
  event.preventDefault();
 setSubmitting(true);

 setTimeout(() => {
   setSubmitting(false);
 }, 3000)
}

return(
  <div className="wrapper">
    <h1>Simple Form</h1>
    {submitting &&
     <div>Submtting Form...</div>
   }
    <form onSubmit={handleSubmit}>
      <fieldset>
        <label>
          <p>Name</p>
          <input name="name" />
        </label>
      </fieldset>
      <button type="submit">Submit</button>
    </form>
  </div>
)
```

Végül a *React* hivatalos oldalán, valamint a *Mozilla* oldalán levő példakódok megértésével, valamint azok mükődési leírásának olvasásával és megértésével, és még más youtube csatornák okatatóvideói megtekintésével gyarapítottam tudásomat a nyár folyamán.

Említett oldalak:

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started#hello_react

https://www.youtube.com/watch?v=QFaFIcGhPoM&list=PLC3y8-rFHvwgg3vaYJgHGnModB54rxOk3&index=1

https://reactjs.org/