Debreceni Egyetem
Informatikai Kar

Mobil webalkalmazás fejlesztés

Témavezető:
Dr. Juhász István
egyetemi adjunktus

Készítette:
Máté Balázs
Programtervező Informatikus MSc

Debrecen
2011

Tartalomjegyzék

Köszönetnyilvánítás

Köszönettel tartozom Dr. Juhász István tanár úrnak, hogy észrevételeivel segítette szakdolgozatom megírását.

Bevezetés

Az okostelefon internet nélkül olyan, mint az autó benzin nélkül.

A mobiltelefonok egyre gyorsabban fejlődnek, egyre több célt szolgálnák és rendkívül széles körben elterjedtek. Óriási előnyük, hogy könnyen kezelhetőek, mindig elérhetőek, személyesek és folyamatos internet elérést biztosítanak, így segítségükkel számos mindennapi feladat gyorsan és egyszerűen elvégezhető. A mai okostelefonok népszerű alkalmazásainak szinte elengedhetetlen feltétele az internet, nem beszélve arról, hogy a legfontosabb alkalmazásuk maga a böngésző, amellyel a már megszokott weboldalainkat vagy azok mobiltelefonra optimalizált verzióját könnyedén, jó minőségben elérhetjük, így nagy figyelmet kell fordítanunk a mobilwebre, amely internetre kapcsolt eszközök, mobiltelefonok és tabletek böngészőalapú internetelérését biztosítja.

A mobiltelefonok hordozhatósága, folyamatos jelenléte és internethez való állandó kapcsolódása rendkívül jól beleillik a web jövőképében fontos szerepet játszó “ubiquitous web”, azaz a mindenütt jelenlévő web szemléletbe, hiszen manapság a mobiltelefonon az asztali számítógéphez hasonlóan szinte mindent elérünk kezdve az e-mailjeinktől, kapcsolati hálónkon keresztül a kedvenc internetes hírportálunkig, így egyre közelebb kerülve a write-once-publish-everywhere elképzeléshez.

Manapság a két legnépszerűbb mobil alkalmazástípus a webalkalmazás, amelyek a mobil telefon böngészőjében futnak, nem szükséges telepíteni őket, HTML, CSS és JavaScript technológiák segítségével készülnek és alkalmazásszerű érzést adnak, valamint a natív alkalmazások, amelyek valamilyen jól meghatározott platformra íródnak kihasználva az adott platform lehetősségeit és a mobiltelefonon telepítésre kerülnek. A két típus közötti különbség előre láthatólag egyre homályosabb lesz köszönhetően annak, hogy egyre több natív eszköz elérhetővé válik a böngészőből is, valamint a sebességben és felhasználói élményben való lemaradás is egyre inkább csökken.

Szakdolgozatomban a mobil webalkalmazásokat, valamint azok funkciógazdagságát biztosító különböző technológiákat és APIkat fogom bemutatni, amelyek segítségével egy látványos, új eszközöket használó példaalkalmazást készítek el.

Úgy gondolom, hogy e fejlesztési irány a jövőben egyre népszerűbbé válik platformfüggetlensége, valamint a gyors fejlesztés miatt. Ezek mellett fontos érv, hogy a mai mobilböngészők rendkívül gyorsak, szabvány-kompatibilisek, könnyed navigációt biztosítanak az érintőkijelző segítségével, valamint hogy már több olyan funkció is implementálásra került, amely számos asztali böngészőnél még nem elérhető. Fontos megjegyezni, hogy webalkalmazások esetén nincs szükség a különböző mobilplatformok mindegyikére külön alkalmazást lefejleszteni, így nagymértékben csökkenthetőek a kiadások és a piacra kerülésig eltelt idő.

Maga az alkalmazás egy menü és étterem keresőszolgáltatás kifejezetten mobilra optimalizált változata, amelynek fejlesztése során kitüntetett figyelmet szenteltem a mobiltelefon korlátainak figyelembevételére, valamint a mobilban rejlő óriási lehetőségek kiaknázására.

Mobil alkalmazástípusok

SMS

A legegyszerűbb mobil alkalmazás az SMS alkalmazás. Ha a cél a legszélesebb közönség elérése, akkor a SMS alkalmazások készítése jó megoldást biztosít, hiszen az alsókategóriás, régi telefonok is képesek szöveges üzenetek küldésére és fogadására. Ilyen alkalmazások az üzenetért cserébe küldött csengőhangok vagy valamilyen oldal prémium részének elérését biztosító titkos kód. Ide tartoznak a mobil parkolásos programok, valamint egyes országokban bolti fizetés is kiváltható SMS küldésével. A fizetési modell is rendkívül egyszerű, mivel az alap vagy emelt díjas SMS a szolgáltatónál lévő számlára íródik fel.

Előnyök:

- Minden telefonon elérhető
- Egyszerű
- Megfelelő értesítések küldésére

Hátrányok:

- Drága
- Csak szöveges élmény
- 160 karakteres limit

Mobil weboldalak

A mobil weboldalak az asztali verzió mobil készülékekre optimalizált verziója, amely általában egy erősen leegyszerűsített verziója a rendes verziónak. A navigáció és tagolás is egyszerűsített, hogy minél szélesebb körben elérhető legyen a tartalom, bár a megjelenítés az eszközök tulajdonságaitól nagymértékben függ.

Előnyök:

- Széles közönség
- Hasonló technikákat használ, mint a teljes verzió így könnyen elkészíthető

Hátrányok:

- Csökkentett, limitált élmény
- A telefonok eltérő tulajdonságai nehezítik a fejlesztést

Natív alkalmazások

A natív alkalmazások a legrégebbi és legelterjedtebb alkalmazások, amelyek valamilyen platformot céloznak meg. A mobil webalkalmazásokhoz képest sokkal több

eszköz-specifikus tulajdonságot érnek el, úgy, mint a kamera, gyorsulásmérő, fájlrendszer, így egyes esetekben a natív alkalmazásfejlesztés lehet az egyetlen járható út.

Fontos megjegyezni, hogy a mai legnépszerűbb platformok esetén a natív alkalmazásokhoz saját alkalmazásboltok üzemelnek, amelyek sok esetben az egyetlen módjai egy alkalmazás telepítésének a telefonra. A fejlesztőknek alkalmazásukat be kell nyújtani az alkalmazásbolt felügyelete felé, amely ezek után egy több hetes felülvizsgálati procedúrán megy keresztül, hogy kiszűrjék a hibás, vírusos, nem megfelelő tartalmat megjelenítő alkalmazásokat. Ez azt jelenti, hogy a változtatások, bug-fixek átvezetése huzamosabb ideig tart, mint a mobil webalkalmazások esetén, hiszen ott nincsen semmilyen harmadik fél, amely ellenőrzést végezne.

Előnyök:

- Leggazdagabb felhasználói élmény
- Legbővebb eszköztár
- Az alkalmazásbolton keresztül bevétel generálható

Hátrányok:

- Platformkötöttség
- Lassabb fejlesztés, kiadás, változtatás
- Harmadik fél általi felülvizsgálat szükségessége
- Magasabb indulási költségek

Mobil webalkalmazások

A mobil webalkalmazások olyan mobil alkalmazások, amelyek a mobil telefon böngészőjében futnak, nem szükséges telepítésük vagy lefordításuk, HTML, CSS és JavaScript technológiák segítségével készülnek, és alkalmazásszerű érzést adnak. Az alkalmazásszerű érzés azt jelenti, hogy az interakció sokkal gördülékenyebb, mint egy mobilra optimalizált weboldal esetén valamint AJAX használatából adódóan a folytonos oldalfrissítések is megszűnnek. A lehetőség mobil webalkalmazások fejlesztésére már régen megvolt, viszont a mobil böngészők gyenge renderelési képességei és a szabványok betartásának inkonzisztenciája miatt nem terjedt el.

Az átütő változást az iPhone bevezetése hozta el, amely rendkívül erős böngészővel rendelkezett, amely a nem mobilra optimalizált tartalmakat is különlegesen jól jelenítette meg. Az Apple App Store alkalmazásbolt megjelenése előtt a fejlesztők az akkor még iPhone OS nevezetű operációs rendszert csak mobil webalkalmazások készítésével tudták elérni, így egyre több magas minőségű mobil webalkalmazás jelent meg kifejezetten iPhone-ra, amelyet észrevéve a többi gyártó is jobb böngészőket rakott eszközeire. Jelentős előrelépést hoznak a HTML 5 részeként megjelenő különböző APIk, amelyek célja azon akadályok leküzdése, amelyek eddig a natív alkalmazásfejlesztés felé terelte a fejlesztőket, lévén, hogy a natív környezet sokkal gazdagabb eszköztárat biztosított. Pénzügyi szempontból fontos megjegyezni, hogy a mobil webalkalmazás boltok hiánya miatt az értékesítés sokkal körülményesebb, hiszen nincs egy pontosan meghatározott hely, ahol az elektronikus fizetés gördülékenyen és biztonságosan menne vagy ahol más felhasználók értékelései olvashatóak lennének. Sok esetben azonban a mobil webalkalmazás célja nem az, hogy közvetlen bevételt nyújtson, annál inkább, hogy a cég weben elérhető szolgáltatásai a mobilon is élvezhetőek legyenek. Ezen alkalmazásokra kiváló példa az ebay.com, flickr.com iOS platformra készített ingyenesen elérhető alkalmazásai. Kisebb cégek esetén azonban minden mobil platformra külön lefejleszteni az alkalmazást túl nagy anyagi és emberi erőforrásokat emésztene fel, így ebben az esetben egy mobil webalkalmazás platformfüggetlensége miatt tökéletes választás lehet.

Előnyök:

- HTML, CSS, JavaScript alapúak
- Egyszerűen telepíthető
- Széles réteg megcélzására alkalmas
- Sokkal jobb felhasználó élményt nyújt a mobil weboldalaknál
- Gyorsabb fejlesztés, gyors hibajavítás

Hátrányok:

- Eszközfragmentáció
- Natív eszközök használatának hiánya
- Szerényebb felhasználói élmény

Mobil böngészők

A teljes mobil ökoszisztéma egy rendkívül bonyolult rendszer, amelyben fontos szerepet játszanak a szolgáltatók, az eszközök, az azokon futó operációs rendszerek és sok más egyéb. A számunkra legfontosabb rész viszont az eszközökön lévő böngészők, amelyek többféleképpen osztályozhatóak.

Navigáció alapján a következő csoportokat hozhatjuk létre:

- Fókuszos navigáció
- Kurzoros navigáció
- Érintéses navigáció
- Multitouch navigáció

Fókuszos navigáció esetén a telefon iránygombjaival változtatható a linkeken, szövegdobozokon vagy gombokon elhelyezkedő fókusz, amelyet különböző háttér vagy keretszín jelölhet. Ezek manapság egyre ritkábbak még az alsó kategóriás telefonok között is.

Kurzoros navigáció esetén egy kurzor irányítható a billentyűzeten elhelyezett nyilak segítségével, kattintáskor pedig a legközelebbi fókuszálható objektumra ugrik a kurzor gyorsítva a navigációt. Leginkább az egérrel való navigációhoz hasonlítható.

Érintéses navigáció során a felhasználó az ujját vagy stylus ceruzát használ a navigációhoz.

Multitouch navigáció esetén a felhasználó több ujját is használhatja, amelyek segítségével különböző gesztusok fejezhetőek ki, így könnyítve a zoomolást vagy scrollozást.

Direkt és proxys böngészők

Megkülönböztethetünk böngészőket aszerint, hogy a tartalom közvetlenül a weboldal szerveréről jön-e vagy valamilyen közbeékelt proxyn keresztül. A proxy szerver feladatai a következők lehetnek:

- Tömörítés
- Titkosítás

- Gyorsítótárazás
- Nem mobil kompatibilis részek törlése a tartalomból

A WebKit böngészőmotor

A WebKit egy Apple által fejlesztett, nyíltforráskódú layout motor böngészők számára, amely HTML és CSS megjelenítésére és JavaScript futtatására szolgál. Kezdetben a Mac OS X-es Safari alapjául szolgált, ma viszont már megtalálható Windows és iOS platformokon is. Szerencsére a mobil világban szinte mindenki ezt a motort használja alapul így elvárható az, hogy a különböző eszközökön a tartalom megjelenítése hasonlóan történjen. A helyzet azonban nem ilyen rózsás, hiszen a különböző implementációk között sok eltérés van.

Előretelepített böngészők

Internet Explorer

Az Internet Explorer Mobile a Microsoft saját mobilböngészője, amelynek első verziója 1996-ban jelent meg a Windows CE 1.0 platformra. A Windows Mobile 6.5-ös verziójáig az IE4-re épült, majd a 6.5-ös verziónál IE6-ra. A Smartphone verzió esetén fókuszos, a Pocket PC verzió esetén pedig stylusos érintéses navigáció volt elérhető. A Windows Phone 7 bevezetésével multitouchos a navigálás valamint a motor az IE7 és IE8 alapjaira épül.

Mobile Safari

A Mobile Safari az iOS operációs rendszeren elérhető WebKit alapú böngésző, amely gyors és látványos böngészést tesz lehetővé. Csak érintéses és multitouch navigáció érhető el a fizikai billentyűzet hiánya miatt.

Nokia S40 család böngészője

Minden Nokia Series 40 készülék egy Nokia által készített, beépített böngészővel rendelkezik, amely a kezdetekben a Nokia saját megjelenítő rendszerét használta. A hatodik verziót követően a Nokia átállt egy WebKit alapú megoldásra, amely új felhasználói élményt hozott az alsó és középkategóriás telefonok piacára.

Symbian böngésző

A 2005 után megjelent Nokia S60-as telefonok sajátkészítésű, szintén WebKit alapú

böngészővel vannak felszerelve, amelyben többféle navigációs módszer is elérhető. Fontos megjegyezni, hogy ez a böngésző a legelterjedtebb, viszont nem a legtöbbet használt.

Android böngésző

Az Android operációs rendszer saját böngészője is WebKit alapú, amely a Mobile Safarihoz hasonlóan rengeteg fejlett funkciót támogat és rendkívül jó böngészési élményt nyújt.

WebOS böngésző

A Palmos webOS szintén WebKit alapú böngészővel van felszerelve, amely támogatja a legtöbb új webes technológiát.

BlackBerry böngésző

Minden Research In Motion (RIM) eszköz előretelepített böngészővel érhető el, amelyek legtöbbje fókuszos navigálást tesz lehetővé. Az 6.0-ás BlackBerry operációs rendszer óta a RIM készülékeken is WebKit alapú böngésző dolgozik.

Felhasználó által telepíthető böngészők

Egyes böngészők a felhasználó által telepíthetőek a vásárlás után vagy valamikor a szolgáltató vagy gyártó telepíti előre.

Opera Mobile

Az Opera Mobile kurzoros navigációs böngésző, amely egyes szolgáltatóknál és gyártóknál előre telepítésre kerül, valamint Windows Mobile és Symbian környezetben a felhasználó által letölthető. Fontos még megjegyezni, hogy az Opera Mini proxys böngésző, amely egyes esetekben akár 80%-kal is csökkentheti a letöltendő tartalom méretét.

Firefox for mobile

Android és Maemo eszközökre elérhető Gecko motort használó böngésző.

JQTouch

A jQTouch egy kifejezetten iOS platformhoz fejlesztett jQuery plug-in mobil webalkalmazások fejlesztéséhez. Nagy előnye, hogy rendkívül gyorsan készíthetőek HTML segítségével olyan alkalmazások, amelyek a natív alkalmazások kinézetéhez közel állnak. A jQTouch lehetővé teszi az oldalváltások közbeni animációk használatát és sok egyéb WebKit specifikus dolgot. A megfelelő konvenciók alkalmazásával a natív alkalmazásoknál megszokott felhasználói felületes minták alkalmazhatóak, amelyek segítik a felhasználókat a könnyed eligazodásban. A jQTouch segítségével készített alkalmazások használhatóak a böngészőben, a kezdő képernyőre elhelyezett alkalmazásként vagy valamilyen többplatformos megoldással, mint például a Rhodes vagy PhoneGap alkalmazások.

Bemutatás

A jQTouch használatához szükség van a forráskódként elérhető JavaScript és CSS fájlokra, valamint jQuery plugin révén a jQueryre. A megszokott kinézet eléréshez szükség van különböző strukturális minták betartására, amelyek a következőekben kerülnek bemutatásra.

Fontos a következő két alapfogalmat és az alapszabályokat tisztázni, mivel azok elengedhetetlenek a jQTouchos fejlesztéseknél:

- A képernyő az, amit a felhasználó lát oldalról oldalra. Tulajdonképpen a HTML body DIV gyermekei.
- Az alkalmazás az a HTML oldal, amely tartalmazza az összes képernyőt függetlenül attól, hogy az dinamikusan kerül betöltésre vagy sem.

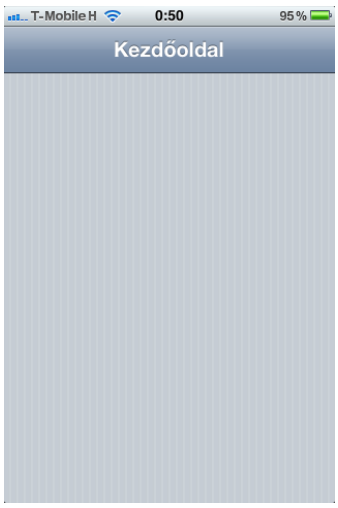
Alapszabályok:

- A felhasználó sosem hagyja el az adott HTML oldalt.
- A képernyők nem külön HTML oldalak, hanem az egyetlen HTML oldal bodyjának közvetlen DIV gyermekei
- ID-k használata nem javasolt a képernyőket kivéve

Az alkalmazás tehát pontosan egy HTML oldalból áll, amely hivatkozik a jQuery és jQTouch JavaScript könyvtárakra, a jQTouch CSSre és egy téma CSSre, amely az alkalmazás kinézetét határozza meg. A jQTouch CSS tartalmazza az konvenciókat felállító CSS osztályokat valamint az átmenetek és az átmenetekhez tartozó WebKit animációk definícióját. Ezen CSS fájl módosítására a fejlesztés folyamán alapvetően nincs szükség, ellenben a konvenciók miatt rengetegszer hivatkozunk rá. A téma tulajdonképpen egy könyvtár egy témát tartalmazó CSS fájllal és a hozzá szükséges képekkel. Maga a jQTouch 3 előre elkészített témával tölthető le, amelyek közül az egyik a natív iPhone UIKit interfészt szimulálja.

Az alkalmazás általában pár előre betöltött képernyőt már tartalmaz, amelyek a body közvetlen DIV gyermekei. Ha egyetlen képernyő sincs aktuálisnak kinevezve a „current” CSS class alkalmazásával, akkor alapértelmezetten a body első DIV gyermeke lesz az. Ezen képernyők közül mindig pontosan egy darab látszik és fontos, hogy a DIVek ID attribútuma meg legyen adva a képernyők közötti navigációk eléréséhez.

```
<!DOCTYPE html>
<html>
<head>
  <title>jQTouch példa</title>
  <link rel="stylesheet" type="text/css" href="jqtouch/jqtouch.css" />
  <link href="themes/apple/theme.css" rel="stylesheet" type="text/css" />
  <script src="jqtouch/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="jqtouch/jqtouch.js" type="text/javascript"></script>
  <script type="text/javascript">
    var jqt = $.jQTouch();
  </script>
</head>
<body>
  <div id="fooldal">
    <div class="toolbar">
      <h1>
        Kezdőoldal</h1>
    </div>
  </div>
</body>
</html>
```



Az képernyők közti váltásokkor előforduló animációk csak olyankor következnek be, ha a kívánt képernyő már a DOM-ban van. Ha belső képernyőre történik hivatkozás, azaz a hivatkozás href tagja valamilyen belső anchorra vonatkozik, mint például #fooldal, akkor a váltás további lépések nélkül megoldható. Ha a hivatkozás valamilyen külső képernyőre vonatkozik, akkor a jQTouch először egy AJAX hívás segítségével letölti a tartalmat, a letöltött DIV-et hozzáfűzi a bodyhoz és a current CSS osztállyal látja el, majd az eredeti hivatkozás href attribútumát átírja a dinamikusan most betöltött DIV azonosítójára.

Fontos megjegyezni, hogy az AJAX hívás eredményének egy HTML darabnak és nem HTML dokumentumnak kell lennie, hiszen egy HTML dokumentum nem szúrható a másikba. Ilyen esetben az alkalmazás nem fog működni, amelyet üres képernyő jelez.

Példa:

Eredetileg az alkalmazás:

```
<body>
  <div id="fooldal">
    <div class="toolbar">
      <h1>
```

```
            Kezdőoldal</h1>
        </div>
        <a href="/ettermek">Éttermek</a>
      </div>
    </body>
```

ettermek.html:

```
    <div>
        <ul>
          <li>Finom Falatok Étterem</li>
          <li>Belvárosi Étterem</li>
          <li>Külvárosi Étterem</li>
        </ul>
      </div>
```

Eredmény:

```
    <body>
      <div id="fooldal">
        <div class="toolbar">
          <h1>
            Kezdőoldal</h1>
        </div>
        <a href="#page-2">Éttermek</a>
      </div>
      <div id="page-2" class="current">
        <div>
          <ul>
            <li>Finom Falatok Étterem</li>
            <li>Belvárosi Étterem</li>
            <li>Külvárosi Étterem</li>
          </ul>
        </div>
      </div>
    </body>
```

Olyan linkek esetén, amikor az alkalmazás elhagyása a cél és nem az új képernyő dinamikus betöltése a hivatkozáshoz target="_webapp" -t kell hozzáadni.

Példa:

Google

A képernyők közötti visszalépés a „back” és „cancel” osztályokkal oldható meg. Az anchor tageken elhelyezett „back” vagy „cancel” osztály esetén a jQTouch a linkeket

gombokká alakítja és a képernyő bal szélére rendezi, majd megnyomásukkor az élőző képernyő kerül aktív állapotba.

```
<div id="page-2" class="current">
    <div class="toolbar">
      <h1>
        Éttermek</h1>
      <a class="back" href="#">Vissza</a>
    </div>
    <div>
      <ul>
        <li>Finom Falatok Étterem</li>
      </ul>
    </div>
  </div>
```

A jQTouch nem működik együtt a böngésző vissza és előre gombjaival, így azok alkalmazása kerülendő, helyettük a „back” és „cancel” osztályok alkalmazása javasolt.

Gombok alkalmazásához nem <input type=”button” /> tagre lesz szükségünk, hanem a linkeket fogjuk „button” és „leftButton” osztályokkal ellátni attól függően, hogy jobbra vagy balra akarjuk azokat igazítani.

- `<a href="/ettermek" class="button">Éttermek</a>`
- `<a href="/ettermek" class="button leftButton">Éttermek</a>`

Fontos, sokszor megjelenő formázási módszer a listák alkalmazása, amely kifejezetten hasonlít az iPhoneon már megszokott tagoláshoz, és amely az ul tagen elhelyezett edgetoedge, plastic, metal osztályokkal érhető el.

Inicializálási paraméterek

A jQTouch inicializálása a $.jQTouch() JavaScript függvényhívással történik, amely eredményeként kapott objektumon különböző jQTouchos függvényeket tudunk meghívni később, így érdemes azt egy változóba lementeni a jQTouch=$.jQTouch() kód segítségével. Jegyezzük meg, hogy ebben az esetben a $ a jQuery JavaScript könyvtár. A jQTouch beállításai a következő táblázatban kerülnek összefoglalásra:

Érték	Alapértelmezett érték	Jelentés
addGlossToIcon	true	glossy effekt alkalmazása az indítási ikonon iPhone esetén
backSelector	.back, .cancel, .goback'	a visszalépést jelző osztályok
cacheGETRequests	true	a GET kérések gyorsítótárazása
cubeSelector	.cube'	cube animációt jelző osztály
dissolveSelector	.dissolve'	dissolve animációt jelző osztály
fadeSelector	.fade'	fade animációt jelző osztály
fixedViewPort	true	biztosítja az oldal alkalmazásjellegét azzal, hogy a felhasználó nem zoomolhat
formSelector	form'	beállítja, hogy mely formok kerülnek küldésre automatikusan AJAX segítségével
fullScreen	true	az oldal teljes képernyős alkalmazás lesz iPhoneról indítva
icon	false	iPhone home screen ikon beállítása
preloadImages	false	azon képek megadása, amelyek az oldal betöltődése előtt letöltődnek
startupScreen	null	kezdőképernyő megadása
useAnimations	true	animációk használata

Geolokáció

Egyik nagy előnye a mai okostelefonoknak, hogy igen sok rendelkezik beépített GPS-szel, amellyel az eszköz pontos helye meghatározható. A későbbiekben látni fogjuk, hogy GPS segítsége nélkül is szerezhetünk egészen pontos helyinformációt telefonunkról.

Egyre több alkalmazás használja fel a helyet, mint kontextust ezzel is növelve a különböző keresések vagy hirdetések relevanciáját, valamint az alapvető felhasználói élményt. Képzeljük el, hogy utazásunk során milyen nagy segítséget jelenthet egy olyan elektronikus útikönyv, amely mindig azokat a látványosságokat mutatja be aktuálisan, amely előtt állunk vagy a legközelebb vannak.

Mindegyik helymeghatározási módszernél előjön a pontosság témaköre, amelyet legtöbbször méterben vagy kilométerben adunk meg, azonban egyes esetekben városi vagy országos pontosságról beszélünk. Helymeghatározás esetén mindenki először szabadban lévő pontosságra gondol, azonban különböző technikák segítségével zárt térben lévő helymeghatározás is lehetséges, amely például repülőterek vagy bevásárlóközpontok esetében segítheti a tájékozódást.

Kliens oldali helymeghatározás

GPS

A legismertebb pozíció meghatározási rendszer valószínűleg a Global Positioning System (GPS), amelyet az amerikai kormány fejlesztett ki különböző eszközök helyének meghatározására Föld körül keringő műholdak segítségével. Egyre több telefon rendelkezik beépített GPS vevővel, amely képes adatokat lekérni a műholdaktól. A telefonnak szabad ég alatt le lennie, pontossága 2 és 100 m között van, valamint a helymeghatározáshoz 5 másodperc és 5 perc közötti időre van szükség.

A-GPS

Az Assisted GPS egy szoftveralapú rendszer olyan eszközök számára, amelyek valamilyen mobil szolgáltató hálózatához kapcsolódnak. Többféleképpen gyorsíthatja meg a helymeghatározást, például jobb műholdas jel elérésének segítésével vagy pontatlanabb adat biztosításával addig, amíg a GPS nem kapcsolódott megfelelően a műholdakhoz.

Cellainformáció

A szolgáltató hálózatának tornyait segítségül véve háromszögelés módszerével meghatározható az eszköz pozíciója. Minél sűrűbben helyezkednek el a tornyok, annál pontosabb helymeghatározásra van lehetőség.

WiFi Pozíciós rendszer

Ezen rendszer azon router pozíciója alapján határozza meg a mi a helyzetünket, amelyhez WiFi-vel csatlakoztunk. Ezen módszerhez szükség van egy olyan meglévő adatbázisra, amely tartalmazza a routerek pontos helyét. Ilyen adatbázissal többek között a Google is rendelkezik, amelyet például a Firefox böngésző használ helymeghatározáshoz.

Szerver oldali helymeghatározás

Szerver oldal esetén a HTTP kérés fejléce alapján van lehetőségünk helymeghatározásra.

IP cím

Léteznek különböző IP címeket tartalmazó listák, amely segítségével országos vagy városi pontosság érhető el. Fontos megjegyezni, hogy a kapott IP cím sokszor a mobilszolgáltató IP címe vagy proxys böngészők esetén a proxy szerver címe, így megeshet, hogy igen pontatlan eredményt érünk el.

Nyelv

Ha a felhasználó az eszközét megfelelően állította be, akkor a HTTP kérések fejléce tartalmazza a felhasználó által használt/preferált nyelvet, amely segítségével országos pontosság érhető el, amely sok esetben elég lehet.

Helymeghatározás böngészőből

W3C Geolocation API

A World Wide Web Consortium egy egységes JavaScript API kidolgozásán dolgozik, amely segítségével a felhasználó helye meghatározható. Ez az API még draft szakaszban áll, azonban számos böngésző köztük számos mobilböngésző már implementálta. Ezen API az előző részben bemutatott technológiákon alapul úgy, hogy a böngésző választhatja ki, hogy pontosan melyiket is használja a helymeghatározáshoz.

A helymeghatározás aszinkron módon történik, mivel a pozíció meghatározása akár percekbe is telhet, valamint a felhasználó explicit engedélye szükséges a művelet elvégzéséhez.

A pozíció meghatározása

```
 [NoInterfaceObject]
 interface NavigatorGeolocation {
   readonly attribute Geolocation geolocation;
 };

 Navigator implements NavigatorGeolocation;

 [NoInterfaceObject]
 interface Geolocation {
   void getCurrentPosition(in PositionCallback successCallback,
                           in optional PositionErrorCallback
errorCallback,
                           in optional PositionOptions options);

   long watchPosition(in PositionCallback successCallback,
                      in optional PositionErrorCallback errorCallback,
                      in optional PositionOptions options);

   void clearWatch(in long watchId);
 };

 [Callback=FunctionOnly, NoInterfaceObject]
 interface PositionCallback {
   void handleEvent(in Position position);
 };

 [Callback=FunctionOnly, NoInterfaceObject]
 interface PositionErrorCallback {
   void handleEvent(in PositionError error);
 };

[Callback, NoInterfaceObject]
  interface PositionOptions {
    attribute boolean enableHighAccuracy;
    attribute long timeout;
    attribute long maximumAge;
  };

[NoInterfaceObject]
  interface Position {
    readonly attribute Coordinates coords;
    readonly attribute DOMTimeStamp timestamp;
  };

[NoInterfaceObject]
  interface Coordinates {
    readonly attribute double latitude;
    readonly attribute double longitude;
    readonly attribute double? altitude;
```

```
    readonly attribute double accuracy;
    readonly attribute double? altitudeAccuracy;
    readonly attribute double? heading;
    readonly attribute double? speed;
  };
```

A geolocation objektum, amely az egész helymeghatározásért felel, a navigator objektum egy csak olvasható tagjaként érthető el. A pozíció a getCurrentPosition függvény segítségével határozható meg, amelynek első két paramétere sikeres és sikertelen helymeghatározás esetén hívandó callback függvény, míg a harmadik paraméter a beállításokat tartalmazza. A függvény a hívás után egyből visszatér, majd aszinkron módon próbálja meg az eszköz jelenlegi helyzetét meghatározni. Ha a próbálkozás sikeres volt, akkor az első paraméterként megadott successCallback függvény hívódik meg egy Position objektummal, amely a jelenlegi pozíciót tartalmazza. A Position objektum tartalmazza a helymeghatározás időbélyegét, a koordinátákat, a pontosságot és opcionálisan a magasságot és a magasság pontosságát. Sikertelenség esetén az errorCallback függvény kerül meghívásra egy új PositionError objektummal, amely a hiba kódját és szövegét tartalmazza. A hibakódok a következőek lehetnek:

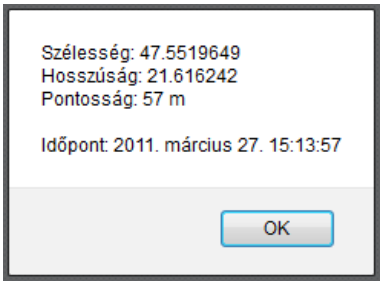
Hiba konstans	Leírás
UNKNOWN_ERROR	A helyzet nem meghatározható
PERMISSION_DENIED	A felhasználó nem adott jogot a helymeghatározáshoz
POSITION_UNAVAILABLE	A helyzet nem meghatározható a helyzeti adatok szolgáltató hibája miatt
TIMEOUT	Időtúllépés

Beállítható paraméterként megadható egy időkorlát (timeout), egy kérés arra vonatkozóan, hogy az eszköz a legpontosabb eredményt szeretné-e megkapni (enableHighAccuracy), valamint egy maximális időintervallum, amin belül még elfogadható a gyorsítótárazott pozíció (maximumAge).

Példa:

```
<script type="text/javascript">
```

```
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(sikeresGeo, errorGeo,
{enableHighAccuracy:true, timeout:10000, maximumAge:30000});
        }
        else {
            alert('A helymeghatározás nem elérhető');
        }
        function sikeresGeo(position) {
            alert('Szélesség: ' + position.coords.latitude + "\n" +
                  'Hosszúság: ' + position.coords.longitude + "\n" +
                  'Pontosság: ' + position.coords.accuracy + " m \n\n" +
                  'Időpont: ' + new Date(position.timestamp).toLocaleString());
        }
        function errorGeo(error) {
            switch (error.code) {
                case error.PERMISSION_DENIED:
                    alert('A helymeghatározás nem engedélyezett');
                    break;
                case error.POSITION_UNAVAILABLE:
                    alert('A helymeghatározás sikertelen');
                    break;
                case error.TIMEOUT:
                    alert('A helymeghatározás időtúllépés miatt sikertelen');
                    break;
                case error.UNKNOWN_ERROR:
                    alert('A helymeghatározás ismeretlen hiba miatt sikertelen');
                    break;
            }
        }
    </script>
```



Egy másik lehetőség a W3C API használatára a felhasználó helyének követése. Ilyenkor a felhasználó helyének változásáról értesítést kapunk, amely könnyedén lekezelhető és felhasználható. Ezen funkció használatához a navigator.geolocation watchPostion függvényére van szükségünk, amelynek az előzőekhez hasonlóan megadható két callback függvény és a beállításokat tartalmazó objektum. A függvényhívás visszatérési értéke egy watchID-t ad vissza, amelyet a clearWatch függvénynek megadva leállítható a követés.

Google Gears

A Google Gears egy böngésző plugin, amely egyes eszközökön előre telepítésre kerül, míg másokra opcionálisan letölthető. A W3C API sztenderddé válásáig egy hézagpótló szerepet tölt be, utána viszont idejemúlttá válik. Abból a szempontból jól jöhet fejlesztéseknél, hogy a Windows Mobile platform Internet Exploreréhez opcionálisan letölthető, valamint az Android 1.5-ös operációs rendszeren előre telepítésre került. Használatához először be kell tölteni a Gears API-t, majd a W3C API-hoz hasonlóan kérhető le az aktuális pozíció.

A W3C API-ban megismert három paraméteren kívül a Gears API tartalmaz még három paramétert. A gearsRequestAddress segítségével fordított geokódolás oldható meg, amely segítségével nemcsak a koordinátákat kaphatjuk meg, hanem a koordinátákhoz tartozó címet is. A gearsAddressLanguage paraméterrel az RFC 3066 szabvány szerinti nyelvet adhatunk meg a címek megjelenítéséhez, a gearsLocationProviderURL pedig a geokódolást végző oldal címének megadására szolgál, amely alapesetben a Google.

Ha a gearsRequestAddress opciót igazra állítottuk, akkor a position objektumnak lesz egy gearsAddress attribútuma a következő adattagokkal:

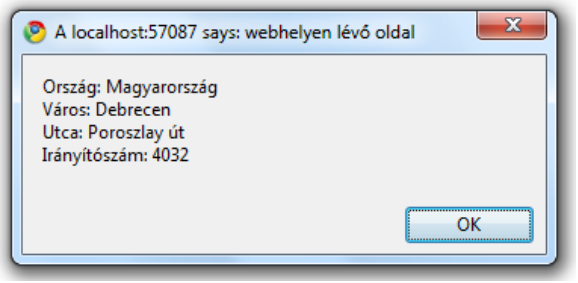
- street
- streetNumber
- premises
- city
- region
- country
- countryCode
- postalCode

A Gears API a W3C APIban lévő POSITION_UNAVAILABLE és TIMEOUT hibakódokat támogatja csak, valamint elérhető a watchPosition függvény a getCurrentPostion függvényhez hasonló paraméterekkel.

Példa:

```
var geolocation = google.gears.factory.create("beta.geolocation");
        geolocation.getCurrentPosition(sikeresGeo, errorGeo, { enableHighAccuracy:
true, gearsRequestAddress: true, gearsAddressLanguage: "hu-HU" });

        function sikeresGeo(position) {
            if (position.gearsAddress) {
                var address = position.gearsAddress;
                alert('Ország: ' + address.country + "\n" +
                  'Város: ' + address.city + "\n" +
                  'Utca: ' + address.street + "\n" +
                  'Irányítószám: ' + address.postalCode);
            }
        }
        function errorGeo(error) {
            switch (error.code) {
                case error.POSITION_UNAVAILABLE:
                    alert('A helymeghatározás sikertelen');
                    break;
                case error.TIMEOUT:
                    alert('A helymeghatározás időtúllépés miatt sikertelen');
                    break;
            }
        }
```



Multiplatform Geolocation API

A geo.js egy nyílt forráskódú MIT-licenszes JavaScript könyvtár, amely meglévő APIk közötti különbségek elsimítására jött létre. Használatához két script taget kell oldalunkra beszúrni, amelyből az egyik a Google Gears inicializáló szkript, a másik pedig maga a geo.js. Az API létrehoz egy geo_positon_js nevű globális változót, amely jelöli, hogy az eszköz képes-e helymeghatározásra. Hasonlóan a W3C APIhoz itt is aszinkron módon, callback függvények segítésével kaphatjuk meg a position objektumot vagy szerezhetünk tudomást a sikertelenségről.

```
if (geo_position_js.init()){
geo_position_js.getCurrentPosition(sikeresGeo, errorGeo);
} else{
```

```
alert("Geolokáció nem  érhető el");
}
```

Nagy pozitívuma a geo.js keretrendszernek, hogy fejlesztés során lehetőségünk van a felhasználó mozgásának szimulációjára anélkül, hogy az eszköz tényleg elmozdulna.

A W3C APIval ellentétben a geo.js esetén nincs lehetőség a helyváltozás figyelésére, így azt setTimeout/setInterval és getCurrentPosition segítségével kell magunknak implementálni.

Tárolás

A natív alkalmazásokkal szemben nagy hátránnyal rendelkeztek a webalkalmazások abban a tekintetben, hogy nehezen tudtak a kliens oldalon adatokat perzisztensen tárolni, hiszen nem volt lehetőség a fájlrendszert elérni vagy lokális adatbázist használni.

A leginkább elterjedt kliens oldali tárolási módszer a sütik (cookie) használata. Azonban ez a módszer több sebből is vérzik a modern web alkalmazások használata során.

- A sütik minden egyes http kérés során elküldésre kerülnek, így feleslegesen növelik az adatforgalmat és feleslegesen lassítják a kommunikációt.
- HTTP használta esetén a sütik titkosítás nélkül utaznak oda-vissza
- 4 KB-os méretkorlát

Olyan tárolási módszerre van szükség, amely a kliens oldalon történik, perzisztens, nincs vagy nagy a méretkorlát és ahol az adatok nem kerülnek elküldésre a szerver felé. Ez nem azt jelenti, hogy a sütik használata, mint megoldás nem jó, azonban számos esetben nem megfelelő.

Web Storage

A Web Storage a HTML 5 specifikáció részeként kulcs/érték párok kliens oldali tárolására ad lehetőséget. Két fajtája létezik, a localStorage, amely hosszú távú tárolásra szolgál, és a sessionStorage, amely munkamenet szintű tárolásra lett kitalálva. Ezen megoldással az adatok megmaradnak, ha a felhasználó elnavigál az oldalról, bezárja a böngészőt vagy

kikapcsolja az eszközét. A sütikkel ellentétben az ily módon tárolt adatok maguktól sosem kerülnek elküldésre a szerver felé.

```
interface Storage {
 readonly attribute unsigned long length;
 DOMString key(in unsigned long index);
 getter any  getItem(in DOMString key);
 setter creator void setItem(in DOMString key, in any value);
 deleter void removeItem(in DOMString key);
 void clear();
};
```

A kulcs mindig sztring, míg az adat lehet bármilyen JavaScript által támogatott típus. A tárolás azonban sztringként történik, így a visszanyerésnél szükség lehet a parseInt(), parseFloat() függvények használatára. A setItem függvény meglévő kulcs esetén az előző értéket figyelmeztetés nélkül felülírja, valamint a getItem nem létező kulcs esetén null értéket ad vissza ahelyett, hogy hibát dobna. A localStorage nem csak a getItem és setItem segítségével érhető el, hanem asszociatív tömbként is kezelhető, lekérdezhető a benne tárolt elemek száma a length tulajdonság segítségével, valamint az egyes elemek integer indexekkel is elérhetők a key() függvény segítségével.

A tárolóban lévő elemek változása követhető a windows objektum storage eseményével, amely meghívódik minden egyes alkalommal, amikor a setItem(), removeItem(), clear() valamilyen tényleges módosítást végez. Tehát ha például a removeItem segítségével olyan értékpárt akarunk törölni, amely nem is volt jelen a tárolóban, akkor a storage event nem fog meghívódni. A StorageEvent objektum adattagjai a key, oldValue, newValue és az url, amely a változtatást végrehajtó oldal címe. Fontos megjegyezni, hogy a változtatás nem lehet érvényteleníteni, így csak arra szolgál, hogy tudomást szerezzünk arról, hogy mi történt.

Példa:

```
function tamogat_web_storage() {
            return ('localStorage' in window) && window['localStorage'] !== null;
        }
        var storageTamogatott = tamogat_web_storage();
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(sikeresGeo, {
enableHighAccuracy: true });
        }
        function sikeresGeo(position) {
```

```
        if (storageTamogatott) {
            var pos = { latitude: position.coords.latitude,
                        longitude: position.coords.longitude,
                        altitude: position.coords.altitude,
                        accuracy: position.coords.accuracy,
                        timestamp: new Date(position.timestamp)
                      };
            var pozok = localStorage.getItem('posok');
            if (pozok != null) {
                pozok = JSON.parse(pozok);
                pozok.push(pos);
                localStorage.setItem('posok', JSON.stringify(pozok));
            }
            else {
                pozok = new Array();
                pozok.push(pos);
                localStorage.setItem('posok', JSON.stringify(pozok));
            }
        }
    }
```

Web SQL Database

A Web SQL Database API, mint azt a neve is sugallja, kliens oldali, JavaScriptben elérhető relációs adatbázist nyújt. Ezen API segítésével könnyedén hozhatunk létre alkalmazásunkhoz adatbázisokat, amelyből tetszés szerint kérdezhetünk le, szúrhatunk be új adatokat vagy tetszőleges időközönként szinkronizálhatjuk azt a szerver oldali adatbázisunk számunkra releváns részével.

```
[Supplemental, NoInterfaceObject]
interface WindowDatabase {
  Database openDatabase(in DOMString name, in DOMString version, in
DOMString displayName, in unsigned long estimatedSize, in optional
DatabaseCallback creationCallback);
};
Window implements WindowDatabase;
[Supplemental, NoInterfaceObject]
interface WorkerUtilsDatabase {
  Database openDatabase(in DOMString name, in DOMString version, in
DOMString displayName, in unsigned long estimatedSize, in optional
DatabaseCallback creationCallback);
  DatabaseSync openDatabaseSync(in DOMString name, in DOMString
version, in DOMString displayName, in unsigned long estimatedSize,
in optional DatabaseCallback creationCallback);
};
WorkerUtils implements WorkerUtilsDatabase;
[Callback=FunctionOnly, NoInterfaceObject]
interface DatabaseCallback {
  void handleEvent(in Database database);
};
```

Új adatbázis létrehozására az openDatabase() és openDatabaseSync() függvények szolgálnak, amelyek az adatbázis nevét, amely lehet üres sztring is, verzióját, kiíratási nevét (display name), becsült méretét és egy opcionális callback függvényt várnak. A callback függvény a changeVersion() függvény meghívásra hivatott, amellyel az adatbázis verziója állítható az előbb megadott paraméter figyelembe vétele nélkül. Ha callback függvényt nem adunk át, akkor a paraméterként megadott verzió kerül beállításra.

SQL utasítások végrehajtásához szükségünk van tranzakciókra. Ha a tranzakció során csak olvasásra kerül sor, akkor az adatbázis objektum readTransaction(), amúgy a transaction() aszinkron metódus meghívásával kerülünk egy lépéssel közelebb SQL utasításaink lefuttatásához. Ebben az esetben a callback függvény egyetlen paramétere az előbb kért SqlTransaction típusú objektum, ahogy az alábbi specifikációból is látható.

```
interface Database {
  void transaction(in SQLTransactionCallback callback, in optional
SQLTransactionErrorCallback errorCallback, in optional
SQLVoidCallback successCallback);
  void readTransaction(in SQLTransactionCallback callback, in
optional SQLTransactionErrorCallback errorCallback, in optional
SQLVoidCallback successCallback);

  readonly attribute DOMString version;
  void changeVersion(in DOMString oldVersion, in DOMString
newVersion, in optional SQLTransactionCallback callback, in
optional SQLTransactionErrorCallback errorCallback, in optional
SQLVoidCallback successCallback);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface SQLVoidCallback {
  void handleEvent();
};

[Callback=FunctionOnly, NoInterfaceObject]
interface SQLTransactionCallback {
  void handleEvent(in SQLTransaction transaction);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface SQLTransactionErrorCallback {
  void handleEvent(in SQLError error);
};
```

A tranzakció executeSql() metódusával futtathatóak SQL utasítások a már megszokott aszinkron stílusban, ahol a callback függvény két paramétere az aktuális tranzakció és az eredményhalmaz, amelyből kiderül a módosított sorok száma vagy az új sor elsődleges kulcsa vagy lekérdezés esetén, annak eredménye.

```
typedef sequence<any> ObjectArray;

interface SQLTransaction {
  void executeSql(in DOMString sqlStatement, in optional
ObjectArray arguments, in optional SQLStatementCallback callback,
in optional SQLStatementErrorCallback errorCallback);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface SQLStatementCallback {
  void handleEvent(in SQLTransaction transaction, in SQLResultSet
resultSet);
};
[Callback=FunctionOnly, NoInterfaceObject]
interface SQLStatementErrorCallback {
  boolean handleEvent(in SQLTransaction transaction, in SQLError
error);
};
```

Példa:

```
<script type="text/javascript">
        var db = window.openDatabase("geo", "1.0", "geo", 10);

        db.transaction(function (t) {
            t.executeSql("CREATE TABLE IF NOT EXISTS coords(" +
                                "id integer primary key autoincrement," +
                                "latitude float," +
                                "longitude float," +
                                "altitude float," +
                                "accuracy float," +
                                "timestamp timestamp)", [], null, error);
        });

        function beszuras(pos) {
            db.transaction(function (t) {
                t.executeSql("INSERT INTO coords
(latitude,longitude,altitude,accuracy,timestamp) VALUES (?,?,?,?,?)",
                                new Array(pos.latitude, pos.longitude,
pos.altitude, pos.accuracy, pos.timestamp), null, error);
            });
        }
        function darab_lekerdezes() {
            db.transaction(function (t) {
```

```
                t.executeSql("SELECT COUNT(*) as darab FROM coords", [], function
(transaction, data) {
                    alert(data.rows.item(0).darab);
                });

            });
        }
        function error(transaction, error) {
            alert(error.message);
        }

        if (navigator.geolocation)
            navigator.geolocation.getCurrentPosition(sikeresGeo);

        function sikeresGeo(position) {
            var pos = { latitude: position.coords.latitude,
                longitude: position.coords.longitude,
                altitude: position.coords.altitude,
                accuracy: position.coords.accuracy,
                timestamp: position.timestamp
            };
            beszuras(pos);
        }

        darab_lekerdezes();

    </script>
```

Gears Storage

A Google Gears az előbb bemutatott Web SQL Database APIhoz hasonlóan relációs adatok kliens oldali tárolását teszi lehetővé. Fontos megjegyezni, hogy az Android platformon támogatott és az előzőtől eltérően szinkron működésű.

Példa:

```
<script src="js/gears_init.js" type="text/javascript"></script>
    <script type="text/javascript">
        var db = google.gears.factory.create('beta.database');
        db.open('geo');
        db.execute("CREATE TABLE IF NOT EXISTS coords(" +
                                "id integer primary key autoincrement," +
                                "latitude float," +
                                "longitude float," +
                                "altitude float," +
                                "accuracy float," +
                                "timestamp timestamp)");

        function lekerdezes() {
            var rs = db.execute('SELECT * FROM coords');
            while (rs.isValidRow()) {
                var id = rs.field(0);
                var name = rs.field(1);
                var pos = { id: rs.field(0),
```

```
                latitude: rs.field(1),
                longitude: rs.field(2),
                altitude: rs.field(3),
                accuracy: rs.field(4),
                timestamp: rs.field(5)
            };
            rs.next();
        }
        rs.close();
    }
</script>
```

Offline webalkalmazások

Nem csak az a fontos, hogy az alkalmazás által kliens oldalon használt adatok a következő bejelentkezéskor is elérhetőek legyenek, hanem az is, hogy az alkalmazás internetkapcsolat nélkül is működjön úgynevezett offline módban. Maga a kifejezés, hogy offline webalkalmazás ellentmondásosnak tűnhet, hiszen a weboldalak letöltésre majd renderelésre kerülnek, a letöltéshez viszont hálózatra van szükség, amely offline esetben pont nincs. Sok esetben azonban szükség lehet rá. Gondoljunk arra, hogy külföldön használva a már megszokott alkalmazást igen magas extra költségek léphetnének fel vagy olyan területeken, ahol nincs internet kapcsolat (repülőgép) egyszerűen el sem tudna indulni az alkalmazás. Ezen probléma megoldására született a HTML 5 részeként az Offline Application Caching API.

A trükk az, hogy amikor van kapcsolat az oldalak, JavaScript fájlok, képek vagy egyéb erőforrások letöltésre kerülnek, és ha megszűnik a kapcsolat, akkor ezen lokális másolatok veszik át az online tartalom helyét.

Ennek a működésnek a menedzseléséhez egy manifest fájlra van szükségünk a webszerveren, amely tulajdonképpen az erőforrások listáját tartalmazó szöveges fájl. Webalkalmazásunk főoldalának pedig hivatkoznia kell erre a manifest fájlra, hogy ez alapján a böngésző letöltse és letárolja a szükséges fájlokat, valamint a később kívánt frissítéseket megtegye.

A DOM-ban elérhetőek események és egy flag arra vonatkozóan, hogy offline vagy online módban működik-e az adott pillanatban az alkalmazás, így a fejlesztő megteheti,

hogy amíg nincs kapcsolat addig a lokális adatokkal dolgozik és az új adatokat kliens oldalon tárolja, majd amikor újra van kapcsolat a változtatásokat szinkronizálja a szerverrel.

Cache manifest

Az offline webalkalmazások alapja a cache manifest fájl, amely MIME típusának mindenképp text/cache-manifest-nek kell lennie, hogy a böngésző megfelelően tudja elvégezni a cacheelést, valamint az alkalmazás minden oldalának hivatkozni kell rá az alábbi módon:

```
<!DOCTYPE html>
<html manifest="/cache.manifest">
```

A cache manifest fájl első sora mindig „CACHE MANIFEST” és a fájl három különböző részre osztható. Az explicit rész, amelynek CACHE fejléce akár el is hagyható, tartalmazza a kliensen tárolandó fájlok listáját, a network részben azon erőforrások kerülnek, amelyek sosem kerülnek cachelésre és nem érthetőek el offline módban, míg a fallback részben megadott erőforrások letárolásra kerülnek és az offline módban nem elérhető erőforrások helyett alkalmazhatóak. A fallback részben megadható például egy olyan állomány, amely tájékoztatja a felhasználót, hogy olyan erőforrásra hivatkozott, amely offline módban nem érhető el. Fontos megjegyezni, hogy azon html oldalak, amelyek az adott cache manifest fájlra hivatkoznak, automatikusan letárolásra kerülnek így azokat nem szükséges a manifest fájlban felsorolni.

Események

Ha a böngésző egy olyan oldalt látogat meg, amely hivatkozik egy cache manifest fájlra, akkor a window.applicationCache objektumon a következő események hívódnak meg.

1. Ha a böngésző észreveszi, hogy az oldal HTML elemének van manifest attribútuma, akkor a checking esemény következik be függetlenül attól, hogy ez az első látogatás vagy sem.
2. Ha a böngésző még nem járt ezen az oldalon, akkor a downloading esemény következik be, majd periódikusan a progress esemény, amiből kiderül, hogy hány

erőforrás került letöltésre és még hány várakozik. Ha minden megadott erőforrás letöltődött a cached esemény hívódik meg.

3. Ha ez a cache manifest fájl egyszer már letöltésre került, akár a most meglátogatott oldal miatt, akár másik miatt, akkor a böngésző megvizsgálja, hogy történt-e változtatás a manifest fájlban.
    a. Ha nem történt változás a noupdate esemény sül el
    b. Ha történt változtatás, akkor a downloading és progress események sülnek el hasonlóan az előző ponthoz, majd a letöltések befejeztével az updateready esemény következik be. A régi verzió lecserélése az újra a windows.applicationCache.swapCache() függvény meghívásával tehető meg.
4. Ha a folyamat során valamilyen hiba lép fel, például azért mert a manifest fájl vagy valamelyik erőforrás nem volt letölthető az error esemény váltódik ki.

Megjegyzendő, hogy a folyamat nincs tekintettel arra, ha a szerveren valamelyik erőforrás megváltozott, csak arra, ha a cache manifest, így a változtatásokat a manifest fájlra is át kell vezetni. Ennek a legegyszerűbb módja, ha a manifest verzióját egy megjegyzésben letároljuk, és minden változtatásnál és azt eggyel növeljük.

Példa:

```
CACHE MANIFEST
# verzio 1
CACHE
fooldal.html
terkep.html
tavasz.css
logo.jpg
NETWORK
ettermek.cshtml
menuk.cshtml
```

HTML 5 formok

A HTML 5-ös formok számos olyan „kényelmi” szolgáltatást biztosítanak, amelyek régebben csak JavaScript használatával vagy még azzal sem volt elérhetők. Egyre több modern böngésző támogatja ezeket az újításokat, amelyek pedig nem azok szimplán nem vesznek róluk tudomást, így problémát még az Internet Explorer 6-os verziójában sem okozhatnak.

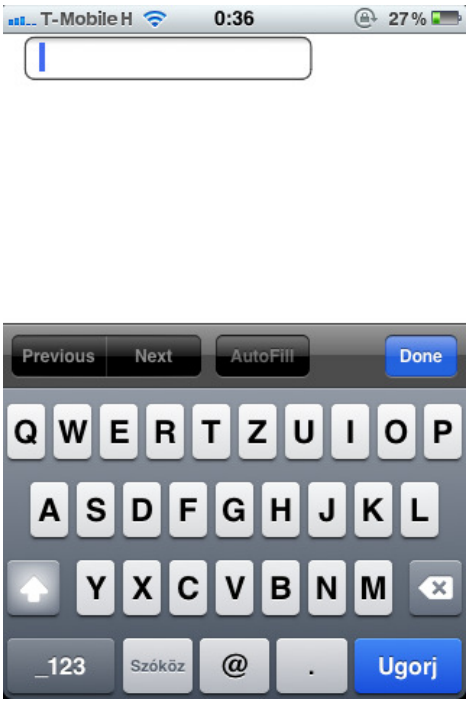
Az első ilyen újítás a placeholder szöveg megadása text input esetén, amely a szöveges mezőben jelenik meg, ameddig a mező üres vagy nincs rajta fókusz. Segítségével a mobil böngészők esetén értékes hely spórolható.

```
<form>
   <input type="text" name="e" placeholder="Étterem neve" />
   <input type="submit" value="Keresés" />
</form>
```

Input mezők esetén megadható az új autofocus attribútum, amely segítségével a form betöltődése után a fókusz egyből a kiválasztott mezőre kerül. Ez valamely esetekben zavaró, így egyes böngészőkben letiltható.

```
<form>
    <input type="text" name="e" autofocus />
    <input type="submit" value="Keresés" />
</form>
```

Az új input típusok bevezetése a mobil böngészők esetén nagyban megkönnyítheti az adatbevitelt. Ezen új típusok alatt az email, url, number, date, time típusokat értjük, amelyek esetén több mobilböngésző is speciális virtuális billentyűzetet tesz elérhetővé ezzel gyorsítva az adatbevitelt. Ez azt jelenti, hogy például az iOS eszközökön email típus esetén, a billentyűzet alján megjelenik a @ szimbólum, így az két kattintással hamarabb írható be, mint text típus esetén. Azon böngészők, amelyek e típusokat nem ismerek fel, azok úgy kezelik e mezőket, mintha text típusúak lennének, így használatuk régebbi böngészők alkalmazása esetén sem okozhat semmilyen problémát.


T-Mobile H
0:36
27 %
Previous
Next
AutoFill
Done
Q W E R T Z U I O P
A S D F G H J K L
Y X C V B N M
_123
Szóköz
@
.
Ugorj
T-Mobile H
0:36
27 %
Previous
Next
AutoFill
Done
Q W E R T Z U I O P
A S D F G H J K L
Y X C V B N M
@123
.
/
.com
Ugorj

Alkalmazás

Projekt adatlap

Cím: Debreceni Menük

Rövid leírás: A rendszer a város éttermeinek menüit teszi elérhetővé és kereshetővé.

Határidő: 2011. március 31.

Alkalmazott módszertan: agilis.

A projekt résztvevői: Máté Balázs (projektvezető)

Vízió

Magyarország legtöbb nagyvárosában rengeteg étterem kínál menüt, amely segítségével változatos és költséghatékony étkezés oldható meg, azonban a menük interneten keresztüli elérésével kapcsolatban számos problémával találjuk szemben magunkat:

- ha az étteremnek nincs saját honlapja, akkor a menüje rendszerint nem elérhető
- a honlapon lévő menük nem naprakészek
- az egyes oldalakon nehézkes a menük elérése
- a meglévő gyűjtőoldalak nehezen használhatóak, nem frissülnek, nincs bennük keresés
- a megfelelő menük kiválasztásához 4-5 oldal megtekintése is szükséges, ami sok időt vesz igénybe
- a mobiltelefonos információszerzés még időigényesebb és körülményesebb

A fejlesztendő rendszer célja ezen problémák orvosolása egy olyan gyors és könnyen kezelhető rendszerrel, amely a menü kiválasztásának idejét töredékére csökkenti. Fontos szerepet kap a mobiltelefonról való elérést biztosító felhasználóbarát alkalmazás, hiszen számos esetben a felhasználó nem ül a számítógép előtt, amikor az ebédjének helyéről akar dönteni. A meglévő problémák a következőképpen oldhatóak meg:

- a felhasználónak egyetlen oldal megtekintésére van szüksége

- a különböző éttermek menüi összehasonlíthatóak
- az adatok közvetlenül az étterem illetékesétől származnak, így mindig naprakészek
- új éttermek felfedezésének lehetősége térképi megjelenítésből fakadóan
- személyre szabható keresési találatok távolság és kedvenc éttermek alapján
- gyors, funkciógazdag mobilelérés

Követelmények

Funkcionális követelmények:

- menük keresése dátum és étterem szerint
- éttermek térképen való megjelenítése
- éttermek alapadatainak elérése
- kedvenc éttermek kiválasztása a keresési eredmények személyre szabásához
- keresés étterem és felhasználó távolságának figyelembevételével
- menü feltöltése az étterem illetékesei által
- adminisztrátorok és étterem adminisztrátorok authentikálása

Nem funkcionális követelmények:

- a rendszer legyen könnyen és gyorsan kezelhető
- a rendszer probléma nélkül működjön a 4 legnagyobb asztali böngészőn
- a mobil verzió működjön iOS és Android platformokon
- a mobil verzió működjön internetelérés nélkül is.
- a rendszer legyen publikusan elérhető

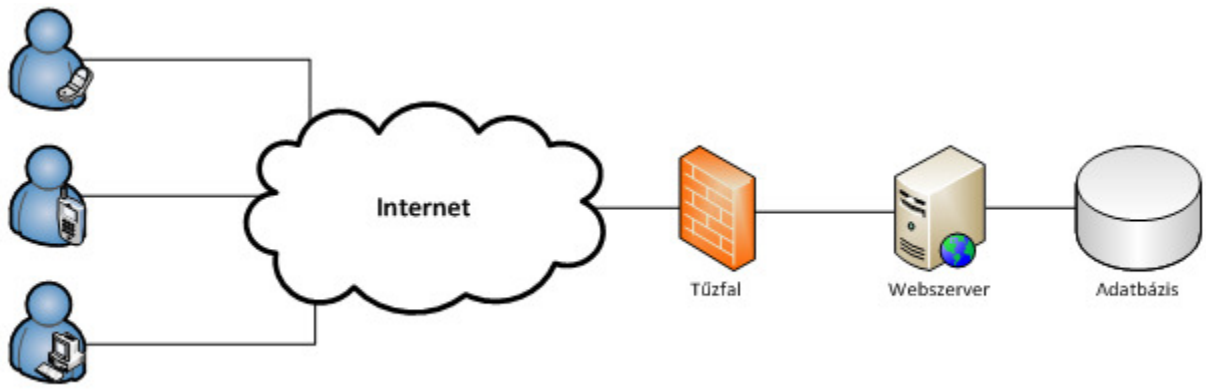
A megvalósításhoz használt technológiák

- Microsoft SQL Server
- ASP.NET WebMatrix
- jQuery

- jQTouch
- Google Maps

Architekturális terv

Az alkalmazás architektúrája a webes környezetben már jól ismert és bevált háromrétegű kliens-szerver architektúra, amelynek szemléltetése a következő ábrán látható.



Kliens oldal

Az asztali alkalmazás esetén a felhasználó a böngészőn keresztül éri el az alkalmazást, így a böngésző, mint vékony kliens funkcionál. A mobil verzió esetén is ugyanaz az architektúra, ugyanúgy böngészőn keresztüli az elérés, azonban „vastagabb” kliensről beszélhetünk, hiszen néhány funkció kliens oldalon kerül megvalósításra JavaScript segítségével, valamint egyes adatok perzisztensen replikálásra kerülnek a kliens oldalon kihasználva ezzel a HTML 5-ben rejlő új lehetőségeket.

Szerver oldal

Webszerver

A webszerver felel a http protokollon keresztüli kliens-szerver kommunikációért, valamint a rendszer funkcióit megvalósító webalkalmazás is itt fut folyamatos kapcsolatban állva az adatbázisszerverrel.

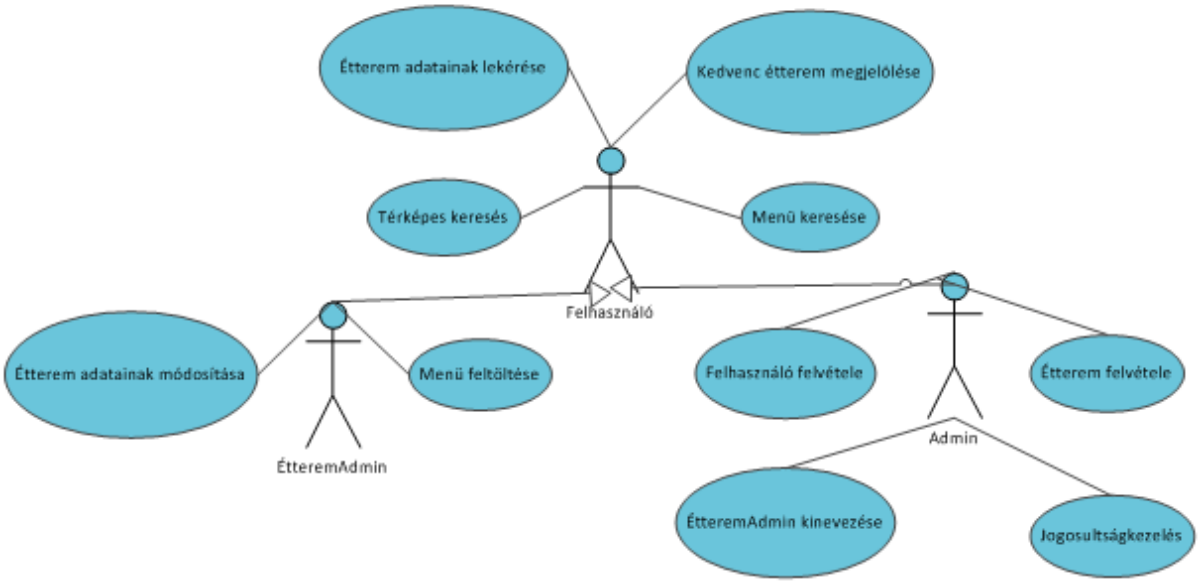
Adatbázisszerver

Az alkalmazáshoz szükséges adatok relációs adatbázisban kerülnek tárolásra, amely a webszerverrel áll kapcsolatban.

Logikai terv

Use case diagram

A funkcionális követelmények alapján a következő use case diagram készíthető el:

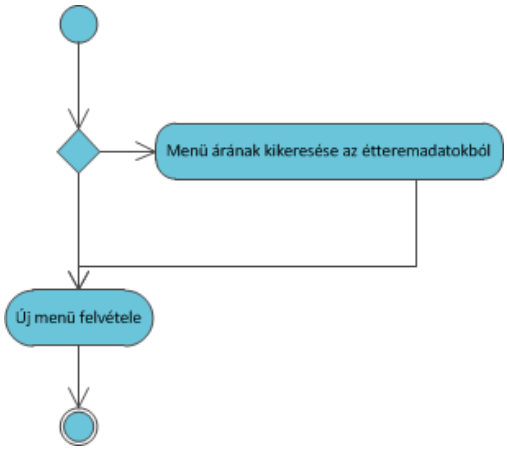


Az alkalmazás mobilverziójában csak a felhasználó actorhoz tartozó use casek érhetőek el, míg az asztali verzióban az összes.

Activity diagramok

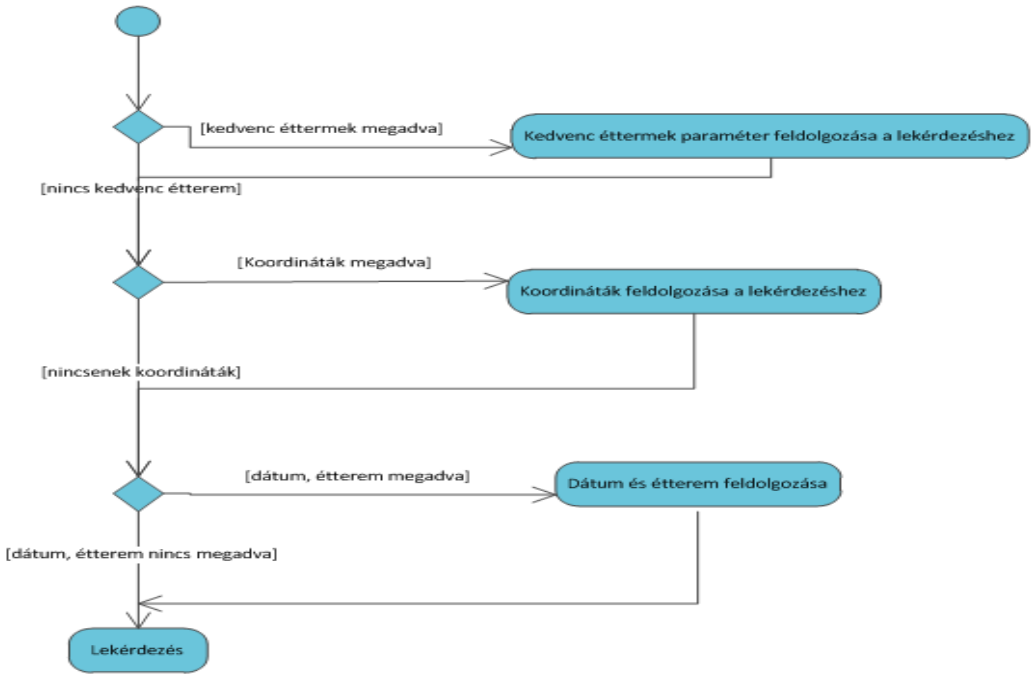
Új menü felvétele

Új menü felvétele esetén figyelembe vesszük, hogy adott menühöz lett-e megadva ár. Ha nincs, akkor az étteremnél alapértelmezett menüár kerül figyelembevételre.

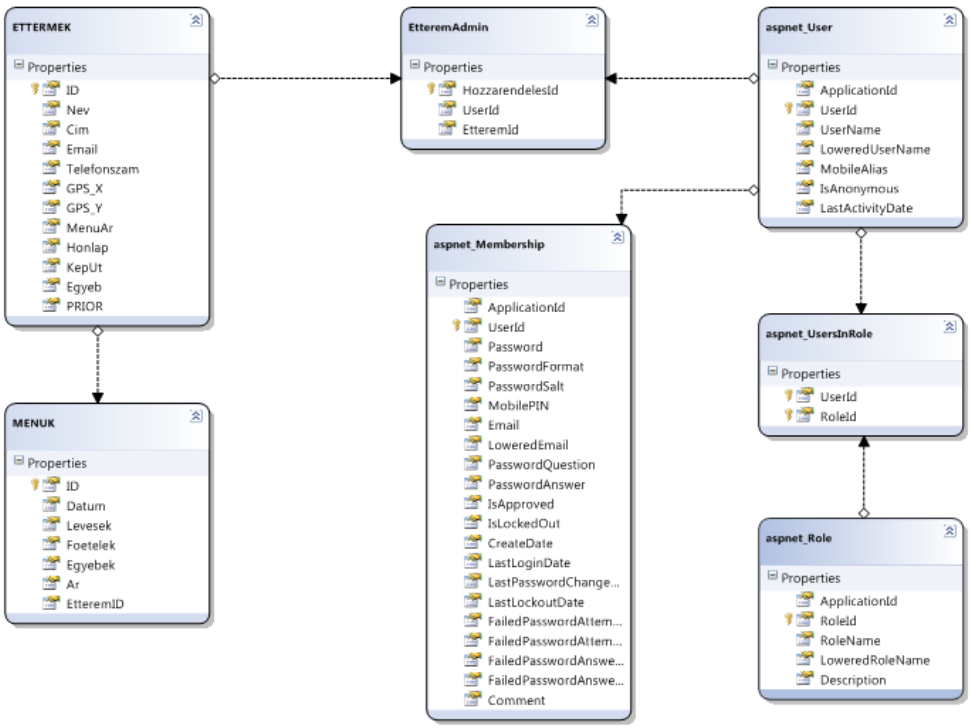


Menü keresése

A menük keresésénél számos paraméter lehetséges, amelyek tetszőleges kombinációja elképzelhető, így a lekérdezést ezek figyelembevételével kell elvégezni.



Adatbázisterv

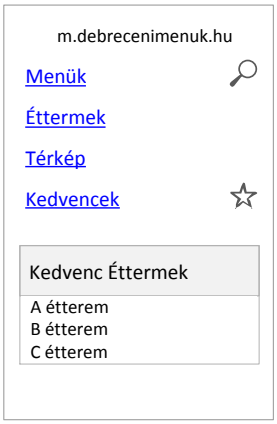
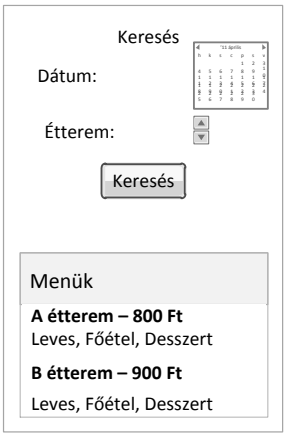


Az adatbázis tervezésénél fontos szerepet játszott, hogy az ASP.NET-ben elérhető felhasználó és szerepkör menedzsmentre fog épülni a felhasználók kezelése. A felhasználóhoz tartozó adatok az aspnet_User és aspnet_Membership táblákban, a különböző szerepkörök az aspnet_Role táblában találhatóak, míg a szerepkörök és felhasználók közötti több-többes kapcsolat az aspnet_UsersInRole tábla segítségével kerül megoldásra.

Az éttermekről az alaptulajdonságokon felül, mint ID, Név, Cím, Telefon, E-mail, Telefonszám, Honlap letárolásra kerülnek a földrajzi koordináták, az étterem rövid leírása, valamint az eredménylistához köthető prioritás is. Minden étteremhez tartozik egy vagy több adminisztrátor, akik az étterem alapadatait módosíthatják, valamint az aktuális menüket tölthetik fel. Az éttermek és ÉtteremAdmin szerepkörben lévő felhasználók közötti több-többes kapcsolatot az EtteremAdmin tábla realizálja, ahogy az az ábrán is látható.

A menüknél annak részeit, a dátumot, az étterem azonosítóját, valamint árát tároljuk le. Figyelembe kell venni, hogy a menü ára a hét különböző napjain eltérhet, így azt menünként és nem éttermenként kell tárolni.

Képernyőtervek

Főoldal	Keresés
	
Étterem lista	**Étterem adatok**

Éttermek

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Példa étterem

Példa utca 11.

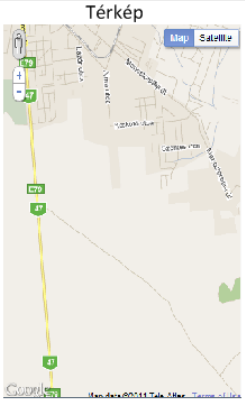
www.pelda.hu

06 52 123 456

800 Ft

Térkép

Térkép



Kedvencek

Kedvenc éttermek

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Étterem A

Összegzés

Szakdolgozatomban arra törekedtem, hogy rövid bevezető után bemutassam a minőségi, látványos mobil webalkalmazások fejlesztését támogató különböző API-kat és technológiákat, amelyek véleményem szerint egyre fontosabb szerepet fognak játszani a mobiltelefonokat célzó alkalmazásfejlesztéseknél.

Témaválasztásomnak az volt az oka, hogy sokakhoz hasonlóan úgy gondolom, a mobiltelefonokban és a mobilwebben rengeteg olyan lehetőség van, amely még nem került kiaknázásra, és amely rendkívül fontos lesz a későbbiekben. Alkalmazásfejlesztés oldaláról megközelítve a dolgokat úgy látom, hogy a webalkalmazások térhódítása főleg platformfüggetlenségük miatt növekedést fog mutatni már a közeljövőben is, így érdemesnek tartottam ezt irányzat mélyreható megismerésre.

Ki szeretném emelni, hogy a szakdolgozatomban bemutatott témák rendkívül újak, még folyamatos változásban vannak, azonban számos eszközön implementálásra kerültek, így azok ismertetése nem elhamarkodott. Ezek a folyamatos fejlesztések és újítások azok, amelyek érdekessé teszik az utóbbi időben formálódó webes technológiákat, hiszen most alakulnak ki a bevált gyakorlatok és minták, havonta jönnek ki új funkciókkal a mobilböngészők, újak a szabályok, a lehetőségek, a korlátok.

A példaalkalmazásban nagy figyelmet fordítottam arra, hogy minél több új funkció felhasználásra kerüljön, valamint, hogy felhasználói élményben minél kisebb lemaradásban legyen a natív alkalmazásokhoz képest. Úgy gondolom, hogy ezen kitűzött célokat sikerült elérnem, mivel egy könnyen kezelhető, funkciógazdag alkalmazás lett a fejlesztés eredménye.

Irodalomjegyzék

- Brian Fling – Mobile Design and Development
- Mark Pilgrim – HTML 5 Up and Running
- Gail Rahn Frederick, Rajesh Lal – Beginning Smartphone Web Development
- Jonathan Strak – Building iPhone Apps with HTML, CSS and JavaScript
- Maximiliano Firtman – Programming the Mobile Web
- Sarah Allen, Vidal Graupera, Lee Lundrigan – Pro Smartphone Cross-Platform Development
- Offline webapps: http://dev.w3.org/html5/offline-webapps/
- Web Storage: http://dev.w3.org/html5/webstorage/
- Web Database: http://dev.w3.org/html5/webdatabase/
- Geolokáció: http://dev.w3.org/geo/api/spec-source.html
- jQTouch: http://jqtouch.com/