Eötvös Lóránd Tudományegyetem

Informatika Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Tanulási keretrendszer a C++ programozáshoz

Témavezető:

**Pataki Norbert**

Adjunktus, PhD

Szerző:

**Török Richárd Dávid**

Programtervező informatikus, Bsc

Budapest, 2021

Tartalom

[1. Bevezetés 2](#_Toc58360629)

[1.1 Témaválasztás indoka 2](#_Toc58360630)

[1.2 Megoldandó feladat 2](#_Toc58360631)

[2. Felhasználói dokumentáció 4](#_Toc58360632)

[2.1 Rendszerkövetelmények 4](#_Toc58360633)

[2.2 Telepítés 5](#_Toc58360634)

[2.2.1 Első indítás 5](#_Toc58360635)

[2.2.2 Már inicializált adatbázissal 7](#_Toc58360636)

[2.3 Content management system felhasználói felület 8](#_Toc58360637)

[2.3.1 Bejelentkezési képernyő 8](#_Toc58360638)

[2.3.2 Kezdő képernyő 9](#_Toc58360639)

[2.3.4 Leckék szerkesztése 9](#_Toc58360640)

[2.3.5 Feladatok szerkesztése 12](#_Toc58360641)

[2.3.6 Nyitó oldal szerkesztése 12](#_Toc58360642)

[2.4 Alkalmazás felhasználói felület 13](#_Toc58360643)

[2.4.1 Kezdő képernyő 13](#_Toc58360644)

[2.4.2 Lecke nézet 14](#_Toc58360645)

[3. Fejlesztői dokumentáció 18](#_Toc58360646)

[3.1 Alkalmazás felépítése 18](#_Toc58360647)

[3.2 Statikus megjelenési tartalom 19](#_Toc58360648)

[3.2.1 Könyvtárszerkezet, komponensek 19](#_Toc58360649)

# 1. Bevezetés

## 1.1 Témaválasztás indoka

A koronavírus által bekövetkezett helyzet rámutatott, hogy milyen nagy mértékű szükségünk van az online felületen is elérhető tananyagokra. Ebből az okból írtam, mint szakdolgozatomként, egy online felületen elérhető oktató jellegű alkalmazást. A szakdolgozatom célja, hogy egy olyan környezetet biztosítson a c++ programozási nyelvnek, hogy azt könnyen fogyasztható formában lehessen elsajátítani.

Az alkalmazásban a c++ programozási nyelv ismereteit leckék által lehet elsajátítani. A leckékhez feladatok is tartoznak, így a felhasználó egyből tudja gyakorolni a frissen szerzett tudását, amellyel sokkal jobban rögzíteni is tudja azt a már elsajátított tudást az emlékezetében. A leckék különböző nehézségi szintek szerint vannak beosztva, így a felhasználó a saját képességeihez mérten választhat a tananyagokból, és fokozatosan fejlesztheti tudását. A leckék egy diasorszerű felületen vannak prezentálva, amik között a felhasználó a navigációs gombok segítségével lépegethet az anyagban.

Mind amiatt, hogy a tudás jobban rögzülhessen a tanuló személyében, az elméleti anyagok mellett gyakorlatban is kipróbálhatja, gyakorolhatja a már megszerzett tudását. Az alkalmazás ugyanis tartalmaz egy online fordítót is, amivel a lecke tanulása alatt kipróbálható az újonnan megszerzett ismeret, illetve a programozási feladatokban próbára tehető az elsajátított tudás is. Ennek működéséről a dokumentáció későbbi szakaszában lesz szó.

## 1.2 Megoldandó feladat

A projekt célja, hogy a végfelhasználónak ne kelljen semmilyen extra programot telepítenie, ezáltal kényelmesebben lehessen használni, és sokak által hozzáférővé váljon. Egy modern böngészőre legyen szüksége mindösszesen, hogy elérhető legyen számára az oktató felület. A kódfordításhoz nem kell, hogy c++ fordítóprogram legyen az eszközünkön, ez mind szerveroldalon történik.

A kód fordítása és futtatása egy külön Docker konténerben történik, amely szabályozható életidővel rendelkezik. Mindez biztonság, pontosabban elővigyázatosság miatt, ugyanis ezáltal ki tudjuk küszöbölni, hogy esetlegesen egy káros kód fusson le a szerveren, amely kártékony hatásokat hajthat végre a szerver rendszerében.

Az oldal irodalmi tartalmát egy headless cms (content management system) szolgáltatja. Ennek tartalma dinamikusan változtatható egy külön felhasználói felületen, ahol akár több szerkesztő felhasználót is létrehozhatunk. Ezen a felületen tudunk új leckéket és feladatokat is létrehozni. Ez a rendszer tartalmaz egy publikációs megoldást is, ami segítséget nyújt, hogy csak az a tartalom jelenjen meg az alkalmazásban, aminek már végeztünk a szerkesztésével.

A projekt készítése során próbáltam egy olyan tanulási módszertant követni, miszerint a felhasználó nem csak megfigyelő a lecke során, hanem aktívan ki is próbálhatja az új ismereteket, és a saját tempójában haladhat az anyaggal.

# 2. Felhasználói dokumentáció

A teljes alkalmazás dockerizálva van, így ha lokálisan szeretnénk futtatni, nincs szükségünk nodeJS-re, vagy egyéb rendszerekre. Két alkalmazást kell telepítenünk, [Docker Desktop](https://www.docker.com/products/docker-desktop), illetve [Postman](https://www.postman.com/downloads/) megnevezésű alkalmazásokat. A Docker biztosítja már azt a lehetőséget is, hogy a virtualizációhoz Windows-os konténereket használhassunk, viszont ennek ellenére sokkal jobban ajánlott a Linux alapú konténereket választani, ugyanis ilyen alappal lett maga az alkalmazás is fejlesztve, ezen okból nem várt problémák adódhatnak ellenkező esetben, pontosabban akkor, amikor is a nem Linux alapú konténereket használjuk. A Postman az adatbázis inicializálásának megkönnyítéséhez szükséges.

## 2.1 Rendszerkövetelmények

Docker Desktop for Windows esetén:

* Windows 10 (64-bit) Pro, Enterprise, vagy Education:
  + Build 16299 vagy nagyobb
  + Hyper-V és a Containers Windows funkció engedélyezve kell hogy legyen.
  + Hardveres követelmények:
    - 64 bit-es processzor SLAT támogatással
    - 4GB memória
    - BIOS-szintű hardveres virtualizáció engedélyezése, [bővebben](https://docs.docker.com/docker-for-windows/troubleshoot/#virtualization-must-be-enabled)
* Windows 10 (64-bit) Home:
  + 1903-as vagy nagyobb verzió
  + WSL 2 funkciók engedélyezése, [dokumentáció](https://docs.microsoft.com/en-us/windows/wsl/install-win10)
  + Hardveres követelmények:
    - 64 bit-es processzor SLAT támogatással
    - 4GB memória
    - BIOS-szintű hardveres virtualizáció engedélyezése, [bővebben](https://docs.docker.com/docker-for-windows/troubleshoot/#virtualization-must-be-enabled)

Illetve egy modern böngésző, az alábbi verzió számokkal vagy nagyobbal:

* Chrome (86.0.4240.198)
* Edge (87.0.664.55)
* Firefox (82.0.2)

## 2.2 Telepítés

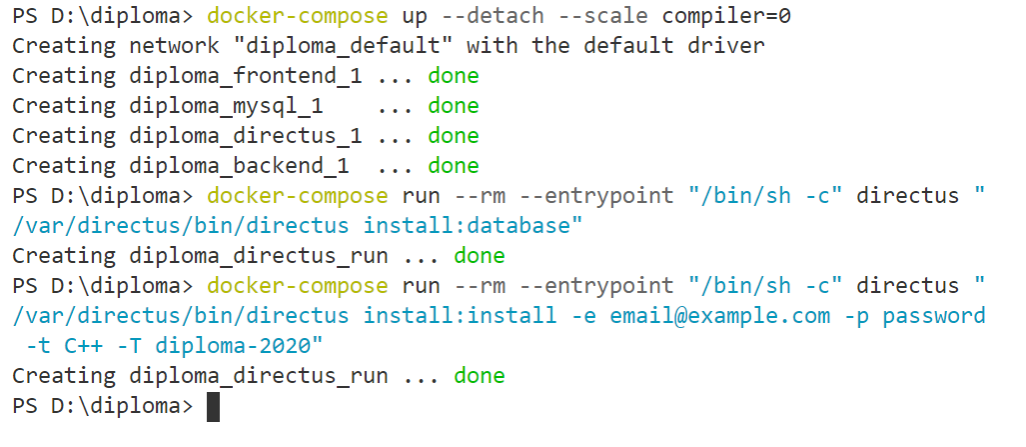
Miután sikeresen feltelepítettük a Docker Desktop és Postman alkalmazást, a projektet a következő képen tudjuk elindítani.

### 2.2.1 Első indítás

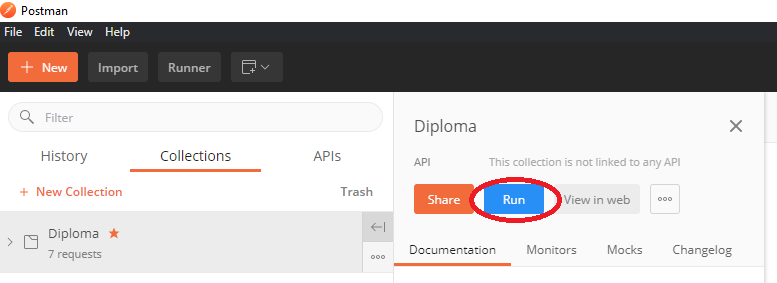
Az elsőindítás kicsit bonyolultabban végezhető el, mint az elkövetkezendő többi indítás, ugyanis inicializálnunk kell az adatbázist. Az adatbázis volume-ok segítségévelvan fenntartva futások között, így élettartama a konténerekétől független. Mindebből az következik, hogy a későbbi indításokkor már nem kell az összes következő lépést végrehajtanunk.

Lépések:

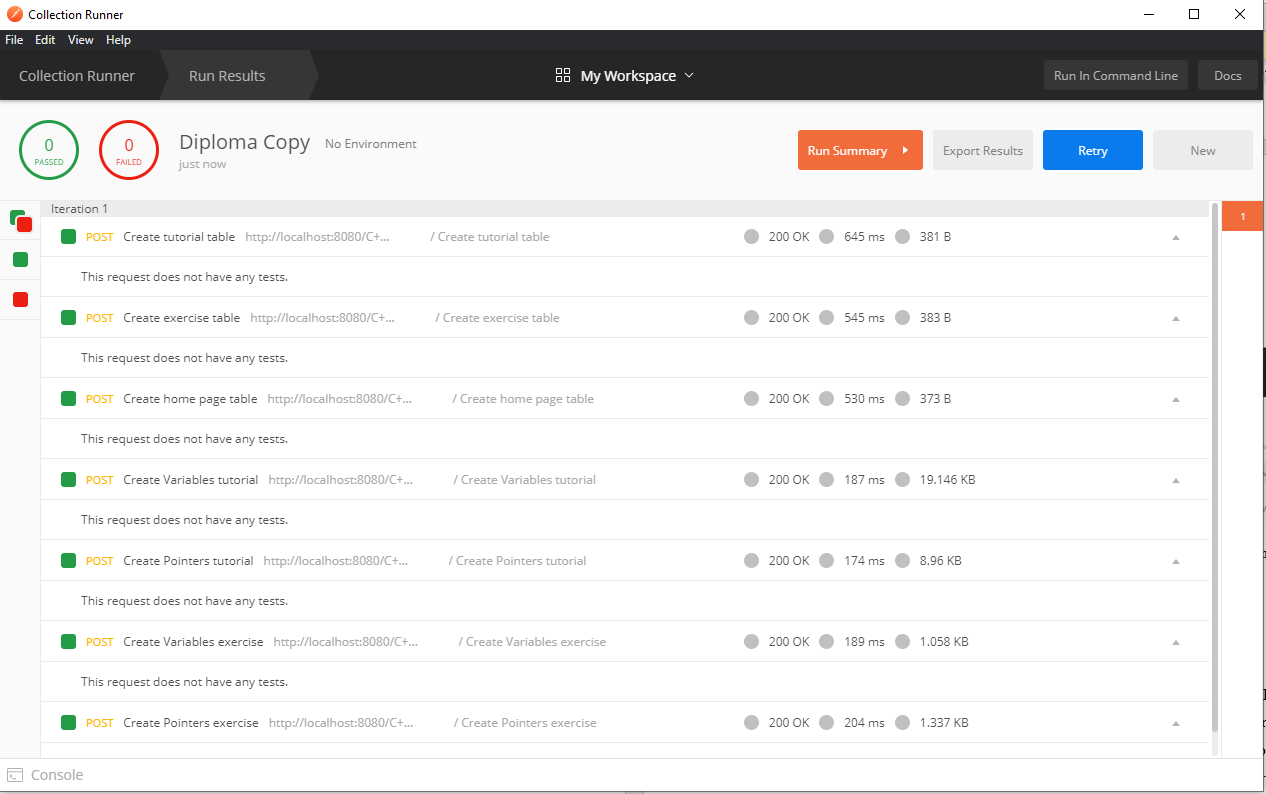
* Service-ek elindítása és adatbázis inicializálása
  + *docker-compose up --detach --scale compiler=0*
  + *docker-compose run --rm --entrypoint "/bin/sh -c" directus "/var/directus/bin/directus install:database"*
* Admin felhasználó létrehozása
  + *docker-compose run --rm --entrypoint "/bin/sh -c" directus "/var/directus/bin/directus install:install -e <email> -p <password> -t C++ -T <access-token>”*
  + Itt az <email> és <password> helyére tetszőleges értékeket írhatunk be.
  + Az <access-token> helyére diploma-2020 értéket írjunk, [lentebb](#access_token) olvasható mi a teendő, ha más értéket szeretnénk

Példa:

1. ábra sikeres indítás konzol kimenetele

* Adatbázis feltöltése
  + Nyissuk meg a Postman alkalmazást
  + Importáljuk be a ./cms/postman\_collection file-t, majd futtassuk le

2. ábra collection futtatása a postman alkalmazásban

* + Állítsunk be 1 másodperces késést a kérések között. Ezt az eredményt kell látnunk: 

3. ábra collection futtatásának eredménye a postman alkalmazásban

* Ha ezeket a lépéseket elvégeztük az alkalmazás futásra kész állapotba került.

Saját access-token érték esetén:

A ’diploma-2020’ értéktől eltérő access token esetén a következő lépéseket kell megtennünk:

* A ./frontend/.env fájlban a **REACT\_APP\_ACCESS\_TOKEN** kulcs értékének a telepítéskor megadott <access-token> értéket kell adni.
* Újrabuildelni a frontend service-t a *docker-compose build frontend* paranccsal
* Leállítani a jelenlegi serviceket (*docker-compose down*)
* Majd újraindítani (*docker-compose up --detach --scale compiler=0*)

### 2.2.2 Már inicializált adatbázissal

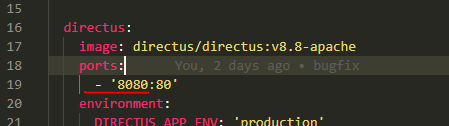
Ha már korábban elvégeztük 2.2.1 lépéseit, a következőképpen tudjuk elindítani az alkalmazást:

* *docker-compose up --detach --scale compiler=0*

## 2.3 Content management system felhasználói felület

Az alkalmazásban megjelenő tartalmat ezenen a felületen keresztük tudjuk szerkeszteni, illetve bizonyos entitások státuszát változtatni.

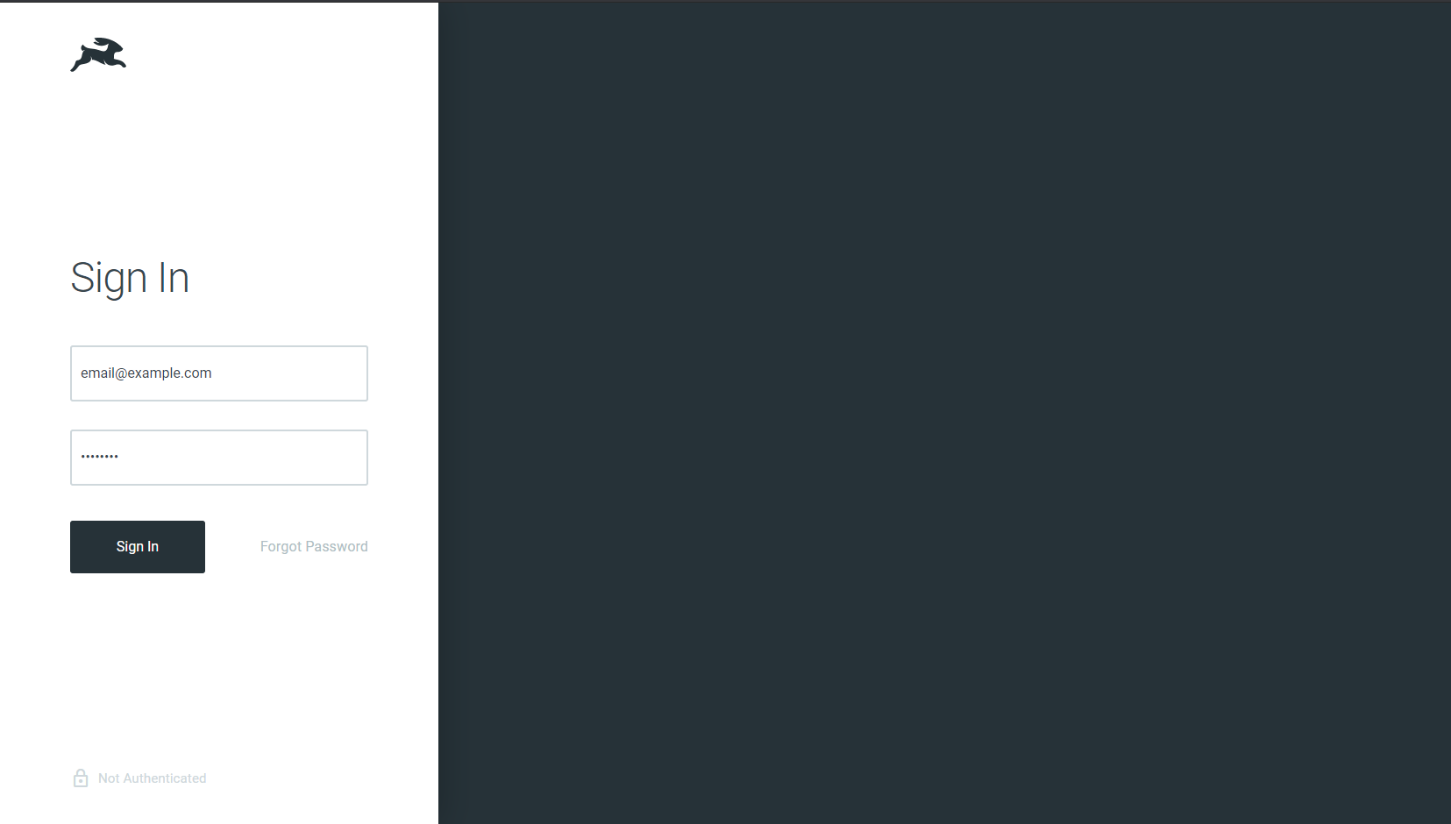
A felületet az alapbeállításokkal a szerver 8080-as portján tudjuk elérni. Esetünkben ez a lokális szerver, tehát az elérési útvonal a következő: <http://localhost:8080>.

A port számot módosíthatjuk, ha a projekt gyökér szintén levő *docker-compose.yml* file-ban átírjuk az alábbi értéket:

4. ábra részlet a docker-compose.yml fájlból

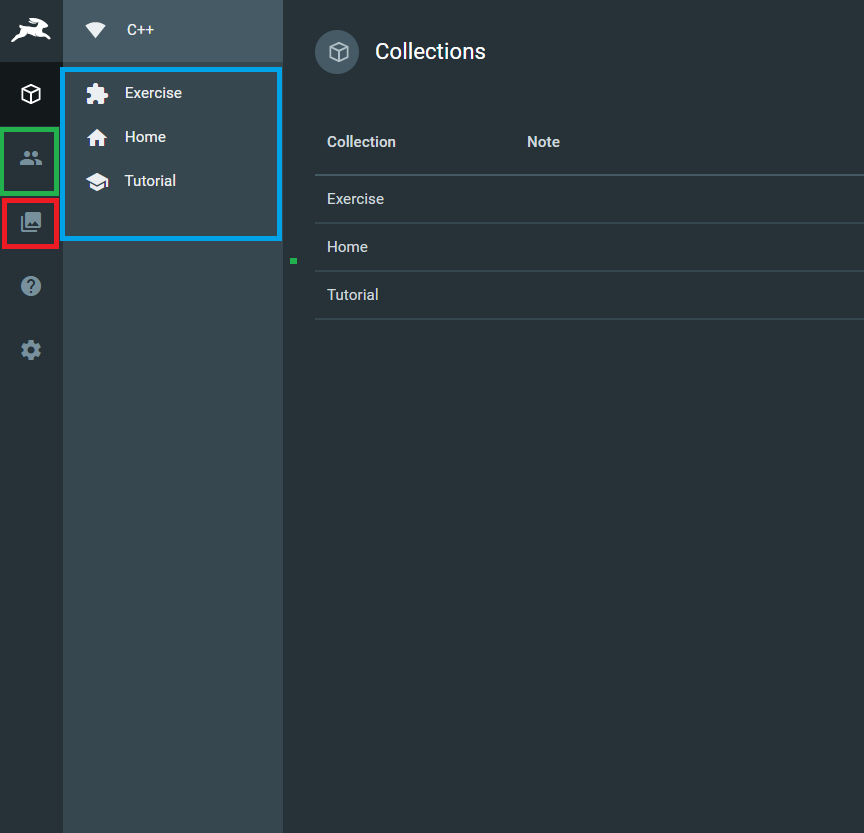
### 2.3.1 Bejelentkezési képernyő

A következő felülettel találkozhatunk, ha meglátogatjuk a fentebb található linket. Itt a telepítésnél megadott email és jelszó párossal léphetünk be. A rendszer sütik segítségével megjegyzi a belépési adatainkat, így azokat nem kell minden alkalommal újra megadnunk.



. ábra CMS belépő képernyő

### 2.3.2 Kezdő képernyő



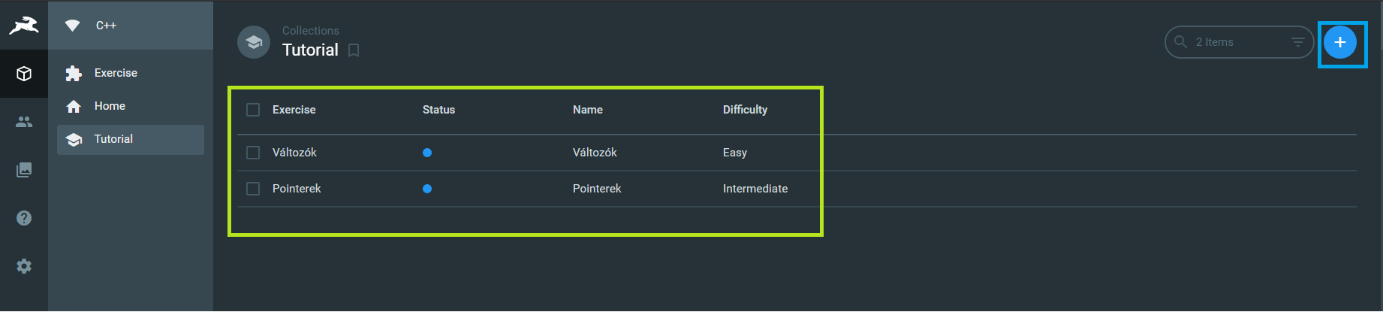
. ábra CMS főmenü

A kékkel jelölt dobozban láthatjuk a collection-öket, ezek az egyes elemek csoportosítására szolgálnak. Itt tudjuk az elemeket kilistázni, bizonyos elemekre szűrni, illetve új elemet hozzáadni. Fontos megjegyezni, hogy a Home collection egyke, ami annyit jelent, hogy a másik kettővel ellenkezőleg csak egy darab példány létezhet. Rákattintva egyből annak az egy elemnek a szerkesztő képernyője jelenik meg.

Zölddel látható a felhasználók kezelésére szolgáló felület, itt lehet új felhasználókat hozzáadni, vagy meglévők adatait, státuszukat szerkeszteni.

Piros színnel pedig a cms-be feltöltött fájlokat-, média tartalmakat tekinthetjük meg, illetve adhatunk hozzá újakat.

### 2.3.4 Leckék szerkesztése



. ábra CMS-ben kilistázott leckék

Két féle képen tudjuk megjeleníteni a részletes szerkesztői nézetet. A kék színnel jelölt hozzáadás gomb megnyomásával, ekkor egy új leckét hozunk létre. Illetve, ha a kilistázott elemek egyikére rákattintunk. A státusz oszlopban a kék pötty jelzi azt, hogy az adott lecke publikálva van, és listázva van az alkalmazásban. Piszkozat esetén ez szürkével jelölt.

Egy leckének nyolc szerkeszthető mezője van:

* status
* name
* url\_alias
* difficulty
* icon
* description
* slides
* exercise

Status:

Három választási lehetőségünk van, *Published, Draft, Deleted*. Draft esetén nincs listázva az adott lecke az alkalmazásban. Fontos, hogy csak olyan leckét állítsunk Published-re, amihez a feladatok részét is elkészítettük már, és össze is kötöttük a kettőt az exercise mező segítségével a lecke szerkesztői képernyőn. Ez egy lenyíló mező, ahol kiválaszthatjuk név alapján a megfelelő feladatot.

Url\_alias:

Ez egy kötelező szövegmező. Az angol ábécé kis betűi, illetve kötőjel használható benne. Ez a szöveg fog megjelenni a böngésző url részében, ha megnyitjuk a leckét.  
Példa:



. ábra url alias megjelenése

Difficulty:

Ez egy kötelező lenyíló mező, ahol a lecke nehézségi szintjét adhatjuk meg. Három lehetőségünk van *Easy, Intermediate, Professional*. Azt befolyásolja ez a mező, hogy a menüben melyik kategória alá kerüljön a lecke, illetve jelzést is az a felhasználónak a lecke nehézségéről.

Icon:

9. ábra lecke kártyák

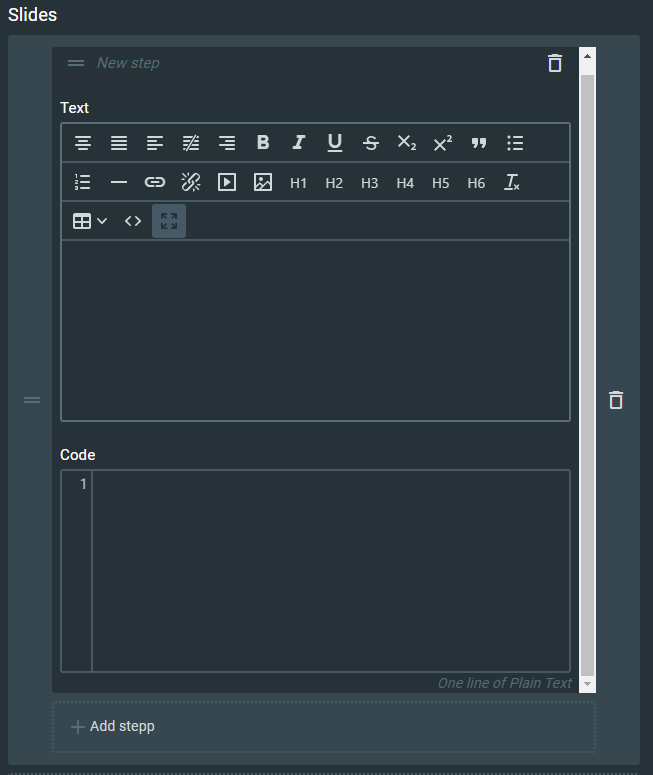
Opcionális mező, a leckekártyán megjelenő illusztráció. Ha nem adjuk meg, helyét a nehézséghez rendelt szín tölti ki.

Description:

Kötelező szövegmező, ami összefoglalja a lecke lényegét, és érdeklődést kelt fel. A leckekártyán jelenik meg, illetve a megnyitott lecke neve alatt. 

10. ábra lecke leírás és cím

Slides:

Ez egy összetett mező, itt lehet megadni a tartalom lényegi részét. A lecke anyaga egy slideshow szerű felületet van prezentálva. Két entitásunk van, slide és step. Egy slide-on belül több step-ünk is lehet, és egy step-ben lehet kódrészlet vagy szöveges tartalom. A slideshow-n van egy léptető, amivel a következő step-et jeleníthetjük meg, vagy visszaléphetünk az előzőre. Ha elszeretnénk tüntetni az előző step-eket a slideshow-ról, kezdjünk egy új slide-ot és arra rakjunk új step-eket. A szöveg valójában html tartalmat hoz létre, tehát be lehet szúrni képeket és multimédiás tartalmat is, továbbá különböző méretű címsorokat. Van egy teljes képernyős gomb is, amivel megkönnyebbíthetjük a szerkesztési élményt. Ez a fenti ábrán a világosabb szürke hátterű gomb. Fontos, hogy igaz két mezőt látunk egy step-en belül, de csak az egyikbe írjuk tartalmat, így elkerülve a sorrendiségi problémákat. A step-eket, illetve a slide-okat átrendezhetjük, ha áthúzzuk a sorrendjüket a baloldali sáv segítségével.

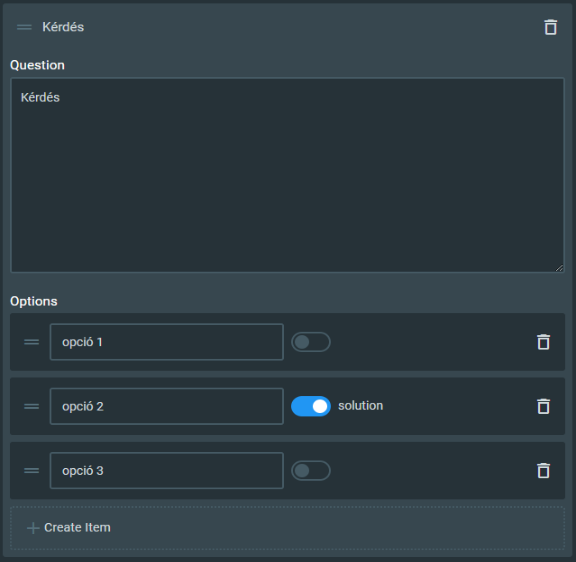
11. ábra slide szerkesztése

Exercise:

Lenyíló mező, a rendszerben tárolt leckéket listázza ki név szerint.

### 2.3.5 Feladatok szerkesztése

Meglévő feladatot ugyanúgy szerkeszthetünk vagy hozhatunk létre, mint leckék esetén. Egy leckéhez tartozó feladat kétfajta feladatból állhat.

**Kvíz feladatok**, ezeknek száma 0-tól a kívánt mennyiségig terjedhet. Nem csak egy helyes válasz lehetséges. Egy kis kapcsolóval lehet állítani, hogy a választási lehetőség helyes-e. Sajnos a cms rendszerben van egy ismert hiba, nem menti el a kezdetben hamisnak jelölt és úgy is maradt opciókat. Ennek egy egyszerű, de kissé idegesítő megoldása, ha egyszerűen igazra, majd hamisra állítjuk a kis kapcsolót.

**12**. ábra kvíz szerkesztése

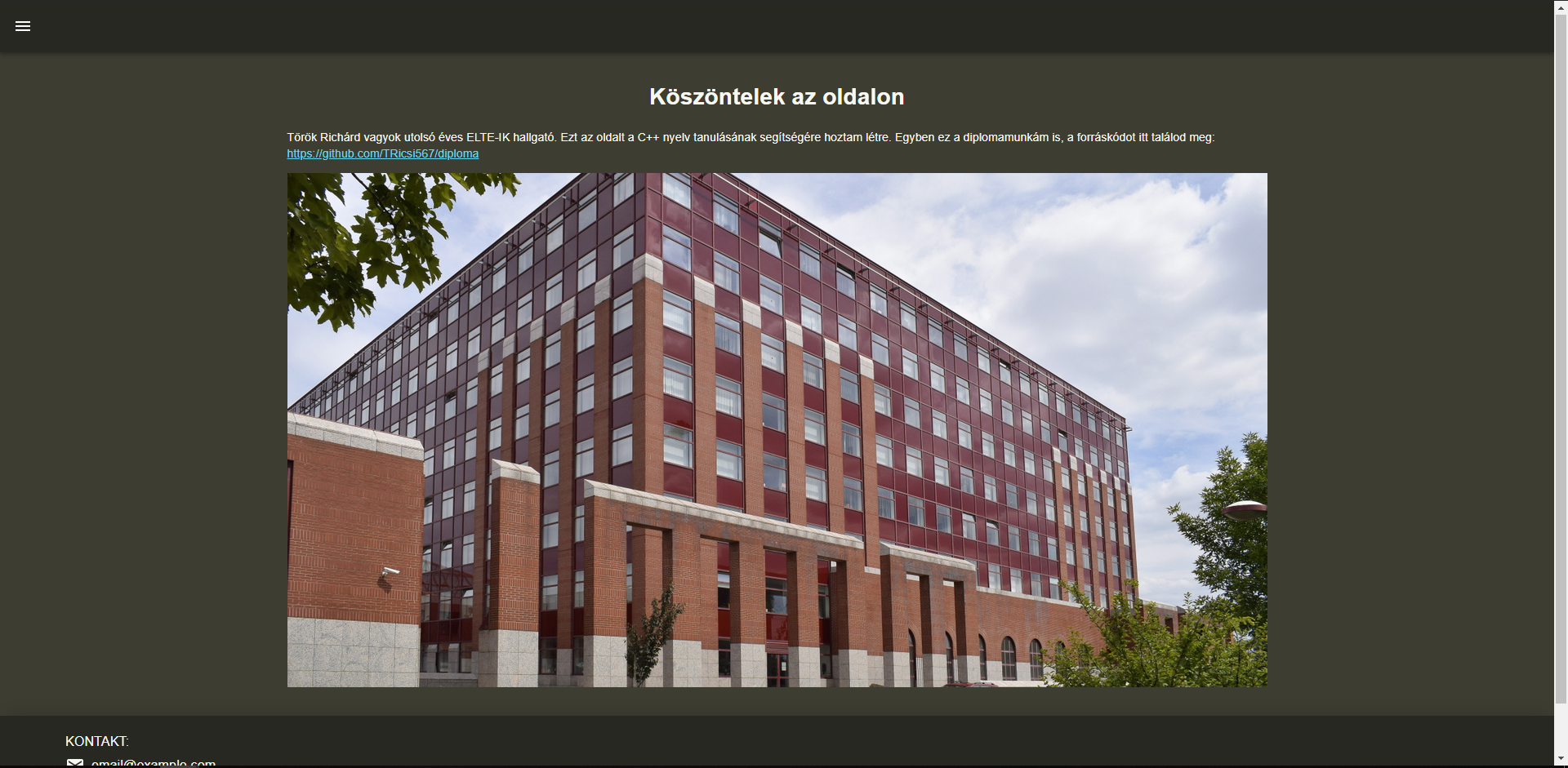
**Programozási feladat,** ebből feladatsoronként egy létezhet. Ennek az entitásnak három mezője van, *question, description, solution*. A description kivételével a többi kötelező mező. Az opcionális description mezőbe írhatunk tippeket a feladat megoldásához, vagy feladat specifikációt. Fontos kihangsúlyozni, hogy a rendszer úgy ellenőrzi a feladatot, hogy a program output-ját hasonlítja össze a solution mezőbe írt értékkel. Itt a betűméret nem számít, a program átalakítja a program kimenetét és a megoldás szövegét egyaránt kisbetűssé.

### 2.3.6 Nyitó oldal szerkesztése

A home collection alatt tudjuk szerkeszteni a kezdőképernyőn megjelenő tartalmat, a láblécben megjelenő kontakt emailt, és telefonszámot, valamint az oldalsó menüben megjelenő ’hasznos linkek’ szekciót. Ha nem adunk meg egyetlen linket sem, akkor maga a szekció címkéje se fog megjeleni az oldalsó menüben. Maga a kezdőoldalon megjelenő tartalom a *content* mezővel szerkeszthető, ez is egy html tartalmat generál, el lehet helyezni benne különböző méretű címsorokat, multimédiás tartalmakat, képeket, és még sok mást.

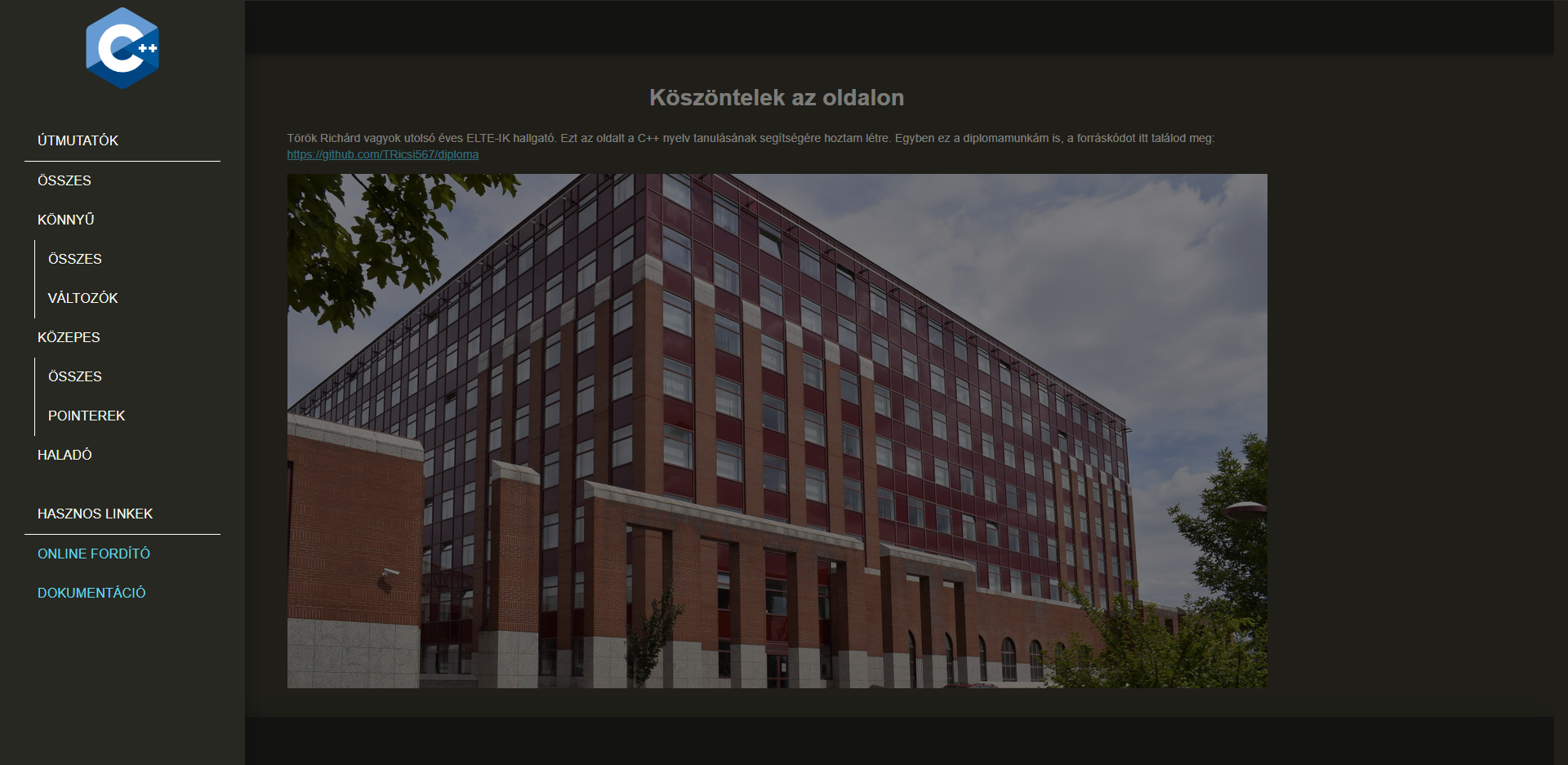
## 2.4 Alkalmazás felhasználói felület

### 2.4.1 Kezdő képernyő



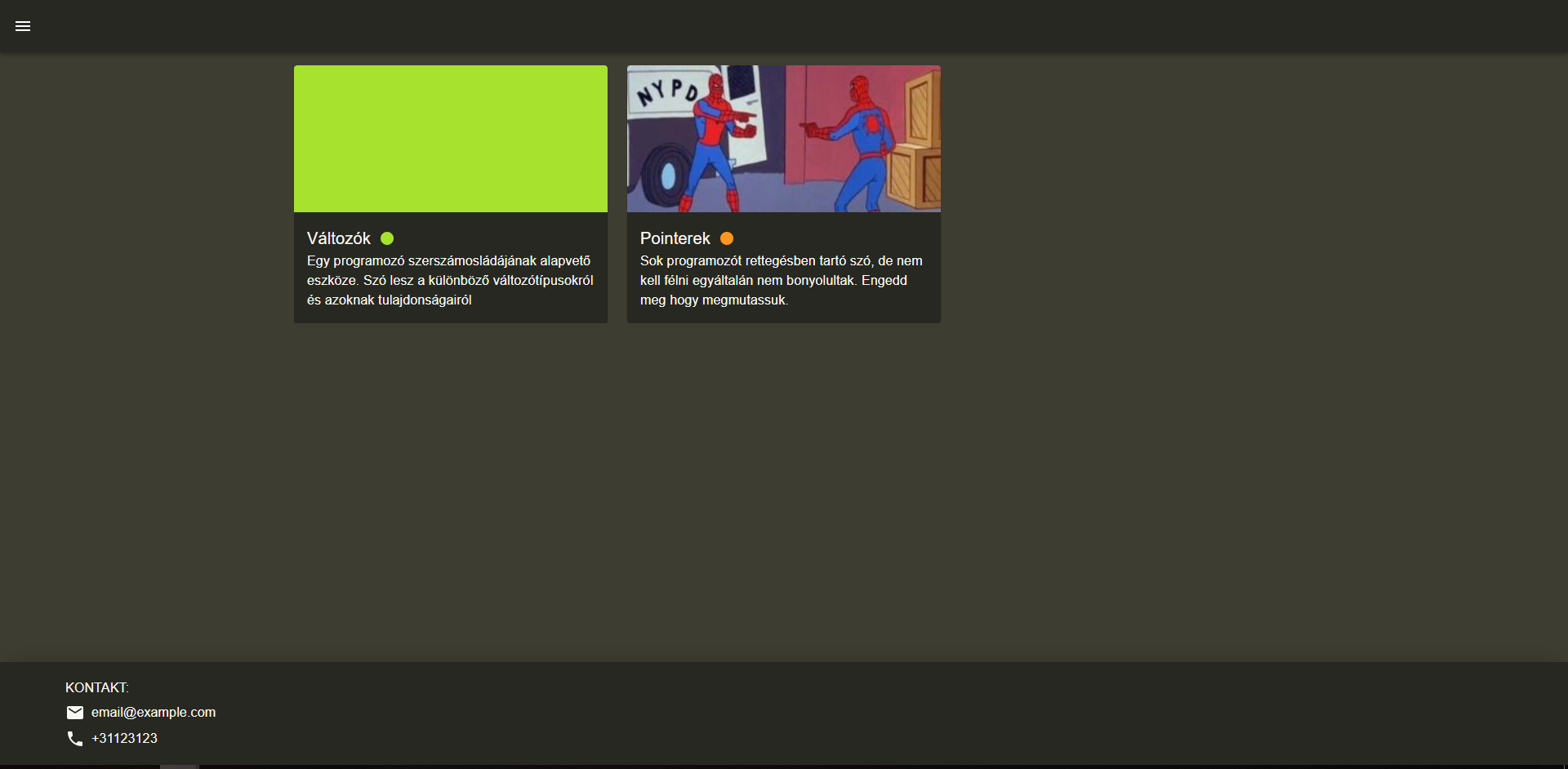
. ábra kezdő képernyő

Az alkalmazást betöltve az 13. ábrához hasonló nézettel találkozunk. Itt a tartalom a szerkesztőtől függ. Ezen az oldalon különböző cikkeket olvashatunk, amit a szerkesztő létrehoz, vagy esetleg más hivatkozásokat is hozzáadhat. Alul a láblécben találtható a kontakt email és telefonszám. Ezek linként is szolgálnak, rájuk kattintva egyből elindul a számítógépünkön alapértelmezett levelező, vagy telefonhívást végző alkalmazás. A bal felső sarokban található a menü gomb. A menüt megnyitva a következő nézetet kapjuk:



. ábra kinyitott menü

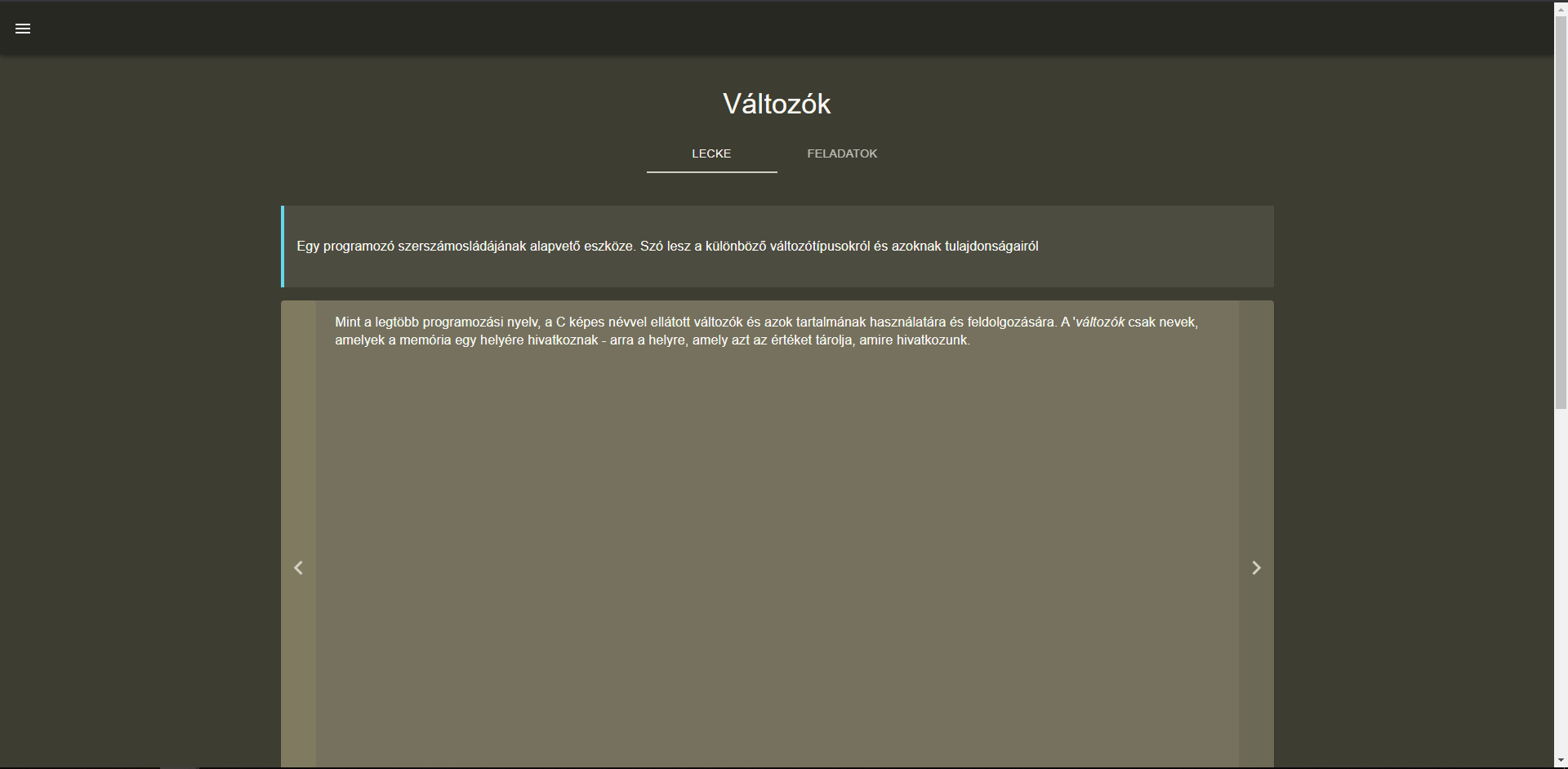
A C++ ikonra kattintva visszajutunk a főoldalra. Az útmutatók címke alatti könnyű, közepes és haladó menüpontok valójában összecsukható almenük, amik újabb menüpontokat nyitnak meg. Bármely kategóriának az ’összes’ menüelemére kattintva a 15. ábrán is látható összegző oldalra jutunk. Konkrét leckére kattintva a választott tananyaghoz navigál az applikáció.



. ábra összegző képernyő

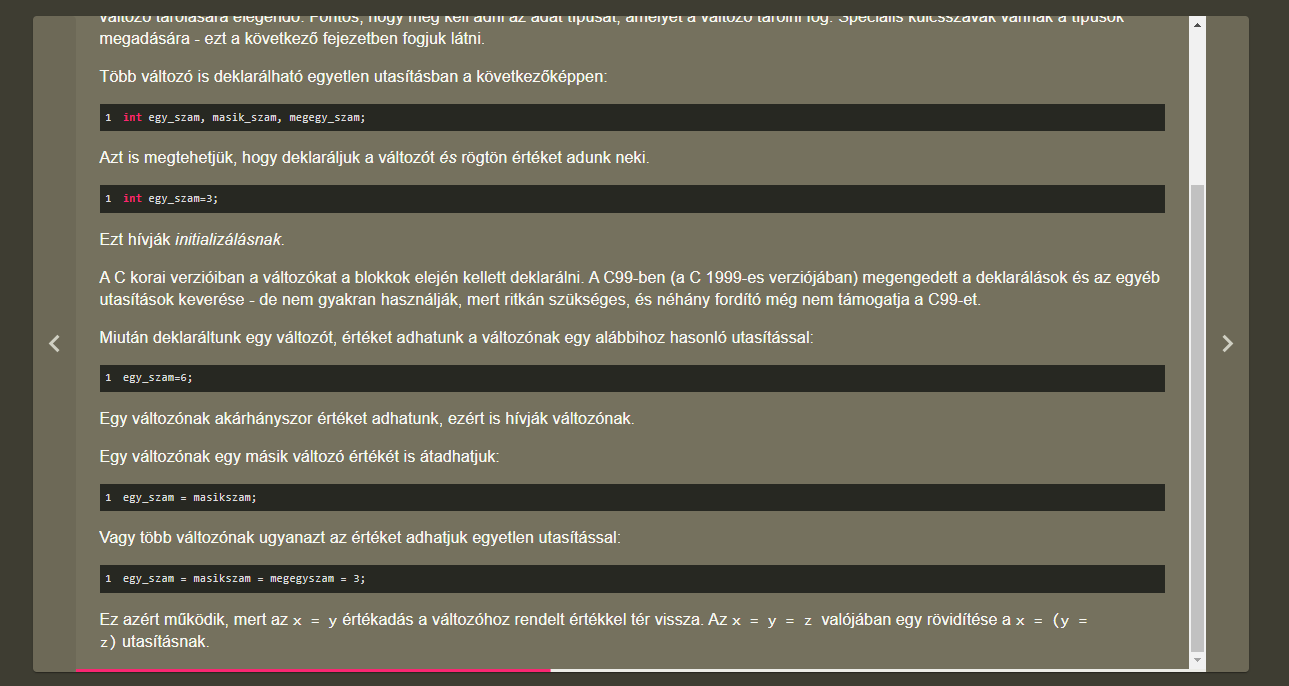
### 2.4.2 Lecke nézet

A főoldalhoz hasonlóan itt is megmarad az oldal szerkezete, fejléc benne a menüvel és a lábléc. Az oldal tartalma változik csak.



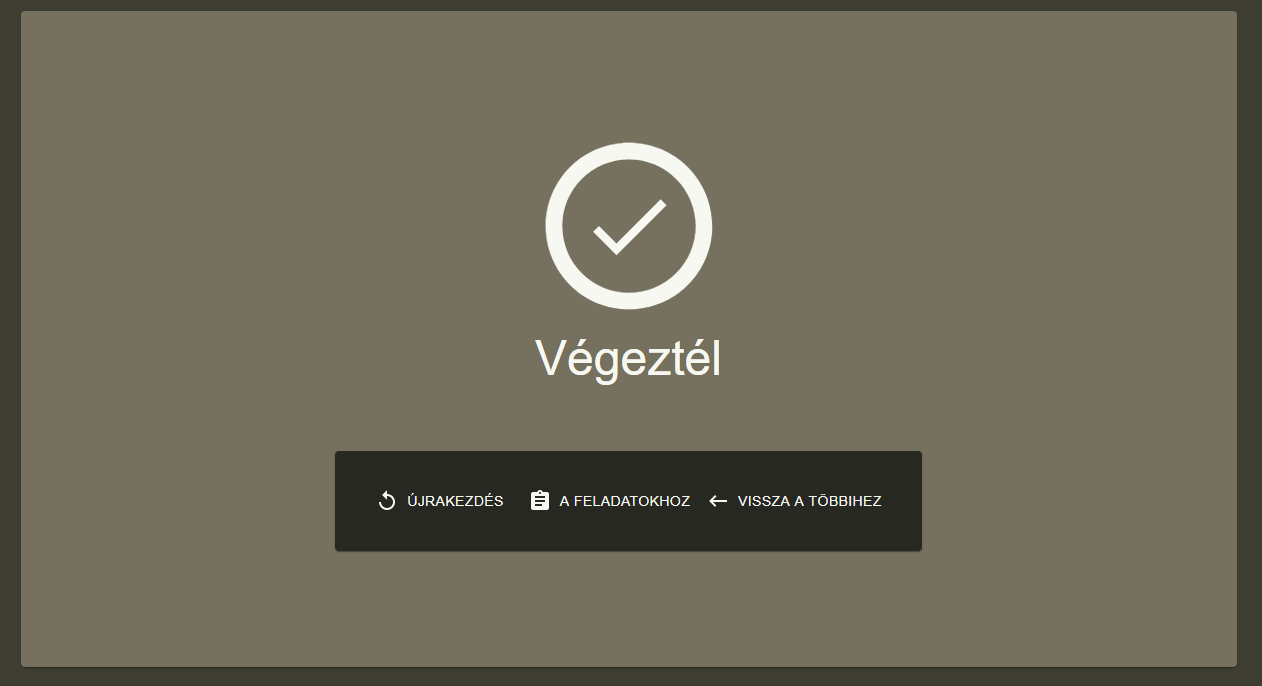
. ábra lecke nézet

Az első sorban a lecke címe található, alatta egy menü, amivel válthatunk a lecke és feladatok nézet között. A menü alatt pedig a lecke leírása található. Itt az egész szöveg megvan jelenítve, mivel előfordulhat, hogy a leckekártyán a szöveget levágjuk ha az túl hosszú. A leírás alatt pedig a diasorok találhatóak, amiben a lecke anyaga van. Itt a nyilakkal jelölt gombokkal lehet előre vagy hátra haladni az anyagban.



. ábra diasor

Alul egy rózsaszín csík jelzi hol tartunk éppen az anyagban. Ha az anyag végére értünk a következő képernyőt láthatjuk:



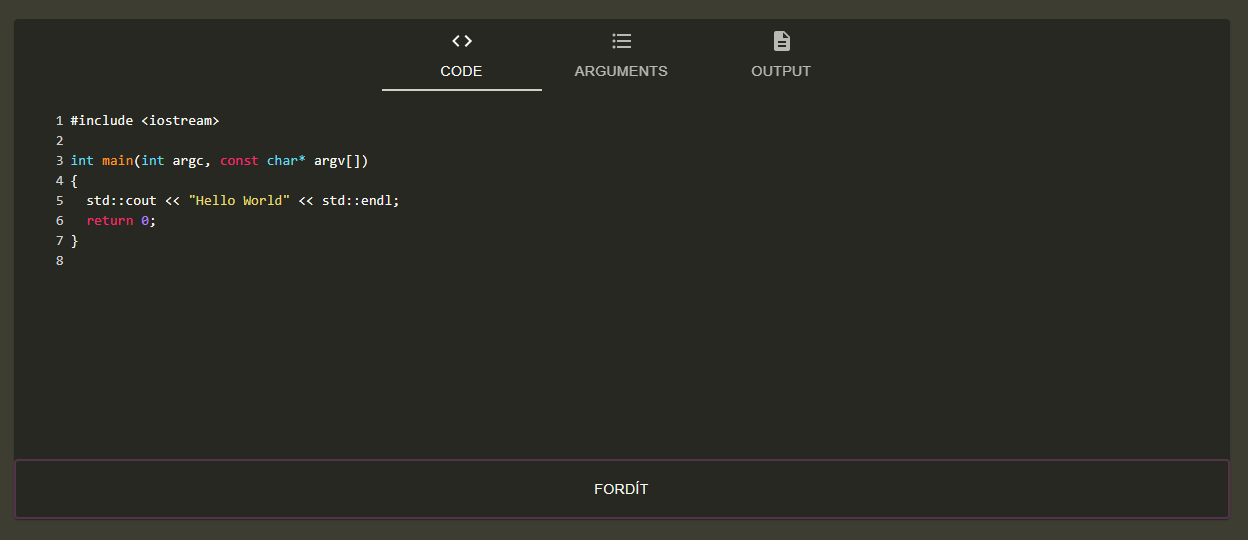
. ábra befejezett lecke

Három opciót kínál fel nekünk a felület. Újrakezdhetjük a tananyagot az elejétől, a feladatok szekcióra ugorhatunk, vagy visszamehetünk a jelenlegi lecke szintjével megegyező összegző képernyőre.

A diasor alatt található egy online kódfordító rész, ez a 19-es ábrán láthatjuk. Ez három fülből áll, code, arguments és output. A code fülön tudjuk szerkeszteni a forráskódot, amit majd le szeretnénk futtatni, az arguments fül alatt adhatjuk meg az argumentumokat, amiket a program kap meg futtatáskor. Az output fülön pedig a program futásának kimenetelét láthatjuk. Ha hibás a forráskódunk és fordítási hibánk van azt is ezen a fülön láthatjuk. A fordít gomb lenyomásával indíthatjuk el a fordítási folyamatot, ameddig ez fut a gomb inaktív állapotba kerül. Ha végzett a folyamat átvált automatikusan a output fülre.

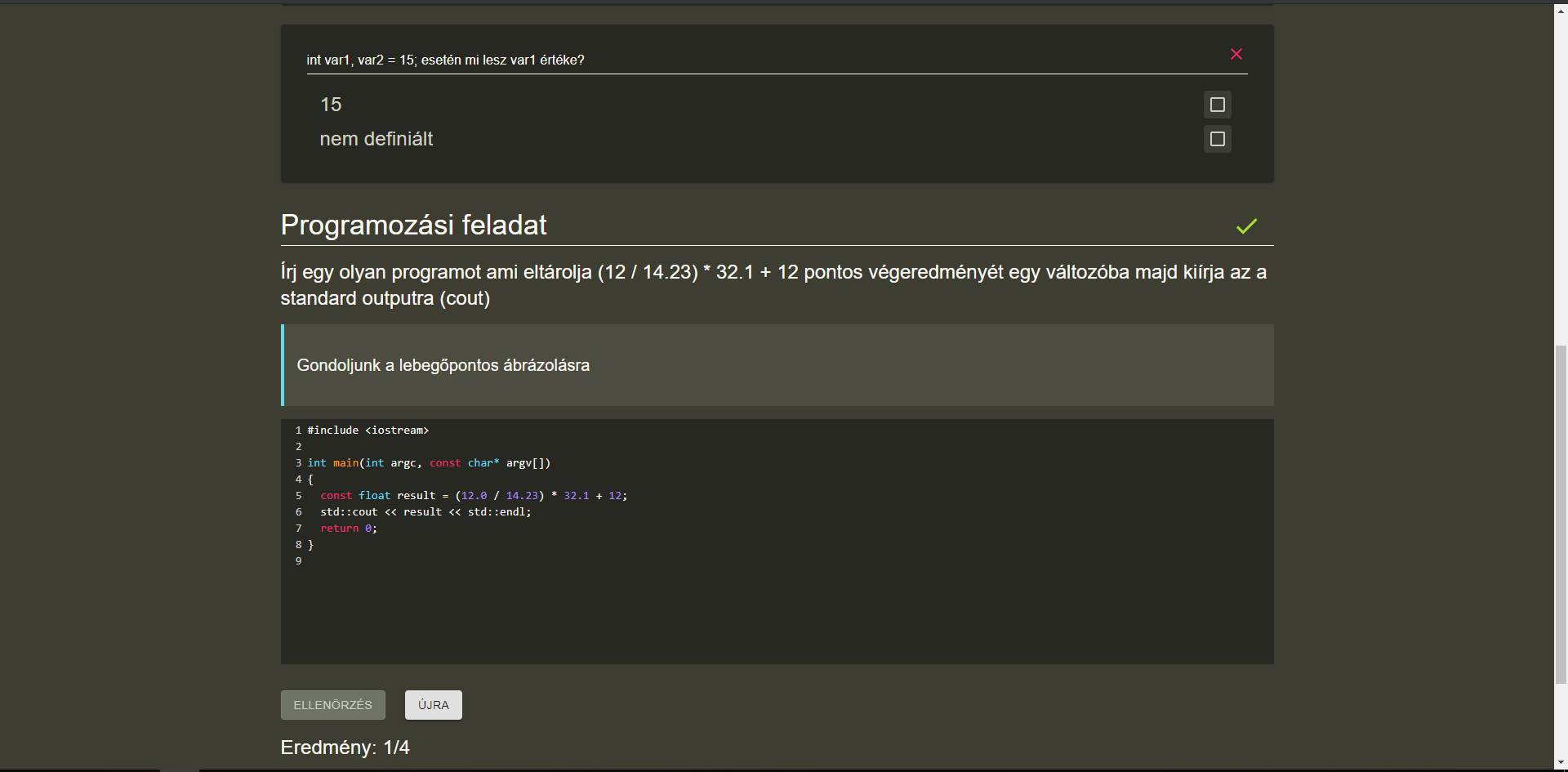
Három lehetséges végződése van a folyamatnak:

* Sikeresen lefut a program, ekkor a program kimenetele kerül kiírásra
* Fordítási hibánk van, ekkor a fordító által írt hibaüzenetet írjuk ki
* A folyamat túllépi az időkorlátot, ekkor a *timed out* üzenetet írjuk ki



. ábra online fordító

A feladatokat nézetre kétféle módon tudjuk eljutni, ha a fenti almenüben a ’Feladatok’ menüpontot választjuk (16. ábra), vagy miután végeztünk a leckével a középső ’A feladatokhoz’ opcióra kattintunk. Először a kvízek vannak kilistázva, ha az adott leckéhez tartozik legalább egy darab, utána pedig a programozási feladat.



. ábra leellenőrzött feladatmegoldás

Az ellenőrzés gombra kattintva, elküldjük a válaszainkat kiértékelésre. Ameddig ez folyamatban van az ’újra’ és ’ellenőrzés’ gomb is inaktív állapotba kerül. Ha végzett a folyamat a kiértékeléssel minden kérdést megjelöl egy zöld pipával vagy piros ikszel a válasz helyességétől függően. Erre a 20-as ábrán láthatunk példát. A gombok alatt pedig kiírja mennyi kérdésre adtunk jó választ az összesből. Az újra gomb megnyomásával a feladatlap visszaáll az eredeti állapotába és újból próbálkozhatunk. Célszerű megjegyezni, ha feladatok és lecke fül között váltakozunk, a feladatlap mindig visszaáll a kezdetleges állapotába, így nem lehet olyan könnyen lenézni a választ a diasorról.

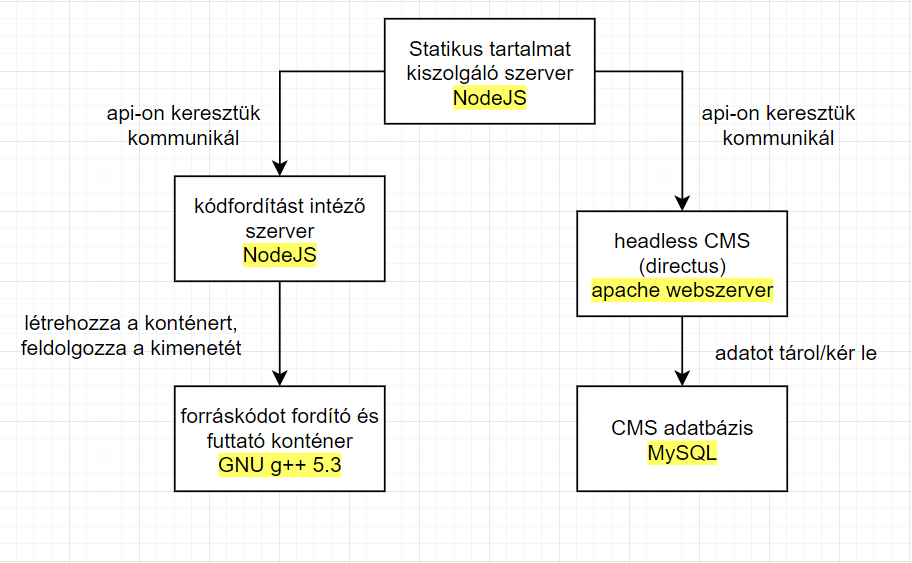
# 3. Fejlesztői dokumentáció

## 3.1 Alkalmazás felépítése

Az alkalmazás minden modulja egy Docker konténerben van elhelyezve, így könnyítve a jövőbeli a skálázhatóságot, és az éles környezetbe való helyezést. A forráskódban több docker-compose fájl is szerepel, ha esetleg a teljes alkalmazás egy kisebb részét szeretnénk lebuildelni, vagy futtatni. A projekt gyökér szintjén található a fő compose fájl. Az alkalmazásban szereplő összes modul/service megtalálható benne.

A compose fájlok fontosságát érdemes kiemelni, ugyanis ezek sok kényelmi funkcióval szolgálnak, mint például volume-ok helybeli létrehozása, amivel a futások közti adatokat tudjuk fenntartani, vagy például egy alap hálózat, amin keresztül a konténerek tudnak kommunikálni, és egyfajta címfordítást is biztosít, hogy könnyebben tudjuk megadni az elérési címeket más konténereknek. Emellett maga a konténerek példányosítását és konfigurálását is megkönnyíti.

Az alkalmazás architektúrája a következőképpen ábrázolható:



ábra az alkalmazás szerkezete

14.15.0-as NodeJS verziót használtam az ezt használó részalkalmazásokhoz. A build folyamat- és a futtatás során használ verzió megegyezik. A build egy ideiglenes Docker konténerben készül, nem szükséges, hogy saját eszközünk rendelkezzünk ezzel a pontos verziószámmal. Ezeknek a konténereknek 3.12-es verziójú alpine linux az operációs rendszere. Ez biztosítja a lehető legkisebb végleges image mértet.

A Docker image egy modellszerű fájl, aminek segítségével az image-ben meghatározott specifikációjú konténereket tudunk létrehozni.

## 3.2 Statikus megjelenési tartalom

A frontend elkészítéséhez a ReactJS keretrendszert használtam. A [create-react-app](https://www.npmjs.com/package/create-react-app) script által generált sablon projektet vettem alapul. Ez magába foglalja a Jest unit tesztelési keretrendszert, a Webpack bundle toolt, ami a build elkészítésében segít, illetve egy fejlesztői szervert is tartalmaz. Ez a *hot reload* tulajdonsága miatt hasznos, biztosítja a forráskód változása esetén az újra transpile-olást, és az új tartalom megjelenését a böngészőben. Így elkerülhetjük, hogy mindig nekünk kelljen kézzel újraindítani a szervert. A csomag tartalmazza még a BabelJS-t ami a fentebb említett transpile-olásához szükséges, hogy új még kísérleti fázisban, vagy bizonyos böngészőkben nem támogatott szintaktikát is használni tudjunk. A transpile-olás egy olyan folyamat amikor egy bizonyos javascript forráskódot azzal megegyező szemantikájú, de legtöbb esetben eltérő szintaktikájú kóddá alakítunk át.

A fejlesztés 16.13-as verziószámú React-al végeztem, de ennél újabb verziókkal is kompatibilis, de figyeljünk, hogy a react-scripts csomag verziószámát is frissítsük, ha újabb verziójú React-al szeretnénk a további fejlesztést végezni. Google által megalkotott Material formatervezés határozza meg a komponensek főbb stílusát. A legtöbb általános komponens ennek okán a [@material-ui](https://www.npmjs.com/package/@material-ui/core) csomagból származik. Az online forráskód szerkesztő részhez a [Codemirror](https://www.npmjs.com/package/codemirror) csomagot használtam, széles elterjedettsége, és egyszerű használata miatt. Jelenlegi állapotában a frontend projekt nem tartalmaz harmadik féltől származó csomagból eredő sebezhetőséget.

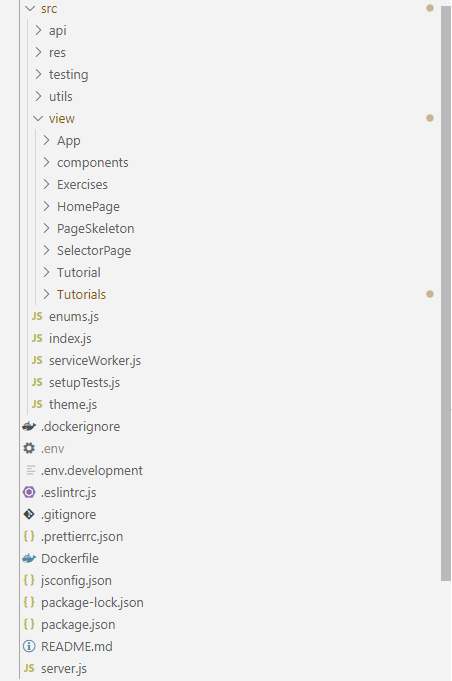
A kódformázási stílus betartását a [Prettier](https://www.npmjs.com/package/prettier) csomag segíti. Ennek konfigurációja a frontend mappa gyökerében található a .prettierrc.json fájlban. A statikus kódellenőrzést az [Eslint](https://www.npmjs.com/package/eslint) végzi. Ennek konfigurációja a .eslintrc.js fájlban található. Ez az eszköz felhívja a figyelmünket az esetleges szintaktikai hibákra, és olyan hiányosságokra, amik később szemantikai hibákhoz vezethetnek, például a függőségi tömb hiányos kitöltése React effektek esetén. Az Eslint nem végez formázást, csakis ellenőriz.

### 3.2.1 Könyvtárszerkezet, komponensek

React v16.8-ig komponenseket, főleg csak javascript osztályok segítségével tudtunk definiálni. Lehetett függvényekkel is, de ezek nem rendelkezhettek belső állapottal, illetve életciklus metódusokkal. Ezeket nevezzük *pure components*-nek. A 16.8-as verziótól kezdve ez viszont megváltozott a [hook](https://reactjs.org/docs/hooks-intro.html)-ok bevezetésével. Ez sok problémát kiküszöbölt, amik az osztályként definiált komponensekből eredtek. Ilyen volt például a javascript-ben a többi nyelvtől jelentősen eltérő, és olykor kaotikusnak tűnhető *this* kulcsszó használata, sok csomagoló (wrapper) komponens egymásba ágyazása és időnként nehezen értelmezhető/olvasható forráskód. Ennek elkerülésére a projektben már az újféle mintát követő functional komponenseket használom hook-okkal.

A komponens stílusok egyénire szabásához [css-in-js](https://cssinjs.org/?v=v10.5.0)-t használok. Ennek előnyei közé tartozik gyorsasága és biztonsága. Minden szabályhoz különböző azonosítót generál, így elkerülhető a névütközés, ami CSS-ben sokszor okoz fejfájást.

Könyvtárszerkezet:

 A komponensek a view könyvtárban találhatóak. Az általánosabbak, amik több helyen is fel vannak használva, a view-en belül a *components* mappában vannak.

A bonyolultabb komponensek saját mappát kaptak, ebben általában a következők fájlok szerepelhetnek:

* Maga a komponens forrás fájlja, ennek elnevezése megegyezik a mappájáéval
* Egy styles.js fájl, amiben a stílus definíciók találhatóak
* Bonyolult állapotrendszer esetén egy state.js
* A teszt fájl
* Egy index.js ami csak újra kiexportálja a szomszédos fájlok tartalmát. Ez a [node modul rezolúciós algoritmusa](https://nodejs.org/api/modules.html) miatt érdekes, hogy a rövidebb formában (.../view/Component) lehessen hivatkozni rá, és ne kelljen a teljes elérési útvonalat megadnunk (.../view/Component/Component), ha használni szeretnénk a komponenst.

ábra frontend könyvtár struktúra

Az *api* könyvtárban két fontos fájl található:

* axios.js, ennek segítségével kommunikálhatunk a kódfordító szerver REST API-ával
* directus.js, ő pedig a CMS-el való kommunikációt egyszerűsíti meg

Api hívások az alkalmazásban két esetben történnek, ha betöltjük az alkalmazást, vagy ha megnyitunk egy leckét. Ez a felhasználói élmény javítása miatt van így, hogy ne kelljen a felhasználónak sok töltőképernyőt néznie. Ennek viszont hátulütője websocket hiányában, ha új lecke kerül publikálásra amíg nyitva van az alkalmazás, erről csak akkor veszünk tudomást, ha újrafrissítjük az oldalt.

A *theme.js*-ben találhatjuk az alap material dizájnt felülírő egyedi stílusokat. Itt vannak deklarálva a téma színei is. Egy esetleges téma színvilág változtatás keretében itt a colors nevű objektum tagváltozóinak értékét kell módosítanunk. Ezen a fájlon kívül nem található direktben megadott színkód. Ha itt megváltoztatjuk az értékeket az az egész applikációra kihatással lesz.