Kisvárdai SZC Fehérgyarmati Petőfi Sándor Technikum
4900 Fehérgyarmat, Május 14. tér 16.
203043/003
Szoftverfejlesztő és -tesztelő technikus képzés
5 0613 12 03

VIZSGAREMEK

ChoosePhone projekt

Konzulens tanár:
Lakatos Sándor
Berki Balázs

Készítette:
Simon István
Magyar Ákos

Fehérgyarmat, 2025

KISVÁRDAI
VIZSGAKÖZPONT

4600 Kisvárda, Mártírok útja 8.
45/556-391
kisvardai.vizsgakozpont@gmail.com
www.kisvardaivizsgakozpont.hu

Vizsgaportfólió/vizsgaremek szabályzat 2.sz.melléklete

Nyilatkozat

Alulírott **Simon István** (születési hely: Fehérgyarmat, születési idő: 2005.11.03) jelen nyilatkozat aláírásával kijelentem, hogy az 5 0613 12 03 **Szoftverfejlesztő és -tesztelő** szakmai vizsgára készített portfólió/vizsgaremek/vizsgaremek dokumentáció saját munkám eredménye. Nyilatkozom, hogy az egyes dokumentumokban foglalt tények és adatok a megadott forrásmunkák felhasználásán alapulnak.

Tudomásul veszem, hogy amennyiben a portfóliom/vizsgaremekemben a vizsgabizottság tagjai olyan dokumentumot találnak, amely problémaorientált feladat esetén 20%-nál nagyobb mértékben nem önálló munkám eredménye, vagy leíró/összegző feladat esetén nem saját kutatásom eredménye és amelyből a források kiválasztásnak gondolatmenete vagy relevanciája nem követhető, valamint nem tartalmazza a pontos forrásmegjelöléseket, abban az esetben a vizsgabizottság tagjai a portfóliót/vizsgaremeket nem fogadják el. Ezen pont alól kivételt képeznek a csoportos projektek, komplex feladatok és esettanulmányok, amelyek keletkezési körülményeit a bevezető tartalmazza.

Jelen nyilatkozat aláírásával tudomásul veszem, hogy amennyiben bizonyítható, hogy a portfóliót/vizsgaremeket nem magam készítettem, ez a vizsgarész elégtelen értékelést kap.

Hozzájárulok ahhoz, hogy vizsgabizottság tagjai, jegyzője, és a *Kisvárdai Vizsgaközpont* munkatársai a portfólió/vizsgaremek tartalmát megismerhessék.

Kijelentem, hogy a portfólióra/vizsgaremekre kapott értékelésemet és a portfóliómat/vizsgaremekemet további felhasználásra sem kinyomtatva, sem pedig elektronikus úton nem továbbítom senkinek.

Kelt: Fehérgyarmat, 2025. 03. 05.

Simon István

.....

aláírás

KISVÁRDAI SZC FEHÉRGYARMATI PETŐFI SÁNDOR TECHNIKUM
4900 Fehérgyarmat, Május 14. tér 16.
203043/003
Szoftverfejlesztő és - tesztelő technikus képzés

Eredetiségi nyilatkozat projektmunkáról

Alulírott SIMON ISTVÁN,72601232131 (saját név, oktatási azonosító) nyilatkozom, hogy a csatoltan bírálatra és védésre beadott projektmunka teljes egészében a saját és csapattársam – MAGYAR ÁKOS,72560885247 (név, oktatási azonosító) - munkája.

A felhasznált forrásokat az irodalomjegyzékben feltüntettem, a rájuk vonatkozó szabályszerű hivatkozásokat a szövegben megtettem. A projektmunka más intézményben sem a saját nevemben, sem más nevében nem került beadásra.

Tudatában vagyok annak, hogy plágium (más munkájának sajátomként történő feltüntetése) esetén a projektmunka érvénytelen, ezért elutasításra kerül.

Fehérgyarmat, 2025. április 14.

Simon István

aláírás

Tartalomjegyzék

Bevezetés

A projekt fő célja

A ChoosePhone projekt célja, hogy jelentős segítséget nyújtson a felhasználóknak a mobiltelefonok széles választékában való felfedezésében. A modern technológia világában, ahol naponta újabb és újabb okostelefonok jelennek meg, elengedhetetlen, hogy a vásárlók megalapozott döntéseket tudjanak hozni. A ChoosePhone egy felhasználóbarát platform, amely a részletes specifikációkon alapuló összehasonlítások révén könnyen navigálható megoldást kínál.

A projekt megalakulása

A projekt indításának hátterében az állt, hogy a mobiltelefonok vásárlása gyakran bonyolult és időigényes feladat. A felhasználók számos tényezőt, például a processzor teljesítményét, a memória kapacitását, a kamerák minőségét és az akkumulátor élettartamát képesek figyelembe venni, de a sok információ között nehéz a legjobb döntést hozni. A ChoosePhone célja, hogy ezen a problémán segítve, egy átfogó és intuitív platformot biztosítson, ahol a felhasználók könnyedén összehasonlíthatják a különböző modellek jellemzőit.

Célfelhasználók

A ChoosePhone projekt széles körű felhasználói bázist céloz meg, amely különböző igényekkel rendelkezik:

1. **Átlagos vásárlók**: Akik új telefont keresnek, de nem feltétlenül értenek mélyrehatóan a technológiai specifikációkhoz. Számukra egyszerű, intuitív szűrési lehetőségeket biztosítunk.
2. **Tech-rajongók és szakértők**: Akik részletes műszaki adatokat szeretnének látni, és összehasonlítani a telefonok teljesítményét. Ők nagyra értékelik a részletes specifikációkat és benchmark-eredményeket.
3. **Mobiltelefon-kereskedők**: Akik gyors és pontos összehasonlításra támaszkodnak az ügyfelek tájékoztatásához.

Áttekintés

Mobiltelefon-specifikációk és iparági szabványok

Az okostelefonok specifikációi kulcsfontosságú szerepet játszanak a felhasználói élményben és a vásárlási döntésekben. A legfontosabb paraméterek közé tartozik a processzor teljesítménye, a RAM mérete, a kamera minősége, a kijelző technológiája és az akkumulátor élettartama. Az iparágban elfogadott szabványok, mint például az 5G hálózati támogatás, az OLED kijelzők fejlődése vagy a többkamerás rendszerek elterjedése, meghatározzák a készülékek funkcionalitását és élettartamát.

Szűrőrendszerek és algoritmusok

A modern adatkezelési és szűrési technológiák lehetővé teszik, hogy a felhasználók testreszabott kereséseket végezzenek. A relációs adatbázisok (pl. MySQL, SQLite) és a NoSQL megoldások (pl. MongoDB) egyaránt alkalmasak nagy mennyiségű telefonos adat kezelésére. Az olyan algoritmusok, mint a többkritériumos döntéstámogatás (Multi-Criteria Decision Analysis – MCDA) vagy a mesterséges intelligencia alapú ajánlórendszerek, tovább növelhetik a felhasználói élményt.

Felhasználói élmény (UX) és design szempontok

A felhasználói élmény kulcsfontosságú egy sikeres mobiltelefon-összehasonlító platform esetében. Az átlátható felhasználói felület, az intuitív szűrési lehetőségek és a vizuálisan jól strukturált összehasonlítások hozzájárulnak a vásárlói döntéshozatal egyszerűsítéséhez. A modern UX-design trendek, mint a minimalizmus és az interaktív grafikonok, tovább növelhetik a ChoosePhone projekt vonzerejét.

Technológiai háttér

A ChoosePhone egy webalapú alkalmazásként működik, amely modern frontend és backend technológiákat alkalmaz. A frontend fejlesztés során népszerű JavaScript-keretrendszerek, például React vagy az Angular használhatóak a dinamikus felhasználói élmény biztosítására, a projekt esetében az Angular került kiválasztásra. A backend esetében Node.js vagy PHP kínálhat megfelelő teljesítményt és skálázhatóságot.

Tervezés és követelmények

A ChoosePhone rendszer tervezése során figyelembe kell venni a funkcionalitást, a teljesítményt és a felhasználói élményt biztosító követelményeket. Az alábbiakban bemutatjuk a legfontosabb követelményeket és az architekturális tervezést.

Funkcionális követelmények

- **Felhasználói regisztráció és bejelentkezés**: A felhasználók számára lehetőséget kell biztosítani a profil létrehozására és kezelésére.
- **Telefonok összehasonlítása**: A rendszernek lehetővé kell tennie, hogy a felhasználók két vagy több készüléket részletesen összehasonlítsanak.
- **Szűrési és keresési lehetőségek**: Több szempont alapján történő keresés és szűrés szükséges (ár, kijelzőméret, processzor, kamera, akkumulátor stb.).
- **Adatbázis frissítése**: Az újonnan megjelent készülékek és friss specifikációk folyamatosan frissüljenek az adatbázisban.
- **Felhasználói értékelések és vélemények**: A látogatók számára biztosítani kell a lehetőséget, hogy értékeléseket írjanak és megoszthassák tapasztalataikat.

Nem funkcionális követelmények

- **Teljesítmény**: Az oldalnak gyorsan kell betöltenie, és a szűréseknek valós időben kell működniük.
- **Biztonság**: A felhasználói adatok védelme és a biztonságos bejelentkezés elengedhetetlen.
- **Skálázhatóság**: A rendszernek bővíthetőnek kell lennie, hogy a növekvő felhasználói bázist kiszolgálja.
- **Mobilbarát kialakítás**: Az alkalmazásnak reszponzív designnal kell rendelkeznie, hogy mobiltelefonon és tableten is jól működjön.

Rendszertervezés

A ChoosePhone architektúrája három fő komponensből áll:

1. **Frontend**: A projektünkhöz az Angular keretrendszert választottuk, amely komponens-alapú megközelítésével lehetővé teszi a dinamikus, jól strukturált felhasználói felület kialakítását. Az Angular előnye a kétirányú adatbinding, a moduláris felépítés és a gazdag ökoszisztéma, ami megkönnyíti a skálázhatóságot és a karbantartást.
2. **Backend**: A backend fejlesztéséhez a NestJS keretrendszert alkalmazzuk, mely a Node.js alapjaira épül, és modern, típusbiztos architektúrát kínál. A NestJS lehetővé teszi a moduláris, könnyen bővíthető RESTful API-k létrehozását, valamint integrált dependency injection és egyéb fejlesztési minták támogatását, amelyek elősegítik a kód tisztaságát és újrafelhasználhatóságát.
3. **Adatbázis**: Az adatok tárolására az SQLite relációs adatbázis-kezelőt választottuk. Az SQLite egy könnyen telepíthető, beágyazott megoldás, amely ideális kisebb és közepes méretű alkalmazások esetén, egyszerűséget és alacsony erőforrásigényt biztosítva.

Fejlesztés folyamata, indoklása

Fejlesztői környezet

A projektünk fejlesztésében két fő eszközt emelnénk ki:

- **Visual Studio Code (VS Code)**

  A VS Code egy modern, könnyen használható és testreszabható integrált fejlesztői környezet (IDE), amely ideális választás Angular és NestJS alapú fejlesztéshez. Az eszköz gazdag bővítmény-kínálata támogatja a különböző programozási nyelvek és technológiák integrációját, így segítve a gyorsabb és hatékonyabb kódolást. Emellett a beépített hibakereső és Git támogatás is megkönnyíti a fejlesztési folyamatot.

- **Git verziókezelő rendszer**

  A Git alkalmazása lehetővé teszi a kódváltozások nyomon követését, a verziók kezelését és a csapatmunkát. Az elosztott verziókezelés révén minden fejlesztő saját lokális példányban dolgozhat, miközben a központi repóban történik a végleges integráció. Ez nemcsak a hibák visszakövetését könnyíti meg, hanem az együttműködést és a fejlesztési folyamat átláthatóságát is jelentősen javítja.

Ezek az eszközök együtt biztosítják, hogy a fejlesztési folyamat gördülékeny, átlátható és jól dokumentált legyen, támogatva a hatékony kódolást és a csapatmunkát.

Backend fejlesztés *[2]*

Technológiai alapok és architektúra

A NestJS egy progresszív Node.js keretrendszer, amely robusztus, skálázható és könnyen karbantartható szerveroldali alkalmazások fejlesztésére alkalmas.

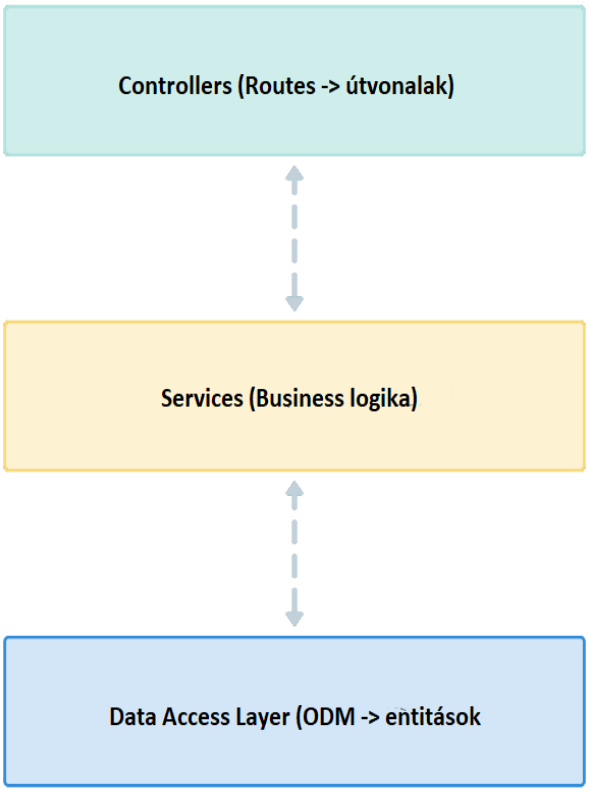
Kombinálja az Objektumorientált Programozás (OOP), Funkcionál Programozás (FP) és Funkcionál Reaktív Programozás (FRP) elemeit egyedi fejlesztési élményt nyújtva.

A Typescript használata és a magas fokú moduláris architektúra révén tiszta, hatékony és biztonságos alkalmazásokat tesz lehetővé.

A NestJS segítségével tiszta, hatékony és biztonságos szerveroldali alkalmazásokat hozhatunk létre, amelyek különösen előnyösek webes API-k, mikroszolgáltatások és valós idejű rendszerek fejlesztésében. Az architektúra modularitása váratlan előnyt jelenthet a fejlesztők számára, például gyorsabb hibajavításban és csapatmunkában.

Jellemző	NestJS	Express	hapi	Fastify
Natív programozási nyelv	TypeScript	JavaScript	JavaScript	JavaScript
Függőség injektálás (Dependency Injection)	Igen – beépített	Igen – külön kell megvalósítani	Igen – külön kell megvalósítani	Igen – külön kell megvalósítani
Útvonalkezelés	Igen	Igen	Igen	Igen
Middleware támogatás	Igen	Igen	Igen	Igen
Hibakezelés	Igen	Igen	Igen	Igen
Tesztelés	Igen- rengeteg beépített eszközzel	Igen	Igen	Igen
GraphQL támogatás	Igen – rengeteg beépített eszközzel	Igen	Igen	Igen
Mikroszervíz támogatás	Igen – rengetek beépített eszközzel	Igen	Igen	Igen
Közösség	Nagy és aktív	Aktív	Aktív	Aktív
Dokumentáció	Kiváló	Jó	Jó	Jó
Teljesítmény	Kiváló	Nem rossz	Jó	Jó
Skálázhatóság	Kiváló	Jó	Jó	Jó
Karbantarthatóság	Jó	Nem rossz	Jó	Jó

1. ábra Összehasonlítás más Node.js keretrendszerekkel



2. ábra A NestJS alap architektúrája

TypeScript

Egy olyan környezetben, ahol a JavaScript uralja a szerveroldali fejlesztést, a TypeScript különleges előnyt biztosít alapos típusellenőrzésével és objektumorientált lehetőségeivel. Ez a rész arra világít rá, hogy miért nem csupán hasznos, hanem szinte nélkülözhetetlen választás a TypeScript, ha a NestJS-szel dolgozunk.

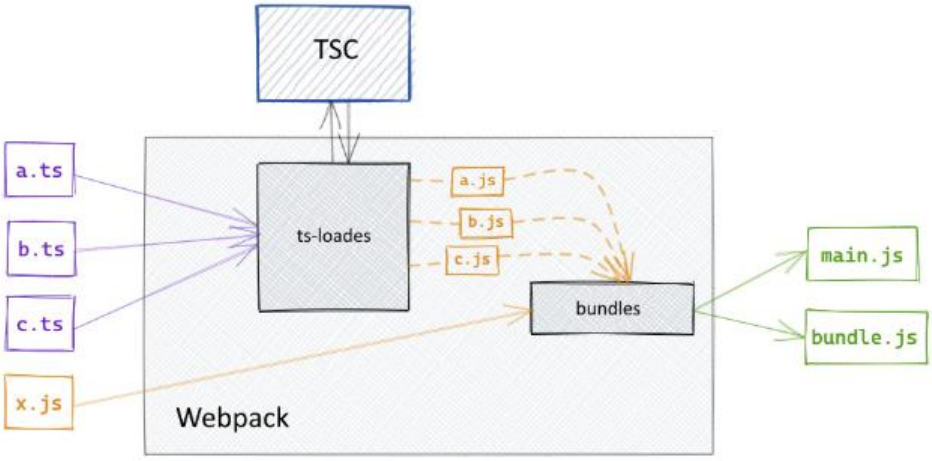
Miért éppen a TypeScript?

A JavaScript, bár széles körben elterjedt és sokoldalú, nem nyújt teljes megoldást a típusbiztonság és az öndokumentáló kód terén. A TypeScript, amely a JavaScript egy kiterjesztett változata, a Microsoft fejlesztésében született meg ezeknek a hiányosságoknak a kiküszöbölésére. Ha egy szerveroldali keretrendszer, például a NestJS kontextusában használjuk, a TypeScript előnyei még inkább kiemelkednek.

Mit kínál a TypeScript?

A TypeScript statikus típuskezelést hoz a JavaScript világába, lehetővé téve az adattípusok meghatározását már a fordítási időben. Ez számos fontos előnnyel jár:

- **Hibacsökkentés**: A típusokkal kapcsolatos hibák fordítási időben történő kiszűrése rengeteg hibakeresési időt takarít meg, és megelőzheti a rendszer esetleges meghibásodásait.
- **Kódminőség**: A statikus típusozás gyakran olvashatóbbá és öndokumentálóvá teszi a kódot, így a csapat tagjai könnyebben értik meg a kódalap tartalmát.
- **IDE-támogatás**: A modern fejlesztőkörnyezetek (IDE-k) jobb IntelliSense és automatikus kiegészítési funkciókat kínálnak a TypeScript használatakor, ami gyorsabbá és hatékonyabbá teszi a fejlesztést.



3. ábra Diagram a TypeScript sima JavaScript-be való fordításáról és a típus-ellenőrzési folyamatáról

```
//javascript code
function add(a, b) {
  return a + b;
}
console.log(add(5, "10"));  // Output: 510
```

Ez a JavaScript kód egy buggot tartalmaz, mivel egy string-es értéket akar összeadni egy numerikus értékkel.

```
// typescript code
function add(a: number, b: number): number {
  return a + b;
}
console.log(add(5, "10"));  // Error: Argument of type
                            // 'string' is not assignable
                            // to parameter of type
                            //                    'number'.

!! Figyelje meg, hogyan észleli a TypeScript a típuseltérést
a fordítási időben, csökkentve annak esélyét, hogy a hibák
bekerüljenek az éles kódba.
```

Ez pedig a TypeScript kód, ami megoldja a problémánkat.

TypeScript és a NestJS – a tökéletes harmónia

A TypeScript funkciói tökéletesen illeszkednek a NestJS alapelveihez. Íme, hogyan:

- **Dekorátorok**: A NestJS és a TypeScript egyaránt használ dekorátorokat a metadatok tükrözésére, ami tisztább és rendezettebb kódot eredményez.
- **Erős típusosság**: Ez biztosítja, hogy a NestJS vezérlők, szolgáltatások és egyéb osztályok robusztusak és könnyen karbantarthatók legyenek.
- **Modularitás**: A TypeScript névtere és modulrendszere segít a kód szervezésében, így skálázhatóvá téve azt, ami összhangban van a NestJS moduláris architektúrájával.

Kódrészlet – NestJs a TypeScript-tel

Íme egy részlet egy egyszerű NestJs service-ről:

```
// typescript kód
import { Injectable } from '@nestjs/common';
@Injectable()
export class AppService {
  getData(): string {
    return 'Hello, NestJS!';
  }
}
```

Ebben a kódrészletben az **@Injectable()** dekorátor és a **: string** visszatérési típus mind példái annak, hogyan segíti elő a TypeScript a NestJS fejlesztést.

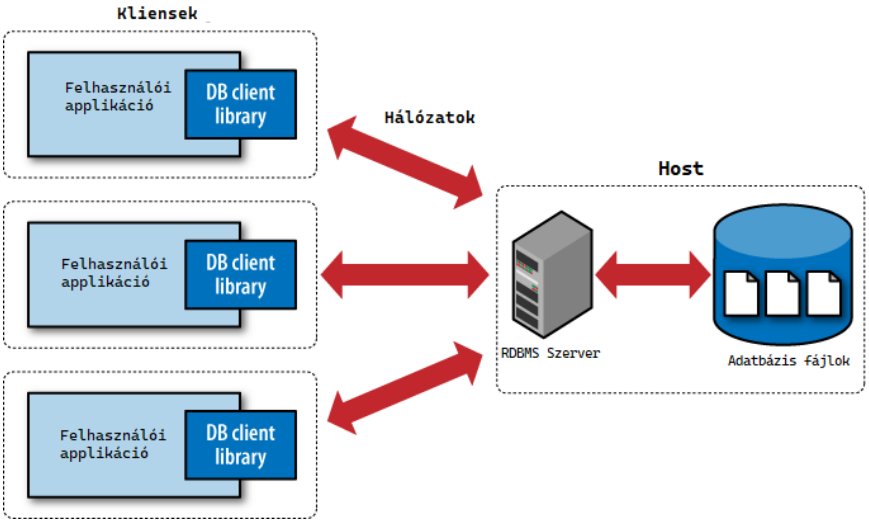
A TypeScript integrálásával a NestJS olyan fejlesztési környezetet kínál, amely robusztus, hatékony, és kevésbé hajlamos a hibákra. A TypeScript statikus típusellenőrzése, fejlettebb IDE támogatása és a modern programozási paradigmákkal való kompatibilitása mind elengedhetetlen részei a NestJS ökoszisztémának.

SQLite adatbázis *[3]*

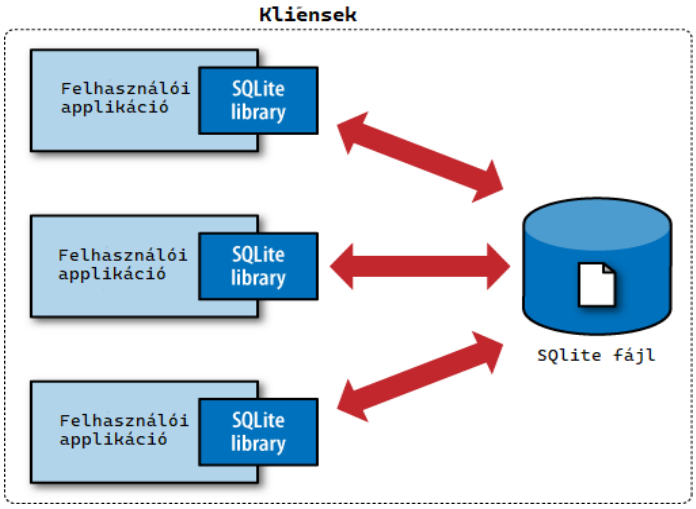
Az adatbáziskezeléshez SQLite-ot alkalmaztunk, mivel a fejlesztési és tesztelési fázis során egy könnyen telepíthető, konfigurálható és fájl-alapú relációs adatbázisra volt szükség. Az SQLite előnye, hogy nem igényel külön adatbázis-szervert, így egyszerűen integrálható és gyorsan beállítható egy prototípus vagy kisebb projektek számára. A későbbi skálázás során szükség esetén egy robusztusabb adatbázis-kezelőre, például PostgreSQL-re lehet áttérni anélkül, hogy jelentős változtatásokat kellene végezni az alkalmazás kódbázisán.

Az SQLite egy beágyazott adatbázis. Nem egy önálló folyamatként fut, hanem az általa kiszolgált alkalmazás folyamatterében, annak szerves részeként működik. A forráskódja közvetlenül beépül a fogadó alkalmazásba, így a külső szemlélő számára láthatatlan marad, hogy az adott program egy relációs adatbázis-kezelő rendszert (RDBMS) tartalmaz. A program egyszerűen végzi a feladatát és kezeli az adatait anélkül, hogy külön figyelmet fordítana arra, hogyan is teszi ezt. A háttérben azonban egy teljes, önálló adatbázis-motor dolgozik.

Az egyik legnagyobb előnye annak, hogy az adatbázis-szerver közvetlenül az alkalmazáson belül fut, hogy nincs szükség hálózati konfigurációra vagy adminisztrációra. Gondoljunk csak bele, milyen felszabadító ez: nincs szükség tűzfalak kezelésére, címfeloldásra vagy összetett jogosultságok beállítására. Az ügyfél és a szerver ugyanabban a folyamatban fut, ami csökkenti a hálózati kommunikációval járó többletterhelést, egyszerűsíti az adatbázis-kezelést, és megkönnyíti az alkalmazás telepítését. Minden szükséges komponens közvetlenül a programba van beépítve.



4. ábra A hagyományos relációs adatbázis-kezelő rendszerek kliens/szerver architektúrája, amely kliens könyvtárat alkalmaz.



5. ábra Az SQLite szerver nélküli architektúrája.

SQL az SQLite-ban

Az SQL az egyedüli (és szinte mindenhol alkalmazott) eszköz, amellyel kommunikálni lehet egy relációs adatbázissal. Ez egy olyan nyelv, amely kizárólag az információk feldolgozására szolgál. A nyelvet az információk struktúrázására, olvasására, írására, rendezésére, szűrésére, védelmére, számítására, generálására, csoportosítására, aggregálására és általában véve az információk kezelésére tervezték.

Az SQL egy intuitív, felhasználóbarát nyelv. Használata szórakoztató lehet, és meglehetősen erőteljes. Az SQL egyik érdekes tulajdonsága, hogy függetlenül attól, hogy szakértő vagy kezdő vagy, mindig találhatsz új módokat a dolgok elvégzésére (akár jobb, akár rosszabb eredménnyel). Gyakran többféle megoldás létezik egy adott problémára, és előfordulhat, hogy élvezettel keresed a hatékonyabb módokat a kívánt eredmény elérésére, akár tömörebb kifejezésekkel, akár elegánsabb megoldásokkal, mintha egy rejtvényt oldanál meg. Ráadásul folyamatosan felfedezheted a nyelv olyan elfeledett sarkait, amelyeket eddig nem vettél észre, és amelyek segítségével tovább fejlesztheted a tudásodat, bárhány éves tapasztalattal rendelkezel is.

Relációs modell

A relációs modell három alapvető részből áll: forma, funkció és konzisztencia. A forma az információ struktúrájára utal. Egyetlen adatstruktúra létezik, amely minden információt reprezentál. Ezt a struktúrát relációnak nevezik (SQL-ben tábla), amely tuplákból áll (SQL-ben sorok), amelyek pedig attribútumokból (SQL-ben oszlopok) épülnek fel.

A relációs modell formája az információ logikai reprezentációja. Ez a logikai reprezentáció egy tiszta, absztrakt nézetet ad az információról, amelyet semmi nem befolyásol kívülről. Olyan, mint egy matematikai fogalom: tiszta és következetes, egy jól meghatározott, determinisztikus szabályrendszer irányítja, amely nem változik. A logikai reprezentáció teljesen független a fizikai reprezentációtól, amely arra utal, hogyan tárolja az adatbázis szoftver ezt az információt a fizikai szinten (pl. lemez). Így a két reprezentáció különálló: semmi, ami a fizikai szinten történik, nem változtathat meg vagy befolyásolhat semmit a logikai szinten. A logikai szintet nem korlátozzák a hardverek, szoftverek, gyártók vagy technológiák.

A modell második alapvető része a funkcionális rész, más néven manipulációs komponens. Ez határozza meg, hogyan lehet műveleteket végezni az információval a logikai szinten. Ezt formálisan Codd 1972-es „Relational Completeness of Data Base Sublanguages” című cikkében vezette be. A relációs modellt a relációs algebra és a relációs kalkulus meghatározásával bővítette. Ezek két formális, vagy "tiszta" lekérdező nyelv, amelyek erőteljes matematikai alapokon nyugszanak. A relációk, mint ahogy az adatmodell leírja, matematikai halmazok (néhány további tulajdonsággal). A relációs algebra és kalkulus a modellre építve halmazelméleti és formális logikai műveleteket ad hozzá, így képezve a relációs modell funkcionális komponensét. Tehát a relációs modell formája és funkciója közvetlenül a matematikai fogalmakkal származik. Minden egyes származék azonban egy kis pluszt ad hozzá, hogy jobban alkalmazkodjon a számítógépekhez és az információfeldolgozáshoz.

Backend fejlesztésének folyamata

Az alkalmazásfejlesztés és az informatikai rendszerek integrációja szempontjából az API-k (Application Programming Interfaces) kulcsszerepet játszanak. Ezek az interfészek biztosítják, hogy különböző szoftverkomponensek és rendszerek hatékonyan kommunikáljanak egymással, lehetővé téve az adatok és funkciók megosztását.

Az API-k használata elengedhetetlen a modern fejlesztésben, mivel segítségükkel a fejlesztők gyorsabban és hatékonyabban építhetnek skálázható, moduláris és könnyen karbantartható alkalmazásokat. Különösen fontosak a webes és mobilalkalmazások esetében, ahol az adatok gyakran távoli szerverekről érkeznek, és az API-kon keresztüli kommunikáció biztosítja a zavartalan működést.

A projektünk kezdetekor létrehoztunk az alapvető API-kat, amik a fejlesztéskor nagy segítség lesz. Itt van néhány kódrészlet amiket felhasználtunk a webalkalmazásunkban:

```
 findOne(id: number) {
     return this.telephoneRepository.findOne({ where: { id }, relations:
['specs', 'os', 'display', 'camera', 'battery'] });
   }
```

Ez a metódus egyetlen *Telephone* entitást kér le az adatbázisból az id alapján. Emellett a kapcsolódó entitásokat is betölti *(specs, os, display, camera és battery*) ugyanabban a lekérdezésben, a TypeORM eager loading funkcióját használva.

```
 async search(searchTerm: string): Promise<Telephone[]> {
     if (!searchTerm || searchTerm.trim() === '') {
       return [];
     }

 const queryBuilder =
this.telephoneRepository.createQueryBuilder('telephone');

     queryBuilder.leftJoinAndSelect('telephone.specs', 'specs');
     queryBuilder.leftJoinAndSelect('telephone.os', 'os');
     queryBuilder.leftJoinAndSelect('telephone.display', 'display');
     queryBuilder.leftJoinAndSelect('telephone.camera', 'camera');
     queryBuilder.leftJoinAndSelect('telephone.battery', 'battery');

     const loweredSearchTerm = searchTerm.toLowerCase();

     return queryBuilder
       .andWhere('LOWER(telephone.modell) LIKE :searchTerm', { searchTerm:
`%${loweredSearchTerm}%` })
       .orWhere('LOWER(telephone.marka) LIKE :searchTerm', { searchTerm:
`%${loweredSearchTerm}%` }).getMany();

   }
```

Ez egy keresőfunkció egy TypeScript osztályban, amely a NestJS alkalmazás része. A megadott keresési kifejezés alapján telefonokat keres az adatbázisban. Az alábbiakban röviden összefoglalva a működését:

- Először ellenőrzi, hogy a keresési kifejezés üres-e vagy csak szóközöket tartalmaz. Ha igen, akkor egy üres tömböt ad vissza.
- Ezután egy lekérdezésépítőt hoz létre az adatbázis-lekérdezés összeállításához.
- A keresés során több kapcsolódó táblát (*specs, os, display, camera és battery*) is csatlakoztat a *telephone* táblához.
- A keresési kifejezést kisbetűssé alakítja, és a modell és marka oszlopokban keresi, figyelmen kívül hagyva a kis- és nagybetűk közti különbségeket. A keresés részleges egyezéseket is támogat, mivel a % karaktereket használja helyettesítőként.
- Végül végrehajtja a lekérdezést, és az eredményeket egy *Telephone* objektumokat tartalmazó tömbként adja vissza.

```
  async findByFilter(termekSzuroDto: TermekSzuroDto): Promise<Telephone[]> {
      const queryBuilder =
this.telephoneRepository.createQueryBuilder('telephone');

      queryBuilder.leftJoinAndSelect('telephone.specs', 'specs');
      queryBuilder.leftJoinAndSelect('telephone.os', 'os');
      queryBuilder.leftJoinAndSelect('telephone.display', 'display');
      queryBuilder.leftJoinAndSelect('telephone.camera', 'camera');
      queryBuilder.leftJoinAndSelect('telephone.battery', 'battery');


      if (termekSzuroDto.markak && termekSzuroDto.markak.length > 0) {
        queryBuilder.andWhere('telephone.marka IN (:...markak)', { markak:
termekSzuroDto.markak });
      }

      if (termekSzuroDto.minAr !== undefined) {
        queryBuilder.andWhere('telephone.ar >= :minAr', { minAr:
termekSzuroDto.minAr })
      }
```

```
      if (termekSzuroDto.maxAr !== undefined) {
queryBuilder.andWhere('telephone.ar <= :maxAr', { maxAr: termekSzuroDto.maxAr })
      }
```

Ez egy TypeScript osztályban található metódus, amely különböző szempontok alapján szűri a telefonkészülékek listáját. A **findByFilter** metódus egy *TermekSzuroDto* típusú objektumot kap paraméterként, amely tartalmazza a szűrési lehetőségeket, például a márkát, az árkategóriát, a RAM-méretet, a tárhelyet és egyéb specifikációkat.

A metódus egy lekérdezésépítőt használ az adatbázis-lekérdezés összeállítására, amely a megadott szűrési feltételek alapján szűri a telefonokat. Egy sor *if* feltételt alkalmaz, hogy csak azokat a szűrési feltételeket vegye figyelembe, amelyek ténylegesen jelen vannak a *TermekSzuroDto* objektumban.

A lekérdezés során több kapcsolódó táblát (*specs, os, display, camera és battery*) is összekapcsol a telephone táblával, hogy részletesebb információt szolgáltasson az egyes készülékekről.

A metódus egy ígéretet (**Promise**) ad vissza, amely egy olyan *Telephone* objektumokat tartalmazó tömbre oldódik fel, amely megfelel a megadott szűrési feltételeknek.

Összességében ez a metódus lehetővé teszi a telefonkészülékek széles körű szűrését, így rugalmas és hatékony eszközt biztosít a kereséshez és a megfelelő eszközök lekérdezéséhez.

Frontend fejlesztés

Technológiai alapok és architektúra *[4]*

Az Angular, az egyik legnépszerűbb frontend fejlesztési keretrendszer, egy fordulóponthoz érkezett. Az elmúlt évek során jelentős fejlesztéseken ment keresztül a teljesítmény, a felhasználói élmény és az új funkciók terén – például az Ivy renderelőmotor bevezetésével, amely csökkentette a csomagméretet és javította a futtatási sebességet. Ezek az előrelépések kedvező helyzetbe hozták a keretrendszert.

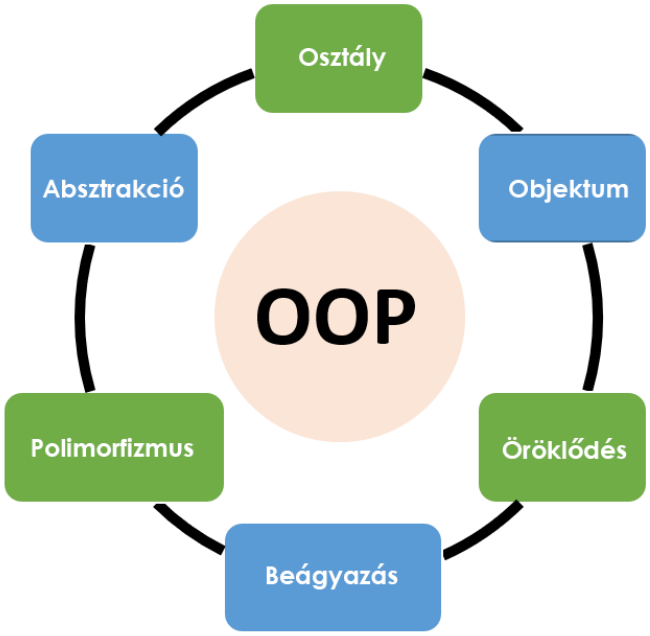
Mostantól a közösség nemcsak a keretrendszer látható elemeinek fejlesztésére összpontosíthat, hanem olyan tényezőkre is, amelyek közvetlen hatással vannak a felhasználói élményre. Különösen nagy hangsúlyt kaphatnak a fejlesztői élményt érintő fejlesztések, például a jobb skálázhatóság és komponálhatóság. Ezek az újítások különösen értékesek azok számára, akik napi szinten dolgoznak az Angularral. A keretrendszer korábbi verziói már jelentős fejlődést hoztak mind a felhasználói, mind a fejlesztői élmény terén, és a jövőbeni kiadások további fejlesztéseket ígérnek.

Az Angular alapvető funkciói

Ebben az írásban az Angular keretrendszer néhány, de nem minden fontos aspektusát tárgyaljuk. Elsősorban azokra a funkciókra összpontosítok, amelyek jelentős átalakuláson mennek keresztül a legújabb kiadások révén.

Objektumorientált programozás

Az objektumorientált programozás (OOP) hosszú ideig az összetett vállalati projektek egyik meghatározó jellemzője volt, különösen a Java és C# nyelvek népszerűségének köszönhetően. Ez volt az alapértelmezett megközelítés a bonyolult alkalmazások kezelésére. Az Angular maga is szoros és összetett kapcsolatban áll az OOP-val.



6. ábra Az OOP programozás felépítése

Az Angular fő építőelemei – például a komponensek, pipe-ok, direktívák és guardok – hagyományosan objektumorientált módon készültek. Mindegyik egy osztályként van reprezentálva, az adatok tulajdonságként tárolódnak, a viselkedésüket pedig metódusok határozzák meg. Azonban a következő fejezetekben, különösen a 3. és 10. fejezetben, látni fogjuk, hogy ezek közül az építőelemek közül néhány nem feltétlenül igényli az osztályalapú megközelítést. Sőt, egyes esetekben az osztályok használata akár félrevezető is lehet, különösen azok számára, akik más frontend keretrendszerekből, például a funkcióorientált Reactból érkeznek. Megnézzük, hogyan változik meg ez a paradigma.

Függőséginjektálás

A függőséginjektálás (DI) az Angular egyik legfontosabb és legvonzóbb jellemzője, amelyet szinte minden Angular fejlesztő használt már valamilyen projektben. Korábban a DI teljes mértékben össze volt kötve az osztályokkal és az objektumorientált programozással: a függőséginjektáláshoz elengedhetetlen volt egy osztály az @Injectable dekorátorral. Ebből adódóan egy szolgáltatás, konfiguráció vagy más DI-fában lévő elem nem volt elérhető egy függvényből, hacsak azt kifejezetten paraméterként nem adtuk át. Ez a megkötés azonban csökkentette a komponálhatóságot. Az új inject függvény bevezetésével ez a korlátozás megszűnt, megnyitva az utat egy rugalmasabb és újrafelhasználhatóbb megközelítés felé.

Modulalapú architektúra

A 14-es verzió előtt minden Angular alkalmazás az NgModules köré épült. Az NgModule egy speciális Angular koncepció, amely egy osztályba szervezte az alkalmazás többi építőelemét, biztosítva azok együttműködését. Az NgModule-ok kulcsszerepet játszottak az architektúra kialakításában és a funkcionalitások megosztásában. Ugyanakkor sok fejlesztő problémákba ütközött az NgModules használata során. Az újabb Angular verziók lehetővé teszik, hogy az alkalmazások NgModules nélkül is épülhessenek, egy úgynevezett „standalone” megközelítést alkalmazva.

RxJS

Az RxJS, a JavaScript reaktív kiterjesztési könyvtára, kulcsszerepet játszik az Angular alkalmazások állapotmegosztásában. Például az autentikációs és jogosultságkezelési események (hozzáférés engedélyezése vagy visszavonása) gyakran ezen keresztül propagálódnak. Az alkalmazások többsége vagy egy egyszerű állapotkezelési megoldást épít egy szolgáltatáson és egy Subjecten keresztül, vagy egy már meglévő állapotkezelő könyvtárat, például az NgRx-et használja, amely szintén RxJS-re épül. Az új Angular verziók jelentősen növelték az Angular és az RxJS közötti interoperabilitást.

Változásérzékelés

A változásérzékelés, amely biztosítja, hogy a komponensek adatváltozásai megfelelően megjelenjenek a felhasználói felületen, egy összetett és nem mindig optimális algoritmus. Jelenleg egy külső könyvtárra, a zone.js-re támaszkodik, amely sok fejlesztő számára jelentős többletterhelést okoz. Sokan keresik azokat a megoldásokat, amelyekkel ki lehet kerülni ezt a réteget. A következő fejezetekben részletesen bemutatjuk a változásérzékelés működését, valamint azt, hogyan lehet hatékonyabbá tenni a folyamatot.

Modern Angular app készítése

Az Angular-projektek létrehozására többféle módszer létezik, beleértve különböző egyedi és harmadik féltől származó builderek, bundlerek és egyéb eszközök használatát
Az Angular CLI számos különböző parancsot és egyedi sémát tartalmaz, amelyek lehetővé teszik a fejlesztők számára, hogy gyorsan és hatékonyan hozzanak létre, konfiguráljanak, valamint kezeljenek Angular projekteket, miközben a legjobb gyakorlatokat követik.

Most azonban a legismertebb parancsra, az ng new-re összpontosítunk, amely új projektek létrehozására szolgál. Az ng new több testreszabási lehetőséget kínál, amelyeket az Angular dokumentáció erre dedikált szekciója részletesen ismertet. Ez a parancs az Angular CLI egyik alapvető eszköze, amely lehetővé teszi a fejlesztők számára, hogy gyorsan és hatékonyan hozzanak létre új Angular projekteket. Amikor futtatjuk az ng new parancsot, az Angular CLI automatikusan generálja a projekt alapvető struktúráját, beleértve a szükséges fájlokat és mappákat, mint például az *app.component.ts,* az *app.module.ts* vagy a *package.json.*

Az ng new parancs működése

Az ***ng new*** parancs alapvetően egy új Angular projekt inicializálására szolgál. A parancs futtatásakor a CLI létrehozza a projekt gyökérkönyvtárát, telepíti a szükséges függőségeket a Node.js csomagkezelőjével (***npm***), és egy működőképes alapprojektet állít elő. Ez a struktúra már önmagában is alkalmas arra, hogy azonnal elkezdhessük a fejlesztést, hiszen tartalmazza az alapvető komponenseket, konfigurációs fájlokat és egy egyszerű induló alkalmazást.

Testreszabási lehetőségek

Az ng new parancs igazi erőssége a testreszabhatóságában rejlik. A fejlesztők számos opcióval finomhangolhatják a projekt kezdeti beállításait, hogy az megfeleljen az igényeiknek. Néhány fontos testreszabási lehetőség:

- **Projekt neve**: A parancs alapértelmezett használata az `ng new projekt-nev`, ahol a projekt-nev meghatározza a mappa nevét és a projekt azonosítóját.
- **Stíluslap formátuma**: Az --style kapcsolóval kiválasztható a projektben használt stíluslap típusa, például css, scss, sass vagy less. Például: ng new projekt-nev --style=scss.
- **Routing modul**: A --routing opcióval eldönthetjük, hogy a projekt tartalmazzon-e előre konfigurált útválasztási modult (pl. ng new projekt-nev --routing).
- **Minimális projekt**: A --minimal kapcsolóval egy egyszerűbb, kevesebb boilerplate kódot tartalmazó projektet hozhatunk létre, ami például tesztelési célokra hasznos.
- **Könyvtár megadása**: A --directory opcióval a projektet egy adott mappába helyezhetjük (pl. ng new projekt-nev --directory=./sajat-mappa).

Ezek az opciók lehetővé teszik, hogy a projektet már az induláskor a kívánt módon állítsuk be, így kevesebb utólagos konfigurációra van szükség. Ezenkívül az Angular csapata még számos új opcióval is fejlesztheti a testreszabási lehetőségeket.

Bevezetés a PrimeNG UI könyvtárába *[1]*

Az Angularral együtt használva a PrimeNG számos előre elkészített UI komponenst kínál, amelyeket kifejezetten Angularhoz terveztek. Ez a páros egy rendkívül hatékony kombinációt alkot, mivel a PrimeNG kiegészíti az Angular képességeit egy kész UI komponenscsomaggal, amely megkönnyíti bármilyen webalkalmazás fejlesztését. Ezek a komponensek támogatják a fejlesztőket esztétikus és felhasználóbarát felületek létrehozásában, leegyszerűsítve a fejlesztési folyamatot, miközben biztosítják a következetes és élvezetes felhasználói élményt.

A PrimeNG egy funkciókban gazdag, nyílt forráskódú UI komponenskönyvtár, amelyet kifejezetten Angular alkalmazásokhoz terveztek. Amint az alábbi ábrán is látható, a PrimeNG jelenlegi verziója lenyűgöző kínálatot nyújt: több mint 90 komponenst, 200-nál is több ikont, valamint 400+ előre elkészített UI blokkot tartalmaz. A kínálat az egyszerűbb elemeket, például gombokat és beviteli mezőket, valamint összetettebb és fejlettebb komponenseket, mint például adattáblázatokat, diagramokat és fastruktúrákat is magában foglal.

A PrimeNG minden egyes komponensét gondos tervezéssel alakították ki. Nem csupán funkcionálisak, hanem esztétikailag is kifinomultak, modern UI elveket követő letisztult dizájnnal. A komponensek alapértelmezés szerint is gazdag funkcionalitással rendelkeznek, emellett rugalmasan testre szabhatók, hogy tökéletesen illeszkedjenek az adott alkalmazás igényeihez.

Az Angular és a PrimeNG párosítása

Az Angular és a PrimeNG együttes alkalmazása hatékony eszköztárat biztosít a korszerű webfejlesztéshez. Ez a kombináció egy rendkívül produktív környezetet hoz létre, amely nagyban megkönnyíti az összetett, interaktív webalkalmazások készítését. Nézzük a legfontosabb előnyöket:

Tökéletes összhang az Angularral

A PrimeNG-t kifejezetten az Angularhoz tervezték, így minden egyes komponense hibátlanul illeszkedik az Angular felépítéséhez. A PrimeNG elemei gyakorlatilag Angular komponensek, ezért ugyanúgy használhatók, mint bármely más Angular elem. Ez a szoros kompatibilitás gördülékeny fejlesztési folyamatot tesz lehetővé, hiszen a PrimeNG komponenseket különösebb extra erőfeszítés vagy integrációs nehézség nélkül építheted be az Angular alkalmazásodba.

Kiegészítő funkcionalitás

Az Angular és a PrimeNG remekül kiegészíti egymást. Az Angular egy stabil keretrendszert kínál az egyoldalas alkalmazások (SPA) fejlesztéséhez, támogatva az összetett interakciókat és az állapotkezelést. Ezzel szemben a PrimeNG rengeteg előre elkészített felhasználói felület (UI) elemet biztosít, amelyekkel látványos és reszponzív dizájnt alakíthatsz ki. Így az Angular erejét az alkalmazás logikájának és szerkezetének megtervezésére használhatod, míg a PrimeNG-vel a felhasználói élményt teheted vonzóbbá.

Jobb kódminőség

A PrimeNG és az Angular integrációja a kód minőségét is javítja. A PrimeNG kész, újrafelhasználható komponensei révén elkerülheted a redundáns kódolást, így az egyedi funkciók és az üzleti logika megvalósítására összpontosíthatsz. Ez letisztultabb és könnyebben karbantartható kódot eredményez, ami különösen nagy projektek esetében előnyös.

Erős közösség és naprakész támogatás

Mind az Angular, mind a PrimeNG aktív közösséggel és részletes dokumentációval büszkélkedhet. Ez rengeteg segítséget jelent, ha elakadsz, vagy mélyebben szeretnél megismerni egy témát. A két eszköz folyamatos frissítései garantálják, hogy mindig a legújabb webfejlesztési trendeknek megfelelő megoldásokkal dolgozhass.

Összegzés

Összességében az Angular erőteljes keretrendszere és a PrimeNG gazdag UI-komponens kínálata teljes körű megoldást nyújt dinamikus és esztétikus webalkalmazások fejlesztéséhez. Ez a párosítás nem csak a hatékonyságot növeli, hanem olyan alkalmazások létrehozását is lehetővé teszi, amelyek gyorsak, skálázhatók és egyszerűen fenntarthatók.

Frontend fejlesztésének folyamata

A frontend fejlesztés egy komplex folyamat, amely több lépésből áll, a tervezéstől a kész alkalmazás telepítéséig. Angular és PrimeNG használatával egy modern, hatékony és vizuálisan vonzó felhasználói felületet (UI) hozhatunk létre. Az alábbiakban részletesen írunk a fejlesztési folyamat fontosabb szegmenseiről.

Tervezés

A fejlesztés első lépése a **tervezési fázis**, amelynek célja az alkalmazás struktúrájának és kinézetének megalapozása. Ebben a szakaszban meghatározzuk:

- Az alkalmazás célját és célközönségét.
- A szükséges funkciókat (pl. űrlapok, táblázatok, navigáció).
- A felhasználói felület (UI) és felhasználói élmény (UX) alapelveit, hogy az alkalmazás intuitív és könnyen használható legyen.

Ezután készítünk egy **drótvázat** (wireframe) vagy **mockupot**, amely vizuálisan ábrázolja az alkalmazás elrendezését. Ez lehet egy egyszerű rajz vagy egy részletesebb, interaktív terv.

Prototípus készítése

A **prototípus** egy kezdeti modell, amely segít bemutatni az alkalmazás kinézetét és működését. Lehet statikus (pl. egy kép), vagy interaktív (pl. egy kattintható demo). Ez a lépés különösen hasznos a stakeholderek (pl. megrendelők) számára, hogy visszajelzést adhassanak, mielőtt a tényleges fejlesztés elkezdődik.

Fejlesztői környezet beállítása

A fejlesztés megkezdéséhez szükséges eszközöket telepítjük:

- **Node.js**: Az Angular futtatásához elengedhetetlen JavaScript környezet.
- **Angular CLI**: Egy parancssori eszköz, amely leegyszerűsíti az Angular projektek létrehozását és kezelését.
- **PrimeNG**: A UI komponensek könyvtára, amelyet az npm segítségével adunk a projekthez.

```
npm install -g @angular/cli # Angular CLI telepítése globálisan
npm install primeng --save # PrimeNG telepítése
npm install primeicons --save # PrimeNG ikonok telepítése
```

Projekt inicializálása

Az Angular CLI segítségével létrehozzuk az új projektet:

```
ng new cp-frontend
```

Ezután a PrimeNG-t integráljuk a projektbe a fenti parancsokkal. A projekt alapstruktúrája automatikusan létrejön, beleértve a szükséges konfigurációs fájlokat (pl. angular.json).

Komponensek létrehozása

Az Angularban az alkalmazás **komponensekből** épül fel, amelyek a felhasználói felület egyes részeit képviselik. Minden komponens három fő részből áll:

- **TypeScript osztály**: A logika helye.
- **HTML sablon**: A megjelenítendő tartalom.
- **CSS/SCSS fájl**: A stílusok.

```
 ng generate component main-page
//Alternatív megoldásként pedig:
ng g c main-page
```

Ennek a parancs segítségével sikeresen legeneráltunk a projektünkbe egy „main-page” elnevezésű plain komponenst.

Szolgáltatások létrehozása (services)

A **szolgáltatások** (services) az alkalmazás logikájának és adatkezelésének központi elemei. Például egy szolgáltatás használható API-hívásokhoz vagy adatfeldolgozáshoz.

Példa egy adatkezelő szolgáltatás létrehozására:

```
 ng generate service data
```

A szolgáltatásban pedig generálhatunk akár egy HTTP kérést a backend oldalhoz:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})

export class DataService {

constructor(private http: HttpClient) { }

getData() {
  return this.http.get('http://localhost:3000/telephones);
 }
}
```

Stílusok és témák alkalmazása

A PrimeNG előre definiált **témákat** kínál (pl. Aura), amelyeket az *angular.json* fájlban adhatunk hozzá:

```
"styles": [ "node_modules/primeicons/primeicons.css",
"node_modules/primeng/resources/themes/aura/theme.css",
"node_modules/primeng/resources/primeng.min.css",
"src/styles.scss" ]
```

Ezenkívül egyedi stílusokat is alkalmazhatunk SCSS vagy CSS segítségével.

Az Angular beépített támogatást nyújt az **egységtesztekhez** (Jest, Jasmine vagy Cypress használatával).
Példa egy komponens tesztjére:

```
describe('MainPageComponent', () => {

  it('should create', () => {

    const fixture = TestBed.createComponent(MainPageComponent);

    const component = fixture.componentInstance;

    expect(component).toBeTruthy();

  });

});
```

Összefoglalás

A frontend fejlesztés Angular és PrimeNG használatával egy strukturált folyamat, amely a tervezéstől a telepítésig tart. A lépések – tervezés, prototípus készítés, környezet beállítás, komponensek és szolgáltatások létrehozása, routing, UI integráció, stílusozás, tesztelés, optimalizálás – együttesen biztosítják egy modern, hatékony alkalmazás elkészítését.

Jövőbeli tervek a projekttel

A **ChoosePhone** célja nem csupán a jelenlegi mobiltelefon-piac átláthatóbbá tétele, hanem egy komplex, intelligens döntéstámogató rendszer kiépítése, amely hosszú távon is megkönnyíti a technológiai eszközök közötti eligazodást. A közeljövőben számos fejlesztést tervezünk bevezetni, amelyek még hatékonyabbá és élvezetesebbé teszik a felhasználói élményt:

1. Árfigyelő és akcióértesítő rendszer

Integrálni szeretnénk egy árkövető funkciót, amely lehetővé teszi, hogy a felhasználók nyomon kövessék a kiválasztott készülékek árának alakulását, és értesítést kapjanak, ha az adott modell akciós áron elérhető.

2. Felhasználói profil és személyre szabott ajánlások

A regisztrált felhasználók beállíthatják preferenciáikat (pl. preferált márka, árkategória, funkciók), mely alapján személyre szabott ajánlásokat és összehasonlításokat kapnak, így még gyorsabban megtalálhatják az ideális készüléket.

3. Közösségi funkciók

Tervezünk bevezetni egy közösségi fórumot és értékelési rendszert, ahol a felhasználók megoszthatják egymással véleményeiket, tapasztalataikat, kérdéseket tehetnek fel, vagy segíthetnek egymásnak a választásban.

4. Kiegészítő eszközök összehasonlítása

A jövőben nem csak okostelefonok, hanem egyéb eszközök – például fülhallgatók, okosórák, töltők – specifikációs összehasonlítását is tervezzük bevezetni, így a ChoosePhone egy teljes körű tech-vásárlási platformmá válhat.

5. Mesterséges intelligencián alapuló kereső és értékelő algoritmus

Egy intelligens rendszer segítségével automatikusan rangsoroljuk a készülékeket a felhasználók igényei alapján, és javaslatokat teszünk a legjobb ár-érték arányú modellekre, figyelembe véve az aktuális piaci trendeket is.

Köszönetnyilvánítás

Köszönettel tartozunk Lakatos Sándor és Berki Balázs konzulens tanárainknak, akik szaktudásukkal, türelemmel és odaadásukkal jelentősen hozzájárultak szakdolgozatunk elkészítéséhez.

Tanácsaik és folyamatos támogatásuk nélkül nem tudtuk volna ezt az értékes tudományos utat végigjárni. Inspirációjuk és szakmai iránymutatásuk révén nemcsak szakmailag, hanem személyesen is fejlődtünk. Nagyra értékeljük az idejüket és energiájukat, amelyet ránk fordítottak a folyamat során.

Ábrajegyzék

Irodalomjegyzék

Felhasznált könyvforrások

[1] Next-Level UI Development with PrimeNG – készítő: Dale Nguyen
https://learning.oreilly.com/library/view/next-level-ui-development/9781803249810/
[2] Scalable Application Development with NestJS – készítő: Pacifique Linjanja
https://learning.oreilly.com/library/view/scalable-application-development/9781835468609/
[3] Using SQLite – készítő: Jay A. Kreibich
https://learning.oreilly.com/library/view/using-sqlite/9781449394592/
[4] Modern Angular – készítő: Armen Vardanyan
https://learning.oreilly.com/library/view/modern-angular/9781633436923/