DIPLOMAMUNKA

Dobi Attila László

Debrecen

2008

Debreceni Egyetem
Informatika Kar

Alkalmazásfejlesztés webre

Témavezető:
Dr. Juhász István
Egyetemi Adjunktus

Szerző:
Dobi Attila László
Programtervező Matematikus

Debrecen
2008

Ezen diplomamunka dolgozattal szeretném megköszönni tanáraimnak, különösképpen témavezetőmnek, Dr. Juhász István tanár úrnak az egyetemi évek alatt nyújtott segítséget, bátorítást és szakmai támogatást.

Tartalom

Ábrajegyzék

1. Bevezetés

1.1. Diplomamunkám témája

Ez a dolgozat egy konkrét alkalmazás fejlesztését írja le a PHP és a Smarty sablonkezelő rendszer segítségével. Az alkalmazás célja egy egyetemen publikált jegyzetek egy helyről való elérése. Másodlagos célként megemlíthető a publikáló tanárok adatainak megtekintése. Egy ilyen alkalmazás az egyetem hallgatói számára megkönnyítheti az elektronikus tananyagok elérését, felhasználását.

1.2. Témaválasztás

Az egyetemi képzés során a programtervező matematikus hallgatók főként elméleti tudást szereznek. A gyakorlati tudás megszerzésere és az elméletben tanultak alkalmazására csak néhány lehetőség adódik. Az egyik ilyen a rendszerfejlesztés technológiája tantárgy, a másik a diplomamunka. Ezért választottam egy konkrét alkalmazás elkészítését, melynek fejlesztése során az elméletben tanultak nagy részét ki tudom próbálni, jobban meg tudom ismerni. Megpróbáltam követni egy teljes alkalmazásfejlesztési módszert, a tervezéstől kezdve, az implementáción át a tesztelésig.

Számos programozási paradigmát illetve nyelvet ismertem meg, sajátítottam el az egyetemi évek alatt. Hiányoltam, hogy az internethez kapcsolódó programozási nyelvekkel illetve technológiákkal igen kevés tantárgy foglalkozik és amelyik foglakozik is, az is csak felszínesen érinti. Véleményem szerint a munkaerő piacon való elhelyezkedés során előnyt jelent ezen ismeretek birtoklása.

Ezért döntöttem úgy, hogy ilyen téren bővítem tudásom és szerzek további tapasztalatokat.

2. Elméleti áttekintés

2.1. Az objektumorientált PHP

A PHP egyike a leginkább elterjedt script nyelveknek. Az internetes oldalak illetve alkalmazások nagy százalékát PHP segítségével fejlesztették és fejlesztik. A PHP nyelv nagyon hasonlít az ANSI C-hez.

Az egyik ok amiért a PHP ilyen nagy népszerűségnek örvend, hogy könnyen tanulható, nincsenek benne konvenciók, keverhető a prezentációs réteg az üzleti logikával.

Ahogy a fejlesztendő alkalmazások mérete nőtt úgy váltak egyre kezelhetetlenebbé.

A PHP-t kifejlesztésekor az objektumorientált vonások nem voltak részesei a nyelvnek.

A PHP3 bevezette az objektumorientáltság alapjait, amelyet tovább fejlesztettek a PHP4-ben. A PHP5 megjelenésével egy teljesen új objektum modell jelent meg a nyelvben.

2.1.1. Különbség a PHP4 és PHP5 objektumorientáltsága között

Az objektum fogalma PHP5-ben nagy mértékben különbözik a PHP4-beli objektum fogalomtól. Az egyik legalapvetőbb különbség, hogy a PHP4 objektumában minden „nyitott" volt, nem volt semmiféle megszorítás a metódusok illetve attribútumok használatát illetően. Nem lehetett használni a `private`, `public`, `protected` láthatósági módosítókat.

PHP4-ben privát láthatóságú metódust a dupla aláhúzás jellel(„__") lehetett deklarálni, de ez nem jelentette ténylegesen, hogy a metódust nem lehet hivatkozni az osztályon kívülről, egyszerű konvenció volt.

PHP4-ben megtalálható az interfész fogalma, de az `abstract` és `final` kulcsszavak nem, nincs többszörös öröklődés interfészek esetén, nincs destruktor. Ezeket a PHP5 vezeti be. Majdnem minden statikus PHP4-ben, osztály metódusa hivatkozható anélkül, hogy példányosítottuk volna az osztályt. PHP5-ben ez már nem lehetséges.

PHP5-ben jelenik meg a kivételkezelés is, illetve a metódus túlterhelés a `__get()`, `__set()` metódusok segítségével.

2.1.2. Néhány alap fogalom

Módosítók

A PHP5 vezeti be, segítségükkel szabályozni tudjuk az attribútumok láthatóságát, ezáltal hozzáférhetőségét.

Az egyes módosítók leírása:

Private(privát):attribútumok illetve metódusok ily módú deklarációja lehetővé teszi, hogy az osztályon kívül ne hivatkozhassuk őket. Ellenkező esetben hibaüzenetet kapunk.

Public(publikus):azok az attribútumok, illetve metódusok amelyeket explicit módon nem deklarálunk privát, vagy védett láthatóságúnak, publikus láthatóságúak lesznek, tetszőlegesen hivatkozhatóak más osztályból.

Protected(védett): az ilyen módosítóval deklarált attribútum illetve metódus csak az adott osztály alosztályaiból lesz elérhető.

Konstruktor, destruktor

A konstruktor az a metódus amely automatikusan végrehajtódik amikor példányosítunk egy osztályt. PHP5-ben két lehetőség van arra, hogy egy osztély konstruktorát létrehozzuk. Az egyik, hogy egy `__construnct()` nevű metódust definiálunk az osztályon belül, a másik, hogy az osztály nevével azonos nevű metódust hozunk létre. Ha egy osztályban mind a két típusú konstruktor definiálva van, akkor a `__construct()` hajtódik végre, a másikat figyelmen kívül hagyja a PHP5.

A konstruktorhoz hasonlóan létezik egy destruktor metódus is, amely az objektum megsemmisítéséért felelős. Létrehozhatunk destruktort explicit módon ha egy metódusnak a `__destruct()` nevet adjuk. Ez a metódus automatikusan hívódik meg amikor a scriptünk végrehajtása befejeződik. Nem kötelező destruktor használata, a PHP5-ben egy beépített szemétgyűjtögető mechanizmus működik.

Öröklődés

Az objektumorientált programozás egyik legfontosabb jellemzője az öröklődés. Segítségével létrehozhatunk új osztályokat már létezőkből, kihasználva a már létező viselkedésmódot, és csak szükség szerint alakítva azt.

Az objektumorientált PHP nem támogatja a többszörös öröklődést, ez azt jelenti, hogy egy osztálynak csak egy szülőosztálya lehet. Osztályok közötti öröklődési kapcsolatot az `extends` kulcsszó segítségével hozhatunk létre. A szülő osztályra a leszármazott osztályból a `parent` kulcsszóval hivatkozhatunk.

Metódus túlterhelés

A származtatott osztályban tetszés szerint túlterhelhetjük a metódusokat, oly módon, hogy a túlterhelni kívánt metódus nevével azonos nevű metódust hozunk létre. Ha egy alosztályban deklarálunk egy olyan változót amely a szülő osztályban is létezik, akkor a változóhoz való hozzáférés az alosztálybeli változón fog megtörténni.

Ahhoz, hogy egy metódust ne lehessen túlterhelni a leszármazott osztályban, a metódus deklarációjában adjuk meg a `final` kulcsszót.

Osztályokat is deklarálhatunk `final`-ként. Ekkor az adott osztályból nem lehet származtatni gyermek osztályokat.

Interfész és absztrakt osztály

Az interfész egy üres osztály, amely a metódusok deklarációját tartalmazza. Tehát minden olyan osztálynak amely implementálja az interfészt, tartalmaznia kell a interfészben deklarált metódusok megvalósítását is. Egy osztály és egy interfész kapcsolatát az `implements` kulcsszó segítségével adhatjuk meg.

Absztrakt osztály nagyon hasonló az interfészhez, azzal a különbséggel, hogy a benne szereplő metódusoknak lehet törzsük. Absztrakt osztály magvalósításánál az `extends` kulcsszót kell használni és nem az `implements`-et.

Absztrakt osztályt nem deklarálhatunk `final`-nak, mivel a `final` kulcsszóval deklarált

osztályból nem lehet származtatni.

Metódusokat is lehet `abstract`-nak deklarálni. Absztrakt metódus deklarációjánál nem adhatunk meg törzset. A szülő osztályban lévő absztrakt metódust a gyermek osztálynak felül kell definiálni.

2.1.3. A PHP5 által felkínált új lehetőségek

Hasznos függvények objektumokhoz

`class_exists()`: segítségével megtudhatjuk, hogy egy osztály létezik e vagy sem.

A következőkben bemutatom a használatát egy példán keresztül.

```php
<?
    include_once('class.User.php');
    if(class_exists('User')) {
        $user = new User();
    }
    else {
        die('Az osztály nem létezik.');
    }
?>
```

1. ábra: class_exists() használata

`get_declared_classes()`: az aktuálisan betöltött osztályok listáját adja vissza.

`method_exists(), property_exists()`: segítségével megtudhatjuk, hogy egy metódus illetve attribútum elérhető e egy osztályon belül. A fent említett függvények igazzal térnek vissza ha a metódus illetve attribútum publikusnak volt deklarálva.

`is_a()`: segítségével osztály típusát ellenőrizhetjük.

A következőkben bemutatom a használatát egy példán keresztül.

```php
<?
    class ParentClass {}
    class ChildClass extends ParentClass {}

    $cc = new ChildClass();
    if(is_a($cc, 'ChildClass'))
        echo 'Az objektum ChildClass típusú.';
    if(is_a($cc, 'ParentClass'))
        echo 'Az objektum ParentClass típusú is.';
?>
```

2. ábra: is_a() használata

Kimenet:

```
Az objektum ChildClass típusú.
Az objektum ParentClass típusú is.
```

Kivételkezelés

Az egyik legjelentősebb fejlesztés a PHP-ban, hogy a PHP5-ben bevezették a kivételkezelést, lehet használni kivételeket. A `Exception` ősosztályból származtatva saját kivételeket hozhatunk létre.

A PHP is, sok más programozási nyelvhez hasonlóan a `try/catche` szerkezetet használja kivételek kezelésére. A `try` blokk tartalmazza azt a kódot ami a hibát okozhatja. Amikor kivétel váltódik ki azt egy `catch` ág kapja el. Minden `try` blokknak rendelkeznie kell legalább egy megfelelő `catch` blokkal. Lehetőség van többszörös `catch` blokkok használatára, ilyenkor különböző osztályú kivételeket kaphatunk el. A program, ha nem volt kivétel a `try` blokkban, vagy a kivétel nem illeszkedik egyetlen `catch` blokkra sem, az utolsó `catch` blokk után folytatódik. A `catch` ág argumentuma mindig egy objektum. Kivétel kiváltása a `throw` kulcsszó segítségével történik. Ha kivételt dobunk a blokkban a követő kód már nem fut le, a PHP megkeresi az első olyan `catch` blokkot amire illeszkedik a kivétel. Ha a kivételt nem sikerül sikeresen elkapni, a PHP fatális hibát(Fatal Error) ad.

Iterátorok

A PHP4-ben a `foreach` utasítást használtuk ahhoz, hogy egy tömböt bejárjunk. Ez használható volt objektumok esetén is, azok publikus attribútumaik bejárására. A PHP5 lehetőséget ad az `Iterator` interfészek implementálására. Az `Iterator` interfészben lévő metódus specifikációk a következők:

`current()`: az aktuális elemet adja vissza

`key()`: az aktuális elem kulcsát adja vissza

`next()`: a következő elemre lép

`rewind()`: visszaállítja az iterátort az első elemre

`valid()`: ellenőrzi, hogy van e aktuális elem a `rewind()` illetve `next()` hívása után

Objektum klónozása

A PHP5 egy új megközelítést vezet be az objektumok klónozásával kapcsolatosan, ami merőben eltér a PHP4 megközelítésétől. PHP4-ben amikor lemásoltunk egy objektumot egy másikba egy mély másolás(deep copy) ment végbe. Ez azt jelentette, hogy egy teljesen új objektumot hozunk létre amely megtartja a lemásolt objektum attribútumait. Bármely nemű változás az új objektumon, nem érintette az eredetit.

A PHP5 különbözik ettől, olyan értelemben, hogy amikor lemásolunk egy objektumot egy másikba, akkor egy sekélyes másolás történik(shallow copy), a másolat egy referenciát fog tartalmazni az eredetire.

```
<?
    $object2 = clone $object1;
?>
```

3. ábra: A clone kulcsszó használata

Ha ugyan azt a hatást akarjuk elérni mint PHP4-ben, akkor a clone kulcsszót kell használjuk.

Objektum életciklusa

Egy objektum életciklusa addig tart míg az őt példányosító script futása be nem fejeződik. A PHP nem rendelkezik globális, illetve alkalmazás szintű hatáskörrel. Tehát nem tarthatunk meg objektumainkat.

2.1.4. MySQLi

A MySQLi a PHP5-ben került bevezetésre és tovább fejleszti az eddigi MySQL lehetőségeket olyan lehetőségekkel mint a tárolt eljárások használata. Bizonyos szempontból a MySQLi kiterjesztés sokkal jobb és hasznosabb mint a MySQL kiterjesztés volt. A MySQLi teljes objektumorientált hozzáférési lehetőséget biztosít a MySQL adatbázishoz, ami a PHP5 előtt nem volt meg a nyelvben.

Néhány újítás amit a MySQLi bevezetett:

Előkészített kifejezések(Prepared Statements): a MySQL 5.0 változata vezeti be a nagyobb biztonság és rugalmasság miatt. Az előkészített kifejezés egy olyan, a MySQL

szerver által előre lefordított adatbázis művelet amelyet később fel lehet használni. Ezáltal csökken az SQL befecskendezés esélye, valamint jobb teljesítményű a hagyományos adatbázis műveleteknél. Nem igényel plusz fordítói lépést.

Az előkészített kifejezések paramétereket fogadnak futási időben abban a sorrendben, ahogy meghatározzuk, amikor előkészítjük a kifejezésünket.

A MySQL előkészített kifejezése négy típust támogat:

- i – a megfelelő változó integer típusú
- d - a megfelelő változó double típusú
- s – a megfelelő változó string típusú
- b – a megfelelő változó blob típusú

A következőkben bemutatok egy példát a hagyományos és az új megoldásra:

```php
<?
    $mysqli = new mysqli("localhost", "username", "password", "db");

    if(mysqli_connect_errno()) exit();

    $stmt = $mysqli->prepare("SELECT name, password FROM users ORDER BY name");
    $stmt->execute();
    $stmt->bind_result($name, $password);

    while($stmt->fetch()) echo $name . "<br />";
?>
```

4. ábra: MySQLi PHP5 előtti használata

```php
<?
    $mysqli = new mysqli("localhost", "username", "password", "db");

    if(mysqli_connect_errno()) exit();

    $stmt = $mysqli->prepare("SELECT name, password FROM users WHERE name = ?");

    $stmt = bind_param("s", $name);
    $name = "béla";
    $stmt->execute();
    $name = null;

    $stmt->bind_result($name, $password);

    while($stmt->fetch()) echo $name . "<br />";
?>
```

5. ábra: MySQLi PHP5 utáni használata

2.1.5. PDO(PHP Data Objects)

Szintén egy új kiterjesztés, amit a PHP5.1-hez adtak hozzá. Segítségével könnyebben karban tudjuk tartani a különböző típusú adatbázisainkat. Több meghajtót(driver) tartalmaz különböző adatbázis motorokhoz. A PDO úgy viselkedik mint egy adat kapcsolati réteg, hogy ugyan azokat a függvény neveket használhassuk az összes adatbázis motor esetén.

A különböző adatbázisokhoz való kapcsolódás a DNS(Data Source Name) karakterlánc segítségével lehetséges.

Néhány példa DNS-re:

- `mssql:host=localhost;dbname=testdb`
- `mysql:host=localhost;port=3307;dbname=testdb`
- `pgsql:dbname:testdb;user:username;password:password;`
  `host:localhost;port:5432`

A következőkben bemutatok egy példát kapcsolódásra, illetve előkészített kifejezések használatára PDO segítségével:

```php
<?
    $dns = 'mysql:dbname=testdb;host=localhost';
    $username = 'username';
    $password = 'password';

    try{
        $pdo = new PDO($dns, $username, $password);
    } catch(PDOException $e) {
        echo 'Connection failed: ' . $e->getMessage();
    }
?>
```

6. ábra: Kapcsolódás MySQL-hez PDO segítségével

A fentebb létrehozott kapcsolatot felhasználva:

```php
<?
    $stmt = $pdo->prepare('SELECT id FROM users WHERE name=:name');
    $name = 'béla';
    $stmt->bindParam(':name', $name, PDO::PARAM_STR);
    $stmt->execute();

    $stmt->bindColumn('id', $id);
    $stmt->fetch();

    echo $id;
?>
```

7. ábra: Előkészített kifejezés PDO segítségével

2.2. A Smarty sablonkezelő rendszer

2.2.1. Mi a Smarty?

A Smarty egy sablonkezelő motor PHP-hez. Segítségével könnyedén szétválaszthatjuk az üzleti logikát és a tartalmat a megjelenítéstől. A Smarty ettől függetlenül tartalmazhat logikát, amennyiben az a megjelenítésre vonatkozik: egyéb sablonok betöltése, tömbök bejárása és a tartalom megjelenítése.

A Smarty egyik fő aspektusa, hogy lefordítja a sablonokat. Ez azt jelenti, hogy beolvassa a sablon állományt és PHP scriptet készít belőle. Ezt követően ezek kerülnek végrehajtásra.

A Smarty néhány jellemzője:

- rendkívül gyors
- kellőképpen hatékony, mivel PHP végzi az elemzést
- csak azok a sablon állományok kerülnek újra fordításra amelyekben változás történt
- a sablonkezelő nyelv kellőképpen rugalmas, hogy saját függvényeket hozzunk létre
- beépített cachelési támogatás

A Smarty lehetőséget biztosít a programozónak és a designer-nek, hogy sokkal hatékonyabban együtt működjenek, ne kelljenek foglalkozzanak egymás munkájával. A designer megtervezi a web oldalt, alkalmazás kinézetét, ehhez megalkotja a szükséges sablonokat, majd kinyeri a programozó által létrehozott PHP állományokból a szükséges adatokat. A programozó átadja a szükséges adatokat a sablonnak, anélkül, hogy HTML kóddal foglalkozna.

2.2.2. Biztonságos e a Smarty?

Alapvetően a Smarty annyira biztonságos mintha egyedül PHP-t használnánk. Ez azt jelenti, hogy Smarty használata nem csökkenti a biztonságot, de vannak esetek amikor a Smarty tökéletesítheti azt.

Amikor egy olyan web alkalmazással állunk szemben ahol a programozó nem használ sablonokat, a desiger-nek hozzáférése van a PHP állományokhoz, módosíthatja azokat ha

rendelkezik a kellő szakértelemmel. A Smarty rendelkezik néhány beépített biztonsági lehetőséggel olyan esetekre amikor illetéktelenek akarják módosítani a sablonokat. Be lehet állítani, hogy használhatunk e PHP kódot a sablonokban vagy sem. Korlátozhatjuk a könyvtárakat ahonnan sablonokat lehet betölteni. Korlátozhatjuk azon állományokat amelyeket a sablon betölthet.

2.2.3. A Smarty szintaktikája

Minden Smarty teget a { és } jel határol. Ez az alapértelmezett, de meg lehet változtatni.

Megjegyzések

Megjegyzést sablonban a {* és *} jelek között helyezhetünk el. Ez nem kerül megjelenítésre a végleges kimeneti állományban mint a HTML megjegyzések.

Változók

A Smarty változói a $ jellel kezdődnek, ezt követően előfordulhat bennük karakter, számjegy, aláhúzásjel. Minden változót a { és } jel kell közre zárjon. Hivatkozhatunk tömbökre, normál illetve asszociatív értelemben, objektum attribútumaira illetve metódusara.

PHP-ból Smarty változóhoz értéket a következő képen rendelhetünk:

```
<?
    $smarty = new Smarty();

    $smarty->assign('name', 'Zivat Arnold');

    $smarty->display('index.tpl');
?>
```

Ahol az `index.tpl` tartalma:

```
Hello {$name} !
```

Asszociatív tömb esetén ez a következő képen néz ki:

```php
<?
    $smarty = new Smarty();
    $smarty->assign('contacts',
        array(
            'fax'   => '111-111-1111',
            'email' => 'za@contactme.com',
            'phone' => array(
                            'home' => '222-222-2222',
                            'cell' => '333-333-3333'
                        )
        )
    );
    $smarty->display('index.tpl');

?>
```

Az `index.tpl` tartalma:

```
Fax:        {$contacts.fax} <br />
Email:      {$contacts.email} <br />
Phone home: {$contacts.phone.home} <br />
Phone cell: {$contacts.phone.cell} <br />
```

A Smarty lehetőséget biztosít változók betöltésére konfigurációs állományból. Ekkor a változó nevét a kettős kereszt(#) karakter kell közre zárja.

Utasítások

Feltételes utasítás:

```
{if feltétel}utasítások
      [{elseif feltétel} utasítások |
      {else} utasítások]
{/if}
```

Ciklus utasítás: a Smarty két -féle ciklus utasításra használható teget ismer: a `{section}` és a `{foreach}`. Mindkettő más helyzetekben hasznos.

A `{section}`-t használhatja a programozó amikor egy tömböt nem asszociatív tömbként kell kezeljen.

A `{section}`számára a következő paramétereket adhatjuk meg:

- `name`: tetszőlegese alfabetikus érték lehet, és minden iteráció alkalmával nő. Megadása kötelező.
- `loop`: annak a tömb változónak a neve amelyet a `{section}` bejár. A ciklus lépésszámát a tömb elemeinek száma határozza meg, vagy befolyásolhatjuk explicit módon is. Megadása kötelező.
- `star`: ami meghatározza a kezdeti számláló értéket
- `step`: a növekmény
- `max`: a maximális érték, ahányszor a ciklus lefut
- `show`: logikai érték, amely meghatározza, hogy a `{section}{/section}` blokk kerüljön e megjelenítésre vagy sem

A következőkben bemutatok egy példát `{section}` használatára:

```
{section name=customer loop=$contacts}
    <ul>
        <li>name:   {$contacts[customer].name} </li>
        <li>home:   {$contacts[customer].home} </li>
        <li>cell:   {$contacts[customer].cell} </li>
        <li>e-mail: {$contacts[customer].email} </li>
    </ul>
{/section}
```

A `{foreach}` nagyobb kényelmet biztosít mint a `{section}` a tömb elemeihez való hozzáférés tekintetében. A tömb minden elemtét úgy kezeli mint egy különálló tömb, amely kulcs-érték párokat tartalmaz, tehát asszociatív tömbként kezelhetjük a tömbünket. Ez könnyebb olvashatóságot ad a kódnak és megkönnyíti mások munkáját ha később fel akarják majd ezt használni.

A `{foreach}` számára a következő paramétereket adhatjuk meg:

- `from`: a tömb amit be akarunk járni. Megadása kötelező.
- `item`: az aktuális elemnek a neve. Megadása kötelező.
- `key`: az aktuális kulcsnak a neve.

- `name`: a ciklus neve. Megadása akkor szükséges ha a ciklus tulajdonságaihoz akarunk hozzáférni.

  Ez a következő képen történik: `{$smarty.foreach.name.property}`

  Ezek a tulajdonságok a következők lehetnek: `index, itration, first, last,show, total.`

A következőkben bemutatok egy példát `{forach}` használatára:

```
<ul>
{foreach from=$myArray item=element key=_key}
    <li>{$_key} : {$element}</li>
{/foreach}
</ul>
```

3. Fejlesztés

3.1. Az alkalmazás fejlesztés során elkészített UML diagramok

3.1.1. Használati eset diagram

Ezzel a diagrammal összegyűjthetjük az alkalmazással szemben támasztott követelményeket, továbbá tisztázhatjuk a felhasználó és a rendszer kapcsolódási pontjait, valamit az egyes felhasználói szerepköröket.

A használati eset diagramon a felhasználókat és egyéb a rendszerhez kapcsolódó külső „elemeket” aktoroknak nevezzük. A kifejlesztendő alkalmazás esetén első lépés lehet az aktorok megválasztása. Jelen esetben a Látogatót, a Tanárt, az Adminisztrátort, a Hallgatót és az Adatbázist határoztam meg.

A Látogató egy általános szerepkör, amelynek a Tanár illetve az Adminisztrátor a pontosítása, ők a rendszer nem nyilvános részéhez is hozzáférhetnek. Hallgató alatt értem azokat akik az alkalmazás publikus részét használják.

A Tanárnak lehetősége van regisztrálni, belépni a rendszerbe, módosítani jelszavát, létrehozni, szerkeszteni, illetve törölni a profilját, jegyzetet feltölteni, a rendszerben szereplő jegyzetet módosítani, illetve törölni.

Az Adminisztrátornak lehetősége van belépni a rendszerbe, módosítani jelszavát, felhasználót törölni, illetve karokat, tanszékeket felvinni, törölni.

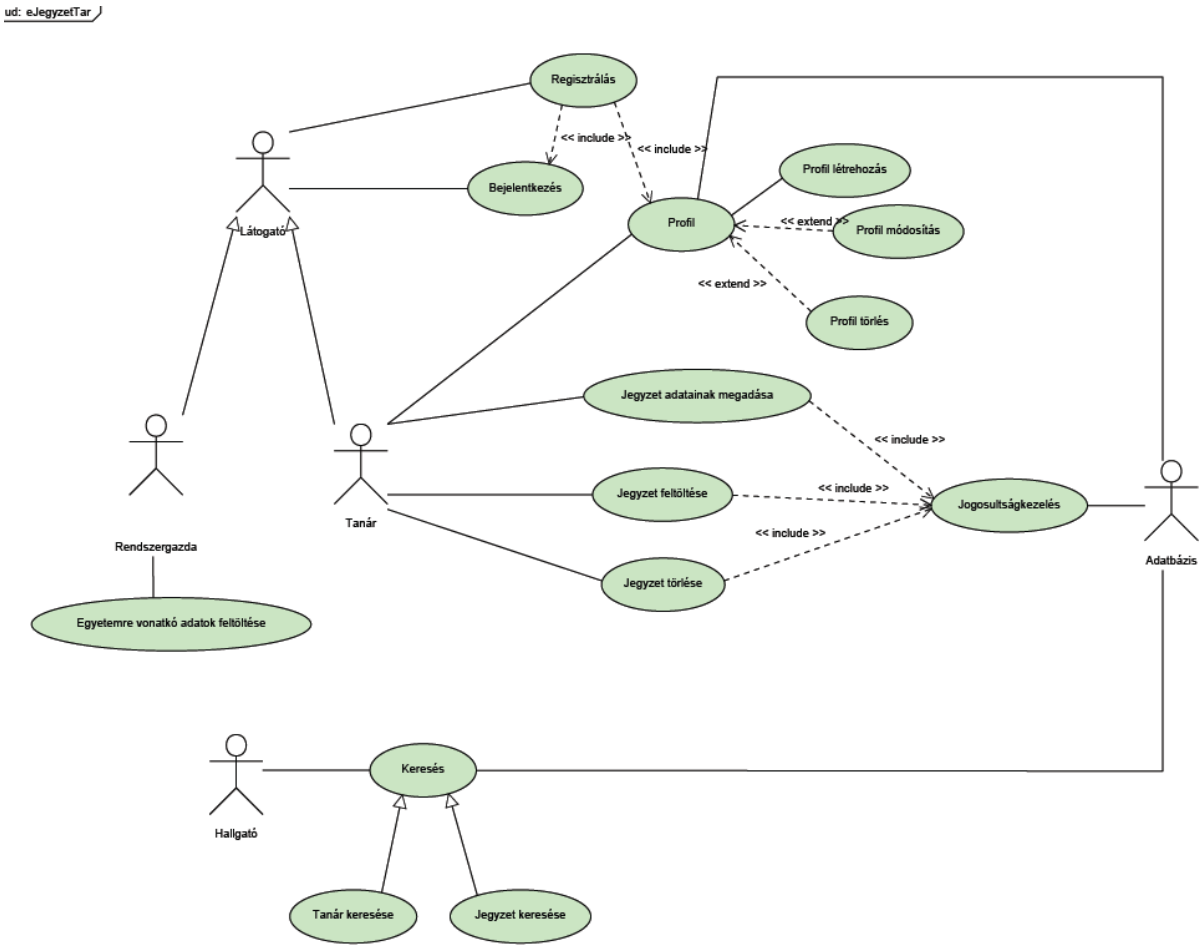
A Hallgatónak lehetősége van keresni jegyzetet illetve tanárt, megtekinteni a tanár adatlapját, letölteni jegyzetet.

A diagramon lévő ellipszisek a használati esetek, amelyek a rendszer funkcióit, külső kapcsolódási pontjait írják le.

A használati eset diagram segítségével egyértelművé válnak a rendszer határai, a rendszert használók szerepkörei és az általuk elérhető funkciók. A programozó számára pedig egyértelmű, hogy milyen funkciókat kell megvalósítani.

A használati eset diagram tekinthető az alkalmazással szemben támasztott követelmények térképének, a funkciók grafikus megjelenítésének.

A bemutatandó rendszer használati eset diagramja:



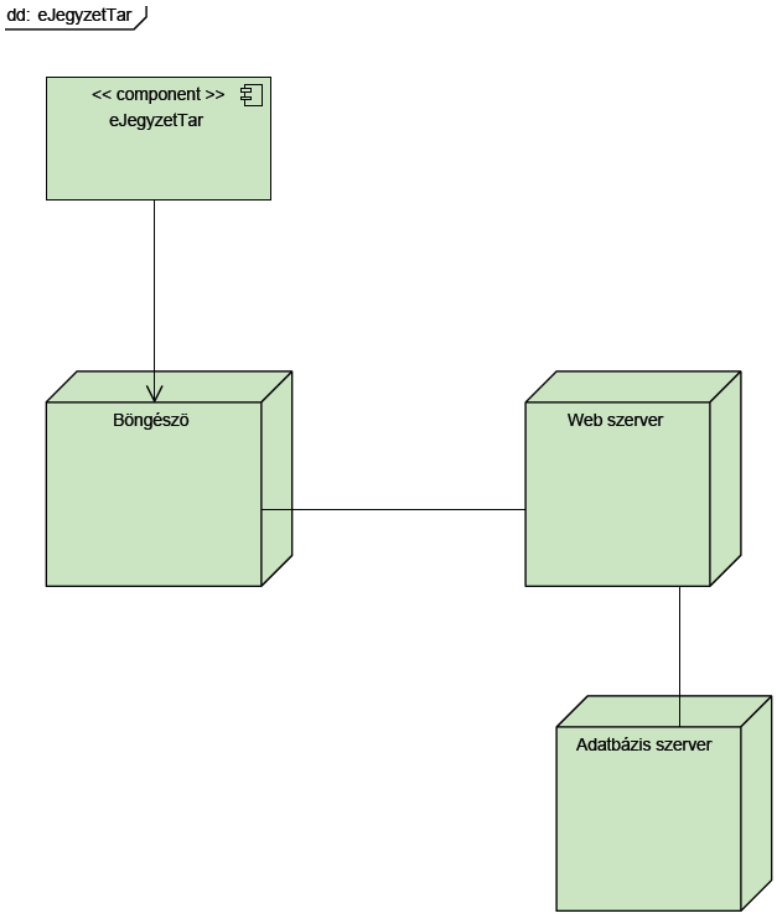
8. ábra: Használat eset diagram

3.1.2. Alkalmazási-diagram

Az alkalmazás-diagrammal a rendszer elemeit és a közöttük lévő fizikai kapcsolatot adhatjuk meg. Az alkalmazás-diagram alapeleme a csomópont, amelyet az UML kocka alakzattal ábrázol. Ez egy számítógépes egységet, legtöbb esetben egy hardver elemet jelent.

Az én esetemben ez három elemet jelent: egy böngészőt, ahol az alkalmazás megjelenik, egy szervert, ahol az alkalmazás fut és egy adatbázist szervert.

A bemutatandó rendszer alkalmazási-diagramja:



9. ábra: Alkalmazási-diagram

3.2. A fejlesztés során elkészített szöveges dokumentumok

3.2.1. A rendszer funkciói

- **Bejelentkezés**: a rendszerbe való belépés. Az alkalmazáshoz különböző felhasználók férhetnek hozzá, szerepkörüknek megfelelően a rendszer különböző részeihez férhetnek hozzá.
- **Kijelentkezés**: a rendszerből való kilépés.

- **Tanár** felhasználónak a következő funkciók állnak rendelkezésére:
  - **Regisztrálás**: új felhasználó felvétele a rendszerbe.
  - **Jegyzet feltöltése**: a Tanárnak lehetősége van jegyzetet felvinni a rendszerbe.
  - **Jegyzet törlése**: a Tanárnak lehetősége van az korábban feltöltött jegyzetek közül tetszőleges számút eltávolítani a rendszerből.
  - **Jegyzet módosítása**: a rendszerben szereplő jegyzetek adatait a Tanár tetszése szerint megváltoztathatja.
  - **Profil létrehozása**: a Tanárnak lehetősége van személyes adatlapjának kitöltésére, amelyen munkahelyi elérhetőségeit adhatja meg.
  - **Profil törlése**: a Tanár törölheti adatlapját a rendszerből.
  - **Profil módosítása**: az adatlapon megadásra kerülő adatokat a Tanár szükség szerint megváltoztathatja.
  - **Jelszó módosítása**: a Tanárnak lehetősége van megváltoztatni jelszavát.
- **Adminisztrátor** felhasználónak a következő funkciók állnak rendelkezésére:
  - **Kar felvitele**: a rendszer működésbe helyezése előtt az Adminisztrátornak fel kell vinni az adott egyetemen szereplő karokat.
  - **Kar törlése**: az Adminisztrátornak lehetősége van adott kar eltávolítására a rendszerből.

- **Tanszék felvitele**: a rendszer működésbe helyezése előtt az Adminisztrátornak fel kell vinni az egyes karokon belül szereplő tanszékeket.
- **Tanszék törlése**: az Adminisztrátornak lehetősége van adott tanszék törlésére.

- **Hallgató** felhasználónak a következő funkciók állnak rendelkezésére:
  - **Tanszék jegyzeteinek listázása**: a Hallgató megtekintheti egy adott tanszéken belül szereplő összes jegyzetet.
  - **Jegyzet keresése**: a Hallgató kereshet jegyzetet annak címe, kulcsszavak illetve szerző alapján.
  - **Jegyzet letöltése**: a Hallgató letöltheti a kiválasztott jegyzetet.
  - **Tanár keresése**: a Hallgató kereshet Tanárt név alapján.
  - **Tanár profiljának megtekintése**: a Hallgató megtekintheti egy Tanár adatlapját.
  - **Tanár jegyzeteinek listázása**: a Hallgató megtekintheti egy adott Tanár által a rendszerbe feltöltött jegyzetek listáját.

3.2.2. Forgatókönyvek

A követelmények pontosításának egyik módszere, az egyes használati eseteket megpróbáljuk részlettesen leírni. Egy forgatókönyvben meg kell adni, hogy az adott funkció milyen párbeszédet igényel az aktor és az alkalmazás között. A fejlesztett rendszerrel kapcsolatos használati eset közül néhányat részletesebben bemutatok. Ezek a használati esetek a Tanár aktorhoz kapcsolódnak

- **Regisztrálás a rendszerbe**:
  - Elindítja az alkalmazást
  - A megjelenő oldalon kiválasztja a tanszéket és a kart amelynek tagja. Megadja az email címét és jelszavát
    - A regisztráció sikeres volt
      - Beléphet a rendszerbe

        - A regisztráció sikertelen volt
            - Próbálkozhat ismét
            - Kilép az alkalmazásból
- **Bejelentkezés a rendszerbe**:
    - Elindítja az alkalmazást
    - A megjelenő oldalon megadja email címét és jelszavát
        - A bejelentkezés sikeres volt
        - A bejelentkezés sikertelen volt
            - Próbálkozhat ismét
            - Kilép az alkalmazásból

Ezek a funkciók csak bejelentkezés után érhetők el.

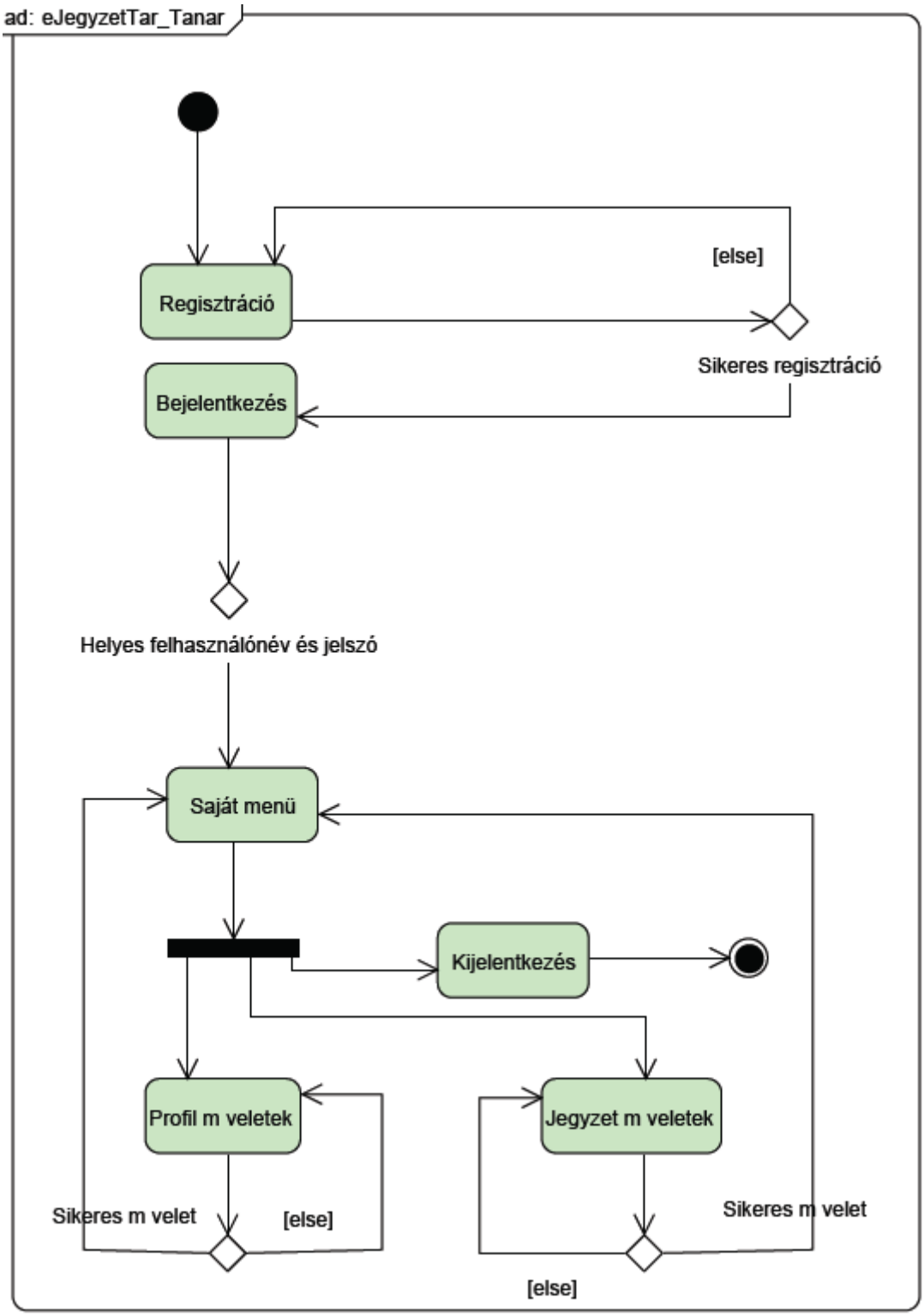
- **Jegyzet feltöltése**:
    - Kiválasztja a ***Jegyzetek*** menüpontot
        - Kiválasztja az ***Új jegyzet feltöltése*** menüpontot
            - Megadja a jegyzet címét
            - Az esetleges társ szerzőket
            - A kiadás dátumát
            - A jegyzet rövid leírását
            - Csatolja a jegyzetet
            - Megnyomja a ***Feltölt*** gombot
- **Profil módosítása**:
    - Megtekinti adatlapját
    - Kiválasztja a módosítani kívánt adatot
    - Megnyomja az adat mezője mellett szereplő ikont

- Elvégzi a kívánt módosítást
- Megnyomja a ***Mentés*** gombot

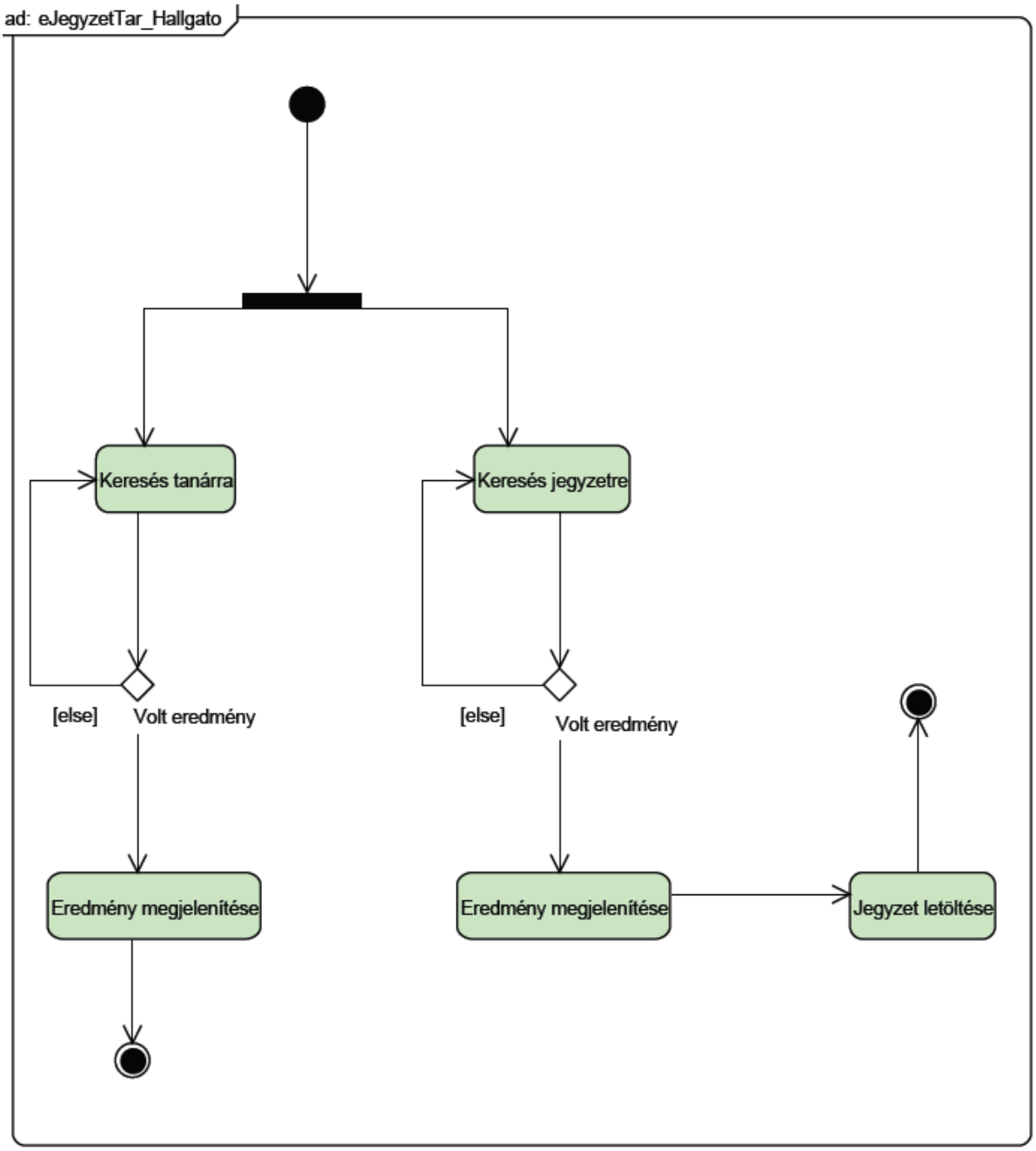
3.2.3. Aktivitás diagram

A használati eset diagrammal megadhatjuk, hogy mit várunk el a rendszertől, az aktivitás diagrammal pedig megfogalmazhatjuk, hogy a rendszer hogyan fogja elérni ezeket a célokat. Ezzel a diagram típussal az alkalmazás dinamikáját, időben lezajló változását a végrehajtandó tevékenységek sorrendjének meghatározásával mutathatjuk be.

Az aktivitás-diagram alapeleme az aktivitás, vagy tevékenység. Ez egy olyan feladat amit végre kell hajtani. Jelölése az ívelt oldalú téglalap. A végrehajtandó tevékenységek időbeli sorrendjét a nyílhegyben végződő vonallal, az átmenettel jelöljük. A nyíl rendre az aktivitás befejeződés utáni következő végrehajtandó tevékenységre mutat. Megadhatunk alternatív tevékenységcsoportot, a választást a döntés szimbólumával egy rombusszal jelöljük. A diagramon két, úgynevezett pszeudo-állapotot is feltüntethetünk: a fekete körrel jelölt start, vagy kezdő állapotot, és a stop, vagy vég állapotot. A start állapot jelöli minden tevékenység kezdetét, a stop állapot pedig minden tevékenység befejeződését.



10. ábra: Tanár aktivitás-diagramja

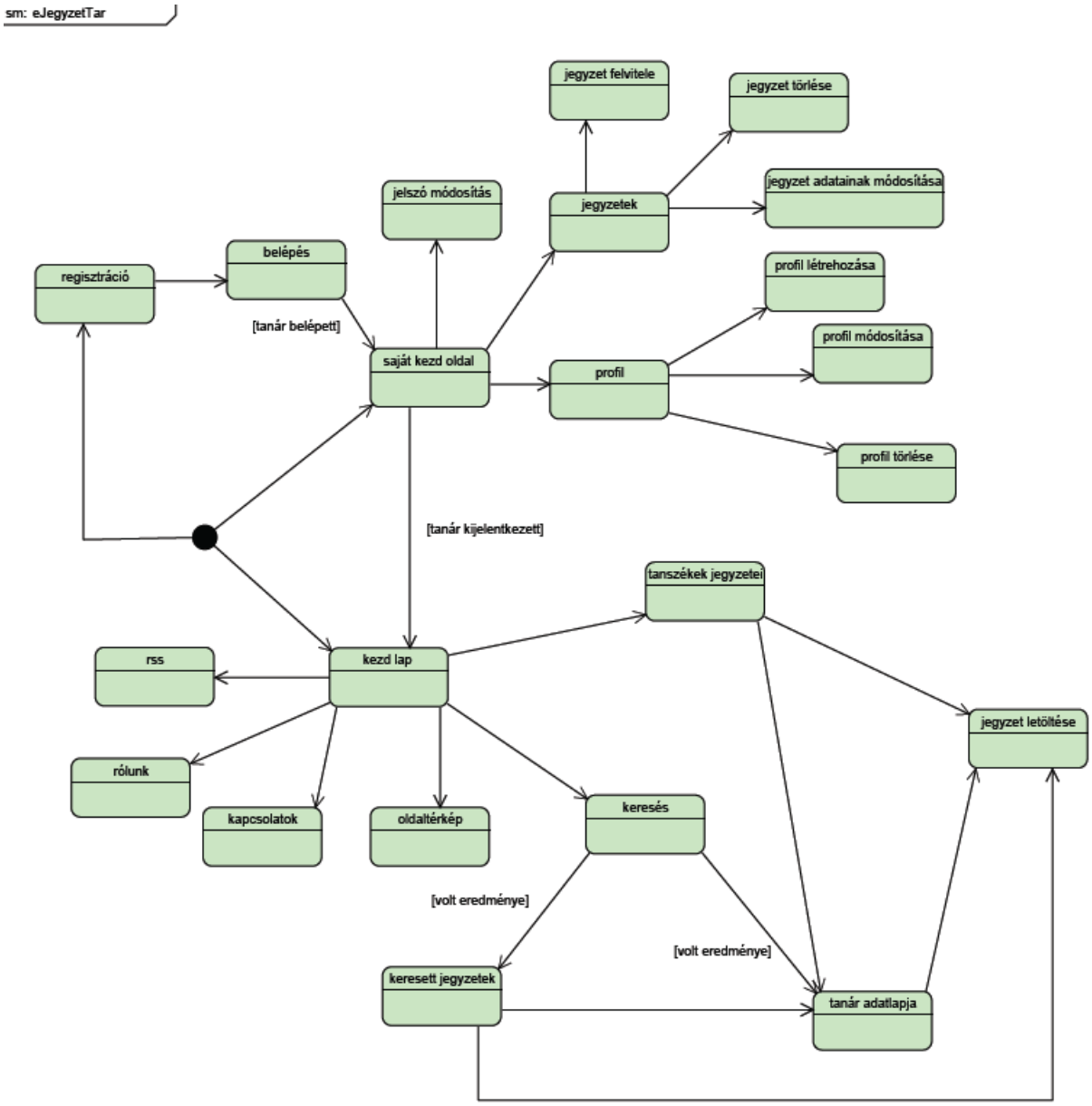


11. ábra: Látogató aktivitás-diagramja

3.2.4. Felhasználói felületek

- **Navigáció tervezése**: a tervezés ezen szakasza megválaszolja egyrészt, hogy a felhasználók az egyes használati esetekben a rendszer milyen képét látják, másrészt az egyes használati esetek közötti mozgási lehetőségeket mutatják be.

  Az én esetemben ez a következő:



12. ábra: Navigációs diagram

- **Az egyes felhasználói felületek**:

  A felhasználói felületet tervezésekor néhány fontos szempontot figyelembe kell venni:

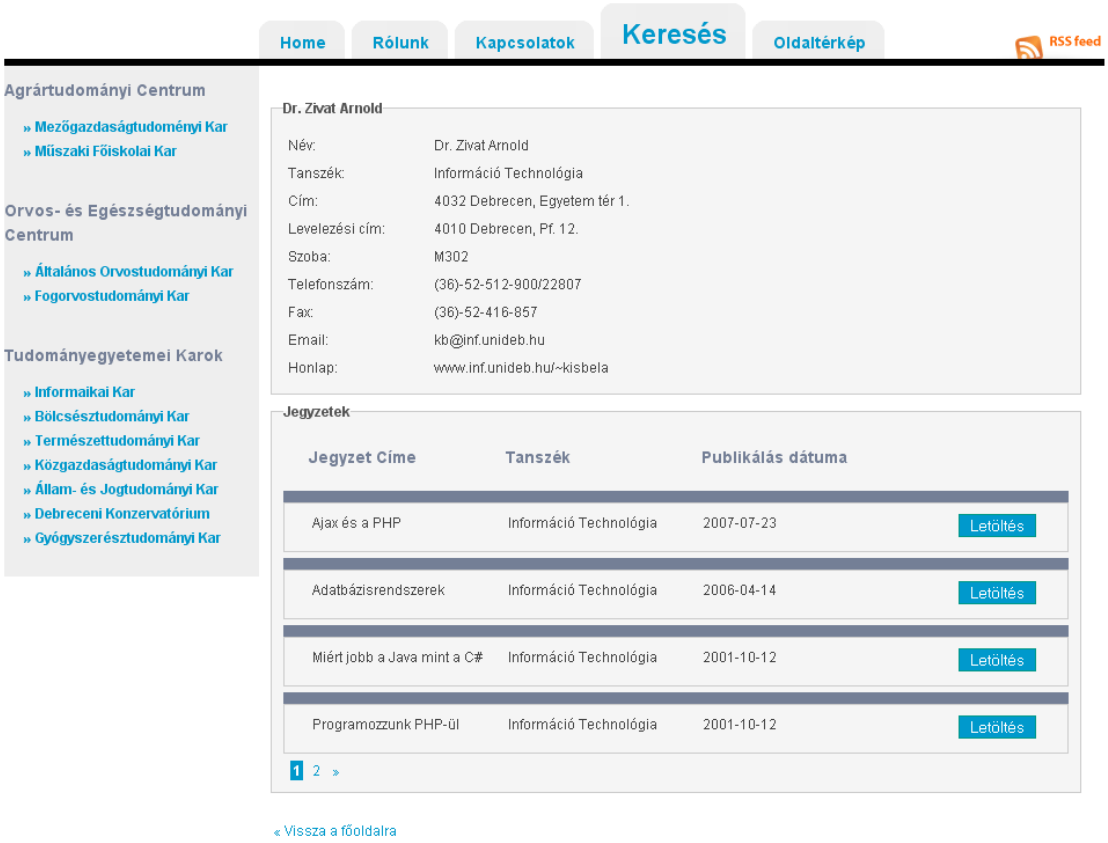
  - a megtervezett felhasználói felület illeszkedjen a feltételezett felhasználói kör tudásához, elvárásaihoz, tapasztalataihoz
  - a rendszert használók a rendszert sokszor a felhasználói felületéről ítélik meg és nem annak funkcionalitásáról
  - a rosszul megtervezett felhasználói felületek miatt a rendszert használók hibákat véthetnek
  - a felületek konzisztens módon hasonló operációkat ugyan úgy végezzenek. Az utasítások, menük legyenek azonos kinézetűek
  - a színek száma legyen limitált, használjuk visszafogottan. A színek változása jelentheti a rendszer állapotának változását. A színek párosítása óvatosan történjen
  - emberi tényezők:
    - az ember általában 7 információs egységet tud fejben tartani, ha ennél többet ajánlunk fel akkor nőhet a hiba ejtés lehetősége
    - használjunk megfelelő, felhasználó barát hibaüzeneteket, legyen kontextus-függő segítség nyújtási szolgáltatás(help). A hibaüzenet legyen udvarias, tömör, konzisztens. A felhasználó háttérismerete legyen a meghatározó tényező a tervezéskor.

A bemutatandó alkalmazás felhasználói felületei megpróbálják ezeket az elveket követni.

Publikus felületek:

13. ábra: Kar tanszékeinek listázása

14. ábra: Tanszék jegyzeteinek listázása



15. ábra: Tanár adatai és jegyzetei

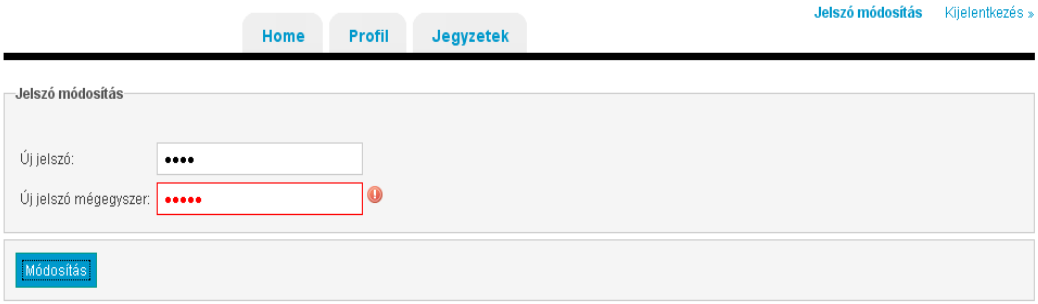
16. ábra: Keresés

A Tanár felhasználói felületei:

17. ábra: Regisztráció és regisztrációs hiba

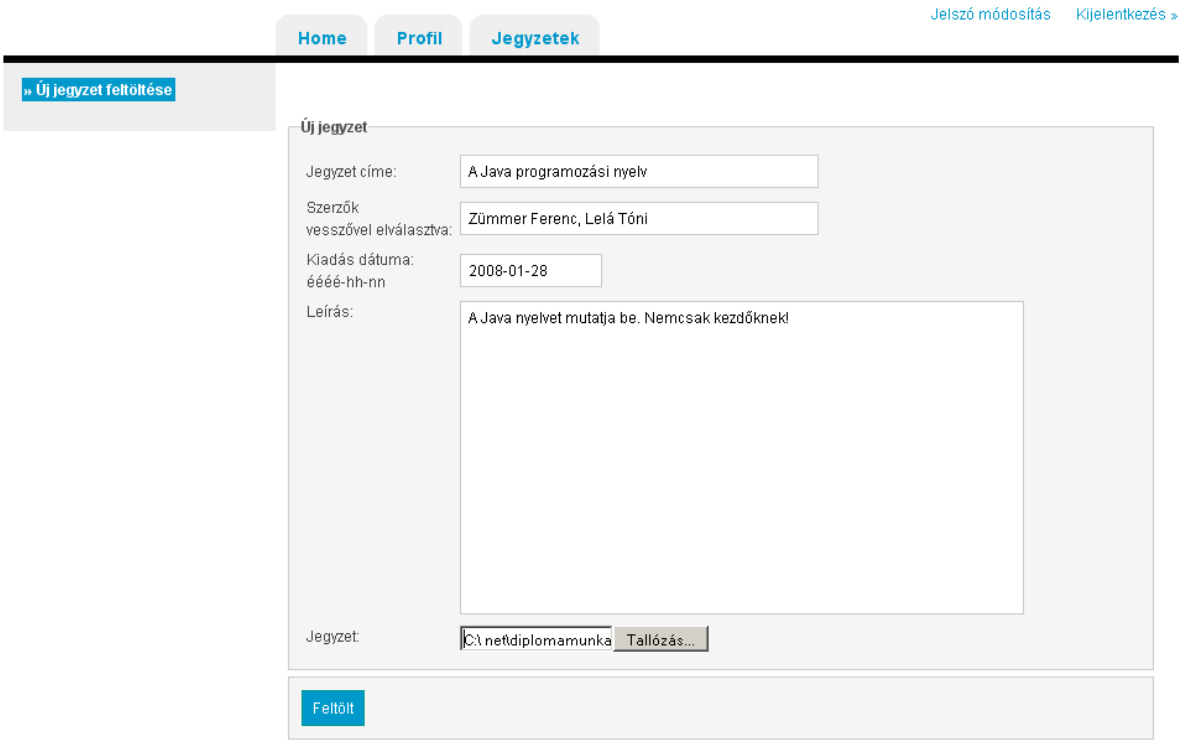
18. ábra: Bejelentkezés, hibás email cím

19. ábra: Sikertelen bejelentkezés

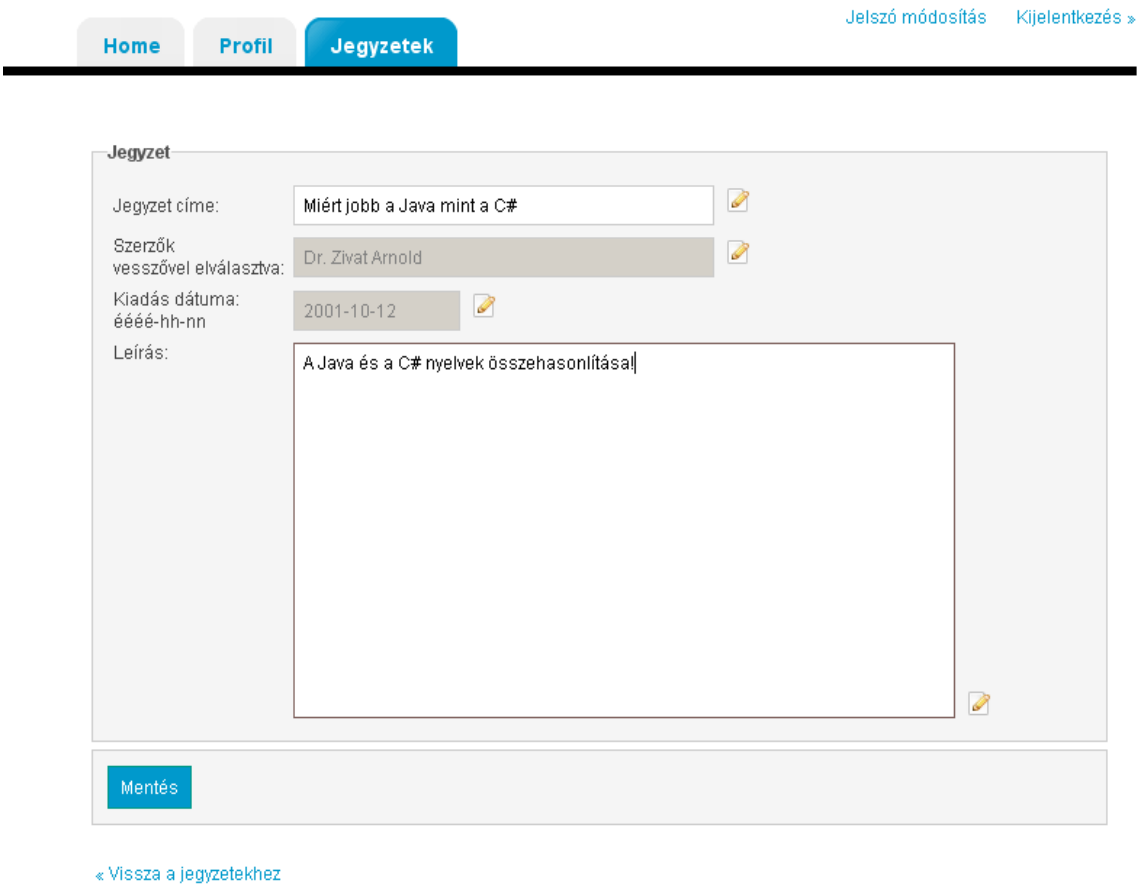


20. ábra: Jelszó módosítás, hiba

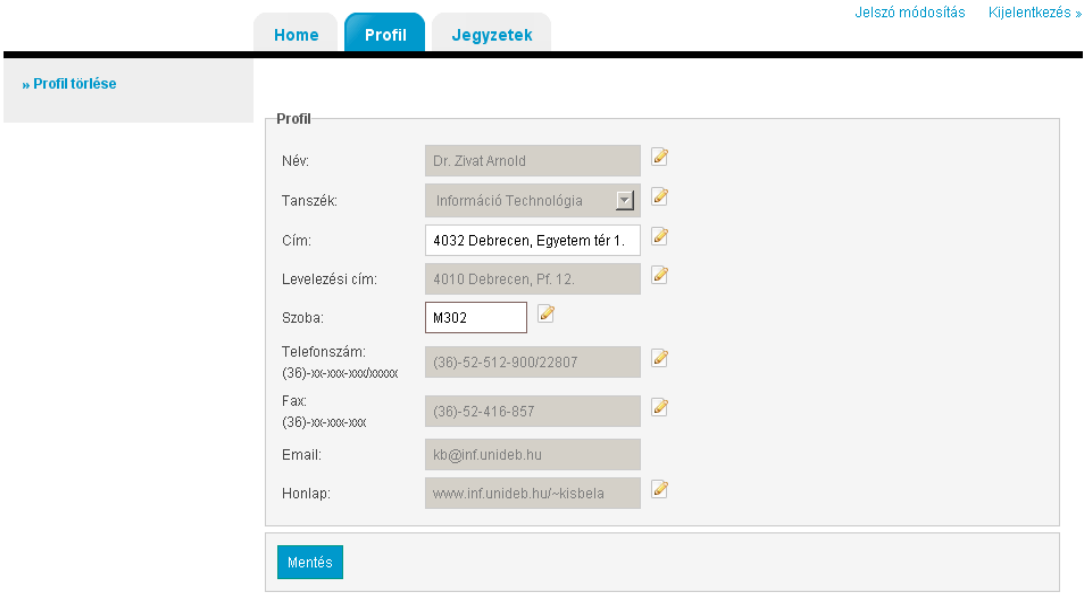
21. ábra: Saját jegyzetek listázása



22. ábra: Jegyzet feltöltése



23. ábra: Jegyzet módosítása



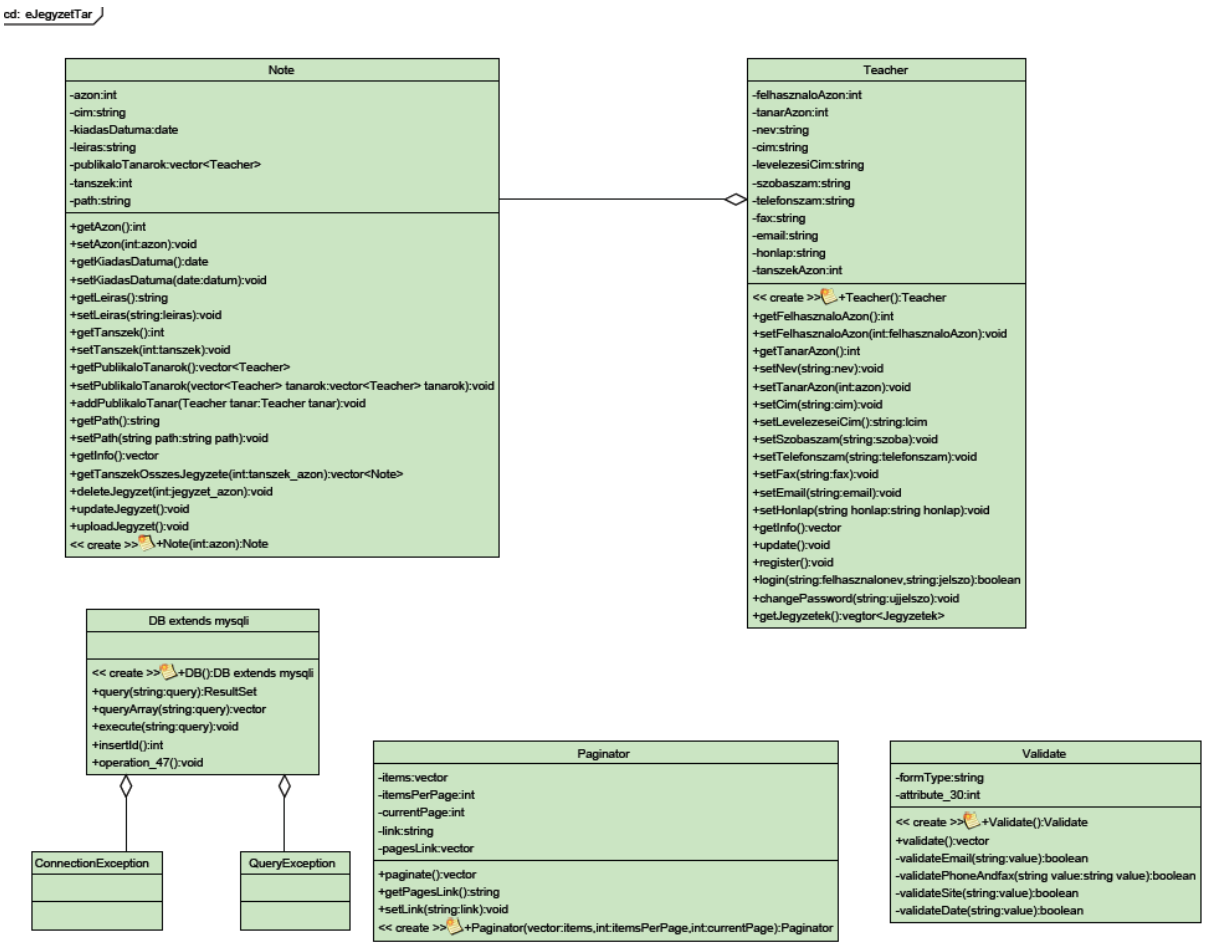
24. ábra: Profil módosítása, törlése

4. Az elkészült alkalmazás leírása

4.1. Az alkalmazás leírása a PHP oldaláról

A következőkben az alkalmazás fejlesztése során létrehozott osztályokat illetve azok funkcionalitását mutatom be, valamit az alkalmazás szempontjából lényeges állományokat.

4.1.1. Osztály diagram



25. ábra: Osztály diagram

Az alkalmazás fejlesztés során megpróbáltam külön választani mind a felhasználói felületet az üzleti logikától, mind pedig az egyes felhasználók által elérhető részeket.

Mivel az alkalmazás interneten működő, a publikus rész bárki számára elérhető.

A Tanár illetve az Adminisztrátor felhasználók az alkalmazást úgy érhetik el, hogy a böngésző címsorában az alkalmazás URL címe után a „`/tanar/`” illetve „`/admin/`” útvonal kiegészítést megadják. Az egyik ok amiért ezt a megoldást választottam a biztonság. Az alkalmazás nyilvános részéről nem szerepel közvetlen link az alkalmazás ezen részeihez. A másik ok pedig, hogy az alkalmazás szerkezete illetve a forrás állományok elhelyezése jobban strukturált és sokkal áttekinthetőbb.

4.1.2. A `Teacher` osztály

A `teacher` adatbázistáblának megfelelő osztály. Tartalmazza az adatbázistábla mezőinek megfelelő attribútumokat, ezen attribútumok lekérdező és beállító metódusait, illetve számos olyan metódust amelyek a Tanár felhasználó által végezhető tevékenységeket reprezentálják.

Az attribútumok

- `felhasznaloAzon`: a Tanárnak mint felhasználónak az azonosítója.
- `tanszekAzon`: annak a tanszéknek az azonosítója amelynek tagja a tanár.
- `tanarAzon`: a tanár azonosítója.
- `nev`: a Tanár nevét reprezentáló karakterlánc.
- `cim`: a Tanár címét reprezentáló karakterlánc.
- `levelezesiCim`:a Tanár levelezési címét reprezentáló karakterlánc.
- `szobaszam`: a Tanár szobaszámát reprezentáló karakterlánc.
- `telefonszam`: a Tanár telefonszámát reprezentáló karakterlánc.
- `fax`: a Tanár fax számát reprezentáló karakterlánc.
- `email`: a Tanár email címét reprezentáló karakterlánc.

- `honlap`: a Tanár honlapját reprezentáló karakterlánc.

A konstruktor

Az osztály egyetlen üres konstruktort tartalmaz.

4.1.3. A `Note` osztály

A `note` adatbázistáblának megfelelő osztály. Tartalmazza az adatbázis táblában szereplő mezőknek megfelelő attribútumokat, ezek lekérdező illetve beállító metódusait, illetve számost technikai jellegű metódust.

Attribútumok

- `azon`: a jegyzet azonosítója.
- `cim`: a címét reprezentáló karakterlánc.
- `kiadasDatuma`: a jegyzet kiadási dátumát reprezentáló dátum.
- `leiras`: a jegyzettel kapcsolatos fontosabb információkat reprezentáló szöveg.
- `publikaloTanarok`: `Teacher` objektumok tömbje akik a jegyzet szerzői reprezentálják.
- `tanszek`: annak a tanszéknek az azonosítója, amelynek a publikáló tanár, tanárok tagja, tagjai.
- `path`: a jegyzet letöltéséhez szükséges elérési útvonal.

A konstruktor

Ha az osztály konstruktorának átadunk egy egész értéket, akkor az egész értéknek megfelelő azonosítójú, már a rendszerben szereplő `Note` példányt hozhatunk létre. Az ily módon létrehozott példány attribútumai az adatbázisból kinyert mezőértékekre állítódnak be.

4.1.4. A `DB` osztály

Ahogy a neve is mutatja ez az osztály felelős a MySQL adatbázissal való kapcsolatért, valamit a szükséges adatbázis műveletek: lekérdezés, módosítás, törlés, beszúrás

végrehajtásáért. Az osztályt a `mysqli` osztályból származtatom.

Itt kerül definiálásra a két saját kivétel osztály is: a `ConnectionException`, valamint a `QueryException`, amelyeket az `Exception` osztályból származtatok.

Az osztályban nem találhatóak attribútumok.

A konstruktor

Az osztály konstruktorának adjuk meg az adatbázis kapcsolathoz szükséges paramétereket: az adatbázis nevét, a felhasználó nevet, a jelszót, a számítógép nevét amelyen a MySQL adatbázis szerver fut. Sikertelen kapcsolódás esetén `ConnectionExceptioin` kivételt dobunk.

4.1.5. A `config.php` állomány

Ez az állomány tartalmazza az alkalmazás fejlesztése során használt konstansok és globális változók definícióját illetve deklarációját.

Itt hozzuk létre a MySQL adatbázissal a kapcsolatot, illetve példányosítjuk a `DB` osztályt. Ezt a példányt használom az alkalmazás fejlesztése során az adatbázis műveletek elvégzésére.

Szintén itt kerül példányosításra a `Note` osztály is, valamit a felhasználói felület megjelenítéséért felelős `Smarty` osztály(`$smarty`). Itt állítjuk be a Smarty osztály számára szükséges paramétereket: az alkalmazás fejlesztéséhez felhasznált `template`-ek(sablonok) elérési útvonalát, valamint a Smarty által használt `cache` könyvtár elérési útvonalát.

Ebben az állományban történik a működés szempontjából alapvető, és a fejlesztés során felhasznált osztályok illetve függvények betöltése is.

4.1.6. Az `index.php` állomány

Ez az állomány végzi az alkalmazás irányítását.

Itt kerül betöltésre a `config.php` állomány.

Az URL-ből kiolvasott értéknek megfelelően kerülnek betöltésre a php állományok. Ha az URL-ből kiolvasott érték hibás vagy nem létezik ilyen oldal, akkor ennek megfelelő

hibaüzenettel térünk vissza.

Itt történnek meg a `$smarty` változóhoz való statikus hozzárendelések, mint például a baloldali menü, felső menü, illetve az oldal címe.

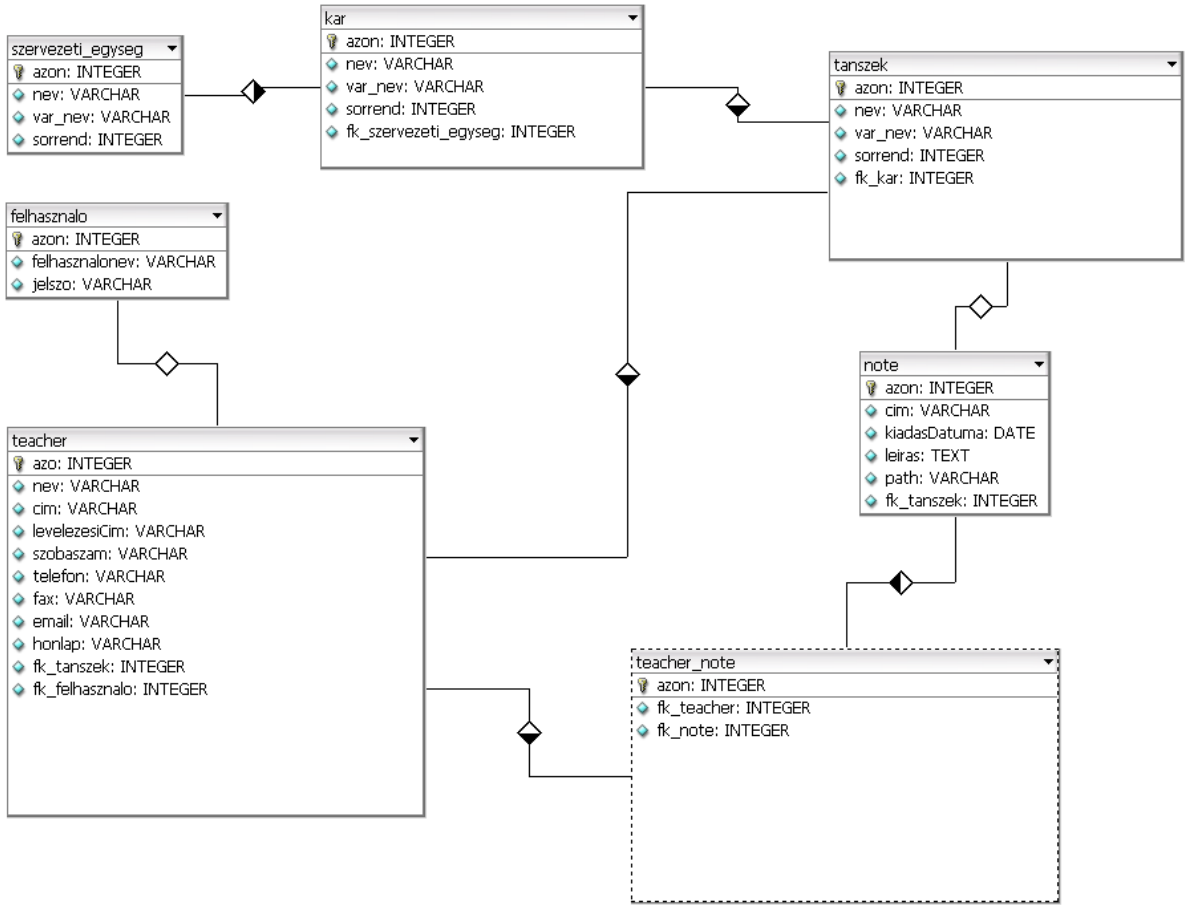
Ha a betöltött php állományokban az aktuális oldalt reprezentáló `$template` változó értéket kapott, akkor ezt megjelenítjük a `$smarty->display($template);` utasítás segítségével.

4.1.7. Egyéb állományok

A **`Validator`** osztály tartalmazza a felhasználó által megadott adatok szerver oldali validálásáért felelős metódust. Ezáltal próbálom meg elkerülni azt, hogy a felhasználók nem megfelelő formátumú adatot(email cím, dátum, telefonszám) vigyenek fel a rendszerbe.

A **`Paginator`** osztály lehetősét biztosít a felhasználó számára megjelenítendő adatok lapozására, ezáltal áttekinthetőbbé válik a tartalom. Szabályozni tudjunk, hogy egy oldalon mennyi tartalom jelenjen meg.

4.2. Az adatbázistáblák leírása



26. ábra: Adatbázis diagram

Az alkalmazásban hét adatbázistáblát használok. Ezekben a táblákban tároljuk a felhasználók, tanárok, jegyzetek adatait, valamint azon egyetem strukturális felépítésére vonatkozó információkat ahol használni kívánjuk az alkalmazást. Ezeket a táblákat az alkalmazás telepítésekor létre kell hozni az adatbázisban.

A következőkben részletesen leírom az egyes táblák szerkezetét.

Minden tábla elsődleges kulcsa az `azon` mező.

4.2.1. A `szervezeti_egyseg, kar, tanszek` táblák

Mivel ezen három tábla szerkezeti felépítése nagyon hasonló, ezért ezek bemutatását egyszerre végzem, kitérve a különbözőségekre.

Tárolásra kerül a szervezeti egységek, karok, tanszékek megnevezése a `nev VARCHAR` típusú mezőben. A `var_name` mezőnek technikai jelentősége van, míg a `sorrend` mező felelős a tábla tartalmának megjelenítésekor a sorrendért.

A `kar` illetve `tanszek` táblákban található egy-egy külső kulcs, az `fk_szervezeti_egyseg` illetve `fk_kar`. Ezek mutatják, hogy az adott kar melyik szervezeti egységhez, illetve az adott tanszék melyik karhoz tartozik.

Ezeket a táblákat az alábbi SQL utasítással hoztam létre:

```
CREATE TABLE szervezeti_egyseg (
    azon            int             PRIMARY KEY auto_increment,
    nev             varchar(75)     NOT NULL,
    var_name        varchar(75)     NOT NULL,
    sorrend         int             NOT NULL
);

CREATE TABLE kar (
    azon                int             PRIMARY KEY auto_increment,
    nev                 varchar(75)     NOT NULL,
    var_name            varchar(75)     NOT NULL,
    sorrend             int             NOT NULL,

    fk_szervezeti_egyseg    int         NOT NULL REFERENCES szervezeti_egyseg(azon)
);

CREATE TABLE tanszek (
    azon            int             PRIMARY KEY auto_increment,
    nev             varchar(75)     NOT NULL,
    var_name        varchar(75)     NOT NULL,
    sorrend         int             NOT NULL,

    fk_kar          int             NOT NULL REFERENCES kar(azon)
);
```

4.2.2. A `felhasznalo` tábla

Ebben a táblában tárolom a felhasználók adatait. Az elsődleges kulcs az `azon`,

ezen kívül minden felhasználó esetén eltárolunk egy felhasználó nevet és jelszót `VARCHAR`

típusú mezőkként. Az alkalmazás továbbfejlesztésénél a jelszó mezőt valamilyen titkosított mezőben lenne ajánlott eltárolni.

A táblát az alábbi SQL utasítással hoztam létre:

```
CREATE TABLE felhasznalo (
    azon            int             PRIMARY KEY auto_increment,
    felhasznalonev  varchar(30)     NOT NULL,
    jelszo          varchar(32)     NOT NULL,
);
```

4.2.3. A `teacher` tábla

Ebben a táblában tárolom a rendszerbe regisztrált tanárokat, valamint adataikat.

A `nev`, `cim`, `levelezesiCim`, `szobaszam`, `telefon`, `fax`, `email`, `honlap` VARCHAR típusú mezők a tanárra vonatkozó információkat tartják nyilván.

A táblába szerepel két külső kulcs mező is, az `fk_felhasznalo`, amely a `felhasznalo` tábla azon sorára hivatkozik, amelyben a tanár felhasználó neve és jelszava található, illetve az `fk_tanszek`, amely a `tanszek` tábla azon sorára hivatkozik, amely azt a tanszéket tartalmazza amelynek a tanár tagja.

A táblát az alábbi SQL utasítással hoztam létre:

```
CREATE TABLE teacher (
    azon            int             PRIMARY KEY auto_increment,

    nev             varchar(30)     NOT NULL,
    cim             varchar(150)    NOT NULL,
    levelezesi_cim  varchar(150)    NOT NULL,
    szobaszam       varchar(10)     NOT NULL,
    telefon         varchar(25),
    fax             varchar(25),
    email           varchar(50)     NOT NULL,
    honlap          varchar(70),

  fk_tanszek        int             NOT NULL REFERENCES tanszek(azon),
  fk_felhasznalo    int             NOT NULL REFERENCES felhasznalo(azon)
);
```

4.2.4. A `note` tábla

Ebben a táblában tárolom a rendszerbe feltöltött jegyzetek információit. A jegyzet címét a `cím VARCHAR` típusú mező, a jegyzet kiadási dátumát a `kiadasiDatum DATE` típusú mező, a jegyzet rövid leírását a `leiras TEXT` típusú mező, a jegyzet letöltéséhez szükséges elérési útvonalát a `path VARCHAR` típusú mező tartalmazza. A tábla tartalmaz még egy külső kulcsot is `fk_tanszek`, arra a tanszékre hivatkozik amelyhez a jegyzet tartozik. Erre azért van szükség, mert az alkalmazásnak kell tudnia listázni a jegyzeteket tanszék szerint.

A táblát az alábbi SQL utasítással hoztam létre:

```
CREATE TABLE note (
    azon            int             PRIMARY KEY auto_increment,

    cim             varchar(250)    NOT NULL,
    kiadas_datuma   date            NOT NULL,
    leiras          text            NOT NULL,
    path            varcher(200)    NOT NULL,

  fk_tanszek        int             NOT NULL REFERENCES tanszek(azon)
);
```

4.2.5. A `teacher_note` kapcsoló tábla

Ennek a táblának a segítségével adom meg, hogy az egyes tanárok mely jegyzetet illetve jegyzeteket publikáltak, és fordítva, hogy az egyes jegyzeteknek mely tanár illetve tanárok a szerzője illetve a szerzői.

A táblát az alábbi SQL utasítással hoztam létre:

```
CREATE TABLE teacher_note (
    azon            int             PRIMARY KEY auto_increment,

  fk_teacher          int           NOT NULL REFERENCES teacher(azon),
  fk_note         int               NOT NULL REFERENCES note(azon)
);
```

5. Összefoglalás

A diplomamunka megírása során mélyebb betekintést nyerhettem a PHP nyelvbe, a benne rejlő lehetőségekbe illetve a nyelv eszközeibe. Alaposabban megismerhettem a Smarty sablonkezelő rendszert. Megtapasztaltam a konkrét tervezési, fejlesztési folyamatot.

A elkészített alkalmazás funkcióit kipróbáltam, alkalmaztam. Az alkalmazás fejlesztése során számos, az egyetemi tantárgyban megszerzett elméleti tudást használhattam fel a gyakorlatban, asz Adatbázisrendszerektől kezdve, Programozás 1 és 2-n keresztül a Rendszerfejlesztés Technológiáig.

Tervezem az alkalmazás egy teljesebb változatának elkészítését, olyan funkciók megvalósítását amelyekre a diplomamunka leadási határideje miatt nem jutott idő.

6. Irodalomjegyzék

Vég Csaba: Alkalmazásfejlesztés a Unified Modeling Language szabványos jelöléseivel, Logos 2000, Debrecen 1999

Hasin Hayder: Object-Oriented Programming with PHP5, Packt Publishing, 2007

Peter Lavin: Object-Oriented PHP, 2006

http://www.smarty.net/docs.php

http://www.php.net/manual/hu/

http://dev.mysql.com/doc/

http://www.php.net/~helly/php/ext/spl/