A POLYMER PROJECT ALKALMAZÁSI LEHETŐSÉGEINEK VIZSGÁLATA EGY KONKRÉT PÉLDÁN KERESZTÜL

Nagy Flóra Boglárka

Budapesti Corvinus Egyetem, Gazdálkodástudományi Kar,
III. évf. gazdaságinformatikus BSc szakos hallgató

Jelen tanulmány kapcsán egy web alapú tesztkitöltő rendszert hoztam létre, amellyel az ukrajnai – elsősorban matematika – külső független értékelés (Зовнішнє Незалежне Оцінювання) előtt álló kárpátaljai magyar diákok felkészülését és önellenőrzését szerettem volna megkönnyíteni.
A rendszer egy innovatív technológia, a Polymer Project alkalmazásának segítségével készült el, amely egy a Web Components specifikációjára épülő könyvtár. A weboldal létrehozása során megvizsgáltam a Polymer könyvtár által nyújtott lehetőségeket – például saját, testreszabott HTML-elemek létrehozása, adatkötések alkalmazása stb. –, majd összefoglaltam a fejlesztés során szerzett tapasztalatokat ezen eszközök használatának előnyeiről és hátrányairól is, illetve mérlegeltem a létrehozott weboldal további fejlesztési lehetőségeit.

Abstract

У статті презентовано результати створення системи он-лайн тестування, яка може допомогти закарпатським угорським абітурієнтам при підготовці до зовнішнього незалежного оцінювання з математики. Система створена з допомогою застосування інноваційної технології Polymer Project, що є бібліотекою, базованою на специфікації Web Components. У статті розглянуто можливості, надані бібліотекою Polymer при складанні сайту: створення своїх власних, індивідуальних HTML налаштувань, зв'язування даних тощо. Підсумовано досвід, отриманий під час розробки системи, узагальнено переваги і недоліки використання цих інструментів, а також досліджено можливості подальшого розвитку створеного сайту.

1. Bevezetés

Ukrajnában 2008-ban döntés született egy új érettségi és egyben felvételi rendszer, a ZNO (ukránul Зовнішнє Незалежне Оцінювання, magyarul külső független értékelés) bevezetéséről. A döntés értelmében a diákok 2008-tól kezdve nemcsak saját tanintézményeikben érettségiznek, hanem központilag kijelölt tesztelő központokban is vizsgát tesznek, és az itt elért eredményeik alapján jelentkezhetnek felsőoktatási intézmények képzéseire.

Fontos azonban megemlíteni, hogy a kárpátaljai fiatalság jelentős hányada magyarországi felsőoktatási intézményben szeretné folytatni tanulmányait, így anyaországi érettségire is szükséges készülniük, melynek feladattípusai, gondolatmenete és követelménye esetenként jelentősen különbözik az ukrajnai rendszerben tanult gondolkodásmódtól. Jómagam 2012-ben érettségiztem, egyaránt tettem matematika és fizika érettségi vizsgát a ZNO keretein belül, valamint Magyarországon is. Ebből kifolyólag saját bőrömön tapasztalhattam a két különböző érettségire való felkészülés és a felmérésen való teljesítés

[1] A tanulmányt dr. Mohácsi László lektorálta.

akadályait, nehézségeit. A két rendszer közötti legfőbb különbség – a matematika vizsgára levetítve – az, hogy a magyarországi felmérés részmegoldásokat is pontoz, míg az ukrajnai vizsga – az idei évig – teszt jellegű volt: egyszeres és többszörös választást, valamint végeredményt igénylő feladatokkal találkozhatunk.

Az ukrajnai matematika érettségi feladatsorának javítása tehát könnyen automatizálható, egy megfelelő űrlapformátum elkészítése után akár program is ellenőrizheti a válaszokat, nem szükséges emberi erőforrás bevonása. Jelen írásban erre a problémára keresek megoldási alternatívákat.

Mindebből kifolyólag munkámban egy web alapú tesztelő rendszert hoztam létre, amellyel a kárpátaljai magyar, ZNO előtt álló diákok felkészülését és önellenőrzését kívánom megkönnyíteni. Ez a megoldás képezi jelen írás alapját.

2. Követelmények és megvalósítási eszközök

A tesztkitöltő rendszer létrehozása kettős célt szolgál. Egyrészt célja a felvetett problémára való megoldás kidolgozása, amely valóban alkalmazható a gyakorlatban is, másrészt a fejlesztés során olyan eszközök vizsgálata és kipróbálása, amelyek újnak számítanak jelenleg a webprogramozás területén, ám alkalmazási gyakoriságuk felívelő tendenciát mutat. Ezáltal célom megismerni az új technológiák előnyeit, illetve hátrányait, amelyek helyes felismerése fontos szerepet játszhat abban, hogy eldöntsük, érdemes-e ezek alkalmazásába befektetni *early adopter*ként.

2.1 Követelmény-analízis

A külső független tesztelésen való részvétel egyik legfőbb problémája a magyar diákok számára a nyelvi akadály. A magyar nyelvű feladatsorok megjelenésével ez szerencsére enyhült, azonban a megoldásokat nem a feladatlapon kell leadni, hanem külön egy erre kijelölt űrlapon, amely továbbra is teljes mértékben ukrán nyelvű. Ebből kifolyólag előfordulhat, hogy bár a diák helyesen oldja meg a feladatot és helyes végeredményre jut, rosszul jelöli meg válaszát az űrlapon. Számos visszajelzést lehet hallani arról, hogy a numerikus választ kívánó feladatoknál a helyes végeredményt rosszul írták be az egyes helyi értékeket jelölő rubrikákba. Ebből kifolyólag fontosnak tartom követelményként megfogalmazni a tesztelő rendszerrel szemben azt, hogy a feladatok megoldása során ne csak a feladattípusokat ismerjék meg a felhasználók, hanem az űrlap elemeit is tanulják meg használni, hogy éles helyzetben az érettségi alatt ez már ne okozzon gondot számukra.

Fontos, hogy a felhasználók az adatbázisból ne ömlesztve férjenek hozzá a kérdésekhez. Ennek kiküszöbölésére kétféle strukturálási opciót tartok szükségesnek. Az egyik értelemszerűen az érettségi feladatsorok szerinti csoportosítás, amellyel vizsgahelyzetet lehet szimulálni. Láthatóvá válik a diák számára ezáltal, hogy milyen típusú és nehézségű feladatok kerülnek egy feladatsorba, illetve hogy milyen a témakörök megoszlási aránya egy vizsgasoron belül. A másik csoportosítási lehetőség, a témakörök szerinti lekérdezése a feladatoknak. Ez azt a célt szolgálja, hogy gyakorolni lehessen az egyes témákhoz tartozó feladatokat is az összes korábbi évből, hozzájárulva ezáltal a tanulás folyamatának támogatásához is. További követelmény természetesen, hogy az összegyűjtött feladatok magyar nyelvűek legyenek.

Ezenkívül lényeges szempont, hogy a tesztelő rendszer használata egyszerű legyen és felhasználóbarát, megjelenése pedig áttekinthető, letisztult, továbbá előny, ha a diákok körében egyre elterjedtebb okostelefonokon, táblagépeken is megjeleníthető és használható a weboldal.

Összefoglalva tehát a web alapú tesztelő rendszerrel szemben támasztott követelmények a következők:

- külalakra és funkcionalitásra a megoldások számára megjelenített beviteli elemek az érettségi űrlapján használatosakkal a lehető legnagyobb mértékben megegyezzenek
- a feladatok vizsgasor szerinti strukturálása az éles helyzet szimulálása céljából
- a feladatok témakör szerinti strukturálása a tanulás és a gyakorlás megkönnyítésének érdekében
- magyar nyelvű feladatok
- áttekinthető, letisztult megjelenés; egyszerű, felhasználóbarát működés
- táblagépeken való használat lehetősége.

2.2 Megvalósítási eszközök

2.2.1 Web Components

A 2007–2012 közötti tesztek feladatainak száma tesztsoronként 32 és 38 között váltakozik, blokkonként követik egymást az egyszeres választás, a többszörös választás típusú és a numerikus végeredményt igénylő feladatok. A nagy adatmennyiségre, valamint a matematikai kifejezések, grafikonok, ábrák webes felületre történő beviteli nehézségeire való tekintettel a feladatok szövege a válaszlehetőségekkel együtt képként kerültek az adatbázisba. A fentiek alapján nyilvánvalóvá vált, hogy a képek és a válasz beviteli elemek kódrészlete a fejlesztésen belül rendkívül sokszor ismétlődne.

Erre a problémára kínál egy lehetséges megoldást a Web Components. A Web Components jelenleg a Google által fejlesztés alatt álló szabványok összessége, melyeknek segítségével újrafelhasználható, szerkeszthető egységbezárt widgetek, elemek, komponensek hozhatók létre. A fejlesztés célja a W3C standard specifikáció elérése. Lényege, hogy az olyan kódrészletekre, amelyek újrafelhasználhatók a webfejlesztés során, új, teljes értékű tagek válnak definiálhatóvá anélkül, hogy W3C standard elemmé válna. (Dodson 2013) Mindez nagy szabadságot biztosít a webfejlesztők számára. A Web Components-nek – a specifikációjának megfelelően – 4 alappillére van, amelyek alapvetően együtt funkcionálnak, de külön is alkalmazhatóak (Rimmer 2012):

- Custom Elements
- HTML Imports
- Templates
- Shadow DOM

2.2.1.1. Custom Elements

A Custom Elements módszer segítségével a fejlesztő új, testre szabott elemeket hozhat létre saját tagnévvel párosítva. Ezeket JavaScript segítségével regisztrálhatjuk a DOM-ba (Document Object Model – egymással „gyerek-szülő” kapcsolatban álló objektumok modellje). Ezáltal valódi HTML-elemekké válnak, és alkalmazhatóak lesznek rajtuk a standard DOM-metódusok, például formázhatjuk a stílusát CSS használatával, rendelhetünk hozzájuk tulajdonságokat (properties), eseménykiszolgáló metódusokat. A testre szabott elemek létrehozásának egyetlen megkötése, hogy a névnek tartalmaznia kell kötőjelet (-) annak érdekében, hogy elkerüljük az ütközést a már meglévő vagy esetleg a jövőben definiálásra kerülő standard HTML-elemekkel. (Bidelman 2013a)

2.2.1.2. HTML Imports

A HTML Imports egyszerű technikát kínál HTML-fájlok – különösen a testre-szabott HTML-elemek kódjának – importálására más HTML-dokumentumokba,

 ISSN 2411-4081

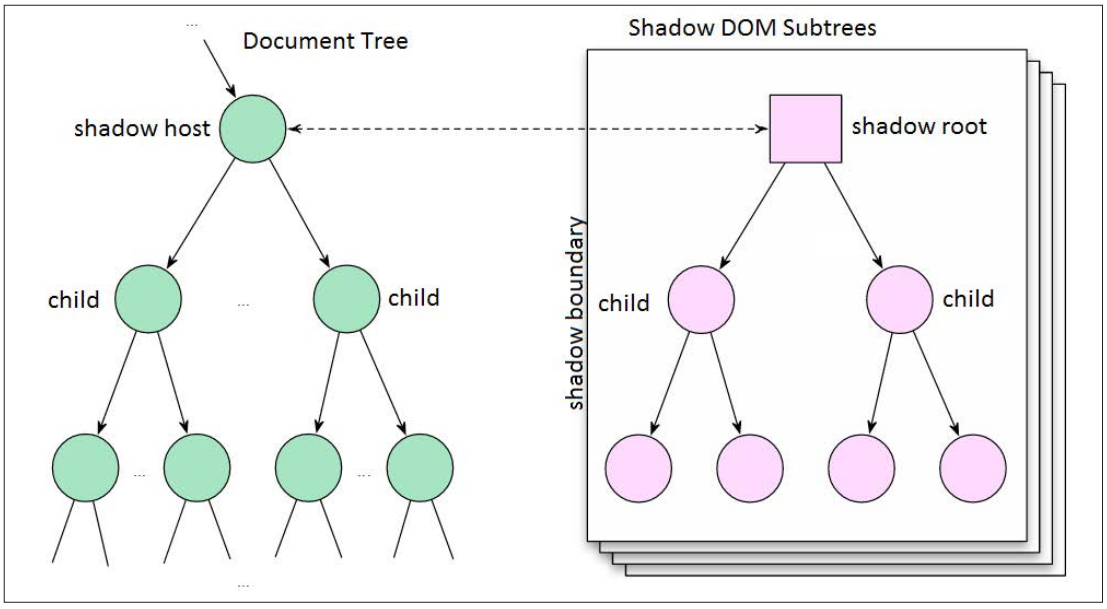
a `<link rel="import">` kód deklarálásával, valamint a fájl pontos elérési útjának megadásával. Ennek köszönhetően egy import nem csak HTML-kódot tartalmazhat, hanem például a hozzá tartozó CSS formázást vagy JavaScriptet is, így akár a logikailag elkülönülő kódrészletek jobban strukturálhatóvá, ténylegesen elválaszthatóvá válnak egymástól. (Bidelman 2013b)

2.2.1.3. Templates

Template-ek – vagy sablonok – alkalmazása az eddigiekben szerver oldali programozás során volt leginkább megszokott, az utóbbi időben azonban használatuk egyre gyakrabban előfordul a front-end megalkotása során is, amelyeket a korábbiakban JavaScript segítségével hoztak létre. A Web Components részeként a Templates specifikációja azt határozza meg, hogyan deklarálhatóak inaktív (inert) DOM-részfák HTML-ben, valamint hogy hogyan alkalmazhatóak ezek azonos tartalommal rendelkező dokumentumrészletek megalkotásához. (Kitamura 2012)

2.2.1.4. Shadow DOM

Igen gyakori, hogy a weboldalak interaktívabb, vonzóbb megjelenése érdekében HTML és JavaScript segítségével alkotott widgeteket használnak. Alkalmazásukat azonban megnehezíti, hogy nincsenek egységbezárással elhatárolva a kód többi részétől, ugyanis elnevezések, id-k ütközései miatt a stíluslapunk véletlenül módosíthatja annak megjelenését vagy a szkriptünk a funkcionálását. Ezt a problémát lehet kiküszöbölni a Shadow DOM-mal. Lényege, hogy funkcionális határokat képezhetünk DOM-részfák között: egy adott elemhez hozzárendelhetünk egy „csomópontot" (node), ezt *shadow root*nak nevezzük. Azt az elemet pedig, amelyhez hozzá van kötve egy shadow root, *shadow host*nak nevezzük. Ezt a módszert használva nem a shadow host tartalma kerül betöltésre, hanem a shadow roothoz tartozó tartalmak. (Cooney 2013)



1. ábra. ***DOM-fa és hozzá tartozó shadow DOM-részfák***

2.2.2. Web Components könyvtárak

Három fő könyvtár van elterjedőben, amely a Web Components standardjaira épül, illetve kiterjeszti azokat:

Az ***X-Tag*** a Mozilla által létrehozott és fejlesztett kis JavaScript könyvtár, amely a Web Components Custom Elements és HTML imports specifikációja által nyújtott lehetőségeket kívánja elérhetővé tenni az összes modern böngészőben. Erre épül rá a Brick, amely 14 előre definiált, újrafelhasználható web-komponenst tartalmaz. (X-Tag)

A ***Bosonic*** egy eszközkészlet, működésének lényege pedig, hogy transcompilerként (egyik programozási nyelvről egy másikra fordít) a Web Components HTML fájljait JavaScript és CSS-fájlokká fordítja le, futtathatóvá téve ezzel a modern böngészőkben. Emellett egy alap, 15 elemből álló gyűjteményt is nyújt. (Bosonic)

A ***Polymer Project*** a Google által fejlesztett könyvtár, amely három konceptuális réteggel rendelkezik: (Polymer)

- *Web components:* ez a könyvtár is e specifikáció alapjain épült. Mivel alapvetően nem minden böngésző támogatja ennek a tulajdonságait, szükséges egy web komponens polyfill réteg (könyvtárak gyűjteménye) alkalmazása, amely implementálja JavaScriptben az API-kat (application programming interface /alkalmazásprogramozási felület vagy inerfész). Futás során a Polymer automatikusan a gyorsabb módszert választja: ha a böngésző támogatja a Web Components specifikációt, akkor nem alkalmazza a polyfill réteget.
- *a Polymer* könyvtár: deklaratív szintaktikát nyújt, amely leegyszerűsíti az új, testre szabott elemek definiálását. Ezen felül olyan tulajdonságokkal is bír, mint a kétirányú adatkötés, az elemek tulajdonságainak megfigyelése, vagy a gesztusvezérlés támogatása, amely tovább növeli az egyes komponensek felhasználási lehetőségeit.
- *Elements:* a Core és Paper element gyűjtemények egy igen átfogó kollekciói különböző UI és non-UI elemeknek, amelyek már felhasználásra készek. Használatuk opcionális, azaz ezek a gyűjtemények felhasználhatóak akár a Polymer nélkül is, de alkalmazhatóak csak a Polymer könyvtárak a Core és Paper elemek nélkül, saját elemek alkotására. Fejlesztés során akár kizárólag a saját elemekből is felépíthető a weboldal vagy webalkalmazás, de kombinálva is alkalmazhatóak Core és Paper elemek saját, vagy standard beépített elemekkel.

3. Fejlesztés

A felsorolt könyvtárak közül a felvetett probléma megoldására a legmegfelelőbbnek a Polymer könyvtár bizonyult, mivel egyrészt igen nagy, kész stílussal rendelkező, azonnal felhasználható elemek készletét bocsátja rendelkezésre a fejlesztéshez, másrészt a tesztelő rendszer működéséhez szükséges adatbázissal való munkát megkönnyíti a Polymer által támogatott adatkötések létrehozása, továbbá a követelményként megfogalmazott táblagépen való kezelhetőséget is megalapozhatja a gesztusvezérlés támogatása.

 ISSN 2411-4081

2. ábra. ***A Polymer Core és Paper elemeit, valamint standard elemeket felhasználó weboldal***

3.1. A Polymer tulajdonságai

3.1.1. Testreszabott elemek definiálása

Ahogy a 2.2.1 részben olvasható, a válaszok számára biztosított beviteli elemeket igen sokszor szükséges megjeleníteni a weboldalon. Ezért a kódrészletek ismétlésének kiküszöbölésére a Polymer által nyújtott lehetőséget alkalmaztam egy új elem létrehozására, amit a `<polymer-element>` tag használatával lehet megtenni, a `name` tulajdonság kötelező megadásával, ahol meghatározhatjuk az új tag nevét, amellyel később hivatkozhatunk rá.

```
<polymer-element name="simple-choice">
    <template>
        <table>
            <tr>
                <th>A</th>
                <th>Б</th>
                ...
        </table>
    </template>
    <script>
        Polymer()
    </script>
</polymer-element>
```

3. ábra. ***Kódrészlet egy saját elem létrehozásának illusztrálására***

A `<polymer-element>` tagen belül az elem definiálása két részre oszlik. Az első részben `<template>` tagek között határozhatjuk meg az elem struktúráját. Használhatunk ehhez pusztán standard HTML-elemeket, vagy a Polymer elemeit is, de kombinálhatjuk is azokat. CSS-formázásra is van lehetőségünk a templateen belül akár `<style>` tagek közé zárva, akár külön-külön az egyes elemekhez kapcsolódóan.

A második részben JavaScriptet rendelhetünk a komponensünkhöz `<script>` tagek között, ahol a kódot a `Polymer()` kulcsszóba foglalva írhatjuk meg. Ha nem szeretnénk szkriptet hozzárendelni, akkor üresen hagyhatjuk ezt a részt, vagy teljesen elhagyhatjuk a `<script>` tageket, ha a definiáló tagbe bekerül a `noscript` tulajdonság. Például:

```
<polymer-element name=”simple-
choice” noscript>
```

Az elemet definiáló kód ezután egyszerűen a `<simple-choice>` nyitó és záró tag használatával meghívhatóvá válik. Majd lehetőségünk van arra, hogy – a Web Components másik tulajdonságát, a HTML Importot alkalmazva – a kódot elmentve egy külön HTML-fájlba, teljesen elválaszthatjuk a weboldaltól, majd importálhatjuk

oda a `<link rel="import" href="URL_helye">` sor segítségével. Így érthetőbbé válik a fejlesztés, és rövidebbek, átláthatóbbak lesznek a fájlok. Ez különösen hasznos például olyan esetekben, amikor egyszerre több fejlesztő dolgozik egy projekten.

3.1.2. Adatkötés Polymerrel

Mivel a Polymer által támogatott adatkötés kliens oldalon fut, a használni kívánt adatok tárolását ennek megfelelően kellett kezelhetővé tenni. Erre a célra megfelelőnek bizonyult a JSON (JavaScript Object Notation) formátuma, amely egy kisméretű, adatcserére szolgáló, csak szöveget tartalmazó fájltípus. Nem csak külön fájlban használhatjuk azonban ezt a formátumot, hanem JavaScript kódon belül is alkalmazhatjuk. A JSON-ben elempárokat és azok tömbjeit tárolhatjuk. A párok első fele egy azonosító kulcs, majd kettőspont után követi az eltárolandó érték, amely lehet szöveg, és szám is, de felvehet akár igaz, hamis, vagy null értéket is. Erre mutat példát a 4. ábra.

A már ismertetett módon definiált új polymer-elementen keresztül mutatja be a 4. ábra az adatkötés módját is. Az elem scriptet tartalmazó részében található meg a JSON formátumú adathalmaz, amelynek 1-1 sora a vizsgasorok adott feladatához tartozó információit tartalmazza.

Amint látható, az elem sablonjának meghatározásánál több `<template>` tag használatára volt szükség. A külső, repeat-et tartalmazó tag egy ciklust testesít meg, amely {{}} zárójelek között valósítja meg az adatkötést, példányosítva a feladatok objektum sorait. Ezután az első adatkötött alkotórész, a kép következik, amely a soron következő feladat képét hívja meg az elérési út alapján. Amint látható, az elem létrehozásakor már alkalmaztam a korábban megalkotott, válaszadáshoz szükséges komponenseket. Ezeket feltételt tartalmazó, adatkötött template-ekbe helyeztem, melyek csak akkor térnek vissza a válaszelemmel, ha a feltétel igaznak bizonyul, vagyis ha az adott választípus kulcsához igaz érték van

```
<polymer-element name="kerdesek-element">
    <template>
        ...
        <template repeat="{{f in feladatok}}">

                <img src="pictures/{{f.picture}}">
                <br />
                <template if="{{f.simpchoice}}">
                    <simple-choice ev="{{f.year}}" tipus="{{f.type}}" sorszam="{{f.no}}"></simple-choice>
                </template>
                <template if="{{f.multchoice}}">
                    <multi-choice ev="{{f.year}}" tipus="{{f.type}}" sorszam="{{f.no}}"></multi-choice>
                </template>
                <template if="{{f.num}}">
                    <num-ans ev="{{f.year}}" tipus="{{f.type}}" sorszam="{{f.no}}"></num-ans>
                </template>
        </template>
    </template>
    <script>
        Polymer({
            ready: function () {
                this.feladatok = [
            { "year": 2007, "type": "vizsga1", "no": 1, "picture": "2007vizsga1/1.JPG", "simpchoice": true,
                "multchoice": false, "num": false, "answer": "A", "topic": "szam_kif" },
            { "year": 2007, "type": "vizsga1", "no": 2, "picture": "2007vizsga1/2.JPG", "simpchoice": true,
                "multchoice": false, "num": false, "answer": "Д", "topic": "szam_kif" },
            { "year": 2007, "type": "vizsga1", "no": 3, "picture": "2007vizsga1/3.JPG", "simpchoice": true,
                "multchoice": false, "num": false, "answer": "B", "topic": "komb_val_stat" },
            ...
            ]}
        })
    </script>
</polymer-element>
```

4. ábra. ***Kódrészlet az adatkötés és a JSON illusztrálására Polymerrel***

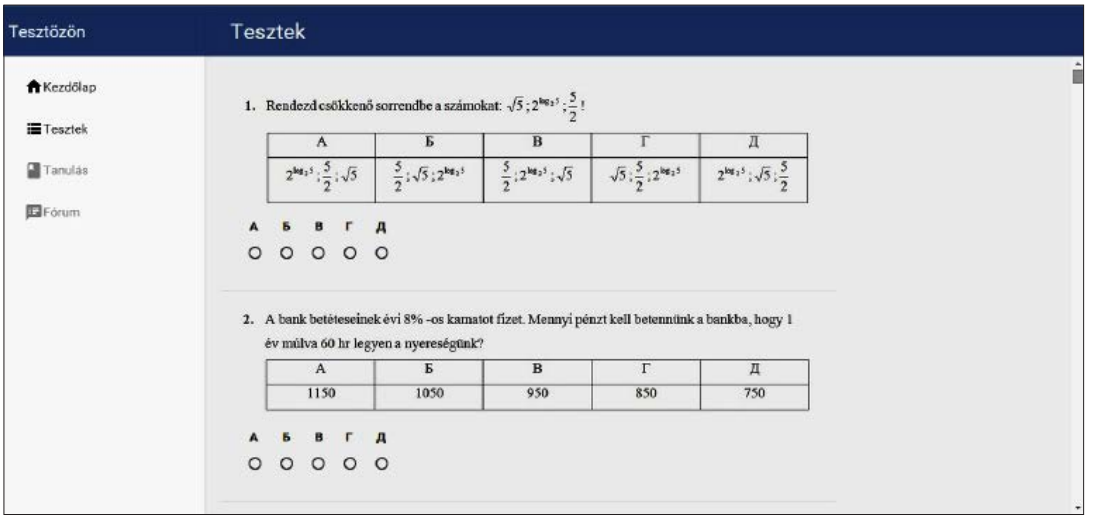
 ISSN 2411-4081

rendelve. Azonkívül a testre szabott elemek létrehozásakor definiált új tulajdonságok is adatkötéssel rendelkeznek.

A fenti kódrészletet tartalmazó teljes kód megjelenési formájával a weboldalon a megoldani kívánt témakörök vagy vizsgasor kiválasztása után találkozhat a felhasználó, melyeket az 5. ábra mutat be.

3.1.3. A Shadow DOM használatának nehézségei

Mivel a Shadow DOM részfák egységbezárással elválasztásra kerültek a dokumentum fájától, a hagyományos DOM metódusok nem érik el a Shadow DOM-ban elhelyezett elemeket. Ilyen metódus például a `getElementById()` vagy a `querySelector()`, melyekkel megfogha-



5. ábra. ***Az adatkötést tartalmazó ismétléses elem megjelenése***

A weboldal másik része, ahol igen fontos szerepet játszanak az adatkötések, az a megoldások ellenőrzése. Mivel eddig nem volt szükség szerver oldali programozásra, és kis mennyiségű adatról van szó, a felhasználók által adott válaszok ideiglenes eltárolására jól alkalmazható a böngészők helyi tárhelye. JavaScript metódusokkal könnyedén menthetünk ide és kérhetünk le adatokat. Felépítése a JSON formátumához hasonló: egy string kulcshoz párosítja az adatot, de az adat típusa is kizárólag string lehet. A feladatok megoldása során eltárolt adatok lekérése után, összehasonlítva a JSON-ben lévő megoldással megállapítja a weboldal, hogy a válasz helyes volt-e vagy sem. Az adatkötés itt abban játszik szerepet, hogy a megfelelő válasz a megfelelő megoldással kerüljön összehasonlításra, melyek azonosítása a feladatot tartalmazó vizsgasor éve, típusa és a feladat sorszáma segítségével valósul meg.

tóvá válnának az egyes elemek, hogy eseményfigyelőt rendeljünk hozzá, vagy módosítsuk annak tulajdonságait JavaScript segítségével. A probléma kiküszöbölését segíti azonban a this.$.id hivatkozási módszer, melyben az id helyén az elérni kívánt elem azonosító nevét kell használnunk. Így például a Shadow DOM-ban elhelyezett, `id="eltunik"` azonosítójú paper-button kattintásához kötött eseménykiszolgáló függvényben a `this.$.eltunik.hidden=true;` kódsor a gomb eltűnését fogja maga után vonni.

Az integrált fejlesztő környezetek alkalmazásának fő előnye, mint például a Microsoft Visual Studio, hogy a JavaScript kódban előforduló hibák követésére is alkalmas. Megnehezíti azonban a Shadow DOM-mal való munkát az is, hogy az ilyen környezetek által a hibakeresés (debugging) során a jelzett problémák gyakran

nem fedik a helyes futást akadályozó valódi problémát, mivel a Shadow DOM-ban felmerülő problémákat – egyelőre – nem tudják megfelelően jelezni.

3.2. Kompatibilitás

Mivel a Web Components még nem standard specifikáció, indokolt az elkészült weboldal megjelenítésének tesztelése különböző böngészőkben, ugyanis azok nem feltétlenül támogatják még a Web Components alapokra épülő weboldalakat. Az 5 legelterjedtebb böngészőt vizsgáltam meg:

- Chrome 41: a Google böngészője, így természetesen teljes mértékben támogatja a Polymer megjelenését.
- Firefox 27: A komponensek betöltése alapvetően megfelelő, néhol azonban megfigyelhetőek apróbb elcsúszások.
- Opera 28: A Chromium alapokon működő böngészőben szintén teljes mértékben jól jelennek meg a komponensek.
- Safari 5: Az Apple böngészőjében sajnos csak a standard elemek jelennek meg, a Polymert egyáltalán nem támogatja.
- IE8: Szintén nem támogatott jelenleg a Microsoft böngészőjének 8. verziójában a Polymer, nem alkalmas a weboldal megjelenítésére.

Ezeken kívül három – Androidos, Windows alapú és egy Kindle – táblagépen is megpróbáltam betölteni az oldalt. Az Android szoftvert futtató tablet saját böngészője az IE-hez és a Safarihoz hasonlóan nem volt alkalmas a weboldal betöltésére, Chrome-ban azonban probléma nélkül használható volt. A Windows operációs rendszert futtató táblagépen is Chrome-ban próbáltam ki a weboldal használatát, a megjelenítés értelemszerűen itt is problémamentes, a gesztusvezérlés megfelelően működik. A Kindle táblagép böngészőjének tesztje is sikeresnek bizonyult, megfelelően funkcionált a weboldal, bár az elvártnál kissé lassabban reagált a gesztusokra.

3.3. A szerveroldal hiányának előnyei és hátrányai

A kizárólag kliens oldali programozással megoldható weboldalak és alkalmazások egyértelmű előnye, hogy a létrehozásához kevesebb nyelv ismerete is elégséges lehet, vagyis elérhető a cél pusztán a HTML-, JavaScript és CSS-nyelvek ismeretével. Ezzel jelentősen leegyszerűsödik a feladat, hiszen a szerveroldal programozása (például PHP, MySQL alkalmazásával) és fenntartása jelentős emberi erőforrást igényelhet.

A jelen tanulmányban megalkotott weboldal esetén a képek mennyisége, illetve az adatbázis kisméretű formátumban való tárolása megengedi, hogy mindez pusztán a kliensoldal igénybevételével megoldható legyen. Abban az esetben azonban, ha a tárolt feladatok mennyisége jelentősen megnövekedne a tesztelő rendszer továbbfejlesztése során – például annak kiterjesztésével más tantárgyakra is – a szerveroldal alkalmazása elengedhetetlenné válik. Emellett más funkciók esetleges beépítése is megkövetelheti a szerver oldal használatát. Ilyen lehet például profilok regisztrálása annak érdekében, hogy statisztikát tartsunk nyilván a helyes válaszok arányáról egyénenként, hogy a diákok követni tudják saját fejlődésüket, vagy fórum beépítése, ahol akár egymástól, akár tanáraiktól/mentoroktól kérhetnek segítséget egy-egy feladat megoldásához. A szerveroldal hiánya tehát jelentősen korlátozhatja lehetőségeinket további funkciók programozása során.

4. Továbbfejlesztési lehetőségek

A tanulmány elején felvetett problémára nyújtott megoldás létrehozásának egyik fő célja a használatra bocsátás a kárpátaljai magyar, érettségi előtt álló diákok számára. Jelenleg a tesztelő rendszer rendeltetésszerűen

 ISSN 2411-4081

funkcionál, azonban a kibocsátás előtt lehetséges és szükséges annak továbbfejlesztése. Fontos a felhasználóbarát élmény fokozása a weboldal informatívabbá tételével, súgók elhelyezésével, a teljes vizsgasorok megoldásához opcionális időzítő hozzárendelésével, hogy teljes értékű legyen a szimuláció.

Ezenkívül számos távlati cél fogalmazható meg. Jelentős előrelépést jelentene a weboldal számára, ha nem csupán a teszteket és annak megoldásait tartalmazná, hanem az érettségihez szükséges témakörök tananyagát, a hozzájuk kapcsolódó magyarázatokat is, egy helyen elérhetővé téve ezzel a felkészüléshez elengedhetetlen elméleti anyagokat.

Technikai jellegű fejlesztést is igényelnek a további lehetőségek. Lényeges lenne, hogy a jövőben a mobil eszközökön történő megjelenítés és használat ne ütközzön akadályokba. A szerveroldali fejlesztési lehetőségek között pedig szerepel a már említett profilrendszer létrehozása, fórum kialakítása, valamint a rendszer kiterjesztése további tantárgyakra. Érdemes megvizsgálni továbbá az iskolák, és/vagy tanárok bevonásának lehetséges módjait.

5. Összegzés

A bevezetésben ismertetett problémára egy web alapú tesztkitöltő rendszer létrehozása nyújthat megoldást, amely egyaránt képes szimulálni éles vizsgahelyzetet, de a tanulási folyamatot is támogatni tudja a feladatok témakör szerinti csoportosításával.

Egy ilyen megoldás létrehozásához vizsgáltam meg olyan innovatív technológiák alkalmazásának lehetőségét, mint a HTML5 Web Components specifikációja, illetve az arra épülő Polymer Project könyvtár. Ezek lényege, hogy a modern weboldalak fejlesztése során felmerülő ismétlődő vagy újrafelhasználható kódrészletek elkülöníthetővé, illetve importálhatóvá válnak, átláthatóbbá és érthetőbbé téve ezáltal a teljes kódot.

Összegezve a Polymer használata során szerzett tapasztalatokat a véleményem az, hogy bár okozhat nehézségeket akár a fejlesztés során, akár az egyes böngészőkben való megjelenítés korlátozottságából kifolyólag, ezeknek a problémáknak jó része visszavezethető arra, hogy a Web Components specifikációja még nem kapta meg a W3C standard besorolást. Ezzel szemben olyan előnyök állnak, mint a kész, azonnal alkalmazható webkomponensek igen széles gyűjteménye, amelyek segítségével minimális stílusformázással kész weboldalakat alkothatunk meg, továbbá nem elhanyagolható a saját, testre szabott komponensek alkotási lehetősége, melyek teljes értékű elemekként funkcionálnak, rendelhetőek hozzájuk JavaSript és CSS-kódok is.

Véleményem szerint érdemes a témában további kutatásokat folytatni, mivel ígéretes fejlődési tendenciát mutat, illetve egyre több böngésző támogatja a Web Components specifikációján alapuló könyvtárak megjelenítését.

A tesztkitöltő weboldal ideiglenesen a web.uni-corvinus.hu/~dcjqea címen érhető el.

Irodalomjegyzék

Solymos Szilárd: *Az ukrajnai emelt szintű matematika érettségi feladatok elemzése 2007 és 2011 között.* Szakdolgozat. Beregszász, 2012

Dodson, Rob: *A Guide to Web Components*, 2013

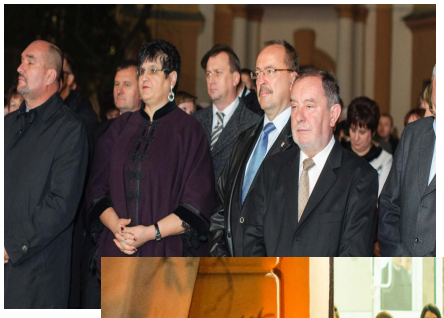
https://css-tricks.com/modular-future-web-components/ (Letöltés dátuma: 2015. március 12.)

Rimmer, Jon: *Are We Componentized Yet?,* 2012

http://jonrimmer.github.io/are-we-componentized-yet/ (Letöltés dátuma: 2015. március 12.)

Bidelman, Eric: *Custom Elements (defining new elements in HTML),* 2013a
http://www.html5rocks.com/en/tutorials/webcomponents/customelements/ (Letöltés dátuma: 2015. március 12.)
Bidelman, Eric: *HTML Imports (#include for the web),* 2013b
http://www.html5rocks.com/en/tutorials/webcomponents/imports/ (Letöltés dátuma: 2015. március 15.)
Kitamura, Eiji: *Introduction to the template elements*, 2012
http://webcomponents.org/articles/introduction-to-template-element/ (Letöltés dátuma: 2015. március 15.)
Cooney, Dominic: *Shadow DOM 101*, 2013
http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/ (Letöltés dátuma: 2015. március 15.)
X-Tag - The Custom Elements Polylib
http://x-tags.org/ (Letöltés dátuma: 2015. március 19.)
Bosonic
http://bosonic.github.io (Letöltés dátuma: 2015. március 19.)
Polymer: *Understanding Polymer*
https://www.polymer-project.org/0.5/docs/start/everything.html (Letöltés dátuma: 2015. március 20.)
Polymer: *Web components polyfills*
https://www.polymer-project.org/0.5/docs/start/platform.html (Letöltés dátuma: 2015. március 20.)

 ISSN 2411-4081



2013. november 8.

Tizenegy éves kitartó munka eredményeképp **megújult a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola épületének tetőszerkezete és homlokzata.**