Budapesti Műszaki Szakképzési Centrum
Neumann János Informatikai Technikum
Szakképesítés neve: Szoftverfejlesztő
OKJ száma: 54 213 05

Záródolgozat

Helperek

Rostagni Csaba
konzulens

Melke Dániel
14. RSZE

Budapest, 2021.

Tartalomjegyzék

1. Helperek

1.1. A Helperek egy webalkalmazás, mely az alap webfejlesztés elemei (HTML, CSS) mellett PHP és Javascript felhasználásával jött létre.

Az oldal célja elsősorban a járvány idején, a veszélyeztetettebb helyzetben lévőknek egy platformot nyújtani, ahol segítséget kérhetnek a napi/heti teendők lebonyolításához. Elsődlegesen a bevásárlásban besegítést tennénk előtérbe, például az idősebb korosztálynak, hogy ne kelljen kitenni magukat a vírus veszélyeinek, minél kisebbre csökkentve az esélyét a találkozásnak/érintkezésnek.

Egy gyors regisztrációt követően a felhasználó láthatja az aktuális kéréseket, vagy "posztokat". Létre is hozhat újat, vagy a már saját meglévő posztjait tudja szerkeszteni, illetve törölni. Ezekre lehet "reagálni", ha szeretne valaki segíteni, lehetőség van kapcsolatfelvételre. Segítség-felajánló posztokat is érdemes létrehozni!

A kérések/segítségek lebonyolítását, menetét a felhasználókra bízzuk, ezért van lehetőség kapcsolat felvételre. Eldönthetik hogy szeretnének kommunikálni a későbbiekben, ha igény van rá.

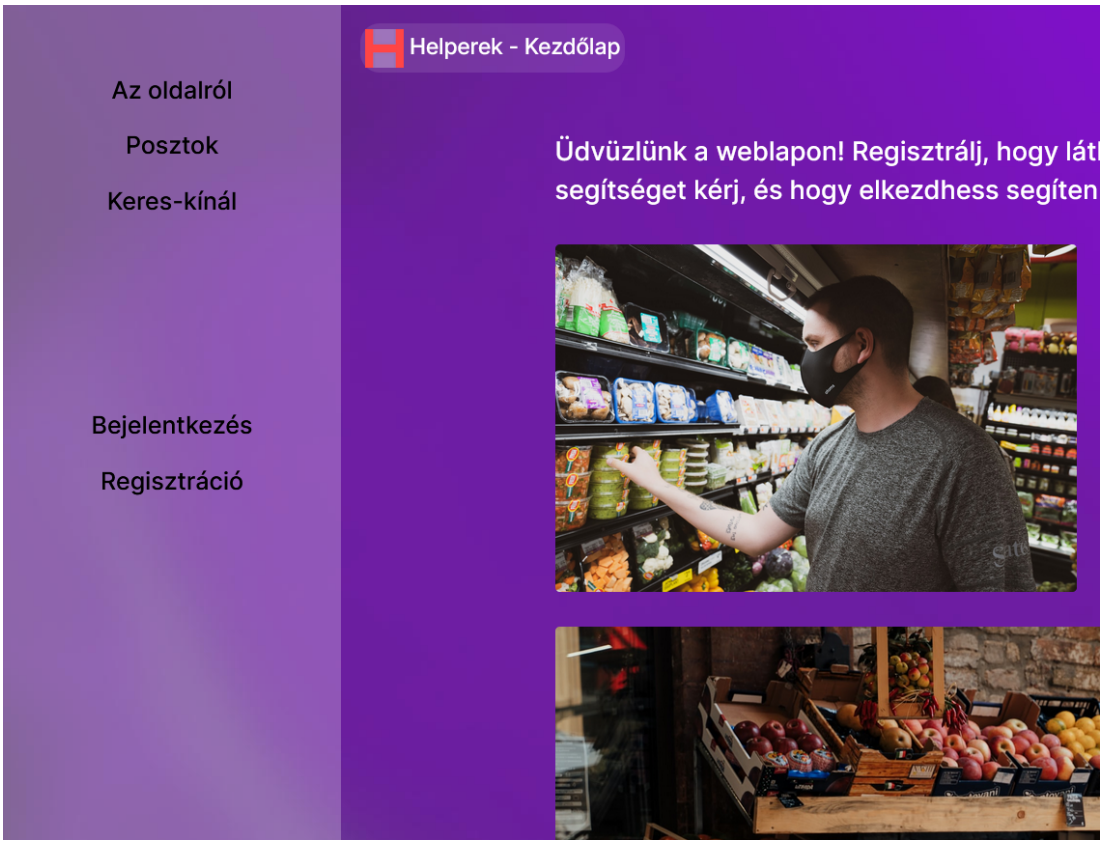
A posztokat kategóriákba soroljuk aszerint, hogy milyen területre vonatkoznak. Van keres/kínál "részleg" is, ahol különböző tárgyakat rakhatunk fel eladásra, vagy keresésre.

A segítők/segítséget kérők tudják egymást értékelni, ha kapcsolatban álltak, például egy sikeresen teljesített bevásárlás után lehet egymást értékelni kedvesség, gyorsaság, pontosság szempontjából, stb.

2. Dokumentáció

2.1. Felhasználói dokumentáció

Az alkalmazás használata nagyon egyszerű, kezelése hasonlít a jól ismert közösségi médiumokhoz, illetve apróhirdetéssel foglalkozó weboldalakhoz. A látogatók lehetőségei korlátozottak, a teljes funkcionalitás eléréséhez regisztrálniuk kell mint felhasználók. Néhány percet és pár adatot vesz igénybe az egész folyamat. A tárolt személyes adatokat nem használjuk fel semmilyen egyéb célra, kizárólag az alkalmazás működése szempontjából fontosak. A felhasználók jelszavai egzakt formában nem kerülnek tárolásra, egyoldalú biztonsági kódoláson esnek át minden esetben.

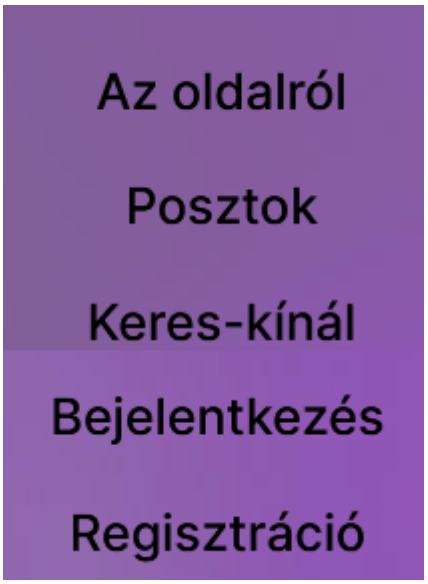


1.ábra

Az alkalmazás kezdőoldala ilyen formában fogad minket asztali számítógépen illetve tableten (1.ábra), valamint olyan eszközökön, ahol a megjelenítő szélessége pixelben mérve legalább 768 pixel. Az oldalsó menüben találhatóak a látogatók számára elérhető funkciók. Mobilon illetve szűk megjelenítőkön hasonló menü érhető el a nyitó gomb megnyomásával, a bal felső sarokban (2.ábra).

2.ábra

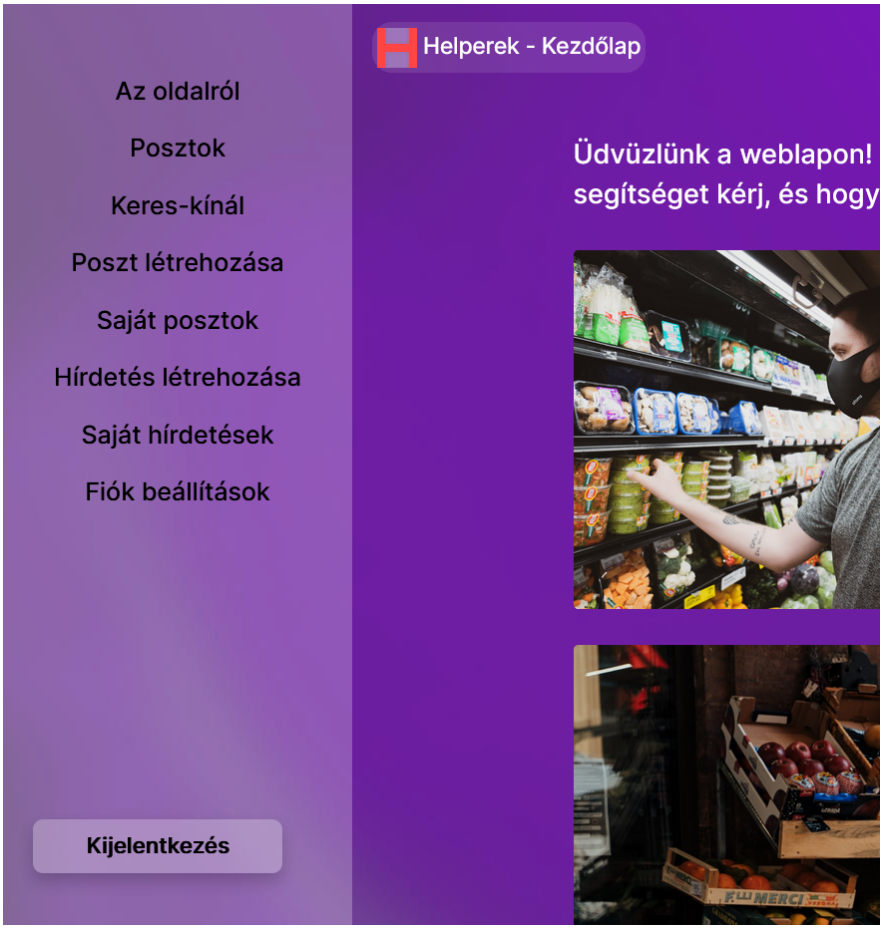
Az elérhető menüpontok és funkcióik (3.ábra):



- *Az oldalról* menüpont kisebb információkat tartalmazó oldalra irányít minket
- A *Posztok* lehetőség a felhasználók által posztolt összes tartalomra mutat
- A *Keres-kínál* a felhasználók apróhirdetéseit jeleníti meg
- A *Bejelentkezés* és a *Regisztráció* menüpontok értelemszerűen az alkalmazáshoz csatlakozást teszik lehetővé

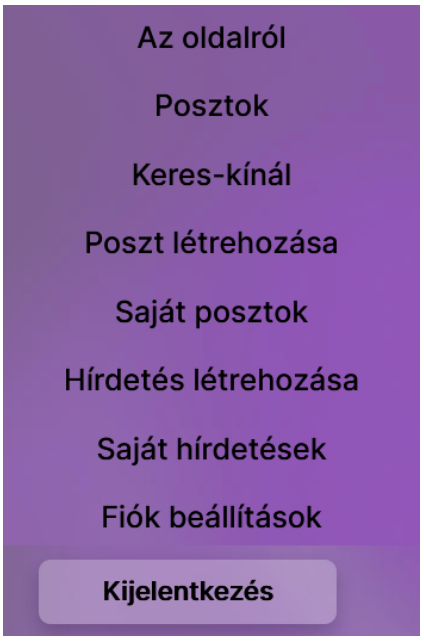
3.ábra

Az alkalmazás további funkciói a regisztrációt, majd a bejelentkezést követően nyílnak meg a látogatók előtt. A 4. ábra mutat egy példát erre.



4. ábra

A bővített opciók természetesen ugyanúgy jelennek meg mobil illetve szűk megjelenítőjű eszközökön.



- *Az oldalról*, *Posztok* és a *Keres-kínál* lehetőségek változatlanul ott vannak
- A *Poszt létrehozása* menüponttal a poszt készítő oldalra juthatunk el, a *Hirdetés létrehozása* gomb hasonlóképp a hirdetés készítőhöz
- A *Saját posztok* illetve *Saját hirdetések* lehetőségek a bejelentkezett felhasználó saját tartalmát jelenítik meg
- A *Fiók beállításokban* a felhasználói fiókra vonatkozó adatokat, beállításokat érhetjük el
- A *Kijelentkezés* gomb értelemszerűen kijelentkezteti az adott felhasználót a rendszerből

5. ábra

Az alap tartalom-megjelenítő oldalakon, pl Posztok, lehetőség van a szűrésre több szempont alapján is. Akár városrész, akár kategória alapján, illetve lehet bevitt szöveg alapján is keresni (6. ábra, 7. ábra).

6. ábra

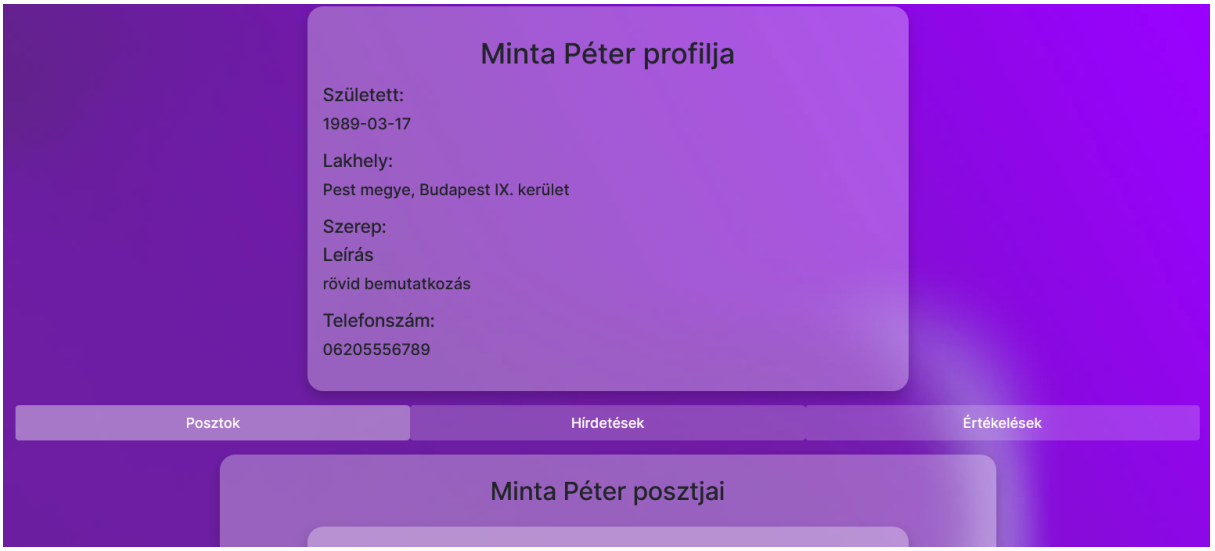

Az oldalról
Posztok
Keres-kínál
Bejelentkezés
Regisztráció
Helperek - Kezdőlap
Keres-kínál hírdetések
Keresés a hírdetések között:
Keresés...
Rendezés:
Legfrissebb elől
Városrész:
Összes
Szűrés

7. ábra


Poszt létrehozása
Téma/Cím megadása:
Egy rövid, tömör, lényegretörő leírás a posztról
Poszt szövege:
Ide írhatod a szöveget
Itt kifejtheted a problémát, megadhatsz részleteket, amik szerinted fontosak lehetnek
Kategória:
Egyéb
Szerep:
Segítséget kér
Posztolás

8. ábra

9. ábra



10. ábra

A 8. ábrán látható a poszt létrehozásának ablaka. Az egyes beviteli mezőkhöz némi segítséget adnak a mellékes szövegek. A kategória és a szerepkör kiválasztása után, illetve ha minden beviteli mezőben van valamekkora mennyiségű szöveg, a ‘Posztolás’ feliratú gomb megnyomásával, ha sikeresen lefut a feladat, a poszt bekerül az adatbázisba, és az oldal átirányítja a felhasználót a poszt főoldalra, ahol meg fog jelenni az újonnan létrehozott poszt.

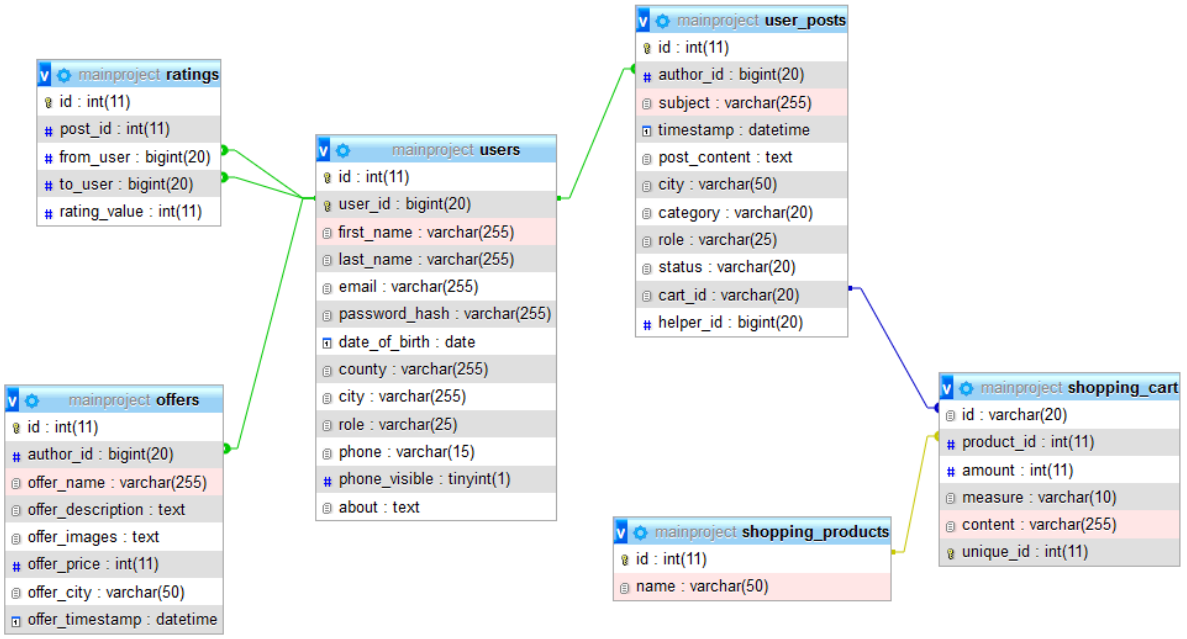
A 9. ábrán látható a hasonló működéssel bíró hirdetés létrehozó form. Itt kategória és szerepkör adatok helyett feltölthet a felhasználó egy képet az adott tárgyról, illetve megadhatja az árat, forintban, egészre kerekítve. A ‘Posztolás’ gombra kattintva, ha sikeresen futnak le a háttérfolyamatok illetve sikeres az adatbázisba beszúrás, az oldal átirányítja a felhasználót a hirdetések főoldalra, ahol meg fog jelenni az újonnan létrehozott hirdetés.

A 10. ábrán egy felhasználói minta profil megtekintése látható. A megadott adatok, személyes információk mellett az adott felhasználó többi tartalma is megtekinthető, mint például a posztjai, hirdetései, illetve ha van, akkor a felhasználó kapott értékelései.

2.2. Fejlesztői dokumentáció

2.2.1. Adatbázis

Az alkalmazás hátterében egy adatbázis üzemel, melyek alapja MySQL nyelv, InnoDB motorral, hogy az idegen kulcsokat kezelje a rendszer. A 11. ábrán látható a szerkezet.



11. ábra

Részletes leírást a következő táblázatok adnak:

users tábla		
id	INT(11)	Egyedi azonosító szám, de nem az elsődleges azonosító
user_id	BIGINT(20)	Egyedi azonosító szám, ez az elsődleges azonosító, függvény segítségével jön létre
first_name	VARCHAR(255)	Felhasználó keresztneve
last_name	VARCHAR(255)	Felhasználó vezetékneve
email	VARCHAR(255)	Felhasználó e-mail címe

password_hash	VARCHAR(255)	Felhasználó jelszava titkosítva, hasítófüggvénnyel
date_of_birth	DATE	Felhasználó születési dátuma
county	VARCHAR(255)	Lakhely megye
city	VARCHAR(255)	Lakhely város/városrész
role	VARCHAR(25)	Felhasználó betölteni kívánt szerepe
phone	VARCHAR(15)	Felhasználó telefonszáma
phone_visible	TINYINT(1)	A felhasználó eldöntheti a megadott telefonszámáról, hogy az látható legyen-e
about	TEXT	Felhasználó rövid bemutatkozása

12. ábra A *users* tábla

user_posts tábla		
id	INT(11)	Poszt egyedi azonosító
author_id	BIGINT(20)	Szerző egyedi azonosító
subject	VARCHAR(255)	Poszt témamegjelölés
timestamp	DATETIME	Létrehozás dátuma
post_content	TEXT	Poszt tartalma
city	VARCHAR(50)	Felhasználó lakhelye (város/városrész)
category	VARCHAR(20)	Poszt kategória
role	VARCHAR(25)	Poszt szerep
status	VARCHAR(20)	Poszt státusza
cart_id	VARCHAR(20)	Ha bevásárlás kategóriájú a poszt, akkor tartozik hozzá egy “bevásárlókocsi”, ennek az egyedi azonosítója
helper_id	BIGINT(20)	Segítő felhasználó egyedi azonosító

13. ábra A *user_posts* tábla

shopping_cart tábla		
id	VARCHAR(20)	Egyedi azonosító, függvénnyel generálva
product_id	INT(11)	Termék egyedi azonosító
amount	INT(11)	Termék mennyiség
measure	VARCHAR(10)	Termék mértékegység
content	VARCHAR(255)	Termékhez fűzött megjegyzés
unique_id	INT(11)	Egyedi azonosító

14. ábra A *shopping_cart* tábla

shopping_products tábla		
id	INT(11)	Egyedi azonosító
name	VARCHAR(50)	Termék megnevezés

15. ábra A *shopping_products* tábla

offers tábla		
id	INT(11)	Egyedi azonosító
author_id	BIGINT(20)	Szerző egyedi azonosító
offer_name	VARCHAR(255)	Hirdetés cím/megnevezés
offer_description	TEXT	Hirdetés leírása
offer_images	TEXT	Hirdetéshez kapcsolódó képek nevei
offer_price	INT(11)	Hirdetés ár
offer_city	VARCHAR(50)	Hirdetés helyszín (város/városrész)
offer_timestamp	DATETIME	Hirdetés létrehozásának dátuma

16. ábra Az *offers* tábla

ratings tábla		
id	INT(11)	Egyedi azonosító
post_id	INT(11)	Kapcsolódó poszt egyedi azonosítója
from_user	BIGINT(20)	Az értékelő felhasználó egyedi azonosítója
to_user	BIGINT(20)	Az értékelni kívánt felhasználó egyedi azonosítója
rating_value	INT(11)	Az értékelés számbeli értéke

17. ábra A *ratings* tábla

A *users* táblában (12. ábra) a felhasználó adatai kerülnek rögzítésre, nagy részük regisztrációkor, a többi pedig regisztráció után, opcionálisan megadhatóak pl telefonszám. A *user_posts* tábla (13. ábra) a felhasználók posztjait tárolja, megadva minden szükséges részletet. Ha bevásárlás kategóriájú posztot hoz létre, és a "kosárba" rakja a szükséges termékeket, úgy a kosár egyedi azonosítója is rögzítésre kerül a *shopping_cart* táblából (14. ábra). E táblán belül a kosarak tartalma kerül mentésre, pl termék egyedi azonosító, mennyiség stb. A *shopping_products* tábla (15. ábra) tárolja a listába rakható elemeket, amiket a felhasználó bátran bővíthet a poszt létrehozás oldalán, ha a keresett termék még nem szerepel az adatbázisban. Az *offers* tábla (16. ábra) tartalmazza a keres-kínál hirdetések tartalmát részletesen. A *ratings* tábla (17. ábra) pedig a felhasználók értékeléseit menti el, tárolva az értékelőt és az értékelendő felhasználó egyedi azonosítóját is az értékelés számbeli értéke mellett.

2.2.2. Javascript

Az oldal egyéb kisebb funkcióit Javascript nyelven megírt függvények, fájlok biztosítják. Egyik példa erre a megfelelő háttér beállításáért felelős *img.js* elnevezésű fájl. A logika itt megkapja az adott oldal címét/megnevezését, majd ez alapján eldönti, hogy melyik képet használja a megadottak közül, mint háttérkép. A háttérképek saját készítésűek, a gyökérkönyvtár *img* mappájában találhatóak, a többi képi elemmel együtt. A logika is ezekre a fájlokra mutat.

A másik fontos alapelem a *sidebar.js* nevezetű fájl. Az ebben lévő logika felelős az oldalsó menü megfelelő működéséért. A fájl folyamatosan vizsgálja az éppen megjelenített oldalt befoglaló ablak méretét, és ahhoz mérten működteti az oldalsó menüt.

`function moveNav(){...}` tartalmazza az oldalsó menü méretezésének logikáját. Ha a megjelenítő képernyő szélessége pixelben mérve kisebb vagy egyenlő, mint 575 pixel, akkor maga a menü felveszi a megjelenítő szélességét, ha ki van nyitva, egyébként pedig alapból el van rejtve, amennyiben a megjelenítő szélessége kisebb, mint 992 pixel, illetve az oldalra tapad és folyamatosan látszik, ha ennél szélesebb a képernyő.

A `window.onresize = function(){...}` logika folyamatosan meghívódik, amikor a megjelenítő szélessége vagy magassága változik, és az éppen aktuális mérethez igazodva állítja be az oldalsó menüt, illetve az azt kezelő gombokat, gombok láthatóságát.

A Post osztályhoz tartozik egy create-shopping.js nevű fájl, amely a bevásárló lista létrehozásánál nyújt segítséget, lehetőséget adva a felhasználónak új termék felvételére. Az ebben a fájlban található logika elvégzi a szükséges műveleteket és meghívja az adott osztályokhoz tartozó, megfelelő függvényeket, hogy a termék bekerüljön az adatbázisba, ezáltal a felhasználó számára választhatóvá váljon.

Ezek a fájlok a gyökérkönyvtár *js* mappájában találhatóak meg.

2.2.3. PHP

2.2.3.1. Modellek

Az alkalmazás úgymond motorja PHP alapokon van, és az MVC nevű tervezési módszerrel lett megalkotva. Az MVC három angol szóból, a **Model**, a **View** és a **Controller** szavak kezdőbetűiből áll össze. A lényege, hogy az adatok megjelenítése el van választva a működési logikától. Részletesebben: adott egy vezérlő (controller), ami megkapja a felhasználótól a kérést egy bizonyos művelet elvégzésére, majd a modellből megszerzi az adatot, amit átad a view-nak megjelenítésre.

Először a Modelleket vizsgáljuk meg. Összesen 7 darab modell lett létrehozva, helyük a gyökérkönyvtár *src* mappájában, az *app* almappa *model* nevű könyvtárában találhatóak meg, névszerint *Cart.php, Offer.php, Post.php, Product.php, Rating.php, ShopPost.php* és *Users.php.* Ezek nagyjából megfelelnek az adatbázisban található tábláknak. A PHP modellek ebben az esetben tartalmazzák az összes, adott táblához tartozó adatmezőt, általában azonos elnevezéssel, megkönnyítve a logika létrehozását.

`Cart.php`

A Cart.php tartalmazza a bevásárlókosár felépítését és függvényeit.

```
private $id;
private $product_id;
private $amount;
private $measure;
private $content;
private $unique_id;
```

Az adattagokhoz tartoznak úgynevezett *getter* illetve *setter* függvények is, melyekkel lekérni, illetve beállítani lehet az adott tag értékét. Például:

```
/**
 * Get the value of id
 */
public function getId()
{
    return $this->id;
}
```

```
        /**
         * Set the value of id
         *
         * @return  self
         */
        public function setId($id)
        {
            $this->id = $id;

            return $this;
        }
```

Ilyen funkciók tartoznak az összes adattaghoz, az összes modellen belül. A legtöbb modellhez tartozik legalább egy adatbázisból adatot lekérő függvény is.

```
        public static function findAll()
        {
            $conn = Database::getConnection();
            $sql = "SELECT * FROM `shopping_cart`";
            $statement = $conn->prepare($sql);
            $statement->execute();
            return $statement->fetchAll(PDO::FETCH_CLASS,
self::class);
        }
```

Ez a függvény megkeresi és visszaadja az összes adatsort a *shopping_cart* táblából, mely sorok mind *Cart* osztály típusúak lesznek. A Database egy külön osztály, később kerül részletezésre.

```
        public static function findAllById($id)
        {
            $conn = Database::getConnection();
            $sql = "SELECT * FROM `shopping_cart` WHERE `id` =
:id";
            $statement = $conn->prepare($sql);
            $statement->execute([
                ":id" => $id
            ]);
            return $statement->fetchAll(PDO::FETCH_CLASS,
self::class);
        }
```

Ebben a függvényben a kosár egyedi azonosítója alapján szűr a logika, és visszaadja az összes talált adatsort, melyek Cart osztály típusúak lesznek. A bemeneti *$id* paraméter a szűrési változó.

```
        public function load($product, $cart)
        {
            $this->amount = $cart['amount'];
            $this->content = $cart['content'];
            $this->product_id = $product['name'];
        }
```

A `load()` függvény két darab bemeneti paramétert kap, a *$product* egy *Product* osztálybeli elem, a *$cart* pedig egy *Cart* osztálybeli elem, azaz ezek a példányok rendelkeznek különböző tulajdonságokkal, amiket itt fel is használunk. Itt az adott, még üres *Cart* példány tulajdonságait megadjuk a paraméterek adataiból.

```
        public function create()
        {
            $conn = Database::getConnection();
            $sql = "INSERT INTO `shopping_cart`
            (`id` ,`product_id`, `amount`, `measure`,
`content`)
            VALUES (:id, :product_id, :amount, :measure,
:content)";
            $statement = $conn->prepare($sql);
            $result = $statement->execute([
                ":id" => $this->id,
                ":product_id" => $this->product_id,
                ":amount" => $this->amount,
                ":measure" => $this->measure,
                ":content" => $this->content
            ]);
            if ($result)
            {
                return true;
            }
            return false;
        }
```

A `create()` függvény a korábban létrehozott *Cart* példányt, miután az fel lett töltve adatokkal, megpróbálja beszúrni az adatbázisba, beillesztve a megfelelő adattagot a megfelelő tábla oszlopba. Sikeres művelet esetén egy *true* vagyis igaz választ kapunk, míg hiba fellépése esetén *false*, azaz hamis visszajelzést kapunk.

Offer.php

A következő osztály az *Offer* osztály, mely a keres-kínál hirdetéseket modellezi le. Az adattagok:

```
private $id;
private $author_id;
private $offer_name;
private $offer_description;
private $offer_images;
private $offer_price;
private $offer_timestamp;
```

Mint az összes osztálynál, így itt is az adattagok nevei megegyeznek az adatbázis hozzá kapcsolódó táblájának oszlopneveivel. További osztályon belüli változók is vannak:

```
private static $loadableCreate = ['author_id', 'offer_name', 'offer_description', 'offer_images', 'offer_price'];
```

A `$loadableCreate` egy adattömb változó, melybe azok a szöveges adattag megnevezések kerülnek, amiket az adatbázisba beszúrásnál az adott példánynak meg kell kapnia.

```
private static $loadableUpdate = ['offer_name', 'offer_description', 'offer_images', 'offer_price'];
```

A `$loadableUpdate` hasonló az előzőhöz, annyi különbséggel, hogy ebben az adott adatsor adatainak frissítéséhez szükséges változók nevei szerepelnek.

```
private $createErrors = [];
private $updateErrors = [];
```

Ezek az üres tömbök a példány adatfeltöltése utáni adatvalidálás során vannak használva, hogy összegyűjtsék az esetleges hibás tagokhoz tartozó hibaüzeneteket.

A következő függvény a `generate_img_name()`, amely a hirdetéshez feltölteni kívánt képnek ad egy 20 karakter hosszú véletlenszerűen összeállított nevet az adott karakter-tömbből, aminek tartalma számok 0-9-ig illetve an angol ábécé kisbetűi. Itt a lehetőségek száma elég nagy, 20 karakter, mindegyik karakternek 36 változata lehet, összesen 20^{36} variáció van, minimalizálva a duplikáció lehetőségét.

```
        public function generate_img_name()
        {
            $numlength = 20;
            $array = ['0', '1', '2', '3', '4', '5', '6', '7',
'8', '9',
                    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j',
                    'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't',
                    'u', 'v', 'w', 'x', 'y', 'z'];
            $img_name = "IMG_";

            for ($i = 0; $i < $numlength; $i++) {
                $img_name .= $array[rand(0, count($array) -
1)];
            }

            return $img_name;
        }
```

A `findAll($order)` függvény itt is hasonlóan működik, mint az előzőekben, viszont itt kap egy bemeneti paramétert ***$order*** néven, ami a rendezési elvet tartalmazza, azaz növekvő vagy csökkenő sorrendben kapjuk meg a hirdetéseket, a létrehozás dátuma szempontjából. Azaz a növekvő vagy ASC szerint a legkorábbiak kerülnek előre, míg a csökkenő vagy DESC szerint a legfrissebbek kerülnek előre. Több hasonló lekérdezéses függvény van, de azoknál nem mutatom be a teljes kódot, mindössze néhány eltérés van a `findAll($order)` és a többi hasonló lekérdezés között. A logikát illetve a paramétereket természetesen ismertetem.

```
        public static function findAll($order)
        {
            $conn = Database::getConnection();
            if (strtoupper($order) == "DESC")
            {
                $sql = "SELECT * FROM `offers` ORDER BY
`offer_timestamp` DESC";
            }
            else if (strtoupper($order) == "ASC")
            {
                $sql = "SELECT * FROM `offers` ORDER BY
`offer_timestamp` ASC";
            }
            $statement = $conn->prepare($sql);
            $statement->execute();
            return
$statement->fetchAll(PDO::FETCH_CLASS,self::class);
        }
```

`public static function findAllByFilters($text, $city, $order){...}`

A `findAllByFilters($text, $city, $order)` a bemeneti paramétereket beilleszti az SQL lekérdezés kódjába, így a kívánt szűrőkkel szűkített eredményt kapjuk meg. A *$text* a címben illetve a leírásban keresi a hasonló szöveget, a *$city* paraméter a város/városrész szűrésében segít, illetve az *$order* itt is meghatározza az időrendiséget.

`public static function findAllOrderByPrice($order){...}`

A `findAllOrderByPrice($order)` függvény a bemeneti *$order* paraméter alapján rendezi az összes adatsort, viszont itt a hirdetések ára szerint állítja azokat növekvő vagy csökkenő sorrendbe.

```
public static function findAllByAuthorId($author_id, $order){...}
```

A `findAllByAuthorId($author_id, $order)` függvény megkeresi és visszaadja az összes olyan hirdetést, amit a bemeneti *$author_id* paraméterrel megadott szerző egyedi azonosító azonosít, lényegében az adott szerző összes hirdetését kapjuk meg, *$order* alapján szintén rendezve.

```
public static function findOneById($id){...}
```

A `findOneById($id)` a bemeneti *$id* paraméter alapján ad vissza egyetlen példányt, ha az szerepel az adatbázisban és sikeresen megtalálta.

```
public function load($data)
{
    foreach (self::$loadableCreate as $item) {
        if (array_key_exists($item, $data) && !empty($data[$item]) || $data[$item] == "0")) {
            $this->$item = $data[$item];
        }
    }
}
```

A `load($data)` függvény betölti az adott példányba a bemeneti *$data* adattömbből az adott értékeket, a *$loadableCreate* tömb segítségével.

```
public function loadUpdate($data){...}
```

A `loadUpdate($data)` a `load()`-hoz hasonlóan működik, annyi különbséggel, hogy itt a *$loadableUpdate* tömböt hívjuk segítségül az adatok betöltéséhez.

```
public function validateCreate(){...}
public function validateUpdate(){...}
```

A `validateCreate()` és `validateUpdate()` függvények validálják az adott példányt, amire meg lettek hívva, ellenőrizve a megadott adatok helyességét, és hiba esetén üzenettel töltik fel az erre a célra létrehozott *$createErrors* illetve *$updateErrors* tömböket. Ezek a függvények *true* vagy *false* értékkel térnek vissza attól függően, hogy a hibaüzeneteket tároló tömbök elemszáma megegyezik-e nullával, vagy sem.

```
public function create(){...}
```

A `create()` függvény az adatbázisba illesztés előtt meghívja a `validateCreate()` függvényt, majd amennyiben az igaz értékkel tér vissza, úgy a feltölteni kívánt képfájlnak generál egy nevet a `generate_img_name()` függvénnyel, azt áthelyezi az ilyen fájloknak létrehozott könyvtárba, majd pedig megkísérli a példány megadott adatait beilleszteni az adatbázisba. Bármilyen felmerülő hiba esetén a függvény *false* értékkel tér vissza, egyébként *true* értéket ad.

```
public function update(){...}
```

Az `update()` függvény szinte megegyezik a `create()` függvénnyel, azonban itt már egy az adatbázisban szereplő adatsort módosítunk a példány megadott adataival.

```
public function delete(){...}
```

A delete() függvény az adott példányt kikeresi az adatbázisból, majd végérvényesen kitörli azt az adatsort, amennyiben létezik és megtalálta.

Post.php

A *Post* osztály a posztok modellje, az adatbázis *user_posts* táblájának megtestesítője. Adattagjai:

```
private $id;
private $author_id;
private $subject;
private $timestamp;
private $post_content;
private $city;
private $category;
private $role;
private $status;
private $cart_id;
private $helper_id;
```

Tartozik hozzá egy hibaüzeneteket tároló adattömb is:

```
private $postErrors = [];
```

```
public static function findAll($order){...}
```

A `findAll($order)` függvény itt is megtalálható, működése már ismert, az összes adatsort lekéri az adattáblából, az *$order* bemeneti paraméter adja az időbeli rendezés módját.

```
public static function findAllByFilters($text, $city, $order, $category, $role){...}
```

A `findAllByFilters($text, $city, $order, $category, $role)` függvény a számos bemeneti paraméter alapján szűri és rendezi a lekért adatsorokat. A *$text* és *$city* már szerepelt, ugyanúgy mint az *$order*. Új változók a *$category* és a *$role*. A *$category* paraméterrel a kategóriát tudjuk beállítani szűrésre, illetve a *$role* változóval pedig a posztok beállított szerepkörét tudjuk szűrni.

```
public static function findAllByAuthorId($author_id, $order){...}
```

A `findAllByAuthorId($author_id, $order)` ezen osztályon belül is szerepel, célja, hogy a megadott *$author_id* mint szerző egyedi azonosító alapján lekérje a keresett adatsorokat *$order* szerint rendezve.

```
public static function findAllByAuthorIdAndFilter($author_id, $order, $text, $category){...}
```

A `findAllByAuthorIdAndFilter($author_id, $order, $text, $category)` függvény az *$author_id* paraméterrel megadott szerző adatsorait kéri le rendezve *$order* szerint és szűrve *$text* szöveg és *$category* kategória szerint.

```
public static function findOneById($id){...}
```

A `findOneById($id)` a megadott *$id* egyedi azonosító alapján megkeresi és visszaadja a keresett adatsort.

```
public function load($data, $user, $cart_id){...}
```

A `load($data, $user, $cart_id)` függvény betölti az adatokat a példányba a megadott *$data* és *$user* adattömbökből, illetve a *$cart_id* paramétert, ami a *Cart* osztály adott függvényéből jön.

```
public function validate(){...}
```

A `validate()` függvény a példány megadott adatainak validálását végzi, hibás találat esetén feltölti a *$postErrors* adattömböt az adott hibaüzenetekkel.

```
public function create(){...}
public function update(){...}
public function delete(){...}
```

A már ismert hármas, a `create()`, `update()`, `delete()` működése ezen osztályon belül sem különbözik a többitől. A `create()` és `update()` függvényeken belül először az adatok validálására kerül sor a `validate()` függvénnyel, majd pedig megkísérli a példány adatainak az adatbázisba beszúrását, illetve az adott adatsor frissítését. A `delete()` pedig végérvényesen törli az adott adatsort, amennyiben létezik. A függvények sikeres végrehajtás esetén *true* értékkel, bármilyen hiba esetén *false* értékkel térnek vissza.

```
public function setHelper(){...}
```

A `setHelper()` függvény akkor kerül meghívásra, ha az adott felhasználó egy segítséget kérő posztjára egy másik felhasználó felajánlja a segítségét. A függvény frissít egy már meglévő adatsort, beállítva a *$status* adattagot 'Folyamatban' értékre, illetve beállítja a *$helper_id* egyedi azonosítót a segítséget felajánló felhasználó egyedi azonosítójára.

```
public function endStatus(){...}
```

Az `endStatus()` függvény egy segítségkérő poszt befejeztekor kerül meghívásra, beállítva az adott adatsor *$status* értékét 'Befejezett' értékre.

Product.php

A *Product* osztály a bevásárlókocsis termékek egyszerű osztálya.

```
private $id;
private $name;
```

Függvényei:

```
public static function findAll(){...}
```

A `findAll()` függvény kilistázza az összes táblában szereplő adatsort.

```
public static function findOneById($id){...}
```

A `findOneById($id)` a megadott *$id* paraméter alapján megkeresi és visszaadja az adott adatsort, amennyiben létezik.

```
public function alreadyExists($name){...}
```

Az `alreadyExists($name)` függvény megvizsgálja az összes adatsort a *$name* paraméterrel egyezés alapján és *true* értékkel tér vissza, amennyiben talál egyezést, egyébként pedig *false* értéket ad.

```
public function addNewProduct($name){...}
```

Az `addNewProduct($name)` függvény veszi a bemeneti *$name* paramétert, lefuttatja vele az `alreadyExists()` függvényt, és ha az *false* értékkel tér vissza, akkor beszúrja a példány adatait a táblába, új terméket létrehozva.

Rating.php

A *Rating* osztály felelős a felhasználók értékeléseinek modellezésére.

```
private $id;
private $post_id;
private $from_user;
private $to_user;
private $rating_value;
```

Az osztály adattagjai, az azonos nevű tábla alapján.

```
private $createErrors = [];
```

A *$createErrors* adattömb tartalmazhatja az értékelés létrehozásakor felmerülő hibák üzeneteit.

```
public static function findAllGivenTo($to_user){...}
```

A `findAllGivenTo($to_user)` függvény a bemeneti *$to_user* paraméter alapján kilistázza az összes talált értékelést, ahol a *$to_user* az értékelt felhasználó egyedi azonosítója.

```
public static function findOneByPostId($post_id){...}
```

A `findOneByPostId($post_id)` függvény a *$post_id* paraméter alapján megkeresi az adott posztot egyedi azonosító alapján, és az ahhoz tartozó értékelést adja vissza, ha létezik.

```
public function load($_to_user, $_value)
{
    $this->to_user = $_to_user;
    $this->rating_value = $_value;
}
```

A `load($_to_user, $_value)` függvény betölti a bemeneti paraméterek értékeit a példány megfelelő adattagjaiba.

```
public function validate(){...}
```

A validate() függvény validálja a példány adattagjainak megadott értékeit, majd hiba esetén feltölti a *$createErrors* adattömböt hibaüzenetekkel.

```
public function create(){...}
```

A `create()` függvény meghívja a `validate()` függvényt, ami ha true értékkel tér vissza, megkísérli a példány adatainak beillesztését a táblába.

ShopPost.php

A *ShopPost* osztály a bevásárlókocsi létrehozásánál és tárolásánál játszik szerepet. Adattagjai:

```
private $cart_id;
private $owner_id;
```

A `generate_cart_id()` függvény hasonlóan az *Offer* osztály `generate_img_name()` nevű függvényéhez, véletlenszerű egyedi azonosítót hoz létre a bevásárlókocsinak. Ez az egyedi azonosító több *Cart* osztálybeli adatsornak is meghatározhatja a csoportosítását, ez alapján kerülnek össze a termékek a kosárban.

```
        function generate_cart_id() {
            $numlength = 20;
            $array = ['0', '1', '2', '3', '4', '5', '6', '7',
'8', '9',
                    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j',
                    'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't',
                    'u', 'v', 'w', 'x', 'y', 'z'];
            $cart_id = "";

            for ($i = 0; $i < $numlength; $i++) {
                $cart_id .= $array[rand(0, count($array) -
1)];
            }

            return $cart_id;
        }
```

`public function create($owner_id){...}`

A `create($owner_id)` függvény feltölti az adott példány adattagjait az *$owner_id* paraméterrel, illetve a `generate_cart_id()` függvény segítségével, majd beszúrja az adatokat a megfelelő táblázatba.

User.php

A User osztály a felhasználót, illetve a users táblát modellezi le. Adattagjai a következők:

```
private $id;
private $user_id;
private $first_name;
private $last_name;
private $email;
private $password;
private $passwordConfirm;
```

```
    private $password_hash;
    private $date_of_birth;
    private $county;
    private $city;
    private $role;
    private $phone;
    private $phone_visible;
    private $about;
```

Továbbá tartozik hozzá három darab adattömb, a különböző függvények végrehajtásánál fellépő hibák üzeneteinek gyűjtésére:

```
    private $registerErrors = [];
    private $loginErrors = [];
    private $updateErrors = [];

    private static $loadable = ['first_name', 'last_name',
'email', 'password', 'passwordConfirm', 'date_of_birth',
'county', 'city', 'role'];
    private static $updateable = ['first_name', 'last_name',
'date_of_birth', 'county', 'city', 'role', 'phone',
'phone_visible', 'about'];
```

A *$loadable* illetve *$updateable* adattömbök tárolják azon változók neveit, amelyeknek értéket kell adni a betöltés során, hogy sikeres legyen az adatbázis táblába való beszúrás.

```
        function generate_user_id() {
            $numlength = 16;
            $user_id = "";

            for ($i=0; $i < $numlength; $i++) {
                $user_id .= rand(0, 9);
            }

            return $user_id;
        }
```

A `generate_user_id()` függvény hasonlít a korábbi véletlenszerű karakter sor generálókhoz, itt azonban mindösszesen 16 karakter hosszúság a cél, illetve csak számokat illesztünk be 0-9-ig. A lehetőségek száma: 16^{10}.

```
public static function findOneByEmail($email){...}
```

A `findOneByEmail($email)` függvény a bemeneti *$email* paraméter alapján megkeresi és visszaadja azt az adatsort, amelynél egyeznek az e-mail címek.

```
public static function findOneByUserId($user_id){...}
```

A `findOneByUserId($user_id)` függvény a bemeneti *$user_id* paraméter, azaz generált egyedi azonosító alapján keresi meg az adott adatsort.

```
public static function findOneById($id){...}
```

A `findOneById($id)` függvény a bemeneti *$id* paraméter, vagy egyedi azonosító alapján keresi meg az adott adatsort.

```
public static function
emailValidation($emailToValidate){...}
```

Az `emailValidation($emailToValidate)` függvény megvizsgálja, hogy az *$emailToValidate* bemeneti paraméterként megadott e-mail cím szerepel-e már az adatbázis adott táblájában. Ha szerepel akkor *true* értékkel tér vissza, egyébként pedig *false* értéket ad.

```
public function load($data){...}
public function loadUpdate($data){...}
```

A `load($data)` illetve a `loadUpdate($data)` függvények betöltik a példány adattagjaiba a bemeneti *$data* adattömbből a megfelelő helyre az értékeket.

```
public function validate(){...}
public function validateUpdate(){...}
```

A `validate()` és `validateUpdate()` függvények validálják az adott példány adattagjainak megfelelő értékadását, illetve hiba esetén a megfelelő adattömbbe (*$registerErrors, $updateErrors*) betöltik a hibaüzeneteket.

```
public function register(){...}
```

A `register()` függvény meghívja a `validate()` függvényt, ami ha true értékkel tér vissza, akkor megkísérli beszúrni a példány adattagjait az adatbázis táblába, ezzel regisztrálva a felhasználót a rendszerben és *true* értékkel tér vissza a végén, bármilyen hiba esetén pedig *false* eredményt ad.

```
public function login($password){...}
```

A `login($password)` függvény megkísérli bejelentkeztetni a felhasználót a rendszerbe, amennyiben a bemeneti *$password* paramétert beillesztve a `password_verify()` függvénybe *true* értéket ad. Ezzel beállítva a felhasználót az eszköz jelenlegi felhasználójának. Bármilyen hiba esetén a függvény *false* értéket ad vissza.

```
public function update(){...}
```

Az `update()` függvény egy már létező adatsor értékeit kísérli meg módosítani, a példány betöltött adattagjaival. Először itt is a `validateUpdate()` függvénnyel validálásra kerülnek az értékek, majd *true* válasz esetén folytatódik az adatok módosítása az adatbázis táblában.Ha sikeres a művelet úgy *true* értéket ad. Bármilyen hiba esetén *false* értékkel tér vissza.

```
public function delete(){...}
```

A `delete()` függvény az adott példány egyedi azonosítója alapján megkeresi az adott adatsort, majd végérvényesen törli azt az adatbázis táblából. Sikeres törlés esetén *true* értéket, bármilyen hiba esetén *false* értéket ad vissza.

2.2.3.2. Nézetek

A nézetek vagy *View*-k azok a fájlok, amik tartalmazzák az adott oldalak megjelenését, dizájnját. Összeköttetésben állnak az adatbázissal, ezáltal a tárolt adatokkal is, természetesen nem közvetlen módon. Nézetfájljai az *Offer*, a *Post*, illetve a *User* osztályoknak van, de az alapértelmezett, sablon nézetek is ebben a könyvtárban találhatóak meg. Ezek kerülnek bemutatásra. A fájlok helyei a gyökérkönyvtár *src* mappáján belül, az *app* mappa *view* könyvtárán belül vannak, külön mappákba rendezve.

Offer

`create.php`

A *create.php* oldalon a bejelentkezett felhasználónak lehetősége van létrehozni egy új hirdetést a megjelenő form segítségével. Az egyes beviteli mezőkhöz fel van tüntetve esetleges segítség is, ha a felhasználó bizonytalan a beírni kívánt adatokkal kapcsolatban.

`index.php`

Az *Offer* osztály *index.php* oldalán megjelenítésre kerül az összes keres-kínál hirdetés a megadott szűrési illetve rendezési feltételeknek megfelelően. Alapvetően két nagyobb részre van osztva az oldal, az első felében találhatóak a keresési és szűrési beállítások, majd az ez utáni részen jelennek meg a hirdetések. Jól elkülönítve egymástól, jól látható kis ablakokban jelennek meg, minden adat jól olvasható. Az oldal betöltésekor lekéri magának az oldal a megadott feltételek alapján a szükséges adatsorokat egy adattömbbe. Ezután a megjelenítő részen egy ciklusos függvény segítségével a tömb minden elemén végigmegy, megjelenítve az egyes tömb elemek minden adatát külön-külön. Ügyelve a kompakt méretekre, egyes elemeket csak részlegesen jelenít meg az oldal, de az elemekre kattintva az oldal átirányít egy olyan oldalra, ahol csak a kiválasztott hirdetés elemei láthatóak, teljes méretben, részletesen.

`view.php`

A *view.php* oldalon az *index.php* oldalról kiválasztott hirdetés jelenik meg teljes méretben, részletesen. Ha olyan hirdetést tekintünk meg itt, aminek esetleg az adott, bejelentkeztetett felhasználó a tulajdonosa/készítője, lehetőség van szerkesztésre, módosításra is, egy ‘Szerkesztés’ feliratú gombra kattintva az oldal átirányít minket a szerkesztő oldalra.

`update.php`

Az *update.php* tartalmazza az adott, szerkeszteni kívánt hirdetés form-ját, amivel gyorsan és egyszerűen lehet módosítani a bevitt adatokon. A beviteli mezőkben megjelennek a már szereplő adatok, amiket szabadon változtathatunk. Amennyiben úgy érezzük, hogy a hirdetés már megfelelő állapotba került, úgy a ‘Mentés’ feliratú gombra kattintva elmenthetjük a módosításainkat. Ha pedig szeretnénk a hirdetést törölni a rendszerből, a ‘Törlés’ gombra kattintva megjelenik egy párbeszédpanel, amelyen utoljára megerősíthetjük, hogy tényleg szeretnénk eltávolítani az adott hirdetést, vagy vissza is léphetünk.

Post

`create.php`

A *create.php* oldalon van lehetőség új poszt létrehozására. A beviteli mezők egyértelmű útmutatást is biztosítanak, hogy a létrehozás gördülékenyen és hiba nélkül menjen végbe. Kategória választáskor amennyiben ‘Vásárlás’-t választ a felhasználó, úgy új lehetőségek jelennek meg közvetlenül a form-on, létre lehet hozni egy virtuális bevásárlólistát is. Ehhez tartozik az is, hogy a felhasználó egy az adott listában nem szereplő terméket hozzáadjon a listához.

`index.php`

Az *index.php* oldalon, hasonlóan az *Offer* osztályhoz, az összes létező poszt megjelenik, tagolva, jól olvashatóan, de itt is limitálva, hogy minél több elem elférjen az oldalon. Két fő részre oszlik az oldal, az első felében lehet beállítani a szűrési és rendezési feltételeket, míg a másik felében megjelennek a feltételeknek eleget tevő adatsorok, illetve adattagjaik, rendezve, formázva. Ha a teljes posztot szeretnénk megtekinteni, itt is lehetőség van rá, csak rá kell kattintani a megtekinteni kívánt posztra.

`view.php`

A *view.php* oldalon teljes egészében megjelenik a kiválasztott poszt, olyan formában, amilyenben létrehozták, ügyelve az üres részekre, nagyobb hézagokra, jó megjelenésre. Amennyiben a poszt vásárlás kategóriájú, úgy a virtuális bevásárlókosár is megjelenik táblázatos formában, minden részletével együtt. Ha a bejelentkeztetett felhasználó a saját posztját tekinti meg, lehetősége nyílik annak szerkesztésére a ‘Szerkesztés’ gombra kattintva, amely átirányítja a megfelelő oldalra.

`update.php`

Az *update.php* oldalon van lehetőség az adott poszt szerkesztésére, illetve törlésére, ha a felhasználó úgy dönt. A már mentett adatok megjelennek, azokat lehet módosítani. A módosítások csak akkor lesznek érvényesek, ha a felhasználó a ‘Mentés’ gombra kattint, ezáltal elmenti az addigi munkáját.

`_rating_modal.php`

A *_rating_modal.php* egy felugró ablakos segéd-oldal, ami egy form-ot tartalmaz, jelen esetben az adott felhasználó értékelésére. A felhasználó megadhatja az általa vélt legmegfelelőbb értéket, majd az ‘Értékelés’ gombra kattintva az értékelése bekerül a rendszerbe az adott felhasználót illetően.

`_shopping_modal.php`

A *_shopping_modal.php* a vásárlás kategóriájú poszt létrehozásánál jelenhet meg, amennyiben a felhasználó olyan új elemet kíván felvenni a listára, amely még nem szerepel az adatbázisban. A felugró ablakban megadhatja a termék nevét, amely ha még nem szerepel a listában, bekerül az adatbázisba, majd a listába is.

User

`register.php`

A *register.php* tartalmazza a regisztrációs form-ot, melyen minden szükséges beviteli mező megtalálható, amely a sikeres regisztrációhoz kell. Amennyiben a regisztráció sikeres, úgy az oldalnak át kell irányítani a felhasználót a bejelentkezési felületre, egyébként pedig a regisztrációs oldalon marad.

`login.php`

A *login.php* tartalmazza a bejelentkezési felületet. Amennyiben a felhasználó sikeresen bejelentkezik, úgy az oldal átirányítja a főoldalra, ahol egy üdvözlő üzenet várja. Ha valamilyen hiba folytán nem sikerül a bejelentkezés, akkor a felhasználó az oldalon marad és egy hibaüzenet tájékoztatja őt a problémáról.

`index.php`

Az *index.php* lényegében a főoldala az alkalmazásnak, ez fogadja a látogatókat is, amikor az oldalra érkeznek. Egy üdvözlő üzenet mellett néhány, az alkalmazáshoz illő kép jelenik meg a felhasználók és a látogatók előtt.

`profile.php`

A *profile.php* a felhasználók profil oldala. A látogatók nem láthatják a felhasználók információit, ahhoz regisztrált tagnak kell lenni, és be kell jelentkezni. Más felhasználók profiljait bármikor megtekinthetjük, amikor be vagyunk jelentkezve. Annyi adatot látunk az adott felhasználóról az alap adatokon kívül, amennyit publikussá tesz. Meg lehet adni egy rövid bemutatkozó szöveget, meg lehet adni telefonszámot is, illetve dönthetnek arról, hogy a megadott telefonszámot inkább mégsem teszik publikussá. A profil oldalon láthatóak az adott felhasználó posztjai, hirdetései, és kapott értékelései is, egy kis ablakban választható ki, hogy mely adatokat szeretné látni.

`posts.php`

A *posts.php* oldalon az adott felhasználó a saját posztjait láthatja, a szűrési és rendezési feltételeknek megfelelően. Ennek az oldalnak a tartalma mindig a bejelentkezett felhasználótól függ. Az itt megjelenő tartalmakra rá lehet kattintani, ekkor az oldal átirányít minket a nézeti oldalra, azaz a *Post* osztály *view.php*-ra, ahol esetleg szerkeszteni is lehet.

`offers.php`

Az *offers.php* az adott felhasználó összes hirdetését tartalmazza, szűréssel és rendezéssel. A megjelenő tartalmakra kattintva az oldal átirányít minket a teljes mértékű megtekintésre, azaz az *Offer* osztály *view.php*-ra, ahonnan akár szerkeszteni is lehet a tartalmat.

Templates (sablon oldalak)

Három darab template azaz sablon oldal van, melyek minden oldalon megjelennek, egy alap külsőt adva az alkalmazásnak. Ezek a *main-template.php*, a *navbar.php*, illetve a *sidebar.php*.

`main-template.php`

A *main-template.php* tartalmazza a HTML alapkódot, amire úgymond épül az alkalmazás. Ebben a fájlban kerülnek meghívásra a különböző extra fájlok, melyek az oldal dizájnjában szerepet játszanak, például a betűtípus, vagy éppen az alkalmazáshoz készített CSS fájl. Továbbá ide kerül behelyezésre a másik két sablon oldal is, ez a 3 fájl együtt alkotja a megjelenés alapját, mely minden oldalon megjelenik.

`navbar.php`

A *navbar.php* tartalmazza a navigációs felület kódját.

`sidebar.php`

A *sidebar.php* tartalmazza az oldalsó menü kódját illetve a megjelenített menü logikáját.

2.2.3.3. Vezérlők

A vezérlők azok az egységek, amelyekben a modellek és a nézetek össze vannak kapcsolva egy bizonyos szinten, a benne lévő logika dönti el, mikor melyik függvényre van szükség a modellből, illetve hogy melyik nézetek legyenek megjelenítve az oldalon adott esetekben. Jelenleg 7 darab ilyen vezérlő van, de nem mindegyik osztályhoz tartozik egy-egy. Van egy fő vezérlő, a *CommonController*, amelyből a többi származik. További vezérlők: *SiteController, OfferController, PostController, ProductController, RatingController,* és *UserController*.

CommonController.php

A *CommonController* az alap vezérlőegység, amelyből a többi ilyen vezérlő származik.

```
public $title = "";
protected $controllerName = null;
```

A *$title* változót a nézet fájlok kapják meg, az adott oldal címét tartalmazza. Ez a cím jelenik meg a böngészőben, a fülön. A *$controllerName* pedig az adott vezérlő nevét fogja tartalmazni a további osztályokban.

```
protected function render($view,$data = [])
{
    extract($data);
    ob_start();
include("src/app/view/{$this->controllerName}/{$view}.php");
    return ob_get_clean();
}
```

A `render($view, $data = [])` függvény a *$view* paraméterben megadott nézet fájlt fogja megjeleníteni a képernyőn, valami a *$data* adattömbben különböző adatokat is át lehet küldeni a nézet fájl számára, hogy azokkal dolgozzon.

SiteController.php

A *SiteController* csupán az egyéb, úgymond mellékes oldalakat kezeli, mint például az *about.php* nézetet.

```
protected $controllerName = "site";
```

Itt a *$controllerName* változó megkapja a ‘site’ értéket, amely alapján később azonosítani tudja a logika a vezérlőt a gyökérkönyvtárban található *index.php*-ban.

```
public function actionIndex()
{
    $this->title = 'Főoldal';
    return $this->render('index');
}
```

Az `actionIndex()` függvény a főoldalra mutat majd, beállítva a *$title* változót, illetve a `render()` függvénynek megadja, hogy melyik nézetfájlt kellene betölteni.

```
protected function render($view, $data = [])
{
    ob_start();
    include("src/helper/{$view}.php");
    return ob_get_clean();
}
```

Ezen a vezérlőn belül a `render()` függvény kicsi átalakuláson esik át, hogy a nézetfájlokat a vezérlő számára egy ideálisabb helyen keresse.

```
public function actionAbout()
{
    $this->title = "Az oldalról";

    return $this->render("about");
}
```

Az `actionAbout()` függvény mutat az *about.php* oldalra, és ahhoz állítja be a vezérlő alapértékeit.

`OfferController.php`

Az `OfferController` vezérlő felelős az *Offer* osztály és a hirdetés nézetek összeköttetéséért.

```
protected $controllerName = 'offer';
```

A *$controllerName* itt természetesen az ‘offer’ értéket kapja. A következőkben ennek a vezérlőnek a különböző függvényei kerülnek bemutatásra részletesen, minden függvény logikája látható lesz. A többi vezérlőnél a hasonló elven működő és hasonló felépítésű függvények nem lesznek ilyen részletesen leírva.

```
public function actionIndex()
{
    if (isset($_GET['order']))
    {
        $offers = Offer::findAll($_GET['order']);
    }
    else
    {
        $offers = Offer::findAll("DESC");
    }

    $this->title = "Keres-kínál hirdetések";

    return $this->render('index', [
        "offers" => $offers
    ]);
}
```

Az `actionIndex()` függvény a hirdetések főoldalát jeleníti meg. Alapesetben, amikor még nincs beállítva semmilyen szűrési és rendezési feltétel, az időrendi rendezést egy alapértékre állítja a logika, ezáltal a hirdetések úgy jelennek meg, hogy a legfrissebbek vannak legelöl. Beállítja továbbá az oldal címét a *$title* változóban, illetve a `render()` függvényben megadja a nézet fájl nevét, illetve beilleszti az adattömbbe a szükséges adatokat.

```
public function actionView($id)
{
    $offer = Offer::findOneById($id);

    $this->title = "Hirdetés megtekintése";

    return $this->render("view", [
        "offer" => $offer
    ]);
}
```

Az `actionView($id)` szolgál a teljes megjelenítős nézet hívására. Az *$id* paraméter tartalmazza az adott, megjeleníteni kívánt poszt egyedi azonosítóját, amelyet az *$offer* változóba megkeres az *Offer* osztály az adott függvényével. Beállítja ezek mellett az oldal címét a *$title* változóba, illetve megadja melyik nézet fájlt hívja meg a képernyőre.

```
    public function actionCreate()
    {
        $this->title = 'Új hirdetés létrehozása';
        $offer = new Offer();
        if (isset($_POST['offer']) &&
isset($_SESSION['user_id']))
        {
            $offer->load($_POST['offer']);
            if ($offer->create()) {
                header("Location:
index.php?controller=offer&action=index&create=success");
                exit();
            }
        }
        return $this->render('create', [
            'offer' => $offer
        ]);
    }
```

Az `actionCreate()` függvény az új hirdetés létrehozásának teljes menetét tartalmazza, leírja. Beállítja az oldal címét a *$title* változóban. A `render()` függvénnyel meghívja az adott oldal nézetét. Megvizsgálja, hogy a PHP fájlból elküldött form megfelelő paraméterei megvannak-e adva, ki vannak-e rendesen töltve, illetve van-e bejelentkeztetett felhasználó, majd meghívja az *Offer* osztály megfelelő függvényeit, hogy az adott példány fel legyen töltve adatokkal, amiket ellenőriz is, majd megkísérli az adatbázis táblába beszúrást. Ha sikeres a művelet, átirányítja a felhasználót a hirdetések főoldalára, ahol már meg is fog jelenni az újonnan létrehozott tartalom. Bármilyen hiba esetén az adott oldalon marad, és tájékoztat a felmerült hibákról.

```
    public function actionUpdate($id)
    {
        $offer = Offer::findOneById($id);
        $this->title = "Hirdetés szerkesztése";
        if (isset($_POST['offer']))
        {
            $offer->loadUpdate($_POST['offer']);
            if ($offer->update())
            {
                header("Location:
index.php?controller=offer&action=index&update=success");
                exit();
            }
            else
            {
                header("Location:
index.php?controller=offer&action=index&update=error");
                exit();
            }
        }
        return $this->render("update", [
            "offer" => $offer
        ]);
    }
```

Az `actionUpdate($id)` egy adott hirdetés módosítását, szerkesztését szervezi meg. Megkeresi az aktuális hirdetést az *$id* egyedi azonosító paramétert behelyettesítve az *Offer* osztály megfelelő függvényébe, majd a `render()`-t megfelelően felparaméterezve megjeleníti a szerkesztés nézetet. Amennyiben a form küldésre került, úgy vizsgálja, hogy a benne lévő adatok megfelelnek-e, ezeket betölti egy példányba, majd megkísérli az adatok módosítását a táblában az adott adatsorban. Ha sikeres a szerkesztés, vagy valamilyen hiba lép fel, a hirdetés főoldalra irányít minket, és üzenettel jelzi a kimenetelt.

```
    public function actionDelete($id)
    {
        $offer = Offer::findOneById($id);
        if ($offer->getAuthor_id() == $_SESSION['user_id'])
        {
            if ($offer->delete()) {
                header("Location:
index.php?controller=offer&action=index&delete=success");
                exit();
            } else {
                header("Location:
index.php?controller=offer&action=index&delete=error");
                exit();
            }
        }
    }
```

Az `actionDelete($id)` függvény az *Offer* osztály törlés függvényét, azaz a `delete()`-t hívja meg, miután egy *Offer* példányt megtalált az *$id* paraméter alapján, és betöltötte az adatait, illetve ellenőrzi, hogy az adott hirdetés szerzője ugyanaz a felhasználó-e aki be van jelentkezve. Ha az eltávolítás sikeres vagy valamilyen hiba lép fel közben, az oldal átirányít minket a hirdetések főoldalára, és üzenettel jelzi, hogy mi lett a végeredmény.

`PostController.php`

A *PostController* vezérlő felelős a posztok nézet fájljainak illetve a *Post* osztály függvényeinek együttműködéséért. A *$controllerName* itt természetesen a ‘post’ értéket kapja. Függvényei a következők:

```
public function actionIndex(){...}
```

Az `actionIndex()` függvény a posztok főoldalát kezeli, meghívja a *Post* osztály *index.php* nézet fájlját, megadja az oldal címét a *$title* változóban, és átadja a megfelelő adatokat a `render()` függvénynek.

```
public function actionView($id){...}
```

Az `actionView($id)` függvény a megadott *$id* egyedi azonosító alapján megkeresi az adott posztot a *Post* osztály megfelelő függvényével, majd a *Post* osztály *view.php* nézet fájlját hívja meg, átadva a keresett példány adatait a `render()` függvénynek.

```
public function actionCreate(){...}
```

Az `actionCreate()` függvény végzi az új poszt létrehozását a háttérben, figyelembe véve a részleteket. Először is beállítja az oldal címét a *$title* változóban illetve megadja a paramétereket a `render()` függvénynek, hogy melyik nézet fájlt kell megjeleníteni az oldalon. Amennyiben érkezik adat a form-ból, úgy azt lekezelei, ellenőrzi az adatok helyességét. Ha a kiválasztott kategória 'vásárlás', úgy mielőtt a *Post* osztályból meghívja a `create()` függvényt, megvizsgálja, hogy az adott kategóriánál szükséges adatok megvannak-e adva, illetve azok helyesek-e, majd a *Cart* osztályt is meghívja, hogy létrehozza a kosarat, amibe ezután betölti a kiválasztott termékeket. Ha ezek probléma nélkül lefutnak, akkor maga a poszt is létrejöhet, megkísérli a beszúrást az adatbázisba a rendszer. Sikeres művelet esetén átirányítja a felhasználót a poszt főoldalra, és egy üzenettel jelzi, hogy sikeresen végrehajtotta a feladatot.

```
public function actionUpdate($id){...}
```

Az `actionUpdate($id)` függvény az adott poszt szerkesztésének hátterét írja le. Beállítja az oldal címét, megkeresi az adott posztot a paraméter *$id* alapján a *Poszt* osztály megfelelő függvényével, majd átadja a paramétereket a `render()` függvénynek. Ellenőrzi, hogy a poszt tulajdonosa megegyezik-e a bejelentkezett felhasználóval, és ha ez stimmel, engedélyezi a szerkesztését. Amennyiben a form adatot küld, azaz a felhasználó szeretné elmenteni a változtatásokat, úgy meghívódik az adott függvény, megkísérelve az adott adatsor módosítását. Sikeres művelet illetve hiba esetén az oldal átirányít minket a poszt főoldalra, ahol a kimenetelnek megfelelő üzenet fogadja a felhasználót.

```
public function actionDelete($id){...}
```

Az `actionDelete($id)` függvény paraméterként megkapja az adott poszt egyedi azonosítóját (*$id*), majd a *Post* osztály megfelelő függvényével megkeresi azt, és betölti egy példányba. Ellenőrzi, hogy a poszt tulajdonosa megegyezik-e a bejelentkezett felhasználóval, és ha igen, megkísérli az eltávolítást az adatbázisból. Ezután a poszt főoldalra irányít, ahol a kimenetelnek megfelelő üzenet fogadja a felhasználót.

```
public function actionHelper($id){...}
```

Az `actionHelper($id)` függvény megkeresi az adott posztot az *$id* paraméter alapján a *Post* osztály megfelelő függvényével, majd a segíteni kívánó felhasználó, azaz aki be van jelentkezve, generált egyedi azonosítóját megkísérli beilleszteni az adott adatsorba a *Post* osztály `getHelper()` függvényével. Sikeres művelet esetén, illetve fellépő hibáknál, az oldal a teljes megjelenítő nézetre azaz a *view.php* oldalra irányítja a felhasználót, és üzenetben közli a művelet végeredményét.

```
public function actionEnd($id){...}
```

Az `actionEnd($id)` függvény a megadott *$id* paraméter alapján megkeresi a szükséges posztot a *Post* osztály megfelelő függvényével, majd meghívja rá az `endStatus()` függvényt. Ezután átirányít a megtekintő nézetre, üzenetben megjelenítve a művelet végeredményét.

`ProductController.php`

A *ProductController* felelős a termék osztály azaz a *Product* osztály műveleteinek megfelelő végrehajtásáért. A *$controllerName* itt 'product' értéket kap. Egyetlen függvénye van:

```
public function actionCreate(){...}
```

Az actionCreate() függvény egy kicsivel összetettebb a többinél olyan szempontból, hogy nem csak a PHP form-okkal kommunikál, hanem Javascript fájlokkal is, konkrétan a *create-shopping.js* fájlban található logikával. Ez végzi ugyanis az új termék felvitelét az adatbázisba. A javascript fájl elküldi az adott form-ot, az `actionCreate()` függvény meghívódik, ezen belül megkísérli az új termék beszúrását az adatbázisba. Majd ha ez sikerül, akkor a javascript fájlnak visszaszolgáltat egy adattömböt JSON formátumban, amit a javascript felbont használható elemekre, ezek pedig megjelennek az adott oldalon.

`RatingController.php`

A *RatingController* szintén egyetlen függvényt tartalmaz, amely az értékelések létrehozásának a hátterét biztosítja. A *$controllerName* itt a 'rating' értéket kapja.

```
public function actionCreate($id){...}
```

Az `actionCreate($id)` a bemeneti *$id* paraméter alapján megkeresi az adott posztot a *Post* osztály megfelelő függvényével, ami alapján fogja az értékelés azaz *Rating* osztály `load()` illetve `create()` függvényét felparaméterezni és használni. Mindezt akkor hajtja végre, ha ellenőrizte, hogy az adott felhasználó be van jelentkezve, és megegyezik a poszt tulajdonosával. Sikeres művelet esetén informálja a felhasználót a végeredményről.

`UserController.php`

A *UserController* felelős a felhasználói osztályt azaz a *User* osztályt érintő műveletek hátteréért, melybe beletartozik a regisztráció, be- és kijelentkezés stb. A *$controllerName* itt a 'user' értéket kapja.

```
public function actionIndex(){...}
```

Az actionIndex() függvény magáért a főoldalért felelős, megadja az oldal címét illetve megadja a render() függvénynek a szükséges nézet nevét, amit be kell tölteni.

`public function actionLogin(){...}`

Az `actionLogin()` végzi el a bejelentkeztetést a rendszerbe. A kitöltött form-ból kapott e-mail alapján megkeresi az adott felhasználót a *User* osztály megfelelő függvényével, majd erre a példányra meghívja a `login()` függvényt, hogy ellenőrizze a megadott jelszót. Sikeres bejelentkezés esetén a főoldalra irányítja át a felhasználót, ellenkező esetben egy üzenetben jelzi, hogy valamilyen probléma lépett fel. A `render()` függvényben megadja, hogy mely nézet fájlt kell megjeleníteni a képernyőn minden esetben.

`public function actionLogout(){...}`

Az `actionLogout()` függvény végzi a kijelentkeztetést a rendszerből. Törli az esetlegesen mentett adatokat az ideiglenes tárolóból, és átirányít a főoldalra, bárhonnan is volt aktiválva.

`public function actionRegister(){...}`

Az `actionRegister()` függvény felelős a regisztráció háttérfolyamataiért. A `render()` függvényben megadja, hogy melyik nézet fájlt kell megjeleníteni, illetve értéket ad a *$title* változónak. Amennyiben jön adat a form-ból, úgy megpróbálja az érkező adatokat beszúrni az adatbázisba, miután ellenőrizte az adatokat. Sikeres művelet esetén a bejelentkező felületre irányít át, egyébként pedig üzenettel jelzi a végeredményt.

`public function actionProfile($user_id){...}`

Az `actionProfile($user_id)` függvény a *$user_id* paraméter alapján eldönti, hogy a bejelentkezett felhasználó melyik felhasználót kívánja megtekinteni ezen az oldalon. Ez alapján kap egy példányt a `render()` függvény paraméterként, mely mellett a szükséges nézet fájl nevét is megkapja.

`public function actionPosts($order){...}`

Az `actionPosts($order)` függvény a bejelentkezett felhasználó saját posztjait jeleníti meg, az *$order* paraméter alapján. Ezekkel meghívja a `render()` függvényt, megadva a megfelelő nézet nevét, amit meg kell jeleníteni.

```
public function actionOffers($order){...}
```

Az `actionOffers($order)` függvény hasonlóan az `actionPosts()`-hoz, a bejelentkezett felhasználó saját hirdetéseit jeleníti meg az *$order* bemeneti paraméter alapján rendezve. Mindezt a `render()` függvényben megadott nézet fájllal, és megadott adatokkal hajtja végre.

```
public function actionUpdate($id){...}
```

Az `actionUpdate($id)` függvény az adott felhasználó profiljának szerkesztésének háttérfolyamatait rendezi. Az *$id* paraméterben megadott egyedi azonosító alapján, ha az egyezik a bejelentkezett felhasználó egyedi azonosítójával, engedélyezi az adatok szerkesztését, melyeket a folyamat betölt. A módosítások mentése esetén a függvény meghívja a `loadUpdate()` majd az `update()` függvényeket, ezekkel megkísérelve az adatok frissítését az adatbázis táblában. A `render()`-ben megadja a szükséges nézet fájl nevét, illetve a betöltendő példányt, mint adattömböt.

```
public function actionDelete($id){...}
```

Az `actionDelete($id)` függvény a megadott *$id* paraméter alapján megkeresi az adott felhasználót a *User* osztály megfelelő függvényével. Amennyiben a függvény meghívója megegyezik a keresett példánnyal, meghívja rá a delete() függvényt, amellyel megkísérli az adatsor végleges törlését az adatbázisból. Ha sikeres a művelet, a főoldalra irányít át, illetve az ideiglenes tárolt adatok is eltűnnek az adott böngészőből, mintha a kijelentkezés funkciót is használta volna.

2.2.3.4. Egyebek

Az egyéb összetevők közé sorolom az alábbi fájlokat:

`Database.php`

A *Database.php* tartalmazza az adatbázishoz való kapcsolódás logikáját és összetevőit, megkönnyítve az adatlekérést a modelleknél.

`config.php`

A *config.php* tartalmazza az adatbázis kapcsolat létrejöttéhez szükséges adatokat, mint például host-név, felhasználónév, jelszó, adatbázis-név. A *Database.php* a kapcsolat létrehozásakor ebből a fájlból kapja az adatokat.

`about.php`

Az *about.php* egy rövid, információs oldal az alkalmazásról. Tartalmazza az oldalon használt, nem saját készítésű képeket forrásmegjelöléssel, linkekkel.

Linkek

Az alkalmazásban használt háttérképek saját készítésüek, a főoldalon megjelenő képek forrása az `unsplash.com` weboldal.

- `https://unsplash.com/photos/fAiQRv7FgE0`
- `https://unsplash.com/photos/wcCwqSglagY`
- `https://unsplash.com/photos/pD0tUqzSCY4`