Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Andrejak

Optimizacija prototipiranja pri razvoju spletnih aplikacij

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Pri procesu razvoja programske opreme je prototipiranje pogosto prisotno, vendar ga kljub temu večina razvojnih metodologij obravnava kot opcijski korak, ki se ga razvijalcem informacijskih sistemov zgolj priporoča. Številna podjetja pa pri svojem procesu razvoja vseeno prototipiranje uporabljajo, vendar v omejeni obliki - večinoma gre za enostavne prototipe zaslonskih mask. V okviru diplomske naloge zato raziščite področje prototipiranja informacijskih sistemov s poudarkom na naprednem prototipiranju spletnih aplikacij. Na podlagi izsledkov pregleda obstoječih pristopov predlagajte izboljšan pristop, za katerega pripravite prototip orodja, ki ga podpira. Predlagan pristop, ki naj vključuje tudi simulacijo zalednega dela in generiranje smiselnih naključnih podatkov, kritično ovrednotite na konkretnem primeru in izpostavite prednosti ter slabosti.

Za usmerjanje pri izdelavi diplomske naloge se zahvaljujem mentorju doc. dr. Dejanu Lavbiču. Posebna zahvala pa gre družini za vso vzpodbudo in potpljenje.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	Asynchronus JavaScript and XML	Asinhroni JavaScript in XML
API	Application Programming interface	Aplikacijski programski vmesnik
CSS	Cascading Style Sheet	Kaskadne slogovne predloge
HTML	Hyper Text Markup Language	Označevalni jezik za oblikovanje spletnih dokumentov
JSON	JavaScript Object Notation	Objektna notacija JavaScript
PHP	Hypertext Preprocessor	Splošno uporaben skriptni programski jezik, ki ga tolmači strežnik
SDLC	Systems development life cycle	Življenjska doba razvojnega procesa

Povzetek

Pri razvoju spletnih aplikacij pogosto uporabljamo prototipiranje. S prototipi se uporabnike v razvojni proces vključi že zelo zgodaj, kar ima lahko pozitivne učinke na sam proces. S pravilno interpretacijo uporabnikovih povratnih informacij se lahko razvojna ekipa izogne morebitnim težavam in razvoju nepotrebnih segmentov aplikacije.Problem, ki se pogosto pojavlja v praksi, so preveč enostavni prototipi, ki tako uporabnikom kot razvijalcem ne nudijo dovolj dobre ponazoritve končne aplikacije. Eno glavnih načel izdelovanja prototipov je v hitri izvedbi, zato se razvijalci, vsaj v začetnih verzijah, ne odločajo za izdelavo kompleksnejših prototipov. Namen diplomske naloge je predstaviti rešitev za izdelovanje boljših prototipov spletnih aplikacij tako, da za njihovo izdelavo ni potrebno veliko časa. Rešitev je izdelana kot JavaScript modul, ki z uporabo razširjenih atributov HTML elementov upravlja z mehanizmi za generiranje naključnih podatkov in simulacijo zalednega sistema. Razvijalcem omogoča hitrejšo izdelavo kompleksnejših prototipov, uporabnikom pa enostavno poročanje napak in podajanje povratnih informacij razvojni ekipi. V diplomski nalogi najprej predstavimo prototipiranje kot primerno metodologijo razvojnega procesa za razvoj spletnih aplikacij. Osrednji del naloge sta predstavitev rešitve za izdelovanje boljših prototipov in testiranje rešitve na praktičnem projektu. Testiranje se izvede kot primerjava procesa razvoja dveh aplikacij, pri čemer smo pri eni od njih uporabili omenjeno rešitev. Izkaže se, da imajo koncepti rešitve pozitiven učinek na razvojni proces, same funkcionalnosti pa bi lahko še dodatno razširili in s tem razvijalcem še bolj pomagali pri izdelavi prototipov.

Ključne besede: prototipiranje, dinamični prototipi, simulacija zalednega sistema, generiranje naključnih podatkov, JavaScript.

Abstract

When developing web applications we often use prototyping. Prototyping allows us to involve end users of an application in early stages of development process, which can have positive effects on process itself. The correct interpretation of user feedback can avoid potential problems during development. The problem that often occurs in practice, are too simple prototypes that do not provide good simulation of the final application. One of the main principles of prototyping is rapid implementation, so developers, at least in the initial versions do not decide to develop complex prototypes. The aim of this diploma thesis is to present a solution to make better prototypes of web applications, so that their development does not take much time. The solution is implemented as a JavaScript module that uses extended attributes of HTML elements for generating random data and simulation of backend system. It enables developers to accelerate the development of complex prototypes. The module also simplifies error reporting and giving feedback to the development team. In the first part of the thesis, we introduce prototyping as an appropriate method for the development of web applications. Next, we present the solution to make better prototypes and test the solution in practice. The test is conducted as a comparison of the development of two applications – one with the use of the aforementioned solution, the other without. We conclude with the presentation of the findings – namely that the proposed solution has a positive effect on the development process. We also present possibilities for further development that would improve prototyping even further.

Keywords: prototyping, dynamic prototypes, backend simulation, generating random data, JavaScript.

Poglavje 1

Uvod

Prototip je *poenostavljen model* oziroma prezentacija izdelka, ki ga želimo izdelati. Uporablja se za testiranje konceptov in procesov. Preko prototipa izdelek bolje spoznamo in ga zato uspešneje načrtujemo. Je tudi izvrstno orodje za boljšo komunikacijo med končnimi uporabniki izdelka ter razvojno ekipo [8].

Na področju razvoja spletnih aplikacij so ti poenostavljeni modeli pogosto preveč enostavi. Posledica so preskromni prototipi, ki končni izdelek ne simulirajo dovolj dobro. Podatki, nad katerimi operirajo, so pogosto presplošni ali pa jih je premalo. Uporabniki se tako ne morejo vživeti v realno situacijo uporabe končne aplikacije, zato so velikokrat njihove povratne informacije preskope za optimalen razvoj[11]. Z operativno verzijo aplikacije se uporabniki srečajo šele v kasnejših fazah razvojnega procesa. V primeru večjih težav ali pomankljivosti je lahko sanacija le-teh draga in pogosto pomeni podaljšanje izdelave projekta[5, 3].

Cilj te diplomske naloge je poiskati rešitev, da bi se omenjenim situacijam izognili brez povečevanja stroška izdelave prototipov. Predstavil bom rešitev, s katero lahko nadgradimo statične prototipe in tako bolj realno simuliramo končni produkt. Prototipe napolnimo s testnimi ali naključno generiranimi podatki ter simuliramo poizvedbe na zaledni sistem. Na drugi strani pa uporabnikom olajša komunikacijo z razvijalci v smislu enostavnega

poročanja težav ob testiranju. Ti mehanizmi nam lahko pomagajo pri detekciji morebitnih težav in pomanjkljivosti zasnove sistema dovolj zgodaj, da se izognemo neprijetnemu procesu prilagajanja ali celo menjavi že razvitega segmenta aplikacije. Obenem pa prihranijo čas pri izdelavi prototipov, saj razvijalcem olajšajo razvoj.

V poglavju 2 bomo spoznali prototipiranje programske opreme. Osredotočili se bomo na razvojno prototipiranje, ker je pri razvoju spletnih aplikacij bolj uporabo od prototipiranja za enkratno uporabo. Predstavili bomo prednosti in nevarnosti prototipiranja ter v katerih situacijah se je smiselno odločiti za uporabo te metodologije razvoja. V poglavju 3 bomo predstavili prototipiranje pri razvoju splethin aplikacij. Opisani bojo tipi prototipov, ki se uporabljajo za prototipiranje aplikacij. Izpostavili bomo pomankljivosti prototipov ter predlagali rešitev za boljše prototipiranje. Poglavje 4 bo opisovalo rešitev za boljše prototipiranje. Spoznali bomo mehanizme, ki razvijalcem, z malo truda, omogočijo izdelavo bolj realističnih prototipov. Uporabnost rešitve bomo preverili tako, da bomo primerjali razvoj dveh projektov. Predlagana rešitev je bila na enem projektu uporabljena tudi v praksi.

Poglavje 2

Vloga prototipiranja pri razvoju programske opreme

Prototipiranje ima lahko pri metodologiji razvoja programske opreme osrednjo vlogo. V tem primeru poteka razvoj z uporabo prototipov. Prototip programske opreme je približek sistema, ki ga želimo razviti in vsebuje zametke njegovih glavnih funkcionalnosti. Uporaba prototipov povečuje sodelovanje uporabnikov in razvijalcev v razvojnem procesu [1, 2]. Morebitne pomanjkljivosti in napake lahko s prototipiranjem detektiramo dovolj zgodaj in se izognemo velikim stroškom ter težavam njihove sanacije v kasnejših fazah razvojnega procesa [3].

Tehnika je še posebej uporabna ko:

- funkcionalnosti sistema niso dovolj definirane
- komunikacija med uporabniki in razvilajci ni dovolj dobra
- je veliko interakcije z uporabniki
- je zagotovitev uporabniških potreb bistvenega pomena

Da se prototipiranje izplača, mora biti prototip [4]:

- dovolj nujen za izvedbo
- enostaven za izdelavo
- vpliven in prepričljiv za uporabnike

Poznamo dva tipa metodologije prototipiranja: prototipiranje za enkratno uporabo (*angl. throwaway prototyping*) in razvojno prototipiranje (*angl. evolutionary prototyping*). Glavni cilj prvega tipa prototipiranja je testiranje hipotez in, kot namiguje že ime samo, jih po uporabi zavržemo. Ravno nasprotno se razvojni prototipi s časom nadgrajujejo, izpopolnjujejo in se uporabijo kot osnovo, na kateri je zgrajena končna rešitev.

2.1 Razvojno prototipiranje

Metodologija razvojnega prototipiranja temelji na procesu nadgrajevanja prototipa. Iz začetnega prototipa se navadno v več vmesnih verzijah (iteracijah) razvije končno rešitev. Poznamo dva podtipa razvojnega prototipiranja, ki se pri razvoju spletnih aplikacij pogosto uporabljata vzporedno. Gre za inkrementalno in ekstremno prorotipiranje [5].

Pri inkrementalnem prototipiranju gre za izdelavo več manjših prototipov, ki se na koncu združijo v eno celoto. Problem, ki ga rešujemo, razbijemo na manjše, bolj obvladljive podprobleme. Na ta način lahko končno rešitev gradimo po korakih.

Ekstremno prototipiranje je metodologija razvojnega procesa, primerna predvsem za razvoj spletnih aplikacij [6]. Razvoj razbijemo na tri faze:

- statični prototipi

- funkcionalni uporabniški vmesnik
- zaledni sistem

Povdarek procesa je na drugi fazi, v kateri izdelamo popolnoma funckionalen uporabniški vmesnik brez delujočega zaledja.

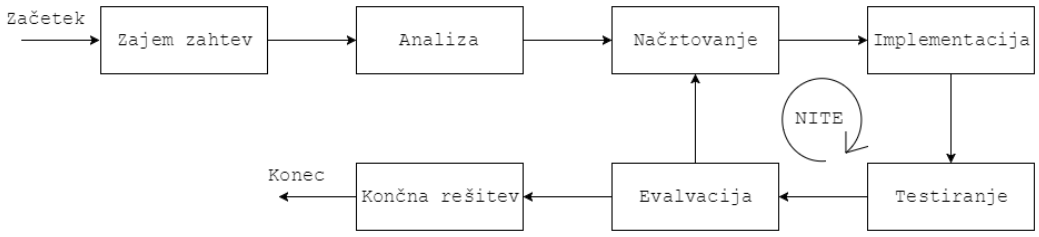
Gre za obliko iterativnega pristopa razvoja prototipov. Prototip konstantno nadgrajujemo in izboljšujemo, dokler ne pridemo do končne rešitve. Verzije prototipa se spreminajo hitro, na tedenski ali pa celo dnevni ravni. Razvojna ekipa mora biti dovolj majhna in agilna, da lahko operira s povratnimi informacijami uporabnikov in jih upošteva v novih verzijah prototipa.

Cilj prve faze je postavitev grobega ogrodja sistema in motivacija končnih uporabikov k sodelovanju pri razvoju [4, 7]. V tej fazi se ne ukvarjamo z uporabniško interakcijo, ampak želimo izdelati prototip, ki vsebuje vse glavne sklope aplikacije. Na ta način pridobimo okvirno sliko funkcionalnosti, ki jih bo aplikacija vsebovala. Sledita izdelava uporabniškega vmesnika in zalednega sistema. Razvoj teh faz lahko, vsaj v teoriji, poteka vzporedno. Proces iterativne izdelave uporabniškega vmesnika zagotavlja konstanten dotok povratnih informacij s strani uporabnikov. Te informacije so ključnega pomena pri zgodnjem odkrivanju in odpravljanju morebitnih težav.

2.2 Življenjski cikel razvojnih prototipov

Pri razvoju informacijskih sistemov poznamo več življenjskih ciklov razvoja (*angl. Software Developement Life Cycle - SDLC*). Življenjski cikel razvojnega procesa po metodologiji razvojnega prototipiranja lahko predstavimo z diagramom na sliki 2.1 [8]

Proces se začne z zajemom zahtev. V tej fazi skušamo pridobiti čim več informacij o aplikaciji, ki jo razvijamo. Sledi faza analize in načrtovanja. Pri modelu prototipiranja je ta faza, v primeru s fazo analize pri ostalih



Slika 2.1: Življenski cikel prototipiranja.

razvojnih modelih[1], krajša. Naslednja faza je implementacija prototipa, ki ji sledi testiranje. Testiranje se izvaja v tesni povezavi z uporabniki aplikacije. Glede na povratne informacije v fazi evalvacije prototipa preverimo ustreznost prototipa. Lahko se vrnemo v fazo načrtovanja ali pa zaključimo s prototipiranjem in prototip označimo za končno rešitev. Cikle *Načrtovanje* → *Implementacija* → *Testiranje* → *Evalvacija* (NITE) ponavljamo tako dolgo, da je prototip dovolj dober za nadgradnjo v končno rešitev.

2.3 Prednosti in nevarnosti

Glavna prednost prototipiranja je v optimizaciji komunikacije med razvijalci in uporabniki. Steve McConnell [9] je že leta 1996 po izvedeni raziskavi ugotovil, da so glavni razlogi za slab razvoj programske opreme[2] pomanjkanje komunikacije z uporabniki, nepopolne zahteve in spreminjanje zahtev v času razvoja. Razlogi za slab razvoj ostajajo enaki tudi danes. Vzrok omenjenih težav, pa je v preslabi komunikaciji med razvijalci in uporabniki. Poleg izboljšane komunikacije so prednosti prototipiranja tudi [7, 3, 2]:

- hitro vključevanje uporabnikov v razvojni proces
- izboljšan način za podajanje ter ugotavljanje zahtev novega sistema
- zgodnja detekcija morebitnih težav, ki zmanjšuje tveganje neuspeha

[1] Poleg prototipiranja poznamo zaporedni, iterativni, inkrementalni in kombinirani model.

[2] Predolg, predrag ali nepopoln razvoj.

- uporabnik je hitro v stiku z delujočo aplikacijo
- povečanje navdušenja za produkt tako razvijalcev kot uporabnikov
- preizkušanje različnih možnosti
- lahko se uporabi tudi kot orodje za učenje uporabnikov

Neustrezna uporaba prototipiranja lahko v razvojnem procesu povzroči negativne učinke. Posledice neustrezne uporabe so:

- nezadostna analiza, ki lahko pripelje do neustreznega končnega produkta
- prekomeren čas razvoja prototipa
- preveliko število verzij prototipa, ki lahko povzroči zmedo pri razvijalcih
- zamenjava prototipa za končni sistem

2.4 Kdaj uporabiti prototipiranje

Kot smo že ugotovili, je prototipiranje najbolj efektivno v tistih primerih, ko pričakujemo veliko interakcije z uporabniki. To potrjuje tudi sama narava prototipov, saj so v končni fazi namenjeni testiranju s strani uporabnikov. Uporaba prototipov je zelo priporočljiva tudi tedaj, ko uporabniki še ne vedo popolnoma, kaj naj bi nov sistem omogočal ali ko se zahteve pogosto spreminjajo [14]. Priporočljiva uporaba pa je tudi v naslednjih primerih, ko:

- imajo uporabniki težave s podajanjem zahtev sistema
- nov sistem spremeni osnove poslovnih operacij
- se pojavi potreba po prilagodivi uporabniškega vmesnika
- potreba po preverjanju različnih alternativnih rešitev
- uporabniki ne razumejo novih pridobitev
- nove poslovne funkcije so bolj jasne uporabnikom kot analitikom

Poglavje 3

Prototipiranje pri razvoju spletnih aplikacij

Pri razvoju spletnih aplikacij je običajno velik povdarek na interakciji z uporabniki. Pogosto se pojavijo problemi pri definiciji zahtev s strani uporabnikov, hkrati pa se zahteve tekom razvoja spreminjajo. Ker živimo v dobi interneta in spletnih tehnologij, se razvoj programske opreme na tak ali drugačen način seli na splet. Spekter uporabnikov spletnih aplikacij je zelo širok in zajema tako napredne uporabnike kot tudi laike. Slednji imajo pogosto težave z razumevanjem in uporabo tehnologije. Posledično imajo težave tudi pri uporabi spletnih aplikacij. Vse od naštetega so pri razvoju spletnih aplikacij močni argumenti za uporabo prototipiranja kot metodologije razvojnega procesa [10, 15].

3.1 Tipi prototipov

Zgodnje pridobivanje dobrih povratnih informacij uporabnikov je pri procesu prototipiranja ključnega pomena. Načinov izdelave prototipov je veliko, glede na dinamiko pa jih lahko razdelomo na dva tipa: statične in dinamične prototipe [6, 13].

3.1.1 Statični prototipi

Pri tem tipu gre za izdelavo prototipov glavnih segmentov aplikacije. Uporabnikom želimo prikazati, kako bo aplikacija izgledala v grobem. Ne spuščamo se v podrobnosti, ampak skušamo zajeti glavne značilnosti. Statični prototipi so lahko ekranske slike ali statične HTML strani, ki služijo kot žični diagrami (*angl. wireframes*) aplikacije. Slednji so za uporabnike bolj privlačni, saj zagotavljajo boljšo uporabniško izkušnjo in s tem boljše povratne informacije.

Ekranske slike so lahko izdelane z oblikovalskimi orodji kot sta Gimp[17] in Photoshop[18] ali z enim izmed spletnih orodij za izdelavo prototipnih skic aplikacij in spletnih strani. Ta orodja vsebujejo osnovne gradnike uporabniških vmesnikov in delujejo po principu povleci-in-spusti (*angl. drag&drop*), tako da je izdelava prototipov hitra in enostavna. Statične HTML strani lahko kreiramo z namenskimi orodji za izdelavo žičnih modelov (Axure[19] in ProtoShare[20]) ali pa z uporabo CSS frameworkov. V zadnjih letih se je pojavila pestra izbira orodij za izdelavo žičnih modelov spletnih aplikacij in spletnih strani. Njihova osnovna funkcija je izdelava prototipa z nekaj kliki in enostaven izvoz HTML kode. Izdelava prototipov s CSS frameworki zahteva več znanja in je ponavadi časovno potratnejša.

3.1.2 Dinamični prototipi

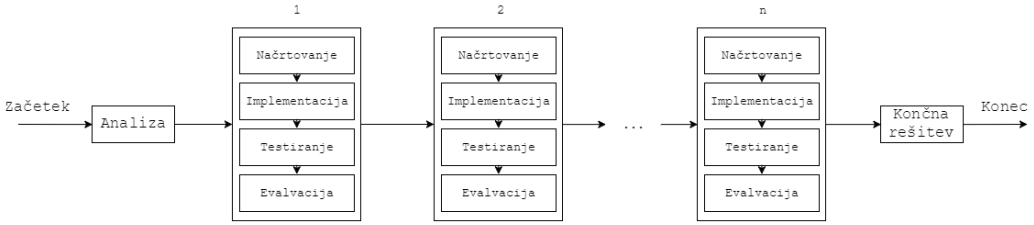
Pri razvojnem prototipiranju se običajno najprej izdela statične prototipe, kasneje pa se jih nadgradi v dinamične. Vse segmente aplikacije, prototipirane s statično metodo, predstavimo bolj detajlno. Dinamični prototipi vsebujejo že čisto prave elemente grafičnega vmesnika z implementirano uporabniško interakcijo. Tak prototip ni samo izdelan grafični vmesnik, temveč tudi sistem za komunikacijo z zaledjem in logiko za prikazovanje podatkov [13]. Tudi, če zaledni sistem ni razvit, daje dinamični prototip občutek popolnoma delujoče aplikacije.

Izdelava dinamičnih prototipov je v primerjavi s statičnimi veliko bolj

kompleksna in zato časovno potratnejša.

3.2 Pomanjkljivosti v začetnih fazah prototipiranja

V začetnih verzijah prototipov želimo uporabnikom prikazati grobo strukturo sistema, glede na informacije, ki smo jih pridobili v fazi analize. Zaradi enostavnejše in cenejše izdelave se razvijalci takrat pogosto poslužujejo statičnih prototipov. Prototipe izdelajo z orodji za generiranje statičnih HTML strani ali pa so prototipi celo v obliki skic. Do bolj funkcionalne verzije prototipa pridemo šele po nekaj verzijah prototipa[1]. Zaradi preslabe analize ali preveč enostavnega prototipa je lahko ta proces dolgotrajen. Velikokrat se zgodi, da uporabniki ugotovijo, kaj res želijo šele ob interakciji s funkcionalnim (dinamičnim) prototipom. Kot posledica se proces prototipiranja podaljša, kar močno vpliva na začrtane časovne in stroškovne plane razvoja aplikacije.



Slika 3.1: Razširjen življenjski cikel prototipiranja. Po n ciklih NITE (načrtovanje $\rightarrow$ implementacija $\rightarrow$ testiranje $\rightarrow$ evalvacija) je prototip dovolj dober za nadgradnjo v končno rešitev.

Poleg preskromnih prototipov v začetnih fazah prototipiranja lahko čas razvoja podaljšujeo tudi slaba odzivnost ali nejasne povratne informacije uporabnikov, ki prototipe testirajo. Ko uporabniki ugotovijo pomankljivost prototipa, se velikokrat ne zavedajo, da je odgovor *"Ne deluje"* veliko premalo

[1]Razvoj prototipa iz začetne verzije v končno rešitev ponazarja slika 3.1.

za sanacijo.

3.3 Predlog za boljše prototipiranje

Zelo pomembni lastnosti prototipov sta prepričljivost za uporabnike in enostavnost za izdelavo. Uporabniki, ki se srečajo z neprepričljivimi prototipi, ne morejo dati dovolj dobrih povratnih informacij, saj jim prototip ne zagotavlja dobre izkušnje s končnim izdelkom. Po drugi strani se lahko stroški za izdelavo prototipov močno povečajo, če se poveča kompleksnost in s tem čas izdelave prototipa. Omenjena izziva bi lahko premagali z mehanizmom, ki bi omogočal enostavno nadgrajevanje cenejših statičnih prototipov v dinamične.

Poglavje 4

Mehanizem za hitrejšo izdelavo boljših prototipov

V prejšnjem poglavju smo predstavili pozitivne efekte prototipiranja in nevarnosti, ki se lahko pojavijo. Da bi se izognili dodatnim stroškom pri izdelavi boljših prototipov in s tem optimizirali proces izdelave, predlagam mehanizem za hitro nadgrajevanje statičnih prototipov. Glavna izziva, ki jih mehanizem rešuje, sta zmožnost hitrega nadgrajevanja statičnih prototipov v dinamične ter enostavnost za uporabo, kar omogoča hitrejšo implementacijo. Predlagani mehanizem sestavljajo štirje večji segmenti:

- generiranje naključnih podatkov
- simulacija zalednega sistema
- enostavna populacija podatkov v statični prototip
- mehanizem za sporočanje napak razvijalcem

Mehanizem je implementiran kot skupek JavaScript knjižnic in modulov [16]. Rešitev vsebuje že razvit jQuery[21] modul Mockajax[23] za simulacijo zalednega sistema in knjižnico Chance[22] za generiranje naključnih podatkov. Moj prispevek je modul pType, ki skrbi za populacijo podatkov v HTML

ter krmili omenjeni rešitvi. Poleg modula pType sem prilagodil knjižnico Chance za slovenski jezik in jo razširil z dodatnimi funkcionalnostmi.

4.1 Uporabljene tehnologije

Svojo rešitev sem implementiral s tehnologijo JavaScript, ki se v sodobnem razvoju spletnih aplikacij uporablja za bogatenje uporabniške interakcije. Sodobne spletne aplikacije so grajene kot enostranske aplikacije (*angl. single-page applications*), ki za komunikacijo z zalednim sistemom uporabljajo AJAX poizvedbe. Zaradi njene razširjenosti sem za izkoriščanje te tehnologije uporabil odprtokodno knjižnico jQuery.

Vseh modulov in knjižnic, ki sem jih uporabil v svoji rešitvi, nisem implementiral sam. jQuery modul Mockajax in knjižnico Chance bom podrobneje predstavil v nadaljevanju poglavja. Gre za odprtokodni rešitvi, ki sem ju uporabil v svoji rešitvi. Predstavitev mojega modula pa sledi v poglavju 4.2.

4.1.1 MockAjax

Pri izdelovanju dinamičnih prototipov je funkcionalnost grafičnega vmesnika velikokrat pogojena z delujočim zalednim sistemom. To pomeni, da njun razvoj ne more biti popolnoma vzporeden. Če izdelujemo spletno aplikacijo, ki za komunikacijo med grafičnim vmesnikom in zalednim sistemom uporablja ajax poizvedbe, lahko ta problem premostimo z uporabo Mockajax modula za odprtokodno JavaScript knjižnjico jQuery. Mockajax omogoča razvijalcem grafičnega vmesnika simuliranje ajax poizvedb, kar pomeni, da za izdelavo popolnoma delujočega grafičnega vmesnika, zalednega sistema ne potrebujemo. Seveda bo delovanje takega grafičnega vmesnika odvisno od kreativnosti simuliranih odgovorov poizvedb in nikakor ne pomeni delujoče aplikacije.

Delovanje

MockAjax razširi osnovno delovanje jQuery metode *$.ajax()*. Modul prepozna v naprej označene poizvedbe na zaledni sistem, jih prestreže in ustvari simuliran XMLHttpRequest objekt pred dejanskim izvajanjem *jQuery.ajax()* metode. Če poizvedba ni označena za simulacijo, se izvede privzeta *$.ajax()* metoda za proženje ajax poizvedbe. Prednost takega pristopa je v tem, da s simuliranjem XMLHttpRequest objekta ne prepišemo privzetega delovanja jQuery knjižnjice, zato lahko implementiramo poljuben potek ajax komunikacije z vsemi privzetimi funkcionalnostmi.

Da lahko poizvedbo prestrežemo, jo moramo najprej označiti in ji določiti odgovor. Odgovor je lahko poljubnega tipa in velikosti, tako kot odgovor pravega zalednega sistema. Simulirani poizvedbi lahko nastavimo tudi časovno latenco, s katero je simulacija še bolj pristna. V nadaljevanju sledi podrobnejša predstavitev delovanja modula Mockjax.

Označevanje poizvedb za simulacijo

Poizvedbe, ki jih želimo simulirati, moramo predhodno označiti. To naredimo tako, da z metodo *$.mockajax()* debifiramo URL naslov poizvedbe. Najpreprostejši način je definicija celotnega naslova:

```
$.mockjax({
  url: "/uporabniki/seznam"
});
```

Primer 4.1: Poizvedbo smo označili s fiksnim URL naslovom **/uporabniki/seznam**.

Za večjo fleksibilnost lahko URL naslovi vsebujejo znak *, ki označuje poljuben znakovni niz.

```
$.mockjax({
  url: "/uporabniki/*"
});
```

Primer 4.2: Simulirane bodo vse poizvedbe katerih URL naslovi se bodo začeli z **/uporabniki/**.

URL naslove lahko definiramo tudi kot regularne izraze.

```
$.mockjax({
  url: /^\/uporabniki\/(seznam|list)$/
});
```

Primer 4.3: Simulirani bosta poizvedbi z URL naslovoma **/uporabniki/seznam** in **/uporabniki/list**.

Poizvedbo lahko označimo tudi kot kombinacijo URL naslova in podatkov.

```
$.mockjax({
  url: "/uporabniki/seznam",
  data: { group: "admin" }
});
```

Primer 4.4: Poizvedba bo simulirana samo, če bo poleg posredovan še podatek **group** z vrednostjo **admin**.

Določanje odgovora

Drugi korak pri simulaciji poizvedb z modulom Mockajax je definicija odgovora, ki ga bo poizvedba vrnila. Odgovor lahko podamo kot *responseText* ali *responseXML*. Obe opciji simulirata odgovor XMLHttpRequest objekta ob običajni izvedbi ajax poizvedbe. Odgovor lahko definiramo na tri načine:

- v naprej pripravljen odgovor
- datoteka
- klic funkcije

Primer v naprej pripravljenega odgovora je JSON objekt

```
$.mockjax({
  url: "/uporabniki/prijavljen",
  responseText: {
    "ime": "Luka",
    "priimek": "Andrejak"
  }
});
```

Primer 4.5: V naprej pripravljen JSON odgovor.

Kadar je kot odgovor definirana datoteka, bo Mocjakax modul prestregel poizvedbo in kot odgovor vrnil datoteko. Tehnika je uporabna takrat, ko nam stranka zagotovi ustrezne testne podatke ali pa si jih pripravimo sami.

```
$.mockjax({
  url: "/uporabniki/seznam",
  proxy: "/podatki/uporabniki.json"
});
```

Primer 4.6: Odgovor poizvedbe bo vsebina datoteke **/podatki/uporabniki.json**.

Odgovor poizvedbe lahko simuliramo tudi s funkcijo, tako da definiramo parameter response ki dinamično generira responseText ali responseXML.

```
$.mockjax({
  url: "/preveriEmail",
  response: function(settings) {
    var rgx = /[A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2,4}/igm;
    this.responseText = (rgx.test(settings.data.email) ?
      "Email OK" : "Email ni veljaven!");
  }
});
```

Primer 4.7: Odgovor poizvedbe je funkcija, ki preveri pravilnost elektronskega naslova, ki je bil poslan s poizvedbo.

Časovna zakasnitev

Pri simuliranju zalednega sistema je pomemben faktor tudi čas, ki bi ga pravi zaledni sistem potreboval za generiranje odgovora. Časovno zakasnitev lahko definiramo kot fiksno ali naključno v izbranem intervalu.

```
$.mockjax({
  url: "/zahtevnaOpeacija",
  responseTime: 2000,
  responseText: { odgovor: "Danes je lep dan" }
});

$.mockjax({
  url: "/seBoljZahtevnaOpeacija",
  responseTime: [1000,3000],
  responseText: { odgovor: "Danes je lep dan" }
});
```

Primer 4.8: V prvem primeru se bo poizvedba izvedla v dveh sekundah, v drugem pa naključno med eno in tremi sekundami.

4.1.2 Chance

Pri dinamičnih prototipih je, poleg uporabniške interakcije, velik povdarek tudi na podatkih, prikazanih uporabniku. V primeru, da od naročnika nismo pridobili kvalitetnih testnih podatkov, si lahko pomagamo z generiranjem naključnih. Pri generiranju naključnih podatkov je pomembno, da imamo na voljo čim več tipov podatkov in da smo sposobni generirati smiselne podatke. Tabela, ki predstavlja seznam uporabnikov, je veliko bolj pristna, če vsebuje prava imena in priimke, kot pa če bi vsebovala naključno generirane znakovne nize.

Chance JS je odprtokodna JavaScript knjižnica za generiranje naključnih podatkov. Je enostavna za uporabo in ponuja dovolj pester nabor podatkovnih tipov za generiranje tudi zelo kompleksnih entitet z medsebojno odvisnimi podatki.

V osnovi knjižnica nima podpore za generiranje podatkov v slovenskem jeziku, zato sem jo nadgradil. V knjižnico sem dodal šifrante slovenskih imen, priimkov, imen krajev, poštnih številk ipd. Dodal sem tudi zmožnost generiranja EMŠO in davčne številke.

Podatkovni tipi

Poleg osnovnih podatkovnih tipov (boolean, character, floating, integer, niz) lahko s Chance generiramo kompleksne tipe, ki so sestavljeni iz več osnovnih ali kompleksnih tipov. Primer kompleksnega tipa je oseba. Oseba je sestavljena iz več osnovnih tipov: ime, priimek, starost, spol, rojstni datum, davčna številka ipd. Podatek generiramo s klicem ustrezne metode objekta *chance*:

```
chance.name()
```

Primer 4.9: Generiranje naključnega osebnega imena.

Osnovni tipi

Knjižnica omogoče generiranje naslednjih osnovnih tipov:

- boolean:
  Privzeta verjetnost rezultata *true* je 50 % in *false* prav tako 50 %. Verjetnost rezultata *true* lahko spremenimo s podajanjem parametra *likelihood*

  ```
  chance.bool({likelihood: 30})
  ```

  Verjetnost rezultata *true* je v tem primeru 30 %, *false* pa 70 %.

- character:
  Metoda vrne naključen znak. Množico potencialnih znakov lahko omejimo s parametrom *pool*.

```
chance.character()
chance.character({pool: 'abcde'})
chance.character({alpha: true})
chance.character({symbols: true})
chance.character({casing: 'lower'})

=> 'd'
```

- integer:
  Generiranje celih števil lahko omejimo na interval.

```
chance.integer()
chance.integer({min: -20, max: 20})

=> 18
```

- floating:
  Decimalnim številom lahko omejimo število mest za decimalno vejico. Generiranje lahko omejimo na interval.

```
chance.floating()
chance.floating({fixed: 7})
chance.floating({min: 0, max: 100})

=> 33,4572947
```

- string:
  Generiranje znakovnih nizov lahko omejimo po dolžini in z množico možnih znakov.

```
chance.string()
chance.string({length: 5})
chance.string({pool: 'abcde'})

=> "hskit"
```

Oseba

- spol:

```
chance.gender()

=> 'male'
```

- ime:

```
chance.name()
chance.name({prefix: true})

=> 'dr. Miha'
```

- priimek:

```
chance.last()

=> 'Novak'
```

- datum rojstva:
  Generiranje datuma rojstva lahko omejimo s parametrom *type*. Vrednost parametra je starostno obdobje osebe. Možne vrednosti: child, teen, adult, senior.

```
chance.birthday()
chance.birthday({type: 'child'})

=> 2011-11-05
```

- davčna številka:
  Generiranje slovenske davčne številke. Številka se generira z upoštevanjem pravila za preverjanje ustreznosti davčne številke.

```
chance.siDS()

=> 78724230

//funkcija za generiranje
function siDS () {
  var base = chance.integer({ min: 1000000,
  max: 9999999 }).toString(),
  baseArray = base.split("").reverse(),
  sum = 0;

  for(var i = 0; i < baseArray.length; i++) {
    sum += Number(baseArray[i]) * (i+2);
  }

  return baseArray.reverse().join("") + (sum % 11);
}
```

- EMŠO:
  Številka se generira glede na datum rojstva ter spol osebe. Generirana številka upošteva zakonitosti EMŠO številk za prebivalce Slovenije.

```
chance.EMSO({ birthday: '1996-05-03', gender: 'male' })

=> 0305996500256

//funkcija za generiranje
function EMSO (birthday, gender) {
  var factor_map = [7, 6, 5, 4, 3,
    2, 7, 6, 5, 4, 3, 2],
  birth = birthay.split('-'),
  nth = 0,
  result = "",
  sum = 0,
```

```
    controll = 10;

    if(gender == 'male') {
      nth = chance.pad(chance.integer({
        min: 0, max: 499
        }),
      3);
    } else {
      nth = chance.pad(chance.integer({
        min: 500, max: 999
        }),
      3);
    }

    while(controll == 10) {
      result = birth[2] + birth[1] + substr(birth[0],1) +
          '50' + nth.toString();

      var factors = result.split("");

      for(var i = 0; i < factors.length; i++) {
        sum += factors[i] * factor_map[i];
      }

      controll = 11 - sum % 11;
      if(controll = 11) {
        controll = 0;
      }

      nth++;
    }

    return result + control;
  }
```

- Telefonska številka:

  Prvi dve številki sta iz seznama 01, 02, 03, ... 07, ostalih sedem pa je

naključnih.

```
chance.phone()

=> 01 230 40 02
```

- Mobilna številka:
  Prve tri številke so iz seznama 031, 041, 051, 040, ostalih šest pa je določenih naključno.

```
chance.mobile()

=> 031 123 123
```

Splet

- avatar:
  Naključna profilna slika z uporabo Gravatar API-ja.

```
chance.avatar()
chance.avatar({fileExtension: 'jpg'})
chance.avatar({email: 'mail@gmail.com'})

=> '//www.gravatar.com/avatar/76697
   df5874c854e3cc8fde1200b4298.jpg'
```

- email:

```
chance.email()
chance.email({domain: "example.com"})

=> someone@example.com
```

- ip:

```
chance.ip()

=> '153.208.102.234'
```

Lokacija

- naslov:

```
chance.address();

=> 'Stritarjeva cesta 10'
```

- ulica:

```
chance.street();

=> 'Slovenska cesta'
```

- mesto:

```
chance.city();

=> 'Ljubljana'
```

- pošta:

```
chance.post();

=> 'Ljubljana'
```

- poštna številka:

```
chance.zip();

=> '1000'
```

4.2 Modul pType

Kot predlog mehanizma za optimizacijo izdelave dinamičnega prototipa sem implementiral JavaScript modul pType. Glavni cilj modula je razvijalcem omogočiti enostavno nadgradnjo statičnega prototipa v dinamičnega. Z uporabo modula je možno nadgraditi kakršen koli statičen HTML prototip.

Lahko je generiran z namenskimi orodji (npr.: Axure in ProtoShare) ali pa je zgrajen z uporabo CSS frameworkov (npr.: Bootstrap, Foundation, Semantic). Modul omogoča avtomatičen vnos podatkov v HTML in preverjanje ustreznosti podatkov brez potrebe po programiranju. HTML elemente je potrebno razširiti z dodatnimi podatkovnimi atributi (*angl. data attributes*), s katerimi določimo želeno obnašanje, za vse ostalo poskrbi modul sam.

Modul je uporaben tudi v fazi potrjevanja prototipa s strani uporabnikov. Ko uporabnik pride do situacije, pri kateri je prišlo do napačnega ali pomanjkljivega delovanja, lahko na enostaven način naredi 'posnetek' stanja in ga doda na seznam pomanjkljivosti.

4.3 Generiranje naključnih podatkov

Glavni izziv pri generiranju naključnih podatkov je v zmožnosti generiranja čim bolj realnih podatkov. Hitro lahko ugotovimo, da tabela, ki prikazuje seznam uporabnikov z naključnimi nizi, ki predstavljajo imena, priimke in email naslove, ni dovolj dober prototip. Modul pType za generiranje podatkov uporablja knjižnico Chance, ki zna poleg osnovnih tipov generirati naključne entitete kot so osebe, naslovi, mobilne naprave, spletni profili oseb ipd. Da poenostavimo uporabo knjižnice, moramo HTML elemente, ki jih želimo napolniti z naključnimi podatki, opremiti z dodatnimi podatkovnimi atributi. Ti atributi modulu zagotovijo potrebne podatke za ustrezne klice Chance knjižnice.

```
<input type="text" data-pt-randomize="true"
  data-pt-type="integer" />
```

Primer 4.10: Generiranje naključnega celega števila v vnosnem polju.

Z atributom *data-pt-randomize* določimo element, ki bo vseboval naključne podatke. V primeru 4.10 želimo v vnosno polje vnesti naključno celo število. Tip podatka definiramo z atributom *data-pt-type*. Seznam podprtih tipov je podan v poglavju 4.1.2. Če želimo biti bolj natančni in generiran podatek podrobneje definirati, mu dodamo atribut *data-pt-options*. Vrednost

atributa je poenostavljen zapis JSON objekta, s katerim se kliče izbrano Chance metodo za generiranje podatka.

```
<input type="text"
  data-pt-randomize="true"
  data-pt-type="integer"
  data-pt-options="{min: 100, max: 1000}" />
```

Primer 4.11: Generiranja naključnega integerja na intervalu *[100, 1000]*.

Ko želimo prikazati eno izmed vrednosti v končni množici, uporabimo atribut *data-pt-choices*. Vrednost atributa je z znakom *","* ločen seznam vseh možnih vrednosti:

```
<input type="text" data-pt-randomize="true"
  data-pt-choices="modra,zelena,bela,rumena" />
```

Primer 4.12: Končna množica vrednosti.

4.3.1 Generiranje podatkov za sestavljene HTML elemente

Ko generiramo naključne podatke za kompleksnejše HTML elemente ali sestavljene strukture, moramo atribut *data-pt-randomize* določiti hierarhično najvišjemu elementu v strukturi. Za primer lahko vzamemo tabelo:

```
<table data-pt-randomize="100">
  <thead>
    <tr>
      <th data-pt-type="first">Ime</th>
      <th data-pt-type="last">Priimek</th>
      <th data-pt-type="gender">Spol</th>
      <th data-pt-type="phone">Telefon</th>
    </tr>
  </thead>

  <tbody>
  </tbody>
</table>
```

Primer 4.13: HTML tabela s podatkovnimi atributi za generiranje naključnih podatkov.

Vrednost atributa *data-pt-randomize* je v tem primeru število vrstic. Generiranje podatkov v primeru 4.13 se izvede nekako takole:

```
var data = [];
  for(var i = 0; i < 100; i++) {
    data.push([
    chance.first(),
    chance.last(),
    chance.gender(),
    chance.phone()
  ]);
}
```

Primer 4.14: Spremenljivka *data* po izvedbi *for* zanke vsebuje 100 naključnih podatkov o uporabnikih.

Pozoren bralec bo opazil, da ima primer 4.14 pomanjkljivost. Generiranje podatkov posamezne osebe ne upošteva medsebojno odvisnih podatkov. Klica metod *chance.first()* in *chance.gender()* lahko vrneta nasprotujoča si podatka. Osebi z imenom Janez bi lahko določili ženski spol. Modul pType take situacije avtomatično prepozna in zagotovi usklajenost podatkov. Podobno se modul obnaša pri generiranju email naslovov in EMŠO številk. Pravilnejši prikaz generiranja podatkov za primer 4.13:

```
data = [];
for(var i = 0; i < n; i++) {
  var spol = chance.gender();
  data.push([
    chance.first({ gender: spol }),
    spol
  ]);
}
```

Primer 4.15: Generiranje imena osebe glede na spol.

4.4 Prikaz podatkov, pridobljenih s poizvedbami

Podatke, ki jih želimo prikazati v uporabniškem vmesniku, vedno pridobimo v neki v naprej določeni strukturi. Navadno gre za XML dokument ali pa JSON objekt. Zaradi enostavnosti se bomo osredotočili na JSON obliko zapisa podatkov. Enostaven primer JSON objekta je seznam uporabnikov:

```
[{
  "firstName": "Luka",
  "lastName": "Andrejak",
  "email": "luka.andrejak@gmail.com",
  "status": 1
},{
  "firstName": "Janez",
  "email": "j.novak@gmail.com",
  "lastName": "Novak",
  "status": 1
},{
  "email": "doe.john@yahoo.com",
  "firstName": "John",
  "lastName": "Doe",
  "status": 0
}]
```

Primer 4.16: JSON objekt treh uporabnikov.

Če hočemo podatke iz take strukture pravilno prikazati, moramo vedeti, v katerem HTML elementu naj določeni podatek prikažemo. HTML elemente enolično označimo z atributom *data-pt-name*. Za tabelaričen prikaz podatkov iz primera 4.16 bi pripravili HTML tabelo s štirimi stolpci:

```
<table class="users">
  <thead>
    <th data-pt-name="firstName">Ime</th>
    <th data-pt-name="lastName">Priimek</th>
    <th data-pt-name="email">Email</th>
    <th data-pt-name="status"
      data-pt-choices="1:aktiven,0:neaktiven">Status</th>
  </thead>

  <tbody>
  </tbody>
</table>
```

Primer 4.17: HTML tabela z označenimi stolpci.

Imena stolpcev tabele se ujemajo s podatkovno strukturo JSON objekta, zato lahko vsako vrstico tabele ustrezno napolnimo s pravimi podatki. Posebnost je stolpec za prikaz statusa. V podatkovni strukturi je vrednost statusa lahko 0 ali 1. Iz vidika uporabniške izkušnje je predstavitev statusa uporabnika veliko boljša, če v polju piše *aktiven* ali *neaktiven*, kot 1 ali 0. Ta stolpec dodatno opremimo z atributom *data-pt-choices*, ki definira preslikavo podatka.

Podatke prikažemo s klicem metode *pType.loadData()*:

```
$.ajax({
  url: "/uporabniki/seznam",
  success: function (data) {
    pType.loadData($('table.users'), data);
  }
});
```

Primer 4.18: Nalaganje podatkov, pridobljenih s poizvedbo, v tabelo uporabnikov.

ali pa HTML element, ki ga želimo napolniti s podatki, opremimo z atributom *data-pt-source*. Vrednost atributa je URL naslov poizvedbe, ki bo priskrbela želene podatke. V tem primeru se bo metoda *pType.loadData()*

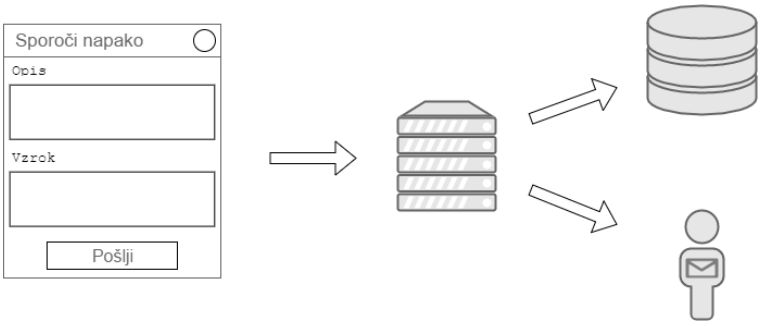
izvedla avtomatično.

```
<table data-pt-source="/uporabniki/seznam">
  ...
</table>
```

Primer 4.19: HTML tabela z definiranim izvorom podatkov.

4.5 Mehanizem za sporočanje napak

Ko je prototip izdelan, ga je potrebno s strani uporabnikov pretestirati in ugotovitve posredovati nazaj razvijalcem. V idealnem svetu bi se testiranje izvajalo tako, da bi razvijalec sedel poleg uporabnika, ko bi prototip testiral in si beležil ugotovitve. V praksi tak način testiranja ni vedno mogoč - še posebej takrat, ko je uporabnikov veliko in niso vedno dosegljivi. Uporabniki imajo velikokrat težavo opisati problem, na katerega so naleteli pri testiranju. Poleg tega ne vedo, kaj vse je lahko vzrok težave, zato njihove povratne informacije pogosto ne vsebujejo dovolj informacij za identifikacijo napake. Za premostitev omenjenih problemov lahko uporabniki povratne informacije sporočajo preko vmesnika, ki ga zagotavlja modul pType. Gre za enostaven obrazec preko, katerega uporabnik sporoči težavo in njen vzrok. Ob posredovanju ugotovitev uporabnika se poleg vpisanih podatkov pošljejo še dodatne informacije o stanju aplikacije ob pojavitvi napake.



Slika 4.1: Informacije o napaki se pošljejo na razvojni strežnik. Podatki se shranijo v podatkovno bazo ter posredujejo razvijalcu na email.

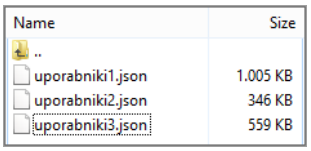
Koraki delovanja:

- Na uporabnikovo zahtevo se prikaže obrazec za sporočanje napak.
- Uporabnik vnese opis težave.
- Uporabnikovim informacijam se dodajo še podatek o operacijskem sistemu, vrsta in verzija brskalnika, točen URL naslov ter vsi podatki, ki so bili prikazani v grafičnem vmesniku.
- Izvede se POST poizvedba na razvojni strežnik.
- Podatki o napaki se shranijo v podatkovno bazo za nadaljnjo obdelavo.
- O napaki se obvesti tudi razvojno ekipo preko elektronske pošte.

4.6 Primeri uporabe

4.6.1 Simulacija zalednega sistema

Pri nadgrajevanju statičnih prototipov v dinamične ponavadi še nimamo izdelanega zalednega sistema. Denimo, da si, za namen testiranja prototipa, pripravimo nekaj testnih podatkov v obliki JSON datotek.



Slika 4.2: Datoteke s testnimi podatki.

Za zajem podatkov simuliramo klic poizvedbe na zaledni sistem z uporabo modula Mockajax:

```
$.mockjax({
  url: "/vrni/uporabnike/*",
  proxy: "/uporabniki1.json"
});
```

Primer 4.20: Odgovor poizvedbe za statistiko oglasa, bo vsebina datoteke **uporabniki1.json**.

Ker je prikaz podatkov v uporabniškem vmesniku odvisen od količine zajetih podatkov, smo pripravili več datotek z različnimi količinami podatkov. Večja kot je datoteka, več podatkov vsebuje. Simulacijo zalednega sistema lahko nadgradimo tako, da datoteko s podatki izberemo naključno:

```
$.mockjax({
  url: "/vrni/uporabnike/*",
  proxy: chance.pick(["uporabniki1.json",
    "uporabniki2.json","uporabniki3.json"])
});
```

Primer 4.21: Naključna izbira datoteke za odgovor poizvedbe.

Simulacijo naredimo bolj realno tako, da ji določimo časovno zakasnitev. Ker so datoteke različnih velikosti, morajo biti tudi zakasnitve različne. Z dodajanjem ustrezne zakasnitve primer 4.21 nadgradimo:

```
var mockData = chance.pickone[
  { file: "uporabniki.json",  time: 1500 },
  { file: "uporabniki2.json", time: 200 },
  { file: "uporabniki3.json", time: 800 }
];

$.mockjax({
  url: "/vrni/uporabnike/*",
  proxy: mockData.file,
  responseTime: mockData.time
});
```

Primer 4.22: Odgovor simulirane poizvedbe bo ustrezno zakasnjen.

V praksi se včasih zgodi, da iz kakšnega razloga, zaledni sistem ne deluje ali ta ni dostopen. V takih primerih bi se poizvedba zaključila kot neobdelana, s statusom 500. Pri simulaciji zalednega sistema lahko obravnavamo tudi take situacije:

```
$.mockjax({
  url: "/vrni/uporabnike/*",
  proxy: mockData.file,
  responseTime: mockData.time,
  status: chance.bool({likelyhood: 5}) ? 500 : 200
});
```

Primer 4.23: Poizvedba se bo z verjetnostjo 5 % zaključila s statusom 500.

4.6.2 Vstavljanje podatkov v HTML

Struktura JSON zapisa je v vseh datotekah (slika 4.2) enaka.

```
[{
  "ime": "Luka",
  "priimek": "Andrejak",
  "status": 1,
  "smer": "racunalniski sistemi",
  "starost": 29
}, ...]
```

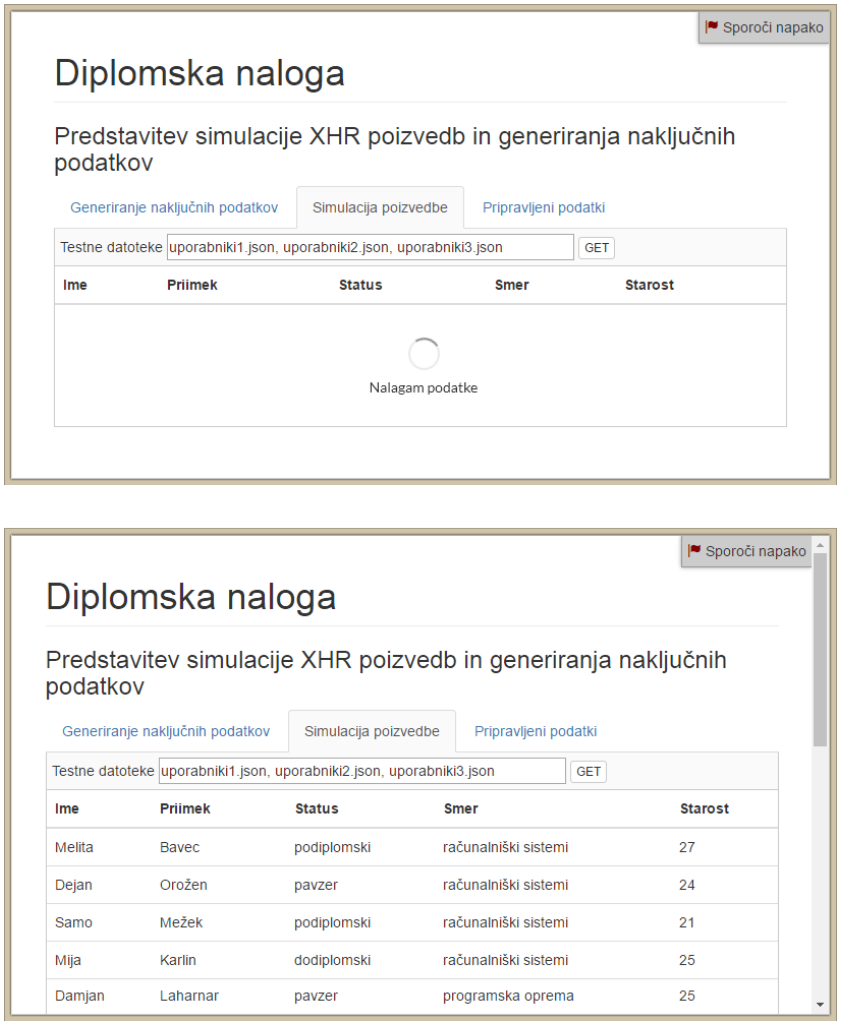
Primer 4.24: Primer zapisa datoteke *uporabniki1.json*

Podatke pridobimo z AJAX poizvedbo, za prikaz podatkov pa pripravimo HTML tabelo z razširjenimi podatkovnimi atributi:

```
<table id="uporabniki">
  <thead>
    <tr>
      <th data-pt-name="ime">Ime</th>
      <th data-pt-name="priimek">Priimek</th>
      <th data-pt-name="status"
      data-pt-choices="0:pavzer;1:dodiplomski;2:podiplomski">
          Status</th>
      <th data-pt-name="smer">Smer</th>
      <th data-pt-name="starost">Starost</th>
    </tr>
  </thead>
</table>

$.ajax({
  url: "/vrni/uporabnike/",
  success: function (data) {
    pType.loadData(
      $('#users'),
      data
    );
  }
});
```

Primer 4.25: HTML tabela z razširjenimi podatkovnimi atributi ter klic AJAX poizvedbe za pridobitev podatkov. Pri uspešnem odgovoru poizvedbe, podatke prikažemo v tabeli s klicem metode *pType.loadData()* z ustreznimi parametri.



Slika 4.3: Napolnjena HTML tabela s testnimi podatki primera 4.25.

4.6.3 Naključni podatki

Za testiranje prototipa, lahko testne podatke generiramo naključno. Tabelo uporabnikov s primera 4.25 opremimo s podatkovnimi atributi, ki določajo tip podatka, ki ga želimo generirati:

```
<table id="uporabniki" data-pt-randomize="100">
  <thead>
    <tr>
      <th data-pt-type="first">Ime</th>
      <th data-pt-type="last">Priimek</th>
      <th data-pt-choices="pavzer,dodiplomski,podiplomski">
         Status</th>
      <th data-pt-choices="racunalniski sistemi,informatika,
         programska oprema">Smer</th>
      <th data-pt-name="integer"
        data-pt-options="{min: 18, max: 30}">Starost</th>
    </tr>
  </thead>
</table>
```

Primer 4.26: HTML tabela z razširjenimi podatkovnimi atributi za generiranje naključnih podatkov.

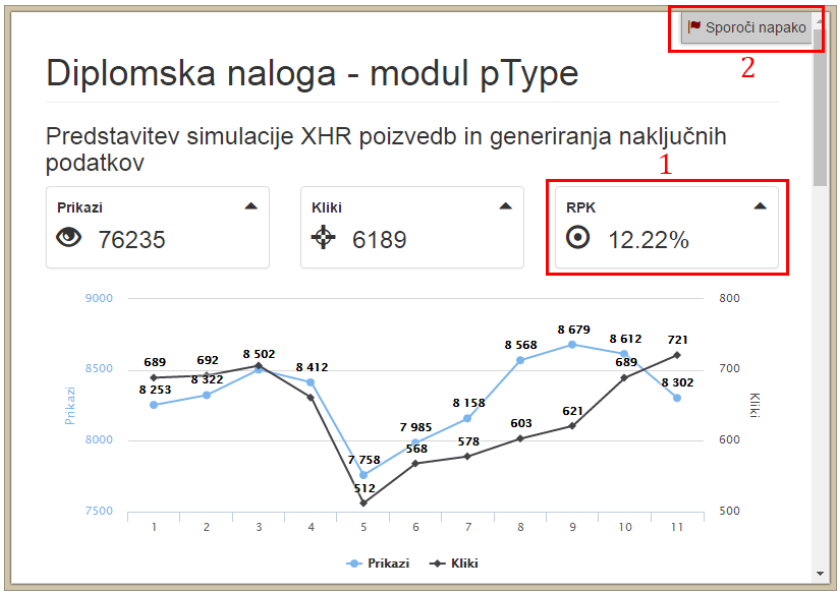
Slika 4.4: Napolnjena HTML tabela z naključno generiranimi testnimi podatki primera 4.26.

4.6.4 Sporočanje napak

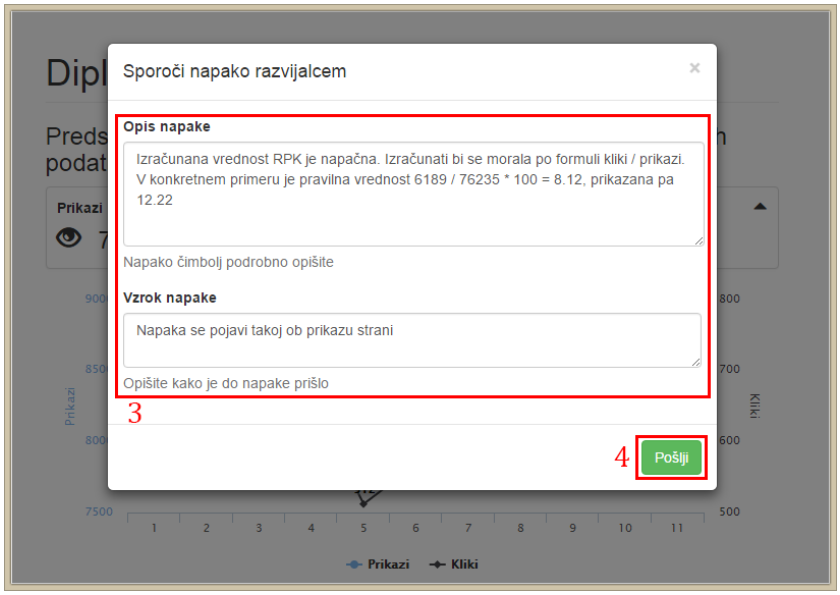
Ko med testiranjem prototipa uporabniki naletijo na nepravilnosti, lahko o tem razvijalce obvestijo s pošiljanjem podatkov o nastali situaciji. Uporaba modula pType omogoča uporabnikom enostavno pošiljanje podatkov preko obrazca, katerega lahko prikličejo kadarkoli med procesom testiranja.

Zmožnost sporočanja napak razvijalcem takoj, ko se te pojavijo, je zelo priročna funkcionalnost. Na ta način zmanjšamo verjetnost, da bi uporabniki pozabili sporočiti kakšno nepravilnost, hkrati pa jim sporočanje olajšamo.

Poleg informacij, ki jih uporabniki vnesejo v obrazec, se razvijalcem samodejno pošljejo še dodatni podatki, katere bi lahko uporabniki pozabili sporočiti. To so podatki o uporabnikovem brskalniku, operacijskem sistemu ter napravi (namizni računalnik, mobilni telefon, računalniška tablica).



Slika 4.5: Uporabnik zazna napako (1) in s klikom na gumb (2) prikliče obrazec za pošiljanje sporočil.



Slika 4.6: Uporabnik vnese podatke o napaki (3) in jih s potrditvijo obrazca (4) pošlje razvojni ekipi.

Poglavje 5

Testiranje rešitve in ugotovitve

V podjetju, s katerim poslovno sodelujem, sem imel možnost testirati rešitev za boljše prototipiranje na realnem projektu. V nadaljevanju bom opisal faze prototipiranja pri razvoju dveh projektov. Projekt A je bil razvit s klasično metodologijo razvojnega prototipiranja, pri projektu B pa smo uporabili tehnike naprednejšega prototipiranja, ki jih implementira modul pType. Primerjal bom časovno zahtevnost in število iteracij prototipiranja.

Projekt A: Spletna aplikacija za spremljanje in optimizacijo oglaševalskih kampanj. Aplikacija zajema statistične podatke o aktivnostih oglasov na različnih oglaševalskih kanalih in jih prikazuje uporabniku. Sestavlja jo več interaktivnih uporabniških vmesnikov, s pomočjo katerih se uporabniki lažje odločajo za upravljanje z oglasi ter vmesnik za kreiranje poročil. Osredotočili se bomo na segment za pregled statistik in kreiranje poročil.

Projekt B: Spletna aplikacija za izvajanje farmacevtskih kliničnih raziskav. Gre za aplikacijo, s katero farmacevtsko podjetje zbira podatke o vplivih potencialnega zdravila na bolnike. Podatke v sistem vnašajo zdravniki ob periodičnih obiskih bolnikov, ustreznost pa preverjajo neodvisni strokovnjaki. Ob koncu raziskave je potrebno generirati različna poročila o poteku. Predhodno so se raziskave izvajale na papirju. Ob koncu raziskave je bilo potrebno

rezultate (v fizični obliki) pridobiti od vseh sodelujočih zdravnikov in jih pretvoriti v elektronsko obliko. Osredotočili se bomo na segment za pregled vnešenih podatkov.

Kljub različni problemski domeni projektov gre pri izbranih segmentih za podobno problematiko ter podoben obseg. Oba projekta je razvijala ista razvojna ekipa, sestavljena iz vodje projekta, specialista za uporabniške vmesnike in dveh programerjev.

5.1 Potek razvoja

Pri obeh primerih je bila uporabljena metodologija razvojnega prototipiranja. Prototipiranje se je začelo z izdelavo statičnih pritotipov, ki so se kasneje nadgradili v dinamične in na koncu v delujočo aplikacijo. Za hitrejše generiranje statičnih prototipov smo v podjetju razvili PHP knjižnico. Knjižnica zna generirati osnovne HTML gradnike, kot so: tabele, seznami, menuji, grafi, gumbi in vnosni obrazci. HTML gradniki so generirani v skladu s CSS frameworkom Bootstrap, tako da so že avtomatično prilagodljivi za različne velikosti zaslonov in različne naprave, hkrati pa imajo tudi sodoben izgled.

5.1.1 Projekt A

Problemska domena

Pregled statistik oglasov omogoča uporabnikom podrobno analizo obnašanja oglaševalske kampanje v času. Dobra analiza pa je predpogoj za optimizacijo. Statistike oglasov predstavljajo metrike, kot so prikazi, kliki, čas izpostavljenosti, cena ipd. Problem pri optimizaciji oglaševalskih kampanj je v veliki količini podatkov, ki jih je potrebno spremljati. Glavni cilj segmenta je bil poiskati najbolj ustrezen način za prikaz podatkov.

Opis procesa

Razvoj smo začeli z izdelavo statičnega prototipa. Za določitev osnovnega okvira uporabniškega vmesnika smo potrebovali tri verzije prototipa. Prvo večje testiranje s strani uporabnikov se je zgodilo po izdelani četrti verziji prototipa. Prototip je vseboval že dobro izdelan interaktiven grafični vmesnik s testnimi podatki. Sledilo je prilagajanje tako vmesnika kot zaledne logike. Za določitev končne rešitve smo potrebovali štiri verzije prototipa (skupno sedem). V osmi verziji smo rešitev dodelali v končno rešitev. Deveta verzija pa je bila namenjena piljenju uporabniškega vmesnika.

Verzije 1, 2 in 3:
Z uporabo statičnega prototipa smo prišli do grobega okvira uporabniškega vmesnika.

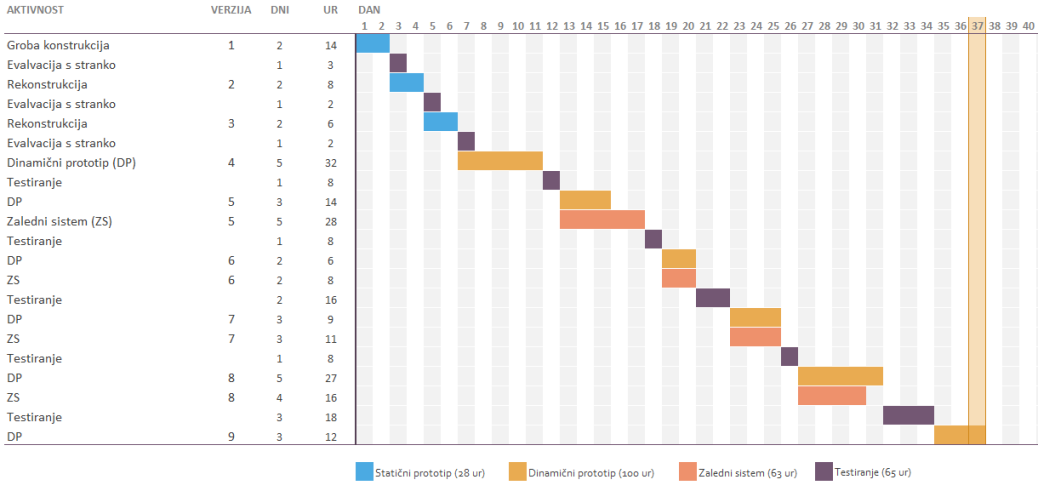
Verzija 4:
Nadgrajevanje v dinamični prototip z interaktivnim uporabniškim vmesnikom in testnimi podatki.

Verzija 5:
Začetek izdelave zalednega sistema.

Verziji 6 in 7:
Prilagajanje vmesnika in zalednega sistema glede na povratne informacije uporabnikov.

Verzija 8:
Nadgradnja prototipa v končno rešitev.

Verzija 9:
Prilagajanje uporabniškega vmesnika za različne tipe uporabnikov ter zaključne izboljšave uporabniškega vmesnika.



Slika 5.1: Diagram poteka razvoja projekta A.

Za razvoj segmenta je bilo potrebnih devet verzij prototipa. Segment je zahteval skupno 191 razvojnih človek-ur in bil razvit v 37 dneh.

5.1.2 Projekt B

Problemska domena

Pregled stanja raziskave omogoča spremljanje napredka vsakega izmed pacientov. Omogoča tudi statistični prikaz odgovorov na določeno vprašanje ter izpostavlja nepričakovana odstopanja. Raziskovalec z izbiro ustreznih nastavitev generira poročila, ki se mu prikažejo na zaslonu.

Opis procesa

Razvoj smo, prav tako kot pri projektu A, začeli z izdelavo statičnega prototipa. Prototip smo že takoj po prvem usklajevanju s stranko nadgradili v dinamičnega. S tretjo verzijo prototipa smo določili vse glavne funkcionalnosti ter končni izgled uporabniškega vmesnika. Sledil je razvoj prototipa glede na bogate povratne informacije uporabnikov. S sedmo verzijo smo prototip

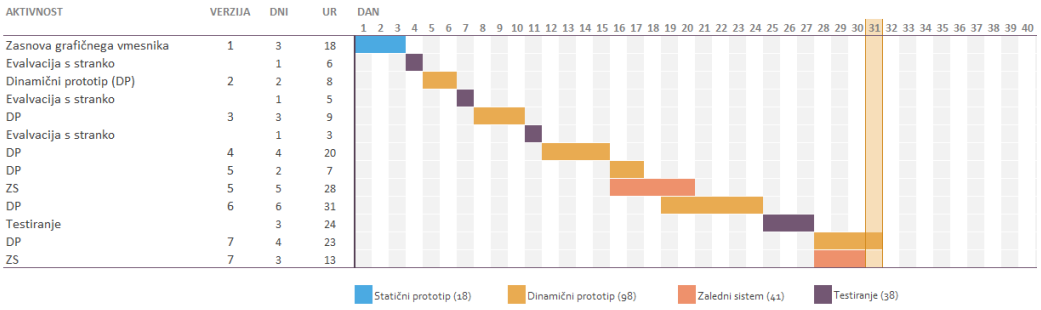
nadgradili v končno rešitev in odpravili vse nepravilnosti.

Verzija 1:
Izdelan statični prototip v obliki žičnega modela.

Verziji 2 in 3:
Nadgradnja v dinamični prototip z interaktivnim uporabniškim vmesnikom in testnimi podatki ter prilagajanje glede na zahteve stranke.

Verzije 4, 5 in 6:
Začetek izdelave zalednega sistema in tesnega sodelovanja z uporabniki. Z uporabo sistema za sporočanje napak so uporabniki posredovali povratne informacije na dnevni ravni.

Verzija 7:
Nadgradnja prototipa v končno rešitev.



Slika 5.2: Diagram poteka razvoja projekta B.

Za razvoj segmenta je bilo potrebnih sedem verzij prototipa. Segment je zahteval skupno 157 razvojnih človek-ur in bil razvit v 31 dneh.

5.2 Analiza

Za primerjavo bomo analizirali porabo časa ter število verzij prototipa. Osredotočili se bomo na štiri pomembnejše segmente razvoja:

- **določitev žičnega modela**: razvojna ekipa, v sodelovanju s stranko, določi vse glavni segmente uporabniškega vmsenika ter funkcionalnosti
- **potrditev dinamičnega prototipa**: v sodelovanju z uporabniki se izdela dovolj dobra verzija prototipa za nadgradnjo v končno rešitev
- **testiranje**: postopek preverjanja prototipa
- **izdelava zalednega sistema**

Metrike, ki jih bomo primerjali, so čas, število dni in število verzij prototipa. Čas bo podan v številu človek-ur vloženih v razvoj, v primeru testiranja pa bo čas število ur, potrebnih za izvajanje testiranja. Primerjavo metrik ponazarja tabela 5.1.

	A			B		
	ver.	dni	ur	ver.	dni	ur
določitev žičnega modela	3	6	28	3	8	35
potrditev prototipa	5	18	88	3	15	58
testiranje	6	11	63	4	6	38
razvoj zalednega sistema	4	12	81	2	8	41

Tabela 5.1: Primerjava časa in števila potrebnih verzij pomembnejših segmentov pri projektih A in B.

5.2.1 Določitev žičnega modela

Določitev žičnega modela je pri obeh projektih zahtevala tri verzije prototipa. Za razliko od primera A smo v primeru B za določitev žičnega modela uporabili tudi dinamični prototip. Z uporabo razvite rešitve za izboljšanje

prototipiranja smo lahko statični prototip, z relativno malo truda, hitro nadgradili v dinamičnega. Za proces določitve je bilo v primeru B potrebno več vloženega časa, a sta bila definicija funkcionalnosti ter izgled uporabniškega vmesnika veliko bolj natančna.

5.2.2 Potrditev prototipa

V primeru A je razvoj prototipa potekal po načelu razvoj - testiranje. Med razvojem verzije uporabniki prototipa niso testirali. Povratne informacije smo dobivali periodično. Pri razvoju projekta B pa smo povratne informacije pridobivali zvezno. Ustreznost delovanja prototipa so uporabniki preverjali na dnevni ravni. Razvoj projekta B se je zato lahko odvil hitreje. Do potrjenega prototipa smo pri projektu A prišli v petih verzijah prototipiranja, pri projektu B pa v treh. Več verzij prototipa pa običajno pomeni tudi več vloženega časa, kar se je tudi pokazalo pri projektu A.

5.2.3 Testiranje

Pri projektu A je testiranje potekalo tako, da se je vsako verzijo prototipa testiralo v celoti. Po opravljenem testu smo od uporabnikov dobili povratne informacije, ki so nam pomagale pri razvoju naslednje verzije prototipa. Pri projektu B je testiranje po istem principu potekalo samo v fazi določanja žičnega modela, potem pa se je testiranje s strani uporabnikov izvajalo tudi med razvojem verzije. Povratne informacije smo dobivali zelo pogosto. Pozitiven učinek sprotnih povratnih informacij je v tem, da je bil čas, vložen v razvoj, veliko bolje izkoriščen kot v primeru A.

5.2.4 Razvoj zalednega sistema

Zaradi nejasnih zahtev je razvoj napačno delujočih funkcionalnosti pri projektu A zahteval kar nekaj časa. Zaradi periodičnega testiranja se je napačna funkcionalnost detektirala prepozno.

5.3 Ugotovitve

Pri obeh primerih se je izkazalo, da je stranka dobila pravo sliko o aplikaciji šele ob stiku uporabnikov z dinamičnim (funkcionlanim) prototipom. Po nekaj preizkusih uporabe so hitro našli pomanjkljivosti in napake v zasnovi. Pri projektu A se je to zgodilo v četrti verziji prototipa, pri projektu B pa v drugi. Sledilo je več verzij prototipov za iskanje ustrezne rešitve. Prav tu se je izkazala prednost dinamičnih prototipov. Z zmožnostjo generiranja testnih podatkov in enostavnostjo vključevanja podatkov v HTML nam je uporaba razvitega modula pri projektu B omogočila testiranje več različic uporabniškega vmesnika v krajšem času.

Generiranje testnih podatkov je pri projektu B omogočilo izdelavo zelo realnega prototipa v zgodnjih verzijah. Vsi elementi grafičnega vmesnika so vsebovali podatke, ki so uporabnikom dajali vtis realne situacije. Generiranje podatkov se je izkazalo uporabno tudi z vidika testiranja prikazovanja raznih elementov grafičnega vmesnika. Interaktivno smo lahko z uporabniki testirali obnašanje tabel pri veliki in majhni količini podatkov in določili ključne funkcionalnosti, predvsem za filtriranje in iskanje vnosov.

Sistem za sporočanje napak se je izkazal za odlično rešitev. Uporabniki so lahko na enostaven način javljali nepravilnosti z razširjenimi informacijami o stanju aplikacije pri pojavitvi napake. Sistem nam je zelo olajšal reševanje težav, ki so se pojavile pri prikazu grafičnega vmesnika v različnih brskalnikih.

Težava modula, ki se je večkrat pojavila, pa je bila v vsakokratnem spreminjaju podatkov ob osveževanju strani. Ob vsakokratnem nalaganju strani modul zgenerira nove naključne podatke. To je uporabnikom v začetku povzročalo nekaj težav.

Poglavje 6

Zaključek

Pri izdelavi prototipov spletnih aplikacij se razvijalci pogosto poslužujejo različnih oblik statičnih prototipov (generirani žični diagrami, skice). Ti s svojimi omejitvami pogosto preslabo simulirajo končni produkt, zato je za iskanje ustrezne rešitve potrebno izdelati več verzij prototipa. S povečevanjem števila verzij se podaljšuje čas razvoja s tem pa tudi strošek celotnega projekta. Izdelava dimaničnih prototipov omogoča podrobnejšo predstavitev končnega produkta in možnost hitrejšega preverjanja domnev. Prednost dinamičnih prototipov je v zmanševanju števila potrebnih verzij prototipa, slabost pa v kompleksnosti izdelave.

Ne glede na tip izdelanega prototipa je pri metodologiji prototipiranja velik povdarek na komunikaciji med razvijalci in uporabniki. Ko razvijalci izdelajo verzijo prototipa, jo uporabniki pretestirajo, zberejo povratne informacije in te informacije posredujejo razvijalcem. Na podlagi teh izdelajo novo, izboljšano verzijo prototipa. V primeru neustreznega obnašanja prototipa, povratne informacije uporabnikov pogosto ne obsegajo dovolj podatkov za razvojno ekipo. Odzivi uporabnikov, kot denimo *"Izračun ni točen"* ali *"Ko kliknem na gumb, se mi podre tabela"* ne vsebujejo dovolj informacij o vseh vplivih za pojavljanje težav, zato je potrebno s strani razvijalcev veliko dodatnega dela, da simulirajo stanje prototipa, pri katerem je do napake prišlo.

Za pomoč pri izdelavi dinamičnih prototipov in posredovanju povratnih informacij sem predstavil rešitev, ki razvijalcem olajša nadgrajevanje statičnih prototipov v dinamične; uporabnikom pa poenostavi sporočanje težav prototipa. Zmožnost avtomatičnega vstavljanja podatkov v HTML, generiranja naključnih podatkov ter simulacije poizvedb na zaledni sistem pohitri proces izdelave dinamičnih prototipov.

6.1 Nadaljnji razvoj

Pri uporabi na praktičnem primeru se je rešitev izkazala za dober prototip. Z vsemi komponentami, ki prototip naredijo bolj realen, uporabnikom omogoča, da se vživijo v situacijo dejanske uporabe aplikacije. To omogoča zajem boljših povratnih informacije uporabnikov, predvsem pa možnost hitrejšega iskanja ustrezne rešitve. Seveda pa bi lahko rešitev še dodatno nadgradil in dodelal.

Generiranje naključnih podatkov
Težava, ki se je pojavila v praksi, je v spreminjaju naključnih podatkov ob osveževanju strani. Generiranje bi razširil tako, da bi lahko uporabnik 'zaklenil' osveževanje. Podatke bi moral na nek način shraniti, da bi jih lahko ob naslednji osvežitvi zopet prikazal.

Sporočanje napak
Pri pošiljanju podatkov o napaki, bi lahko poleg uporabnikovih in podatkov o stanju prototipa, izvedel še zajemanje slike zaslona (*angl. printscreen*). To bi naredil z uporabo PhantomJS[24].

Dodatne funkcionalnosti
Izdelal bi množico pogosto uporabljenih funkcionalnosti, ki bi jih lahko razvijalci enostavno vgradili v svoje prototipe. Take funkcionalnosti so denimo preverjanje ustreznosti raznih vnosnih polj, možnost sortiranja tabel ipd.

Slike

Literatura

[1] P. Szekely, "User interface prototyping: Tools and techniques", *Software Engineering and Human-Computer Interaction*, str. 76–92. Springer, 1994.

[2] M. Fitzgerald (2007) How Simulation Software Can Streamline Application Development. [Online]. Dosegljivo: http://www.cio.com/article/2442714/developer/how-simulation-software-can-streamline-application-development.html. [Dostopano 6. 5. 2016].

[3] R. Kaur, J. Sengupta. "Software process models and analysis on failure of software development projects", *arXiv preprint arXiv:1306.1068*, 2013.

[4] J. Crinnion, "The evolutionary development of business systems", *Software Prototyping and Evolutionary Development, IEE Colloquium on.* IET, 1992.

[5] A. T. Sadabadi, N. M. Tabatabaei. "Rapid prototyping for software projects with user interfaces", *Department of Computer Systems and Informatics Seraj Higher Education Institute, Iran, št*, št. 9, str . 85–90, 2009.

[6] K. Beck. "Extreme programming explained: embrace change". Addison-Wesley Professional, 2000.

[7] A. Aitken, V. Ilango. "A comparative analysis of traditional software engineering and agile software development", *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, str . 4751–4760. IEEE, 2013.

[8] M. F. Smith. "Software Prototyping". McGraw-Hill, 1991.

[9] S. McConnell. "Rapid development: taming wild software schedules". Pearson Education, 1996.

[10] A. M. French. "Web development life cycle: a new methodology for developing web applications", *The Journal of Internet Banking and Commerce*, 2015.

[11] M. Bogers, W. Horst. "Collaborative prototyping: Cross-fertilization of knowledge in prototype-driven problem solving", *Journal of Product Innovation Management*, št. 4, zv. 31, str . 744–764. Wiley Online Library, 2014.

[12] A. Saxena, P. Upadhyay. "Waterfall vs. Prototype: Comparative Study of SDLC", *Imperial Journal of Interdisciplinary Research*, št. 2, zv. 6, 2014.

[13] S. Komatineni (2006) Reshaping IT Project Delivery Through Extreme Prototyping. [Online]. Dosegljivo: http://archive.oreilly.com/pub/a/onjava/2006/11/15/reshaping-it-project-delivery-through-extreme-prototyping.html. [Dostopano 8. 5. 2016].

[14] K. Kendall, J. Kendall, C. S. Wasson. "Systems analysis and design", št. 19. Year Prentice Hall, 2014.

[15] J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna, F. Montero, M. Gaedke. "Mockup-driven development: Providing agile support for model-driven web engineering", *Information and Software Technology*, št. 56, zv. 6, str . 670–687. Elsevier, 2014.

[16] Modul pType. [Online]. Dosegljivo:
https://github.com/lakiluks/pType. [Dostopano 28. 6. 2016].

[17] GIMP. [Online]. Dosegljivo:
https://www.gimp.org. [Dostopano 21. 6. 2016].

[18] Adobe Photoshop. [Online]. Dosegljivo:
http://www.adobe.com/products/photoshop.html. [Dostopano 21. 6. 2016].

[19] Axure. [Online]. Dosegljivo:
http://www.axure.com. [Dostopano 22. 6. 2016].

[20] ProtoShare. [Online]. Dosegljivo:
http://www.protoshare.com. [Dostopano 22. 6. 2016].

[21] jQuery. [Online]. Dosegljivo:
https://jquery.com. [Dostopano 24. 6. 2016].

[22] Chance. [Online]. Dosegljivo:
http://chancejs.com. [Dostopano 22. 6. 2016].

[23] jQuery Mockjax: Ajax request mocking. [Online]. Dosegljivo:
https://github.com/jakerella/jquery-mockjax. [Dostopano 25. 6. 2016].

[24] Phantom JS. [Online]. Dosegljivo:
http://phantomjs.org. [Dostopano 25. 6. 2016].