UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Kariž

Spletna aplikacija za objavljanje in naročanje novic

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00121/2011
Datum: 05.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TOMAŽ KARIŽ**

Naslov: **SPLETNA APLIKACIJA ZA OBJAVLJANJE IN NAROČANJE NOVIC**

WEB APPLICATION FOR SUBSCRIBING FOR NEWS AND FOR BROADCASTING NEWS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Zasnujte in razvijte spletno aplikacijo, ki uporabnikom omogoča naročanje na novice in objavljanje novic. Uporabite podatkovno bazo MySQL, programski jezik Pyton in programski okvir Flask.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil Microsoft Word.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Tomaž Karič,

z vpisno številko 63070303,

sem avtor diplomskega dela z naslovom:

Spletna aplikacija za objavljanje in naročanje novic

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja: _____

ZAHVALA

Zahvaljujem se mentorju doc. dr. Roku Rupniku za vse nasvete in vodenje pri pisanju diplomske naloge. Iskreno se zahvaljujem tudi mojim staršem, ki so me skozi vsa ta leta podpirali in mi omogočili študij. Velika zahvala gre tudi moji babici. Nazadnje bi se rad zahvalil tudi bratu Primožu, s katerim je bilo zabavno opravljati izpite skozi vsa leta in za njegovo pomoč pri urejanju diplomske naloge.

To diplomsko delo je posvečeno starim staršem.

Kazalo

Seznam uporabljenih kratic

AJAX (angl. Asynchronous JavaScript and XML) tehnologija za ustvarjanje interaktivnih spletnih aplikacij

CSS (angl. Cascading Style Sheets) slogovna predloga, ki določa izgled spletnih strani napisanih v označevalnem jeziku

HTML (angl. HyperText Markup Language) označevalni jezik za izdelavo spletnih strani

IDE (angl. Integrated Development Environment) integrirano programsko okolje

JIT (angl. Just In Time) izraz, ki se uporabi za predčasne prevajalnike

JSON (angl. JavaScript Object Notation) standard namenjen berljivi izmenjavi podatkov

ORM (angl. Object Relational Mapper) služi preslikovanju tabel v objekte in obratno

PHP (angl. Hypertext Preprocessor) jezik za programiranje spletnih strani

SQL (angl. Structured Query Language) strukturirani povpraševalni jezik za delo s podatkovnimi bazami

URL (angl. Uniform Resource Locators) tekstovni niz, ki predstavlja internetni naslov

XSS (angl. Cross Site Scripting) eden izmed možnih napadov na spletno stran

Povzetek

Splet je zadnja leta v velikem vzponu in vse več uporabnikov je raje obveščenih o novicah preko spletnih strani, kot preko televizije oziroma ostalih medijev. Eden izmed razlogov za to je, da je novice na internetu mogoče pregledati ob kateremkoli času v kolikor imamo povezavo z internetom. Zaradi velike količine uporabniku nezanimivih novic na eni strani in potrebe po pregledovanju več različnih spletnih strani na drugi, sem se odločil za izdelavo spletne aplikacije, ki bi v primeru aktivnosti uporabnikov utegnila rešiti tovrstni problem.

Namen diplomskega dela je bil izdelati spletno aplikacijo, ki uporabniku omogoča naročanje na novice drugih uporabnikov. V nalogi se seznanimo z orodji in tehnologijami, ki sem jih uporabil pri razvoju aplikacije. Aplikacija je bila izdelana v programskem jeziku Python v povezavi s podatkovno bazo MySQL. Podrobno opišem tudi sam potek načrtovanja aplikacije. Razvita aplikacija je le začetna različica, ki se v prihodnosti utegne razširiti z dodatnimi funkcionalnostmi.

Ključne besede:

Spletna aplikacija, novice, MySQL

Abstract

In recent years web is rising. More and more users wish to be informed about the news via the web rather than via television or other media. One reason for this is that the news on the internet can be retrieved at any time as far as we have a connection to the internet. Due to large amounts of user uninteresting news on one hand and the need for surfing over various different websites on the other, I decided to create a web application which in case of user activity might solve this problem.

The aim of thesis was to create a web application that allows user to subscribe to news from other users. In thesis we are acquainted with the tools and technologies that I used to develop the application. The application was made in the programming language Python in conjuction with MySQL database. I also describe in detail the process of designing. The developed application is only the initial version, which may in future be extended with additional functionalities.

Key words:

Web application, news, MySQL

1 Uvod

Opredelitev problema

Številne spletne strani, ki ponujajo pregled novic (http://www.24ur.com, http://www.tmz.com, http://www.bbc.co.uk/news/) vsebujejo veliko novic, ki uporabnika ne zanimajo. Nekatere uporabnike zanimajo le novice iz sveta športa, drugi pa prav teh ne prebirajo radi. Seveda je tu še problem, da nobena stran nima vseh novic iz posamezne kategorije, temveč imajo večinoma le najpomembnejše, kar ne zadovolji vseh uporabnikov. Večina spletnih strani se omeji zgolj na novice iz najpopularnejših kategorij kot recimo šport, politika, vreme, ne objavljajo pa podrobnejših novic iz sveta mobitelov, filmov. Zelo nepraktično je, da mora uporabnik pregledovati številne spletne strani, da se seznani z novicami, ki ga zanimajo. Moja želja je bila razviti spletno aplikacijo, kjer bi vsak registriran uporabnik lahko objavljal svoje novice, ostali uporabniki pa bi se lahko, v primeru da so jim novice tega uporabnika všeč, naročili na njegove novice.

Struktura diplomske naloge

V diplomski nalogi želim predstaviti potek razvoja aplikacije.

V prvem delu bom predstavil orodja in programske jezike, ki sem jih uporabil pri izdelavi aplikacije za objavljanje in naročanje novic. Na kratko bom opisal zakaj sem se za določeno tehnologijo odločil in čemu je namenjena. Sledil bo opis načrtovanja strukture datotek, ki je namenjeno predvsem skalabilnosti in hitrejšemu nadaljnemu razvoju ter opis načrtovanja podatkovne baze. Glavni del zajema tako opis različnih funkcionalnosti aplikacije kot tudi način implementacije le-teh.

2 Uporabljena tehnologija in orodja

2.1 Predstavitev uporabljenih tehnologij

Izbira tehnologij je pri razvoju spletne aplikacije ena najpomembnejših odločitev. Aplikacija ni bila izdelana za naročnika, zato sem imel proste roke pri izbiri tehnologij in orodij.

Začetna različica aplikacije je le neka osnova, ki bi se jo kasneje po potrebi razširilo z novimi idejami. Iz tega razloga sem aplikacijo želel narediti skalabilno, da bi s tem v prihodnje prihranil čas pri izdelavi dodatnih modulov. Na podlagi mojega znanja in izkušenj s spletnim programiranjem sem se odločil za uporabo najprimernejših tehnologij, V nadaljevanju jih bom tudi povzel.(ki so se mi zdele najprimernejše, katere bom v nadaljevanju na kratko tudi povzel.)

2.1.1 Označevalni jezik HTML

HyperText Markup Language [1] je označevalni jezik za izdelavo spletnih strani. Je osnovni jezik za spletno programiranje in je eden enostavnejših za učenje. HTML dokument sestavljajo oznake (angl. tags), ki so lahko poljubno gnezdene. Vsaka oznaka je definirana znotraj znakov '<' in '>'. Sestavlja jo začetek, kjer lahko opišemo tudi vrednosti atributov oznake in konec, ki se od začetka razlikuje po tem, da je brez atributov in je označen z '/' znakom. V primeru, da pozabimo zaključiti oznako, znajo v nekaterih primerih spletni brskalniki to napako sami odpraviti. Ena izmed najbolj uporabljenih oznak je "<a>", ki s klikom na vsebino omogoča povezavo na drugi dokument. Leta 2008 je prišla zadnja različica HTML 5.

Ob prejetju HTML strukture dokumenta, nam brskalnik ne prikaže oznak, ampak na njegovi osnovi zgradi izgled strani.

2.1.2 Stilski jezik CSS

Cascading Style Sheets oz. CSS [2] je jezik s katerim lahko opišemo slog dokumenta napisanega v označevalnem jeziku. Razvit je bil z namenom, da poveča preglednost izvorne kode tako, da loči slog dokumenta od njegove vsebine. Najpogosteje se uporablja za oblikovanje sloga spletnih strani. Sestavljen je iz pravil, ki lahko opisujejo postavitev, pisavo in barvo. Eno pravilo lahko uporablja več različnih elementov in en element lahko uporablja več pravil.

2.1.3 Programski jezik Python

Python [3] je dinamičen objektno orientiran jezik namenjen razvoju tako spletnih kot sistemskih aplikacij. Prva različica je prišla leta 1991, razvil pa ga je Guido Van Rossum. Python je predvsem znan po svoji enostavni berljivosti programov in zelo hitrem razvoju. Spada med visoko nivojske jezike in je podoben jezikom kot so Ruby, Boo in Perl. Obstajajo tri glavne implementacije jezika. CPython je implementiran v jeziku C in je najbolj razširjen. Jython je implementacija jezika v Javi, kjer se program prevede v Java bytecode katerega lahko Java Virtual Machine zažene. Podobno deluje tudi IronPython, ki je implementacija jezika v C#. Vse bolj popularna postaja tudi implementacija PyPy napisana v jeziku Python, predvsem zaradi vključenega JIT (angl. Just In Time) prevajalnika. Python je bil dvakrat izbran za jezik leta na TIOBE lestvici.

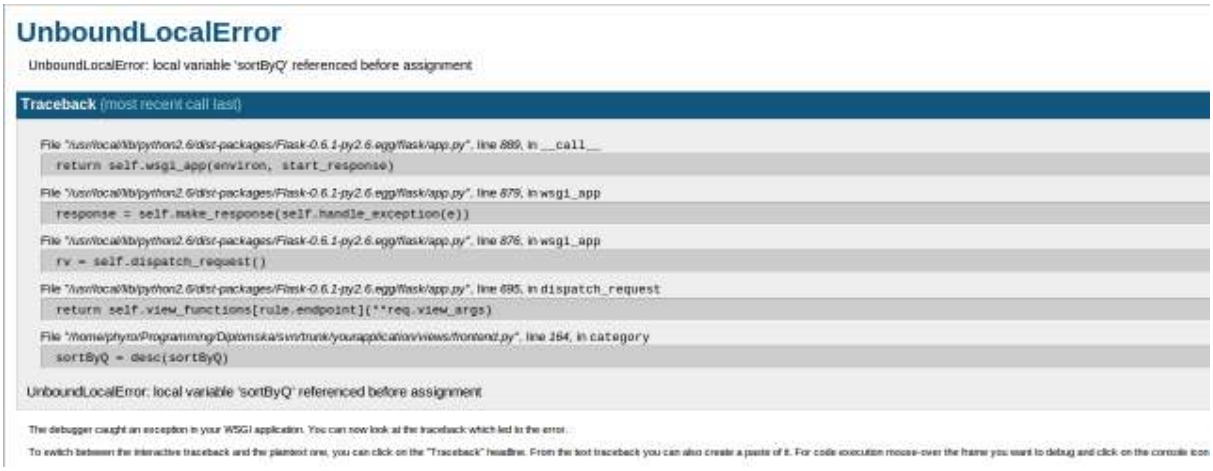
2.1.4 Programski jezik JavaScript

JavaScript [4] je objektno orientiran skriptni programski jezik. Razvit je bil z namenom, da omogoči obiskovalcem strani bolj dinamično brskanje po spletu. Zaradi podpore gnezdenja funkcij (angl. closures) in funkcij višjega reda (angl. higher-order function) ga uvrščajo tudi med funkcijske jezike. Ena izmed prednosti uporabe JavaScript jezika je, da se izvaja na strani odjemalca in s tem zmanjša obremenitev strežnika.

Jezik postaja vse bolj priljubljen, podpirajo ga tudi vsi novejši spletni brskalniki.

2.1.5 Framework Flask

Flask [5] je mikro programski okvir za izdelavo spletnih aplikacij v jeziku Python. Temelji na knjižnici Werkzeug in Jinja2. Flask je bil javnosti najprej predstavljen kot prvoaprilska šala, vendar je bil zaradi velikega zanimanja kasneje tudi razvit. Glavni razlog za pripono mikro je, da je ideja okvira imeti jedro enostavno a razširljivo. Okvir ima vgrajen strežnik za razvoj in razhroščevalnik (slika 2.1), kateri nam omogoča lažje odpravljanje napak.



Slika 2.1: Razhroščevalnik v Flasku

Flask nam omogoča tudi tako imenovano usmerjanje (angl. routing) spletnih naslovov, katere lahko na enostaven način povežemo z željeno funkcijo. Okvir je zasnovan na konceptu razširljivosti, kar uporabniku omogoča enostavno dodajanje dodatnih funkcionalnosti okolju. Ena izmed razširitev, ki sem jih uporabil je Flask-WTF, ki uporabniku omogoča enostavno integracijo z WTForms.

2.2 Predstavitev orodij

Razvojno orodje oziroma IDE (angl. Integrated Development Environment) je integrirano razvojno okolje namenjeno programerjem, da jim z dodatnimi funkcionalnostmi olajša razvoj. Glavne stvari, ki jih novejše razvojno okolje običajno vsebuje so:

- urejevalnik izvorne kode
- prevajalnik
- predloge dopolnjevanja kode
- številne bližnjice s kombinacijami tipk
- razhroščevalnik.

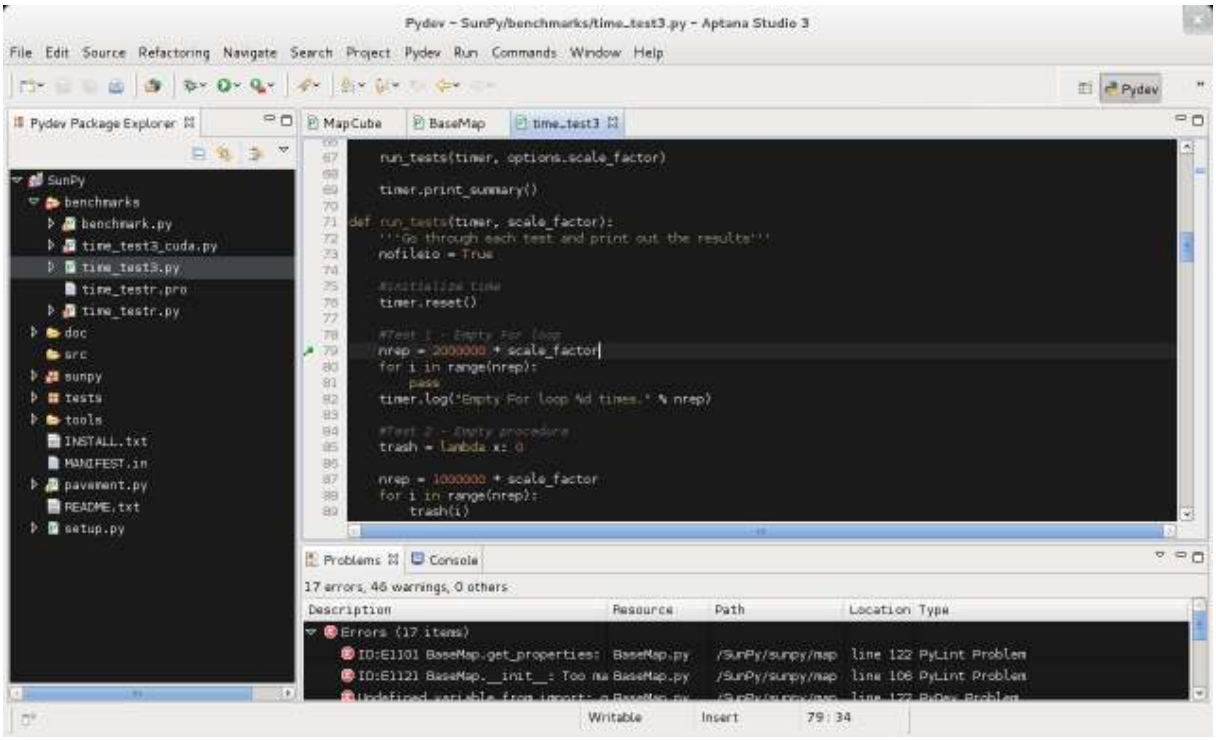
Nekateri za določene jezike ponujajo tudi izdelavo uporabniškega vmesnika (angl. Graphical User Interface) na principu povleci in spusti (angl. drag and drop).

V nadaljevanju bom opisal razvojno okolje Aptana Studio katerega sem uporabil pri razvoju aplikacije in orodji phpMyAdmin ter PowerDesigner s katerima sem načrtoval oziroma urejal podatkovno bazo.

2.2.1 Aptana Studio

Aptana Studio (slika 2.2) je razvojno okolje namenjeno pretežno izdelovanju spletnih strani [6]. Izbral sem ga zaradi odlične podpore spletnih tehnologij, ki sem jih uporabil kot recimo Python, JavaScript, Ajax, HTML ter CSS. Program temelji na razvojnem okolju Eclipse, zato ga je mogoče namestiti tudi kot dodaten Eclipse vtičnik (angl. plugin). Aptana studio je na voljo za vse vodilne platforme, njena največja prednost pa je, da je popolnoma brezplačen in na voljo za vsakogar.

Za to razvojno okolje sem se odločil zaradi odlične Python podpore preko PyDev vtičnika, kateri omogoča sprotno preverjanje sintakse, prikaz opisa funkcij s standardnih knjižnic, dopolnjevanje kode ipd.. Okolje ima podporo tudi za implementaciji Jython in IronPython.



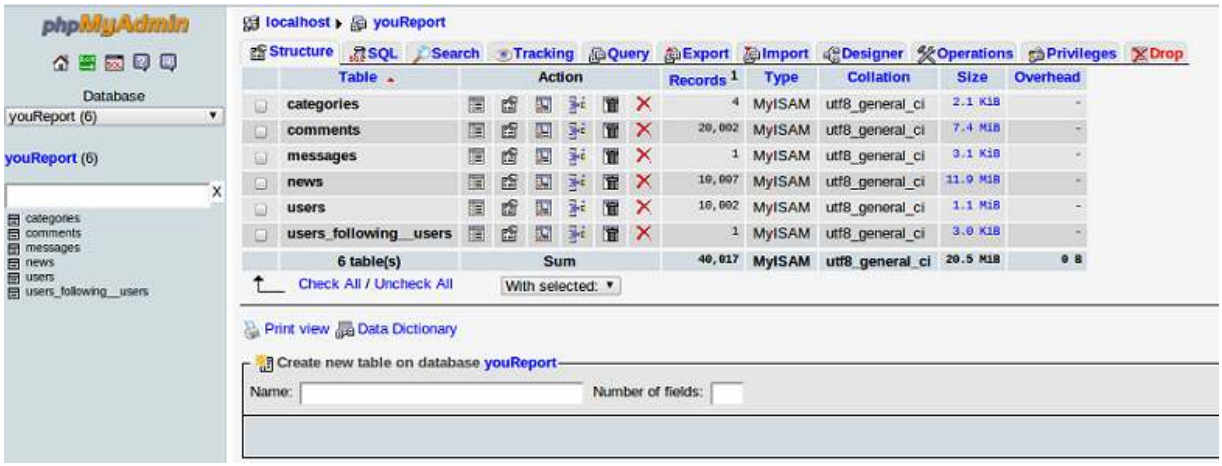
Slika 2.2: Razvojno okolje Aptana Studio

2.2.2 phpMyAdmin

PhpMyAdmin [7] je orodje za delo z bazami, ki že nekaj let spada med najbolj priljubljena orodja za tovrstna opravila. Orodje je napisano v programskem jeziku PHP in nam omogoča upravljanje z MySQL bazami kar preko spletnega brskalnika (slika 2.3). MySQL [8] je eden izmed sistemov za upravljanje s podatkovnimi bazami, ki za delo s podatki uporablja SQL stavke.

PhpMyAdmin nam omogoča:

- kreiranje in brisanje podatkovnih baz,
- kreiranje, brisanje in upravljanje tabel ter njenih podatkov,
- upravljanje s ključi in indeksi na poljih,
- enostavno iskanje po bazi,
- izvajanje SQL stavkov in
- enostaven uvoz in izvoz podatkovnih baz.



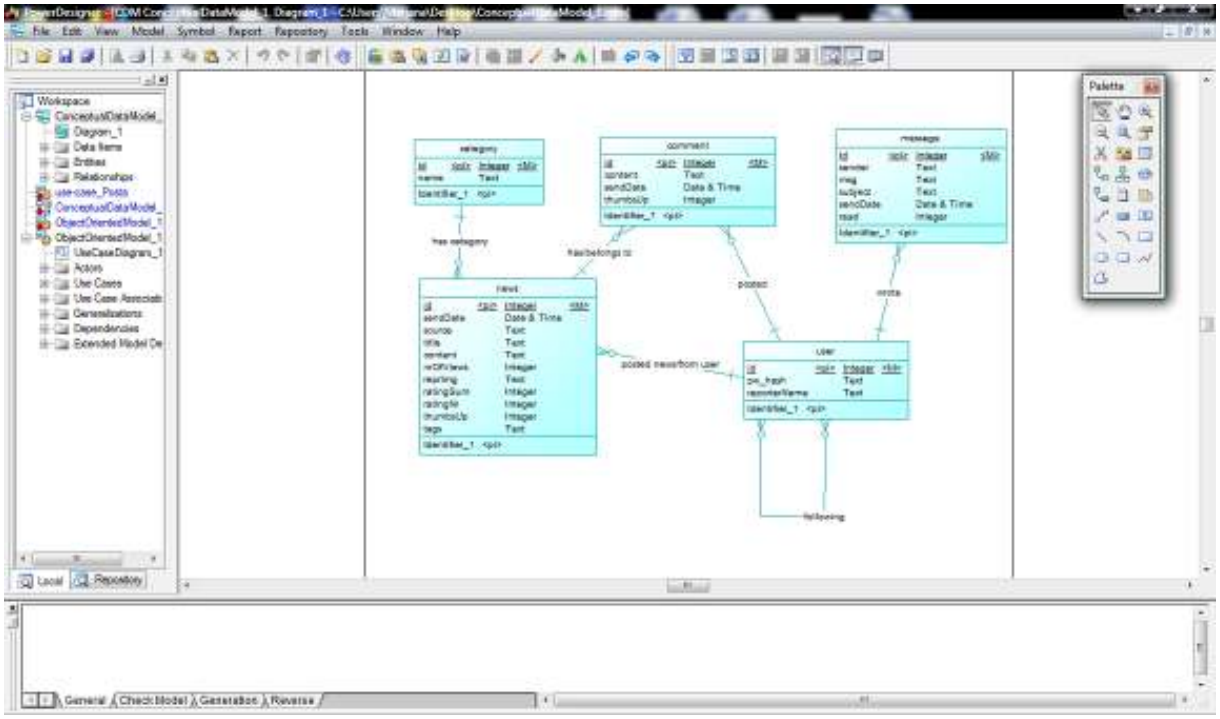
Slika 2.3: Upravljanje MySQL baze preko brskalnika z orodjem phpMyAdmin

2.2.3 Orodje PowerDesigner

PowerDesigner [9] je orodje namenjeno načrtovanju podatkovnih baz. Izdelalo ga je podjetje Sybase.

PowerDesigner (slika 2.4) nam med drugim omogoča razvoj konceptualnih, logičnih in fizičnih podatkovnih modelov. Podpira širok spekter podatkovnih baz, med katere spada tudi MySQL, katero sem tudi sam uporabil. Na voljo je le za operacijski sistem Microsoft Windows.

Pri aplikaciji sem ga uporabil za izdelavo konceptualnega in fizičnega podatkovnega modela ter diagrama primerov uporabe.



Slika 2.4: Orodje PowerDesigner

Omogoča tudi kreiranje diagramov kot so:

- entitetni diagram

Entitetni diagram nam prikaže relacije med entitetami v podatkovni bazi. Entiteta je objekt, ki ima shranjene podatke. Relacija definira kako sta dve entiteti povezani.

- diagram podatkovnih tokov

Diagram podatkovnih tokov nam prikaže potek informacij med objekti.

Povezave oziroma relacije med entitetami (slika 4) so lahko naslednje:

- Relacija 0,1
- Relacija 1,1
- Relacija 0,n
- Relacija 1,n

2.3 Uporabljene knjižnice

2.3.1 SQLAlchemy

SQLAlchemy [10] je orodje in ORM (angl. Object Relational Mapper) za delo z SQL v jeziku Python. Razvit je kot samostojen projekt, ki ga je razvil Mike Bayer in ožja skupina razvijalcev. Prva različica je bila na voljo februarja 2006 in SQLAlchemy je kmalu postal eden izmed najbolj razširjenih ORM-jev za jezik Python. Razvijalcu omogoča delo z SQL na način, kot smo vajeni v jeziku Python. Poskrbi za osnovne stvari, kot so recimo povezava z bazo, poizvedovanje, pa tudi za preslikave podatkov v objekte. Eden izmed ciljev SQLAlchemy je, da je na voljo širši vrsti podatkovnih baz, kot recimo SQLite, MySQL, Oracle, PostgreSQL, MS-SQL. Eden izmed razlogov, da je projekt uspel je tudi ta, da se je razvila bogata skupnost in je dandanes mogoče najti veliko razširitev in vtičnikov (angl. plugins). Elixir je eden izmed bolj znanih katerega sem uporabil pri razvoju aplikacije.

2.3.2 Elixir

Elixir [11] je deklarativna plast nad SQLAlchemy. Je tanek ovoj, ki omogoča kreiranje Python objektov sledoč Active Record načrtovalskem vzorcu (angl. design pattern). Ena izmed lepot uporabe Elixirja je, da s kreiranjem objekta definiramo objekt, tabelo in njuno preslikavo v enem koraku in z manj vrsticami kode. Relacije med entitetami v Elixirju enostavno opišemo na naslednji način.

```
news = OneToMany('News', order_by="-sendDate")
```

2.3.3 Jinja2

Jinja2 [12] je eden izmed najpopularnejših mehanizmov za upravljanje s predlogami (angl. template engine) za jezik Python, ki služi ločevanju logike aplikacije od njenega izgleda. Izdelala ga je ekipa Pocoo, katere vodja je Armin Ronacher. Temelji na mehanizmu, ki ga uporablja spletno razvojno okolje Django, prednost jinja2 pa je večji nabor funkcij in bolj prožno izvajanje. Mehanizem omogoča dedovanje predlog, avtomatsko zaščito pred XSS (angl. cross site scripting) napadi ter enostavno razhroščevanje. Omogoča nam tudi uporabo makrojev s katerimi si lahko olajšamo delo.

```
{% macro flashMessages() %}
  {% with messages = get_flashed_messages(with_categories=true) %}
          {% if messages %}
            <ul class=flashes>
            {% for category,message in messages %}
              <li class={{ category }}>{{ message }}</li>
            {% endfor %}
            </ul>
          {% endif %}
  {% endwith %}
{% endmacro %}
```

2.3.4 WTForms

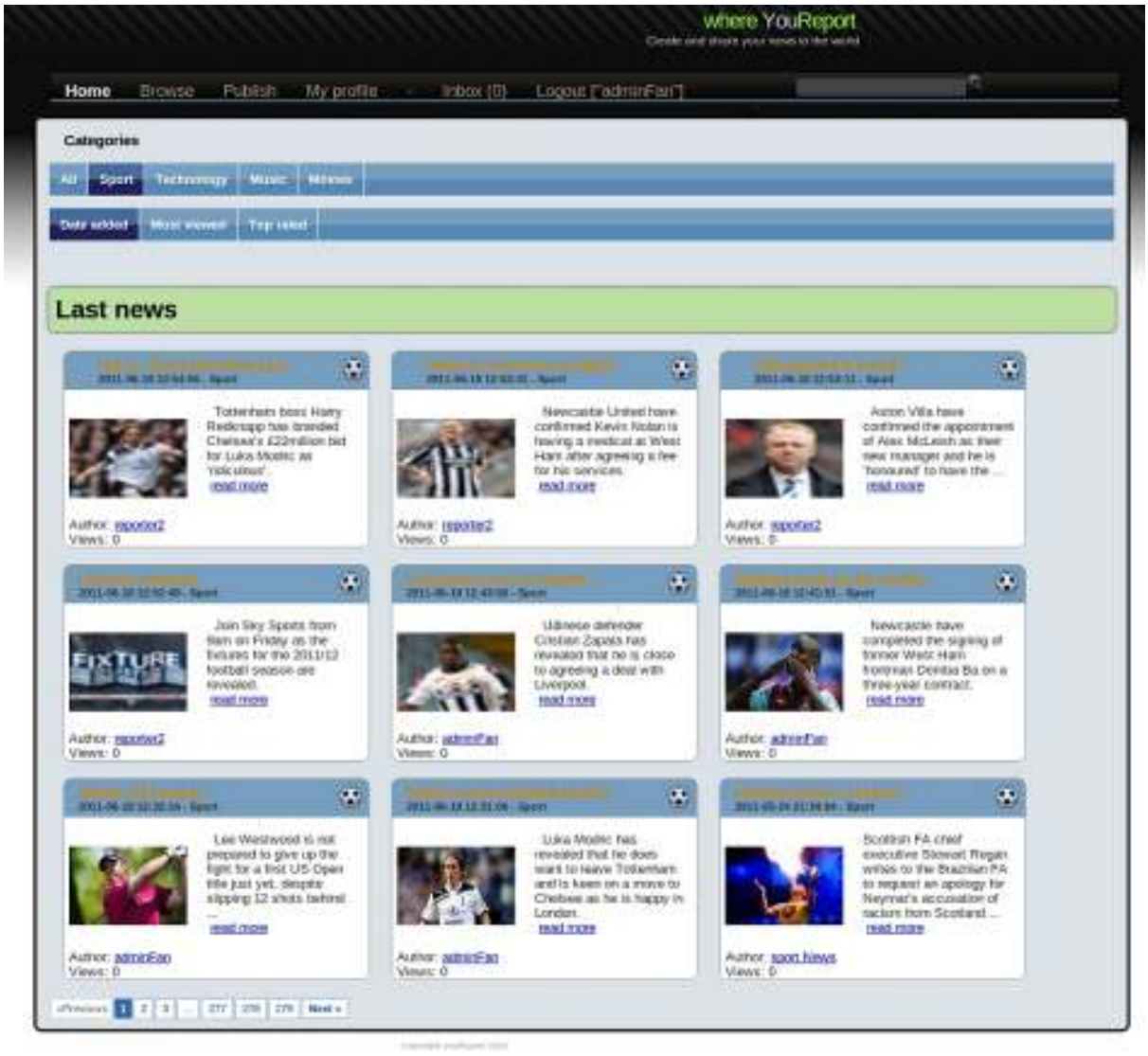
WTForms [13] je knjižnica za spletno programiranje v jeziku Python, ki nam omogoča lažjo validacijo obrazcev (angl. form) in izris le teh. Obrazec je mogoče opisati kot objekt, kjer lahko razvijalec za vsako polje definira pogoje, ki morajo veljati ob procesiranju obrazca. Primarna naloga je ločitev lastnosti obrazcev od njihovega izrisa.

2.3.5 JQuery

Jquery [14] je JavaScript knjižnica, ki nam na enostaven način omogoča ustvarjanje čudovitih interaktivnih spletnih menijev, pošiljanje AJAX zahtev in ustvarjanje slikovnih diaprojekcij (angl. slideshow) v le nekaj vrsticah kode. Poleg naštetih prednosti je ena izmed glavnih, da JQuery koda deluje na vseh bolj znanih spletnih brskalnikih. Prva različica je bila izdana leta 2006, ki jo je napisal John Resig, kateri je še danes glavni razvijalec. Knjižnica ima zelo preprosto sintakso, kar uporabniku omogoča pisanje vtičnikov na enostaven način. Danes je najbolj razširjena JavaScript knjižnica, ki se še vedno razvija in je eno izmed najboljših orodij za ustvarjanje interaktivnih spletnih strani.

3 Razvoj aplikacije

Razvoj aplikacije (slika 3.1) je potekal v treh fazah. Najprej sem načrtoval strukturo aplikacije, ki je zajemala razporeditev datotek zaradi lažje preglednosti in izboljšane skalabilnosti aplikacije. Sledilo je načrtovanje podatkovne baze. Tu sem s pomočjo orodja PowerDesigner izdelal potrebne modele za lažji pregled nad podatki. Po končanem načrtovanju sem izdelal še posamezne module aplikacije katere sem v nadaljevanju tudi opisal.



Slika 3.1: Slika izdelane spletne aplikacije

3.1 Načrtovanje strukture aplikacije

Z izkušnjami sem ugotovil, da je načrtovanje ena izmed glavnih stvari za samo implementacijo in boljšo skalabilnost aplikacije, zato sem se odločil, da si najprej strukturiram datoteke, ki jih bo aplikacija zajemala. Zaradi lepše preglednosti sem se odločil, da bom ločil izgled, podatkovne modele, opis obrazcev, statične datoteke in logiko izgleda vsako v svojo mapo. V vsaki izmed map sem ustvaril datoteko z imenom __init__.py. Ta datoteka je potrebna, da Python obravnava mapo kot paket (angl. package). Datoteka je vključila vse datoteke s končnico .py, ki se nahajajo v isti mapi, tako da je bilo potrebno za celotno funkcionalnost paketa uvoziti le __init__.py datoteko.

3.1.1 Izgled

Za izgled strani sem uporabil jinja2 mehanizem, ki nam omogoča dedovanje predlog. V mapi z imenom »templates«, sem ustvaril datoteko baseNew.html, ki je bila osnova za vse strani.

3.1.2 Logika izgleda

Logiko izgleda sem hranil v mapi »views«. Tu sem logiko razdelil med posamezne datoteke glede na to kdo lahko dostopa do teh strani.

3.1.3 Podatkovni modeli

Podatkovne modele sem imel definirane v mapi »models«. Tu sem imel vsak model v svoji Python datoteki predstavljen kot objekt z atributi in relacijami z drugimi modeli. Relacije sem lahko s pomočjo Elixirja definiral na zelo enostaven način. Relacijo "novica ima lahko nič ali več komentarjev" lahko naredimo tako, da objektu News dodamo atribut comments katerega definiramo takole:

```
comments = OneToMany('Comment')
```

3.1.4 Statične datoteke

V mapi »static«, v kateri sem hranil statične datoteke, sem ustvaril 3 glavne podmape zaradi lepše razporeditve in sicer:

- images (mapa hrani slike, ki jih spletna aplikacija vsebuje),
- css (mapa, ki vsebuje stilske datoteke) in
- js (mapa, ki vsebuje JavaScript datoteke).

3.1.5 Opis obrazcev

Obrazce sem opisal s pomočjo knjižnice WTForms v mapi »forms«. V mapi sem ustvaril tudi datoteko validators.py, v kateri sem imel definirane različne validacije, katere sem lahko uporabil pri obrazcih.

3.2 Načrtovanje in izdelava podatkovne baze

3.2.1 Diagram primera uporabe

Diagram primera uporabe (angl. Use Case Diagram) grafično predstavi načine komuniciranja med uporabniki in sistemom. Diagram zajema naslednje gradnike:

- akterje,
- primere uporabe,
- relacije akterjev s primeri uporabe,
- relacije primerov uporabe z drugimi primeri uporabe in
- relacije akterjev z drugimi akterji.

Sistem je predstavljen kot množica procesov, katere lahko določen uporabnik uporablja [15].

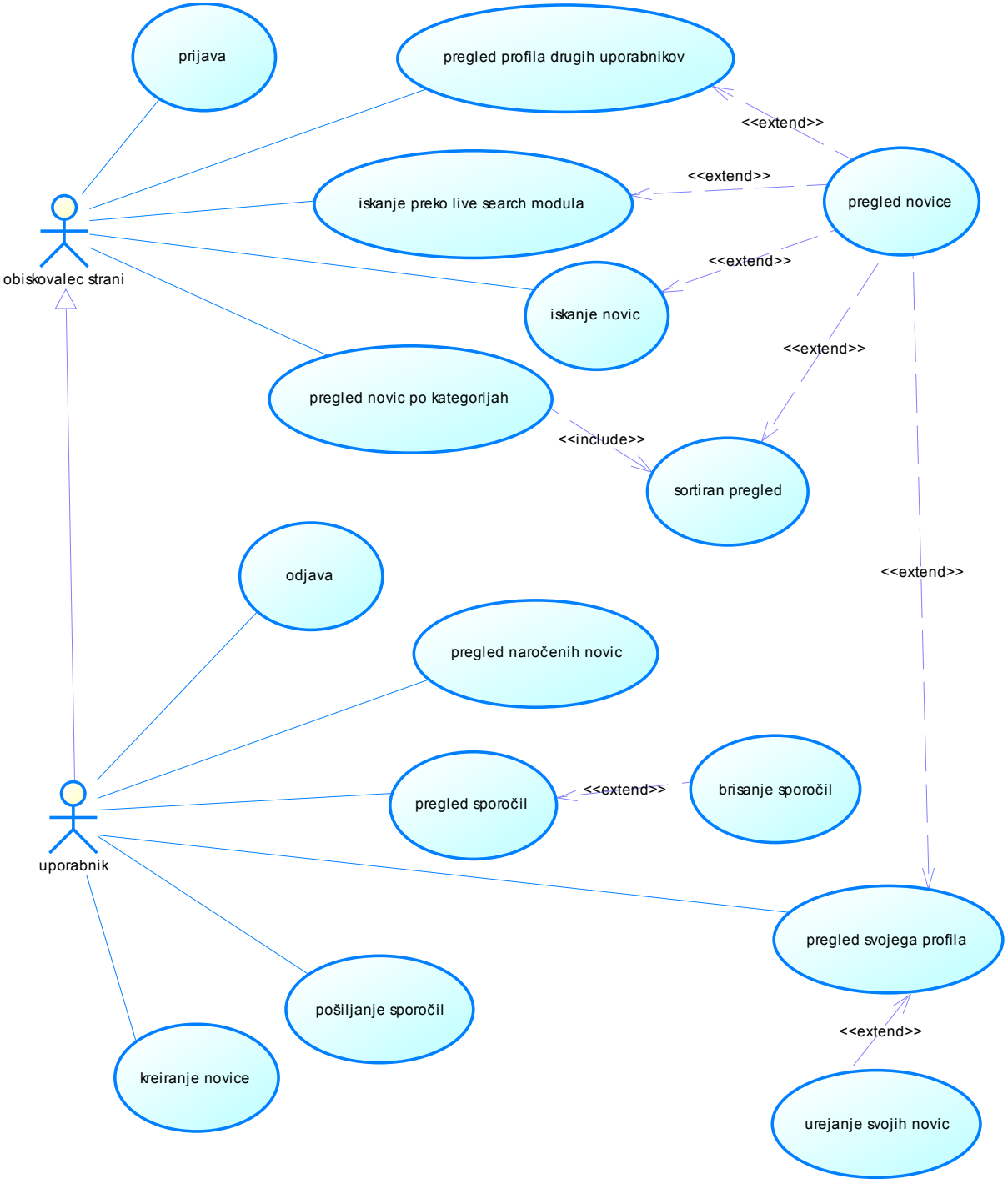
Zaradi preglednosti nad celotno funkcionalnostjo aplikacije, ki nam jo prikaže diagram, sem se odločil za izdelavo diagrama, ki zajema celotno aplikacijo (slika 3.2). V nadaljevanju bom predstavil funkcionalnosti posameznih akterjev v sistemu.

Obiskovalec strani ima naslednje dodatne funkcionalnosti:

- Prijava (po prijavi ima uporabnik možnost odjave)
- Pregled profila drugih uporabnikov
- Pregled novice
- Iskanje preko live search modula
- Iskanje novic
- Pregled novic po kategorijah
- Iskanje po zgodovini (lahko iščemo po naročniku, datumu, številki svetovanja ter izbranem nizu)

Uporabnik ima poleg vseh možnosti prijavljenega uporabnika še naslednji možnosti:

- Pregled naročenih novic
- Pregled, pošiljanje in brisanje sporočil
- Pregled svojega profila
- Urejanje svojih novic
- Kreiranje novice



Slika 3.2: Izdelan diagram primerov uporabe

3.2.2 Konceptualni podatkovni model

Konceptualni podatkovni model je predstavljen z entitetnim diagramom.

Gradniki modela so entitete in povezave med entitetami.

Izdelava konceptualnega podatkovnega modela je ena izmed najpomembnejših načrtovanj pri izdelavi podatkovne baze. Dodaten razlog za izdelavo modela pri mojem projektu je, da bodo moji podatkovni modeli tudi v kodi prikazani kot entitete z atributi in relacijami. Konceptualni podatkovni model nam lahko prihrani čas potreben za izvedbo projekta, saj imamo jasno definirana razmerja med entitetami.

Konceptualni model (slika 3.3) vsebuje naslednje entitete:

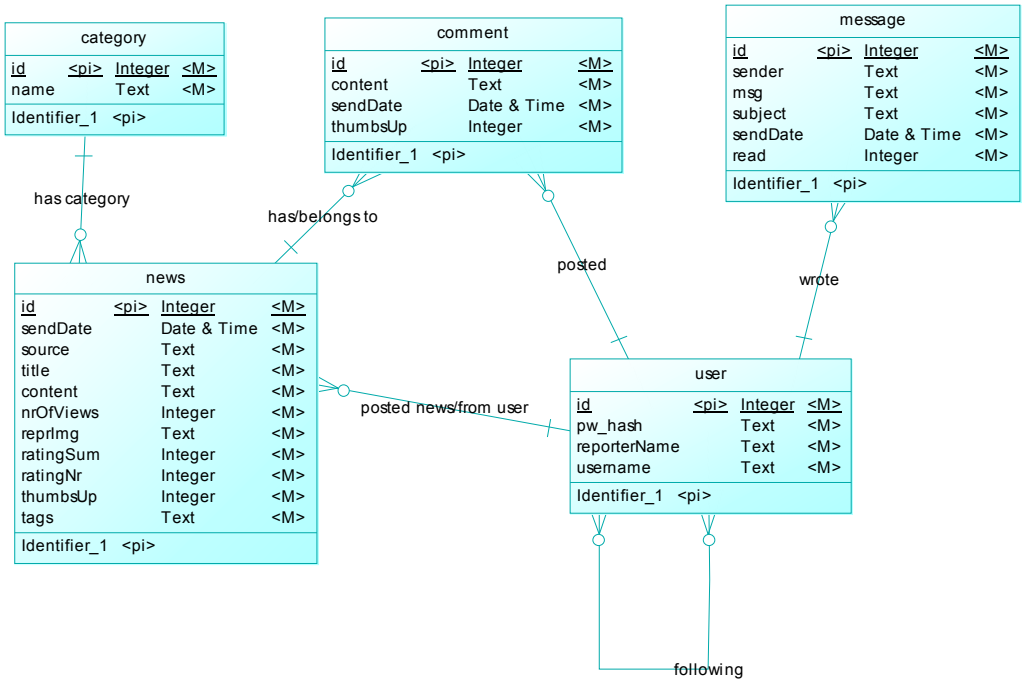
category – zajema novice, ki so vnaprej določene v bazi. Vsebuje primarni ključ id in ime kategorije.

comment – entiteta hrani podatke o komentarjih. Komentar ima svoj enoličen id, vsebino, datum objave in število pozitivnih ocen komentarja. Namenjen je prikazu pod določeno novico.

news – najpomembnejša entiteta, ki hrani novice. Vsaka novica ima svoj enolično določen id, datum objave, naslov novice, vir novice, vsebino novice, število ogledov novice, sliko, ki predstavlja novico, podatke o ocenah novice in ključne besede.

message – entiteta, ki hrani podatke o poslanih sporočilih. Vsebuje primarni ključ id, naziv pošiljatelja, datum vnosa, zadevo in vsebino sporočila ter vrednost, ki nam pove ali je bilo sporočilo že prebrano.

user – entiteta hrani podatke o registriranih uporabnikih. Vsebuje primarni ključ id, uporabnikovo uporabniško ime in geslo ter naziv uporabnika v sistemu.

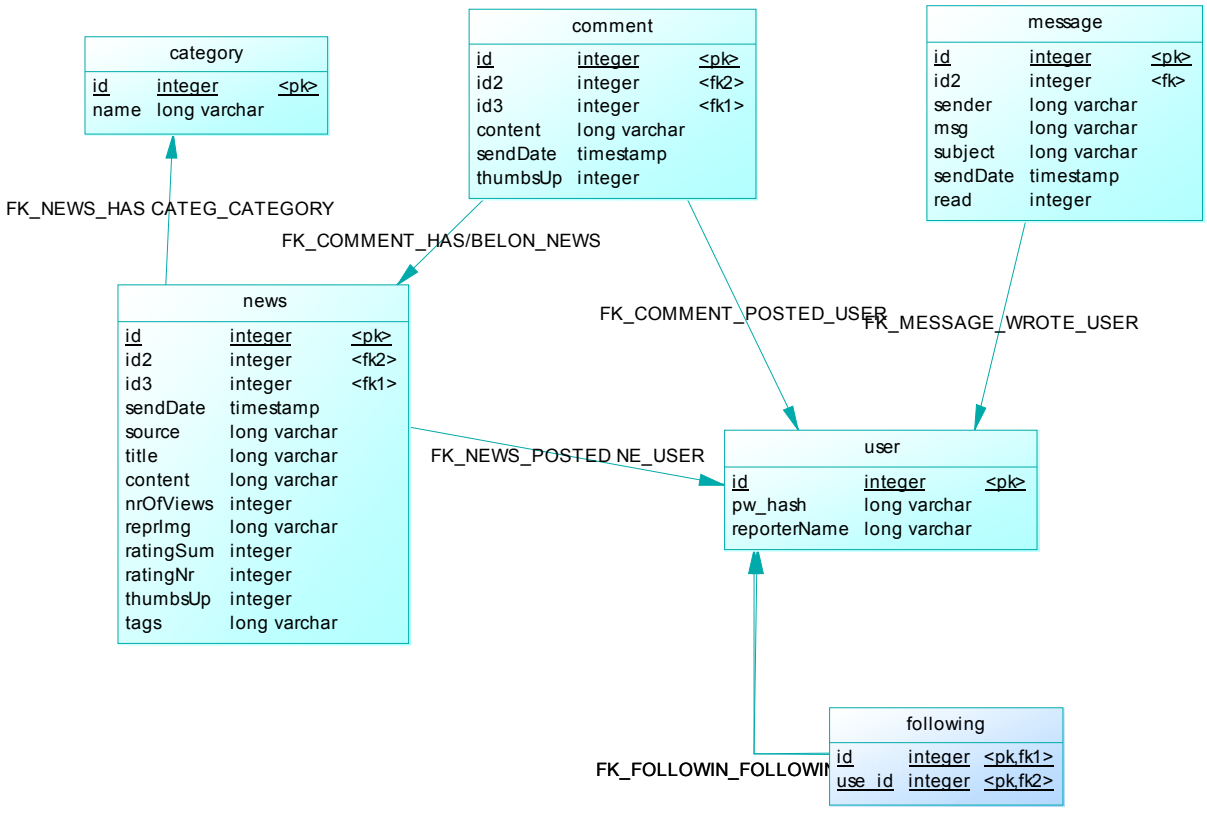


Slika 3.3: Izdelan konceptualni podatkovni model

3.2.3 Fizični podatkovni model

Fizični podatkovni model (slika 3.4) se od konceptualnega razlikuje po tem, da nam prikaže dejansko podatkovno bazo s podatkovnimi tipi ter primarnimi in tujimi ključi. Orodje PowerDesigner nam omogoča pridobitev SQL kode za kreiranje podatkovne baze na podlagi modela. Omenjene funkcionalnosti nisem potreboval, saj sem podatkovno bazo ustvaril s pomočjo Elixirja. Po opisu objektov entitet sem si ustvaril datoteko create_db.py, ki je med drugim vsebovala naslednji dve vrstici za kreiranje podatkovne baze:

```
setup_all()
create_all()
```



Slika 3.4: Izdelan fizični podatkovni model

3.3 Opis in razvoj aplikacije

Za izdelavo spletne aplikacije sem moral implementirati različne module in funkcionalnosti. Moduli so med seboj neodvisni in enostavno razširljivi.

V nadaljevanju bom predstavil nekatere module in funkcionalnosti, ki sem jih implementiral v aplikaciji.

3.3.1 Live search modul

V današnjih časih uporabniki interneta raje obiskujejo interaktivne strani, saj se hitreje odzivajo na uporabnikove zahteve. To omogoča tehnologija AJAX, ki se je v zadnjih letih zelo razširila. Zaradi enostavnejšega iskanja novic, sem implementiral iskalnik (slika 3.5), ki prikazuje rezultate neposredno, medtem ko uporabnik vpisuje ključne besede. Iskalnik išče po atributu »tags« in v kolikor je katera izmed vpisanih besed najdena v vrednosti atributa obravnava novico kot zadetek. Iskalnik uporabniku za vsak zadetek prikaže predstavitveno sliko novice, naslov novice in število ogledov, ki jih novica trenutno ima. V primeru, da si uporabnik želi prebrati novico ima možnost klika na željen zadetek, ki ga preusmeri na stran, kjer ima možnost ogleda celotne novice. Število prikazanih zadetkov sem zaradi boljše preglednosti omejil na pet. Iskalnik sem implementiral s pomočjo jezika JavaScript, kateri je ob spremembi vsebine vnosnega polja, poslal JSON zahtevo na strežnik. Na strežniku se je izvedla skripta, ki je na podlagi informacij o iskanju pridobila željene zadetke. Pridobljene zadetke je nato poslala predlogi, katera je generirala ustrezen html z zadetki.

```
outputHtml = render_template('ajaxLiveSearch.html', newsResult=newsResult)
```

Po končanem generiranju je skripta oddala podatke v JSON zapisu in nato je bilo potrebno podatke le še prikazati na strani.

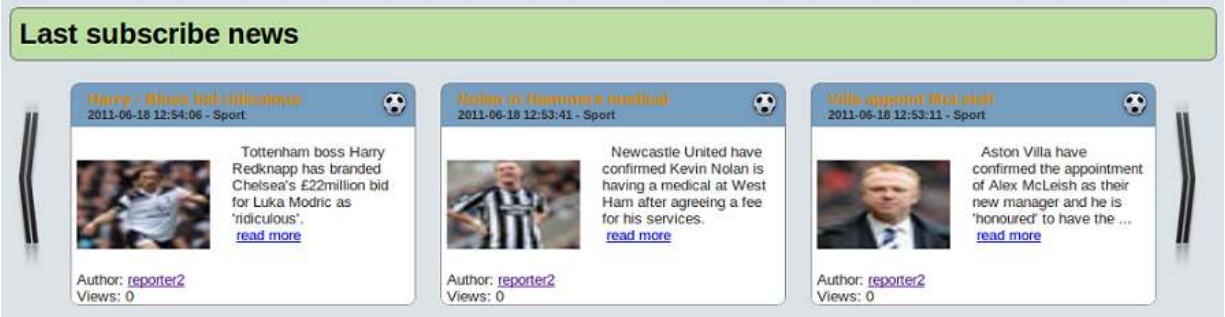
```
return jsonify(result=outputHtml)
```

Slika 3.5: Live search iskalnik

3.3.2 Pregled naročenih novic

Uporabniki pogosto radi berejo novice uporabnikov za katere mislijo, da prispevajo kvalitetne oziroma zanimive novice. S tem namenom sem aplikaciji implementiral možnost naročitve na novice uporabnika. Opisal bom način možnosti pregleda najnovejših novic na katere je uporabnik naročen in postopek implementacije modula. Za prikaz naročenih novic sem izbral osnovno stran, saj tako uporabniku prihranimo nepotreben klik ali dva. Naročene novice sem prikazal v primeru, da je bil uporabnik prijavljen in naročen na vsaj enega uporabnika. Zaradi lepše preglednosti sem se odločil, da bodo prikazane le tri novice in bo uporabnik imel možnost prehoda na naslednje oziroma prejšnje tri s klikom na ustrezno puščico (slika 3.6).

Za olajšanje dela sem ustvaril objekt Paginate, katerega bom v nadaljevanju tudi bolj podrobno predstavil. Prehode med stranmi sem implementiral z jezikom JavaScript in sicer tako, da se je ob kliku na naslednjo oziroma prejšnjo stran poslala JSON zahteva na strežnik, kjer je skripta pridobila naslednje oziroma prejšnje naročene novice in na podlagi novic ustvarila prikazani html ter ga v JSON obliki poslala kot odgovor na zahtevo. Postopek generiranja HTML niza je zelo podoben tistemu pri modulu live search.



Slika 3.6: Pregled naročenih novic

3.3.3 Pregled novic po kategorijah

Uporabniki velikokrat želijo pregledati novice posameznega področja (npr. novice iz športa), zato jim je bila dodana možnost prikaza novic le iz določene kategorije, ki jo lahko izberejo s klikom nanjo. Brskanje po novicah bi bilo dokaj nepregledno v kolikor novice ne bi bile razporejene v kategorije. Zaradi tega sem naredil, da je vsaka novica uvrščena v eno izmed kategorij, ki so na voljo. Kategorije, ki sem jih dodal začetni različici aplikacije so razvidne iz slike 3.7.

Slika 3.7: Kategorije novic v aplikaciji

Po izbrani kategoriji se prikažejo tri opcije sortiranja (slika 3.8), ki nam omogočajo pregled novic sortiranih glede na datum objave, števila ogledov novice in povprečne ocene novice. Privzeto sortiranje je glede na datum objave in sicer prikaže najnovejše novice najprej.

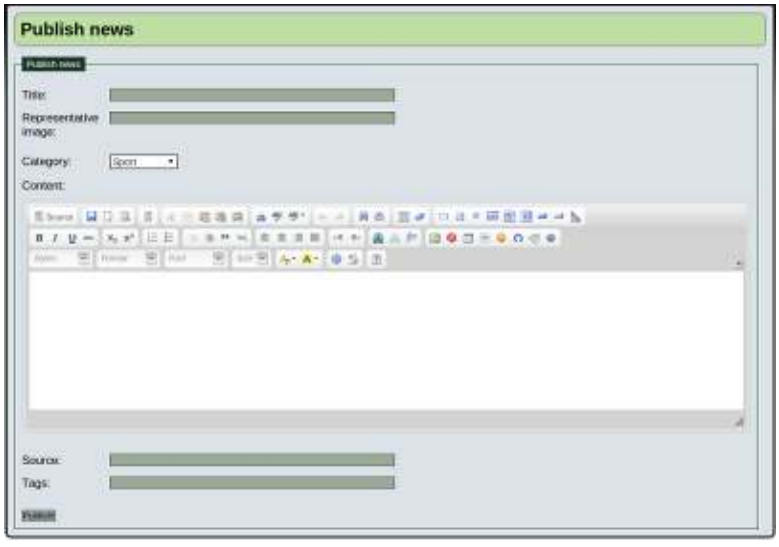
Slika 3.8: Opcije sortiranja kategorij

Zaradi velikega števila novic sem se odločil, da prikažem le devet zadetkov na stran. Za prehajanje med stranmi sem ponovno uporabil objekt Paginate, ki sem ga ustvaril v ta namen. Podatke o številki trenutne strani in sortiranja sem hranil v spletnem naslovu, saj menim, da v tem primeru ni potrebe po uporabi AJAX tehnologije.

3.3.4 Objavljanje in urejanje novic

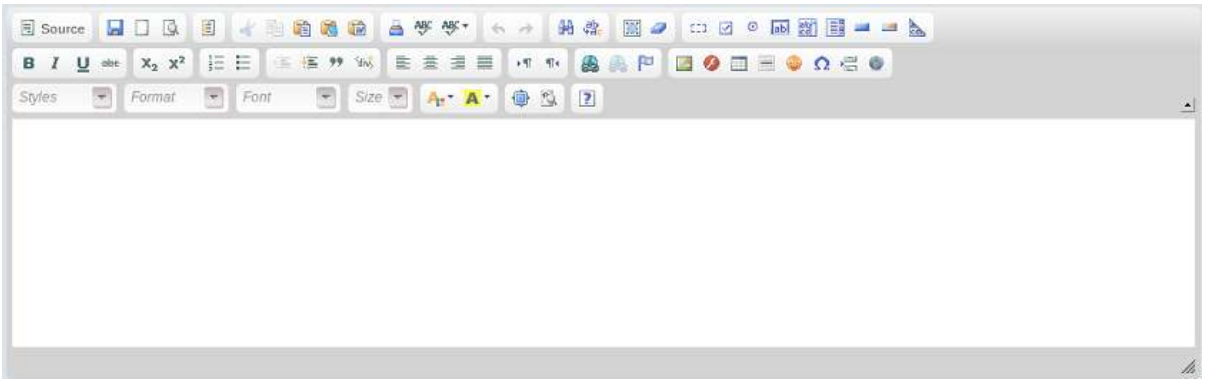
Vsak registriran uporabnik ima možnost objavljanja novic. Po prijavi se mu v glavnem meniju prikaže povezava »Publish«, ki ga preusmeri na stran za objavo novice (slika 3.9). Izgled strani je preprost, saj tako uporabnik lažje ve čemu kaj služi. Uporabnik mora vnesti naslednja polja:

- naslov novice,
- sliko, ki predstavlja novico,
- kategorijo pod katero novica spada,
- vsebino novice,
- vir novice in
- ključne besede novice.



Slika 3.9: Postopek objave novice

Pri sliki, ki predstavlja novico mora uporabnik vnesti celotno povezavo do slike. Novico uvrsti v kategorijo tako, da izbere kategorijo preko spustnega menija (angl. dropdown menu). Glavni del objave novice je vnos vsebine novice. Novica bi pri običajnem tekstovnem polju vsebovala samo besedilo in bi bila neprivlačna za bralce novic. Ta problem sem rešil z uporabo enega najpopularnejših urejevalnikov besedil znotraj spletnega brskalnika imenovanega CKEditor. Urejevalnik (slika 3.10) je napisan v jeziku JavaScript in je zelo enostaven za implementacijo. Uporabniku omogoča vstavljanje tabel, slik, spreminjanje pisave in poravnavo besedila ter še mnogo ostalih stvari, ki jih imajo ostali znani urejevalniki.



Slika 3.10: Urejevalnik CKEditor

Uporabnik mora po vsebini napisati še vir novice in nekaj ključnih besed za novico preko katerih bodo lahko drugi uporabniku to novico našli. Ključne besede so ločene s presledki in zapisane v bazo za izboljšano iskanje.

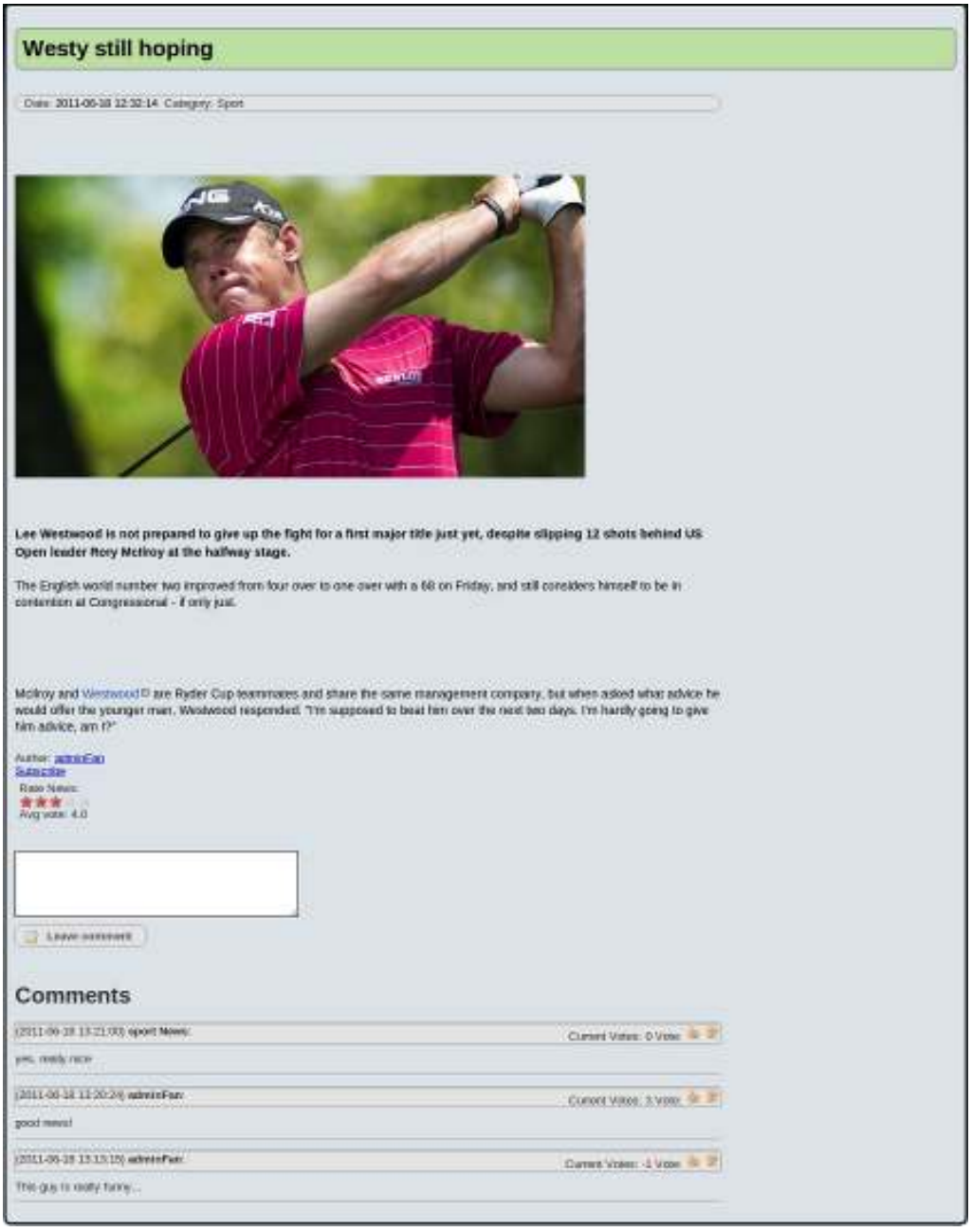
Ob vnešenih vseh podatkih o novici lahko uporabnik novico objavi s klikom na gumb »Publish«. V kolikor so bila vsa polja pravilno vnešena uporabniku izpiše, da je bila novica uspešno objavljena (slika 3.11).

Slika 3.11: Izpis o uspešno objavljeni novici

Uporabnik ima po objavi novice možnost urejanja le-te. Novico lahko uporabnik ureja tako, da na glavnem meniju izbere možnost »My profile«, ki prikaže profil uporabnika in seznam vseh novic, ki jih je objavil. Novica v seznamu ima možnost klika na povezavo »edit«, ki uporabnika preusmeri na urejanje izbrane novice. Urejanje novice sem želel ohraniti čim bolj podobno objavljanju, zato je edina razlika, da so bila polja pri urejanju že izpolnjena in jih ima uporabnik možnost spreminjati. Po končanem urejanju uporabnik s klikom na gumb »Update« shrani spremembe novice v bazo.

3.3.5 Pregled in ocenjevanje novice, objavljanje komentarjev in naročanje na uporabnika

Obiskovalec ima možnost prebrati celotno novico na katerikoli strani, kjer je kratek opis novice, in sicer s klikom na povezavo »read more«. Ob kliku, ga preusmeri na stran, kjer je objavljena celotna novica (slika 3.12).



Slika 3.12: Prikaz novice

Prva stvar, ki se zgodi ob prikazu celotne novice, je, da se števec ogledov trenutne novice poveča. Vsaka novica ima povprečno oceno, ki je izračunana na podlagi ocen dodeljenih s strani uporabnikov. Aplikacija ni izdelena na način, da bi administrator lahko sledil kateri uporabnik je podal kakšno oceno, zato je bilo ocenjevanje lažje implementirati. Novica ima atributa ratingSum, ki predstavlja vsoto vseh ocen in ratingNr, ki predstavlja število ocen. S tema podatkoma sem izračunal povprečno oceno novice. Za ocenjevanje sem uporabil enega izmed JQuery vtičnikov, ki sem ga našel na internetu. Stvar sem si malo drugače zamislil, kot je bila implementirana, zato sem moral skripto tudi nekoliko spremeniti, da je poslala JSON zahtevo na strežnik.

```
$.getJSON(container.url+String(rate), {}, function(data) {
    $("#rate2").html(data.result);
    $("#rate2").removeClass('rating');
});
```

Slika 3.13: Ocenjevanje novice

Prijavljeni uporabniki imajo tudi možnost naročitve na novice uporabnika s klikom na gumb »subscribe«. Uporabniki so danes že vajeni interaktivnih spletnih strani, zato sem, da bo stvar bolj interaktivna, tudi to implementiral s pomočjo jezika JavaScript.

V današnjem času obiskovalci zelo radi komunicirajo med seboj o novicah. Veliko strani ponuja objavljanje komentarjev pod novicami (slika 3.14), zato sem tudi sam dodal to možnost. Komentira lahko samo prijavljen uporabnik, saj bi v nasprotnem primeru lahko prišlo do neželjenega avtomatskega objavljanja komentarjev. Vsak komentar lahko uporabnik oceni kot dober ali slab. V Primeru, da je imel komentar več pozitivnih ocen je številka, ki predstavlja oceno komentarja obarvana z zeleno barvo, v nasprotnem primeru je številka obarvana rdeče. Težava na katero sem naletel pri implementaciji komentarjev je bila pri skokih v novo vrstico. Funkcionalnost sem implementiral s pomočjo jezika JavaScript. Pravilen zapis skokov v novo vrstico pri komentarjih sem dosegel tako, da sem dodal naslednjo vrstico v skripto.

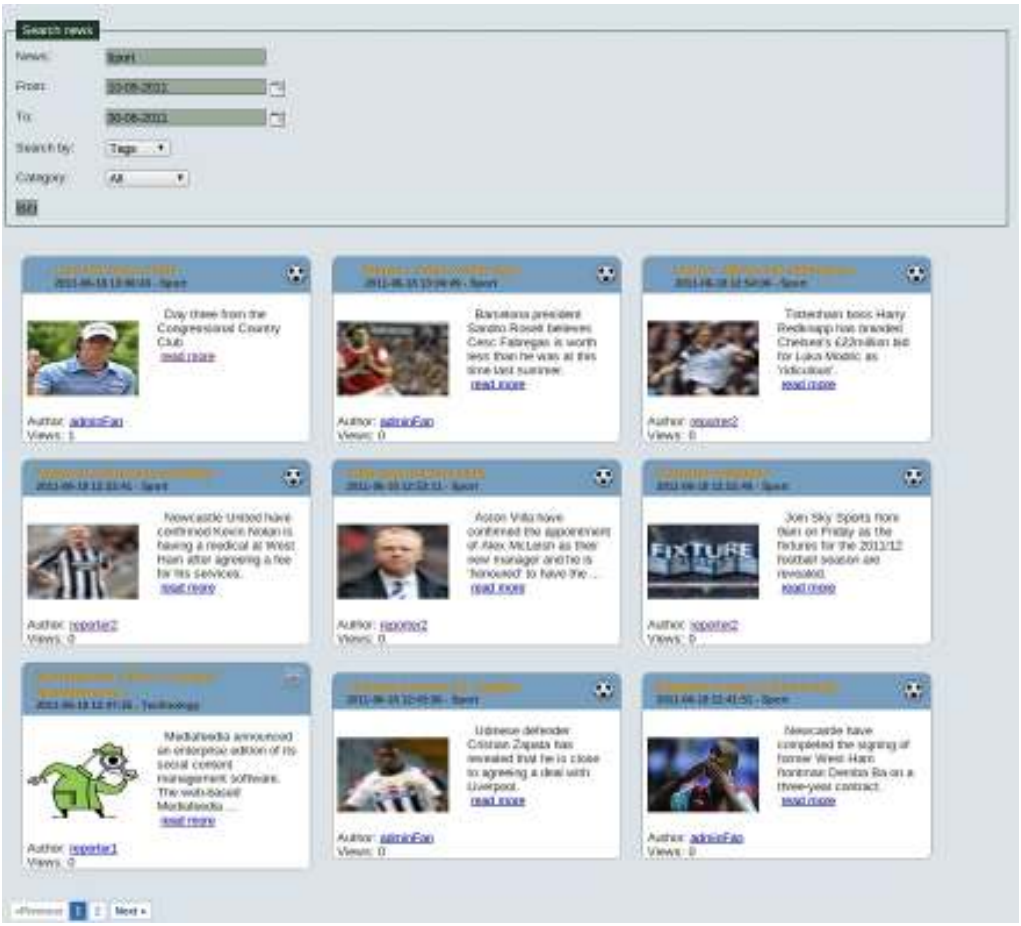
```
commentMsg = commentMsg.replace(/\n/g,"<br>");
```

Slika 3.14: Prikaz komentarjev pod novico

3.3.6 Iskanje novic

Poleg »live search« modula, sem implementiral tudi dodaten iskalnik, ki je namenjen podrobnejšemu iskanju (slika 3.15). Vnosna polja za iskanje so bila:

- iskani niz (ključne besede, po katerih bo iskal),
- izbira intervala datuma (interval v katerem je bila novica objavljena. Implementiral sem ju s pomočjo JavaScript skripte, ki sem jo našel na internetu),
- področje iskanja (uporabnik ima na voljo izbiro med »Tags«, »Title«, »Content« in »All«) in
- izbira kategorije (uporabnik lahko izbere kategorijo novice).



Slika 3.15: Naprednejši iskalnik

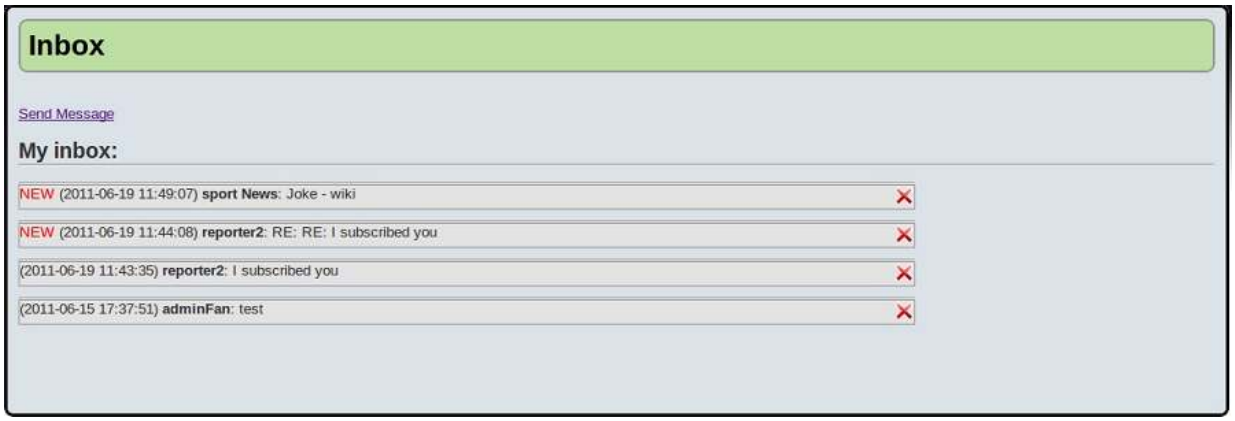
Iskanje po več besedah sem implementiral na zelo preprost način z naslednjimi vrsticami.

```
orQuery = []
for x in searchString.split(" "):
        orQuery.append(areaFilter.like("%%%s%%" % (x)))
allNews = allNews.filter(or_(*orQuery)).order_by(desc(News.sendDate))
```

Zaradi možnosti večjega števila zadetkov, sem tudi tukaj uporabil objekt Paginate za generiranje povezav na prejšnje oziroma naslednje strani. Za razliko od implementacije pri pregledu novic po kategorijah, sem tukaj stvar implementiral s tehnologijo AJAX. Glavni razlog za to odločitev je bil, da preprečim ponovno pomikanje navzdol po strani in da ohranim vnešene podatke, saj bi v nasprotnem primeru pri novem iskanju z zelo majhno spremembo uporabnik moral ponovno vnesti podatke iskanja. Podatke bi sicer lahko ohranil tudi pri ponovnem nalaganju strani vendar se mi je zdela rešitev s tehnologijo AJAX elegantnejša. Ena izmed prednosti uporabe AJAX tehnologije je tudi, da se spremembe hitreje prikažejo, saj brskalniku ni potrebno ponovno izrisati celotne strani temveč le del.

3.3.7 Pregled in pošiljanje zasebnih sporočil

Veliko uporabnikov želi komunicirati z ostalimi uporabniki in hkrati ne želijo izdati svojih kontaktnih podatkov, kot je recimo naslov elektronske pošte. V takih primerih se izkaže, da je zelo dobro implementirati sistem za pošiljanje sporočil med uporabniki. Prijavljenemu uporabniku sem dodal možnost klika na gumb »Inbox«, kjer lahko pregleda seznam prejetih sporočil (slika 3.16). V oklepajih je napisana številka, ki predstavlja število neprebranih sporočil uporabnika.



Slika 3.16: Seznam prejetih sporočil

S klikom na »Inbox« uporabnika preusmeri na stran z njegovimi prejetimi sporočili. Neprebrana sporočila so označena z rdečim besedilom »New«, tako da uporabnik takoj opazi neprebrana sporočila.

Pri pošiljanju sporočila mora uporabnik nujno vnesti naslednja polja:

- Naziv prejemnika (Receiver),
- Zadeva (Subject) in
- Vsebina sporočila (Content).

V primeru, da želi uporabnik pošiljatelju odgovoriti na sporočilo, ima možnost klika na gumb »reply«, kateri avtomatsko vnese podatke o pošiljatelju in zadevi s predpono »RE: » (slika 3.17).

Slika 3.17: Prikaz odgovorjanja na sporočilo

Na vnosnih poljih sem naredil tudi validacijo. To sem implementiral s pomočjo knjižnice WTForms, ki razvijalcu omogoča enostavno dodajanje pogojev. Ti morajo biti izpolnjeni ob potrditvi vnosa.

```
msgReceiver = TextField("Receiver:", validators=[
                    required(message="Field receiver cannot be empty."),
                    userExists,
                    length(max=20)                ])
```

V primeru, da je katero izmed polj prazno oziroma napačno vneseno uporabnika na to opozori. Prav tako uporabnika opozori v primeru, da vpisani prejemnik ne obstaja. Ob uspešnih vnosih se uporabniku pojavi opomba o uspešno poslanem sporočilu. Uporabniku je omogočeno tudi brisanje sporočil s klikom na znak »X«.

3.3.8 Objekt za oštevilčevanje strani

V naslednjem poglavju bom opisal funkcionalnost objekta Paginate, katerega sem že omenjal v prejšnjih poglavjih. Za kreiranje objekta sem se odločil, ker sem se zavedal, da bom moral na nekaterih straneh implementirati oštevilčevanje strani. Metode, ki jih objekt zajema so:

__init__(self, query, page, per_page, hrefUrl="")

Metoda predstavlja konstruktor objekta, ki prejme tri obvezne parametre in enega poljubnega. Query parameter predstavlja poizvedbo nad katero želimo iterirati po straneh. Parameter page nam pove katero stran želimo prikazati, parameter per_page pa nam pove koliko zadetkov naj

bi bilo na eni strani. Zadnji (neobvezni) parameter hrefUrl sem dodal za lažje generiranje povezav na naslednje oz. prejšnje strani tako, da sem podal predpono URL naslova. Ob inicializaciji objekta se ustvari paginateQuery kateri pridobi podatke o zahtevani strani, na koncu inicializacije pa se izvede še klic metode loadPagingBottom().

hasNext(self)

Metoda vrne vrednost 'True' v primeru da ni na zadnji strani. To sem enostavno izračunal tako, da sem preveril ali velja naslednji izraz:

število vseh zadetkov > trenuten odmik + število zadetkov na stran.

hasPrev(self)

Metoda vrne vrednost 'True' v primeru da ni na prvi strani oziroma če je številka trenutne strani večja od ena.

loadPagingBottom(self)

Ustvaril sem dinamično generiranje povezav na naslednje strani. Za to je poskrbela metoda loadPagingBottom, ki je generirala seznam z informacijami o vsaki povezavi, ki bo prikazana za skoke med stranmi. Seveda nisem želel, da bi mi v primeru stotih strani izpisalo povezave do vseh strani, saj bi postala stvar hitro nepregledna. V ta namen, sem se odločil da napišem algoritem, ki bo znal prikazati le povezave začetnih in končnih strani ter tistih, ki so v bližini trenutne strani. Ostale strani bo izpustil in dopisal tri pike vmes.

makeHrefLink(self, pageNr)

Metoda je namenjena generiranju povezave na stran, katera je podana kot parameter. V njej se izvede tudi klic metode getPageClass.

getPageClass(self, page)

Pri podani številki strani, metoda vrne vrednost, ki naj bi jo imel atribut 'class'. V vseh primerih razen v primeru da je trenutna podana stran enaka tisti na kateri se nahajamo vrne prazen niz.

renderPagingBottom(self, template)

Parameter template je ime datoteke, kateri podamo naš objekt. Datoteka pri prevajanju na podlagi našega objekta izriše povezave za iteracijo po straneh in vrne generirano HTML kodo.

Primer uporabe objekta:

```
q = News.query.options(eagerload_all(News.user,
User.following)).filter(News.user_id.in_(user_ids)).order_by(News.sendDate.
desc())
paginate = Paginate(q,2,6)
```

4 Sklepne ugotovitve

Diplomsko delo zajema načrtovanje in razvoj spletne aplikacije. Izdelava aplikacije je posledica problema, ki ga vidim pri pregledovanju novic na spletnih straneh. Pri razvoju sem naletel na nekaj manjših težav zaradi slabega poznavanja orodja Elixir. Tako nisem znal povezati novice z eno izmed definiranih kategorij, vendar sem to težavo rešil z malo truda in pomoči s strani mojih kolegov. Naslednja težava s katero sem se soočil je bila, da sem želel uporabiti Paginate objekt, ki ga vsebuje razširitev Flask-SQLAlchemy. Po posvetu s kolegi, ki sem jih spoznal preko interneta med katerimi so tudi razvijalci okolja Flask, sem prišel do zaključka da bi bilo lažje spisati svoj objekt, saj sem uporabljal Elixir in ne njihove razširitve. Pisanje objekta se je izkazalo za dokaj preprosto in je bil problem s tem hitro rešen.

Za nadaljni razvoj bi lahko implementiral predvsem pametnejše iskanje zadetkov, saj trenutno iskanje ne zajema neke logike temveč le prikaže rezultate, ki vsebujejo iskani niz. Ena izmed razširitev bi lahko bila prikaz trenutno priljubljenih novic. Veliko bi se lahko storilo tudi na samem izgledu aplikacije, saj je trenutno zelo okrnjen. V primeru, da bi se aplikacija razširila, bi bilo mogoče potrebno optimizirati nekatere dele kode, saj bi v nasprotnem primeru aplikacija potrebovala preveč časa za pridobivanje podatkov. Za konec bi bilo potrebno izboljšati še varnost same aplikacije.

Priloge

Programska koda

customObjects.py

```
from flask import render_template
import re
import math

class Paginate():

    query = ""
    paginateQuery = ""
    page = 0
    per_page = 0
    limit = 0
    offset = 0

    countAll = 0
    hasNext = False
    hasPrev = False

    hrefUrl = ""
    curPageClass = "curPage"
    pagesList = []

    def __init__(self, query, page, per_page, hrefUrl=""):
        self.query = query
        self.page = page
        self.per_page = per_page
        self.limit = per_page
        self.offset = (page-1)*per_page
        self.countAll = self.query.count()
        self.paginateQuery =
self.query.limit(self.limit).offset(self.offset).all()

        self.hrefUrl = hrefUrl

        self.loadPagingBottom()

    def hasNext(self):
        return self.countAll > self.offset+self.per_page

    def hasPrev(self):
        return self.page > 1

    def loadPagingBottom(self):
        allPages =
int(math.ceil(float(self.countAll)/float(self.per_page)))
        print "ALL PAGES:", allPages, ", countAll:", self.countAll, ",
per_page:", self.per_page
        l = []
        lTriPike = False
```

```
        dTriPike = False
        curRange = 3   #range v katerem bodo levo in desno prikazani
zadetki
        for i in range(1, allPages+1):
            if abs(i - self.page) < curRange-1: # ce je v rangu na sredini
                l.append([i, self.makeHrefLink(i), "link" ])
            elif i-curRange <= 0:   #leva stran page-i
                l.append([i, self.makeHrefLink(i), "link" ])
            elif i-allPages+curRange >0:  #desna stran page-i
                l.append([i, self.makeHrefLink(i), "link" ])
            else:

                    if i < self.page and not lTriPike:
                        l.append([i, "...", "dots"])
                        lTriPike = True
                    elif i > self.page and not dTriPike:
                        l.append([i, "...", "dots"])
                        dTriPike = True

        self.pagesList = l


    def makeHrefLink(self, pageNr):
        return """<a href="%s%s" %s>%s</a>""" % (self.hrefUrl, str(pageNr),
self.getPageClass(pageNr), str(pageNr))

    def getPageClass(self, page):
        if self.page == page:
            return """class="%s" """ % (self.curPageClass)
        return ""

    def renderPagingBottom(self, template):

        return render_template(template, paginate=self)


#-----------------------------------------------------
#       CUSTOM FUNCTIONS
#-----------------------------------------------------

def remove_html_tags(data):
    p = re.compile(r'<.*?>')
    return p.sub('', data)
```

usefulFunctions.js

```
function is_int(value){
  if((parseFloat(value) == parseInt(value)) && !isNaN(value)){
      return true;
  } else {
      return false;
  }
}
```

```
/*
 * Main
 *
 */
$(document).ready(function() {

    //-------------------------------------------------------------------------
---------------------
    //                                    INDEX Z NOVICAMI
    //-------------------------------------------------------------------------
---------------------
   //Nalozi zacetne paginge z zacetno (default) stranjo
   goPaginate('ajaxSubscribeIndex',1, 0);

    //-------------------------------------------------------------------------
---------------------
    //                                        NEWS
    //-------------------------------------------------------------------------
---------------------

    var newsId = location.href.split("/")[location.href.split("/").length-
1];
    $('#rate2').rating($SCRIPT_ROOT + '/member/newsRate/'+newsId+'/',
{maxvalue:5, curvalue:3});


});

$(function() {


    $("#searchLiveField").keyup(function(e){
            if ($("#searchLiveField").val().length < 2 ){
                $("#searchLiveFieldResults").slideUp("slow");
            } else if ($("#searchLiveField").val().length > 1){


                    $.getJSON($SCRIPT_ROOT +
'/ajaxLiveSearch/'+$("#searchLiveField").val() ,{ },
                        function(data) {
                            $("#searchLiveFieldResults").html(data.result);
                            $("#searchLiveFieldResults").slideDown("slow");
                        }
                    );

            }
    });


});



function goPaginate(caller, page, doEffect) {
    //callers = ["ajaxSubscribeIndex"]
    $.getJSON($SCRIPT_ROOT +
'/member/ajaxPagingIndex/'+caller+'/'+String(page), {
      //a: 2,
      //b: 3
```

```
    }, function(data) {
        //alert(divId);
      if(doEffect) runEffect("clip", caller);
      $("#"+caller).html(data.result);
      if(doEffect) runEffect("clip", caller);
    });

};


// run the currently selected effect
function runEffect(effect, divId) {
    // get effect type from
    //var selectedEffect = $( "#effectTypes" ).val();
    selectedEffect=effect;
    // most effect types need no options passed by default
    var options = {};
    // some effects have required parameters
    if ( selectedEffect === "scale" ) {
        options = { percent: 0 };
    } else if ( selectedEffect === "size" ) {
        options = { to: { width: 200, height: 60 } };
    }

    // run the effect
    $( "#"+divId ).toggle( selectedEffect, options, 200 );
};

function goSearch(whichSearch, page) {
    if (whichSearch == "newsSearch") {
        var newsString = $('#searchNewsString').val();
        var newsDate1 = $('#searchNewsDate1').val();
        var newsDate2 = $('#searchNewsDate2').val();
        var area = $('select[name="area"]').val();
        var category = $('select[name="category"]').val();
        $.getJSON($SCRIPT_ROOT +
'/searchLive/'+String(newsString)+'/'+String(newsDate1)+'/'+String(newsDate
2)+'/'+String(area)+'/'+String(category)+'/'+String(page), {
            /*newsString: newsString,
            newsDate1: newsDate1,
            newsDate2: newsDate2
            */
            }, function(data) {
                $("#searchDiv").html(data.result);
            }
        );
    }
}

function goSubscribe() {

    repName = $("#repName").val();
    $.getJSON($SCRIPT_ROOT + '/member/subscribe/'+String(repName), {

        }, function(data) {
            //alert(data.result);
            $("#subscribeButton").html(data.result);
        }
    );
}
```

```
function getCurPage() {

    if(location.href.replace("/news/", "") != location.href)    return
"news";
    if(location.href.replace("/member/user/", "") != location.href)
return "userInfo";
}


function imgError(img) {
    img.src = "http://1.bp.blogspot.com/_Sd63dhXRW4M/S6onBtXS-
eI/AAAAAAAAAFM/2mDIqo11zcU/s1600/image-not-found.gif";
    img.onerror = "";
    return true;
}

function showSubCatNavigation(category) {

    $("#subCatNavigation").html("<ul class=\"solidblockmenu\">"+
                                "<li><a
href=\"/category/"+category+"/date/d/1\">Last news</a></li>"+
                                "<li><a
href=\"/category/"+category+"/views/d/1\">Most viewed</a></li>"+
                                "<li><a
href=\"/category/"+category+"/rating/d/1\">Top rated</a></li>");
}

function commentNews() {

    var newsId = location.href.split("/")[location.href.split("/").length-
1];
    var commentMsg = $("#commentTextArea").val();
    commentMsg = commentMsg.replace(/\n/g,"<br>");  //hvala lepa
stackoverflow za pomoc.

    //alert(commentMsg);
    //alert("News ID:"+newsId);
    $.getJSON($SCRIPT_ROOT +
'/member/commentNews/'+String(newsId)+'/'+commentMsg, {

            }, function(data) {
                $("#addComment").html(data.result);
            }
    );

}

function commentThumbs(commentId, thumb) {

    $.getJSON($SCRIPT_ROOT +
'/member/commentVote/'+String(commentId)+'/'+String(thumb), {

            }, function(data) {
                $("#comment"+commentId).html(data.result);
                $("#commentThumbs"+commentId).html("");
            }
    );
}
```

Kazalo slik

Literatura in viri

[1] (2011) HTML. Dostopno na:
http://en.wikipedia.org/wiki/HTML

[2] (2011) CSS. Dostopno na:
http://en.wikipedia.org/wiki/Cascading_Style_Sheets

[3] (2011) Python. Dostopno na:
http://www.python.org

[4] (2011) JavaScript. Dostopno na:
http://sl.wikipedia.org/wiki/JavaScript

[5] (2010) Flask. Dostopno na:
http://flask.pocoo.org/

[6] (2010) Aptana studio. Dostopno na:
http://www.aptana.com

[7] (2011)PhpMyAdmin. Dostopno na:
http://en.wikipedia.org/wiki/PhpMyAdmin

[8] (2011) MySQL. Dostopno na:
http://en.wikipedia.org/wiki/MySQL

[9] (2011) PowerDesigner. Dostopno na:
http://www.sybase.com/products/modelingdevelopment/powerdesigner

[10] (2011) SQLAlchemy. Dostopno na:
http://www.sqlalchemy.org/

[11] (2007) Elixir. Dostopno na:
http://elixir.ematia.de/trac/wiki

[12] (2008) Jinja2. Dostopno na:
http://jinja.pocoo.org/docs/

[13] (2009) WTForms. Dostopno na:
http://wtforms.simplecodes.com

[14] (2010) JQuery. Dostopno na:
http://www.jquery.com

[15] (2006) Modeliranje primerov uporabe. Dostopno na:
http://mitjab.homelinux.net:8080/~mitjab/Sola/VAJE/OIS/Modeliranje%20primerov%20uporabe%20(PU)/modeliranje_USE_CASE.pdf