

Softversko inženjerstvo

Specifikacija projekta

Računarski fakultet

11. mart 2021.

Važne napomene

1. Projekat se radi u četvoročlanim timovima. Ukoliko neko ne može da pronađe tim, uvek može da mi se obrati za pomoć. U specifičnim slučajevima, biće dopuštena izrada zadataka u timovima od 3 čoveka ili u paru. Samostalan rad nije prihvatljiv. Kriterijum ocenjivanja će biti identičan za sve timove, bez obzira na to koliko članova broje.
2. Timovi se moraju prijaviti do subote, 13. marta 2021. u 14h. Studenti koji ne prijave tim do predviđenog roka neće moći da rade projekat i samim tim neće moći da polože predmet ove školske godine.
3. Obavezno je korišćenje dodeljenog timskog repozitorijuma. Modeli moraju biti spakovani u PD projekat, pa u .zip arhivu pod nazivom "Modeli", i okačeni na glavnu granu repozitorijuma. Ostali segmenti koda se mogu kačiti na glavnu, ili na proizvoljne grane tokom ostatka semestra.
4. Korišćenje biblioteka i/ili specifičnih tehnologija je dozvoljeno isključivo na frontendu. Serverska strana može biti napisana u proizvoljnom programskom jeziku, ali ne treba da se osloni na posebnu tehnologiju (npr. Apache Zuul nije pogodan za okupljanje servisa i rutiranje zahteva ka njima).

Rok za predaju projekta

KT1: Nedelja 21. Mart 2021 u 23:59h. Odbrana je prema rasporedu, u četvrtoj nedelji vežbi.

KT2: Nedelja 11. april 2021. u 23:59h. Odbrana je sedmoj nedelji vežbi.



Pregled alata za modelovanje

Tema projekta je alat namenjen modelovanju softverskih rešenja, namenjen projektnim timovima. Projektnim timom se smatra skup registrovanih korisnika koji će zajedno raditi na dizajnu proizvoljnog softvera. U radnom prostoru alata, korisnik može kreirati projekat, i u okviru projekta napraviti novi model. Uz oslonac na ovo rešenje, on može da kreira modele zahteva, funkcionalne modele i klasne dijagrame.

Kako bi se ubrzali procesi dizajna i razvoja, alat obezbeđuje podršku za transformaciju: modela zahteva u funkcionalni model i klasnog modela u Java kod.

Zajednički rad na projektu je obezbeđen kroz praćenje verzija projekata, slično kao što funkcionišu i Git/SVN. Svoje verzije modela korisnik komituje u neku bazu ili na server, a saradnici je mogu povući sa servera i spojiti sa svojim verzijama modela. Ovo može da funkcioniše i na drugačiji način, zavisno od izbora arhitekture. Zamisao je da praćenje i sinhronizovanje verzija bude jedna od funkcionalnosti.



KT1

Rok za predaju projekta: Nedelja 21. Mart 2021 u 23:59h. Odbrana je prema rasporedu, u četvrtoj nedelji vežbi.

1. Definirati Readme.md na timskom Git repozitorijumu. Readme fajl treba da bude u formatu kraće Wiki stranice.
2. Isplanirati aktivnosti i postaviti Ciljeve i Zadatke na GitHub-u. Predvideti postojanje kontrolnih tacaka, a zasebne zadatke izdavati prema specifikaciji svake KT (kada je dobijete).
3. Kreirati timski PD projekat i specificirati RQM i UseCase modele. Postaviti modele u skladu sa opisom iz odeljka “Važne napomene”.
4. Na osnovu analize arhitektonskih šablona koje smo radili na vežbama, odabrati 3-4 šablona koja bi mogla biti pogodna za razvoj alata za modelovanje. Napisati kratku studiju slučaja kojom se ističu prednosti i mane svakog od odabranih šablona, i odabrati neki pristup (ili kombinaciju) za arhitekturu vašeg rešenja. Obrazložiti. Format za predaju je zasebna .md stranica.

KT2

Rok za predaju projekta: Nedelja 11. april 2021. u 23:59h. Odbrana je sedmoj nedelji vežbi.

Broker (Middleware) - SOA/Mikroservisi

Osnovne funkcionalnosti

Broker treba da bude realizovan kao web servis koji će imati REST rute za:

- Administraciju* servisa (njihovih endpointa, specifikaciju ulazno/izlaznih parametara, i uloga koje tim rutama mogu da pristupe**)
- Rutiranje zahteva
- Administraciju korisnika***

*** Administracija podrazumeva kompletan set CRUD funkcionalnosti.**

**** Uloge opciono mogu biti razrešene i uz pomoć zasebnog servisa/na konkretnim servisima koji prihvataju zahteve.**

***** Administracija korisnika može da bude realizovana i kao zaseban servis. Podrazumeva registraciju novih korisnika, promenu lozinke, prijavu na sistem (log in), i organizaciju korisnika u timove.**


Broker radi uz oslonac na relacionu MySQL bazu podataka.

Registracija servisa

Servisi koji su deo sistema se automatski registruju na brokera prilikom njihovog pokretanja.

Registracija znači da oni javljaju brokeru da:

1. Kako se zovu (npr. Search)
2. Na kom domenu rade i kom portu rade (npr. google.com:80)
3. Na kojoj ruti (ili kojim rutama) pružaju funkcionalnosti (npr. /image i /text)
4. Kojim metodom HTTP zahteva se aktivira
5. Izgled HTTP zahteva/odgovora



Definisanje naziva servisa i rute na kojoj on radi (npr. Search servis iz prethodnog pasusa) može da ima funkcionalnost za pretragu po unetom tekstu koja radi na ruti /text (znači da je cela putanja google.com:80/text) i po slici koji radi na ruti /image. Ne zaboravimo da je servis u SOA krupnije granularnosti od mikroservisa: zavisno od toga kakvu organizaciju napravite, moguće je da ćete na servisu imati jednu ili više funkcionalnosti. Svaka ruta treba da ima metod HTTP zahteva kojim se aktivira funkcionalnost. Za nju pišemo i specifikaciju parametara koje prihvata, kao i onoga što vraća. Ta specifikacija je kao metašema za zahtev/odgovor. Za svaki parametar unosa moramo da znamo kako se prenosi (request parametar, URL promenljiva, u telu zahteva), koji je naziv (labela) sa kojom želimo da programer sa klijenta prosledi vrednost, kao i tip podatka. Slično je i sa rezultatom koji će se vratiti - programer koji poziva servis mora da zna šta da očekuje kako bi adekvatno modelovao svoju aplikaciju!

Ako sledimo Search primer, na kraju procesa registracije, servis Search će imati dve funkcionalnosti, jednu na ruti /image i jednu na /text (google.com:80/search/text i google.com:80/search/image, respektivno) koje dopuštaju pretragu po unetom stringu ili slici. Za te dve putanje bi se razlikovale šeme zahteva - jedna prihvata tekstualni podatak u recimo parametru zahteva searchTerm, dok drugi očekuje da mu prosledimo sliku. Od vas se ne očekuje da radite sa slikom, ovo je samo primer. Registracija servisa, kao i registracija korisnika, podatke smešta u bazu.

Rutiranje zahteva

Broker će imati jednu rutu na kojoj prihvata zahteve. Pošto se servisi mogu registrovati dinamički, nema smisla praviti po jednu rutu koja će se mapirati na svaki mogući endpoint, već ovo treba da bude objedinjeno jednom funkcijom brokera (npr. jednom rutom sa mnogo URL promenljivih).

Core - Mikrokernel

Osnovne funkcionalnosti

Core treba da bude realizovan kao središnji deo koji obezbeđuje:

- Registraciju komponenti
- Konfiguraciju aplikacije
- Rutiranje poruka između komponenti *
- Administraciju korisnika**

***** Rutiranje poruka je u slučaju da komponente nisu povezane Observerom, ili nekim sličnim mehanizmom.***

***** Administracija korisnika može da bude realizovana i kao zasebna komponenta. Podrazumeva registraciju novih korisnika, promenu lozinke, prijavu na sistem (log in), i organizaciju korisnika u timove.***

Jezgro aplikacije treba se ponašati kao okupljajuća komponenta za sve druge komponente koje će se pojaviti u aplikaciji.

Ono može da uključi administraciju korisnika i uloga, a može to da prepusti i konkretnoj komponenti. Treba da obezbedi registraciju komponenti u formi plagina: Core deo je zadužen da okupi standardne interfejsa i da ima ulogu orkestriranja razmene poruka. On instancira konkretne komponente i vezuje njihove implementacije za gorepomenute interfejsa. Dodatno, pored inicijalizacije komponenti, ono treba da bude zaduženo i za konfigurisanje aplikacije prilikom pokretanja.

Primer modularnog rešenja:

- Core
- Komponenta1 ima interfejs GUI sa standardnom metodom start(). On ima dve standardne implementacije SwingGUI i FxGUI (očigledno, zbog interfejsa, oni moraju imati iste metode za uključivanje grafičkog korisničkog interfejsa).
- Komponenta 2 ima interfejs Repository koja predstavlja model aplikacije i njegovu osnovnu implementaciju RepositoryImpl.
- Konfiguracija predstavlja podešavanje komponenti prilikom pokretanja aplikacije: jedna validna konfiguracija je SwingGUI-RepositoryImpl, a druga FxGUI-RepositoryImpl. U oba slučaja, promena GUI komponente ne utiče na rad Repository komponente, niti iziskuje promene u kodu da bi aplikacija nesmetano radila, jer su komponente potpuno nezavisne i koriste interfejsa za komunikaciju. Ukoliko su neke od komponenti povezane Observer šablonom, što može biti zgodno za automatsko pozivanje funkcija nakon proizvoljnog događaja, ovo se takođe određuje konfiguracijom.

Administracija korisnika

Aplikacija treba da obezbedi registraciju i dodeljivanje uloga prema specifikaciji opisanoj u modelu zahteva. Neophodno je da se korisnici mogu registrovati, da mogu ukloniti nalog, promeniti lozinku i prijaviti se na sistem.

Potrebno je realizovati grupisanje korisnika u timove. Članovi tima dele resurse u aplikaciji. Potrebno je rešiti:

1. Ko kreira tim?
2. Kako dodeljuje nove članove?

Autorizacija i autentifikacija

Prilikom logovanja korisnik dobija JWT token sa dovoljno informacija da koristi sistem. Ukoliko je predviđeno modelom zahteva, korisnik prilikom registracije dobija ulogu u sistemu (npr. Tim lider / Korisnik). Uloga diktira stepen prava i postavlja restrikciju na rutama kojima korisnik pristupa.

Timske mogućnosti

Članovi tima mogu da kreiraju projekte i u projekte grupišu modele. Jedan projekat može imati više modela i ti modeli mogu biti istog tipa (npr. projekat može da ima više klasnih dijagrama).

Perzistencija

Struktura timova i projekata u njihovom vlasništvu se čuva u relacionoj MySQL bazi. Baza će se nalaziti na javnom serveru, kredencijale vam šaljem naknadno.

Model baze

U okviru KT2, potrebno je napraviti konceptualni model šeme baze koja će se koristiti za organizaciju timova, projekata i modela koje timovi razvijaju. Podsetnik za konceptualne modele ćemo imati u 6. nedelji vežbi.

RQM modelovanje

Potrebno je realizovati alat za modelovanje zahteva uz oslonac na izabrani arhitektonski šablon. Alat treba da bude GUI orijentisan, pri čemu GUI može da bude desktop ili veb klijent. Mobilna aplikacija nije opcija.

Kompozitna struktura u kontekstu modela zahteva je Projekat - RQM - Zahtev.

Svaki zahtev je bliže određen:

1. Rednim brojem i "nazubljenosti" (1, 1.2, 1.2.3, ...)
2. Naslovom
3. Opisom
4. Tipom (funkcionalni/nefunkcionalni)
5. Prioritetom (1-5)
6. Rizikom (H/M/L)
7. Dodeljenim korisnikom (krajnji korisnik mora da ima mogućnost da samostalno definiše korisnike i ponovo ih u potrebi u nekom drugom zahtevu)

Pošto RQM može a ne mora da uključi sve ove podatke, ima dosta teksta i organizaciju zahteva po dubini, nije idealan za konceptualno modelovanje. Zadatak je da se direktno mapira na JSON opis koji ćemo skladištiti u MongoDB (Mongo server će takođe biti na javnoj IP adresi, i dobićete kredencijale). To znači da je u relacionom modelu posebno naznačiti gde je uskladišten sadržaj modela.

GUI podržava funkcionalnosti za dodavanje/izmenu/brisanje/nazubljivanje i podešavanje bilo kog parametra pojedinačnog zahteva, kao i njegovo trajno čuvanje na serveru. Undo/Redo operacije nisu potrebne za RQM modelovanje.

KT3

Rok za predaju projekta: Sreda 5. maj 2021. u 23:59h. Odbrana je 11. nedelji vežbi.

Modeli

Ažurirati RQM i Use Case prema sugestijama koje ste dobili od profesora. Manje se fokusirati na elemente koje ćemo implementirati, a više na funkcionalnosti koje razlikuju naš alat od PD: to su verzionisanje i kolaborativni mehanizmi, koji su u dosadašnjim modelima mahom bili predstavljeni jednim zahtevom. Modele praviti uz oslonac na [SWEBOK](#): uključiti dodatne uloge (npr. analitičar zahteva), i definisati njihove poglede na sistem. Ovo važi za sve tipove modela koji mogu da nastanu u vašem alatu!

GUI

U sklopu KT3 razvijaju se grafički alati, zasnovani na *drag-and-drop* mehanizmu. Korisnik ima na raspolaganju paletu elemenata (na osnovu apstraktne sintakse) koji se mogu naći na modelu, i pravi objekte na osnovu toga - obratiti pažnju da i veze treba da se realizuju prevlačenjem miša po ekranu.

Paleta:

- elementi modela i veze između njih

Akcije po kanvasu:

- Selektuj element, prepoveži, pomeri, obriši, izmeni
Podsetnik: State šablon je pogodan za implementaciju grafičkog editora (materijali sa DSW).

Funkcionalnosti editora:

- UNDO i REDO operacije sa neograničenom istorijom za sve akcije po kanvasu.

Podsetnik: *Command šablon je pogodan za implementaciju istorije (materijali sa DSW).*

Implementaciona napomena: *Undo/redo realizovati ili kao zasebnu komponentu ili kao podkomponentu grafičkog editora.*

Use Case

Funkcionalni modeli treba da objedine aktere (akter može da ima ime), i slučajeve korišćenja (definisani samo nazivom). Postoje 4 tipa veze: generalizacija, include, extend, asocijacija.

Class model

Klasni dijagrami se sastoje od klasa (mogu biti apstraktne) i interfejsa, koji su povezani vezama asocijacije, generalizacije ili realizacije (veza može biti na nivou *klasa-klasa*, *klasa-interfejs*, *interfejs-interfejs*). Svaka klasa može da ima attribute određenog tipa i opsega vidljivosti (privatni, javni) i metode (privatne, javne, imaju povratni tip). Nazivi za svaki element su obavezni.

Tip može da bude String, int ili neka druga klasa koja je definisana u modelu.

Metamodel i validacija

Za funkcionalne i klasne dijagrame izvojiti POSEBNU KOMPONENTU za validaciju. Validacija se radi u odnosu na jezik: to efektivno znači da morate da napravite DS jezik (metamodel i skup pravila) za svaki od ovih modela, i da ta pravila isforsirate prilikom čuvanja (izmene). **Kako su modeli koje pravite poznati još iz druge godine, pretpostavlja se da znate skup pravila i ograničenja za svaki od njih.** S obzirom na to da je validacija korak koji prethodi transformaciji, nije dovoljno zabraniti određene akcije kroz grafički editor.

Primer: između dve klase je dozvoljena veza nasleđivanja. Validacijom prilikom čuvanja garantujemo da nije modelovano dvostruko nasleđivanje.

Pitanja i zadaci:

- ***Kako napraviti i sačuvati metamodel u komponenti?***

- 
- ***Kako odraditi usaglašavanje modela sa metamodelom?***

Napomena: Metamodel na osnovu koga radite validaciju priložiti i u grafičkoj formi (neophodno za odbranu, npr. može da se realizuje upotrebom UML-a).

Čuvanje modela

Modele možete čuvati u dokumentalističkoj ili relacionoj bazi. Savet je da funkcionalni model ima dokumentalističku reprezentaciju kako bi M2M transformacije bile lakše.

Integracija

Editore integrisati sa ostatkom projekta - obezbediti da se koriste sa adekvatnom autorizacijom i autentifikacijom, kao i da se njihove funkcionalnosti aktiviraju kroz Middleware/AppCore, u zavisnosti od izabrane arhitekture. Napravljene modele je potrebno povezati sa odgovarajućim timom i projektom.

KT4

Rok za predaju projekta: Nedelja 30. maj 2021. u 23:59h. Odbrana je poslednjoj nedelji vežbi. U kolokvijumskoj nedelji je predviđena prezentacija kod profesora.

Modeli

BPM - Realizovati model koji prikazuje proces dizajniranja (modelovanja) jednog softverskog proizvoda: od zahteva do kostura koda. Koji sve akteri (zainteresovane strane) učestvuju u modelovanju?

EAM - Uz oslonac na studiju slučaja iz KT1 dati prikaz arhitekture softvera.

Verzionisanje

Obezbediti praćenje verzija modela prema nekom identifikatoru. Različite verzije modela se čuvaju uz detalje o identifikaciji, i korisniku koji je napravio izmenu (inicirao čuvanje verzije).

Kolaboracija

Opredeliti se za mehanizam kolaboracije: push/pull ili real-time. Zahvaljujući ovoj funkcionalnosti, više članova tima može da radi na istom modelu u isto vreme.

Potrebni koncepti: **verzija** modela koja se menja (čini se da je većina timova izmene čuvala na zahtev korisnika), **spajanje** modela (merge), **konflikt** i **razrešenje**. S obzirom na to da spajanje modela i konflikti spadaju u domen upravljanja modelima, obratiti pažnju na to da li ih treba izdvojiti u zasebnu komponentu (u zavisnosti od izabranog tipa arhitekture - granularnost mikroservisa ne trpi da sve bude smešteno u jednu komponentu).

Prilikom modelovanja korisnik menja sadržaj/strukturu dokumenta koji predstavlja serijalizovani oblik modela. Save/Push operacija šalje novu verziju modela u bazu. Može se dogoditi da dva ili više korisnika menjaju model u istom trenutku, polazeći od iste verzije. To povlači da se oni moraju spojiti kada oba korisnika pokušaju da sačuvaju svoje izmene, jer je na primer, jedan korisnik poslao svoju verziju nešto pre drugog korisnika. To znači da, kada drugi korisnik pokuša

da pošalje svoje izmene, mora da ih integriše sa izmenama prvog korisnika. Tokom tog procesa treba:

1. Pronaći elemente koji su promenjeni
2. Proveriti da li su izmene preklapajuće
3. Ako nisu, modeli se puštaju kroz proces spajanja (vaš algoritam) i prikaze osvežiti
4. Ako jesu, nastaje konflikt
 - a. Prikazati korisniku gde je došlo do problema uz odgovarajuću poruku (npr. "Pokušavate da izmenite element klase koji je već bio promenjen u _____")
 - b. Ponuditi da korisnik izabere svoje ili tuđe promene
 - c. Odraditi spajanje prema izabranim opcijama i proveriti validnost modela prilikom čuvanja nove verzije

Kod real-time kolaboracije treba implementirati nešto slično kao u Google Docs. U tom slučaju se spajanje radi prilikom svakog "poslatog inputa" od proizvoljnog korisnika, i konflikti nastaju u realnom vremenu... To znači da se strategija razrešavanja konflikata potencijalno mora prilagoditi.

Pitanja:

1. Da li forsirati da se pull radi pre svake push operacije?

2. Kako predstaviti konflikt u implementaciji?


3. Koja komponenta radi spajanje modela?

Transformacije

Potrebno je implementirati transformacije modela: iz RQM u USE CASE, i iz CLASS u Java kod. Transformator treba da bude posebna komponenta/servis. Samo modeli koji su validni mogu da se transformišu.

Za M2M transformaciju, definisati mapiranje između elemenata na osnovu kog će se raditi generisanje novog modela.

Proces:

- 
1. Provera da li je model koji želimo da transformišemo korektan
 2. Prosleđivanje modela transformatoru
 3. Transformator generiše novi FUM, i kreira se veza između polaznog i generisanog modela
 4. U slučaju izmene polaznog RQM-a, korisnik se obaveštava da modeli nisu sinhronizovani, i može da odabere ažuriranje generisanog modela

Napomena: Ne ograničavati broj dijagrama slučajeva korišćenja koji se mogu generisati i povezati sa jednim polaznim modelom zahteva.

Za M2C transformaciju, realizovati eksportovanje Java projekta. Za generator koda je takođe potrebno unapred odrediti mapiranje konstrukata iz klasnog dijagrama na elemente koda.

Klase i interfejsi se generišu kao posebni fajlovi, u klasi se pojavljuju svi atributi (public/private String/int/custom_type attr_name), generišu se geteri i seteri za sva polja (sa gotovom implementacijom), podrazumevani konstruktor, i potpisi metoda koje treba implementirati iz nadklase/interfejsa (mora biti prikazano i koje klase su nasleđene/interfejsi implementirani). Kolekcija ovih fajlova treba da predstavlja kostur koda za dalje proširenje. Proces:

1. Provera da li je model koji želimo da transformišemo korektan
2. Prosleđivanje modela transformatoru
3. Na osnovu strukture modela i poznatih pravila a mapiranje, iz modela se generiše kod
4. Rezultat je novi folder sa Java implementacijom

Pitanja:

1. ***Kako predstaviti mapiranja za transformacije?***
2. ***Da li je generator koda komponenta unutar transformatora/zasebna/nešto treće?***
3. ***Koja komponenta radi spajanje modela?***

Arhitektura

Poseban akcenat staviti na arhitekturu i modularnost rešenja. Integrisati sa KT2 i KT3. Ažurirati frontend i pripremiti za prezentovanje u kolokvijumskoj nedelji.