ZÁRÓDOLGOZAT

Czipa András

2023

Magyar Agrár- és Élettudományi Egyetem

Károly Róbert Campus

Műszaki Intézet

Felsőoktatási szakképzés

Programtervező informatikus

JavaScript vs. TypeScript

Belső konzulens: Dr. Novák Tamás

Egyetemi docens

Belső konzulens intézete/tanszéke: Műszaki Intézet

Készítette: Czipa András

Gyöngyös

2023

1. Témaválasztás

A szakmai gyakorlatom során fejlesztettem a webprogramozási készségeimet, kifejezetten a JavaScript, TypeScript és React technológiákra összpontosítva. Ezek a technológiák napjainkban rendkívül aktuálisak és meghatározók a webfejlesztés terén. A JavaScript és TypeScript nyelvek használata lehetővé teszi a dinamikus webalkalmazások kialakítását, amelyek egyre inkább a digitális tér alapvető részeivé válnak.

A React keretrendszer szintén kiemelkedő fontosságú ebben a kontextusban, mivel a komponensalapú fejlesztési megközelítés egyre inkább terjed, és hatékonyan segít a felhasználói felületek kialakításában. A gyakorlat során elsajátított készségek segítségével már most képes vagyok komplex és skálázható webalkalmazások létrehozására.

A projektek során mélyedtem el az állapotkezelés, routing és komponensépítés területén, amelyek napjainkban kritikus szerepet játszanak a webfejlesztésben. Az optimalizáció és a teljesítmény növelése szintén kulcsfontosságú szempont, hiszen a felhasználói élmény fokozása és a webalkalmazások gyors működése elengedhetetlen a versenyképesség szempontjából.

Ezen kívül a modern fejlesztőeszközök, például a Git és a GitHub használata nélkülözhetetlen a hatékony és együttműködésre képes fejlesztőként. Ezek az eszközök nemcsak az aktuális, hanem a jövőbeli webfejlesztési projektek során is kulcsfontosságúak.

A webprogramozási gyakorlatom során elért fejlődésem és a megszerzett tapasztalatom kiváló alapot nyújtanak a JavaScript és TypeScript közötti különbségek elemzéséhez a záródolgozatomban. Az aktuális szakmai ismeretek és gyakorlati tapasztalatok birtokában kész vagyok egy alapos és releváns elemzést készíteni a két nyelv közötti különbségekről, amelyek jelentősége és alkalmazása napjainkban kritikus fontosságú a webfejlesztés világában.

2. Szakirodalmi feldolgozás

2.1 JavaScript vs. TypeScript

A JavaScript és a TypeScript két olyan programozási nyelv, amelyeket webfejlesztéshez és alkalmazások készítéséhez használnak. Míg mindkettő a webes technológiák alapját képezi, fontos megérteni a köztük lévő különbségeket, mivel ezek hatással lehetnek a fejlesztési folyamatra és az alkalmazás minőségére.

2.1.1 Típusos vs. Dinamikus Nyelv

A legjelentősebb különbség a típusosságban rejlik. A JavaScript dinamikus típusú nyelv, ami azt jelenti, hogy a változók típusa futás közben kerül értékelésre. Ezzel szemben a TypeScript szigorú típusosságot használ, amely lehetővé teszi a változók típusának előre definiálását, és a fordítási időben történő típusellenőrzést.

A JavaScript egy dinamikus típusú nyelv, ami azt jelenti, hogy a változók típusa futás közben kerül értékelésre. Ez azt eredményezi, hogy a program futása során a változók típusa rugalmasan változhat, ami előnyökkel és hátrányokkal is járhat. Például egy változó kezdetben szám(int) típusú lehet, majd később egy karakterlánccá (list) vagy más típussá változhat anélkül, hogy hibaüzenetet jelezne a program.

Ezzel szemben a TypeScript egy szigorú típusosságot alkalmazó nyelv, amely lehetővé teszi a változók típusának előre definiálását, és a fordítási időben történő típusellenőrzést. Ez azt jelenti, hogy a fejlesztők előzetesen meghatározhatják, hogy egy változó milyen típusú lehet, és a TypeScript azonnal jelezni fogja, ha egy típushiba lép fel. Így sok hibát már a fejlesztési fázisban kiszűr ezzel hozzájárulva a kód stabilitásához és megbízhatóságához.

Ezt alátámasztja a "TypeScript Deep Dive" című könyv, melyet Basarat Ali Syed írt. A könyvben a szerző kiemeli a TypeScript szigorú típusosságának fontosságát és a dinamikus típusosság hátrányait. Az alábbi szövegrészlet bemutatja ezt a különbséget:

"TypeScript requires you to declare your types at design time. This means that if you make a mistake, you'll be told about it during design time and not when your application is running." (Basarat,2017)

A szerző itt kiemeli a TypeScript előnyeit a típusellenőrzés terén és azokat a problémákat, amelyekkel a dinamikus típusú nyelvek, mint a JavaScript, szembesülhetnek futás közben. A "TypeScript Deep Dive" című könyv teljeskörűen elemzi a TypeScript használatát és előnyeit.

2.1.2 Statikus Analízis

A TypeScript előnye, hogy statikus analízist végez a kódban, mielőtt futtatná azt. Ez segít kiszűrni a potenciális hibákat és hibás kódokat már a fejlesztés során, javítva a hibakeresést és a kódminőséget.
A statikus analízis és annak fontossága a TypeScript használatában a kódminőség és a hibakeresés szempontjából kiemelkedő téma. Az alábbiakban bemutatom ezt a különbséget.

A TypeScript statikus analízist végez a kódban, mielőtt futtatná azt. Ez azt jelenti, hogy a TypeScript a kód írásakor ellenőrzi a típusokat és a lehetséges hibákat, és jelzi azokat a fejlesztőnek már a fordítási időben. Ez a gyakorlatban azt eredményezi, hogy a potenciális hibák és hibás kódok kiszűrhetők még mielőtt az alkalmazás valós időben futna.

Ez a hozzáállás jelentősen javítja a hibakeresés hatékonyságát és a kódminőséget, mivel a fejlesztők korán értesülnek a hibákról és tudnak azokra reagálni. Emellett hozzájárul az alkalmazás stabilitásához és megbízhatóságához is.

Ezt alátámasztja a "TypeScript Handbook" című online útmutató is, amelyet a TypeScript hivatalos dokumentációjában találunk. A kézikönyv a TypeScript statikus típusellenőrzését és az előnyeit hangsúlyozza:

"TypeScript adds static types to the language to help you avoid mistakes, and make you a more productive developer."

Ez rámutat arra, hogy a TypeScript statikus típusellenőrzése segít a fejlesztőknek elkerülni hibákat és produktívabbá válni. A "TypeScript Handbook" a TypeScript hivatalos dokumentációjának része.

2.1.3 Erősebb Típusellenőrzés

A TypeScript erősebb típusellenőrzést biztosít, ami segít a hibák és hibás működések kiszűrésében. Ez növeli az alkalmazások stabilitását és megbízhatóságát.
Az erősebb típusellenőrzés és annak fontossága a TypeScript használatában az alkalmazások stabilitásához és megbízhatóságához jelentős téma. Az alábbiakban bemutatom ezt a különbséget.

A TypeScript erősebb típusellenőrzése azt jelenti, hogy a fejlesztőknek pontosan meg kell határozniuk a változók típusát, és a TypeScript ellenőrzi, hogy a típusok helyesen vannak-e használva a kódban. Ez segít a hibák és hibás működések kiszűrésében a fejlesztési folyamat során, még mielőtt az alkalmazás valós időben futna.

Az erősebb típusellenőrzés jelentősen növeli az alkalmazások stabilitását és megbízhatóságát, mivel a fejlesztők könnyebben elkerülhetik a típushibákat és az olyan problémákat, amelyek futás közben vezethetnek hibákhoz.

Egy példa az alátámasztásra a "TypeScript Handbook" című online útmutatóból, amelyet a TypeScript hivatalos dokumentációjában találunk. A kézikönyv hangsúlyozza a TypeScript erősebb típusellenőrzésének fontosságát:

"TypeScript helps you avoid many errors regarding types. For example, if you write a function expecting a string, you can be sure it won't be called with a number by mistake."

Ez rámutat arra, hogy a TypeScript segít elkerülni hibákat a típusokkal kapcsolatban, és ezzel hozzájárul az alkalmazások stabilitásához.

2.1.4 Nyelvi Kibővítések

A TypeScript számos olyan nyelvi kibővítést is tartalmaz, mint például az osztályok és interfészek, amelyek a JavaScriptben nincsenek jelen.
A nyelvi kibővítések és a TypeScriptben található továbbfejlesztések a JavaScripthez képest fontos téma a fejlesztők számára.

A TypeScript számos nyelvi kibővítést tartalmaz, amelyek továbbfejlesztik a JavaScript nyelvet. Például, a TypeScript támogatja az osztályok és interfészek használatát, amelyek a JavaScriptben nincsenek jelen. Ez lehetővé teszi a fejlesztők számára, hogy a kódjukat

strukturáltabbá és olvashatóbbá tegyék, és az osztályok segítségével objektumorientált elveket alkalmazhassanak.

Az alátámasztásra bemutatok egy konkrét szakmai irodalmi hivatkozást, amely a "TypeScript for JavaScript Programmers" című könyvből származik, amit Steve Fenton írt:

"TypeScript adds a range of features that enable you to write cleaner and more maintainable code. One of the biggest features is the introduction of classes." (Fenton, 2013)

Ez az idézet kiemeli a TypeScript nyelvi kibővítéseinek fontosságát, különösen az osztályok bevezetését. A "TypeScript for JavaScript Programmers" című könyv részletesen elemzi a TypeScript kibővítéseit és használatuk előnyeit a JavaScript fejlesztők számára.

2.1.5 Kódskálázhatóság

A kódskálázhatóság és annak fontossága a TypeScript alkalmazásában, különösen nagy kódbázisok esetén, kulcsfontosságú téma a fejlesztők számára. A TypeScript különösen előnyös nagy projektjeinkben, mivel segít a kód karbantarthatóságában és a csapatmunkában.

Az "Effective TypeScript: 62 Specific Ways to Improve Your TypeScript" című könyv, melyet Dan Vanderkam írt, alaposan elemzi a TypeScript alkalmazását nagy projektekben. Ebben a könyvben bemutatja, hogy a TypeScript miként segít a kód karbantarthatóságának javításában, a stabilitás növelésében és a fejlesztők általános hatékonyságának fokozásában.

Ezzel a hivatkozással hangsúlyozom, hogy a TypeScript valós előnyöket kínál a nagy projektek esetén a kód minőségének és karbantarthatóságának javításában.

2.1.6 Közösség és Eszközök

Mindkét nyelvnek aktív fejlesztői közössége van, de a TypeScriptnek különféle fejlesztői eszközök és keretrendszerek is rendelkezésre állnak.
A fejlesztői közösség és az elérhető fejlesztői eszközök és keretrendszerek fontos téma a JavaScript és a TypeScript közötti összehasonlításban. Az alábbiakban bemutatom ezt a különbséget.

Mind a JavaScript, mind a TypeScript nagy fejlesztői közösséggel rendelkezik, amelyek folyamatosan hoznak létre és támogatnak új fejlesztői eszközöket és keretrendszereket. Azonban a TypeScriptnek számos olyan eszköze van, amelyek segítik a fejlesztőket a kód írásában, ellenőrzésében és karbantartásában. (React, Angular, Vue)

Az alátámasztásra bemutatok egy konkrét szakmai irodalmi hivatkozást, amely a " Effective TypeScript: 62 Specific Ways to Improve Your TypeScript " című könyvből származik, amit Dan Vanderkam írt:

"TypeScript's robust ecosystem means there are tools to help you write idiomatic TypeScript code. Even better, these tools evolve quickly to keep pace with language developments. This helps you stay productive." (Vanderkam, 2019)

Ez az idézet kiemeli a TypeScript gazdag ökoszisztémáját és az elérhető fejlesztői eszközök fontosságát. A "Effective TypeScript" című könyv részletesen elemzi a TypeScript használatát és az eszközök használatát a hatékony fejlesztés érdekében.

2.2 Kódminőség:

2.2.3 TypeScript előnyei:

Erősebb típusellenőrzés: A TypeScript segít kiszűrni a típushibákat már a fejlesztés során. Ez javítja a kódminőséget, mivel a típushibák nehezen észrevehető hibák lehetnek, amelyek futás közben okozhatnak problémákat.
Olvashatóság: A TypeScript segíti a kód struktúráltabbá tételét, mivel támogatja az osztályokat, interfészeket és modulokat. Ez lehetővé teszi az olvashatóbb és karbantarthatóbb kód írását.

2.3.4. JavaScript kihívásai:

Gyenge típusellenőrzés: A JavaScript dinamikus típusú nyelv, ami lehetőséget ad hibás típusok használatára.
Kevesebb strukturális támogatás: A JavaScript kevésbé strukturált, és nem támogatja az osztályokat és interfészeket, ami nehezítheti a kód karbantartását.

2.3 Hibakeresés:

2.3.1 TypeScript előnyei

Statikus analízis: A TypeScript előre értékeli a kódot, és kiszűri a potenciális hibákat. Ez lehetővé teszi a hibák korai felfedezését és javítását, ami hozzájárul a hibakeresés hatékonyságához.

2.3.2 JavaScript kihívásai

Dinamikus típusok: A JavaScript dinamikus típusokat használ, ami lehetővé teszi hibás típusok használatát. Ezek a hibák csak futás közben jönnek elő, ami nehezítheti a hibakeresést.

2.4 Teljesítmény

2.4.1 TypeScript előnyei

Optimalizált kód: A TypeScript fordítás során gyakran optimalizálja a kódot, ami javíthatja a teljesítményt.

2.4.2 JavaScript kihívásai

Dinamikus típusok: A JavaScript dinamikus típusokat használ, amelyek néha lassabb futást eredményezhetnek.

2.5 Szakirodalom

A konkrét példák és részletek érdekében a "TypeScript vs. JavaScript"(medium.com 2021.06.21) című összehasonlító cikk a Medium-on jó példa lehet. Ebben a cikkben a szerző részletesen elemzi a kódminőséget, hibakeresést és teljesítményt mindkét nyelv szempontjából.

2.6 Konklúzió

Azáltal, hogy átgondolod ezeket a szempontokat, a projekted igényeit és a fejlesztői csapat képességeit, könnyebben dönthetsz arról, melyik nyelvet érdemes használni egy adott projektben.

3. Szakmai gyakorlat

A beszámolóm bevezetése a Docler Holding és annak egyik jelentős platformja, a LiveJasmin bemutatására szolgál. Egy luxemburgi székhelyű vállalat, amely a digitális tartalomgyártás és online szórakoztatás terén tevékenykedik. A LiveJasmin pedig az egyik legismertebb online élő kamerás felnőtt szórakoztató platform, amely interaktív élményt nyújt a felhasználóknak.

3.1 Docler Holding

Széleskörű iparági tevékenységei és innovatív megközelítése a digitális szórakoztatás területén kiemelkedővé teszik. A vállalatot Gattyán György alapította, aki a vállalat elnökeként is tevékenykedik. Az alábbiakban bemutatom néhány területet, ahol a Docler Holding jelentős szerepet játszik:

3.1.1 Médiaipar

A Docler Holding olyan online média platformokat működtet, amelyek szórakoztató és informatív tartalmakat kínálnak a felhasználóknak. Ezek a platformok számos nézőt vonzanak és a legújabb technológiákra épülnek.

3.1.2 Technológia

A vállalat komoly hangsúlyt fektet az innovációra és a technológia fejlesztésére. A Docler Holding számos technológiai projektet és megoldást hajt végre, amelyek elősegítik az online szórakoztatást és a digitális tartalomgyártást.

3.1.3 Ingatlan

A vállalat ingatlanberuházásokkal is foglalkozik. A Docler Holding épületeket és ingatlanokat birtokol és kezel, amelyekben székhelyeik és egyéb üzleti tevékenységeik zajlanak.

3.2 LiveJasmin

A LiveJasmin, amely a Docler Holding platformja, az online élő kamerás felnőtt szórakoztatás egyik vezető neve. Az alábbiakban bemutatom a LiveJasmin főbb jellemzőit és szolgáltatásait.

3.2.1. Élő kamerák

Lehetővé teszi a felhasználók számára, hogy valós idejű élő videóközvetítéseken keresztül kapcsolatba lépjenek modellekkel és szereplőkkel. A platformon számos kategória közül választhatnak, beleértve nőket, férfiakat, párokat és transznemű személyeket.

3.2.2 Interaktív élmény

A felhasználók a LiveJasmin platformon interaktív módon léphetnek kapcsolatba a modellekkel. A chat ablakon keresztül üzeneteket küldhetnek, kifejezhetik kívánságaikat és kommunikálhatnak a választott modellel. Ez lehetővé teszi a személyreszabott és egyedi élmények megélését.

3.2.3 Privát előadások

A LiveJasmin lehetőséget nyújt privát előadások igénylésére. Ez azt jelenti, hogy a felhasználók kizárólagosan foglalkozhatnak egy modellel, és a tartalom csak nekik lesz elérhető. Ez a lehetőség intimebb élményt biztosít, és lehetővé teszi a felhasználók számára, hogy a modellel közvetlenebb kapcsolatot alakítsanak ki.

3.2.4 Fizetési lehetőségek

A LiveJasmin különböző fizetési lehetőségeket kínál a felhasználóknak. A kreditek vásárlása révén a felhasználók fizethetnek az előadásokért, ajándékokért vagy a modellel való interakcióért. A platform biztosítja a biztonságos és diszkrét fizetési folyamatokat, hogy a felhasználók nyugodt környezetben élvezhessék az élményt.

3.2.5 Magas minőségű szolgáltatások

A LiveJasmin magas minőségű videó- és audiofunkciókkal rendelkezik, hogy a felhasználók élvezhessék az optimális élményt. A platform folyamatosan fejleszti technológiai infrastruktúráját, hogy biztosítsa a stabil és zavartalan közvetítéseket.

3.2.6 Fontos megjegyezni

A LiveJasmin egy felnőtt tartalmakat kínáló weboldal, és csak azok számára ajánlott, akik elérték a helyi jogszabályok által előírt jogi korhatárt. A platform a felhasználók privát és bizalmas környezetét biztosítja, és szigorú szabályokat alkalmaz a tartalom és a felhasználói viselkedés ellenőrzésére.
A Docler Holding és a LiveJasmin egyaránt fontos szereplők az online szórakoztatás terén, és kiemelkednek a digitális tartalomgyártás területén.

3.3 Szakami gyakorlat menete

3.3.1 Fejlesztési terület

A szakmai gyakorlati munkám során Frontend developer intern beosztásban dolgoztam a LiveJasmin oldalát monitorozó admin oldalon, amelyből konkrét részleteket vagy megoldásokat nem oszthatok meg a titoktartási nyilatkozat értelmében.

3.3.2 Csapat

A szakmai gyakorlatom során egy mintegy 8 fős csapatban dolgoztam, ahol több kolléga is segítette a folyamatok megértését és fejlődésemet. A csapatmunkának rengeteg pozitív hozadéka volt, köztük az is, hogy napról napra egyre többet fejlődtem és megismerkedhettem különböző feladatokkal, folyamatokkal.

3.3.3 Scrum

A csapatunk a Scrum keretrendszer szerint működött, amely egy rugalmas és hatékony projektmenedzsment módszer. Két hetes sprintekben dolgoztunk, amelyek az időt kis részletekre osztották, hogy könnyebben nyomon követhessük a feladatokat és elérjük a kitűzött célokat. A sprintek kezdetén és végén retrospektív értekezleteket tartottunk, ahol értékeltük a munkát és tanulságokat vontunk le az előző időszakról.
A napi munkafolyamatunk részeként minden reggel tartottunk egy rövid "daily meeting"-et, ahol a csapat tagjai összegzik, hogy mit csináltak az előző napon, milyen feladatok várnak rájuk a jelenlegi napon, és ha szükséges, akkor megosztják a problémákat vagy kihívásokat, amelyekkel szembesülnek. Ez a találkozó rendkívül hasznos volt a kommunikáció és az

összehangolás szempontjából, és segített abban, hogy mindenki naprakész legyen a projekt állapotáról.

3.3.4 Jira

A feladatok nyomon követésére és a csapatmunka hatékonyabbá tételére használtunk ticketing rendszert, amelynek neve Jira volt. Ez a rendszer lehetővé tette számunkra, hogy könnyen létrehozzunk és rendszerezzünk feladatokat, és hogy nyomon kövessük azok állapotát és fejlesztését. A Jira rendszer használata rendkívül hasznos volt a feladatok kezelésében és a projekt átláthatóságának biztosításában.

3.3.5 Git

A verziókezeléshez és a kódmegosztáshoz a Git-et használtuk. Ez a nyílt forráskódú verziókezelő rendszer lehetővé tette számunkra, hogy párhuzamosan dolgozzunk a projekten, és könnyen nyomon kövessük és kezeljük a kódváltozásokat. A Git használata segített abban, hogy hatékonyan együttműködjünk a csapattagokkal és biztonságosan kezeljük a kódváltozásokat.

3.3.6 Összegzés

Összességében a szakmai gyakorlatom során értékes tapasztalatokat szereztem a csapatmunkáról és a projektmenedzsmentről. A Scrum keretrendszer, a napi értekezletek, a Jira és a Git mind hozzájárultak a hatékony és eredményes munkavégzéshez. Ez a tapasztalat nagyban megerősítette a kollégáim iránti tiszteletet és hálával töltött el a támogatásukért és a közös erőfeszítésekért.

3.4 Szakmai gyakorlati készségek

A szakmai gyakorlatom során lehetőségem volt elmélyülni a JavaScript és a TypeScript nyelvben. Ezeket a nyelveket használtam a webes alkalmazások fejlesztésére, ahol a JavaScript a hátteret, míg a TypeScript a statikus típusellenőrzést biztosította. A TypeScript segítségével hatékonyabban tudtam dolgozni, mivel a kódban található hibák és a típusosztályozás segítségével könnyebben felismerhetők voltak.

Ezenkívül megismerkedtem a React keretrendszerrel, amelyet a felhasználói felületek készítésére használnak. A React egy hatékony és moduláris megoldást nyújt a felhasználói felület fejlesztésére, és lehetővé teszi a komponensek újrafelhasználását. Az alkalmazások különböző részeinek elkülönítése és moduláris felépítése révén egyszerűbben tartható a kód rendszerezése és karbantartása.

Emellett megtanultam használni a Reduxot, amely egy állapotkezelő könyvtár. A Redux segítségével könnyedén kezelhető az alkalmazás állapota és a komponensek közötti adatáramlás. Az egyetlen globális állapotú adattároló révén egyszerűbben tartható a kód összehangolása és a hibák felderítése.

Végül a Hookokat is elsajátítottam, amelyek a React funkcionális komponenseinek kiegészítői. A Hookok segítségével állapotokat és mellékhatásokat kezelhetünk a React komponensekben, anélkül, hogy osztálykomponensekre lenne szükségünk. Ez egyszerűbb és olvashatóbb kódot eredményez, valamint segít az alkalmazás hatékonyabb és skálázhatóbb felépítésében.

Összességében a szakmai gyakorlatom során sokat tanultam a JavaScript, TypeScript, React, Redux és a Hookok terén. Ezeket a technológiákat sikeresen alkalmaztam gyakorlati feladatokban, és megtapasztaltam, hogy milyen hatékonyan lehet velük webes alkalmazásokat fejleszteni.

3.4.1 JavaScript

A JavaScript programozási nyelvet alaposan megismertem és használtam interaktív webes alkalmazások fejlesztéséhez. Az alábbi egyszerű példa mutatja az 1.ábrán, hogy hogyan alkalmaztam a JavaScriptet egy funkció implementálására:

```javascript
// Egyszerű összeadás függvény
function addNumbers(a, b) {
  return a + b;
}
```

1. ábra- saját szerkesztés

3.4.2 TypeScript

A TypeScriptet a JavaScript kibővítésére és típusbiztosításra használtam. Ez segített a kód stabilitásának növelésében és a hibák előfordulásának csökkentésében. Az alábbi példa bemutatja a 2. ábrán egy egyszerű TypeScript osztály definícióját:

```typescript
// Egyszerű osztály definíció TypeScriptben
class Car {
  private brand: string;
  private model: string;

  constructor(brand: string, model: string) {
    this.brand = brand;
    this.model = model;
  }

  getCarDetails(): string {
    return `Brand: ${this.brand}, Model: ${this.model}`;
  }
}
```

2. ábra- saját szerkesztés

3.4.3 React

A React keretrendszert használtam felhasználóbarát felhasználói felületek fejlesztéséhez. Az alábbi példa a 3.ábrán egy egyszerű React komponensre mutat példát:

```jsx
// Egyszerű React komponens
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

3. ábra- saját szerkesztés

3.4.4 Redux

A Reduxot alkalmaztam a React alapú alkalmazások állapotkezeléséhez és adatfolyamának kezeléséhez. Az alábbi példa bemutatja a 4. ábrán egy egyszerű Redux akció és reducer példányát:

```javascript
// Egyszerű Redux akció típus
const INCREMENT_COUNTER = 'INCREMENT_COUNTER';

// Egyszerű Redux reducer
function counterReducer(state = 0, action) {
  switch (action.type) {
    case INCREMENT_COUNTER:
      return state + 1;
    default:
      return state;
  }
}
```

4. ábra- saját szerkesztés

3.4.5 Hookok

A Hookokat alkalmaztam a React komponensekben az állapot és az életciklus események kezelésére. Az alábbi példa az 5. ábrán bemutatja a useState Hook használatát egy egyszerű számláló komponensben:

```jsx
// Egyszerű számláló komponens useState Hookkal
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment
```

5. ábra- saját szerkesztés

3.4.6 CSS és Chakra UI

A szakmai gyakorlatom során jobban eltudtam mélyülni a CSS (Cascading Style Sheets) nyelvben, amelyet a webes felületek megjelenésének és stílusának kialakítására használnak. A CSS segítségével szabályokat definiálhatunk, amelyek meghatározzák a HTML elemek megjelenését, például a szöveg formázását, a háttérszínt, a betűtípust és méretet, a dobozmodellt stb. Ismerem és alkalmaztam a CSS szelektorokat is, amelyek segítségével hatékonyan lehet a stílusokat kiválasztani és a kívánt elemeket szerkeszteni a weboldalon.

A gyakorlati feladatok során a CSS-t használtam az alkalmazások felhasználói felületeinek stílusának kialakításában. A dobozmodell segítségével pontosan pozícionáltam és elrendeztem az elemeket a weboldalon. Emellett alkalmaztam az elosztási technikákat, például a Flexbox-ot és a Grid-et, hogy rugalmasan és reszponzívan alakítsam ki az alkalmazások elrendezését különböző képernyőméretekhez.

Azonkívül megismertem és használtam a Chakra UI-t is, amely egy könnyen használható UI (felhasználói felület) komponenskönyvtár a React keretrendszerhez. Ez a könyvtár előre elkészített komponenseket és stílusozási lehetőségeket kínál, amelyek egyszerűsítik a felhasználói felületek kialakítását és fejlesztését. A Chakra UI kifejezetten testreszabható, lehetővé téve a komponensek stílusának és megjelenésének könnyű személyre szabását.

A gyakorlat során a Chakra UI-t használtam a felhasználói felületek fejlesztéséhez. A következő példa a 6. ábrán bemutatja, hogyan hozhatunk létre egy gombot a Chakra UI segítségével:

```jsx
import { Button } from "@chakra-ui/react";

function App() {
  return (
    <Button colorScheme="blue" size="md">
      Kattints ide
    </Button>
  );
}
```

6. ábra- saját szerkesztés

Ez a kód egy Chakra UI gombot hoz létre, amelynek kék színe van és közepes méretű. A Button komponens a Chakra UI-ban előre definiált stílussal rendelkezik, így nem szükséges külön CSS-t írnunk a gombhoz. Egyszerűen importáljuk a Button komponenst a Chakra UI csomagból, és használjuk a kívánt tulajdonságokkal.

3.5 Összefoglalás

A szakmai gyakorlatom során három hónapon keresztül heti 2-3 alkalommal folytattam a front-end fejlesztés tanulmányozását. Ez az időszak számos hasznos ismeretet és tapasztalatot hozott számomra. Az általam megszerzett készségek és tudás egy erős alapot jelentenek a jövőbeli karrierem szempontjából.

Az elmúlt hónapokban aktívan foglalkoztam a JavaScript, TypeScript, React, Redux, Hookok, CSS és Chakra UI területeivel. A JavaScript és TypeScript segítségével hatékonyan tudtam dolgozni és a kódban található hibákat könnyebben felismerni. A React keretrendszer és a Redux állapotkezelő könyvtár használatával egyszerűbben tudtam fejleszteni a felhasználói felületeket és hatékonyabban kezelni az adatáramlást. A CSS és a Chakra UI pedig lehetővé tették, hogy vonzó és reszponzív felhasználói felületeket hozzak létre a projektekben.

A gyakorlati feladatok során valós projekteken dolgoztam, ahol gyakorolhattam az alkalmazott technológiák alkalmazását. Ezáltal megtanultam rugalmasan alkalmazkodni a felmerülő kihívásokhoz és hatékonyan együttműködni a csapattagokkal. Az eredmények, amiket elértem, inspirálóak voltak, és megerősítették bennem a hivatás iránti elkötelezettséget.

A front-end fejlesztés világa számomra lenyűgözővé vált, és nagyon izgatott vagyok, hogy folytathatom ezt a pályát a jövőben. A szakmai gyakorlatom során megszerzett készségek, tapasztalatok és tudás útmutatásul szolgálnak számomra a további tanulásban és fejlődésben. Bizakodó vagyok, hogy sikeresen helyt fogok állni a valós projektekben, és folytatni fogom a szakmai fejlődésemet.

Büszke vagyok arra, amit elértünk a szakmai gyakorlatom során, és hálás vagyok a lehetőségért, hogy részt vehettem ebben a fejlődésben. Megerősített azzal a tudattal, hogy a front-end fejlesztés terén szeretném építeni a jövőbeli karrieremet. Bízom benne, hogy a megszerzett ismereteket és a lelkesedést sikeresen kamatoztathatom majd a valós projektekben, és hosszú távon szakemberként tevékenykedhetek ezen a területen.

4. JavaScript vagy TypeScript – Vélemény

Szakmai gyakorlatom során szerzett tapasztalatok és a végzett kutatásom eredményeként egyértelművé vált számomra, hogy a modern webfejlesztés terén a TypeScript már nélkülözhetetlen eszközzé vált. A TypeScript használata számos előnnyel jár, amelyek között kiemelkedik a statikus típusellenőrzés, a jobb kódminőség, és a könnyebb karbantarthatóság. Az erőteljes eszközkészlettel rendelkező TypeScript segítségével fejlesztők hatékonyabban tudnak dolgozni, és a kódbázisok hosszú távú fenntarthatóságát is növelik.

Azonban fontos megérteni, hogy a TypeScript önmagában nem lenne olyan eredményes, ha nem alapozódna a JavaScriptre. A JavaScript a webfejlesztés alapja, és bár rendelkezik néhány korlátoltan érthető tulajdonsággal, a TypeScript ezen korlátokat képes áthidalni. A TypeScript és a JavaScript közötti szoros kapcsolat lehetővé teszi a fejlesztők számára, hogy a legjobbat hozzák ki mindkét világból.

Ezért azt a véleményt vallom, hogy a TypeScript nélkülözhetetlen a modern webfejlesztésben, de ugyanakkor nem szabad elfelejtenünk, hogy a TypeScript a JavaScript nélkül nem létezhetne. A két technológia egymást kiegészítve teszi lehetővé, hogy hatékony és minőségi webalkalmazásokat hozzunk létre, és a jövőben is elengedhetetlenek maradnak a fejlesztői közösség számára. Az itt szerzett ismeretek és tapasztalatok alapján bizakodva tekintek a jövőbe, ahol a TypeScript és a JavaScript továbbra is kulcsfontosságú szerepet játszanak majd a webfejlesztés terén.

5.Irodalomjegyzék

5.1 Irodalmi hivatkozás

Basarat Ali Syed – TypeScript Deep Dive – 2017.

Steve Fenton – TypeScript for JavaScript Programmers – 2013.

Dan Vanderkam – Effective TypeScript: 62 Specific Ways to Improve Your TypeScript – 2019

5.2 Internetes Irodalom

https://www.typescriptlang.org/docs/handbook/2/basic-types.html#types-for-tooling

https://www.typescriptlang.org/docs/handbook/2/basic-types.html#static-type-checking

https://medium.com/geekculture/typescript-vs-javascript-e5af7ab5a331

6. Ábrajegyzék

NYILATKOZAT

a záródolgozat nyilvános hozzáféréséről és eredetiségéről

A hallgató neve:	Czipa András
A Hallgató Neptun kódja:	B9UBKB
A dolgozat címe:	JavaScript vs. TypeScript
A megjelenés éve:	2023
A konzulens intézetének neve:	Műszaki Intézet, Károly Róbert Campus
A konzulens tanszékének a neve:	Alkalmazott Informatika Tanszék, Gyöngyös

Kijelentem, hogy az általam benyújtott záródolgozat egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, és az irodalomjegyzékben szerepeltettem.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem könyvtári repozitori rendszerébe. Tudomásul veszem, hogy a megvédett és

- nem titkosított dolgozat a védést követően
- titkosításra engedélyezett dolgozat a benyújtásától számított 5 év eltelte után

nyilvánosan elérhető és kereshető lesz az Egyetem könyvári repozitori rendszerében.

Kelt: 2023 év 11 hó 12 nap

Hallgató aláírása

NYILATKOZAT

Czipa András (név) (hallgató Neptun azonosítója: B9UBKB) konzulenseként nyilatkozom arról, hogy a záródolgozatot áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A záródolgozatot védésre **javaslom / nem javaslom**[1].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[2]

Kelt: 2023. év 11. hó 11. nap

belső konzulens

[1] A megfelelő aláhúzandó.
[2] A megfelelő aláhúzandó.