

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Dav proti vizuálnemu smogu

Inžinierske dielo

Vedúci tímu: Ing. Jakub Šimko, PhD.

Členovia tímu: Bc. Jana Egriová, Bc. Alexander Ferenčík, Bc. Richard Filipčík, Bc. Tomáš Melicher, Bc. Juraj Slavíček, Bc. Jaroslav Zigo

Akademický rok: 2014/2015

OBSAH

1	Úvod	3
2	Slovník pojmov	3
3	Ciele.....	4
3.1	Používateľ odfotoí obrázok	4
3.2	Domased si zobrazí úlovky na mape	4
3.3	Domased nahrá úlovok do mapy	4
3.4	Registrácia a prihlásenie	4
4	Opis prototypu	5
4.1	Funkcionálne požiadavky	5
4.1.1	Webová aplikácia	5
4.1.2	Mobilná aplikácia	5
4.2	Architektúra.....	6
4.2.1	Serverová časť	6
4.2.2	Mobilná časť.....	8
4.3	Dátový model	8
4.3.1	Logický model	8
4.3.2	Fyzický model.....	11
5	Moduly	13
5.1	Webová aplikácia	13
5.1.1	Registrácia a prihlasovanie používateľov.....	13
5.1.2	Mapa	14
5.1.3	Upload obrázka	17
5.2	Android aplikácia	19
5.2.1	Záznam neestetickéj reklamy	19
6	Technická dokumentácia	22

1 ÚVOD

V dnešnej dobe plnej produktov sa vo svete vo veľkej miere rozšírila reklama každého druhu. Reklamám v televízii rádiu a tlačovinách sa človek pri určitej snahe dokáže vyhnúť avšak skryť sa reklamným pútačom, billboardom a svetelným reklamám v uliciach našich miest je takmer nemožné. Všetky tieto druhy pouličných reklám môžeme pomenovať súhrnným názvom vizuálny smog. Vo vyspelejších krajinách bol problém vizuálneho smogu vypuklý už v dávnejšej minulosti a preto je boj s ním v plnom prúde. Naopak u nás sa stáva problém čoraz závažnejším a zároveň stále neriešeným.

Náš tím s názvom Včeličky sa preto v rámci predmetu Tímový projekt snaží tento problém napomáhať riešiť. Navrhli sme aplikáciu, ktorá ma za cieľ s vizuálnym smogom bojovať. Využívame pri tom silu davu tzv. "crowdsourcing". Keďže mestská samospráva pod ktorej kompetencie riešenie tohto problému spadá proklamuje, že jej pre riešenie problému vizuálneho smogu chýbajú spoľahlivé dáta v prvej fáze projektu, riešime práve ich zber.

Aplikácia sa skladá z dvoch vzájomne kooperujúcich častí. Jedná sa o mobilnú a webovú aplikáciu. Cieľovým používateľom je bežný človek zaujímajúci sa o okolie, v ktorom žije s chuťou zlepšovať verejný priestor, ktorý všetci užívame. Na splnenie týchto činností mu stačí mobilný telefón s fotoaparátom, pomocou ktorého bude fotiť, a mať nainštalovanú našu aplikáciu. Používateľ bude na zbieranie dát motivovaný tzv. gamifikáciou, čo znamená, že bude za svoju snahu po dosiahnutí istej hranice odmeňovaný. Webová aplikácia slúži na spresňovanie získaných dát a ich prezentáciu v bežnom webovom prehliadači. Výstupom prvej fázy projektu bude podrobný set dát, ktorý bude možné poskytnúť tretím stranám na prípadné spracovanie, analýzu a z nich vyplývajúce zlepšenie verejného priestoru.

2 SLOVNÍK POJMOV

Pojem	Vysvetlenie
Terén	Vonkajšie prostredie, v ktorom má používateľ možnosť nasnímať neestetickú reklamu
Špina	Akákoľvek neestetická, nelegálna alebo iným spôsobom nevhodná reklama v meste
Úlovok	Nasnímaný záznam špiny v teréne, ktorý bol odoslaný na spracovanie
Domased	používateľ webového rozhrania, ktorý má aplikáciu spustenú na osobnom počítači z domova.

Aktivista	používateľ mobilného rozhrania v teréne, ktorý má záujem odstrániť neestetickú reklamu
------------------	--

3 CIELE

Našími prioritnými cieľmi na zimný semester je zapracovanie nasledovných user stories do funkcionality či už mobilnej alebo webovej aplikácie.

3.1 Používateľ odfotí obrázok

Pomocou webovej aplikácie používateľ v teréne odfotí úlovok a v prípade záujmu aj vyplní doplňujúce informácie. Záznam sa automaticky odošle na server po pripojení k sieti wi-fi.

3.2 Domased si zobrazí úlovky na mape

Používateľ webovej aplikácie si priamo na mape zobrazí jednotlivé úlovky. Podrobné informácie zobrazí kliknutím na ikonku vybraného úlovku z mapy. V prípade záujmu môže z vlastnej iniciatívy údaje o úlovku upraviť.

3.3 Domased nahrá úlovok do mapy

Pomocou drag&drop používateľ webovej aplikácie pridá nový úlovok priamo na požadované miesto na mape. V prípade záujmu môže k úlovku napísať aj doplňujúce informácie o danom úlovku.

3.4 Registrácia a prihlásenie

Používateľ si vytvorí účet, pomocou ktorého sa môže do aplikácie prihlasovať a používať tak funkcionality len pre registrovaných používateľov. Aplikáciu môžu používať aj neregistrovaní používatelia.

4 OPIS PROTOTYPU

4.1 Funkcionálne požiadavky

Táto kapitola sa zaoberá funkcionálnymi požiadavkami, ktoré sme identifikovali po diskusiách v rámci tímu, ale hlavne s potencionálnymi používateľmi. Je rozdelená na dve logické časti podľa častí z ktorých sa celá aplikácia skladá.

4.1.1 WEBOVÁ APLIKÁCIA

- registrácia používateľa - používateľ musí byť schopný registrovať sa do aplikácie pomocou formulára vo webovom rozhraní.
- autentifikácia používateľa - systém musí byť schopný overiť správnosť údajov zadaných používateľom a na ich základe mu povoliť prípadne odoprieť prístup do aplikácie
- prezeranie mapy s vizuálnym smogom - systém musí umožňovať zobrazenie prehľadnej mapy všetkých pridaných billboardov aj s podrobnými informáciami ku každému z nich
- pridanie nového billboardu - systém musí umožňovať pridať nový billboard na ľubovoľné miesto na mape a pridať k nemu podrobné informácie vo webovom rozhraní
- spájanie rovnakých billboardov - systém musí privilegovanému používateľovi umožniť spojiť dva a viacero pridaných billboardov po zistení že sa jedná o ten istý billboard pridaný duplicitne
- zobrazenie štatistík - systém musí umožňovať zobrazenie rôznych štatistík o zozbieraných billboardoch či už globálne alebo pre jednotlivých používateľov

4.1.2 MOBILNÁ APLIKÁCIA

- odfotenie billboardu - aplikácia musí umožňovať odfoťiť fotku billboardu a zároveň jednoznačne identifikovať rámčekom billboard v prostredí fotky
- upload billboardu - aplikácia musí používateľovi umožniť nahrať fotku na server kde sa trvalo uloží do databázy spolu s meta informáciami získanými z používaného zariadenia ako napríklad GPS lokácia
- zadanie dodatočných informácií - používateľ musí byť schopný zadať informácie k odfoťenému vizuálnemu smogu bližšie špecifikujúce danú fotku (typ reklamy, zadávateľ, inzerovaný produkt)

- prezeranie billboardov - používateľovi musí byť po prihlásení umožnené prezerať si galériu vlastných sebou nahraných fotiek v prehľadnej galérii použiteľnej na mobilnom zariadení
- prihlásenie - systém musí umožňovať prihlásenie používateľa na základe údajov poskytnutých pri registrácii cez webovú časť aplikácie

4.2 Architektúra

4.2.1 SERVEROVÁ ČASŤ

V rámci serverovej časti bol použitý webový server Apache a na ňom postavený aplikačný server PHP. Serverová časť bola navrhnutá v štýle štandardnej MVC architektúry. Pôvodne sme navrhli celý MVC model od základov v čistom PHP bez použitia akéhokoľvek frameworku, čo nám prinieslo výhodu "rýchleho odpichu". Nebolo totiž potrebné zaškoľovať členov tímu na framework, s ktorým sa pred tým ešte nestretli. V neskorších štádiách projektu, kde bolo treba riešiť bezpečnostné prvky aplikácie a integráciu s databázou sa stalo použitie existujúceho frameworku priam nevyhnutným krokom nášho projektu. Nakoniec sme sa zhodli na minimalistickom frameworku CodeIgniter, ktorý disponuje iba funkcionalitou akoby presne prispôbenou potrebám nášho projektu.

RESTful rozhranie

Komunikácia medzi mobilnou aplikáciou a webovým serverom prebieha prostredníctvom RESTful rozhrania. Ide o moderný spôsob komunikácie medzi zariadeniami prostredníctvom internetu, ktorý ponúka viacero výhod, napríklad prehľadnosť, či relatívne jednoduchá implementácia.



Na príklade nižšie môžeme vidieť, akým spôsobom webový server ponúka zoznam vlastníkov billboardov. K serveru pristupujeme pomocou adresy v tvare:

/adhunter/owners/current_list/zoradenie/podla/

Výstup vo formáte JSON je možné prispôbiť niekoľkými parametrami:

- **zoradenie**
 - určuje, či sa zoznam vlastníkov zoradení zostupne, alebo vzostupne
 - prípustné hodnoty
 - asc - zoradiť vzostupne
 - desc - zoradiť zostupne
 - predvolená hodnota
 - asc
- **podla**
 - určuje stĺpec, podľa ktorého sa budú hodnoty zoradovať
 - prípustné hodnoty
 - id - ID vlastníka v databáze (čím vyššie ID, tým neskôr bol do databázy vložený)
 - name - názov/meno vlastníka
 - predvolená hodnota
 - name

CodeIgniter

Pri implementácii webovej časti nášho projektu sme sa rozhodli využiť prostredie PHP frameworku CodeIgniter. Okrem základných požiadaviek, ktoré sme pri výbere frameworku kládli na jeho funkcionality (framework založený na jazyku PHP s podporou databáz MySQL, postavený na MVC architektúre, so strmou krivkou učenia a nízkymi požiadavkami na výkon), ponúka CodeIgniter aj niekoľko črt dôležitých pre ďalšiu prácu na webovej časti projektu.

- Zabudovaná trieda pre prácu s URL adresami značne uľahčuje implementáciu RESTful rozhrania potrebného pre komunikáciu s mobilnou aplikáciou.
- Trieda pre prácu s databázou umožňuje jednoduché a bezpečné spojenie so serverom MySQL. Možnosť využiť zápis SQL požiadaviek pomocou Active Record zabezpečuje rýchlejší zápis i väčšiu zrozumiteľnosť týchto požiadaviek a rovnako i základnú ochranu pred útokmi typu SQL injection.

- Framework tiež poskytuje triedu umožňujúcu písať jednoduché unit testy, ktoré sú nevyhnutné pri vývoji riadenom testami (TDD).
- Prítomný je aj jednoduchý šablónovací systém, ktorý značne zjednodušuje a zlepšuje využitie architektúry MVC.
- V prípade nutnosti existuje celý rad rozšírení dopĺňajúcich chýbajúcu funkcionality, napr. jednoduchý objektovo-relačný mapovač.

Používateľské rozhranie

Keďže naša aplikácia bola navrhnutá tak, aby oslovila čo najväčšie spektrum používateľov, bolo potrebné prispôbiť používateľské rozhranie tak, aby prácu s ním zvládol aj človek, ktorého pôsobenie je absolútne mimo oblasť informačných technológií.

4.2.2 MOBILNÁ ČASŤ

Kostra mobilnej aplikácie

Mobilné rozhranie je vyhotovené pre platformu Android v jazyku Java. Je rozdelené na štyri obrazovky s rôznymi komponentmi navzájom tvoriacimi funkcionality mobilnej aplikácie. Informácie o typoch nosičov reklamy a vlastníkoch reklamy je potrebné doťahovať zo servera prostredníctvom rozhrania Restful.

4.3 Dátový model

4.3.1 LOGICKÝ MODEL

Ústrednú časť aktuálnej verzie nášho dátového modelu predstavujú hneď niekoľko entity. Každá používateľom vyhotovená fotografia reklamy sa v dátovom modeli reprezentuje entitou *Úlovok*. Z toho implicitne vyplýva, že fotografiu jednej konkrétnej reklamy umiestnenej na jednom nosiči môže používateľ vyhotoviť v podstate v neobmedzenom množstve a vždy sa bude jednať o jedinečný úlovok. Náš dátový model rozlišuje medzi samotnou reklamou a nosičom (reprezentovanom entitou *Nosič*), v ktorom je fyzicky umiestnená, z čoho plynú viaceré výhody.

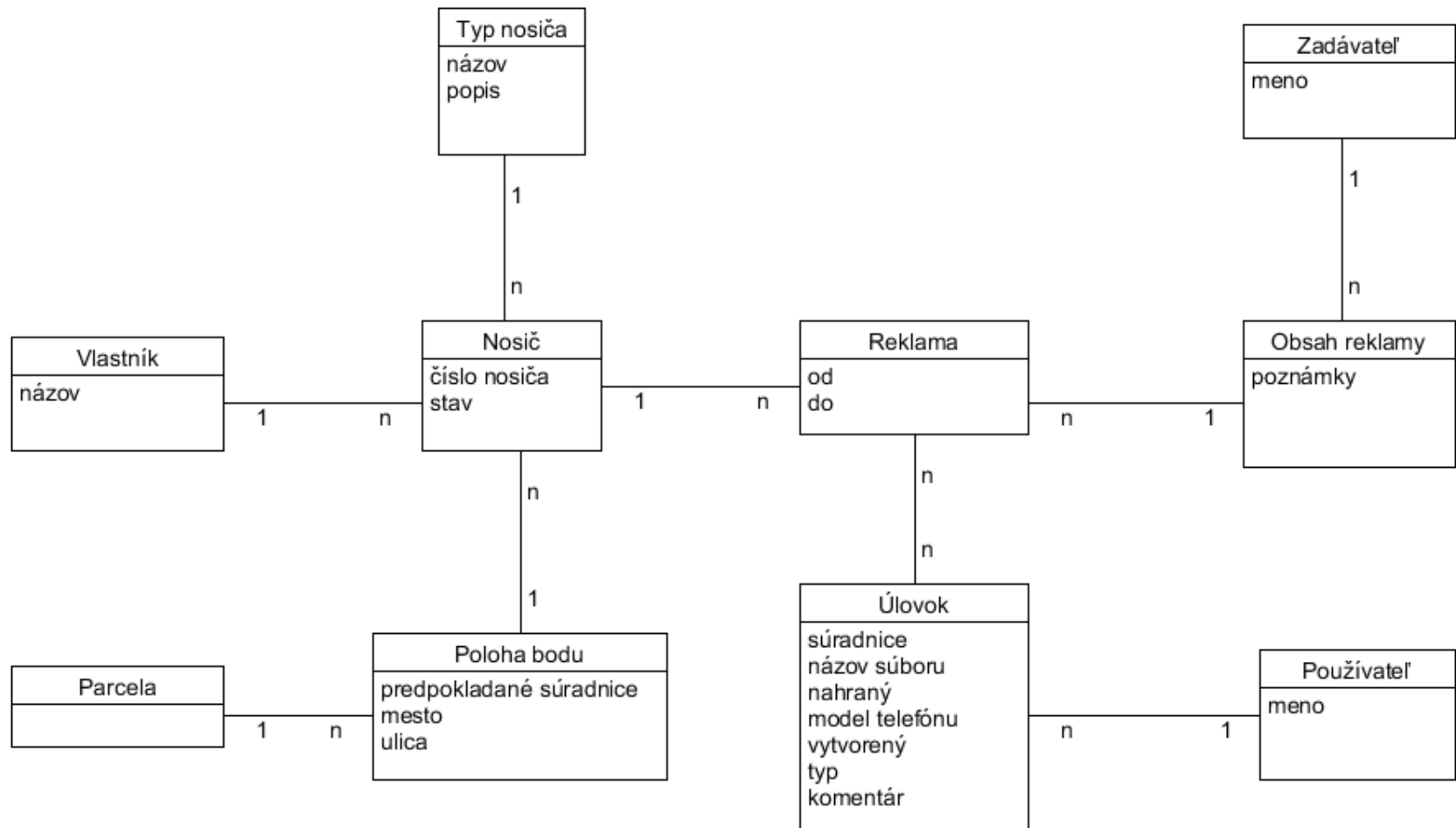
Takéto rozdelenie je však zároveň aj veľmi dôležité, ak uvažujeme časové závislosti umiestňovania reklám, pretože v rôznych časových úsekoch je možné predpokladať, že jeden nosič môže obsahovať množstvo rôznych reklám. Jedna reklama teda môže byť v modeli reprezentovaná viacerými úlovkami a jeden nosič môže mať priradených viacero reklám.

Rovnaké reklamy sú zvyčajne v rámci reklamnej kampane umiestňované na väčšom množstve nosičov. Zachytenie tejto skutočnosti môže byť zaujímavé najmä zo štatistických dôvodov, preto náš model ráta aj s takouto situáciou prepojením medzi entitami *Reklama* a *Obsah reklamy*, ku ktorej je tiež naviazaná entita *Zadávateľ*.

Aktuálna verzia modelu ráta tiež s väčším množstvom atribútov vzťahujúcim sa k nosičom.

- Entita *Typ nosiča* hovorí o spôsobe, ako sú v nosiči umiestnené reklamy, resp. aký tvar má samotný nosič, napr. „bilbord“, „megabord“.
- Entita *Vlastník* hovorí o vlastníkovi konkrétneho umiestneného nosiča. Na Slovensku existuje niekoľko najväčších prevádzkovateľov bilbordov, preto je ich zachytenie v našom modeli relatívne jednoduché.
- *Poloha bodu* určuje pravdepodobné umiestnenie nosiča z hľadiska geografických súradníc. V procese určovania skutočnej polohy môže existovať viacero pravdepodobných polôh nosiča určených na základe GPS modulu telefónov, z ktorých sa bude odhadovať finálna poloha.
- Pomocou entity *Poloha bodu* sa určuje aj parcela, na ktorej nosič leží.

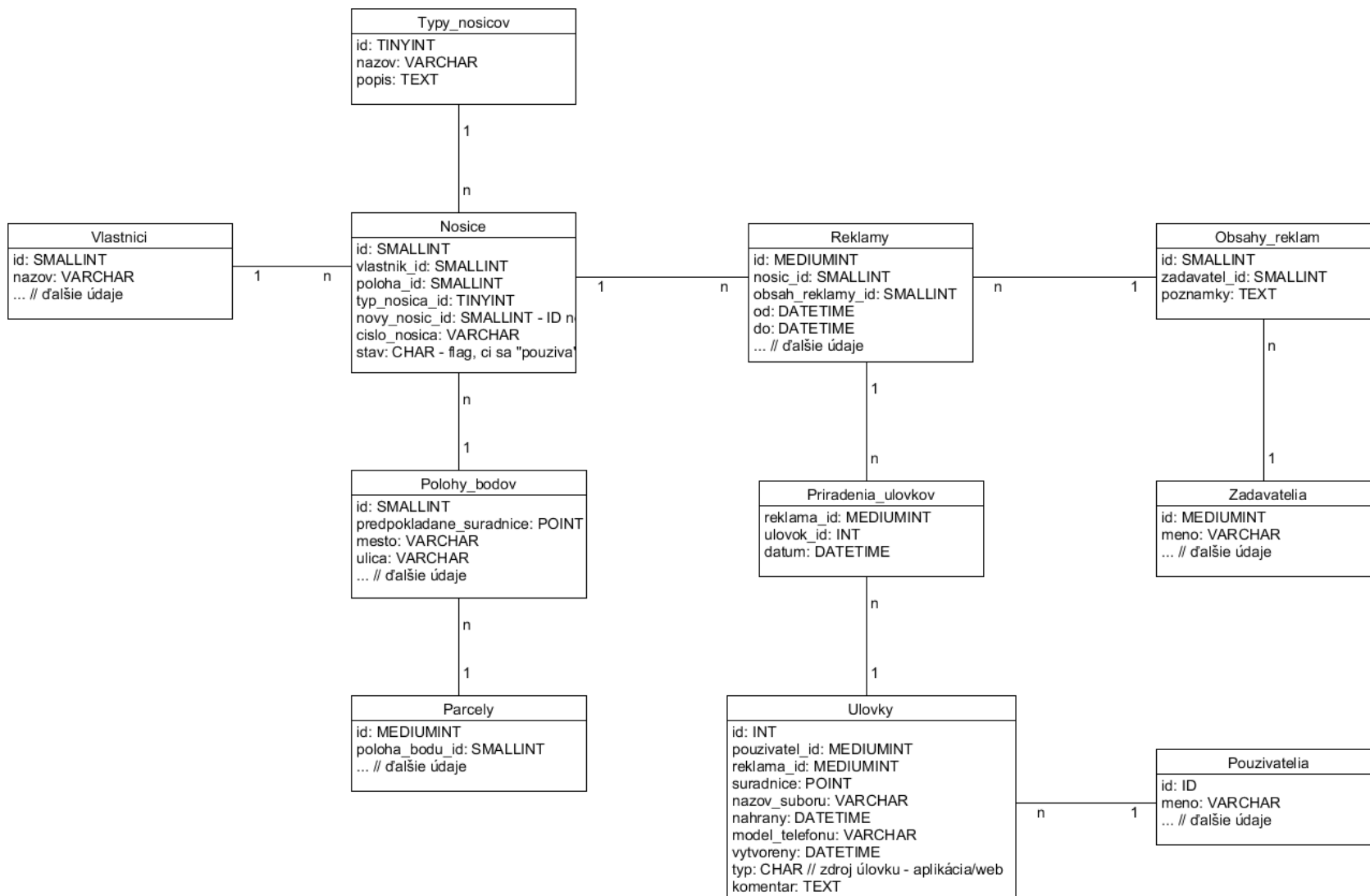
Na obrázku nižšie sa nachádza logická reprezentácia nášho dátového modelu.



4.3.2 FYZICKÝ MODEL

Vo fyzickom dátovom modeli si môžeme všimnúť realizáciu N:N väzby medzi Úlovkom a Reklamou. Tá je dosiahnutá pomocou entity *Priradenia_ulovkov*, ktorá mapuje jednotlivé úlovky na reklamy, a naopak. Príčina N:N väzby medzi týmito dvoma entitami spočíva v možnosti, že istá časť úlovkov sa kvôli nepresnosti GPS modulov priradia k nesprávnym reklamám. V záujme zachovania konzistencie nášho modelu však chcem uchovávať aj tieto nesprávne priradenia, z čoho vzniká potreba mať možnosť priradiť jeden úlovok k viacerým reklamám (úlovok sa postupom času môže priradiť k inej reklame, než tej pôvodnej), no taktiež mať priradených viac úlovkov k tej istej reklame (z rovnakého dôvodu).

Na obrázku nižšie môžeme vidieť realizáciu všetkých spomínaných entít vo fyzickom modeli.



5 MODULY

5.1 Webová aplikácia

5.1.1 REGISTRÁCIA A PRIHLASOVANIE POUŽÍVATEĽOV

Analýza

Pri návrhu tohto modulu bolo nutné uvažovať nad rôznymi aspektami registrácie a s ňou úzko súvisiaceho prihlasovania. Prvým z nich je objem údajov ktorý chceme o našich používateľoch vedieť a ktoré sú nám stále ochotní poskytnúť cez registračný formulár. Mimo nevyhnutných údajov ako sú email a heslo by mal byť povinný aj údaj o pohlaví používateľa . Je totiž zaujímavé neskôr zo zozbieraných dát analyzovať aké reklamy považujú za neestetické muži a ženy. Ďalšou otázkou bolo využívanie prezývky tzv. nicku alebo mena a priezviska daného používateľa. Na základe rozhovorov s aktivistami v oblasti vizuálneho smogu sa ako lepšia možnosť javí meno a priezvisko keďže ľudia snažiaci sa zlepšiť verejný priestor sa týmto radi verejne prezentujú a pochvália a málokedy sa radi skrývajú za prezývky.

Vzhľadom na zadávanie hesiel pri registrácii je v súčasnosti kladený dôraz na jeho čo najväčšiu bezpečnosť. Bežným štandardom v tejto oblasti je hešovanie hesla. Tento proces predstavuje aplikáciu komplexnej matematickej funkcie na zadané heslo prevedené na odlišný reťazec znakov ktorý spätne nie je prevediteľný na pôvodné heslo. Na výber bolo viacero algoritmov ktorých implementácie jazyk PHP poskytuje. Zvažovali sme funkcie MD5(Message Digest Algorithm) a funkcie typu SHA (Secure Hash Algorithm). Problémom funkcie MD5 je, že je síce rýchlejšia ako ostatné ale jej algoritmus bol prelomený je to takisto najbežnejšie používaná funkcia a preto voči nej existuje veľké množstvo tzv. dúhových tabuliek (rainbow tables) čo znamená veľkú zraniteľnosť navyše je známe že v nej ľahšie nastávajú tzv. kolízie. To v praxi znamená že dva a viacero používateľských hesiel môžu byť zahešované ako rovnaký výstupný reťazec. Funkcie typu SHA majú už niekoľko generácií. Funkcie z rodiny SHA-1 trpia menšími aj keď podobnými problémami ako MD5. Ich nástupnícke funkcie z rodiny SHA-2 sú momentálne ešte stále na dobrej úrovni aj keď aj k nim existuje lepšia alternatíva v podobe funkcii SHA-3. Tie však nie sú bežne poskytované v základnom balíku jazyka PHP a bolo by nutné používať rozširujúce balíčky.

Návrh

V module registrácie a prihlasovania sme sa na základe vyššie uvedenej analýzy rozhodli od používateľov žiadať nasledujúce údaje. Sú nimi email, heslo, meno, priezvisko a pohlavie. Email

bude zároveň vzhľadom na svoju unikátnosť slúžiť aj ako používateľské meno pre vstup do aplikácie. Za hešovací algoritmus bola zvolená funkcia z rodiny SHA-2 a to konkrétne SHA512 čo prináša rozumný kompromis medzi bezpečnosťou a zložitosťou implementácie. Pre zvýšenie bezpečnosti tohto algoritmu bude použitá tzv. salt ktorá bude náhodne generovaná unikátne pre každého používateľa a bude spolu s jeho ďalšími údajmi uložená v databáze znemožňujúca prípadne pokusy o prelomenie hesla na základe tabuliek.

Implementácia

Modul bol implementovaný na základe navrhnutého riešenia. Používateľ zadá všetky požadované údaje do bežného formulára poskytovaného jazykom PHP. Následne údaje prejdú validáciou ako napríklad na správny formát emailovej adresy (xx@yy.zz) a je takisto skontrolovaná prítomnosť všetkých povinných údajov. Následne sa pomocou zabudovaného generátoru náhodných 32bitových reťazcov vygeneruje pre nového používateľa jeho unikátny salt. takto vytvorený salt je pripojený k heslu a následne zahešovaný pri návrhu zvolenou funkciou SHA512. Zahešované heslo a salt spolu s ostatnými zadanými údajmi sa následne uložia do databázy.

Pri autentifikácii používateľa je opäť využitý štandardný PHP formulár. Formulár obsahuje textové polia meno a heslo pričom menom je používateľov email zadaný pri registrácii a ten je opäť validovaný. Následne je z databázy vytiahnutá salt a rovnakým postupom ako pri registrácii je vytvorený hash. tento hash je následne porovnávaný so záznamom uloženým v databáze a na základe výsledku porovnania je používateľ oprávnený vstúpiť do aplikácie alebo mu je prístup odopretý.

Testovanie

Pre potreby overenia správnej funkčnosti tohto modulu bolo implementovaných niekoľko testov. testy sa zameriavali na testovacie vytváranie používateľov a následné overovanie správnosti uloženia údajov do databázy. V neposlednom rade boli napísané testy overujúce správnu autentifikáciu už registrovaných používateľov na základe správnych či nesprávnych prihlasovacích údajov.

5.1.2 MAPA

Analýza

Pri výbere API pre našu mapu sme uvažovali medzi slovenským poskytovateľom mapy.sk, ktorý v dnešnej dobe spadá pod spoločnosť Atlas a mapy majú od spoločnosti eurosense s.r.o. Napriek tomu, že ide o slovenského poskytovateľa, k dispozícii sú aj mapy krajín celej Európy.

API daného poskytovateľa je však veľmi zle zdokumentovaná a pre developera je k dispozícii relatívne málo funkcií.

Ďalšou možnosťou bola použitie API od poskytovateľa Google Maps. Nakoľko v pozadí tohto projektu stojí gigant Google, je tu množstvo výhod oproti slovenským mapy.sk. Tím vývojárov ustavične pracuje na nových vychytávkach a odstraňovaní chýb. Celá API je výborne zdokumentovaná, pričom je k dispozícii množstvo tutoriálov, ktoré prispievajú k rýchlejšiemu pochopeniu celej API. Oproti spomínaným mapy.sk sú omnoho detailnejšie, sú na nich zobrazené zastávky MHD, ku významným budovám sú častokrát doložené aj fotografie z okolia, recenzie na daný objekt a otváracie hodiny. Takéto detailné zobrazenie je pre náš projekt dôležité nakoľko používateľ si častokrát nepamätá presnú adresu, na ktorej odfotil neestetický billboard, pamätá si, že to bolo v blízkosti nejakej autobusovej zastávky prípadne nejakej významnej budovy. To bolo kľúčové pri výbere technológie, a preto sme sa nakoniec rozhodli pre mapy od spoločnosti Google.

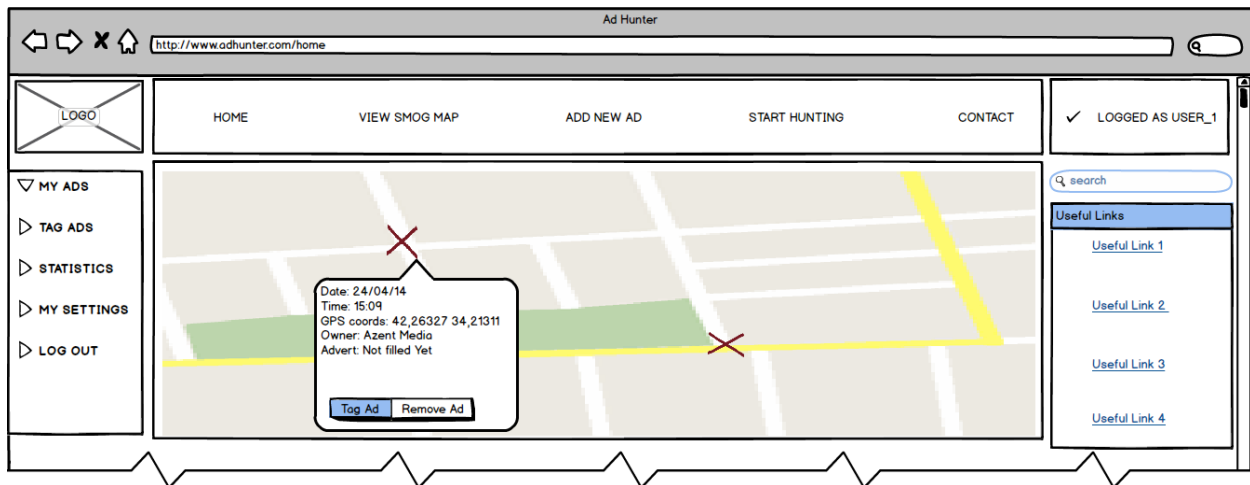
Bonusom k nášmu výberu sa stali možnosti ako napríklad street view, kde sa používateľ môže prechádzať ulicami a vidí tak všetko z pohľadu chodca. Táto možnosť je obzvlášť užitočná v našom projekte. Uvedme teda konkrétny príklad z terénu. Používateľ vystúpi na autobusovej zastávke prejde cez 2 ulice a hneď za zeleným domom nájde neestetický billboard. V štandardnej mape pri pohľade zhora sa však nevie zorientovať, tu však siahne po funkcii street view, nasimuluje si cestu, ktorú prešiel a na konkrétnom mieste zaznačí výskyt billboardu. Takto zabezpečíme, aby sme nestrácali úlovy od používateľov, ktorí majú problém zorientovať sa na bežnej satelitnej mape.

Do úvahy prišla aj myšlienka implementácie vlastnej mapy, to by však ale bolo časovo veľmi náročne a neefektívne. Google mapy je možné implementovať dvomi spôsobmi. Prvý je jednoduché vloženie vygenerovaného iframe. Takáto implementácia však ponúka iba obmedzenú funkcionality a ide o rapídne pomalšie riešenie, čo sa týka času potrebného na kompletne načítanie stránky. Nad kódom v iframe totižto nie je možné vykonávať vlastný JavaScript kód z bezpečnostných dôvodov, aby sa predišlo cross-site útokom. Podobne nie je možné ani dodatočne štýlovať kód v iframe pomocou jazyka CSS.

Druhým spôsobom pre implementáciu google máp je pomocou JavaScriptu. Ide o náročnejšiu možnosť v rámci implementácie, prináša však viacero výhod. Je tu možné vykonávanie vlastného JavaScript kódu a používanie vlastných CSS štýlov, čo možno využiť napríklad pri zmene markerov v mape, prípadne ovládacích prvkov mapy.

Návrh

Pre potreby nášho projektu sa ukázali byť najlepšou možnosťou mapy od spoločnosti Google, pričom je potrebná implementácia aj dodatočných funkcií ako napríklad street view, press & drag a podobne. Ovládanie mapy musí byť jednoduché a intuitívne, ideálne s prvkami, na ktoré je používateľ zvyknutý z používania webovej aplikácie maps.google.sk. Na mape musia byť prehľadne zobrazené zozbierané billboardy, ideálne ikonou, ktorá by ich jasne vystihovala.



Implementácia

Naše mapy boli nakoniec implementované pomocou API ponúkané spoločnosťou Google. Vzhľadom na množstvo výhod sme ich implementovali pomocou technológie JavaScript na strane klienta a na strane servera je využitý jazyk PHP. Zo strany servera sú najprv posielané geodata vo forme JSON. Následne sa na strane klienta spracujú zaznačia do mapy. Vykreslenie mapy prebieha zavolaním externého JavaScriptu, umiestneného na serveroch spoločnosti Google. Po úspešnom vykonaní daného skriptu sa vykoná definovaná callback funkcia, konkrétne v našom projekte ide o funkciu `initMap()`. V spomenutej callback funkcii je následne definovaný štýl zobrazenia našej mapy, rozloženie a správanie jednotlivých ovládacích prvkov a samotných billboardov. Následne sa spracuje získaný objekt v tvare JSON a naplní sa tak naša mapa billboardmi z databázy. Pridávanie billboardu prebieha jednoduchým zaznačením miesta na mape, pričom používateľovi je zobrazený sprievodný text. Po zaznačení polohy na mape je používateľovi zobrazený formulár, v ktorom zadá dodatočné informácie o danom billboarde a tie sú následne odosielané na server formou POST requestu. Na serveri sa následne spracujú prijaté dáta a uložia do databázy v požadovanom tvare. Naša mapa ponúka funkcionalitu pre press & drag, čiže používateľ sa vie jednoducho pohybovať po mape bez použitia klávesnice prípadne smerových tlačidiel na mape. Priblíženie a oddialenie mapy je možné taktiež pomocou

štandardného ovládacieho prvku, prípadne pomocou kolieska myši. Samozrejmosťou je aj podpora funkcie street view.

Testovanie

Počas vývoja bola vykonaných niekoľko unit testov, ktoré boli zamerané na predídenie rôznym typom útokom vo webovom prostredí a na korektné spracovanie dát z databázy, prípadne na korektný zápis dát do databázy. Jeden z unit testov bol pridaním billboardu s názvami obsahujúcimi predpripravený kód pre útok typu SQL injection, ďalšie testy boli na strane klienta zamerané na kontrolu odpovedí(response) zo servera. V rámci finálneho testovania bolo vykonaných niekoľko skúšobných pridaní billboardu, zamerané na hraničné prípady vzhľadom na vytvorený zdrojový kód.

5.1.3 UPLOAD OBRÁZKA

Analýza

Pri uploade obrázka je dôležité určiť v akom formáte budú dáta posielané na server a následne spracované. Vyskytlo sa viacero možností pričom bolo potrebné dbať na objem prenášaných dát, keďže dnešné moderné fotomobily vytvárajú fotografie o veľkosti niekoľkých megabajtov. Keďže bolo potrebné myslieť aj na to, že naša aplikácia pozostáva aj z mobilnej časti, ktorá je implementovaná ako aplikácia v jazyku JAVA, bolo teda potrebné nájsť riešenie, ktoré by fungovalo aj vo webovej aplikácii aj v mobilnej. Vo webovom prostredí je možné dáta prijímať v týchto formátoch:

- `application/x-www-form-urlencoded`: ide o defaultný formát pre posielané dáta cez web formuláre. Pri tomto formáte sú medzery v posielaných dátach konvertované na znak '+' a špeciálne znaky sú konvertované na ich ascii hodnotu v hexadecimálnom kóde. Nevýhodou je, že sa konvertuje dosť veľa znakov, čo má za následok dlhší čas na spracovanie dát a výsledný objem konvertovaných dát je relatívne veľký.
- `multipart/form-data`: tento formát dát sa odporúča pri spracovaní rôznych typov súborov cez formuláre. Parametre sú posielané ako čisté dáta bez akéhokoľvek konvertovania, čo má za následok kratší čas, potrebný na spracovanie zasielaných dát. Keďže pri tomto formáte neprebíha žiadna konverzia špeciálnych znakov na sekvenciu čísel určujúcich ich ASCII hodnotu, objem dát je podstatne nižší ako pri prvom spomínanom formáte.
- `text/plain`: pri tomto formáte sú medzery konvertované na znak '+' podobne ako pri formáte `application/x-www-form-urlencoded`, avšak špeciálne znaky nie sú konvertované.

Z

HTML5

špecifikácie

(<http://www.w3.org/TR/html5/forms.html#text/plain-encoding-algorithm>) vyplýva, že tento formát sa používa iba na účely debugovania zasielaných dát.

- base64 reťazec - v prípade zasielania textov a reťazcov prostredníctvom web formulárov, je tu možnosť aj každý obrázok prekonvertovať na base64 reťazec a následne poslať na server iba textové parametre. Pri tejto možnosti však treba zobrať do úvahy, že objem prenášaných dát je podstatne vyšší než pri hociktorom inom formáte a rovnako aj čas potrebný spracovanie dát je podstatne dlhší. Výhodou je jednoduchá implementácia či už v mobilnej alebo webovej aplikácii.

V mobilnej aplikácii je možné dáta odosielať v ľubovoľnom formáte, niektoré sú natívne podporované, na iné je potrebné použiť externú knižnicu.

Návrh

Primárnym faktorom pri voľbe formátu dát bola rýchlosť uploadu nasnímanej fotografie. Bežného používateľa mobilnej aplikácie by mohlo odradiť od používania našej aplikácie neúmerne dlhé čakanie na upload jedinej fotografie. Preto sme pri výbere brali ohľad na množstvo odosielaných dát na server a rýchlosť spracovania dát. Ako najlepšou voľbou sa ukázalo upload fotografie a k nej prislúchajúcich údajov prostredníctvom formátu multipart/form-data. Dôležitým faktorom pri implementácii však musí byť konzistencia medzi jednotlivými modulmi, z mobilnej i webovej aplikácie teda musia byť zasielané dáta v rovnakom formáte.

Implementácia

Do existujúceho kontrolera billboards bola implementovaná funkcia add, ktorá pri volaní spomínaného kontrolera zistí, či ide o request uploadu obrázka. Ak áno, je potrebné skontrolovať, či sú splnené podmienky pre uloženie fotografie na server (pričínok musí existovať a musíme mať práva doňho zapisovať), následne je nutné overiť, či boli zadané potrebné informácie k danej fotografii. Pri spracovaní zaslaných parametrov sa vykoná validácia na strane servera aby sa predišlo rôznym typom útokom. Ak sú všetky podmienky splnené, do priečinka assets/pics sa uloží zaslaná fotografia s názvom očisteným od špeciálnych znakov a znakov s diakritikou. Následne sa zavolá funkcia save_ulovek pre model Ulovok_model, ktorá uloží dodatočné informácie ku konkrétnemu úlovku do databázy. Ak upload fotografie prebehol úspešne, zavolá sa view uploaded_billboard, ktorý používateľa informuje o úspešnom pridaní jeho úlovku.

Testovania

V rámci testovania bolo odoslaných niekoľko fotografií, či už vo webovom prostredí alebo z mobilnej aplikácie a následne ich naša webová aplikácia úspešne spracovala.

5.2 Android aplikácia

5.2.1 ZÁZNAM NEESTETICKEJ REKLAMY

Analýza

Aplikácia má slúžiť primárne na zachytávanie a ukladanie záznamov neestetickéj reklamy na server, kde sa budú zhromažďovať do databázy. Prvotná myšlienka bola, aby rozhranie pre používateľa bolo navrhnuté tak, aby s ním vedel rýchlo a intuitívne manipulovať. Na úvodnej obrazovke sme sa teda rozhodli používateľovi ihneď poskytnúť možnosť fotenia, nakoľko toto je jeho primárnym záujmom pri spustení mobilnej aplikácie. Používateľ teda môže zamieriť na cieľ, neestetickú reklamu, a odfotiť ho intuitívnym tlačidlom fotenia.

Pred uploadnutím fotky je vhodné, aby mal používateľ možnosť zadať rozširujúce informácie o reklame, ako napríklad typ nosiča reklamy alebo vlastníka reklamy. Túto obrazovku ponúkžeme používateľovi, pričom bude mať možnosť rozhodnutia, či dané informácie chce zadať alebo nie. Ak nie, nechceme v používateľovi vyvolať pocit "nekompletnosti záznamu" a teda rozširujúce informácie budú ponúknuté len nenútenou formou.

Používateľ takisto nemusí súhlasiť so záznamom, ktorý odfotil, a v tomto prípade by mal dostať možnosť zopakovať svoj pokus o zachytenie záznamu. Táto možnosť sa mu naskytne bezprostredne po odfotení a bude sa môcť rozhodnúť, či fotenie opakovať, alebo postúpiť k uploadu záznamu.

Ak sa používateľ ocitne mimo pripojenia na internet, je potrebné poskytnúť mu možnosť neskoršieho uploadu záznamu. Nie všade v teréne je totiž dostupný signál mobilného internetu a taktiež nie každý používateľ našej aplikácie bude mať k dispozícii tento spôsob pripojenia. Fotka teda počká uložená v mobilnom telefóne, až kým používateľ nemá možnosť pripojenia na internet. O čom bude upovedomený pomocou notifikácie.

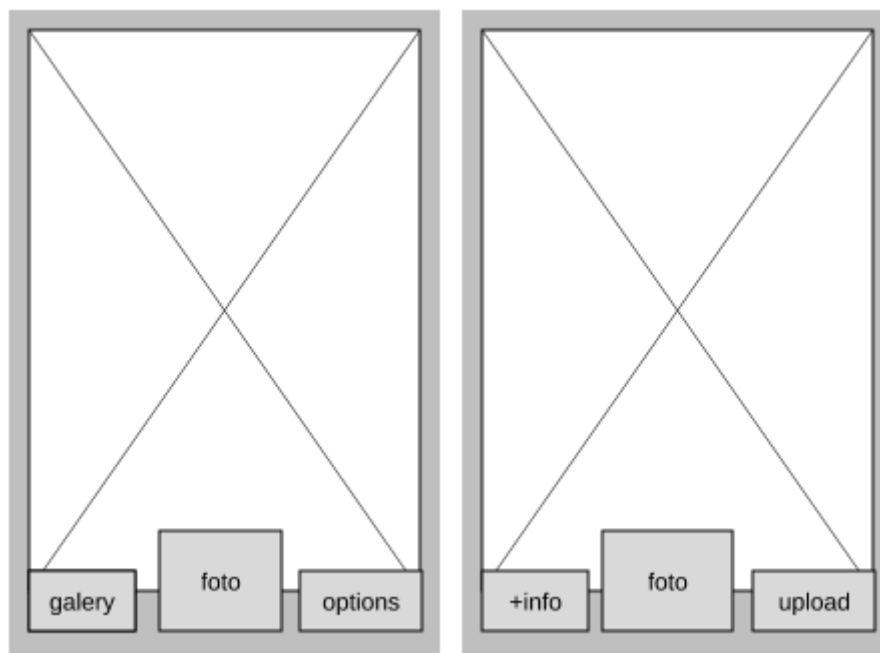
Návrh

Po odfotení fotky používateľom sa daná fotka uloží do samostatného priečinka. Následne má používateľ možnosť rozhodnúť sa, či odfotenu fotku, ktorá sa mu vyobrazí, považuje za vhodnú na odoslanie, a teda či ju chce odoslať alebo nie. Ak nechce, má možnosť vrátiť sa späť a

pokračovať vo foteaní. Taktiež v prípade záujmu môže používateľ pridať rozširujúce informácie o fotke, ktoré ju pomôžu lepšie špecifikovať.

Pri foteaní reklám v teréne používateľ môže a nemusí mať pripojenie na internet. Aby aplikácia neodradila používateľov, ktorí majú väčšinu času svoje mobilné zariadenia v offline režime, aplikácia bezproblémovo funguje aj bez pripojenia na internet, avšak s tým rozdielom, že fotky sa ukladajú do samostatného priečinka, pričom pri najbližšom pripojení na internet je používateľ prostredníctvom notifikácie upovedomený o existujúcich neodoslaných fotkách a má ich možnosť odoslať.

Low-Fidelity návrhy obrazoviek mobilnej aplikácie:



vlastník nosiča:	<input type="button" value="vyber >"/>	
typ nosiča:	<input type="button" value="vyber >"/>	
poznámka:	<input type="text"/>	
<input type="button" value="-info"/>	<input type="button" value="foto"/>	<input type="button" value="upload"/>

typ vlastníka 1	typ vlastníka 2	
typ vlastníka 3	typ vlastníka 4	
typ vlastníka 5	typ vlastníka 6	
iný:	<input type="text"/>	
<input type="button" value="spät'"/>	<input type="button" value="foto"/>	<input type="button" value="upload"/>

Implementácia

Aplikácia je implementovaná vo vývojom prostredí Android Studio, napísaná v programovacom jazyku Java, pričom UI je definované pomocou jazyka XML. Hlavnú časť aplikácie tvorí kamera. Nakoľko sme však potrebovali vyvinúť kameru "šitú na mieru", použili sme Android Camera API, ktorého použitie je vhodnou alternatívou k použitiu štandardnej vstavanej kamerovej aplikácie, pretože sme tým získali možnosť vytvoriť plne modifikovateľnú kamerovú aplikáciu.

Druhú nemenej dôležitú časť predstavuje komunikácia aplikácie so serverom. Táto komunikácia prebieha prostredníctvom REST služieb. Pri každom uploade fotky sa odosiela POST request zo strany mobilnej aplikácie, pričom server je pripravený na prijatie potrebných dát (ktoré zahŕňajú samotný bytestream fotky (vo formáte multipart/form-data) spolu so súradnicami miesta odfotoenia a dodatočnými informáciami).

Aplikácia získava súradnice miesta odfotoenia prostredníctvom GPS senzora, ktorý je zabudovaný v každom súčasnom Android telefóne. Nakoľko však funkcia lokalizácie štandardne nebýva zapnutá, používateľ je vyzvaný túto funkciu povoliť pri každom štarte aplikácie (pokiaľ už, samozrejme, daná funkcia nie je povolená).

Kvôli zjednodušeniu vývoja sme spodnú hranicu podpory nastavili na verziu Android 3.0 (čiže aplikácia podporuje všetky funkcie pre API 11+), nakoľko pre staršie verzie je pri implementácii potrebné využívať viaceré podporné funkcie.

Zdrojový kód aplikácie dodržiava zásady objektovo-orientovaného programovania, pričom zachováva pravidlá písania kódu pre Android podľa oficiálnej Android dokumentácie a taktiež zohľadňuje metodiku správneho písania zdrojového kódu, ktorá je súčasťou našich interných tímových metodík.

Po odfotení fotky užívateľom sa daná fotka uloží do samostatného priečinka. Následne je užívateľ vyzvaný rozhodnúť sa, či danú fotku považuje za vhodnú na odoslanie, a teda či ju chce odoslať alebo nie. Ak nechce, má možnosť zrušiť dialógové okno a pokračovať vo foteaní.

Pri foteaní reklám v teréne používateľ môže a nemusí mať pripojenie na internet. Aby aplikácia neodradila užívateľov, ktorí majú väčšinu času svoje mobilné zariadenia v offline režime, aplikácia bezproblémovo funguje aj bez pripojenia na internet, avšak s tým rozdielom, že fotky sa ukladajú do samostatného priečinka, pričom pri najbližšom pripojení na internet je používateľ prostredníctvom notifikácie upovedomený o existujúcich neodoslaných fotkách a má ich možnosť odoslať.

6 TECHNICKÁ DOKUMENTÁCIA

- Vygenerovaná dokumentácia mobilnej časti (Javadoc)
- Vygenerovaná dokumentácia webovej časti (phpDoc)

com.vcelicky.smog

Class BaseActivity

```
java.lang.Object
  Activity
    com.vcelicky.smog.BaseActivity
```

Direct Known Subclasses:

CameraActivity, OnlineActivity, TakenActivity

```
public class BaseActivity
extends Activity
```

Created by Jerry on 10. 10. 2014. Activity that all the other activities inherit from.

Field Summary

Fields

Modifier and Type	Field and Description
protected Location	mCurrentLocation
protected LocationManager	mLocationManager

Constructor Summary

Constructors

Constructor and Description
BaseActivity ()

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
static boolean	isWiFiConnected (Context context)
boolean	isWifiOrMobileConnected (Context context) Checks current connectivity status of device.
protected void	onCreate (Bundle savedInstanceState)
boolean	onCreateOptionsMenu (Menu menu)
void	onLocationChanged (Location location)
boolean	onOptionsItemSelected (MenuItem item)
void	onProviderDisabled (java.lang.String s)
void	onProviderEnabled (java.lang.String s)
void	onStatusChanged (java.lang.String s, int i, Bundle bundle)
void	requestLocationUpdate () Requests location update.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

mLocationManager

protected LocationManager mLocationManager

mCurrentLocation

protected Location mCurrentLocation

Constructor Detail

BaseActivity

```
public BaseActivity()
```

Method Detail

onCreate

```
protected void onCreate(Bundle savedInstanceState)
```

onCreateOptionsMenu

```
public boolean onCreateOptionsMenu(Menu menu)
```

onOptionsItemSelected

```
public boolean onOptionsItemSelected(MenuItem item)
```

isWiFiConnected

```
public static boolean isWiFiConnected(Context context)
```

Parameters:

context - Context of calling activity.

Returns:

true if WiFi is enabled and connected, otherwise false

isWifiOrMobileConnected

```
public boolean isWifiOrMobileConnected(Context context)
```

Checks current connectivity status of device.

Parameters:

context - Context of calling activity.

Returns:

true if WiFi or Mobile internet connection is established, false if not

requestLocationUpdate

```
public void requestLocationUpdate()
```

Requests location update. Initializes LocationManager and current location. If current location is not yet available, last known location is used.

onLocationChanged

```
public void onLocationChanged(Location location)
```

onStatusChanged

```
public void onStatusChanged(java.lang.String s,
                             int i,
                             Bundle bundle)
```

onProviderEnabled

```
public void onProviderEnabled(java.lang.String s)
```

onProviderDisabled

```
public void onProviderDisabled(java.lang.String s)
```

com.vcelicky.smog

Class CameraActivity

```

java.lang.Object
  Activity
    com.vcelicky.smog.BaseActivity
      com.vcelicky.smog.CameraActivity
  
```

```

public class CameraActivity
  extends BaseActivity
  
```

Created by jerry on 10. 10. 2014.

Field Summary**Fields**

Modifier and Type	Field and Description
static java.lang.String	DIRECTORY_MAIN
static java.lang.String	DIRECTORY_UPLOAD
static int	MEDIA_TYPE_COMPRESSED
static int	MEDIA_TYPE_IMAGE

Fields inherited from class com.vcelicky.smog.BaseActivity

mCurrentLocation, mLocationManager

Constructor Summary**Constructors**

Constructor and Description
CameraActivity ()

Method Summary**All Methods** **Static Methods** **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
static Camera	getCameraInstance () Initializes Camera object by opening Camera.
protected void	onCreate (Bundle savedInstanceState)
protected void	onPause ()
protected void	onStart ()
protected void	onStop ()

Methods inherited from class com.vcelicky.smog.BaseActivity

isWifiConnected, isWifiOrMobileConnected, onCreateOptionsMenu, onLocationChanged, onOptionsItemSelected, onProviderDisabled, onProviderEnabled, onStatusChanged, requestLocationUpdate

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail**MEDIA_TYPE_IMAGE**

public static final int MEDIA_TYPE_IMAGE

See Also:

Constant Field Values

MEDIA_TYPE_COMPRESSED

public static final int MEDIA_TYPE_COMPRESSED

See Also:

Constant Field Values

DIRECTORY_MAIN

public static final java.lang.String DIRECTORY_MAIN

See Also:

Constant Field Values

DIRECTORY_UPLOAD

public static final java.lang.String DIRECTORY_UPLOAD

See Also:

Constant Field Values

Constructor Detail**CameraActivity**

public CameraActivity()

Method Detail**onCreate**

protected void onCreate(Bundle savedInstanceState)

Overrides:

onCreate in class BaseActivity

onStart

protected void onStart()

onStop

protected void onStop()

onPause

protected void onPause()

getCameraInstance

public static Camera getCameraInstance()

Initializes Camera object by opening Camera.

Returns:

if success, then initialized Camera

com.vcelicky.smog

Class CameraPreview

```

java.lang.Object
  SurfaceView
    com.vcelicky.smog.CameraPreview

```

```

public class CameraPreview
  extends SurfaceView

```

Created by jerry on 10. 10. 2014. A basic Camera Preview class.

Constructor Summary

Constructors

Constructor and Description

CameraPreview(Context context, Camera camera)

Initializes Camera object and installs a SurfaceHolder.Callback in order to get notified when the underlying surface is created and destroyed.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

void	surfaceCreated (SurfaceHolder surfaceHolder)
void	surfaceDestroyed (SurfaceHolder holder)
void	surfaceChanged (SurfaceHolder holder, int format, int w, int h)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CameraPreview

```

public CameraPreview(Context context,
                    Camera camera)

```

Initializes Camera object and installs a SurfaceHolder.Callback in order to get notified when the underlying surface is created and destroyed.

Parameters:

context - Context of given activity.
camera - Camera object (open Camera).

Method Detail

surfaceCreated

```

public void surfaceCreated(SurfaceHolder surfaceHolder)

```

surfaceChanged

```

public void surfaceChanged(SurfaceHolder holder,
                          int format,
                          int w,
                          int h)

```

surfaceDestroyed

```

public void surfaceDestroyed(SurfaceHolder holder)

```

com.vcelicky.smog

Class NetworkChangeReceiver

java.lang.Object

BroadcastReceiver

com.vcelicky.smog.NetworkChangeReceiver

```
public class NetworkChangeReceiver
```

```
extends BroadcastReceiver
```

Created by jerry on 20. 10. 2014.

Constructor Summary

Constructors

Constructor and Description

`NetworkChangeReceiver ()`

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

void `onReceive`(Context context, Intent intent)

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Detail

NetworkChangeReceiver

```
public NetworkChangeReceiver ()
```

Method Detail

onReceive

```
public void onReceive(Context context,  
                    Intent intent)
```

com.vcelicky.smog

Class TakenActivity

java.lang.Object

Activity

com.vcelicky.smog.BaseActivity

com.vcelicky.smog.TakenActivity

```
public class TakenActivity
extends BaseActivity
```

Created by jerry on 13. 11. 2014. Class to process taken picture.

Field Summary

Fields inherited from class com.vcelicky.smog.BaseActivity

mCurrentLocation, mLocationManager

Constructor Summary

Constructors

Constructor and Description

TakenActivity ()

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method and Description
protected void	onCreate (Bundle savedInstanceState)

Methods inherited from class com.vcelicky.smog.BaseActivity

isWifiConnected, isWifiOrMobileConnected, onCreateOptionsMenu, onLocationChanged, onOptionsItemSelected, onProviderDisabled, onProviderEnabled, onStatusChanged, requestLocationUpdate

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TakenActivity

public TakenActivity()

Method Detail

onCreate

protected void onCreate(Bundle savedInstanceState)

Overrides:

onCreate in class BaseActivity

- Billboards
- Login_Registratio
- Login_registration
- MY_Loader
- Owner_model
- Owners
- Registration_mode
- Ulovok_model

\ ↑

Billboards

obsahuje funkcie pre pracu s billboardami

Summary

Methods	Properties	Constants
<ul style="list-style-type: none"> ✓ index() show() get_ulovky() add() 	No public properties found	No constants found
<ul style="list-style-type: none"> 🛡 No protected method... 	No protected properties ...	N/A
<ul style="list-style-type: none"> 🔒 _clean_name() _get_filename() 	No private properties fo...	N/A

Methods

✓ index()

```
index()
```

funkcia zobrazi view s hlavnou ponukou

✓ show()

```
show()
```

funkcia zobrazi view s mapou vsetkych billboardov

✓ get_ulovky()

```
get_ulovky() : object
```

funkcia vracia zoznam vsetkych ulovkov

Returns

object —
\$json funkcia vracia zoznam vsetkych ulovkov aj s dodatocnymi informaciami

✓ add()

```
add()
```

funkcia sluzi na pridanie billboardu

tato funkcia je volana z webovej i mobilnej aplikacie, pricom sa vzdy vratia data rovnakeho formatu vo funkcii sa spracovavaju globalne premenne \$_POST["lat"], \$_POST["lng"] a \$_FILES["photo"]

- v pripade, ze je funkcia volana bez potrebnych parametrov, zobrazi sa view pre pridanie billboardu vo webovom prostredi

🔒 _clean_name()

```
_clean_name(string $name) : string
```

funkcia na ocistenie nazvu suboru

funkcia ocisti meno suboru od specialnych znakov, znak medzery je nahradeny znakom '-' a znaky s diakritikou su nahradene ich ekvivalentom bez diakritiky (á => a)

Parameters

string \$name nazov suboru obsahujuci zakazane znaky

Returns

string —
funkcia vracia nazov suboru ocisteneho od specialnych znakov

🔒 _get_filename()

```
_get_filename( $folder, string $name) : string
```

funkcia vracia nazov uploadovaneho suboru

funkcia ocisti meno suboru a ak sa na serveri subor s danym nazvom uz nachadza prida mu prisluchajuci postfix (_2)

Parameters

	\$folder	
string	\$name	nazov uploadovaneho suboru

Returns

string —
nazov uploadovaneho suboru

FILE
controllers/billboards.php

CLASS HIERARCHY

- \CI_Controller
- \Billboards

Tags

None found



Tags

None found



Tags

None found



Tags

None found



Tags

None found



Tags

None found



Tags

None found



Class Hierarchy Diagram

Errors Markers

- ▼ \
- ⊞ Billboards
- ⊞ Login_Registrati
- ⊞ Login_registration
- ⊞ MY_Loader
- ⊞ Owner_model
- ⊞ Owners
- ⊞ Registration_mode
- ⊞ Ulovok_model

Login_Registration

Summary

Methods	Properties	Constants
<ul style="list-style-type: none"> ✔ login() register() hash_password() generate_salt() add_user() authenticate_user() 	No public properties found	No constants found
<ul style="list-style-type: none"> 🛡 No protected method... 	No protected properties ...	N/A
<ul style="list-style-type: none"> 🔒 No private methods f... 	No private properties fo...	N/A

Methods

✔ login()

```
login()
```

Funkcia zobrazi view login s prihlasovacim formularom

✔ register()

```
register()
```

Funkcia zobrazi view register s registracnym formularom

✔ hash_password()

```
hash_password( $rawPassword, $salt) : string
```

Funkcia vyuziva standardnu hashovacu funkciu md5 na hashovanie pouzivatskeho hesla pred vlozenim do databazy

Parameters

\$rawPassword

\$salt

Returns

string —
hashedPassword zahesovane heslo

✔ generate_salt()

```
generate_salt( $max) : string
```

Funkcia generuje nahodny x bitovy retazec tzv salt ktore znemoznuje prelamovanie hesla pomocou duhovych tabuliek retazec je zlozeny z vopred zadanych ASCII znakov

Parameters

\$max

Returns

string —
salt vygenerovany max znakovy retazec

✔ add_user()

```
add_user()
```

Funkcia parsuje udaje zadane pouzivatelom vo formulari zavola modelovu funkciu na ulozenie udajov do databazt a na zaver zobrazi udaj o uspesnosti registracie

✔ authenticate_user()

```
authenticate_user()
```

Funkcia overujuca spravnost zadanych prihlasovacich udajov na zaklade udajov poskytnutych v prihlasovacom formulari pokiaľ su spravne zobrzy view o uspesnosti prihlaska v opacnom pripade o neuspesnosti

FILE

controllers/login_registration.php

CLASS HIERARCHY

- ▶ \CI_controller
- ▶ \Login_Registration

Tags

None found



Tags

None found



Tags

None found



Tags

None found



Tags

None found



Tags

None found



Tags

None found



Class Hierarchy Diagram

Errors
Markers

Login_registration_model

Summary

Methods	Properties	Constants
<ul style="list-style-type: none"> ✓ <code>__construct()</code> <code>get_password_for_...</code> <code>save_user()</code> 	<ul style="list-style-type: none"> <i>No public properties found</i> 	<ul style="list-style-type: none"> <i>No constants found</i>
<ul style="list-style-type: none"> 🛡️ <i>No protected method...</i> 	<ul style="list-style-type: none"> <i>No protected properties ...</i> 	<ul style="list-style-type: none"> <i>N/A</i>
<ul style="list-style-type: none"> 🔒 <i>No private methods f...</i> 	<ul style="list-style-type: none"> <i>No private properties fo...</i> 	<ul style="list-style-type: none"> <i>N/A</i>

Methods

✓ `__construct()`

```
__construct()
```

✓ `get_password_for_login()`

```
get_password_for_login( $email ) : string
```

Modelova funkcia na ziskavanie hashovaneho hesla z databazy na zaklade pouzivatel'skeho emailu

Parameters

\$email

Returns

string —
result vysledok query do databazy

✓ `save_user()`

```
save_user( $id, $name, $surname, $email, $password, $salt)
```

Modelova funkcia nukladajuca uzivatela na zaklade zadanych parametrov

Parameters

\$id

\$name

\$surname

\$email

\$password

\$salt

FILE

[models\login_registration_model.php](#)

CLASS HIERARCHY

- ▶ \CI_Model

- ▶ \Login_registration_model

Tags

None found



Tags

None found



Tags

None found



Tags

None found



Class Hierarchy Diagram

Errors
Markers

\ ↕

Owners

</>

Summary

Methods	Properties	Constants
<ul style="list-style-type: none"> ✔ <code>index()</code> <code>current_list()</code> 	<i>No public properties found</i>	<i>No constants found</i>
⚠ <i>No protected method...</i>	<i>No protected properties ...</i>	<i>N/A</i>
🔒 <i>No private methods f...</i>	<i>No private properties fo...</i>	<i>N/A</i>

Methods

✔ `index()`

`index()`

</>

✔ `current_list()`

`current_list(string $order, string $orderby)`

</>

Return current list of billboard owners from database.

Parameters

string `$order` Ascending (asc) or descending (desc) order

string `$orderby` Column to be sorted by (possible values - id, name)

FILE

[controllers\owners.php](#)

CLASS HIERARCHY

- ▶ \CI_Controller
- ▶ \Owners

Tags

None found



Tags

None found



Tags

None found



Class Hierarchy Diagram



Errors Markers

\ ↕

Registration_model

</>

Summary

Methods	Properties	Constants
<ul style="list-style-type: none"> ✓ <code>__construct()</code> <code>get_password_for_login()</code> <code>save_user()</code> 	<ul style="list-style-type: none"> <i>No public properties found</i> 	<ul style="list-style-type: none"> <i>No constants found</i>
<ul style="list-style-type: none"> 🛡️ <i>No protected methods found</i> 	<ul style="list-style-type: none"> <i>No protected properties found</i> 	<ul style="list-style-type: none"> <i>N/A</i>
<ul style="list-style-type: none"> 🔒 <i>No private methods found</i> 	<ul style="list-style-type: none"> <i>No private properties found</i> 	<ul style="list-style-type: none"> <i>N/A</i>

Methods

✓ `__construct()`

`__construct()`

</>

✓ `get_password_for_login()`

`get_password_for_login($email)`

</>

Parameters

\$email

✓ `save_user()`

`save_user($id, $name, $surname, $email, $password, $salt)`

</>

Parameters

\$id

\$name

\$surname

\$email

\$password

\$salt

FILE

models\registration_model.php

CLASS HIERARCHY

▸ \CI_Model

▸ \Registration_model

Tags

None found

Tags

None found

Tags

None found

Tags

None found

Class Hierarchy Diagram

Errors
Markers