Debreceni Egyetem

Informatika Kar

CRM rendszer fejlesztése Zend-Doctrine integráció tükrében

Témavezető:

Kollár Lajos

egyetemi tanársegéd

Készítette:

Balogh László

programtervező matematikus

Debrecen

2010

Tartalomjegyzék

1. Bevezetés

Napjainkban egy vállalatnak nem elég csupán a készleteit számon tartania. Hiába van több száz alkalmazás, melyekkel számlákat adhatunk ki, árukészletünket ellenőrizhetjük. A jelenkor igényeinek megfelelően ma már szükség van egy olyan rendszerre, amely képes követni kapcsolatainkat, melyek meghatározzák cégünket. Segít azok karbantartásában, kezelésében, legyen szó a cég alkalmazottairól, vagy éppen cég vezetőjéről akivel kapcsolatban állunk, álltunk. Minden fontos, vagy nem túl fontos információt, dokumentumot, eseményt, jegyzetet vagy egy megjegyzést tudjunk csatolni bármelyik vállalkozásunkkal kapcsolatban álló személyhez. Ezen kívül, a mai kornak megfelelően akár twitteren vagy egyéb közösségi oldalon is nyomon tudjuk követni vásárlóinkat, ezáltal új kommunikációs csatornák nyílhatnak meg közöttünk.

Az elkövetkező oldalakon szeretném magát a CRM-et (Customer relationship management), mint a közép és nagyvállalatok egy elengedhetetlen eszközét bemutatni. Mire is jó, miért is lehet hasznos. Azonban diplomamunkám fő célja, hogy megmutassa a Doctrine és a Zend Framework által képviselt erőt, amellyel egy objektum orientált, MVC (Model-View-Controller) mintára épülő, felhasználóbarát alkalmazás építhető fel webes környezetben. Mindezt ingyen, nyílt forrású szoftverek felhasználásával. Tapasztalataim azt mutatják, hogy ezek az eszközök kis és közép vállalkozásoknál megállják a helyüket. Azonban tanulmányok és más CRM rendszerek alapos vizsgálata után azt tudom mondani, hogy mindezen eszközök nagyvállalati környezetben is helyt tudnak állni.

A Zend Framework segítségünkben lesz az MVC szerkezeti minta megvalósításában, és az objektum orientált paradigma eszközeit is könnyebb a keretrendszeren belül alkalmazni. A Doctrine integrációja még nem történt meg a Zend keretrendszerébe, ez majd csak a Zend Framework 2.0 -ás verziójával fog bekövetkezni 2010 novemberében. Ezért diplomamunkában szó esik majd a Doctrine Zend-be való beillesztéséről is.

Harmadik nagyon erős eszközöm a jQuery lesz, melynek kiegészítői és egyszerű használhatósága megkönnyítik az AJAX használatát és szkript programozást. Ezekkel az eszközökkel segít felhasználóbarátabbá tenni programomat legyen szó validációról, vagy akár az oldalt megszépítő legördülő menüről.

Fontosnak tartottam azt, hogy ezek a technológiák bárki számára elérhetőek legyenek és kis befektetéssel is üzemben tartható legyen a rendszer. Telepítése nem bonyolult csupán egy Apache alkalmazás szerverre és egy MySQL adatbázisra van szükség, melyek mind szabadon letölthetőek. A program maga webes felületen érhető el, ezért mondhatni platformfüggetlen, csupán egy böngészőt kell telepítenünk, ha még nem rendelkeznénk vele. A program hordozhatósága is könnyebbé válik ezáltal. A „böngészők háborúja” se kell hogy megzavarja munkánkat, hiszen a Zend Framework által generált kód szabványos és a legtöbb jelenkori böngésző számára ideális.

Törekedtem arra, hogy a kezelőfelület egyszerű, átlátható és letisztult legyen. Ez megnyilvánul a menük felépítésében, illetve a színek szolid használatában.

2. Doctrine

2.1 Doctrine

ORM rendszer PHP 5.2.3+ környezetben, amely az adatbázis egy absztrakt rétegével kommunikál (DBAL). Előnye hogy az adatbázissal kapcsolatos lekérdezéseinket SQL helyett DQL (Doctrine Query Language)-ben írhatjuk, amelyet a Java környezetben jól ismert Hibernate ihletett. Ezzel az eszközzel a fejlesztés felgyorsul, a kódok egyszerűsödnek és mindez kód duplikáció nélkül történik.

2.2 ORM

Az objektum relációs leképzés egy technika, amellyel akkor találkozhatunk, ha egy a relációs adatbázissal valamilyen programozási nyelvből származó, a relációs adatbázissal nem kompatibilis adatot szeretnénk feldolgoztatni. Az ORM lényegében megengedi nekünk, hogy egy virtuális objektum adatbázison dolgozzunk, amely már együtt tud működni a meglévő programozási nyelvvel.

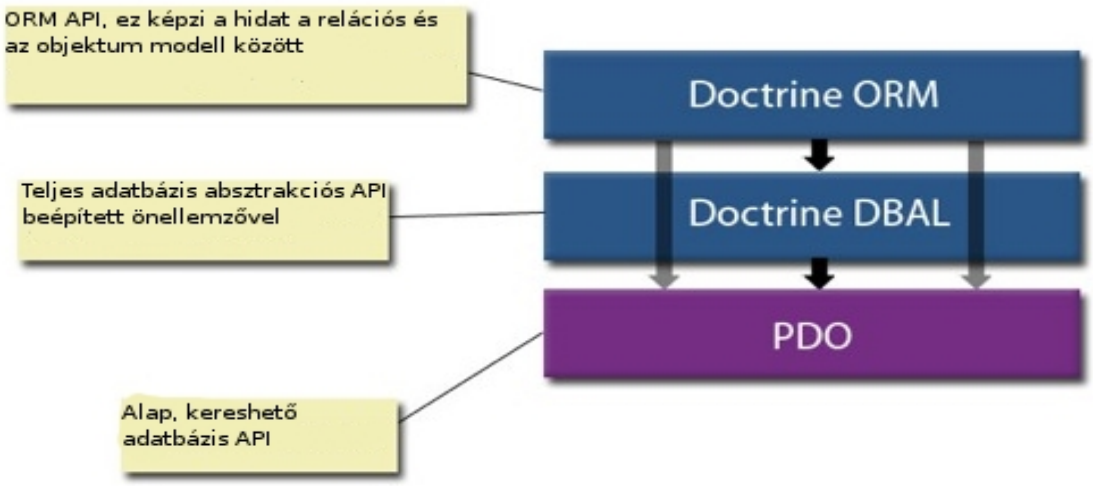
2.3 Mi is a probléma?

Ha egy webes alkalmazást fejlesztünk, akkor jó eséllyel objektum orientált programozást fogunk használni, amely objektumokkal manipulál. Ezek általában nem skalár értékek, viszont a legtöbb adatbázis kezelő csak ilyen adatokat tud tárolni, illetve csak skalár értékekkel tud műveleteket elvégezni A fejlesztőnek kell az objektumokat az adatbázis számára feldolgozható formára hozni, vagy ha az adatbázisból kinyert adatot szeretnénk a programunkkal feldolgozni ezt meg kell tennünk visszafelé is. Az ORM erre nyújt egy jó megoldást.

A fő problémát az jelenti, hogy úgy alakítsuk át az objektumokat, hogy az adatbázisból könnyen elérhetőek legyenek és mégse veszítsék el eredeti tulajdonságaikat valamint egymáshoz való viszonyukat. Az ilyen objektumokat perzisztensnek tekintjük.

2.4 Alapvető felépítés

Az objektum relációs leképzés eszköze PHP-ban a Doctrine. A PDO-val kommunikál. A PDO (PHP Data Objects) nem más, mint egy objektum az adatbázis kapcsolatok, lekérdezések, stb. kényelmes, hatékony, átlátható kezelésére[1]. A Doctrine két fő részből tevődik össze, a DBAL és az ORM. Az alábbi kép illusztrálja, hogyan függnek össze ezek a rétegek.



1. ábra: ORM felépítése

A DBAL (adatbázis absztrakciós réteg) kiegészíti és kiterjeszti a PDO által nyújtott absztrakciót, függetlenséget. A DBAL használható önállóan is, ha csak egy erős absztrakciós rétegre van szükségünk a PDO tetején. Az ORM réteg függ a DBAL rétegtől és ezért, amikor az ORM csomag betöltődik a DBAL már benne szerepel.

A Doctrine ORM-je az Acitve Record, Data Mapper és Meta Data Mapping tervezési mintákra épül. A Doctrine_Record alap osztály kiterjesztése révén, minden leszármazott osztály rendelkezni fog a tipikus Active Record interfészekkel úgymint; törlés, mentés, stb. és a Doctrine számára lehetőséget ad, hogy könnyedén figyelemmel kísérje a rekord életciklusát. Egyéb adatbázis műveletek más komponensekhez vannak továbbítva, ilyen komponens a Doctrine_Table osztály is. Ez az osztály Data Mapper interfésszel rendelkeznek tipikusan, createQuery(), find(id), findAll(), findBy*(), findOneBy*() stb.

1 http://deadlime.hu/2006/02/11/mi-is-az-a-pdo/

2.6 Modellek

Az adatbázist a legalsó szinten a Doctrine PHP osztályok segítségével reprezentálja. Ezek az osztályok adják meg a sémáját és a viselkedését a modelljeinknek. Minden Doctrine_Record-ból öröklött osztály tartalmaz egy setTableDefiniton() és egy setUp() metódust. A setTableDefiniton() segítségével tudunk oszlopokat, indexeket és egyéb információkat definiálni a táblával kapcsolatban. A setUp() eljárásban írhatjuk le a táblával kapcsolatos viselkedést, illetve a táblák közötti relációkat.

```php
<?php
class User extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('username', 'string', 255);
        $this->hasColumn('password', 'string', 255);
    }

    public function setUp()
    {
        $this->actAs('Timestampable');
    }
}
```

2.ábra: Doctrine Record

Így néz ki egy tipikus táblát leíró osztály, rendelkezik az előbb említett két osztállyal és a Doctrine_Record osztályból származik *(2.ábra)*.

A modelljeinket nem szükséges kézzel begépelnünk. Mindezt PHP kóddal, illetve a Doctrine parancssoros interfészével, vagy már létező adatbázisból, vagy pedig a YAML fájlból kigenerálhatjuk a modelljeiket.

2.7 YAML

Az oka, hogy YAML (Yet Another Markup Language) séma fájlokat használok az az, hogy így könnyebben tudom kezelni a modelljeimet. Közvetlenül a YAML fájl szerkesztésével, ahelyett hogy PHP kódhoz kellene folyamodnom. A YAML fájl segítségével le tudjuk generálni az összes modellt, amire a fejlesztés folyamán szükségünk lesz. És ez az, ami Doctrine modelleket hordozhatóvá teszik. A séma fájlokkal mindent megvalósíthatunk, amit a PHP kóddal, így semmilyen hátrányunk nem származik abból, hogy csupán ezeket a

fájlokat használjuk. Ugyanúgy be tudjuk állítani a adatbázis kapcsolathoz szükséges információkat, a táblák közötti kapcsolatokat, attribútumokat, viselkedésmódokat, indexeket, stb.

2.7.1 Rövidítés a szintaxisban

A Doctrine lehetőséget ad arra, hogy a sémafájlokat rövidített szintaxissal írjuk. Számtalan, a sémában szereplő paraméter rendelkezik alapértelmezett értékkel, amely lehetővé teszi a rövidített szintaxis használatát és így hagyhatjuk, hogy a Doctrine a beépített értékekkel dolgozzon.

Érdemes megemlíteni detect_relation opciót, melyet ha YAML sémánkba igaz értékre állítva írunk, akkor megpróbálja az oszlopok neve alapján megtalálni az adatbázistáblák között lévő kapcsolatokat.

2.7.2 Szintaxis

Ha nincs kapcsolat a tábláink között, akkor egész egyszerűen megadhatjuk azok definícióját YAML fájlunkban:

```
1 táblaneve:
2   columns:
3     mező_neve:
4       type: tipus(méret)
```

3.ábra: YAML szintaxis

Fontos a sorrend kötöttsége, valamint a bekezdéseket is tartanunk kell. Minden bekezdés két szóközzel kezdődik beljebb, tabulátort nem használhatunk *(3.ábra).*

Ha van kapcsolat a tábláink között, akkor meg kell adnunk a külső kulcsot azon az oldalon amelyen az létezik. A kapcsolat másik oldala visszatükrözi ezt és felépíti a kapcsolatot a szemközti oldalról is. A séma szintaxisa lehetőséget nyújt alias nevek használatára, amelynek az jelentősége, hogy egy helyen tarthatjuk a fontos információkat a kapcsolatainkról.

2.8 DQL

Doctrine Query Language (DQL) a Doctrine saját lekérdező nyelve, mely segítségünkre van abban, hogy a komplex lekérdezések eredményesek és effektívek legyenek. Számos előny származik abból, hogy SQL, helyett a DQL nyelvet használjuk.

Először is objektumokkal tudunk dolgozni, nem pedig eredmény sorokkal. Nem kell kézzel megírnunk a JOIN művelteket, mivel a DQL felismeri a kapcsolatokat a táblák között. Több adatbázis kapcsolattal is dolgozhatunk, amely adatbázis kapcsolat végén különböző adatbázisok lehetnek. Sok olyan funkcionalitást tartalmaz, amely megkönnyíti munkánkat, főleg ha 1:n vagy n:m típusú táblákból kell lekérdeznünk. Ha nem lennénk ezzel megelégedve a Doctrine kínál más lehetőséget is, a RawSql-t.

3. Zend Framework

A Zend Framework egy nyílt forráskódú, objektum orientált keretrendszer web alkalmazások számára, PHP 5+ környezetben. A keretrendszert nevezhetnék komponensek könyvtárának is, hiszen nem áll másból, mint lazán kapcsolt, többnyire független komponensekből, melyek könyvtárakba vannak szervezve. Ezen felül a Zend Framework MVC (Model-View-Controller) megoldást kínál az alkalmazásunk felépítéséhez.

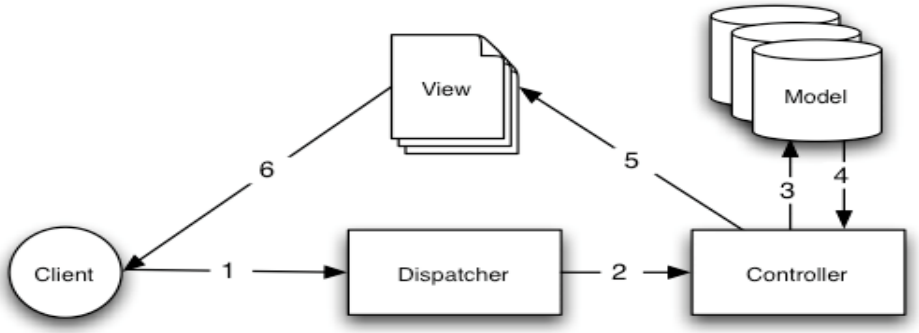
3.1 Model-View-Controller

MVC szerkezeti minta, amit napjainkban a legtöbb web-alkalmazás fejlesztésénél használnak. Az MVC minta lényege, hogy külön válassza az üzleti logikát, az adatbázishoz való hozzáférést, és a megjelenítési réteget. Ez a szeparálás segít megtartani a kódunk jól szervezettségét, ami akkor nagyon fontos ha egy alkalmazáson több ember is dolgozik.

- Modell : a modellekben általában adateléréssel, vagy az üzleti logikához kapcsolódó rutinok találhatóak
- Nézet: a nézet definiálja azt, amit a felhasználónak meg akarunk mutatni. Általában a vezérlő adja át az adatokat a nézetnek, amely valamilyen formában megjeleníti azt. A nézet feladata az is, hogy a felhasználóktól adatokat gyűjtsön be, majd továbbítsa a vezérlőnek. Ez az a része az alkalmazásnak, ahol HTML kódokat is használunk, ez a

minta többi részére nem jellemző.

- Vezérlő: a vezérlő köti össze a tervezési mintát, manipulálja a modelleket, eldönti, hogy mit jelenítsünk meg. A vezérlő mondja meg, hogy melyik adat melyik nézethez tartozik, ő dolgozza fel a felhasználótól beérkező kéréseket, vagy akár átadja a vezérlést más vezérlőnek.



4.ábra: MVC

3.2 Zend Framework projekt készítése

3.2.1 A futtatókörnyezet

A Zend Framework használatához, PHP 5+ verziójára van szükség, illetve egy Apache alkalmazás szerverre. Ezek után csak annyi a dolgunk, hogy engedélyezzük a Rewrite engine modult a .htacces fájlban, és már el is kezdhetjük az alkalmazás fejlesztését.

3.2.2 Könyvtárszerkezet

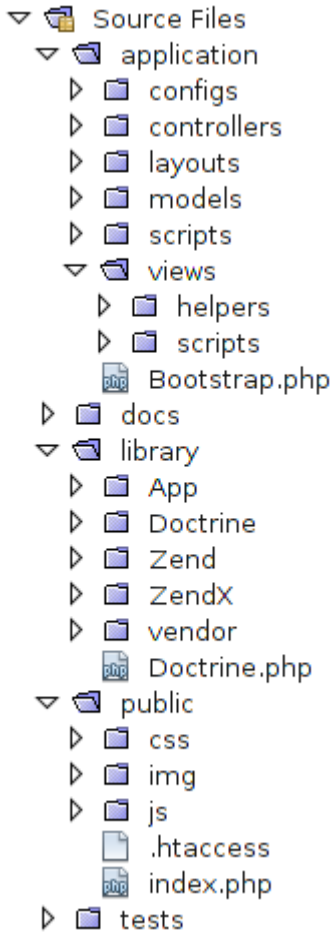
Az MVC mintát meghatározó könyvtárszerkezet az alábbi módon néz ki. Ennek a generálására képes a Zend_Tool parancssoros eszköz is.

A könyvtárfa fő elemei az application, library, public, illetve test könyvtárak alkotják.

- Az application mappa tartalmazza a nem publikus, az alkalmazásunk szempontjából fontos könyvtárakat, illetve fájlokat. Ez a mappa magában foglalja szeparálva az MVC minta elemeit.
    - A vezérlőket a controllers mappában találjuk. Névkonvenció, hogy minden vezérlő

neve Controller-re végződik.

- A models mappába kerülnek a Doctrine által használt modellek, de ha Zend_Table-t használnák, akkor is itt kellene elhelyeznünk a modell osztályokat.
- A views tartalmazza a nézettel kapcsolatos mappákat, legyenek azok akár helperek, vagy akár az actionok-hoz kapcsolódó nézetek. A nézetek, a vezérlő neve szerint vannak csoportosítva további alkönyvtárakban. A layout az alkalmazás sablonja, az általam felépített alkalmazásban a view mappán kívülre került, az egyszerűbb kezelhetőség érdekében.
- Az application mappában található a Bootstrap.php, amelyről a későbbiekben még szót fogok ejteni.
- A configs mappában találhatóak a keretrendszerek beállításával kapcsolatos fájlok, mind a Doctrine mind a Zend Framwework-höz tartozóak. A Zend többféle fájltípust is támogat a konfigurációs fájl számára, én az .ini kiterjesztésűt részesítem előnyben, mivel azt a PHP által használt parse_php_ini metódus gond nélkül fel tudja így dolgozni.



5.ábra: Zend könyvtárszerkezet

- A library könyvtárban a két keretrendszer könyvtár csomagjai helyezkednek el, illetve az általam használt egyéb PHP osztályok számára itt található egy App mappa, a ZendX pedig a Jquery integráció miatt szükséges.
- A public könyvtár tartalmazza a js, css, img mappákat, melyekben a JavaScript fájlok, stílus fájlok, illetve képek vannak. Ez a mappa a külvilág számára is elérhető.
- A test mappa - amelybe tesztjeinket írhatjuk - melyekkel ellenőrizhetjük, hogy egy-egy leprogramozott folyamat jól működik e.

3.2.3 Bootstrap állomány

A Bootstrap osztályban definiálhatunk erőforrásokat, és az alkalmazás számára fontos komponenseket, valamint kulcsfontosságú szerepet tölt be az alkalmazás betöltése előtt. Ebben az állományban minden metódus, amely _init-el kezdődik az alkalmazásunk futása előtt betöltődik.

```
1. // application/Bootstrap.php
2.
3. class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
4. {
5. }
```

6.ábra :Bootstrap állomány

3.2.4 Vezérlők

A vezérlők írják le a munkafolyamatainkat és dolgozzák fel a beérkező kéréseket, azokat tovább irányítva a modellek vagy nézetek felé. A Front Controller tervezési mintát követik a kontrollerek. Egy vezérlőnek több metódusa is lehet, amely Action-re végződik. Ezek a metódusok azok, amelyek fogadják a kéréseket, amik a felhasználók felől érkeznek.

```
class IndexController extends Zend_Controller_Action
{
    public function init()
    {
        /* Itt tudjuk megtenni a vezérlő inicializálást*/
    }

    public function indexAction()
    {
        /* az index action, amely az index/index oldallal kommunikál*/
    }
}
```

7.ábra: IndexController

Alapértelmezetten a Zend Framework a következő URL sémát követi: /controller/action. Ahol a 'controller' határozza meg a vezérlőt, az 'action' pedig a vezérlő által tartalmazott action függvényt. Tipikusan mindig van egy IndexController, amely az alkalmazásunk kezdőoldalát biztosítja, illetve egy ErrorController, amely kezeli és megjeleníti

az esetleges hibákat, amelyek a program futása közben keletkeznek.

3.2.5 Nézetek

Ahogy azt az előző fejezetben is említettem, a nézet a scripts mappában helyezkedik el vezérlőnként csoportosítva. Az alapértelmezett vezérlő és a hozzá tartozó aciton a következő: applications/views/scripts/index/index.phtml.

A nézetekben használhatunk úgynevezett View Helpereket. Az alábbiakban ezekről lesz szó. Vannak olyan részei egy weboldalnak amiket újra és újra felhasználunk, legyen szó akár egy űrlap elemről, vagy egy formázott dátumról. Ezeknek fenn tarthatunk egy külön helper osztályt, amelyekben megvalósíthatjuk a kívánt funkciókat. A helper egy egyszerű osztály, alapértelmezés szerint Zend_View_Helper_ előtaggal kell ellátnunk. Az aláhúzás jel után írhatjuk a nevet, amit e helper-ünknek szánunk. Ennek az osztálynak rendelkeznie kell minimum egy metódussal, ami a helper nevét viseli.

A Zend_View már tartalmaz beépített helper osztályokat, amelyeknek legtöbbje űrlapok kigenerálásánál van segítségünkre. Mindezek elvégzik a bemenet megfelelő formázását. Ezen kívül még vannak helperek, melyekkel URL -t adhatunk meg, listákat vagy akár egyszerűen változót deklarálhatunk. Néhány általam is sűrűn használt ilyen helper:

- formText($név,$érték,$tulajdonságok): egy <input type="text" /> elemet készít
- formSelect($név, $érték, $tulajdonságok, $beállítás): egy <select>...</select> blokkot készít, a $beállítás változó lehet egy asszociatív tömb is, melynek kulcs értéke lesz a <option> tag értéke, a kulcshoz tartozó érték pedig az <option> címkéje
- formCheckbox($név, $érték, $tulajdonságok, $beállítások): <input type="checkbox" /> form elemet generál, ha nincs $érték és $beállítások beállítva, akkor az alapértelmezett az, hogy ha be van pipálva a kijelölő négyzet akkor a form értéke egy, különben nulla. Egyébként a $beállítások mint asszociatív tömb határozza meg annak értékét, a kulcs a bepipált értéket, az érték pedig a pipa hiányának értékét adja meg.
- formSubmit($név,$érték,$tulajdonságok): <input type="submit" /> elemet generál.

A $tulajdonság megadása mindig asszociatív tömb formájában történik a következőképpen 'tulajdonság, amit meg szeretnénk változtatni' => 'érték, amire be szeretnénk állítani'.

4. jQuery

Manapság a világháló teljesen dinamikussá vált, a felhasználóknak magas elvárásaik vannak mind az oldal kinézetével, stílusával, mind pedig funkcionalitásával kapcsolatban. Ahhoz, hogy interaktív, érdekes oldalakat tudjunk fejleszteni, szükségünk van valamilyen JavaScript könyvtárra ebben az esetben a jQuery csomagjára.

4.1 Jellemzői

A jQeury könyvtárcsomag sokféle alkalmazhatóságot kínál a webes szkriptek esetében. A következőek a legfontosabbak, amiket megtehetünk a jQuery-vel:

- Elérhetünk bármely DOM elemet. A könyvtárcsomag nélkül elég sok sorba kerülne az, hogy meg tudjuk fogni a DOM (Document Object Model) fa szerkezet egy elemét és meghatározzuk a pontos helyét a HTML struktúra egy darabjának. De a jQuery-vel mindez könnyűvé válik, hiszen rendelkezik egy nagyon hatékony kijelölő mechanizmussal, amellyel pontosan meg tudjuk fogni azt az elemet amire szükségünk van, és képesek vagyunk manipulálni azt.
- Módosíthatjuk weboldalunk kinézetét. A css() nevű metódus segítségével megváltoztathatjuk azt, hogy milyen stílusú legyen az oldalunk. Azonban nem minden böngésző támogat minden szabványt. De a jQuery-vel áthidalhatjuk ezt a problémát, mivel az addClass() és removeClass() függvényekkel saját „szájunk íze” szerint alakíthatjuk elemeink osztály attribútumát, azt követően is, hogy az oldal kirajzolódott.
- Megváltoztathatjuk a kijelölt dokumentumelem tartalmát. Ez nem korlátozódik le pusztán kozmetikai változtatásokra. Egy pár billentyűleütéssel képes a jQuery átalakítani az egész oldalt. Megváltoztathatjuk a szöveget, képeket szúrhatunk be, cserélhetünk le, listák rendezhetőek át, vagy az egész HTML szerkezetet átírhatjuk.

- Reagálhatunk a felhasználói beavatkozásokra. Még a leginkább kidolgozott funkciók is használhatatlanok, ha nem tudunk megfelelő választ adni a külvilág számára. A jQuery könyvtár használata elegáns módot nyújt ahhoz, hogy elfogjuk a legkülönbözőbb eseményeket, és kezeljük azokat a jQuery által nyújtott esemény kezelővel.
- Animációkkal tűzdelhetjük meg az oldalunkat, annak érdekében, hogy hatékonyan tudjuk érzékeltetni az interaktív viselkedést. A jQuery kínál egy sor beépített effektet, mint például elhalványulás, eltűnés, és ha új elemet adunk a DOM-hoz, azokhoz is tartozhat hasonló animáció.
- Adatokat tölthetünk le a szerverről, anélkül hogy az oldalt frissítenünk kellene. Ez a programozási minta úgy ismert, mint aszinkron JavaScript és XML (AJAX), mely segíti a fejlesztőt egy funkciókban gazdag oldal fejlesztésében. A jQuery eltünteti a böngészőkre jellemző komplexitást ebben a témában, és lehetővé teszi a programozónak, hogy a szerveroldali funkcionalitásra koncentrálhasson.
- Egyszerűsíti az általános JavaScript feladatokat. A dokumentum specifikus sajátosságok mellett, a könyvtárcsomag kiegészíti a JavaScript eszközeit, legyen szó tömb manipulációról, vagy elemek bejárásáról.

A dinamikus HTML népszérűsödésével, egyre több JavaScript keretrendszer jelent meg. Egyesek specializálódtak egy-egy feladatra az alábbiak közül. Mások kísérletet tettek arra, hogy az összes animációt, funkcionalitást előre csomagolva tudják kínálni. A jQeury a következőket kínálja:

- Meghatározó CSS tudás. CSS kiválasztókkal meg tudunk találni DOM elemeket az oldal struktúrájában, ez a legkifejezőbb és legtömörebb módja annak, hogy kijelöljünk egy elemet. És mivel a legtöbb web-fejlesztő ismeri a CSS szintaxisát, ennek az elsajátítása már nem okoz problémát.
- Kiterjesztések támogatása. Annak a módja, hogy új kiterjesztést írjunk nagyon egyszerű, és jól dokumentált, amely azt ösztönözte, hogy számos ötletes és hasznos modul létrejöjjön. Az alap jQuery fájl is erre az elvre épül, azaz eltávolíthatunk belőle

részeket amire nincs szükségünk, melynek az eredménye egy még kisebb csomag lesz.

- Böngészőfüggetlenség. Sajnos minden böngészőnek megvan a saját készlete, szabványa arra, hogy hogyan is jelenítsen meg egy weboldalt. A jQuery biztosít egy absztrakt réteget, amivel ez a különbség eltűnik. Ezzel redukálva kódunk méretét úgy, hogy nem kell minden böngészőre külön-külön kódot írnunk.
- Objektumok halmazával dolgozik. Ahelyett, hogy mindig egy elemet fognánk meg, azok halmazát együtt tudjuk kezelni. Ezt a technikát implicit iterációnak nevezik. Ez azt jelenti, hogy számos bejárás szükségtelenné válik, jelentősen megrövidítve a kódot.
- Lehetővé tesz több utasítást egy sorban. Így elkerülhetjük az ideiglenes változók túlzott használatát. Láncolásnak nevezett programozási mintát követ a jQuery. Ami azt jelenti, hogy ha egy objektumon elvégzünk egy műveletet annak eredménye maga az objektum lesz, készen állva arra hogy a következő tevékenységet is végrehajtsuk rajta.

Ezek a stratégiák segítik a jQuery tömörített változatát 20KB alatt tartani, minden jó tulajdonsága ellenére a jQuery ingyenes, nyílt forráskóddal rendelkezik.

4.2 Használata

A hivatalos weboldalon mindig elérhető a legfrissebb verziója a jQuery-nek. Csupán annyi a dolgunk, hogy letöltjük. Az oldalon több verziója is megtalálható a jQuery-nek, számunkra a tömörített verzió a legoptimálisabb, a tömörítetlen verziót. Nem szükséges telepíteni, bemásoljuk a Zend által JavaScript fájlok számára biztosított mappába. Amely oldalon szükségünk van rá csupán hivatkoznunk kell a letöltött fájlra.

```
<script type="text/javascript" src="jquery.js"></script>
```

8. ábra: Hivatkozás a jQuery-re

Biztosnak kell lennünk abban, hogy hozzáadjuk az eseményeket stb., amint a DOM készen van. Ehhez van egy beépített eseménykezelő, amit szinte mindig használunk.

```
$(document).ready(function() {
   // ha a DOM készen van
});
```

9. ábra: Beépített esemény kezelő mely figyeli hogy a DOM felépült e

4.2.1 A kellő elem kiválasztása

Alapvető művelet az, hogy egy DOM elemet kiválasszunk, ez a $() konstruktorral történik, melynek paramétere jellemzően egy sztring, ami tipikusan egy CSS kijelölő kifejezés. Egy jQuery objektum magába foglalhat egy vagy több DOM elemet is és lehetővé teszi számunkra, hogy különböző módokon interakcióba lépjünk velük.

5. Doctrine Zend integrációja

A választásom azért esett a Doctrine-ra, a Zend Framework által használt Zend_Db_Table helyett, mivel véleményem szerint az nem elég praktikus. Ellenben a Doctrine a gyakorlati életben jobban használható. Ennek oka pedig, hogy a Zend_Db_Table nem kínál elég kiterjedt funkcionalitást. Akár kapcsolatokról van szó, akár hierarchikus adattípusokkal szeretnénk dolgozni. Szerencsére a Doctrine 2.0 -val a Doctrine elnyeri a Zend Framework teljes támogatását. Ezzel a fúzióval egy igazán jól használható MVC alapú keretrendszert kapunk. Mivel még nincs integrálva a Zend-be a Doctrine könyvtárcsomag, ezért néhány lépésben be fogom mutatni, hogy mit is kell tennünk ahhoz, hogy alkalmazásunkban használni tudjuk a Doctrine-t.

Először is a projekthez létrehozott könyvtárszerkezeten kell változtatni, a Doctrine hivatalos honlapjáról letöltött csomagjában találjuk a lib könyvtárat, ami magában foglal mindent, amire szükségünk van a keretrendszer futtatásához. Ebből a lib könyvtárból a saját library könyvtárunkba be kell másolnunk a Doctrine, vendor mappákat, illetve a Doctrine.php fájlt. Létre kell hoznunk néhány új mappát is, amelyek szükségesek a Doctrine működéséhez. Az applications/configs mappán belül egy data könyvtárat, amiben az adatkezeléssel kapcsolatos fájljaink fognak kerülni. Ennek a könyvtárnak két alkönyvtára van, az egyik a

fixtures, amely datafixture-ket tartalmaz, melyek arra szolgálnak, hogy tesztadatokkal fel tudjuk tölteni az adatbázisunkat anélkül, hogy az adatbázishoz kellene kézzel nyúlnunk. A másik almappája a data mappának az sql.

A data mappán kívül kell még egy migration mappa, ahová az adatbázis migrálással kapcsolatos adatok kerülnek. Migrálásról akkor beszélhetünk, ha az adatbázis egyik verziójáról egy másik verzióra szeretnénk váltani.

A config mappába, már csak az adatbázist leíró YAML fájlt kell létrehoznunk. Az application könyvtárba kialakítunk egy scripst mappát, amiben a doctrine shell szkript helyezkedik el. Ami csak annyit tesz hogy beállítja a PHP környezeti változót, mappát vált, majd lefuttatja a mellette található doctrine.php-t, ezzel a szkript fájlal tudjuk elérni a Doctrine parancssoros interfészét. A doctrine.php-ban értéket adunk a környezeti változóknak, és elérési utaknak. Majd lefuttatjuk a bootstrap állományban létrehozott _initDoctrine() függvényt az $application->getBootstrap()->bootstrap('doctrine') paranccsal. Az alkalmazás 'doctrine' tagból tudja, hogy a Bootstrap.php melyik metódusáról van szó.

Miután létrehoztuk az előbb említett mappákat azután, az application.ini konfigurációs fájlban be kell állítanunk az időzónát, amit használni szeretnénk. Ezután meg kell adni az adatbázis kapcsolati sztringet. Majd az előbbiekben létrehozott könyvtáraknak meg kell adnunk az elérési útját.

```
doctrine.dsn                  =  "mysql://root:33333@localhost/dipo_crm"
```

10. ábra: Adatbázis kapcsolatot leíró sztring

```
doctrine.data_fixtures_path = APPLICATION_PATH "/configs/data/fixtures"
doctrine.sql_path           = APPLICATION_PATH "/configs/data/sql"
doctrine.migrations_path    = APPLICATION_PATH "/configs/migrations"
doctrine.yaml_schema_path   = APPLICATION_PATH "/configs/schema.yml"
doctrine.models_path        = APPLICATION_PATH "/models"
```

11. ábra: Elérési utak

Regisztrálnunk kell a 'Doctrine' névteret az autoloader számára, hogy az alkalmazásunk felismerje a Doctrine mappáját a library könyvtárban és így használhatóak legyenek a keretrendszer állományai.

Utolsó lépésként a Bootstrap.php -ba kell megírnunk az _initDoctrine() metódust, melyben beállítjuk az adatbázis-kapcsolatra vonatkozó tulajdonságokat. Ezeket az attribútumokat az alkalmazás az előbbikben megírt .ini fájlból tölti be $this-> getOption('doctrine') parancs által, így a konfigurációs fájlban minden ami 'doctrine'-al kezdődik be fog töltődni egy megadott változóba, mint asszociatív tömb. További beállításokat is eszközölünk például, hogy a modelljeinket konzervatív módon töltse be, ami azt jelenti, hogy csak azok az osztályok legyenek betöltve amelyekre szükségünk van.

A Zend-nek van saját Autoloader-re, amely a kívánt osztályokat tölti be, fontos, hogy a Doctrine-nak is regisztrálnunk kell egy Autoloader-t. Ezzel már használható is a Doctrine. De mi azt szeretnénk, hogy a Doctrine használja a Zend által előírt névkonvenciókat, hogy programunk minden pontja egységes maradjon. Ennek érdekében az application.ini-t további sorokkal bővítjük, engedélyezzük pearStyle tulajdonságot, amely az említett konvenció használatának lehetőségét adja. Már csak meg kell adnunk az előtagokat, baseClassPrefix attribútumot 'Base_' -re állítjuk, amely azt jelenti, hogy az alap modelljeink, amelyikből majd a többi modell származik, kezdőjenek Base-zel. Ezután már csak a leszármazott modelleknek kell megadnunk a prefix tagját classPrefix opcióval melynek értéke 'Model_' lesz. Így most már a models mappában elhelyezkedő osztályaink a következőképpen néznek ki: Model_Modellneve.

Sajnálatos módon egy hiba miatt a parancssoros módból nem tudjuk használni a modellek konzervatív betöltését. Ezért itt a konfigurációs fájlban megadjuk, hogy amikor parancssorból adunk ki utasítást az alkalmazás használjon agresszív modellbetöltést, de maga a program ugyanúgy konzervatívan tölti be a modelleket, ezzel erőforrást spórolva.

6.CRM rendszer

Az elkövetkező fejezetben leírom azt, hogy mivel is kell foglalkoznia egy CRM rendszernek, milyen követelményeket állíthatunk fel vele szemben.

6.1 Mi is az a CRM

Az angol rövidítés feloldása Customer Relationship Management, amit ügyfélkapcsolat kezelésnek fordíthatunk. A CRM olyan rendszer, amely lehetővé teszi egy vállalkozás számára azt, hogy az ügyfelet helyezze a középpontba. Ezzel megteremtve egy új szemléletmódot, és a gazdálkodó szervezet számára új működési stratégia kialakítását. Ez egy információs rendszer, amelynek célja az ügyfelekkel való kapcsolattartás valamint a kapcsolattartási és mindenféle olyan információ rendszerezése, amely az ügyféllel kapcsolatosak; lehet az akár marketinghez, értékesítéshez, vagy valamilyen egyéb üzleti folyamathoz kötődő. Ezekkel az eszközökkel az üzletfelek menedzselése nívósabbá válik. Másik fő célja a jelenlegi üzletfelek megőrzése, és új potenciális ügyfelek megnyerése.

6.2 Követelmények

Az alábbiakban 5 fő részre osztanám azok a funkcionális követelményeket, amelyek egy ilyen rendszertől elvárhatóak. Megjegyezném, hogy ezeket a követelményeket tártam fel a program írásának elkezdése előtt. Véleményem szerint ezek azok, amelyeket egy jelenlegi CRM rendszernek tudnia kell.

6.2.1 Partnerek

Ahogy említettem ez egy ügyfélközpontú rendszer, melynek legfontosabb eleme maga az ügyfél. Mindaddig amíg ügyfélről van szó, addig CRM rendszerről beszélünk. Nagyon fontos a kapcsolattartás, az üzleti partner adatainak nyilvántartása. Lehetőséget kell nyújtanunk ahhoz, hogy egy-egy kliensünkről több rekordnyi kapcsolati információt tudjunk tárolni. Ezek a mai kornak megfelelően nem csak egy telefonszámig, vagy címig terjednek. Lehetőséget kell adnunk például azonnali üzenetküldésre alkalmas alkalmazások azonosítójának, twitter azonosító vagy akár a weboldalnak a címének a tárolására, ahol el tudjuk érni a partnerünket. De ezen felül persze nem feledkezhetünk el a sztenderd adatokról sem. Kereshetőnek kell lenniük, biztosítanunk kell egyszerű és összetett keresési funkciókat. Az ügyfél adatait tudnunk kell módosítani, törölni. Partnerek csoportosításáról is

gondoskodnunk kell. A jegyzetek hozzáfűzésének opciója, lehetőséget ad arra, hogy a felhasználó a partnereihez jegyzeteket fűzzön, és ehhez a rendszer egyéb felhasználója is akár kommentálni tudjon, vagy adatot csatolni hozzá.

6.2.2 Tevékenységek

Minden ügyfélről tudjunk vezetni egy tevékenységlistát, ahová beírhatjuk a klienssel kapcsolatos fontos eseményeket, legyen szó egy ebédmeghívásról, vagy arról, hogy az illetővel fel kell bontanunk az üzleti szerződést. Ezekről az emlékeztetőkről e-mailben, vagy sms szolgáltatáson keresztül kaphassunk értesítést.

Különböző fontossági szintekkel, kategóriákkal kell, hogy rendelkezzen egy-egy tevékenység, ezzel is segítve a felhasználót. A programnak csoportosítania kell a meglévő dátumokat, megfelelő időintervallumonként. De nem elég csupán az eseményeket megjegyeznünk, fontos az, hogy partnerünkkel kapcsolatos lényeges dátumokat is el tudjuk menteni. És ha kell ezekről is értesítést kapjunk. Naplózzuk ezeket, ezzel elérve, hogy elemzéseket készíthessünk a későbbiekben partnerünk szokásairól.

6.2.3 Ügyek

A partnereket képesek legyünk ügyek alá szervezni, ezek alá az ügyek alá akár több partner is tartozhat. Ugyanúgy lehetőséget kell biztosítani, hogy kommentálhassuk ezeket, illetve fájlokat fűzhessünk egy-egy ügy alá. Tevékenységet kapcsoljunk hozzájuk.

6.2.4 Más programokkal való kapcsolat

Ha már van egy meglévő rendszerünk, amiben ügyfeleinket, határidőnaplónkat tartjuk, egyszerűen szinkronizálható legyen az alkalmazásunkkal. Legyen szó akár az MS Outlook-járól, vagy a Google címjegyzékéről. Ide tartozik még, az adatok kiexportálása xls-be, pdf-be, vagy xlsx-be, ami manapság egy nagyon népszerű formátum. Ezen felül meglévő ügyfél adatainkat táblázatból, xls vagy xlsx -ből fel tudjuk dolgozni.

6.2.5 Kérdések, problémák

Fontosnak tartom, hogy az ügyfélnek legyen lehetősége visszajelzést küldenie arról, hogy hogyan is működik a szervezetünk. Meg kell adnunk kapcsolattartási információkat, és kérdőív formájában értékelhetővé kell tennünk cégünket bizonyos időközönként, ezzel meggyőződve partnerünk elégedettségéről.

6.2.6 Jogosultságkezelés

Mivel ez egy zárt rendszer, új felhasználó meghívása csak e-mailen keresztül történhet. Definiálnunk kell jogosultsági köröket vállalatunk felépítésétől függően. Felhasználókat törölni tudjon, vagy jogkört módosítani a megfelelő jogosultsággal rendelkező személy. Naplózza a rendszert használókat, hogy ki mikor volt bejelentkezve.

7. CRM rendszer megvalósítása

Itt tárgyalom az általam megírt web-alkalmazást, amely közel sem teljes. De egy jól működő rendszernek az alapjaival és fő funkcionalitásaival rendelkezik. Valamint az MVC modellnek köszönhetően átlátható, és egyszerűen bővíthető.

7.1 Autentikáció

Autentikáció, azaz egy felhasználó azonosítása, felhasználóneve, ami a mi esetünkben megegyezik az e-mail címével és jelszóval történik. Ezt a műveletet az Zend_Auth és a Doctrine használatával implementáltam.

Először is, a már meglévő App mappában létrehozunk egy Auth könyvtárat, amely majd az autentikáció alapját szolgáló adapternek ad helyet. Az azonosításra a Zend_Auth_Adapter interfészből implementált saját adaptert fogunk használni. Ennek az interfésznek egyetlen egy metódusa van az autentication(), ami egy Zend_Result objektummal kell hogy visszatérjen és képes legyen Zend_Auth_Adapter_Exception kivétel dobására, arra az esetre ha valamilyen hiba történne az azonosításkor. Az Auth csak egy szimpla osztály,

azonban mi szeretnénk valahol tárolni azt, hogy a felhasználó be van jelentkezve. Ennek az információnak a tárolására a Zend_Session-t alkalmazza, amely létrehoz egy session-t Zend_Auth névvel és a session változóban tárolja az értékeket, amire szüksége van.

Az adapter írását a konstruktor megírásával kezdjük, ami egy felhasználónevet és egy jelszót vár paraméterként. Eltároljuk ezt a két változót, mint az osztály attribútumait, mindkettő sztring. Ezután implementáljuk az interfész autentication() metódusát, de az azonosítás nem itt fog megtörténni, hanem a User modell osztályban. Ezért létre kell hoznunk egy új Model_User-t, aminek meghívjuk az autentikációra szolgáló függvényét. A User osztályban nagyon egyszerűen megírható a kellő metódus.

A Doctrine segítségével elindítunk egy lekérdezést felhasználónévre, ezt megtehetjük a findOneByUsername segítségével, ami egy univerzális eszköz. A findOneBy után bármilyen oszlopnevet írhatunk, a keretrendszer tudja, hogy aszerint kell a lekérdezéseit lefuttatnia. Ha van visszakapott User objektum és az adatbázisban lévő jelszó megegyezik a paraméterként átadott jelszóval, akkor sikeresen megtörtént az azonosítás és a függvény visszatérhet a User objektummal. Ha nem egyezik a jelszó, akkor létrehozunk egy kivételt. Konstansokat definiálunk a kivételek számára, ha nem találjuk meg a felhasználót, és arra is ha a jelszó nem megfelelő. Így majd később a program kódja könnyebben olvasható lesz. Tehát ha nem találunk felhasználót, akkor is dobunk egy kivételt a megfelelő konstanssal.

Majd visszatérhetünk az adapter írásához, amikor létrehozzuk az új User objektumot, azt egy try-catch ágban tesszük, ezek után tudjuk a kivételeket arra használni, hogy a felhasználónak a megfelelő hibaüzeneteket jelenítsük meg. Mivel több eredmény objektumra is szükségünk lesz, ezért azok példányosításra lehetőséget adunk egy külön erre a célra írt függvényben. Amelyben Zend_Auth_Result objektumokat példányosítunk, mely egy kódot, azonosítót és üzenetek tömbjét várja paraméterként. Ha minden rendben zajlik, akkor létrehozunk egy Zend_Auth_Result objektumot a megfelelő kóddal, ami esetünkben SUCCES, ez egy a Zend által erre a célra használatos konstansok közül. Ha hiba történne, összehasonlítjuk a már meglévő hiba konstansokkal és a megfelelő kóddal ellátott eredmény objektumokat hozzuk létre FAILURE_CREDENTIAL_INVALID kóddal, ami a hibás jelszót, illetve FAILURE_IDENTITY_NOT_FOUND kóddal, ha a felhasználó nem található. A

hibaüzenetek számára szintén konstansokat definiálunk, és ezeket adjuk át a létrehozandó objektum üzenet paramétereiként. Ezekkel a létrehozott eredményobjektumokkal tér vissza az autenticate() függvény.

Annyi dolgunk van még, hogy a nézetben megírjuk az autentikációhoz tartozó egyszerű űrlapot, amely két beviteli mezőből áll, egy a felhasználónévnek, egy pedig a jelszó számára. A kontrollerben példányosítjuk a megírt adaptert, majd meghívjuk az adapterre az Zend_Auth::getInstance()->authenticate($adapter) metódust. Ez visszatér egy eredmény objektummal. Ezután már a Zend_Auth::getInstance()->hasIdentity() függvénnyel meg tudjuk vizsgálni, hogy a felhasználó valóban az akinek állítja magát. Ha nem, kiírjuk a megfelelő hibaüzenetet. Annak érdekében, hogy illetéktelenül ne lehessen megtekinteni az alkalmazás oldalait, a layout-ban vizsgáljuk, hogy azonosítva van-e a felhasználó, ha nincs akkor nem rajzolódik ki az oldal. Helyette egy hibaüzenet talál a felhasználó, ami tartalmazza a bejelentkező oldalra mutató linket is. A kijelentkeztetést ugyan ilyen egyszerűen megtehetjük ->clearIdentity() metódussal, amely törli a megfelelő session változókat.

Sajnálatos módon van egy hibája ennek a megközelítésnek, azt feltételezi, hogy az adatbázisunk mindig működik. Ez egy adatbázisra épülő alkalmazásnál szinte nem is lehet másképp, hiszen a funkciók nagy része adatbázis nélkül egyébként sem működik.

7.2 Partnerek

A customer/index oldal fogadja elsőként a regisztrált felhasználót a bejelentkező képernyő után, miután sikeresen azonosította magát. Itt kereshet, illetve egyéb alapvető műveleteket tehet meg a felhasználó. Vagy a linkeknek köszönhetően az oldalsávból vagy a menüből eljuthat, az új ügyfél létrehozására kialakított oldalakra.

7.2.1 Keresés

A keresésnek két fajtája lehetséges, vagy egyszerű keresés, amikor a legyen szó akár cégről, akár személyről név szerinti keresés fog megvalósulni. Minden egyes billentyűleütés után, egy asszinkron kérés indul el a szerver felé, ekkor nem frissül az egész oldal, csupán a listázásra szolgáló rész.

12. ábra: Egyszerű keresés

Ennek megvalósítása jQuery-vel egyszerűbb, minta csupán JavaScriptet használtam volna, mivel a jQuery több beépített metódussal is rendelkezik AJAX műveletekkel kapcsolatban. Programomban a legegzaktabban módját használtam, ennek formája: $.post(url ami feldolgozza a kérést, [adat], [success(adat, státusz, XMLHttpRequest)], [típus]), Zend környezetben arra kell ügyelni, hogy amikor a kontrollerben megírjuk az action-t, amelyik feldolgozza a kérést, akkor tiltsuk le a layout-ot, illetve az oldal kirajzolását.

Az adatok cseréjénél JSON (JavaScript Object Notation)-t használtam, mely név/adat párosokkal dolgozik, ebben a környezetben asszociatív tömbként tudunk hivatkozni elemeire. Egy tag lenyomásával előhozhatjuk az összetett keresést szolgáló űrlapot, ekkor az egyszerű keresés formja eltűnik. Tetszés szerint váltogathatunk a két mód között, és mindig csak az egyik lesz látható.

13. ábra: Összetett keresés

A másik keresési fajta az összetett keresés, ahol megadhatunk város, megye, irányítószám, ország, telefonszám, email adatokat. Ezek begépelése után a 'Keres' gomb megnyomását követően indul el az asszinkron kérés, és a megfelelő action feldolgozza ezt.

Csak olyan sorok térnek vissza amelyek pontosan megfelelnek a megadott feltételeknek, míg egyszerű keresésnél a LIKE kulcsszó használatával az SQL lekérdezésben minden olyan sort vissza fog adni a Doctrine_Query, amely csak hasonlít a begépelt névre.

7.2.2 Listázás

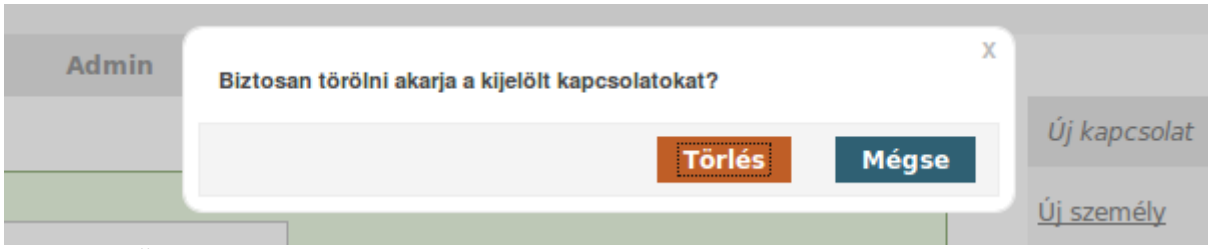
Az ügyfelek listázása egy egyszerű táblázat formájában történik, az alábbi felépítést követve az első oszlopban található egy kijelölő négyzet, melynek segítségével további műveleteket végezhetünk majd el a kijelölt ügyfeleken. Ezután következik az ügyfél avatarja, amellyel látványosan meg tudjuk különböztetni már első pillantásra is a partnereinket. Ezután pedig személy esetében a vezeték illetve keresztnév, cég esetében pedig a cég neve következik. Ezek mindegyike link, és a megfelelő azonosítót tartalmazva a hivatkozási címben el tudunk jutni az ügyfelet részletesen leíró oldalra, ahol szerkeszthetjük, illetve a tevékenységgel és jegyzetekkel kapcsolatos információkat adhatjuk meg. A név alatt helyezkedik el közvetlenül az e-mail cím, amely link formátumú. Ha rákattintunk lehetőségünk van az operációs rendszerünk vagy a böngészőnk által használt alapértelmezett levelezővel e-mailt írni. A táblázat fölött található egy legördülő űrlap elem, mellyel szűrhetjük felhasználóinkat. Ezekkel a feltételekkel beállíthatjuk, hogy az összes kapcsolatunkat, vagy csak a személyeket, illetve csak a cégeket szeretnénk látni. Ez is az oldal újratöltődése nélkül megy végbe.

14.ábra: Ügyfelek listázása

7.2.3 Törlés

Ügyfél törlésére két lehetőség is adott, vagy a partner részleteit bemutató oldalon a partner törlése gomb segítségével, vagy pedig az éppen bemutatott kezdőoldalon, a kijelölő négyzetek segítségével több felhasználót is tudunk egyszerre törölni. Az oldal alján található

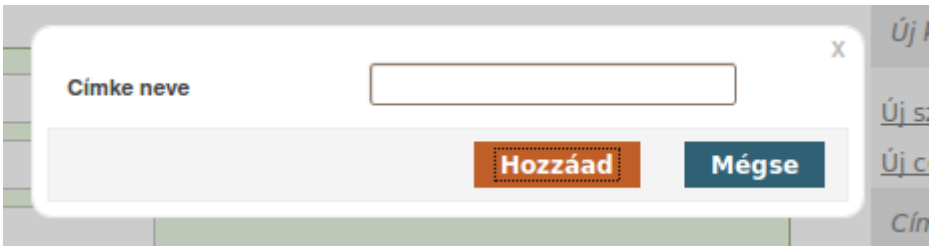
törlés elemre kattintva, az oldal háttere elhalványul és a megjelenik egy figyelmeztetés, hogy biztosan törölni szeretnénk-e a felhasználókat. Az igen gombot megnyomva a törlés megtörténik, törlődnek az ügyfelek mind az alaptáblából, mind pedig a kapcsolódó táblákból. Itt vigyáznunk kell, hogy ne sértsük meg a külső kulcsokra vonatkozó megszorításokat. A Doctrine unlink() parancsával, minden függőséget megszüntethetünk a paraméterként megadott táblával, vagy ha tömb formájában adjuk meg a paramétert, akkor táblákkal. A törlés is AJAX-osan történik, tehát valójában nem tűnnek el az oldalról a törölt tábla sorok, csak CSS által manipuláljuk a display tulajdonságot. Az oldal újra betöltősével nyilvánvalóan eltűnnek ezek a sorok.



15. ábra: Ügyfél törlése megerősítéssel

7.2.4 Címkék

A címkék nagyon fontos szerepet töltenek be az alkalmazásban, mivel egy CRM rendszerben lényeges, hogy az ügyfeleinket csoportosítani tudjuk. Ennek legjobb módja véleményem szerint az, ha minden egyes partnerhez tetszőleges számú címkét tudunk fűzni. Nyilvánvalóan egy címkéhez is több ügyfél tartozhat. A címkék hozzáadásának megkönnyítése végett erre a célra is használhatóak a kijelölőnégyzetek, a kijelölt ügyfelekhez hozzárendelődik a címke. A címke hozzárendelését az előtérben előtűnő ablakban tehetjük meg miután a megjelenő beviteli mezőbe begépeljük a címke nevét, amit szeretnénk használni és megnyomjuk a 'Hozzáad' gombot. A gomb megnyomása csak akkor lehetséges, ha tartalmaz valamilyen éréket az input mező.



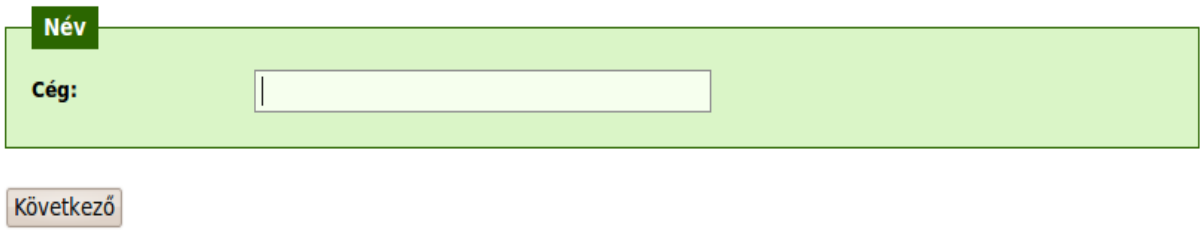
16. ábra: Címke hozzáadása ügyfélhez

A címkék listája megtalálható az oldalsávban ábécé sorrendben a könnyebb kereshetőség érdekében. Minden címke egy link amellyel eljuthatunk a címke oldalára, ahol a címke alá tartozó ügyfeleinket megtekinthetjük.

17. ábra: Címkék listája

7.2.5 Új ügyfél felvétele

Ez az oldal vagy a menüből vagy főoldalról érhető el. Két külön űrlapot hoztam létre, egyet a személyeknek egyet pedig a cégeknek. A különbség a két form között csupán annyi, hogy míg az elsőnél meg lehet adnunk kereszt és vezetéknevet, titulust, illetve cégnevet ahol az illető dolgozik, addig új cég felvételénél csak a cég neve adható meg, majd a cég részletei oldalon lesz lehetőség arra, hogy a céghez tartozó alkalmazottakat felvegyük.



18. ábra: Lapozható, kötelező űrlap elem

Az űrlap egy jQuery plugin segítségével több részre van osztva, így mintha egy varázslót futtatnánk, tudunk a következő lépésre ugrani, illetve vissza. Ezzel átláthatóbbá tévé az űrlap kitöltését. Ha egy űrlapelem nem érvényes inputot tartalmaz akkor nem engedélyezzük a továbblépést. Azokat a mezőket, amelyeknek kitöltése kötelező hátterét világoszöldre színeztem.

Mivel törekedtem arra, hogy a legtöbb kapcsolati információ megadható legyen, így lehetőség van minden elérhetőségből öt darabot megadni. Ha elkezdünk gépelni a beviteli mezőbe, megjelenik egy 'Hozzáadás' és egy inaktív 'Eltávolítás' gomb, amelyek arra szolgálnak, hogy lenyíljon egy új elem, amibe szintén be tudjuk gépelni a megfelelő információt. Ha már vettünk fel új elemet, az 'Eltávolít' gomb aktívvá válik és lehetőség van az utolsó hozzáadott beviteli mező eltávolítására. A hozzáadás a jQuery clone() metódusa által lett megvalósítva. Ami klónoz egy kijelölt elemet, esetünkben a beviteli mezőt, minden tulajdonságával, és viselkedésével együtt.

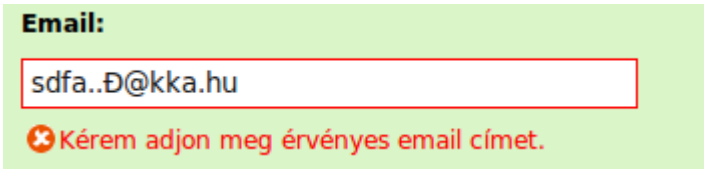
19. ábra: Klónozható beviteli mező

Itt tartom a legfontosabbnak megemlíteni a validációt, mivel az alkalmazás ezen részén fordul elő a legjelentősebben. Egyébként a rendszer egészét tekintve ugyanaz jellemző az adatok ellenőrzésére. Mindig csak kliens oldalon történik a vizsgálat. Szerver oldalon a Doctrine véd az SQL inject támadásokkal szemben, de egyéb validáció nem történik. A kliens oldalon a jQuery biztosítja a validációt. Csupán egy jQuery plugin egy fájlból álló JavaScriptjét kell hozzáadnunk az oldalhoz, és az kívánt CSS fájlt, és máris használhatjuk a validate() metódust. Mely paramétereként az űrlapunk valamilyen azonosítóját várja. Mindez igen széles rétegét kínálja a validációnak. Pár általam is használt metódusa :

- required() : ha mező kitöltése kötelező
- minlength(minimális hossz) : a minimális hosszt el kell érnie a bevitt adatnak
- maxlength(maximális hossz) : a maximális hosszt nem haladhatja meg a bevitt adat

- email() : a bevitt adatnak érvényes e-mail címnek kell lennie
- url() : a bevitt adatnak érvényes webcímnek kell lennie
- number() : a bevitt adat decimális egész kell, hogy legyen
- digits() : csak szám lehet a mezőben
- equalTo(másik elem) : a két elem értékének meg kell egyeznie

Ezeken kívül még több, mint egy tucat beépített metódussal rendelkezik, de ha nem lenne elég ez számunkra, akkor a rules() paranccsal létrehozhatunk saját szabályokat. Amelyekhez saját hibaüzeneteket is írhatunk. A működéséhez elég annyi is, ha a beviteli mezőinknek a fenti metódusok neveit adjuk meg osztálynévként. Ekkor is tudni fogja a szkript, hogy validáció szükséges.



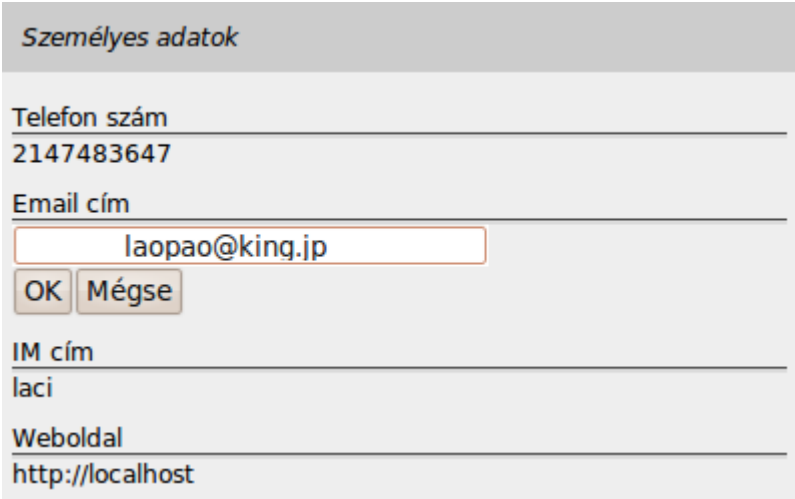
20. ábra: Validáció

7.2.6 Személyes adatok

Az ügyfél részletek oldalon megtalálhatjuk a partnerünk személyes adatait, amelyeket korábban felvettünk, csak azok a kapcsolattípusok látszanak, amelyekhez adtunk meg értéket. Ha belekattintunk bármely mezőbe, akkor az szerkeszthetővé válik. Az oldal teljes frissítése nélkül, tehát azonnal láthatóvá válik a változás. A JavaScript hozzáadását, egy helper osztállyal valósítottam meg, melynek addOnload() metódusa betölti a szkriptet az oldal fejrészébe.

Amennyiben a kapcsolatunk rendelkezik twitter azonosítóval, akkor a a legutóbbi öt tweet-jét láthatjuk az oldalsávban és egy link által meg tudjuk látogatni a hozzá tartozó oldalt is. Mivel több azonosító is megadható, legördülő listából választhatjuk ki a számunkra

megfelelőt.



21. ábra: Személyes adatok szerkeszthető mezővel

7.3 Emlékeztető

Emlékeztetők eseményekről, dátumokról. Amely azért fontos, hogy ne feledkezzünk meg semmiről sem az ügyféllel kapcsolatban. Akár találkozó, megbeszélés, vagy születésnap fog következni. Ez az oldal is elérhető a menüből, vagy az ügyfélhez tartozó részletek oldalon is megtekinthetjük a partnerhez kapcsolódó eseményeket, dátumokat

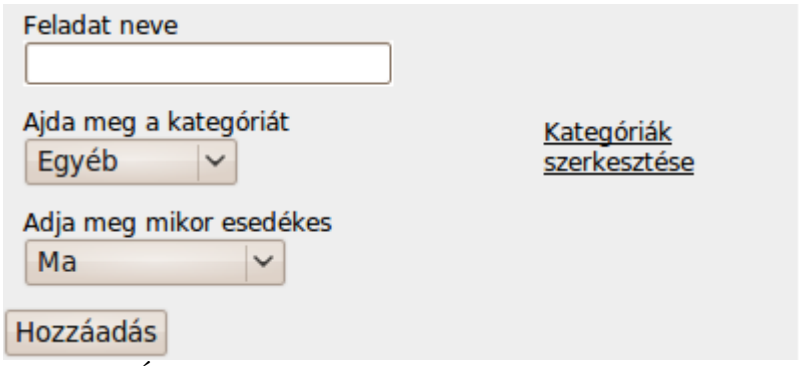
7.3.1 Események

Az ügyfelünkkel kapcsolatos eseményeket itt adhatjuk meg. Alapállapotban csak az események felsorolása látszik egy kijelölőnégyzettel, amit a kategória követ, majd pedig az esemény neve. A kijelölő négyzetekkel tudjuk törölni a már teljesített feladatot. Ha kipipáljuk eltűnik a felsorolásból, és az adatbázisból.

Az 'Új esemény' feliratú gombra kattintva megjelenik egy űrlap, melyben megadhatunk nevet, kategóriát, és teljesítési dátumot. A keltezés maga nem jelenik meg a kilistázáskor, ellenben besorolásra kerül aszerint, hogy a naptárban hol helyezkedik el. Megkülönböztettem a mai, holnapi, ezen a héten, jövő héten esedékes teendőket, illetve külön csoportot hoztam létre a később esedékeseknek és a lejártaknak.

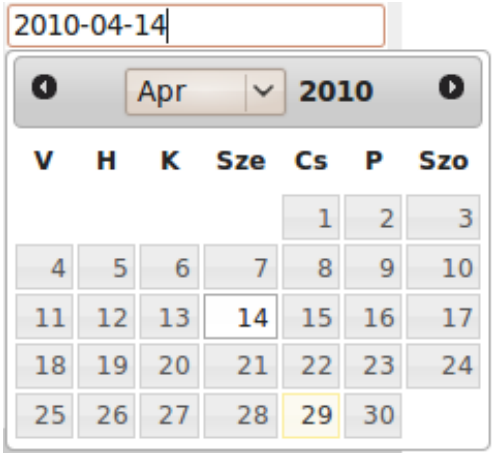
22. ábra: Események listája

A kategóriáknál választhatunk az alapértelmezettek közül, amelyeket az alkalmazás az adatbázisból olvas ki, vagy bővíthetjük azt a legördülő mezőből az egyéb opciót választva. Ekkor megjelenik egy új form elem a képernyő közepén, melybe begépelhetjük a kívánt kategóriát. Ez azonnal meg is jelenik, az egész oldal frissítése nélkül. Lehetőségünk van a kategóriák további szerkesztésére is, de erre majd a dolgozat későbbi részében térek ki részletesebben.



23. ábra: Új esemény hozzáadása

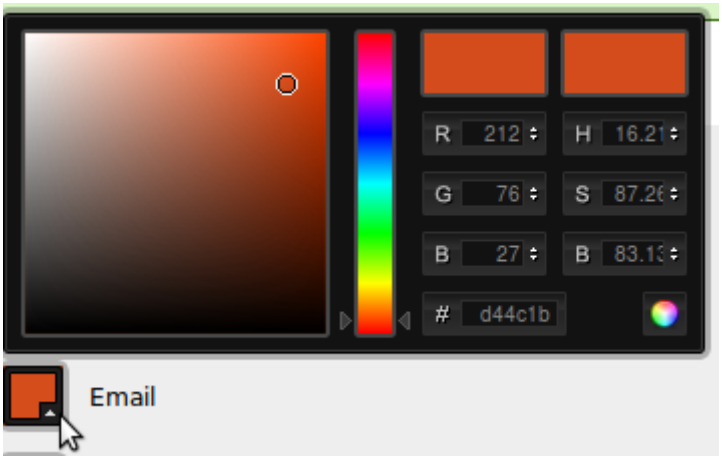
A dátum megadásánál a már említett naptári időintervallumok adhatóak meg. Illetve ha 'Egyéb időpont' nevű lehetőséget választjuk, akkor a legördülő lista mellett egy beviteli mező jelenik meg, melybe belekattintva egy dátumválasztó tűnik elő. Ezzel könnyedén szabványosan, az alkalmazás számára jól feldolgozható módon adhatjuk meg a dátumot. Ez a időpont is besorolásra kerül majd amikor hozzáfűződik az oldalhoz.



24. ábra:Egyéb időpont kiválasztása

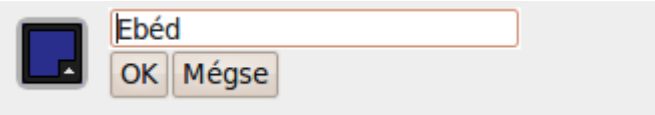
7.3.2 Kategóriák szerkesztése

A 'Kategóriák szerkesztése' nevű link megnyomásával juthatunk az oldalra. Lényegesnek tartottam, hogy lehessen színeket hozzárendelni egy-egy kategóriához, ezzel jelezve annak fontosságát. Ezen az oldalon tudunk új színt beállítani egy színkiválasztó alkalmazás segítségével, melynek köszönhető bármilyen színt kikeverhetünk. Alapértelmezetten a kategóriákhoz fekete színt rendeltem.



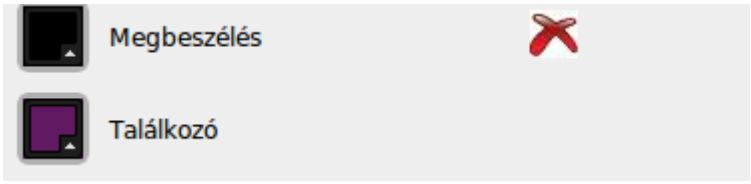
25. ábra: Szín kiválasztása kategóriához

A kategória nevének megváltoztatását is megtehetjük, csak rá kell kattintanunk arra, amelyiket szerkeszteni szeretnénk és már meg is változtathatjuk annak értékét. Az adatbázisban is azonnal végbemegy a változás az asszinkron kérésnek köszönhetően. Az adatbázisban ez érték hexadecimális számként tárolódik, ennélfogva a tagek háttérszínének a beállítása nagyon leegyszerűsödik.



27. ábra: Kategória nevének megváltoztatása

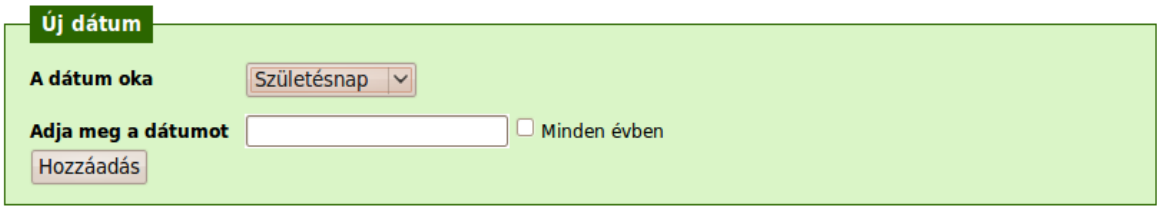
A törléshez ebben az esetben is megerősítő kérdés ugrik fel, hogy biztosak vagyunk e benne, hogy töröljük a kategóriát. Csak egyesesével tudjuk eltávolítani az elemeket és csak azokat a kategóriákat, amelyekhez nem tartozik feladat. Egyébként a törlés nem engedélyezett. Megadhatunk új kategóriát is, ekkor csak egy névre van szükség, és az űrlap elküldése után már használhatjuk is az új típust.



28. ábra: Kategória törlése

7.3.3 Dátumok

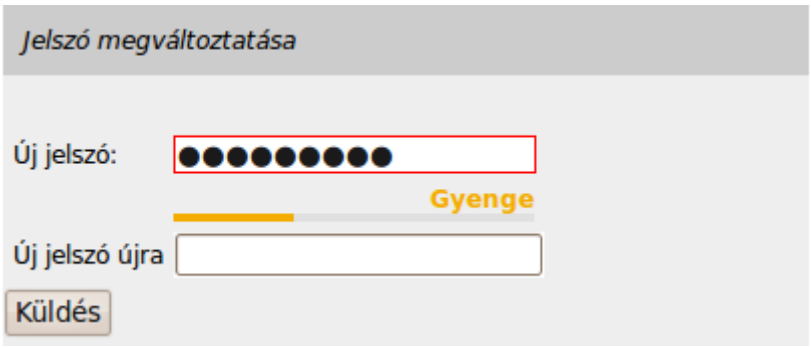
Az ügyféllel kapcsolatos dátumok listáját találhatjuk az oldalsávon, egy 'Új dátum' feliratú gombbal, amellyel eljuthatunk az új dátum elkészítéséért felelős oldalra. Az oldalsávon csupán az időpontok találhatóak és az időpont okának megnevezése, amit a törlés gomb követ a már prezentált megerősítést kérő törlés funkcionalitással. Itt is csak egyesével a törlést szolgáló gomb sorát tudjuk eltávolítani. A dátum létrehozása oldalon, az eseményeknél már látott dátumválasztó felülettel tudjuk megadni a pontos időpontot, előtte egy beviteli mezőbe az okát, majd egy jelölőnégyzet található, amit ha bepipálunk, az adatbázisban igaz értéket kap a minden évben tulajdonság. Ez a rendszer további fejlesztésénél jöhet jól, ezzel a kiegészítéssel a felhasználó például minden évben megkaphatja a figyelmeztetést a közelgő alkalomról. A dátum létrehozása oldalon is megtalálható ugyanabban a formában az időpontok listája mint az oldalsávon.



29. ábra: Új dátum megadása

7.4 Adminisztráció

Az adminisztrációs oldalt jelen pillanatban mindenki tudja használni, mivel a jogkörök, és azok kezelése még nincs definiálva, ezért nem implementáltam más felhasználók törlésének lehetőségét, hiszen így bárki tudna törölni bárkit. Viszont megvalósítottam más felhasználók meghívásának lehetőségét, mivel ez egy zárt rendszer és csak ezen a módon lehet azonosítót kapni. A Zend_Mail segítségével küldöm ki a levelet egy SMTP szerveren keresztül, melyet a application.ini-ben konfiguráltam a Zend által biztosított e-mail-el kapcsolatos beépített erőforrások módosításával. A levél tartalmazza a felhasználónevet, ami a meghívott e-mail címe, illetve egy tíz jegyű generált jelszót, ami tartalmaz 0-9-ig számokat és az angol ábécé kis és nagybetűit. A jelszó az adatbázisba md5(base64_encode()) kódolás után kerül be. A programot használó személy által meghívottak listája is megtalálható ezen az oldalon, a meghívott e-mail címével, amelyre kattintva közvetlenül írhatunk neki, illetve tartalmazza a meghívás dátumát másodpercre pontosan. Ez egy kliensoldalon rendezhető táblázatba kerül, bármely oszlopa szerint lehet rendezni a benne található adatokat.



30. ábra: Jelszó megváltoztatása

A felhasználó a saját jelszavát megváltoztathatja, az oldalsávon található menüpontból,

melybe csak be kell gépelnie az új jelszót. Ez a jelszó validáción megy keresztül, hogy elég erős e. Ha igen akkor a felhasználónak meg kell ismételnie a jelszót. Amennyiben megegyezik a két beviteli mező értéke, akkor a küldés gomb megnyomása után, már érvénybe is lép az új azonosító. A jelszóról a felhasználó e-mailben is kap értesítést.

7.5 Az oldal felépítése

Az oldal három fő szerkezeti egységre tagolható. A fejrész, ahová a cím, illetve a menü került. A tartalommal kapcsolatos információk az oldal közepén helyezkednek el. Ez a rész is két részre van osztva. Az egyik részén az űrlapelemek halmaza, a másik részén általában a lekérdezések eredményei kapnak helyet. Az oldal szélén található egy sáv, amely az fő tartalomhoz fűződő egyéb információkat, illetve funkciókat tartalmazzák.

A fő CSS-t a main.css fájl tartalmazza, melyet a layout-ban adok az oldalhoz. A layoutban találhatóak az oldal felépítését meghatározó tagok is. Próbáltam elhatárolódni az elavult táblás megoldástól az alkalmazás egészében, mindenhol tageket, illetve stílus fájlt használni.

A navigáció történhet a menüből, mely egy jQuery-vel manipulált lista, így bővítése nagyon egyszerű, jól átlátható, és könnyen kezelhető. De van lehetőség az oldalsávban elhelyezett linkekkel való navigálásra is.

31. ábra: Menü felépítése

8. Az adatbázis felépítése

Elég sok időt töltöttem az adatbázis megtervezésével, hogy a későbbiekben a szoftver fejlesztése közben már ne kelljen nagyobb módosításokat véghez vinnem. Mint már említettem ez egy adatbázis orientált alkalmazás. Ezért nagyon fontos, hogy a futó adatbázis

stabil legyen, illetve a nap 24 órájában képes legyen üzemelni. Választásom a MySQL adatbázist kezelőre esett InnoDB motorral.

A tervezést grafikus felülettel rendelkező DBDesigner programmal vittem véghez, mely XML formátumban menti le a létrehozott sémát, de lehetőséget ad, hogy SQL szkript formájában kiexportáljuk az általunk létrehozott táblákat, kapcsolatokat. Lehetőséget ad régiók kialakítására ezáltal átláthatóbbá téve a tervezési felületet.

Az elkövetkezőnkben régiókra bontva szeretném elemezni az általam készített tervet. Igaz az egyes régiók közötti kapcsolatok miatt, néha muszáj lesz egyéb régió tábláiról is beszélnem. A teljes terv a dolgozat függelék részében lesz megtalálható.

Minden tábla rendelkezik elsődleges kulccsal, ezek mindig INTEGER(10) adattípusúak, előjel nélküliek, nem vehetnek fel NULL értéket, illetve automatikusan növekszenek.

8.1 Felhasználó régió

Ebben a régióban a felhasználóval és a rendszer adminisztrációval kapcsolatos táblák találhatóak.

- User: A felhasználó adatai találhatóak itt, amelyek legfőképp az autentikációnál lényegesek.
    - user_name: a felhasználó neve, az általam írt alkalmazás esetében ez megegyezik a felhasználó e-mail címével.
    - password: a felhasználó jelszava, amellyel be tud jelentkezni
    - last login: naplózzuk, hogy mikor lépett be utoljára az alkalmazásunkat használó személy
    - business_idbussines: külső kulcs
    - usergroup_idusergroup : külső kulcs
- Usergroup: különböző csoportokba rendezhetjük a felhasználóinkat. Ezzel a táblával a

célom az ACL megalapozása volt. Csupán a csoport nevét lehet megadni.

- Invite: Mivel rendszerünk zárt, ezért a regisztráció csak e-mailen keresztül meghívás által lehetséges.
    - email : a meghívandó e-mail címe
    - invite_date: a meghívás időpontja
    - user_userid: külső kulcs, ezzel nyomon tudjuk követni, hogy melyik felhasználó, mely másik felhasználót hívta meg
- Business: Mivel több cég is használhatja a rendszerünket, minden regisztrált vállalkozás kap egy külön azonosítót. Ez a tábla kapcsolódik a többi főtáblához, így a lekérdezéseknél, mindig tudjuk, hogy épp mely cég adataira van szükség. Tartalmaz még egy hash kódot is, amelyre azért van szükség, hogy a cég ne legyen elérhető másik cég alkalmazottja által. Ha valamiért GET metódussal szeretnénk valamit elküldeni, akkor ezt a hash kódot használhatjuk.
- Project: egy cégen belül több projekt is folyhat, ha külön szeretnénk kezelni az ügyfeleket projektenként, de véleményem szerint ez nem szükséges. Csupán azért szerepel ez a tábla, hogy minden lehetőségnek eleget tegyen a rendszer.

8.2 Kapcsolat régió

Itt helyezkedik el az alkalmazás szíve, mivel a CRM rendszer lelke az ügyfél, és mivel lényegesnek tekintettem, hogy a kapcsolattartás minden formájában részt vegyen az adatbázisban. Ez a régió tölti ki az adatbázisterv nagy részét.

- Contact: Ebben a táblában adhatóak meg a partnerünk adatai, legyen szó személyről vagy akár cégről. Fontos, hogy itt csak a legalapvetőbb adatok szerepelnek.
    - first_name : kereszt név
    - last_name : vezeték név
    - title : bármilyen titulus

    - companyname : ha személyről van szó, és valamilyen vállalatnál dolgozik akkor annak neve ebbe a rekordba kerül
    - bio : bármilyen egyéb információ az ügyfélről
    - avatarurl : megadható avatar is, ennek csak az elérési útját tároljuk
    - iscompany : 0 vagy 1 lehet az értéke attól függően, hogy cég vagy személy e a kapcsolat
    - companycontact : ha személyről van szó és létezik az adatbázisunkban hozzá cég, akkor annak azonosítóját itt megadhatjuk

- Kapcsolódó táblák: hat kapcsolódó tábla van, amelyek arra szolgálnak, hogy különböző kapcsolattartási adatokat tudjunk megadni. Mindegyik kapcsolattípusnak adhatunk nevet, és megadhatjuk a típusát. Ezzel is tovább bontva az osztályozást. Ezekben a táblákban megadhatunk telefonszámot, részletes címet, e-mail címet, weboldalt, azonnali üzenetküldésre alkalmas szoftver azonosítóját, twitter felhasználó nevet.
- Contact_Tag: fontos a kapcsolataink csoportosítása, ezt címkékkel valósítottam meg. Egy címke tartozhat több kapcsolathoz és fordítva, tehát n:m kapcsolat áll fenn a két tábla között. Emiatt szükség van egy köztes táblára is melynek funkcióját a contact_has_contact_tag tábla látja el.
- Contact_Note : jegyzeteket készíthetünk felhasználókhoz. De ebben a táblában kapnak helyet az ügyekhez kapcsolódó jegyzetek is. Kapcsolatban áll még a dokumentum táblával is, mivel itt van lehetőség arra, hogy dokumentumokat csatoljunk egy egy jegyzethez.
    - subject : a jegyzet tárgya
    - contact_note : ide írhatjuk az ügyféllel vagy üggyel kapcsolatos információkat
    - notedate : a jegyzet elkészültének időpontja
- Note_Log: Ha a Contact_Note táblából törlünk egy sort, ide kerül be az. Tehát ez a

tábla másolata az előbbinek. Azért van rá szükség, hogy nyomon követhessük azokat a partnerrel kapcsolatos jegyzeteket, amelyek a múltban történtek.

8.3 Tevékenység régió

Az ügyféllel kapcsolatos bármilyen tevékenység tárolásra céljából jött létre ez a régió. Legyen szó eseményről, emlékeztetőről, vagy dátumról.

- Task: Eseményeket rögzíthetünk. Kapcsolódik a Contact táblához, és az ügyekkel foglalkozó táblához is.
    - task_name : a feladat neve
    - expiration : annak időpontja, hogy meddig kell teljesíteni a feladatot
- Task_type : Egy eseményhez tartozhat egy kategória, de több esemény is tartozhat egy kategóriához.
    - task_type_name : a kategória neve
    - taskcolor : a kategóriához tartozó színkód, hexadecimális számjegyekkel tárolva
- Dates : Az ügyféllel kapcsolatos fontos időpontok tárolása a célja, az időpontot, és az okot adhatjuk meg, illetve, hogy minden évben bekövetkezik e az esemény.

8.4 Dokumentum régió

Jegyzetekhez tartozó dokumentumok kezelésének lehetőségét biztosítja.

- Document: A dokumentummal kapcsolatos információkat tartalmazza
    - document_name : a fájl neve, amit fel töltöttünk
    - document_path : az elérési útvonala a fájlnak
    - document_type : a fájl típúsa
- Document_uploader: A Document táblához kapcsolódik, a fájl feltöltésének időpontja,

illetve a fájl feltöltőjét tárolhatjuk benne.

8.5 Ügyek régió

Itt találhatóak az ügyekkel kapcsolatos tábláink. Jelenleg itt egy tábla található, mivel a többi régióban lévő tábla ellátja minden olyan kapcsolótáblával, ami egy ügyhöz tartozhat. Az ügyfelekkel, a jegyzetek, és a tevékenység táblákkal áll relációban. Tehát egy ügy alá tartozhat több ügyfél is, illetve egy ügyhöz tartozhat több tevékenység, illetve jegyzet.

8.6 Egyéb régió

Itt az adatbázishoz kapcsolódó egyéb táblák találhatóak, amelyek nincsenek szoros összefüggésben egyéb régiókkal. Ilyen tábla a country, amely országokat tartalmaz. Az ügyfél létrehozásánál van haszna, ennek a táblának a soraiból jön létre a megfelelő űrlapelem.

9. Összefoglalás

Az alkalmazás fejlesztése során, új technológiákkal ismerkedtem meg és számos tapasztalattal gazdagodtam. Úgy érzem a fejlesztés minden fázisban el tudtam mélyülni, legyen szó az adatbázis megtervezéséről, vagy a felületi elemek elrendezésről.

Célomat elértem, sikeresen megtörtént a Doctrine Zend-be való integrálása, az MVC minta kialakítása. És a jQuery könyvtár használata minden nézethez fűződő egyéb funkció kapcsán, amik a felhasználóbarát kialakítást lehetővé teszik. Igaz az alkalmazás közel sem teljes, de azokkal a funkciókkal rendelkezik, melyek segítségével azok a technológiák, amelyeket alkalmazni kívántam jól szemléltethetőek. A dolgozat elején megtettem az elméleti alapozást, amelynek működését a későbbiekben ábrákkal személtettem.

Úgy érzem, hogy nem volt egy könnyű témaválasztás a rendszer összetettsége miatt, számos kihívást tartogatott, és még tartogat is ez a téma. Folyamatosan bővíthető az alkalmazás a felhasználói, illetve üzleti igényeknek megfelelően. Az MVC minta által a kód átlátható, és ezáltal a további fejlesztése a programnak akár csapatban, mások bevonásával is történhet.

Itt szeretném megragadni az alkalmat, hogy megköszönjem témavezetőmnek Kollár Lajosnak a segítő ötleteket, és a diplomamunkám elkészítése közben felmerülő problémákban való készséges segítség nyújtást.

10. Irodalomjegyzék

Zend Framework [Online]. - http://framework.zend.com/

Zend Framework tutoriálok [Online]. - http://www.zendcasts.com/

jQuery [Online]. - http://docs.jquery.com/How_jQuery_Works

jQuery UI [Online]. - http://jqueryui.com/home

Doctrine [Online]. - http://www.doctrine-project.org/

Doctrine és Zend Framework [Online]. - http://dev.juokaz.com/php/zend-framework-and-doctrine-part-1

CRM tanulmány [Online]. - http://www.hatekonysag.hu/crm_customer_relationship_management.php

CRM tanulmány [Online]. - http://www.pwc.com/hu/hu/services/customer-relationship-management-crm.jhtml

CRM rendszer [Online]. - http://en.wikipedia.org/wiki/Customer_relationship_management

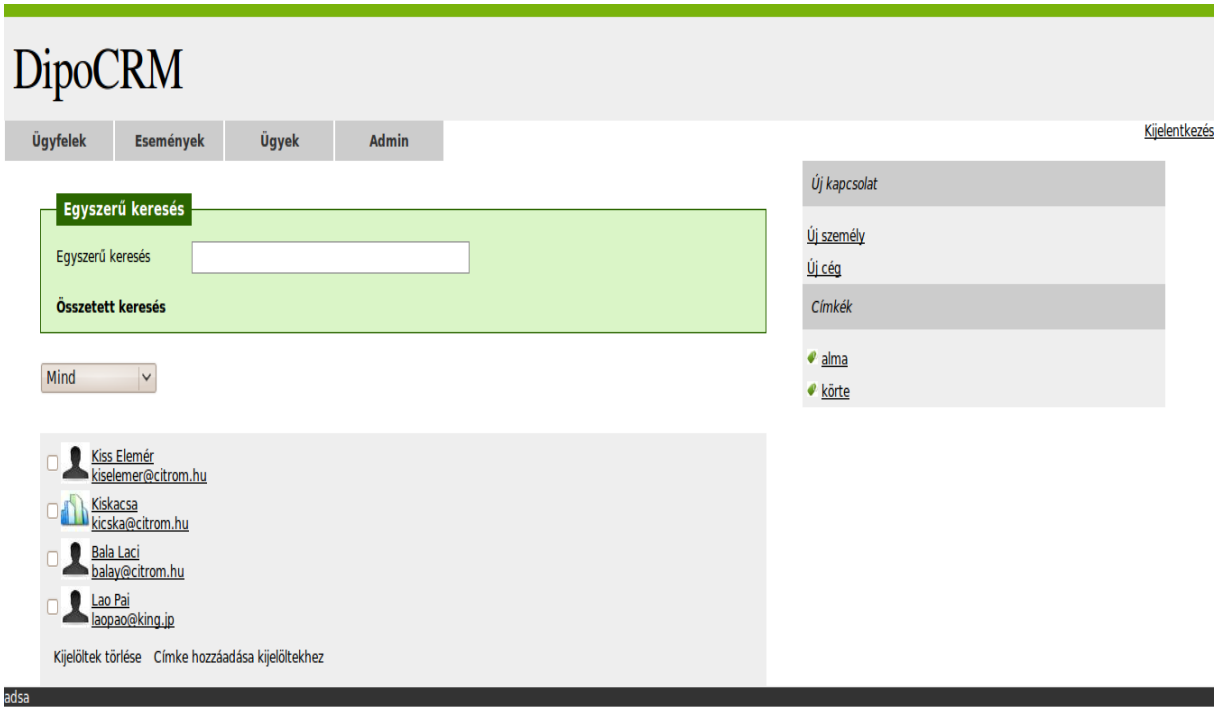
CRM rendszerek implementálása [Tanulmány] / szerző Szűcs István, Bátori Zolán - MEB 2008 – 6th International Conference on Management, Enterprise and Benchmarking

Learning jQuery 1.3 [Könyv] /szerző Jonathan Chaffer, Karl Swedberg . - 2009

Birmingham, B27 6PA, UK.

ZEND Framework in Action [Könyv] / szerző Rob Allen, Nick Lo. - 2007

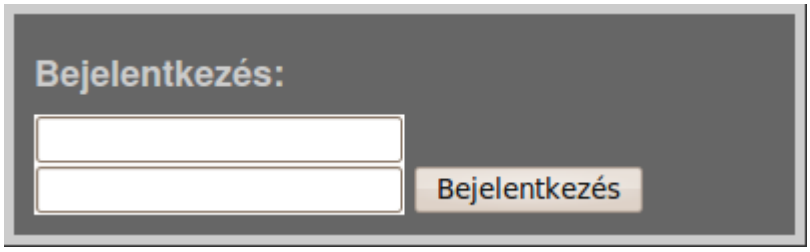
11. Függelék



32. ábra: Teljes oldal felépítése

Email	Meghívás dátuma
vadkocka@gmail.com	2010-04-29 17:02:59
lazlobalogh@gmail.com	2010-04-29 17:17:23
kicsember@gmail.com	2010-04-29 20:24:34

33. ábra: Rendezhető táblázat



34. ábra: Bejelentkező űrlap

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
phpSettings.date.timezone = "Europe/Budapest"
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory = APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 1
resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/"
resources.view[] =
autoloaderNamespaces[] = "Doctrine"
autoloaderNamespaces[] = "App"

; ---
; Email
; ---
resources.mail.transport.type = smtp
resources.mail.transport.host = "smtp.gmail.com"
resources.mail.transport.auth = login
resources.mail.transport.username = "vadkocka@googlemail.com"
resources.mail.transport.password = "titok"

resources.mail.defaultFrom.email = "vadkocka@googlemail.com"
resources.mail.defaultFrom.name = "dipoCrm"

; ---
; Database
; ---
doctrine.dsn                =  "mysql://root:33333@localhost/dipo_crm"
doctrine.data_fixtures_path = APPLICATION_PATH "/configs/data/fixtures"
doctrine.sql_path           = APPLICATION_PATH "/configs/data/sql"
doctrine.migrations_path    = APPLICATION_PATH "/configs/migrations"
doctrine.yaml_schema_path   = APPLICATION_PATH "/configs/schema.yml"
doctrine.models_path        = APPLICATION_PATH "/models"

```

Kód részlet 1: Application.ini

```

doctrine.generate_models_options.pearStyle = true
doctrine.generate_models_options.generateTableClasses = false
doctrine.generate_models_options.generateBaseClasses = true
doctrine.generate_models_options.baseClassPrefix = "Base_"
doctrine.generate_models_options.baseClassesDirectory = null
doctrine.generate_models_options.classPrefixFiles = false
doctrine.generate_models_options.classPrefix = "Model_"

; Conservative Model Loading:
doctrine.model_autoloading = 1

[doctrineCLI : production]
; Aggressive Model Loading
doctrine.model_autoloading = 2

[staging : production]

```

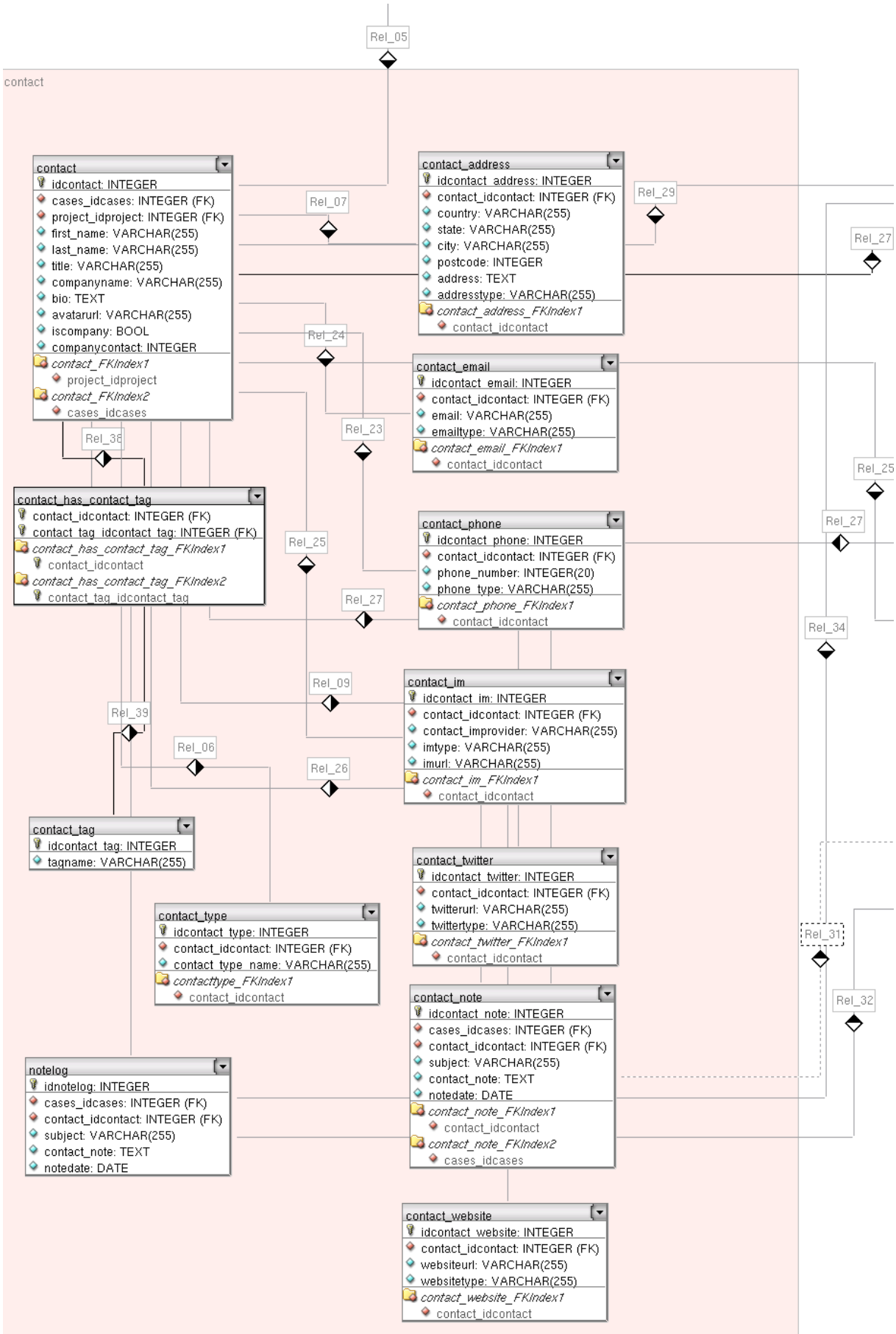
Kód részlet 2: Application.ini

```
[testing : production]
phpSettings.display_startup_errors  = 1
phpSettings.display_errors          = 1
doctrine.dsn                        =  "mysql://root:33333@localhost/dipo_crmtest"


[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1

```

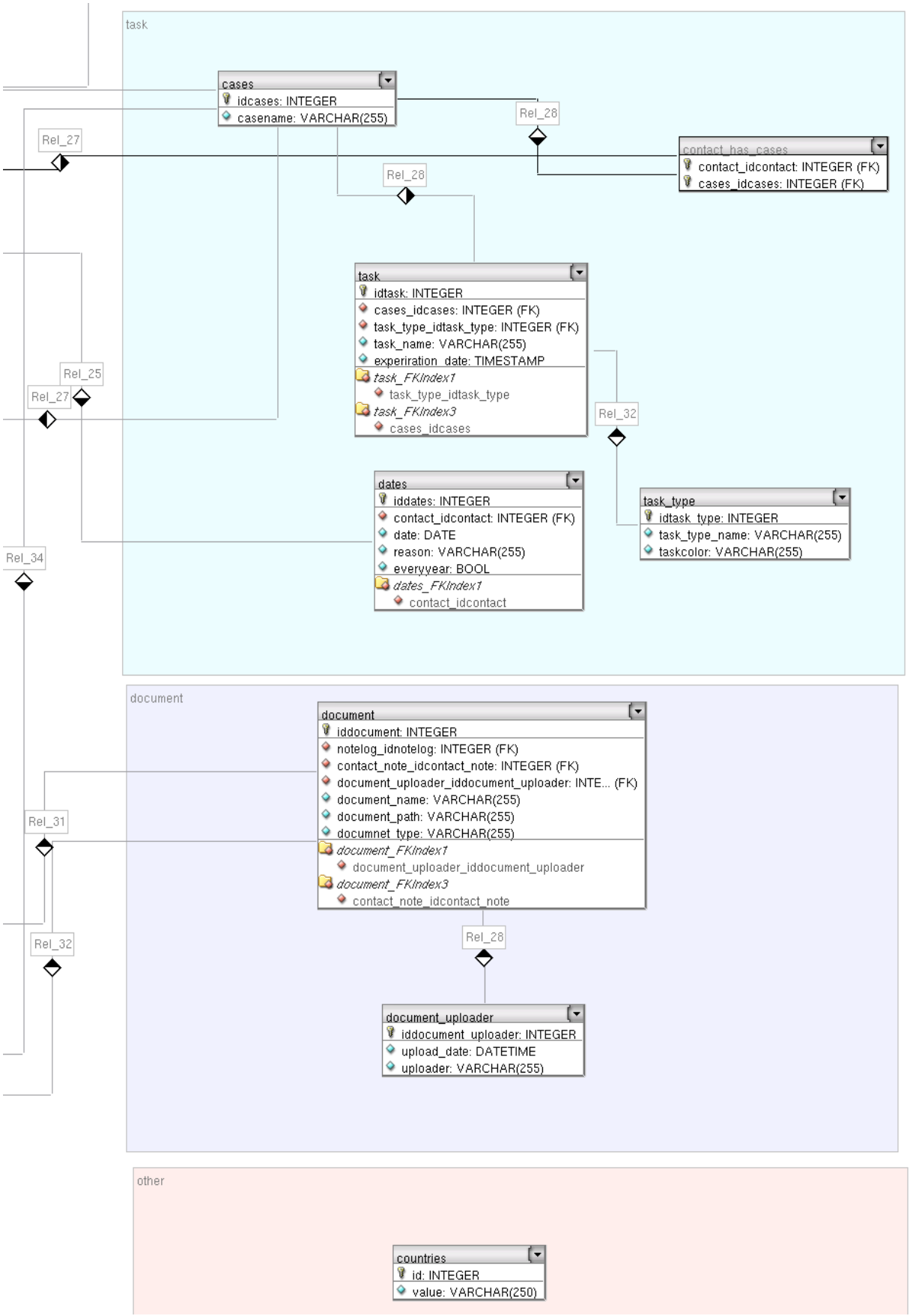
Kód részlet 3: Application.ini

```
Business:
  connection: doctrine
  tableName: business
  columns:
    idbusiness:
      type: integer(4)
      fixed: false
      unsigned: true
      primary: true
      autoincrement: true
    business_name:
      type: string(255)
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
    hash:
      type: string(255)
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
  relations:
    Project:
      local: idbusiness
      foreign: business_idbusiness
      type: many
    User:
      local: idbusiness
      foreign: business_idbusiness
      type: many
Cases:
  connection: doctrine
  tableName: cases
  columns:
```

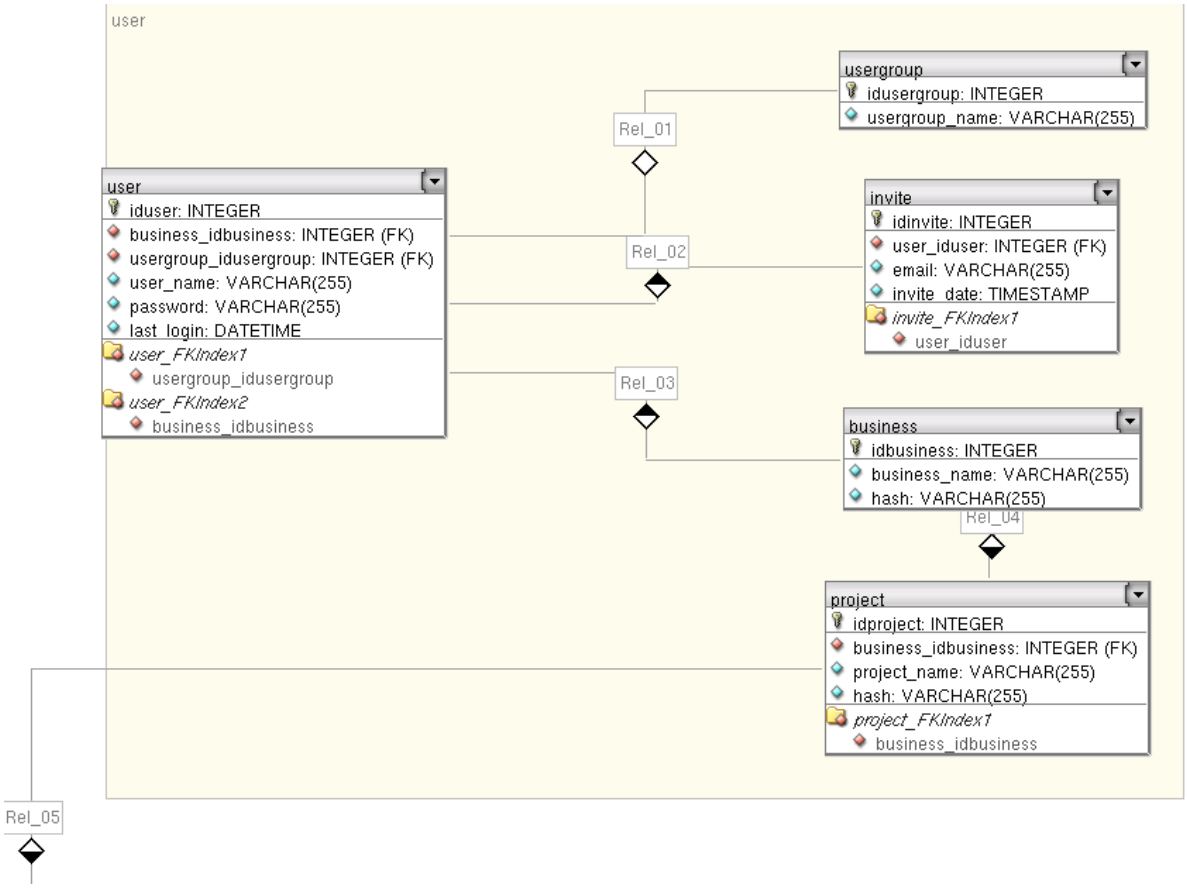
Kód részlet 4: YAML fájl részlet



35. ábra: Adatbázis részlet 1.



36. ábra: Adatbázis részlet 2.



37.ábra: Adatbázis részlet 3.