

queri

Verzija 1.1

Istorija revizija

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Verzija** | **Opis** | **Autor** |
| 06.03.2023. | 1.0 | Urađena je inicijalna verzija dokumenta. | Mladen Todorović |
| 07.03.2023. | 1.1 | Dorađen dokument. | Mladen Todorović |

Sadržaj

1. Uvod 4

1.1 Svrha 4

1.2 Područje primjene 4

1.3 Definicije, akronimi i skraćenice 4

1.4 Reference 4

1.5 Pregled 4

2. Opšte smjernice za dizajn i implementaciju 4

2.1 Mapiranje dizajna na implementaciju 4

2.2 Specifikacija interfejsa za podsisteme 4

2.3 Dokumentovanje koda 5

2.4 Otkrivanje, rukovanje i izvještavanje o greškama i izuzecima 5

2.5 Upravljanje memorijom 5

2.6 Distribucija softvera 5

2.7 Ponovna iskoristivost komponenti 5

2.8 Posebna upotreba jezičkih karakteristika 5

2.9 Struktura projekta 5

2.10 Smjernice za razvoj i izmjene sistema 5

3. Smjernice za dizajn baze podataka 5

4. Smjernice za dizajn arhitekture 6

5. Smjernice za mehanizme 6

# Uvod

Dokument *Smjernice za dizajn* je proizvod definicije arhitekture. Opisuje smjernice za dizajn koje obezbjeđuju efikasnu implementaciju.

## Svrha

Svrha dokumenta se ogleda u opisu standarda za dizajn, konvencija i idioma koji se koriste u dizajnu sistema.

## Područje primjene

Dokument *Smjernice za dizajn* predstavlja sponu i način za razmjenu informacija između različitih dijelova razvojnog tima, uključenih u planiranje arhitekture i implementaciju sistema. Olakšava usklađivanje njihovog rada, te sprječava nastanak problema, u toku realizacije, koji su uzrokovani različitim poimanjem sistema.

## Definicije, akronimi i skraćenice

**JavaDoc** - JavaDoc je proširivi sistem za generisanje dokumentacije koji čita posebno formatirane komentare u Java izvornom kodu i generiše kompajliranu dokumentaciju. Obično se koristi za izradu API dokumentacije u obliku HTML web stranica.

**Garbage collector –** Garbage collector je sistem automatizovanog upravljanja memorijom, tj. proces kojim Java briše nekorištene objekte iz memorije.

Sve potrebne definicije, akronimi i skraćenice, u domenu problema koji *queri* rješava, sadržane su u dokumentu *Rječnik*, koji je dio projektne dokumentacije.

## Reference

[1] *Projektovanje i dizajn softvera: koncepti, principi i projektni obrasci*, prof. dr Vladimir Vujović

[2] [*Software Design Principles and Guidlines*](https://www.dre.vanderbilt.edu/~schmidt/PDF/dp4.pdf), Douglas C. Schmidt

## Pregled

U nastavku su opisane stavke dizajna sistema koje su, zbog bolje preglednosti, grupisane u odgovarajuće cjeline.

# Opšte smjernice za dizajn i implementaciju

## Mapiranje dizajna na implementaciju

Dizajn sistema je nastajao postupno. Na osnovu analize zahtjeva zainteresovanih strana, kreiran je model zahtjeva čiji su funkcionalni zahtjevi direktno predstavljeni dijagramima slučajeva korištenja. Domenski i klasni dijagrami nastaju pod uticajem ovih dijagrama. Prepoznata je potreba za razdvajanjem modela podataka, pogleda i kontrolera, implementiranog kroz komandni projektni obrazac, koji će ih uskladiti.Cilj je dobiti bolje strukturiran MVC model koji je lako proširiv, ali i razumljiv za razvojni tim. Modelovanje se vrši u softverskom alatu SAP PowerDesigner, koji omogućava kreiranje osnovnih paketa MVC modela (*model*, *view*, *controller*), te detaljno specificiranje potrebnih klasa, interfejsa i modela, kao i veza između njih. U procesu implementacije, oni se prevode u istoimene pakete sa pripadajućim klasama, u Eclipse razvojnom okruženju. Preporučuje se kreiranje dodatnih potpaketa u cilju dobijanja bolje strukturiranog koda. Postojanje posebnog paketa *observer* (posmatrač) izdvaja i pojašnjava primjenu istoimenog projektnog obrasca.

## Specifikacija interfejsa za podsisteme

Gore navedeni paketi, tj. klase koje posjeduju, projektuju se sa ciljem da budu autonomni, ali je od presudnog značaja, za ispravno funkcionisanje sistema, njihovo povezivanje. Zbog toga se definišu interfejsi, kao izlaz, prema drugim klasama. Unutrašnje ponašanje klasa je “sakriveno”. Ovim su omogućene izmjene koje neće uticati na ostale dijelove sistema. Klase/interfejsi paketa *view* “kreiraju” standardizovan izgled softverskog alata, na koji potencijalna nadogradnja *modela* nema uticaja.

## Dokumentovanje koda

Cilj razvojnog tima je sistem koji je moguće lako održavati u budućnosti. Zbog toga je potrebno adekvatno dokumentovati sve bitne elemente. JavaDoc dokumentacija koda je obavezna, te svaka metoda i klasa moraju biti dokumentovane. Ovo obuhvata, opis klase/metode, povratna vrijednost (ako postoji), parametri metode, kao i naziv autora metode/klase. U toku izrade softverskog alata, dozvoljeni su inline komentari kako bi programeru olakšali implementaciju. Za njih nisu definisana posebna ograničenja. Imenovanje elemenata u kodu je detaljno obrađeno u dokumentu *Smjernice za programiranje.*

## Otkrivanje, rukovanje i izvještavanje o greškama i izuzecima

Neke potencijalne greške moguće je predvidjeti u toku projektovanja i spriječiti njihovo nastajanje, dok je mnoge teško prepoznati prije uvođenja sistema u eksploataciju. Takve greške se najčešće dešavaju pri interakciji sa korisnikom. Neophodno je u toku implementacije obezbijediti mehanizme za otkrivanje grešaka, te rukovanje njima, tj. sprovođenje određenih operacija u slučaju nastanka greške koje će obavijestiti korisnika o nastaloj grešci i, ukoliko je moguće, dovesti do oporavka od greške, bez prekida u radu sistema. Korištenje *try-catch* blokova predstavlja najjednostavniji način rukovanja izuzecima. Dio koda, za koji se može reći da je “problematičan”, tj. na kojem se očekuje izuzetak, potrebno je obmotati *try* blokom, uz adekvatne prateće rukovaoce (handler) unutar *catch* blokova.

## Upravljanje memorijom

Za ispravno funkcionisanje sistema neophodno je i efikasno upravljanje memorijom, tj. redukovanje bespotrebnog iskorištavanja memorijskih resursa. Sa smanjenjem slobodne memorije postaju lošije i performanse sistema. Korišćenje MVC arhitekture pri modelovanju obezbjeđuje smanjenje redundantnosti podataka, odnosno, efikasno korišćenje resursa. Za upravljanje memorijom koristiće se *Garbage collector* mehanizam Java programskog jezika.

## Distribucija softvera

Softverski alat *queri* je dostupan za preuzimanje na zvaničnom sajtu na internetu.

## Ponovna iskoristivost komponenti

Kao što je već navedeno, MVC arhitektura obezbjeđuje da se isti podaci, sadržani u modelu, koriste uz različite prikaze. To se može iskoristiti za projektovanje istog sistema prilagođenog različitim okruženjima. Kako je MVC sastavljen iz 3 potpuno nezavisne komponente, one se mogu iskoristiti i za projektovanje nekih sličnih softverskih alata tj. grafičkih editora.

## Posebna upotreba jezičkih karakteristika

Ispravan način projektovanja podrazumijeva da se sistem modeluje tako da ne zavisi od programskog jezika koji će se koristiti za implementaciju. To znači da se sistem može implementirati u bilo kom jeziku, bez potrebe za izmjenom modela. Zbog toga se prilikom dizajna ne navode nikakve restrikcije koje nameće neki određeni programski jezik.

## Struktura projekta

Kompletan projekat treba da bude podijeljen u logičke cjeline predstavljene paketima. Svi paketi treba da se nalaze u okviru *source* paketa tj. da budu njegovi potpaketi. Pored toga, potrebno je obezbijediti i direktorijume u koji će se smještati ostali resursi softverskog alata, kao što su ikonice, slike i ostalo.

## Smjernice za razvoj i izmjene sistema

Ako projektni tim uvaži preporuke sadržane u ovom dokumentu, ali i u ostaloj pratećoj dokumentaciji, kao proizvod svog rada će imati sistem, softverski alat, pogodan za izmjene uz minimalne napore. Dobro izmodelovan sistem je moguće implementirati, bez većih problema, koristeći i druge programske jezike i tehnologije.

# Smjernice za dizajn baze podataka

Softverski alat *queri* ne koristi sopstvenu bazu podataka. Kreirani projekti i modeli se skladište na sekundarnoj memoriji računara.

# Smjernice za dizajn arhitekture

U skladu sa karakteristikama problema koji se rješava, za projektovani sistem je izabrana MVC (Model – View – Controller) arhitektura koja obezbjeđuje različite načine interakcije i prikaza podataka u sistemu, kao i njihove nezavisne promjene tokom životnog ciklusa (održavanje i modifikacija). Ovim je sistem strukturiran u tri logičke cjeline razdvajajući podatke i njihovu prezentaciju. Model upravlja podacima, a implementiran je kao aktivni. Posjeduje mehanizam za ažuriranje i obavještavanje pogleda i kontrolera o nastaloj izmjeni. Ovaj mehanizam je omogućen primjenom obrasca Posmatrač (Observer).

# Smjernice za mehanizme

*queri* je softverski alat koji je projektovan kao alat sa jednostavnim i intuitivnim mehanizmima za korištenje tj. modelovanje baza podataka. Razvojni tim, pri izradi alata, može koristiti standardne metode i mehanizme implementirane kroz Java programski jezik, tj. Java Swing. Upotreba alata neće predstavljati problem za korisnike sa osnovnim poznavanjem rada na računaru. Naravno, dostupno je korisničko uputstvo i online pomoć.