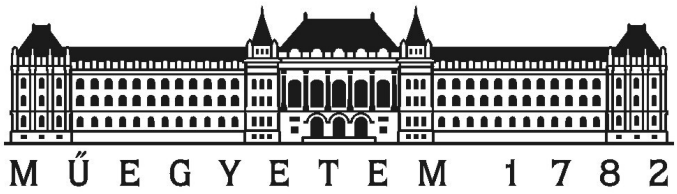


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

Hatékony és automatikus programozás oktatást támogató rendszer

TDK Dolgozat

2017

Salyámosy András, Sári László

Konzulensek: Csuka Barna, Dr. Kollár Zsolt

Tartalomjegyzék

Kivonat

Az egyetemi oktatásban komoly problémát jelent a hallgatók által készített feladatmegoldások értékelése. A hatalmas mennyiségű beadott feladat kézi javítása sok időt von el az oktatástól, így gyakorlatilag mindenhol megfigyelhető azon törekvés, hogy a javító-értékelő munkát minél nagyobb mértékben automatizálják. A BME Villamosmérnöki és Informatikai karán a német nyelvű mérnökképzésen oktatott Programozás alapjai 1 és 2 tárgyakból már évek óta hetente kiadott házi feladatok segítik a hallgatók tanulását, így itt is régen megjelent ez az igény.

A fejlesztés során ezen házi feladatok, amelyeket sok ideig az oktatók kézzel ellenőriztek, kezelésének és javításának automatizálása volt a cél, hogy minél több emberi munkát a gép végezzen el. Az elkészült rendszerben lehetséges a házi feladatok kezelése és a feladatokhoz kötődő jogosultságok finomhangolása is. A feltöltött feladatmegoldások közül a rendszer képes felismerni a másolt megoldásokat, és automatikusan tesztelni és eldönteni, hogy helyesek-e.

A hallgatók így gyorsabban kapnak visszajelzést a munkájuk helyességéről, míg oktatói részről csak a ténylegesen helyes munkák kézi jóváhagyására redukálható az elvégzendő munka. Az eddigi visszajelzések (mind hallgatói, mind oktatói) szerint is sokat segít az elkészült környezet hallgatóinknak a programozás gyakorlásában és a kötelező feladatok beadásában. A rendszer egy kényelmes platformot biztosít jelenleg a C és C++ programnyelveken való programozás gyakorlására, de a moduláris felépítésnek köszönhetően könnyen bővíthető további programozási nyelvekkel is.

Abstract

One large problem at universities is correcting the students tasks. The correcting of this large amount of submitted tasks by hand requires much time, which could be spended on teaching. Because of that, most of the teachers at universities want this process to be automated. At the Faculty of Electrical Engineering and Informatics of BME we teach Basics of Programming 1. and Basics of Programming 2. at the german education the students get small tasks every week to help them learn C and C++ programming. Thus, we wanted to automate the correction of these tasks.

The goal of this project was solving the automatization of task submitting and correcting, to reduce human work by using machines. In this system it is possible to handle these tasks and fine tune the permissions related to the tasks. The system can detect plagiarism amongst uploaded solutions and automatically test and decide whether they are correct or not.

This way the students get feedback about their solutions sooner, and the work of the teachers is reduced to only accepting the solutions with correct output. According to the students and based on the experience of the teachers this system helps in learning C and C++ programming effectively. The system provides a convenient platform in practicing programming, and the set of supported programming languages can be extended thanks to modular architecture.

1. fejezet

Bevezető

1.1. Előzmények és célok

A Budapest Műszaki és Gazdaságtudományi Egyetem (BME) Villamosmérnöki és Informatikai karán a német nyelvű mérnökképzésen több éve már, hogy az elsőéves hallgatóknak tartott Programozás alapjai 1. és Programozás alapjai 2. tárgyak keretén belül minden hallgatónak kötelezően kell beadnia hetente kis házi feladatokat. Az első féléves hallgatók heti hat programozási feladatot kapnak, amelyeket C nyelven kell megoldaniuk, míg a másodévesek heti egy, C++ nyelven megoldandó feladatot kapnak. A tapasztalataink szerint a hetente kiadott házi feladatok nagy mértékben segítik a hallgatókat a C és C++ programnyelveken történő programozás gyakorlásában. Ezen feladatok beadását igyekszik támogatni a HAWEB rendszer.
A HAWEB rendszer eddigi rövid története:

- 2008-ban Dady Róbert készítette el az első változatot (HAWEB1), ezen keresztül zip fájlokban lehetett feltölteni a megoldásokat. Ezeknél még semmilyen ellenőrzés nem volt, a megoldásokat a javító tanárok közvetlenül a HAWEB szerveréről töltötték le, a saját számítógépükön lefordították és ellenőrizték.

- 2009-től Braun Patrik fejlesztette a rendszer második verziót (HAWEB2). Ezen rendszer már képes volt fordítani és jelezni, hogy a megoldás hiba- illetve figyelmeztetésmentes is. Továbbá a hallgatóknak már direktben tudtak az oktatók visszajelzéseket adni. Ez a rendszer azonban biztonságtechnikailag még nem volt megerősítve.

- 2012-től Bessenyei Balázs vette át a rendszert (HAWEB3), ekkora a rendszer már biztonsági kritériumoknak is megfelelt, de automatizált teszteket még nem végzett. Előnye volt még, hogy a Neptun kód alapján már automatikusan tudta a hallgatókat regisztrálni és emailt küldeni számukra. A feladatokat már nem csak szöveges fájlként, hanem mint latex kód is létre lehetett hozni, amelyből pdf-et generált. Első kísérletként GitHub is be lett vezetve, azonban az első éves hallgatók számára ez bonyolultnak bizonyult, így a későbbi rendszerben el lett vetve.

- 2016-tól vettük át a rendszer fejlesztését (HAWEB4). Először C++-ra lett a rendszer kifejlesztve. A rendszerben a feladatok és a hallgatók már csoportosíthatóak és összerendelhetőek voltak, a feladatkiírás is külön felhasználói felületet kapott.

Ennek a rendszernek a továbbfejlesztése a HAWEB5, amely a feltöltött kódokat képes a fordításon túl automatikusan kijavítani, statisztikai adatokat gyűjteni a feladatmegoldásokról és a felhasználókról illetve felismerni a plágiumgyanús megoldásokat. Ezáltal a hallgatók nem csak a fordítás eredményéről kapnak szinte azonnali visszajelzést, hanem a megoldás tényleges helyességéről is. A kézi ellenőrzés így egy opcionális lehetőség, a rendszer képes akár felügyelet nélkül, önműködően is adminisztrálni.

A dolgozat következő alfejezetében összefoglaljuk a miénkhez hasonló megoldásokat: előbb a BME-n belüli majd a világ más egyetemein megtalálható hasonló rendszerekről írunk. A második

fejezetben részletesen kifejtjük a rendszerünk működését, modulokra bontva, majd a harmadik fejezetben esetleges további felhasználásokról és fejlesztési irányokról írunk. Végül dolgozatunkat egy összefoglalással zárjuk.

1.2. Hasonló megoldások összefoglalása

Ebben a részben a HAWEB5-höz hasonló rendszerekről írunk, előbb a Műszaki Egyetemen belül elérhető megoldásokat, majd a külföldi példákat elemezzük. Egy példától eltekintve minden rendszer célzottan programozási feladatok kezelésére készült, bemutatjuk ezen rendszerek fő jellemzőit, majd leírjuk a legfőbb hasonlóságokat és különbségeket a mi rendszerünkhöz képest.

Automatikus beadórendszerek a BME-n

InfoC

A mérnökinformatikus hallgatók magyar képzésen futó Programozás alapjai I. tantárgyának tárgyhonlapja, rengeteg segédanyaggal. A tanulást kötelező házi feladatok, illetve hetente kiadott szorgalmi feladatok segítik. A honlapra csak a forráskód feltöltése lehetséges, a feltöltött forráskódokat a rendszer lefordítja és eltárolja. Ha a feltöltött kód szintaktikai hibát tartalmaz, vagy a fordító bizonyos figyelmeztetései miatt a kódot a rendszer nem menti el.

A plágiumkeresést az Amszterdami Egyetemen írt sim nevű rendszer [1] végzi, a feladatmegoldások ellenőrzését egy szkript segíti, ezzel minimális emberi közreműködés mellett futtatható le a plágiumkeresés. Az oktatói tapasztalatok alapján a 80%-os vagy nagyobb egyezést mutató megoldások a ténylegesen plágiumgyanúsak, ezeket érdemes kézzel is ellenőrizni.

A plágiumkereső előbb a feltöltött forráskódokból egy ún. parse tree-t (elemzési fát) állít elő, amely a programkód minden függvényét, operátorát és operandusát tartalmazza. Az előállított fákat ezután egyszerű szöveggé alakítja, és ezen szövegek között keres hasonlóságokat. Így a rendszert nem lehet megkerülni változónevek vagy függvénynevek átnevezésével, illetve whitespace karakterek beszúrásával. A folyamat végén a rendszer minden programkódpárhoz egy 0 és 1 közti számot rendel hozzá, amely a hasonlóságuk mértékét fejezi ki.

CPorta

Ez a rendszer szintén a magyar képzésen készült, a villamosmérnök hallgatóknak tartott Programozás alapjai I. és II. tárgyak számára. Az oldal C és C++ nyelvű forráskódok feltöltését és fordítását támogatja, a feltöltött kódokat nem csak lefordítja, le is futtatja és tesztesetekkel ellenőrzi azok helyes működését. Ha a feltöltött kód nem fordul le, vagy valamely teszteset nem futott helyesen, esetleg memóriaszivárgás jelentkezett a futás során, a feladat elutasításra kerül. Egyéb fájlok is feltölthetőek a rendszerbe, támogatja a nem lefordítandó fájlok feltöltését is. A feltöltés során ki kell választani, hogy mely .c, illetve .cpp kiterjesztésű fájlokat forduljanak le a szerveren, a fájlok saját gépen való kiválasztása után a rendszer automatikusan kiválasztja ezeket, de a felhasználó ezt módosíthatja.

Adatbázisok labor

Az Adatbázisok laborhoz készült házifeladat kezelő rendszert képes a hallgatók által beadott feladatokat tárolni, és azokat lefordítani, részben automatikusan kiértékelni. A fordítás és a kiértékelés feladatonként eltérő módon történik, mivel eddig az évig a tárgy során 5 teljesen eltérő labort végeztek el a hallgatók. A rendszer képes az egyszerű feladatokat tárolni, Javaban írt megoldásokat SCons segítségével fordítani, illetve a beadott forráskódokban és jegyzőkönyvekben található egyezéseket detektálni. A plágiumkeresést a félév végén végzik el, az előző évfolyamok feladatsoraival is összehasonlítják a tárgyat aktuálisan végzők megoldásait.

A feladatok a szerveren csak lefordításra és eltárolásra kerülnek, utána a javítás kézzel történik, a javítótanárok a javításhoz megoldókulcsot kapnak, hogy egységes módon javítsák a feladatokat.

Grafika házi beadó rendszer

A tárgyból egy félév alatt 3 házi feladatot írnak ki, amelyeket a tárgyhoz tartozó honlapon lefordítanak és lefuttatnak. A rendszert Domokos Balázs volt doktorandusz fejlesztette, majd Tóth Balázs fejlesztette tovább.

A házi beadó rendszer a programok be- és kimenetét figyeli, képernyőképet készít a futó programról egy megadott időpillanatban, így a programok ellenőrzése során nem elég a standard ki/bemenet átirányítása, szükséges a felhasználói inputok szimulálása. A tárgyból a házi feladatokhoz egy skeletont adnak ki. A kiadott skeleton ennek bizonyos részein tilos változtatni, mivel a feltöltött fájlokat feldolgozza a házi beadó rendszer, és két megjelölt sor közti kódrészletet kivág belőle, majd beilleszti egy másik forráskódba - így biztosított, hogy a hallgatók nem használnak olyan könyvtárakat, amelyek használata nem megengedett. Ha például egy hallgató fájlkezelő függvényeket akar használni, a programja le se fordul, így kártékony kódok lefuttatása nem lehetséges a rendszerben. Évente mintegy 10.000 feltöltött feladatmegoldást kezel a rendszer az egy hallgató általi többszöri feltöltéseket is számolva.

A feladatok javítása itt is kézzel történik, egyrészt ellenőrzik a házi beadó rendszer által készített képernyőképeket, másrészt a forráskódot. Azok a megoldások is kaphatnak pontot, amelyek adott esetben nem rajzolnak ki semmit a képernyőre, de a feladat matematikai része helyes. A házi beadó rendszer egy plágiumkereső modult is használ, amelyet a tanszék saját szerverén futtatnak.

Automatikus beadórendszerek más egyetemeken

WebAssign

A WebAssign [2] fejlesztése 1997-ben kezdődött a North Carolina State University-n, a projekt célja kezdettől fogva egy általános keretrendszer létrehozása volt, amely a tudományos, technikai, mérnöki és matematikai oktatás számára biztosít egy bővíthető platformot, házi feladatok kezelésére.

Minden feladathoz tartozik egy beadási határidő, a megoldások feltöltése után a WebAssign rendszer képes meghívni automatikus teszteket azonnal a feltöltés befejezése után, illetve a beadási határidő lejárta után szintén lehetséges automatikus teszteket használva kiértékelni a feladatokat a kézi ellenőrzése mellett. Az automatikus tesztelés egyszerű feladatok esetében (feleletválasztós kérdések, egyszerű szöveges válasz) a WebAssign szerverén lefutnak, de például programozási feladatok esetében a rendszert használó oktatónak kell gondoskodnia egy saját szerverről, amely együtt tud működni a WebAssign rendszerével. Ha egy feladathoz tartozik ilyen automata javító rendszer, a WebAssign jelzi a külső szervernek, hogy érkezett egy új megoldás, a szerver pedig egy API-n keresztül jelezheti az eredményt.

A Hageni Távegyetemen a bevezető számításelmélet kurzuson 1999-ben vezették be a WebAssign használatát, a korábbi papír alapú rendszer helyett. Kezdetben az online feladatmegoldás opcionális volt, az online feladatok pontosan megegyeztek a papír alapúakkal. Már az első évben a hallgatók harmada átállt az online rendszer használatára, hetente 4000 hallgatói megoldást értékelt ki automatikusan a rendszer.

Programozás oktatás Hageni Távegyetemen

A Hageni Távegyetemen különösen fontos megoldani a feladatok javításának minél nagyobb mértékű automatizálását, hiszen ezzel biztosítható, hogy a feladatokat nem papíron kell beadni, amely egy távegyetem esetében a hallgatóknak jelentős könnyebbséget jelent.

A programozás oktatásánál [3] Prolog és Scheme nyelveken írt programokat ellenőriztek: A 2.2.1 pontban leírt WebAssign rendszer tudását bővítette ki az elkészült automata javító környezet. A javításhoz a rendszernek szüksége van oktatói oldalról tesztesetekre, illetve egy mintamegoldásra, amelyet futtatva ellenőrizhető a megoldás helyessége. A WebAssign az automata javító környezet meghívásakor átadja a felhasználó megoldásának azonosítóját, a feladat alapadatait és magát a megoldást. Ha egy program túl hosszú ideje fut, esetleg hiba lépett fel a működése közben, a feladatot megoldó hallgató értesítést kap róla, és újra feltöltheti a megoldását, kijavítva a hibákat.

C programozás a Római Egyetemen

A Római Egyetemen [4] a hallgatók az ottani Programming 2 és Programming Laboratory tárgyak keretében a C programnyelvvel ismerkedhetnek meg. A diákoknak minden második héten kell házi feladatokat megoldaniuk, ezeknek a feladatoknak a javítását automatizáltan végzik el a tárgy oktatói. A javítást egy igen komplex rendszerben oldják meg, amellyel nem csak a teljesen elkészült megoldások értékelhetőek, hanem akár a megoldás részei, akár egy nem teljesen befejezett feladatmegoldást is képes kijavítani és értékelni a rendszer.

A rendszer unit testeket használ, a hallgatóknak kiadott feladatok pontosan specifikálva tartalmazzák az elkészítendő függvények nevét és fejlécét - az oktatók minden függvényhez külön teszteseteket készítenek, hogy a program működését vizsgálják. A feladatokhoz tartozik egy mintamegoldás, amelyben minden kötelezően megírandó függvény helyesen szerepel. Ha a hallgató megoldásában valamely függvény hibásan működik, azt lecserélik az oktató által készített helyes függvényre, így a megírt részletek helyességétől függetlenül az összes teszteset le fog futni, és ha esetleg egy hallgatónak van helyes és hibás programrészlete is, akkor visszajelzést kap, hogy mely függvények voltak helyesek, és melyikek voltak hibásak.

A rendszer nagy hátránya a tesztesetek elkészítésének nehézsége mellett a tesztesetek megkerülhetősége: mivel ezek a tesztek a C/C++ programnyelvek preprocesszorát használják, amely a program fordítása előtt lefut, a hallgatók, ha ismerik a preprocesszor működését, egyetlen sor hozzáadásával a kódjukhoz minden tesztesetet megkerülhetnek. Ez természetesen kiszűrhető külön ellenőrzések beiktatásával, de igen nehéz kizárni, hogy egy hallgató nem képes megkerülni az ellenőrzést. Ebből kifolyólag egy ilyen rendszer biztonságosan, kézi ellenőrzés nélkül csak a hallgatók számára használható egy gyakorlófelületként, a teljesítményük értékelésére már nehezebb ezt a rendszert használni.

Egy másik nagy probléma a rendszerrel kapcsolatban a függvényenkénti ellenőrzésből adódik: a fent leírt tesztesetekkel tesztelhető egy-egy függvény helyes működése, de a hallgatók használhatnak globális, esetleg statikus változókat és sok egyéb megoldást, amely egy függvénynél hosszabb életciklussal és nagyobb láthatósággal rendelkeznek - ebből következően elképzelhető, hogy egy teszteset helyesen lefut, ám egy másik függvény működéséhez szükséges feltételeket közben elrontja. A rendszer készítői azt javasolták, hogy egyszerre több tesztet futtassanak, azonban ez még mindig nem zárja ki a hiba lehetőségét valahol. Ha feltesszük, hogy egy tesztesetet elég egyszer lefuttatni az ilyen függvényeken kívüli hibák megtalálásához, akkor 2^n különböző teszteset-kombinációról beszélhetünk n teszteset esetén, és tesztesetek esetleges sorrendjét még nem vettük figyelembe. Így valamilyen statikus kódanalízist használó eszköz használata elengedhetetlennek tűnik egy programozási feladatokat kezelő rendszer esetében.

1.1. táblázat. A rendszerek összehasonlítása

Egyetem	Rendszer	Támogatott nyelvek	Automatikus javítás	Plágiumkereső	Kézi javítás
BME	InfoC	C C++	Fordítás	Van, külső fejlesztés	Szükséges
BME	CPorta	C C++	Fordítás Futtatás Ellenőrzés	NA	Nem szükséges
BME	Adatbázisok	SQL Java Python	Fordítás	Van, saját fejlesztés	Szükséges (javítókulcs alapján)
BME	Grafika	C++	Fordítás Futtatás	Van, saját fejlesztés	Szükséges (részpontszámok érdekében)
Hageni Távegyetem		Prolog Scheme	Fordítás Futtatás Ellenőrzés	NA	Nem szükséges
Római Egyetem		C++	Fordítás Futtatás Ellenőrzés	NA	Nem szükséges
BME	HAWEB5	C C++	Fordítás Futtatás Ellenőrzés	Van, saját fejlesztés	Opcionális

2. fejezet

HAWEB5

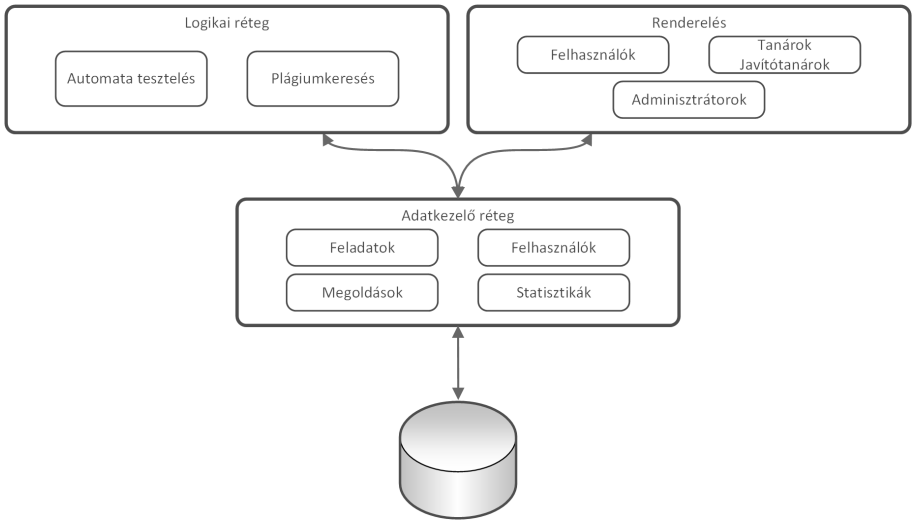
2.1. Célok

A HAWEB5 fejlesztése során a következő célokat tűztük ki:

- az oktatók és a javítótanárok által elvégzett munka minél nagyobb mértékű automatizálása, hogy az egyértelműen hibás esetekkel (nem forduló vagy hibás kimenetet adó program) ne keljen foglalkozni;
- az automata tesztelés megkerülésének vagy kijátszásának minél nehezebbé tétele;
- a hallgatók minél hamarabb kapjanak visszajelzést az elvégzett munkájukról;
- átfogó statisztikai kimutatások készítése;
- a házi feladatok másolásának megelőzése illetve felismerése.

2.2. Áttekintés

A rendszer az oldal mögött álló adatbázisból és a rá épülő három rétegből áll, melyek felépítése az alábbi 2.1 ábrán látható.



2.1. ábra. A rendszert felépítő rétegek és modulok áttekintése

- Logikai réteg:

  Ez a réteg felelős a rendszer automatizált részének működésért, a többi rétegről teljesen leválasztva működik, azokkal TCP kapcsolaton kommunikál, a benne lévő modulok Python-ban készültek. A rendszer többi rétegével TCP kapcsolaton keresztül kommunikál, ennek

köszönhetően akár más szerveren is futtathatóak az itt található modulok. Ezzel a rendszer skálázhatósága jelentősen javítható.

  - Automata javítás:

    A feladatmegoldások automatikus fordítását és kiértékelését végző modul (bővebben lásd a 2.4 fejezetben).

  - Plágiumkereső:

    A hallgatók által feltöltött megoldásokat egy saját fejlesztésű plágiumkereső ellenőrzi, részletes leírása a 2.5 fejezetben olvasható.

- Megjelenítő réteg (renderelés):

  A felhasználók számára rendereli az oldalt, illetve kezeli a kliens és szerveroldal közti kommunikációt. A réteg a különböző jogosultságokkal rendelkező felhasználóknak különböző felépítésű és funkcionalitású oldalt mutat. Az alábbi felsorolásban látható, hogy milyen felhasználói szerepek lettek definiálva a fejlesztés során (bővebben lásd a 2.3. fejezetben) :

  - felhasználók;
  - javítótanárok;
  - oktatók;
  - adminisztrátorok;

- Adatkezelő réteg:

  - Felhasználók:

    A felhasználói profilhoz kötődő műveleteit (pl. regisztráció, bejelentkezés, csoportba való jelentkezés stb.) kezelő modul.

  - Feladatok:

    A különböző feladatok kategóriákba sorolását, a felhasználói csoportokhoz rendelt jogosultságokat kezeli és az ezekkel kapcsolatos lekérdezéseket (feladat módosítása admin oldalról, feladatmegoldás feltöltése, stb.) hitelesíti.

  - Megoldások:

    A feladatokhoz feltöltött megoldások adatait kezelő modul, a megoldásokhoz tartozó hozzászólásokat is kezeli. Ha egy megoldás státusza megváltozik (automata javítás esetén: Helyes / Hibás / Javításra vár; kézi javítás esetén: Elfogadva / Elutasítva / Nincs elbírálva), email üzenetben értesíti a feltöltőt. Ha egy megoldáshoz új hozzászólás érkezik, akkor értesíti a korábban hozzászóló javítótanárokat, illetve a megoldás feltöltőjét.

  - Statisztikák:

    Statisztikai adatokat tárol a feltöltött megoldásokról, illetve az eltárolt adatokból különböző statisztikai kimutatásokat készít.

2.3. Felhasználói szerepek

A rendszer felépítése során 4 felhasználó szerepet különítettünk el, amelyek egymásra épülnek, így amit egy alacsonyabb jogosultságokkal rendelkező felhasználó meg tud tenni, azt a nála magasabb jogokkal rendelkezők is.

Hallgató

Bejelentkezés (A.1. ábra) után hozzáférnek a saját profiljukhoz és a nyilvános csoportok listájához (A.2. ábra). Jelentkezhetnek ezekbe a csoportokba, a csoporttagságaikhoz rendelt feladatokat megtekinthetik (A.3. ábra, A.4. ábra), és megoldást tölthetnek fel a feladatokhoz határidőn belül. Megtekinthetik saját megoldásaikat (A.5. ábra), és hozzászólást írhatnak azokhoz.

Az automata javító rendszer lefutásáról email üzenetben értesítést kapnak, ugyanígy a feltöltött megoldásaik státuszának változásáról, illetve a javítótanárok által a megoldásikhoz küldött hozzászólásokról.

Javítótanár

Rendelkeznek minden jogosultsággal, amivel a hallgatók, a javítótanárok látják a hallgatók által feltöltött megoldások listáját (A.6. ábra), megnyithatják az egyes megoldásokat (A.7) amelyeket elfogadhatnak, vagy elutasíthatnak, illetve a feltöltött megoldásokhoz hozzászólhatnak, így tudatva a hallgatókkal, hogy miért utasítottak vissza egy feladatot.

Oktató

A javítótanári jogokon felül az oktatók szerkeszthetik a feladatokat (A.8), létre tudnak hozni új feladatokat, kitörölni a létezők közül. A feladatok szerkesztése mellett létrehozhatnak a feladatokhoz kötődő teszteseteket, amelyek az automatikus javítás működéséhez szükségesek. Láthatják a kategóriánkénti statisztikákat (A.9).

Admin

Minden fent leírt jogosultsággal rendelkeznek, megtekinthetik a felhasználók, illetve felhasználói csoportok listáját, jogokat rendelhetnek hozzá csoportokhoz. A csoportokba való jelentkezéseket jóváhagyhatják vagy elutasíthatják.

2.4. Automata javítás

A modullal szembeni követelmények

Az automata tesztrendszer elkészítése során a fő céljaink a következők voltak:

- Oktatói oldalról egyszerű kezelhetőség:

  Az automata tesztrendszer mögötti technikai részleteket nem ismerő oktatók könnyen tudják az új feladatokat létrehozni, kezelni és az azokra beérkező megoldásokat igény esetén kézzel is ellenőrizni.

- A hallgatók számára gyors visszajelzés:

  Az automata javítás előnye, hogy nem kell az oktatók beavatkozására és visszajelzésére várni; a hallgatók a feltöltés után nem sokkal látják, hogy a megoldásuk megfelelt-e a követelményeknek.

- Könnyű bővíthetőség:

  A rendszer használhatósága ne korlátozódjon csak a C/C++ nyelvekre, könnyen bővíthető legyen további nyelvek támogatására.

- Biztonság:

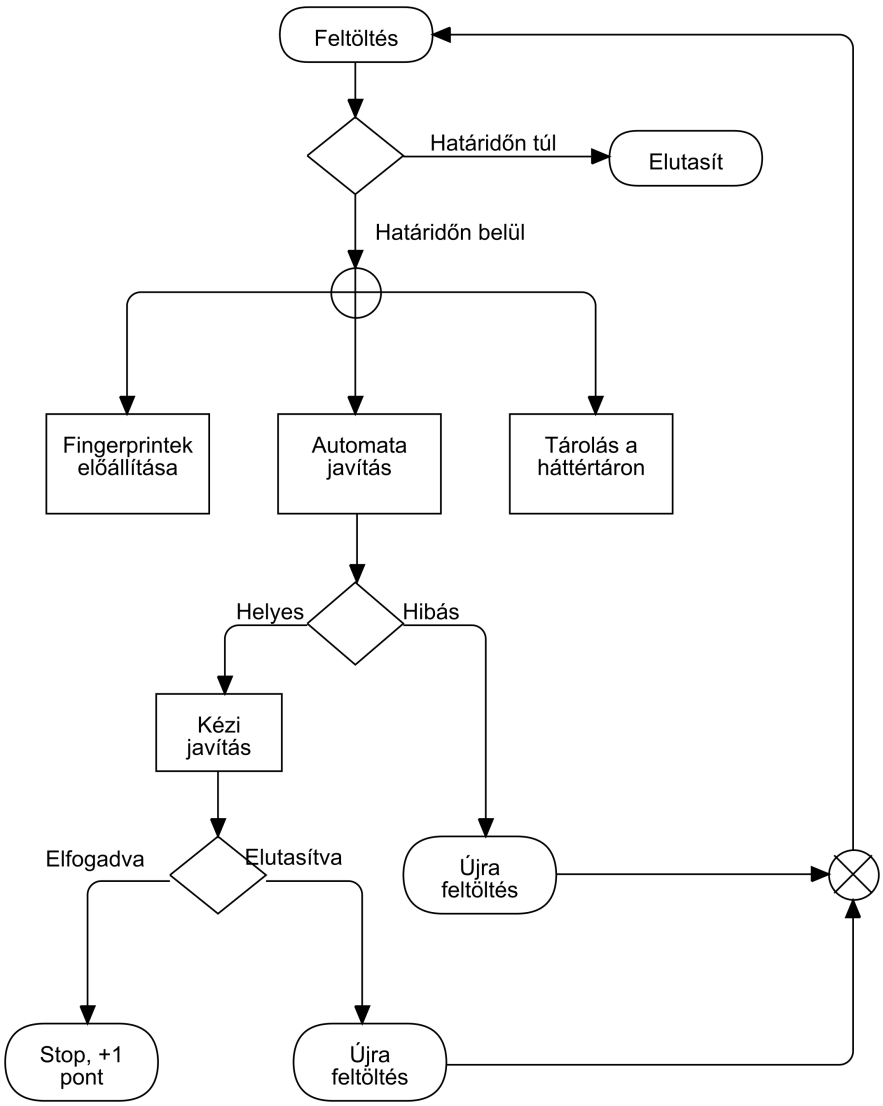
  A helyesség ellenőrzése érdekében a feltöltött megoldások fordításra, majd futtatásra kerülnek. Mivel azonban a beérkező forrásokról nem tudni semmit, így fel kell készülni kártékony kódok érkezésére is, ezért a programokat valamilyen elkülönített környezetben kell tesztelni.

- Stabilitás:

  Szorosan kapcsolódik az előző ponthoz: a rendszer teljesítménye a gyakori programozói hibák miatt nem romolhat látványosan, nem kerülhet inkonzisztens állapotba. Ilyen problémákat a teljesség igénye nélkül a következő hibák okozhatnak: fordítási hiba, végtelen ciklus, futási idejű hiba, magas memóriahasználat, stb.

Feltöltés és kiértékelés

A megoldások feltöltésének és a szerveren való kiértékelésének folyamata lent, a 2.2 ábrán látható.



2.2. ábra. A megoldások feltöltésének folyamata

Miután egy hallgató feltöltötte a megoldását a szerverre, az automatikusan lefordításra kerül ha határidőn belül töltötte fel. Ha a fordítás nem volt sikeres, akkor a kód vagy hibás, vagy olyan elemet tartalmaz, amelynek használata a feladat megoldása során nem megengedett. Ezzel párhuzamosan elindul az automata plágiumkereső és előállítja a megoldást jellemző fingerprinteket.

Ha a fordítás sikeres volt, akkor a lefordított programot elindítjuk egy teljesen izolált tesztkörnyezetben. A program bemenetére tesztadatokat ír a rendszer, és figyeli a kimenetét. Az oktatók által elérhető felületen különböző tesztesetek hozhatók létre, amelyek mindegyike kiértékelésre kerül. Minden teszteset esetében különböző bemenetek és különböző kimenetek állíthatók be. Ha ezek közül bármelyik nem volt helyes, a feltöltött megoldást a rendszer hibásnak jelöli.

A tesztesetek bemenete lehet akár konstans, ekkor minden futtatásnál, minden hallgató megoldása ugyanazt a bemenetet kapja, esetleg kiegészítve a hallgatókhoz kötődő egyedi adatokkal, például a felhasználónévvel. Emellett bonyolultabb feladatok esetén lehetséges teljesen egyedi bemenetet generálni, ekkor a helyes kimenetet egy az oktató által elkészített mintamegoldás állítja elő.

Ha a feltöltött megoldás megfelelt az automata teszteseteknek, akkor a javítótanárok még kézzel ellenőrzik, hogy a megoldás tényleg helyes-e, nem kerülte-e meg valamilyen módon a felhasználó az automatikus tesztet. Ha valamelyik fázisban a megoldás elutasításra került, a felhasználó újra feltöltheti azt.

Sandboxing módszerek

A szerverre bármilyen forráskódot fel lehet tölteni, így azokat túl veszélyes lenne csak úgy, ellenőrizetlenül futtatni. Szükség van egy, a rendszertől valamilyen szinten elkülönített környezetre (Sandboxra), ahol azokat többé-kevésbé biztonságosan lehet kezelni. A következőkben két szóba jöhető módszert mutatunk be.

Virtualizáció (WMWare)

A virtualizáció a Hypervizoron keresztül valósul meg. A gépet, amin a hypervizor fut gazda gépnek (host machine), míg az egyes virtuális gépeket vendég gépeknek (guest machine) nevezik. A hypervizor a virtuális gépek szervezéséért felelős. Minden vendég operációs rendszernek (guest OS) egy-egy külön virtuális gépet biztosít, melyek mind rendelkeznek a maguk virtualizált hardware-eikkel. Mindez nagyfokú izolációt eredményez és egyik vendég sem tudja – a hálózaton keresztüli kommunikációt nem számítva – hátrányosan befolyásolni a másikat, a hostot beleértve.

A módszer erőssége a nagyfokú biztonság, azonban "nehézsúlyúsága" (heavyweight) miatt több hátránya is van.

- Nehezen skálázható

  Egy VM-imagenek szüksége van egy komplett operációs rendszerre, annak minden elemével együtt (kernel, system lib, ...). Ennek mérete 1 GB körül fog alakulni. Ha egyszerre n db gépet kell futtatni, akkor minimum $n * 1GB$ helyre lesz szükség.

  A gépek elindítása meglehetősen lassú, ráadásul minden egyes futás után vissza kell őket állítani eredeti állapotukba, hogy a biztonsági kockázatokat minimalizálni lehessen. Ha hirtelen sok gépre lenne szükség, az nagy mértékben lassítaná a rendszer válaszidejét.

- Sebesség

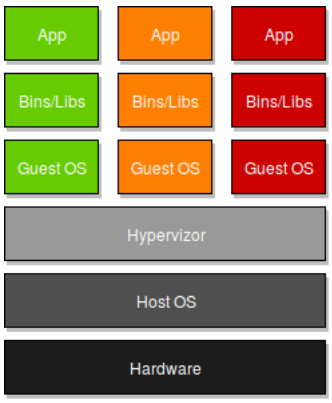
  Mivel a virtuális gép nem tud közvetlenül rendszerhívásokat kiszolgálni, azokat a hypervizoron keresztül végzi, bizonyos esetekben (pl. gyakori I/O műveletek) ez nagy overheadet generálhat, amely lassítja a virtuális gépet. A modern hypervizorok megpróbálják ezeket minimalizálni, de még így is megfigyelhető a teljesítménycsökkenés.

Konténerizáció (Docker)

A Docker virtuális gépek helyett úgynevezett konténereket (container) kezel. A konténer egy lazán izolált környezet, mely a gazda gép kernelén fut. Mivel nincs szükség hypervizorra, a konténerek "pehelykönnyűek" (lightweight), így egy adott hardware konfiguráció mellett több konténer futtatható egyszerre mint virtuális gép.

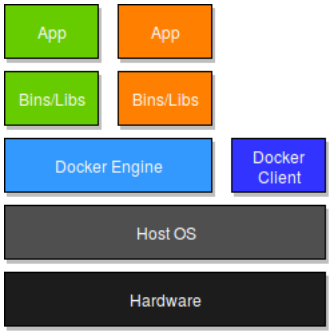
Egy-egy konténer elindítása maximum másodpercek kérdése, ráadásul a virtuális gépekkel szemben n konténer futtatásához csak kicsivel több memóriára van szükség, mint az alap image mérete.

A Docker Engine API-jához számos nyelven készültek SDK-k, ezzel kényelmes kezelhetőséget és programozhatóságot biztosítva a Docker konténerekhez.



2.3. ábra. Virtualizáció

Fontos megjegyezni, hogy mindezen előnyei mellett, pontosan azért, mert a gazda kernelen futnak a konténerek, nagy energiát és odafigyelést kell az egyes imagek konfigurációjára fordítani. Főleg, ha nem biztos forrásból származó kódokat akarnak futtatni bennük.



2.4. ábra. Docker

Választott megoldás

A kevésbé izolált környezet ellenére azért döntöttünk a Docker által nyújtott konténerizáció mellett, mert könnyen skálázható és kényelmesen programozható.

A tesztelés feladatát (azaz fordítás, futtatás, tesztelés) egy TestServer daemon látja el. Ehhez a daemonhoz egy megadott TCP-porton keresztül, json formátumú kérésekkel lehet fordulni. A szerver fogadja a kéréseket és egy FIFO-ba helyezi azokat. Ha a FIFO nem üres, és az aktív tesztelő szálak száma nem érte el a felhasználó által definiált maximumot, akkor automatikusan elindul egy újabb tesztelő thread, mely kiveszi a sorból az éppen következő elemet és feldolgozza a kérést. A kérésnek tartalmaznia kell a feladat típusát (azaz nyelvét, pl. cpp), a tömörített megoldás elérési útvonalát, a megoldás azonosítóját, illetve opcionálisan a feladathoz tartozó teszteseteket. Ha nincsenek tesztesetek, akkor csak a megoldás fordíthatósága lesz ellenőrizve, a program kimenete nem.

A megkapott megoldást kicsomagolja a szál egy adott mappába, majd minden tesztesetre elindít egy, a megadott feladattípushoz tartozó konténert, read only módban hozzácsatolva a megoldás könyvtárát. A konténer standard inputjára az adott tesztesetben definiált input kerül. A konténerben fordításra, majd futtatásra kerül a program. Ha túl sokáig fordulna vagy futna a program, az alkalmazás leállítja azt. Miután végzett a konténer a futtatással a tesztelő szál összeveti az aktuális elvárt outputot a konténerből származóval. Ha ezek bárhol eltérnek egymástól, akkor a feladat hibásnak minősül. Ellenkező esetben, ha minden teszteset helyesen lefutott, a megoldást elfogadja a rendszer, s rögzíti az egyes eseteknél mért futási időt és memóriahasználatot.

A modul az eredményt egy POST request body-jában küldi el JSON formátumban, mely tartalmazza a megoldás azonosítóját. Ha a megoldás helyes, akkor elküldi a futási időt és a memóriahasználatot is minden egyes tesztesetre külön.

Biztonság

A beadott forráskódok egy jogosulatlan (unpriviledged) Alpine Linux alapú konténerben fordulnak le, majd kerülnek futtatásra. Mielőtt a konténerben bármi is történne, lefut egy inicializáló script, mely törli a legtöbb shell programot, hogy minél kevesebb támadási felület és eszköz legyen a konténeren belül. A konténerben a hálózati kapcsolatok teljesen le vannak kapcsolva, így azt ilyen formában elérni, illetve róla hálózaton keresztüli kommunikációt kezdeményezni nem lehet. A konténer kimenete semmilyen formában nem érheti el a felhasználókat. A kimenetek egy switch-case blokkban kerülnek feldolgozásra, mely egy string összehasonlításból áll, és a folyamat végén egy előre meghatározott halmazból kerül ki egy üzenet, amelyet a modul elküld az adatkezelő rétegnek. Abból kifolyólag, hogy nem lehet látni a feltöltött kódok tényleges kimenetét, nagyban megnehezül egy esetleges, a rendszer ellen intézett támadás, hiszen a támadók nem kaphatnak információt a rendszer felépítéséről, a támadási próbálkozás sikerességéről.

2.5. Plágiumkeresés

Az oktatásban újra és újra fölmerülő probléma, hogy a hallgatók megpróbálnak csalni, egymásról másolni. A tanároknak a beadott feladatok javítása közben nehéz még az esetleges plágiumokra is figyelni, akkor is, ha kevés megoldás érkezett. Mindemellett az sem garantálható, hogy egy-egy feladatcsoportot egy oktató fog javítani, így ha az esetlegesen másolt feladatok más-más javítóhoz kerülnek, azok átcsúsznak az ellenőrzésen. Egy automata feladatjavító rendszer esetén, ahol a hallgatók egy géptől kapnak visszajelzést, nem pedig egy tekintéllyel rendelkező tanártól, pszichológiai okokból is több csalási kísérletre lehet számítani.

A fenti indokok miatt lett része a rendszernek egy plágiumkereső modul is. Egy hallgatót csalással vádolni nagyon súlyos és bizonyítást igénylő vád. Épp ezért fontos megjegyezni, hogy nem szabad teljes mértékben a keresőre támaszkodni. A modul által jelzett számokat a gyanú mértékeként kell felfogni, és minden esetben további, emberi vizsgálatok szükségesek annak eldöntésére, tényleg csalás történt-e.

Mi String alapú plágiumkeresési módszert választottunk a feladatra, mert ezen módszerek könnyen lehetővé teszik a gyanús részletek kiemelését a forráskódokban. A továbbiakban részletezésre kerülnek ezek a módszerek, majd ezt követően az általunk választott megoldás is.

Plágiumkeresési módszerek

Bag of Words

A dokumentumokból vektorokat képeznek, ami több módon is történhet. Egyik megvalósítás, hogy a dokumentumban felmerülő szavak gyakoriságai alkotják a dokumentumhoz tartozó vektort. Másik módszer, hogy a dokumentum szavaiból egy neurális háló segítségével vektorokat képeznek, majd ezeket összeadva jutnak a dokumentumok vektorához. Két dokumentum közti hasonlóságot az őket reprezentáló két vektor által bezárt szög fogja jellemezni.

LSA (Latent Semantic Analysis)

A Bag of Words egy változata. A dokumentumokból készítenek egy $\underline{\underline{A}}$ mátrixot, ahol $a_{i,j}$ az i. dokumentumban a j indexű szó gyakoriságát jelenti. A elemeit valamely módszer(ek) szerint súlyozzák. A zaj csökkentése érdekében szinguláris érték felbontással (SVD) előállítják az $\underline{\underline{U}}, \underline{\underline{\Sigma}}, \underline{\underline{V}}$ mátrixokat, ahol $\underline{\underline{U\Sigma V}} = \underline{\underline{A}}$. $\underline{\underline{\Sigma}}$ dimenzióját csökkentik, majd az $\underline{\underline{A'}} = \underline{\underline{U\Sigma' V}}$ mátrix-val dolgoznak tovább. A hasonlóságokat ugyanúgy két-két dokumentum-vektor által bezárt szöggel mérik. Az

$\underline{\underline{A'A'^T}}$ művelettel előállítható a $szó - szó$ mátrix, ahol az i. sor j. eleme azt mutatja, mekkora a hasonlóság az i és j indexű szavak között. E mátrix segítségével kiemelhetők a másolt részletek.

Fingerprinting

A dokumentumokból n–gram-kat (fingerprinteket) választanak ki, majd ezen n-gram-k közti egyezéseket keresnek. Ennek egyik előnye, hogy könnyen el lehet tárolni az egyes fingerprintek helyzetét a dokumentumokban, így megjeleníthetőek a dokumentumok közti egyezések. Másik előnye, hogy hatékony, nagy adathalmazokra is könnyen alkalmazható.

Suffix fák [5]

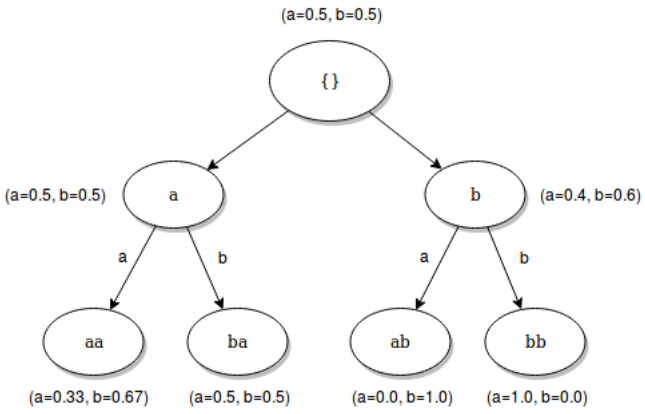
Minden dokumentumhoz készítenek egy valószínűségi suffix fát (PST). A PST-ben minden node-ban két tulajdonságot tárolnak:

- a stringet, amit az adott node reprezentál;
- a relatív frekvenciavektort a következő lehetséges karakterekre, amik az aktuális string után következhetnek a szövegben. A klasszikus PST definícióval szemben itt nemterminálló node-oknál is szokás felvenni ezeket a tulajdonságokat.

Egy ilyen maximum $d-1$ mélységű fának a következő tulajdonságai vannak:

- A fa gyökere az üres stringet reprezentálja.
- Minden nem gyökér node a szülője stringjét reprezentálja egy, az elejéhez hozzáadott karakterrel kibővítve.
- Egyik node sem reprezentál $d-1$ hosszú stringet.
- A teljesen felépített fa egy változó hosszúságú Markov-láncot reprezentál maximum $d-1$ memóriával.

Ha elkészültek a modellek, megnézik, hogy az adott dokumentum suffix fájában mekkora valószínűséggel lehet eljutni a másik vizsgált dokumentum stringjeihez.



2.5. ábra. Az `aabbabbaaabb` szövegből generált PST

Stíluselemzés

A forráskód szerzőjének stílusáról tárolnak statisztikákat. Például átlagos kommentelési frekvencia, blokkok stílusa (hagy-e ki sort, rak-e space-t a '{' karakter elé), stb. Kezdő programozóknál nem alkalmazható, hisz még biztosan váltogatnak a különböző konvenciók között. Általános esetben azonban jó indikátor a nagy mértékű eltérés a nyilvántartott és a vizsgált forráskód stílusai között.

Választott módszer

Mi a fingerprinting módszert választottuk, mert e módszer kereső adatbázisa könnyen bővíthető, illetve a másolt részek kiemelése könnyen implementálható és nagyban segít eldönteni, valóban csalás történt-e.

Működés [6]

- Preprocess

  A forrásfájlok először az alábbi többlépcsős előfeldolgozási folyamaton mennek keresztül.

  - kommentek eltávolítása;
  - idézőjelben lévő stringek átírása fix S-re;
  - változók átnevezése fix V-re;
  - függvények átnevezése fix F-re.

  Annak érdekében, hogy az egyezések kiemelését az eredeti forrásfájlon lehessen elvégezni, ne csak a feldolgozott szövegen, a feldolgozás és ujjlenyomatvétel alatt a rendszer nyilvántart egy karakter map-ot, melyben a feldolgozott szöveg i. karakter eredeti pozícióját tárolja el.

- Fingerprinting

  - A program sorban előállítja a szöveg $k - gramjait$ (szövegben k egymást követő karakter), ahol k a felhasználó által előre rögzített.
  - A kapott $k - gramokat$ a rendszer elhasheli. (Megvalósításnál e két első lépést egybevontuk, és rolling hashing segítségével egyből a hasheket kapjuk meg.)
  - Fingerprintek kiválasztása *winnowing* algoritmussal. A hasheket w hoszzú ablakokban vizsgálja a program (egy ablakban w darab egymást követő hash van). Az ablakban a legkisebb értékű hasht, annak pozíciójával együtt tárolja el. Ha több ilyen lenne egy ablakban, akkor a jobb szélsőt választja ki. Ezek után eggyel továbbcsúsztatva az ablakot folytatja ugyanezt, míg az ablakkal el nem éri a hashek végét. A kiválasztott fingerprinteket a dokumentum ID-jával feltölti az adatbázisba

```
bbaaaabcabcdefg
```
(a) A szöveg

```
bba baa aaa aaa aab abc bca cab abc bcd cde def efg
```
(b) A szövegből generált 3-gram-k sorozata

```
66 65 50 50 64 23 26 76 23 29 40 33 42
```
(c) A 3-gram-k lehetséges hasheinek sorozata

```
(66 65 50 50) (65 50 50 64) (50 50 64 23) (50 64 23 26)
(64 23 26 76) (23 26 76 23) (26 76 23 29) (76 23 29 40)
(23 29 40 33) (29 40 33 42)
```
(d) Az ablakok a *winnowing* algoritmus alatt

```
50 23 23 29
```
(e) A kiválasztott fingerprintek

```
(50, 3) (23, 5) (23, 8) (29, 9)
```
(f) Fingerprintek kiegészítve a pozíciójukkal

2.6. ábra. Példa a fingerprintek kiválasztására $k = 3$, $w = 4$ esetén

- Lekérdezés

  A program egy feladatazonosítót vár a felhasználótól. Ezen azonosító alapján kikeresi az erre a feladatra érkezett megoldások ID-ját. Ezen ID-k mentén lekérdezi az adott megoldások fingerprintjeit és összeveti ezeket egymással. A hasonlóság mértéke két dokumentum között

  $$\frac{|F_1 \cap F_2|}{|F_1 \cup F_2|}$$

  lesz, ahol F_1 az egyik dokumentum fingerprintjeinek halmaza, F_2 a másik dokumentum fingerprintjeinek halmaza.

2.6. Statisztikai funkciók

Jelenleg csak egyszerű statisztikák érhetőek el a rendszerben: az oktatók csoportonként és kategóriánként lebontva ellenőrizhetik az összes felhasználó haladását hétről hétre. Ha egy hallgató egy feladatra több megoldást is feltöltött, akkor a rendszer az időrendben utolsó megoldást számítja bele a statisztikákba. A rendszer mostani verziója a következő felhasználónkénti, illetve feladatonkénti statisztikai kimutatásokat készíti el:

1. Helyes, hibás, nem elbírált megoldások száma felhasználónként

   Ebben a nézetben a felhasználók összes feltöltött megoldásának számát listázza ki a rendszer, felhasználónként, oszlopdiagramon. Lekérhetőek egyes felhasználók megoldásai, vagy egy csoporté, illetve lehetséges a vizsgált feladatok körét szűkíteni egy-egy kategóriára.

2. Helyes, hibás, nem elbírált megoldások száma feladatonként

   Ebben a nézetben egy feladatra érkezett összes megoldás adatai tekinthetőek meg az első ponthoz hasonlóan itt is szűkíthetőek a mérési adatok, egy-egy felhasználói csoportra, így például összehasonlítható egymással több évfolyam teljesítménye.

3. fejezet

Továbbfejlesztési lehetőségek és tapasztalatok

3.1. Tapasztalatok

Az automata javításról érkező hallgatói visszajelzések szinte kivétel nélkül pozitívak voltak, a véleményük szerint a hetente kiadott gyakorló feladatok nagy mértékben segítették a C/C++ programnyelvek használatának elsajátítását. Bár a hallgatók számára már a korábbi rendszer is nagy segítséget nyújtott, amely csak a feladatok fordítását végezte el, és kézzel történt a megoldások helyességének vizsgálata, az új házi feladat kezelő rendszerben megvalósított automatikus kiértékelésnek köszönhetően a hallgatók azonnal kapnak egy előzetes visszajelzést, hogy a feladatmegoldásaik helyesek-e.

A félév elején a hallgatók és az oktatók számára is nehéz volt megszokni a rendszer működését, mivel az automata tesztkörnyezet szigorúbban javítja ki a feladatokat, mint azt korábban a javítók tették: ha a kimenet nem felel meg az előre meghatározott teszteset mintamegoldásának, a rendszer azonnal elutasítja a hallgató megoldását. Általában az első eset után, amikor egy-egy hallgató rossz formátumban írta ki a kért feladat megoldását, megszokták a rendszer használatát, ezzel később kevés probléma volt. Ehhez hasonló gyakori hiba volt, amikor a program futásának vége előtt a hallgatók megoldása egy billentyű leütésére várt – a rendszer az ilyen megoldásokat hibásnak jelölte, mivel a program nem fejezte be futását a megadott időkereten belül.

Oktatói részről a rendszer az eddigi tapasztalataink alapján beváltotta a hozzá fűzött reményeket – két félév alatt eddig hozzávetőleg 2000 feltöltött feladatmegoldás érkezett be, ebből az első 1000 javítása kézzel történt, mivel az automata tesztrendszer még nem működött. A 2017/18-as tanév őszi félévében ugyanakkor a feladatok javítása már automatikusan történik, kézi ellenőrzéssel – így megelőzhető, hogy a hallgatók csak egy-egy adott tesztesetre optimalizált megoldást adjanak be. Az oktatók által végzett munka mennyisége ugyanakkor így is jelentős mértékben csökkent: csak azokat a házi feladatokat kell ellenőrizni, amelyeknek legalább a kimenete helyes, egy megadott időn belül befejezi a futását, és nincs benne memóriaszivárgás.

A 2016/17-es tanév tavaszi félévének hallgatói teljesítményét összehasonlítjuk majd a 2017/18-as tavaszi félévével, így mérhetővé válik, hogy az automatizált javítás segített-e a hallgatóknak a feladatmegoldás során, illetve, hogy az automata rendszernek köszönhetően mennyivel kevesebb javítómunkát kellett kézzel elvégezni.

3.2. Továbbfejlesztési lehetőségek

Plágiumkereső

A plágiumkereső jelenlegi állapotában is jelentős mennyiségű plágiumgyanús programkódrészletet képes felismerni, ám több irányba is lehetséges a továbbfejlesztése:

- A felhasználó kódolási szokásainak követése

Ha képesek vagyunk különböző stílusjegyeket tárolni egy felhasználó kódjáról, és a tárolt adatok alapján megítélni, hogy általánosságban milyen programkódot ír, akkor felismerhető az újonnan feltöltött megoldásokban az ehhez képest túl nagy eltérés. Ez arra utalhat, hogy a felhasználó külső segítséget vett igénybe a feladat megoldásához. Ilyen jellemzők lehetnek például a forráskódot és kommentet tartalmazó sorok aránya; a for vagy while ciklus előnyben részesítése; hogy a felhasználó a '{' karaktert új sorba írja-e, illetve egyéb helyeken whitespace karakterek használata.

Ezáltal az is felismerhető, ha egy hallgató a megoldását nem másolta másik hallgatóról, hanem valaki más írta meg helyette a feladatát - ekkor a jellemző kódolási szokásoktól való eltérés felismerhető és az oktatót figyelmeztetni tudja a rendszer.

- A programkódok felépítésének elemzése

  Mivel jelenleg egy stringeken alapuló megoldást használunk, ha a programkód struktúrája jelentősen megváltozik, a plágiumkereső tévedhet. Ez kivédhető, ha a program struktúráját elemezzük, és abban keresünk tipikusan felcserélhető elemeket (for ciklus helyett while, if elágazás két ágának felcserélése, stb.).

Statisztikai funkciók

Jelenleg a rendszer csak alapvető statisztikai funkciókat támogat: a felhasználók helyes megoldásainak számát veszi alapul a statisztikák elkészítéséhez. Természetesen ennél jóval komplexebb kimutatások készíthetők a feltöltött megoldások alapján, ezek fejlesztése folyamatban van. A terveink szerint a következő jellemzőkről készítene a rendszer statisztikákat:

- Kód metrikák

  A feltöltött megoldásokról a lehető legkülönfélébb adatok gyűjthetők össze – köztük egyszerűbbek, mint például a feltöltött és lefordított megoldások futási sebessége, amelyeket már most is mér és eltárol a rendszer; illetve bonyolultabbak, például a forráskód bonyolultsága, a programsor - komment arány.

  A terveink szerint a következő statisztikai adatokat szeretnénk összegyűjteni a feltöltött megoldásokról:

  - forrássorok száma;
  - kommentsorok száma;
  - futási idő;
  - memóriahasználat;
  - a lefordított bináris fájl mérete;
  - a programkód bonyolultsága - a feltöltött kód bonyolultságának mérésére két módszert kívánunk használni: a cyclomatic complexity és function point analysis nevű eljárásokat. Előbbi a program forráskódjából egy gráfot állít elő, és a gráf szerkezetét vizsgálja, míg utóbbi a programban lévő függvények felépítéséből következtet a program bonyolultságára. [7]

- A feltöltött kód "szépsége"

  A mérések alapjául egy statikus kódelemző szolgál, amely a gyakran elkövetett hibákat automatikusan megkeresi a feltöltött forráskódban, majd súlyosság szerint több kategóriába sorolja őket. A hiba súlyosságától függően a feltöltött megoldás pontokat kap, majd ezen pontok összegzése után a rendszer elosztja az összeget a programsorok számával.

- A felhasználók fejlődésének figyelése

  A fenti adatokat felhasználva, illetve a jelenleg is mért adatok alapján egyedi visszajelzést adnánk a regisztrált felhasználóknak, amivel nyomon követhetnék tudásuk fejlődését.

Egyéb programnyelvek támogatása

A HAWEB jelenleg a C és C++ programnyelveket támogatja, de úgy terveztük a rendszert, hogy könnyen bővíthető legyen – a terveink szerint a különböző programnyelvek közül első lépésként a Java és C# programnyelveket akarjuk támogatni, mindkettő esetében a mostani tesztkörnyezet megtartása mellett egy egységteszt alapú tesztelést támogató megoldást is be akarunk építeni, ami a megoldások részletesebb tesztelését is lehetővé tenné.

Piaci felhasználás

Az elkészült rendszert a későbbiekben szeretnénk piaci körülmények is megméretni. Az első felmérések alapján a következő igényekkel - mint lehetséges fejlesztési célok - találkoztunk:

- Felhasználók tudásának felmérése

  A jelenlegi oktatási felhasználás során többnyire kezdő programozók veszik igénybe a rendszer. Azonban vannak haladó, előismerettel rendelkező felhasználók is, akiknek szeretnénk értékelni az előzetes tudását, szeretnénk látni, hogy egy adott skálán hol helyezkedik el a sokasághoz (többi programozóhoz) képest.

  Ehhez egy nagyobb mennyiségű adatbázist szeretnénk építeni, amellyel értékelni lehet a forráskódok, algoritmusok "minőségét" megfelelő súlyozással. Ebből egy összesített pontszám számható, amely kellően sok felhasználó és eredmény esetén egy 0-100 közötti skálára átváltható, ahol a legjobb felhasználó a 100, a legrosszabb pedig a 0 pontos.

  Ez az eljárás természetesen ciklikusan ismételhető, így a felhasználó fejlődése is felmérhető.

- Versenyek rendezése

  Habár a HAWEB már most is kétszintű, és a kötelező feladatokon kívül vannak szorgalmi feladatok, azt láttuk, hogy érdemes lenne a kettőt különválasztani (jelenleg nincs különbség technikailag köztük), és a plusz feladatokat versenyként meghirdetni. Ezzel olyan (akár valós) problémákat lehetne feladni a felhasználóknak, amikkel a problémamegoldási és algoritmizálási képességeiket lehetne felmérni, és a feladatokon keresztül fejleszteni. A verseny természetesen lehet éles verseny is, ahol az indulók láthatják a többiek eredményeit és a teljes rangsort.

Szükséges változtatások

Ahhoz, hogy a fenti célokra is használható legyen a keretrendszerünk, bizonyos változtatásokat végre kell majd hajtanunk, és módosítani kell az eredeti megoldásokat:

- Kategóriák kezelése

  A jelenlegi kategória rendszerünk szigorúan egy kategóriához rendeli hozzá a feladatokat, de ahhoz, hogy több különböző szereplő használhassa ugyanazon feladatokat, a kategóriák kezelését meg kell változtatnunk hogy egy feladat több kategóriához is tartozhasson.

- Jogosultságkezelés

  Az előző ponttal összefüggésben a jogosultságok kezelését úgy kell átdolgoznunk, hogy az egyes felhasználókhoz vagy felhasználói csoportokhoz tartozó jogosultságokat kategóriánként osszuk ki - például egy adott felhasználó csak egy adott kategóriához tartozó statisztikákat lásson, de a kategóriában lévő feladatokat ne tudja szerkeszteni.

- Statisztikák

  Az elérhető statisztikai funkciók körét bővíteni kell, illetve egy sorrendet kell felállítani. Ezt a terveink szerint úgy oldjuk meg, hogy bizonyos kiválasztott statisztikai adatokhoz az oktatói felületen egy pontszámot lehet majd hozzárendelni, és a rendszer a hozzárendelt

pontszámot lineárisan osztja ki a feladatot megoldó felhasználók között (a legjobb eredményt elérő felhasználó megkapja a maximális pontszámot, a legrosszabb pedig nem kap pontot). Így a statisztikai adatok súlyozásával alakul ki a végső sorrend.

4. fejezet

Összefoglalás

Munkánk során egy programozás oktatást segítő portált fejlesztettünk, amelynek legfőbb tulajdonságai a feltöltött megoldások automatikus fordítása és ellenőrzése, illetve az alkalmazott plágiumkereső rendszer.

A rendszer jelenlegi képes a felhasználó profilok kezelésére, és minden azokhoz kapcsolódó művelet elvégzésére (regisztráció, bejelentkezés, profil megtekintése, jelszóváltoztatás, emailek küldése, stb.) illetve a felhasználókat képes csoportokba rendezni, az egyes felhasználóknak illetve felhasználói csoportoknak jogosultságokat kiosztani.

Feladatok terén jelenleg csak C és C++ nyelvű programkódokat támogat, amelyeket először Java és C# nyelvek támogatásával akarunk bővíteni. A feltöltött megoldásokat a rendszer automatikusan kijavítja, amely mellett lehetőséget ad a kézi ellenőrzésre. Bemutattunk több megoldást a plágiumgyanús esetek keresésére, és ezek közül egyet mi magunk is implementáltunk. A tervezés során megvizsgáltuk a biztonságtechnikai megoldásokat amelyekkel a rosszindulatú kódoktól védhetjük a rendszert, és aktívan használjuk is e megoldásokat.

A rendszerrel a kitűzött célokat eddigi tapasztalataink alapján sikerült elérni, valóban nagy mennyiségű időt spórol meg az oktatóknak, a hallgatók a korábbiaknál hamarabb kapnak visszajelzést a munkájukról, és a visszajelzések alapján egy kényelmesen és könnyen használható rendszerben kezelhetik a házi feladataikat.

Jelenleg a statisztikai funkciók körének bővítésén, illetve a plágiumkereső továbbfejlesztésén dolgozunk. Ezen kívül terveink között szerepel a jelenlegi kategóriarendszer és a jogosultságkezelés átdolgozása. Közben az oktatási célok mellet fel akarjuk mérni, hogy piaci körülmények között a rendszerünk bevezethető-e, hogyan vezethető be. Hosszabb távon felmerült még egy e-learning portál indítása, amelyet a HAWEB gyakorlófeladatokkal támogatna a tanulás segítése érdekében. Egy ilyen portál megvalósításához még egy hasznos funkció implementálása merült fel: ha a rendszer figyeli a felhasználók teljesítményét, akkor képes ahhoz alkalmazkodni, és szükség esetén könnyebb vagy nehezebb feladatokat adni a felhasználó tudásának megfelelően.

Irodalomjegyzék

[1] David Gitchell, Nicholas Tran
Sim: a utility for detecting similarity in computer programs
Thirtieth SIGCSE technical symposium on Computer science education, 1999, pp. 266-270

[2] J. Brunsmann, A. Homrighausen, H.-W. Six, J. Voss
Assignments in a Virtual University - The WebAssign System [Online]
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.3768&rep=rep1&type=pdf

[3] Christoph Beierle, Marija Kulas, Manfred Widera
Automatic Analysis of Programming Assignments [Online]
http://cs.emis.de/LNI/Proceedings/Proceedings37/GI-Proceedings.37-17.pdf

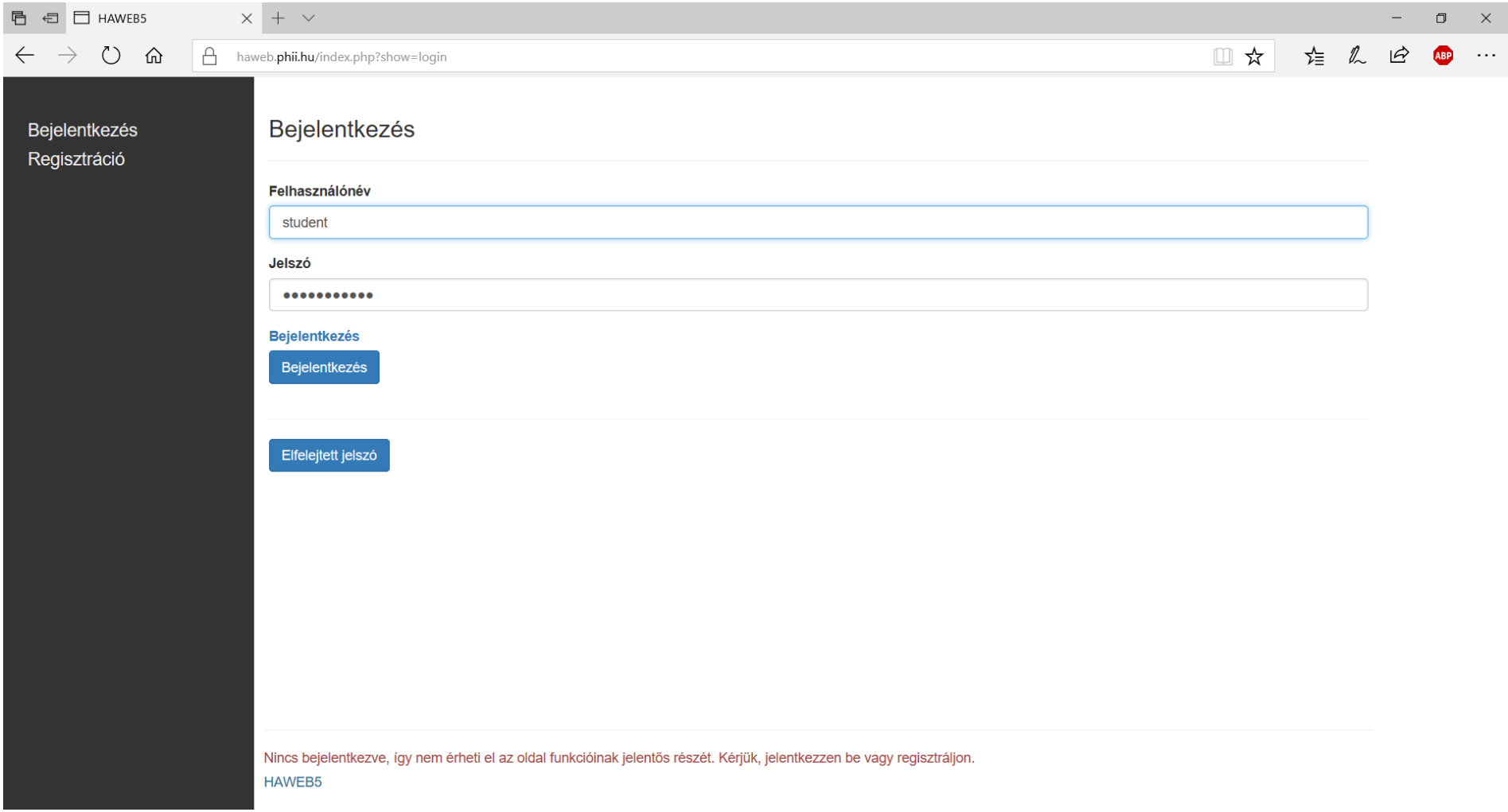
[4] Andrea Sterbini, Marco Temperini
Automatic correction of C programming exercises through Unit-Testing and Aspect-Programming [Online]
http://twiki.di.uniroma1.it/pub/Users/AndreaSterbini/Ricerca/6-eista04b.pdf

[5] Thomas Bakker
Plagiarism Detection in Source Code Bachelor thesis, July 10th 2014 [Online]
https://www.math.leidenuniv.nl/scripties/BSC-Bakker.pdf

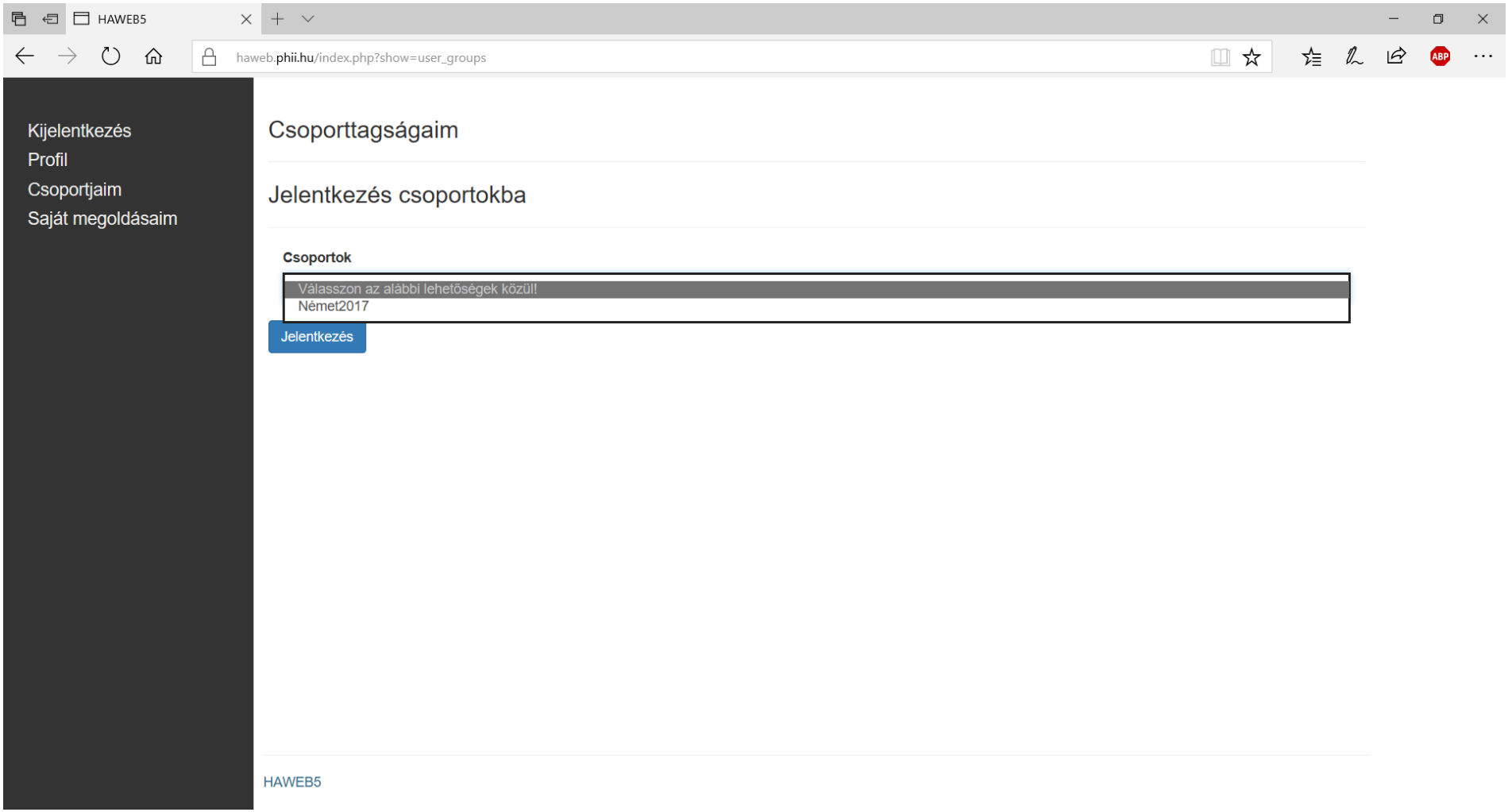
[6] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken
Winnowing: Local Algorithms for Document Fingerprinting [Online]
http://theory.stanford.edu/ aiken/publications/papers/sigmod03.pdf

[7] C. Jones
Software metrics: good, bad and missing
Computer, 1994, pp. 98-10

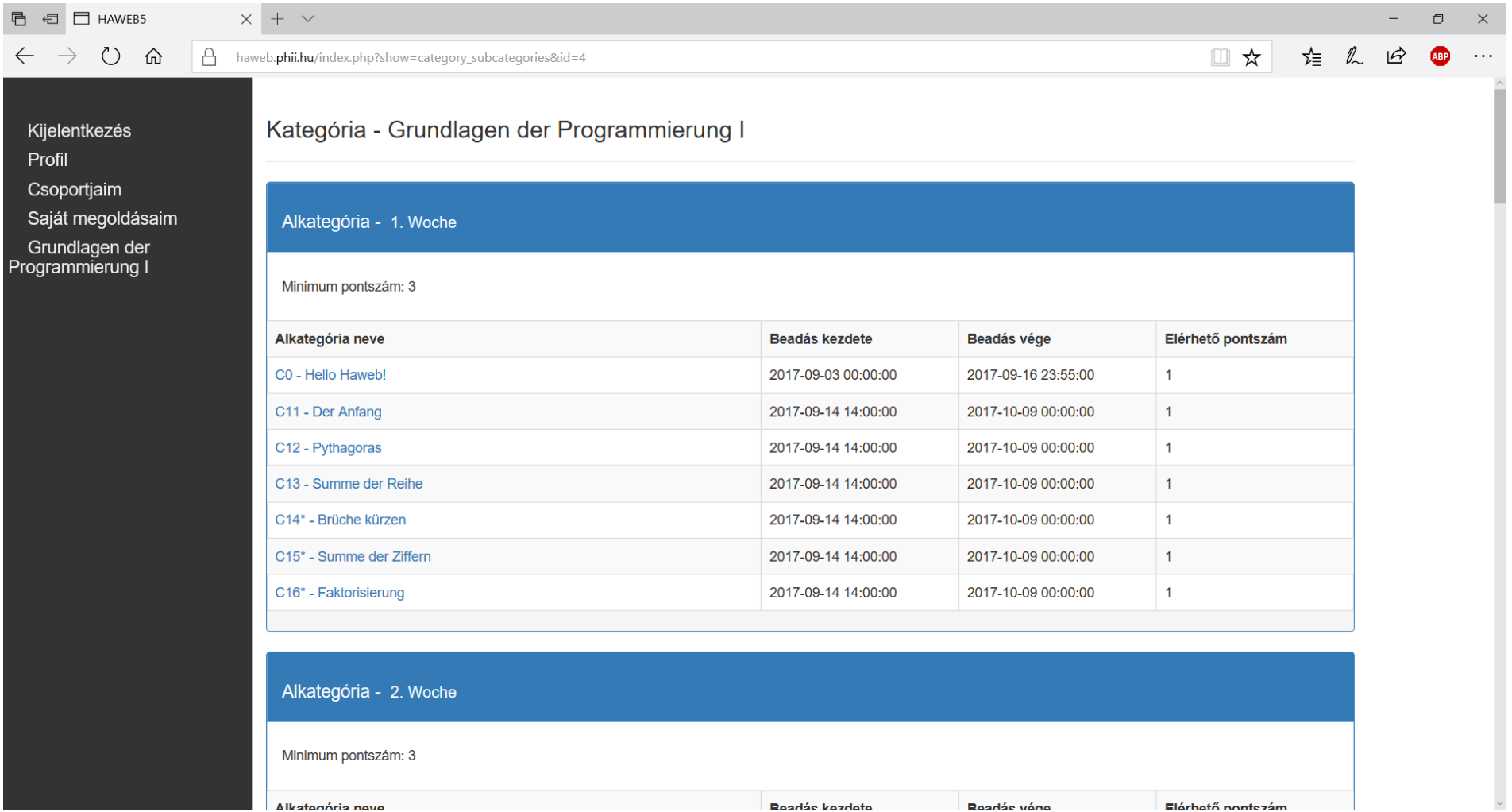
A függelék – Felhasználói felület



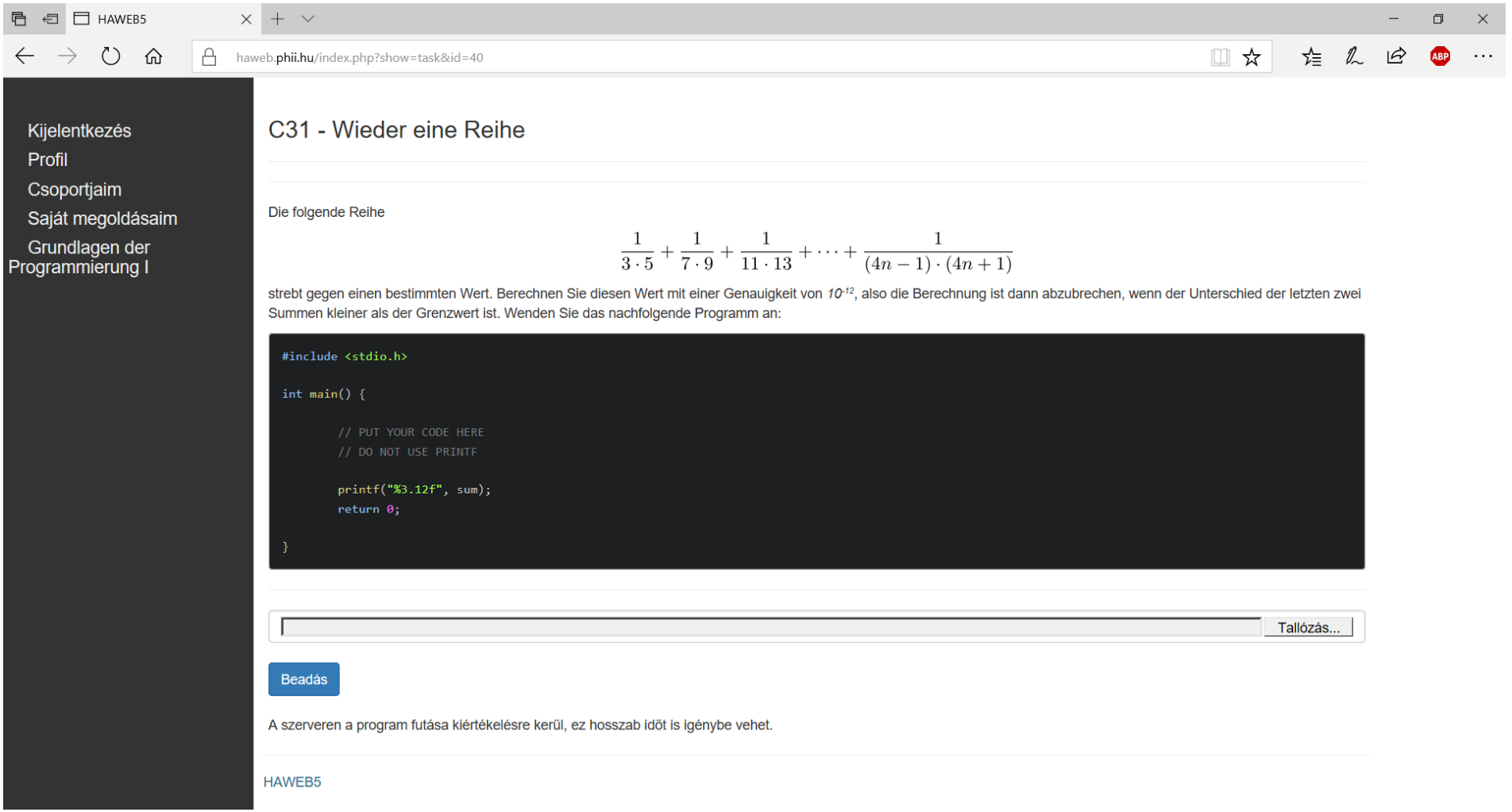
A.1. ábra. A bejelentkezés képernyő



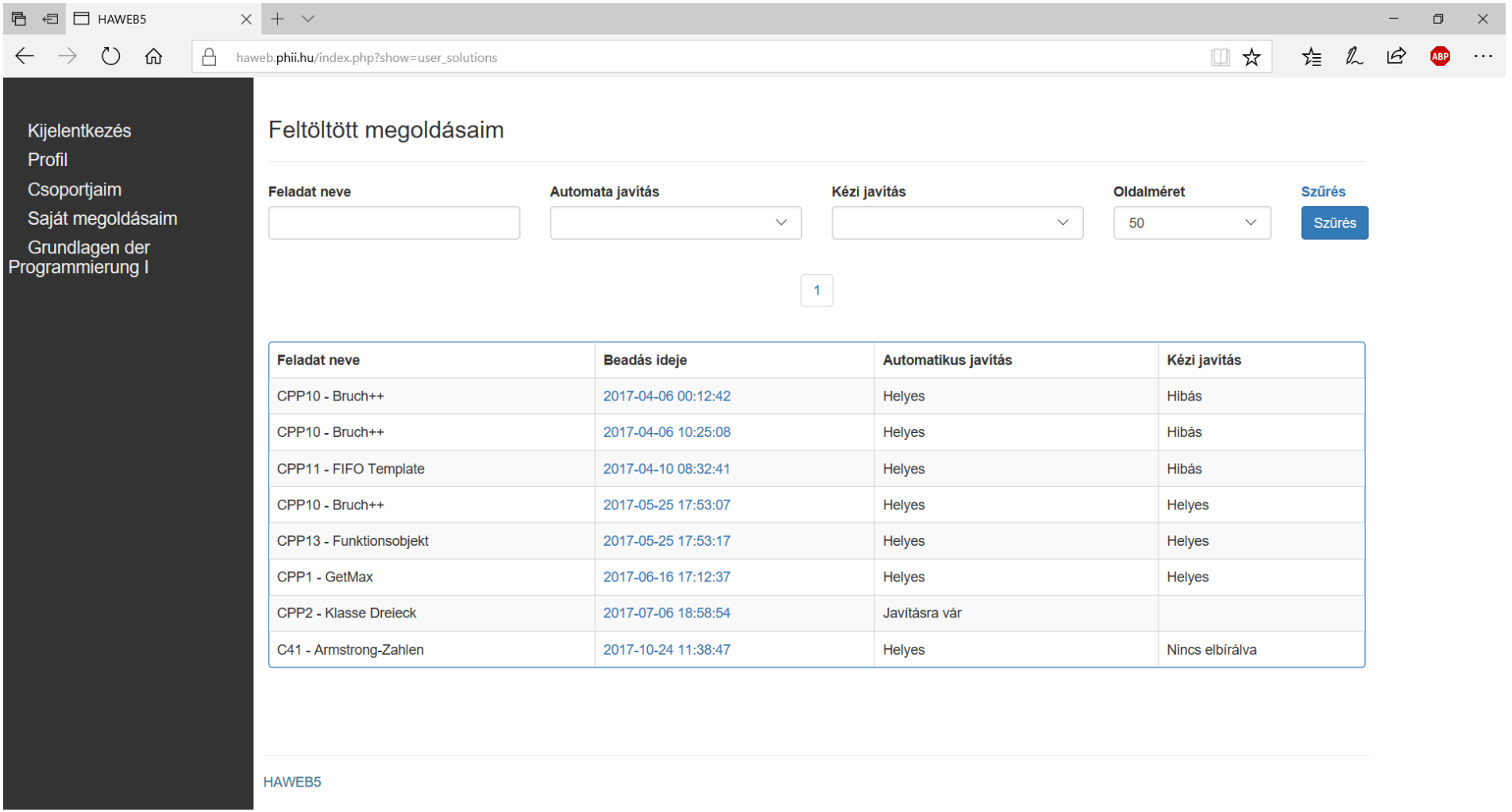
A.2. ábra. A felhasználói csoportok listája



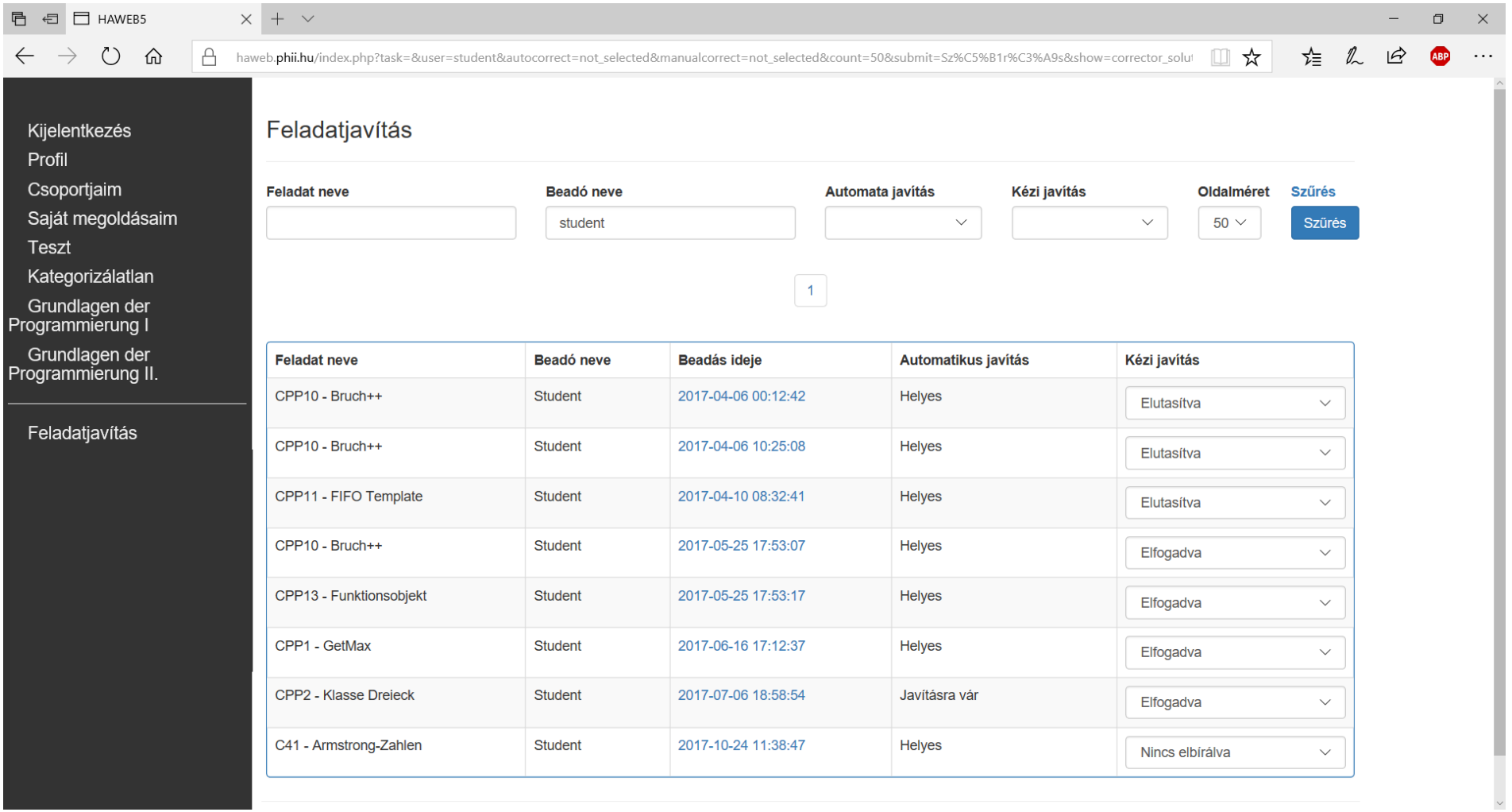
A.3. ábra. Egy kategóriába tartozó feladatok listája



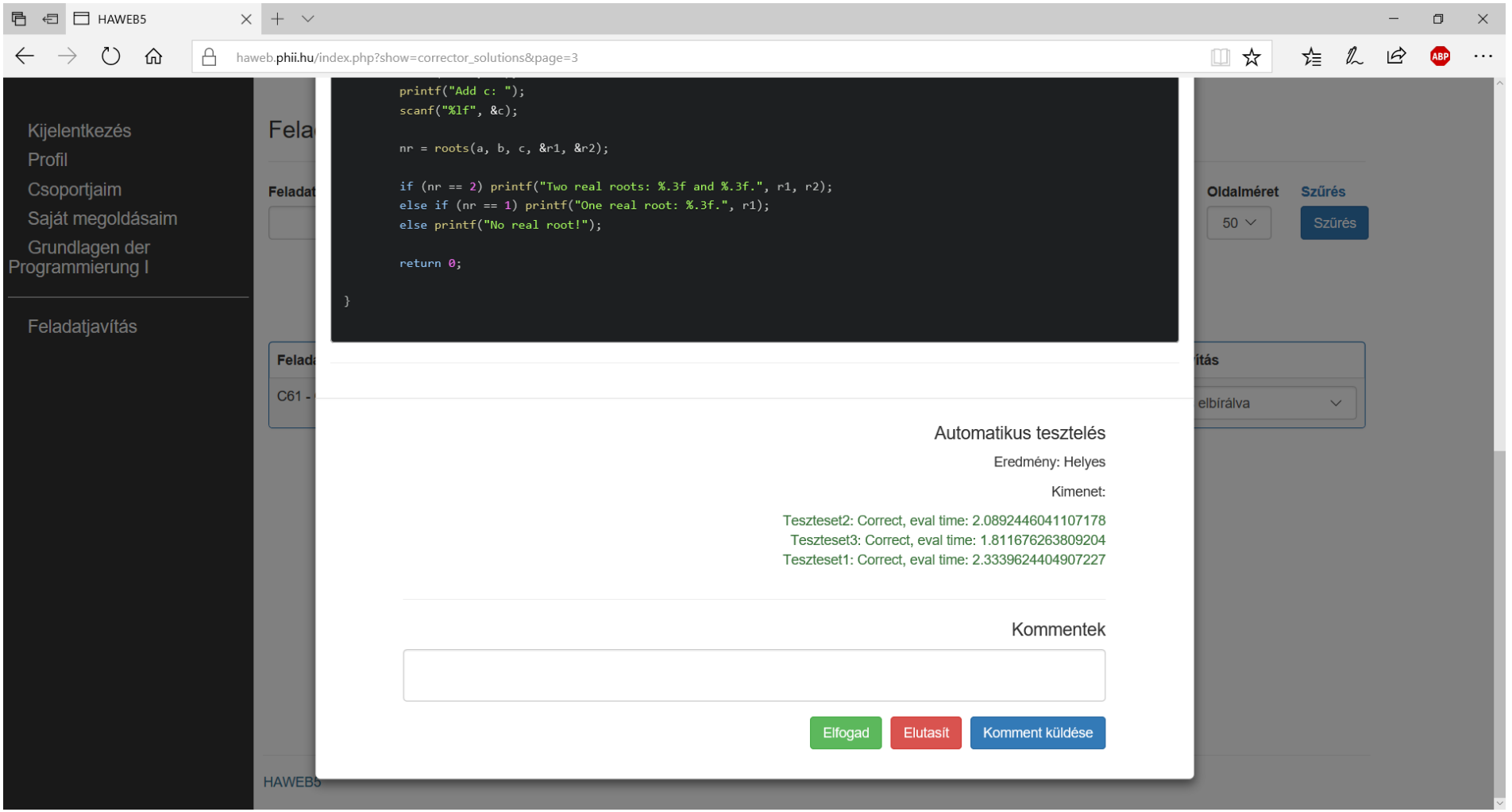
A.4. ábra. Egy konkrét feladat



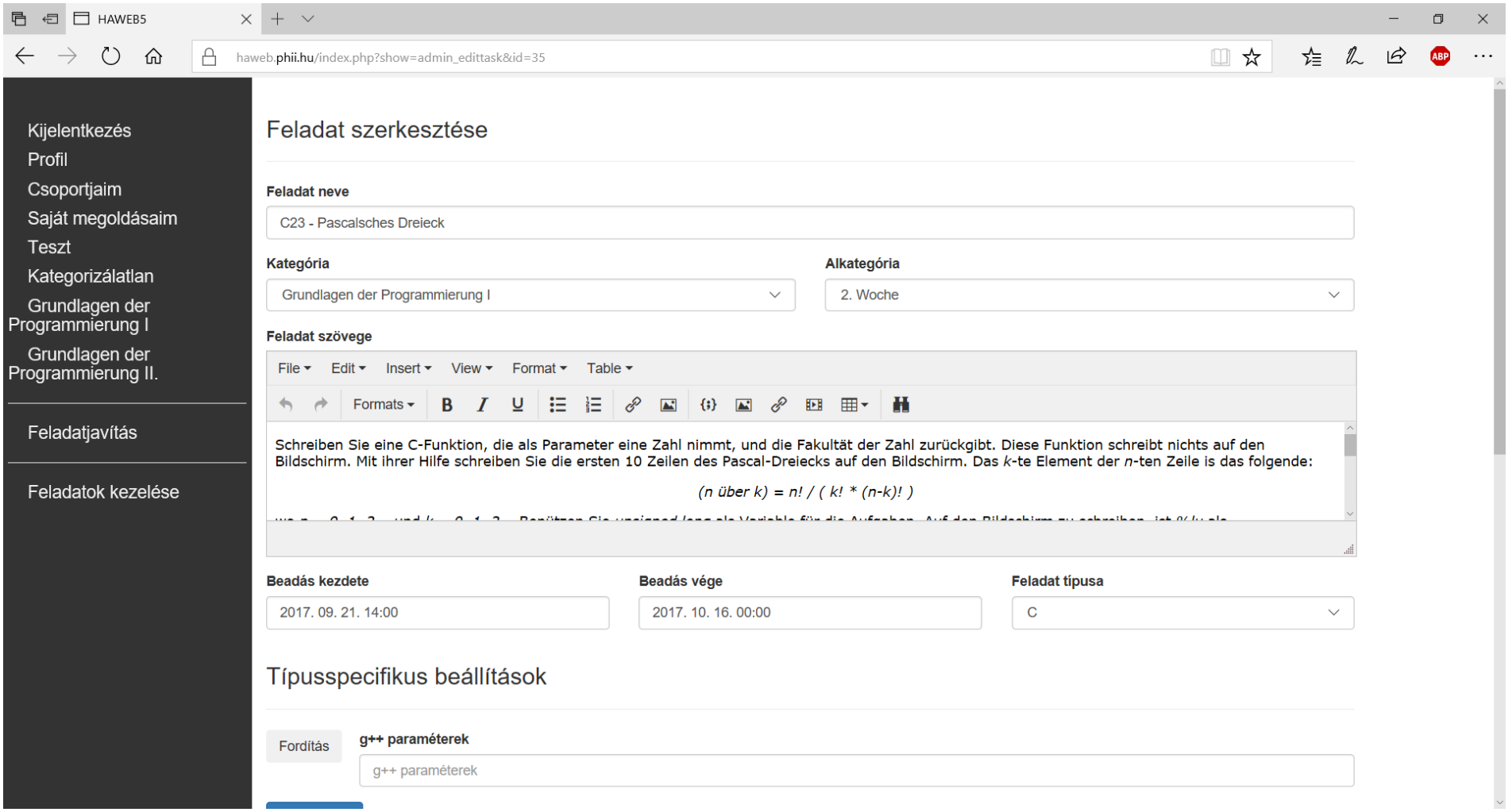
A.5. ábra. A felhasználó által feltöltött megoldások



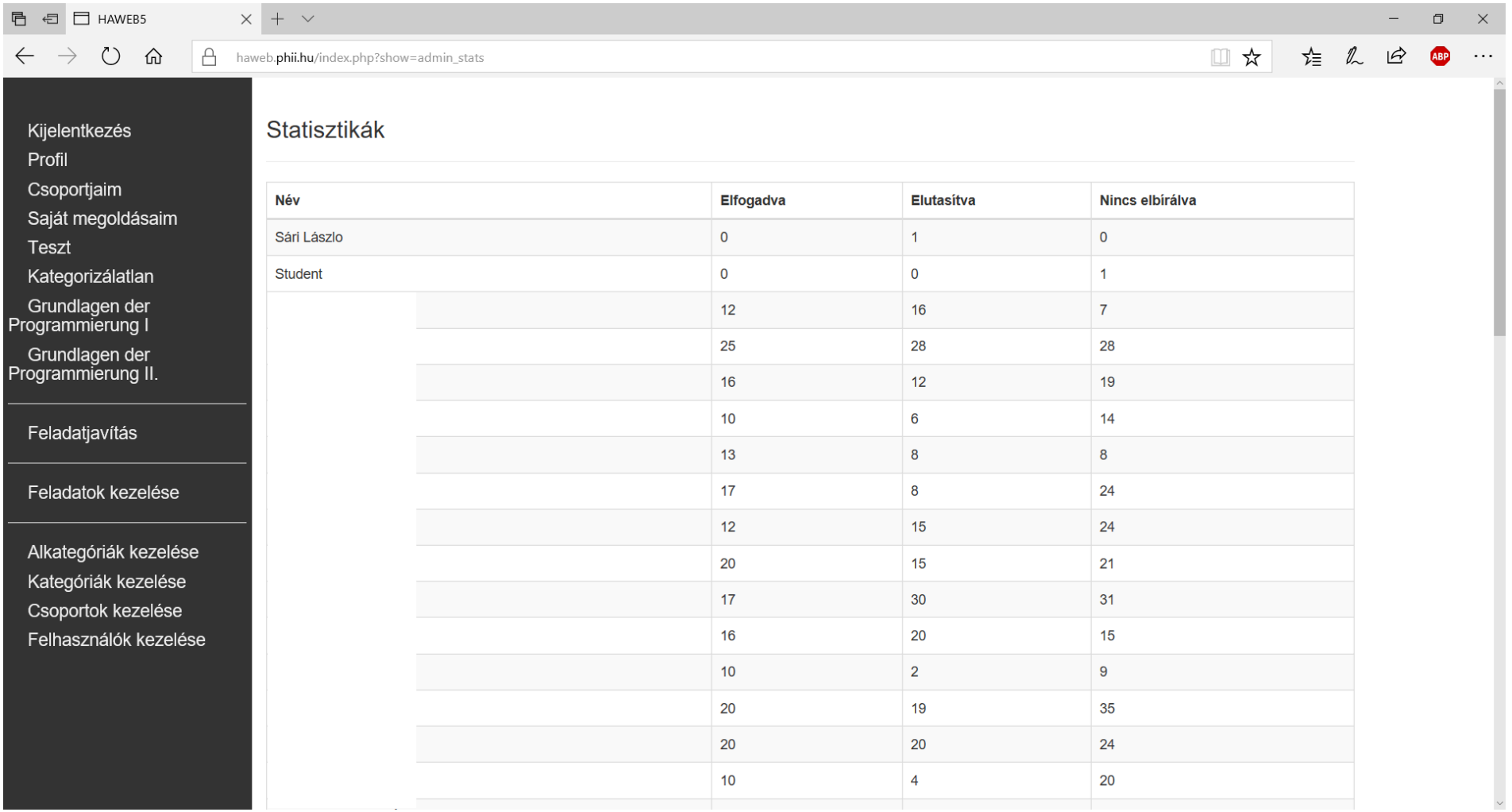
A.6. ábra. A javítótanárok által látott felület



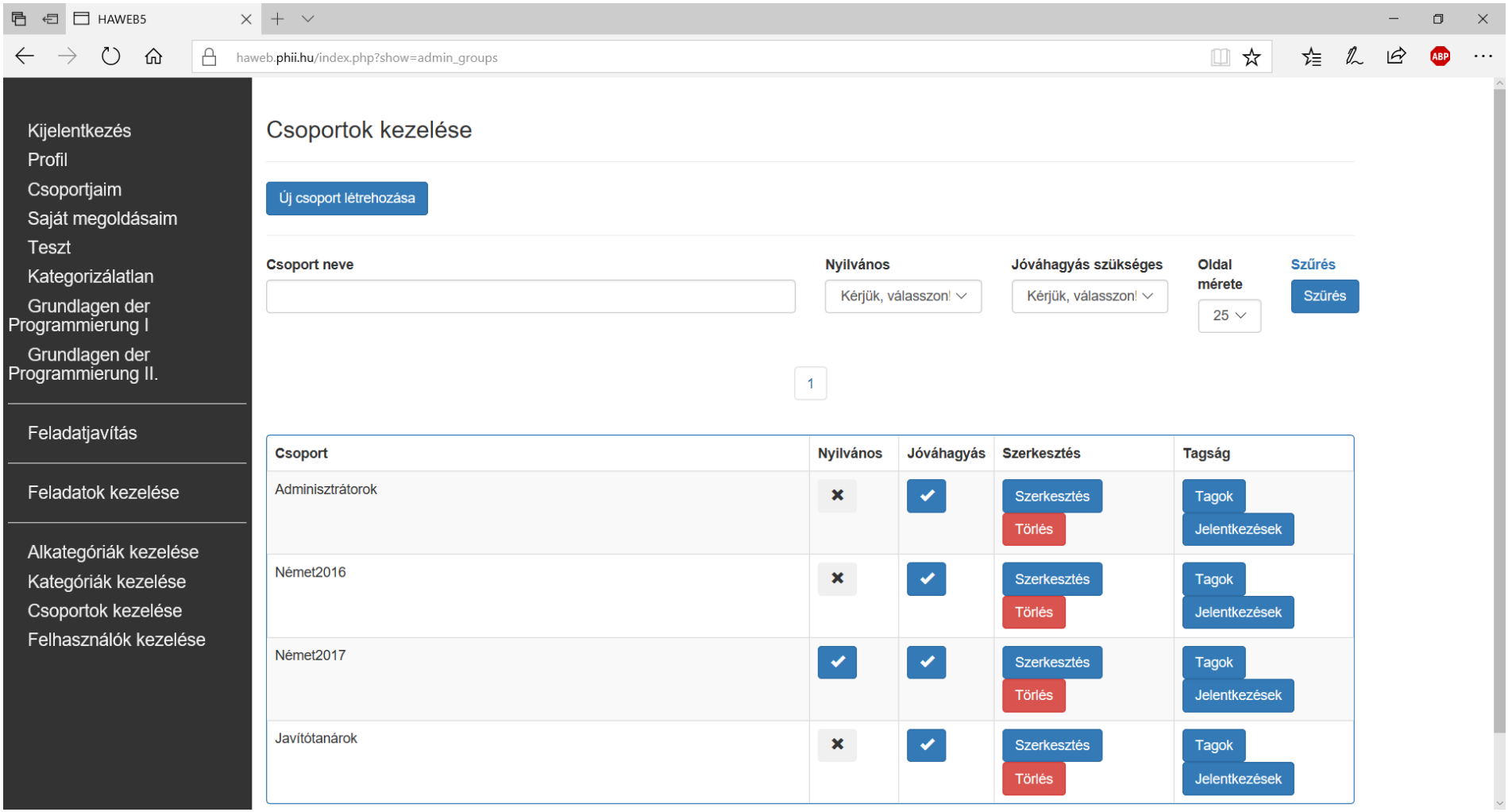
A.7. ábra. Egy feladatmegoldás a javítótanári felületen



A.8. ábra. A feladatszerkesztő felület



A.9. ábra. Az egyszerű statisztikákat mutató felület



A.10. ábra. A felhasználói csoportok listája adminisztrátori oldalon