Projekt iz spletnega programiranja – 2. faza

Izvedba strežniškega dela aplikacije

December 2016, v1.2

Rok oddaje na učilnici: 15. 1. 2017, možni (in priporočeni) predhodni zagovori.
Predavatelj: dr. Aleš Smrdel
Asistenta: Matevž Jekovec, dr. Gašper Fele-Žorž

Uvod

V drugi fazi boste nadaljevali z delom na spletni aplikaciji in razvili strežniški del. Razvoj je v grobem sestavljen iz naslednjih korakov: (I) načrtovanje podatkovnega modela in diagrami poteka s pripadajočimi URL-ji, (II) izvedba podatkovnega modela v aplikaciji, (III) izvedba poslovne logike in pogledov in (IV) objava, testiranje ter dokumentacija aplikacije.

Izbira primernega ogrodja

Izberite ogrodje, ki se vam zdi najbolj primerno in upošteva načela MVC (angl. *model-view-controller*) pristopa razvoja aplikacij. Nekaj priporočenih ogrodij za izbrane programske jezike:

- **Python**: Django, Flash, Pyramid;
- **Ruby**: Ruby on Rails, Rack, Sinatra, Lotus;
- **PHP**: Codeigniter, Zend framework, CakePHP, Symfony, Laravel, Nette;
- **Javascript**: Node.js, Meteor, MEAN, DerbyJS, Sails.js;
- **.NET**: ASP.NET in ASP.NET MVC;
- **Java**: Spring, Dropwizard, Play, Spark.

Načrtovanje podatkovnega modela

Glede na namen, zasnovo in načrt aplikacije, ki ste jih definirali v prvi fazi, boste v drugi fazi načrtali podatkovni model in ustrezno pripravili strukturo podatkovne zbirke, vključno s:

- tabelami (z ustreznim indeksiranjem, omejitvami in relacijami),
- pogledi, procedurami in funkcijami,
- prožilci (angl. *triggers*),
- transakcijami.

Narišite ustrezno strukturo tabel z medsebojnimi relacijami. Uporabite lahko orodja za zasnovo in administracijo baz (MySQL Workbench, Microsoft SQL Server), splošno-namenski Graphviz s temo za izris baz (https://code.djangoproject.com/wiki/DjangoGraphviz) ali pa svinčnik in papir. Načrt podatkovnega modela shranite v mapo docs/.

Izvedba podatkovnega modela

Pri izvedbi podatkovnega modela glede na izbrano ogrodje in programski jezik uporabite ustrezen ORM *(*angl. *Object Relational Mapping)*. Običajno imajo ogrodja ORM že vključen oz. obstaja priporočen ORM za dotičen programski jezik. V programski kodi definirajte razrede in njihove atribute, ki se preslikajo v ustrezno tabelo in stolpec. Bodite pozorni na podatkovne tipe, uporabljene v SQL (npr. VARCHAR namesto TEXT, če so nizi krajši itd.). Pri hrambi uporabniških geslih uporabite soljeno zgoščevalno funkcijo.

ORM zagotavlja robustnejše delovanje aplikacije in enostavnejši razvoj, vendar pri velikih aplikacijah lahko upočasni njihovo delovanje. Zgodi se tudi, da pri nekaterih poizvedbah ORM ni dovolj in morate ročno zgraditi poizvedbo SQL. Dober ORM vsebuje bogato zbirko že vgrajenih operacij, obenem pa omogoča čisto vgradnjo lastnih poizvedb SQL.

Kljub temu, da uporabljate ORM, mora vaša aplikacija v ozadju uporabljati relacijsko zbirko tipa SQL (npr. MySql/MariaDB, Microsoft SQL Server, PostgreSQL, Sqlite ipd.). Pri oddaji naloge morate priložiti zagonske skripte SQL, ki vzpostavijo relacijsko bazo. Skripte zgenerirajte z izbranim ogrodjem neposredno iz podatkovnega modela (npr. ukaz `migrate` v Rails in Django). Skripte shranite v mapo install/.

Diagram poteka in preslikava naslovov

Pri laboratorijskih vajah v prvi fazi smo risali diagrame poteka uporabe aplikacije z vidika uporabnika. Sestavite celoten diagram za vse vloge uporabnikov v vaši aplikaciji in k vsakemu stanju v diagramu pripišite pripadajoč relativen URL. Poleg tega pripišite tip vsebine, ki jo strežnik vrača (HTML ali JSON), ter tip zahtevka HTTP (GET/POST, PUT/DELETE itd.). Za risanje diagrama lahko uporabite prosto dostopna spletna orodja (npr. www.draw.io) ali namizne aplikacije (npr. ArgoUml). Diagram poteka oddajte kot sliko ali PDF skupaj z dokumentacijo projekta v mapo docs/.

Izvedba poslovne logike, ACL, obrazci in beleženje

Poslovna logika povezuje podatkovni model in pogled ter glede na vsebino seje uporabnika krmili delovanje aplikacije. Pri spletnih aplikacijah je vsaka vstopna točka oz. akcija pokrita z enim izmed krmilnikov (angl. *controllers*). Vaša naloga je, da glede na načrt strani zasnujete in sprogramirate vse krmilnike in s tem poslovno logiko.

Glede na izbrano ogrodje poskrbite za sejo uporabnika (angl. *sessions*) in določite, katere akcije v vaši aplikaciji lahko prožijo posamezne vrste uporabnikov (administrator, navaden uporabnik, …). Omenjeno obnašanje sprogramirajte z uporabo ACL (angl. *access control list*), ki je običajno že vgrajen v izbrano ogrodje.

Posebno pozornost namenite obrazcem. Vse vnose ustrezno validirajte na strani strežnika. Validacija na strani brskalnika ne zagotavlja veljavno izpolnjenih vnosnih polj, saj zahtevek HTTP uporabnik brez težav sestavi sam, mimo brskalnika, in ga pošlje na strežnik.

Premislite, katera mesta v vaši aplikaciji so kritična in sprogramirajte ustrezno beleženje (angl. *logging*). Npr. če programirate spletno trgovino, je kritičen korak oddaja naročila. V primeru programerske napake je lahko oddaja naročila neuspešna. V tem primeru morate zagotoviti, da je v dnevniških datotekah dovolj podatkov, da administrator lahko restavrira naročilo in ga obdela oz. povpraša stranko o vsebini naročila. Za beleženje uporabite obstoječe mehanizme, ki vam jih prinaša ogrodje. Običajno na enem mestu v konfiguracijski datoteki aplikacije nastavite beleženje v datoteko, v podatkovno zbirko, obvestilo po eletronski pošti itd.

Ravno tako poskrbite za konsistentnost vaše podatkovne baze. npr. pri spletni trgovini proces oddaje naročila vsebuje več korakov (podatki glede plačila, naslov za dostavo, končna potrditev). Vsak korak »dogradi« košarico in ločeno komunicira s podatkovno bazo. Lahko uporabite transakcije in v primeru, da uporabnik ne želi oddati naročila, transakcijo stornirate.

Izvedba pogleda

Pogled razvijte tako, da je popolnoma ločen od poslovne logike (krmilnika) in podatkovnega modela. Smiselno naj uporablja gradnike, ki jih boste pripravili: zaglavje strani, noga strani, tabelarična struktura (ali kakšna druga, odvisno kako ste si stran zamislili) za prikaz podatkov, del strani za prikaz podatkov o uporabniku ipd.
Pri gradnji končne kode HTML uporabite ustrezen pogon za predloge (angl. *templating engine*), ki vam ga ponuja izbrano ogrodje. Ta lahko teče tudi na strani odjemalca (npr. AngularJS).
CSS iz prve faze lahko sedaj opišete posredno z jezikom SASS ali LESS, ki vam zagotovita ves čas veljavno skladnjo CSS in podprtost na širokem naboru brskalnikov.
Aplikacijo prevedite v vsaj en dodaten jezik. Uporabite mehanizme za lokalizacijo, ki vam jih ponuja ogrodje oz. programski jezik.

Samodejno generirana dokumentacija

Iz vaše izvorne kode lahko zgenerirate referenčni priročnik (angl. *reference guide*), namenjen bodočim razvijalcem. Spišete lahko tudi kratek uporabniški priročnik (angl. *user's guide*), namenjen končnim uporabnikom vaše aplikacije. Pri tem uporabite orodja za samodejno gradnjo priročnikov iz predloge (npr. sphinx, doxygen, autodoc itd.). Končno verzijo dokumenatcije shranite v mapo docs/.

Testiranje, profiliranje in zvezna integracija

Izdelek temeljito preklikajte, preverite če so vsi elementi skladni, če koda deluje pravilno in če se vsi podatki prikazujejo pravilno. Preverite, da lahko uporabniki dostopajo le do njim dovoljenih vsebin.
Predvsem za testiranje modela in poslovne logike uporabite testiranje enot v izbranem programskem jeziku. Za preverjanje pravilnega delovanja funkcionalnosti strani prek uporabniškega vmesnika uporabite katero od orodij za funkcionalno oz. integracijsko testiranje (z uporabo Selenium, Windmill, CasperJS itd.). Vse teste shranite v ustrezno mapo, ki jo določa vaše ogrodje.
Za ugotavljanje odzivnosti vaše spletne aplikacije izvedite performančne teste (z uporabo JMeter, Selenium, Gatling, Tsung, The Grinder itd.). Izmerite pričakovano število zahtevkov na sekundo, ki jih lahko vaša aplikacija obdela. Nato z uporabo orodij za profiliranje v izbranem jeziku ugotovite ozka grla v vaši aplikaciji (npr. cProfile, valgrind, Visual Studio Profiling Tools itd.). Izsledke profiliranj shranite v mapo docs/.
Z uporabo zvezne integracije (angl. *Continuous Integration*) poskrbite za »zdravje« vaše kode, da boste z nadaljnjim razvojem ustvarili čim manj novih hroščev in pokvarili delovanje aplikacije za nazaj (angl. *regressions*). Vse zgoraj omenjene vrste testov podpira orodje za zvezno integracijo Jenkins. GitHub za uporabnike brez instance Jenkins omogoča integracijo z zunanjo storitvijo Travis-CI.

Objava in optimizacija

V mapi install/ ustvarite datoteko INSTALL.md z opisom postopka namestitve vaše aplikacije na izbrani platformi, npr. na fizičnem strežniku z nameščenim Ubuntu LTS oz. Microsoft Windows Server ali oblaku Heroku, Amazon WS, Azure in podobnih. Poleg navodil priložite skripte orodij za konfiguracijo okolja (npr. Puppet ali Chef) in avtomatizirajte postopek objave. Če objavljate na oblak Heroku, uporabite in prilagodite njim lastne skripte. Aplikacijo lahko objavite namesto na strežnik oz. oblak tudi lokalno v izolirano okolje Docker ali Vagrand. V vsakem primeru morate poskrbeti za namestitev vseh odvisnosti, ki jih vaša aplikacija potrebuje.
Pri objavi poskrbite, da bo koda in statična vsebina aplikacije pravilno prevedena (npr. izvorne datoteke SASS/LESS v CSS, koščki pogledov v končni pogled) in izvedeni morebitni popravki podatkovne zbirke. Poleg tega v produkcijskem okolju poskrbite za učinkovito delovanje vaše spletne aplikacije. Preučite in uporabite mehanizme za predpomnjenje (angl. *caching*) bodisi na nivoju spletnega strežnika, na aplikacijskem nivoju ali na nivoju podatkovnega modela. Poskusite optimizirati velikost vaših virov (npr.

optimalne velikosti slik glede na velikost zaslona odjemalca, pomanjšana koda javascript – angl. *minify*). Stran lahko optimizirate tudi tako, da odjemalec v enem zahtevku pridobi celotno spletno stran z vsemi viri, vgrajenimi kar v datoteki HTML.
Alternativna možnost je, da uporabite HTTP/2 na vašem spletnem strežniku in aplikacijo ustrezno prilagodite (večji poudarek na predpomnjenju manjših resursov na strani odjemalca namesto združevanja celotne strani v eno datoteko HTML – angl. *inlining*).

Končni izdelek druge faze

Končni izdelek oddajte v zapisu ZIP, zgeneriranem iz repozitorija git. Poleg tega v datoteko ZIP vključite tudi zgenerirano dokumentacijo projekta. Datoteka naj vsebuje:

- Dokumentacijo projekta v mapi docs/:
  - načrt podatkovnega modela;
  - diagram poteka vaše aplikacije s pripadajočimi relativnimi URL naslovi, tipi vsebine in tipi HTTP zahtevkov;
  - poročilo o profiliranju aplikacije (rezultati performančnih testov, ozka grla aplikacije);
  - poročilo o optimizaciji aplikacije.
- Informacije o namestitvi v mapi install/:
  - navodila za namestitev spletne aplikacije (zahtevano programsko okolje, postavitev strežnika, postavitev baze, namestitev aplikacije);
  - zagonske skripte SQL za vzpostavitev relacijske baze;
  - skripte za objavo in zvezno integracijo (angl. *Continuous Integration*);
- Ustrezno mapo s testi glede na ogrodje:
  - testi enot;
  - funkcionalni in integracijski testi;
  - performančni testi.
- Izvorno kodo same spletne aplikacije.

Kriteriji za ocenjevanje

V drugi fazi je možno doseči 125 točk. Pogoj za uspešno opravlje vaje je v povprečju zbranih vsaj 50 % točk pri obeh fazah (pri posamezni fazi lahko zberete tudi manj kot 50 %, vendar mora biti zato preostala faza toliko boljše narejena).

Točkovalnik pri drugi fazi je naslednji:

Načrt podatkovnega modela	10
Izvedba podatkovnega modela v izbranem ogrodju in migracije	10
Uporaba ORM	5
Diagram poteka in preslikave naslovov (angl. *routes*)	5
Izvedba poslovne logike v izbranem ogrodju in njena ločitev od modela/pogleda	10
Validacija obrazcev na strani strežnika	5
Nadzor dostopa uporabnikov do virov (angl. *access control list*)	10
Izvedba pogleda v izbranem ogrodju s pogonom za predloge (angl. *templating engine*)	10
Dobra programerska praksa, komentarji, izbrana konvencija programiranja	10
Lokalizacija	5
Beleženje z uporabo izbrane metode (angl. *logging*)	5
Referenčni priročnik ali uporabniški priročnik za aplikacijo (Sphinx)	5
Testi enot	5
Funkcionalni in integracijski testi (Selenium)	5
Performančni testi in profiliranje (Selenium, JMeter)	10
Zvezna integracija (Jenkins, Travis-CI)	5
Objava (dokumentacija, Puppet, Chef, po potrebi skripte bash)	5
Optimizacija nalaganja strani (predpomnjenje, optimizacija vsebine, HTTP/2, minify)	5

Viri

Pri izdelavi si pomagajte s predlagano literaturo predmeta, tu pa je še nekaj povezav:

- SASS: http://sass-lang.com
- LESS: http://lesscss.org
- Optimizacija strani: http://designingforperformance.com/basics-of-page-speed
- Sphinx: http://www.sphinx-doc.org
- Regresijsko testiranje: http://smartbear.com/resources/articles/what-is-regression-testing/
- Selenium: http://www.seleniumhq.org
- JMeter: http://jmeter.apache.org
- Jenkins: https://jenkins.io
- Travis-CI: https://travis-ci.org
- Puppet: https://puppet.com
- Chef: https://www.chef.io
- Docker: https://www.docker.com
- Vagrand: https://www.vagrantup.com