UNIVERZA NA PRIMORSKEM

Fakulteta za matematiko, naravoslovje in informacijske tehnologije

Računalništvo – 1. stopnja

ELVIS SUKANOVIĆ

RAZVOJ RAČUNALNIŠKE APLIKACIJE ZA IZRAČUN IN PREDSTAVITEV STROŠKOV PORABE RAZLIČNIH ENERGENTOV SKOZI VSE FAZE RAZVOJA APLIKACIJ

ZAKLJUČNA PROJEKTNA NALOGA

Mentor: prof. dr. CENE BAVEC

Ajdovščina, december 2012

UNIVERSITY OF PRIMORSKA

Faculty of Mathematics, Narutal Sciences and Information Technologies

Computer Science – 1st degree

ELVIS SUKANOVIĆ

DEVELOPING APPLICATION FOR THE CALCULATION AND PRESENTATION OF THE DIFFERENT ENERGY CONSUMPTION COSTS THROUGH ALL PHASES OF THE APPLICATION DEVELOPMENT

FINAL PROJECT PAPER

Mentor: CENE BAVEC, Ph.D

Ajdovščina, December 2012

ZAHVALA

Rad bi se zahvalil doc. dr. Cenetu Bavcu za mentorstvo ter strokovno pomoč in svetovanje, ki mi jih je nudil pri izdelavi zaključne naloge.

Zahvalil bi se podjetju GEN-I, d. o. o., ker mi je omogočilo izdelavo in pomoč pri zaključni nalogi, še posebej ge. Adrijani Juren.

Največja zahvala gre moji družini in punci, ki so me podpirali in mi stali ob strani vsa študijska leta.

POVZETEK

Zaključna naloga se osredotoča na predstavitev razvoja računalniške aplikacije za izračun in predstavitev stroškov porabe električne energije in plina skozi vse faze razvoja aplikacij. Informacijski sistem, ki je potreben za tekoče delovanje same aplikacije, je implementiran z uporabo večih razvijalskih orodij in ogrodij ter je sestavljen iz zaključenih podsistemov in podatkovne baze. Celoten sistem je podrobno dokumentiran v zaključnem delu.

V nalogi najprej predstavimo ozadje razvoja aplikacij, torej podrobno opišemo vse faze življenjskega cikla razvoja aplikacij ter splošne modele razvoja le teh. V naslednjem poglavju opišemo analizo vseh uporabniških in sistemskih zahtev ter načrtovanje sistema na podlagi analize, torej omenjeno teorijo apliciramo na zastavljeni cilj. V tem delu poleg načrta razvoja predstavimo izbrane tehnologije in orodja s katerimi smo sistem razvili in izbiro skupka tehnologije primerno utemeljimo ter zgradimo okvirno podatkovno bazo. Po fazi načrtovanja in izbire orodij je opisana še dejanska implementacija aplikacije in podatkovne baze ter ostalih celot potrebnih za delovanje aplikacije.

Zaradi frekvence spreminjanja podatkov, potrebnih za izračune stroškov in razdrobljenosti trenutnih rešitev za te namene v podjetju je ta informacijski sistem zelo primerna rešitev za poenotenje delovanja in razbremenitev dela končnih uporabnikov.

Ključne besede

Življenjski cikel razvoja aplikacij, aplikacija, kalkulator, uporabniški vmesnik, Kohana, MooTools. električna energija, zemeljski plin

ABSTRACT

The final project paper focuses on presentation of application development throughout all phases of system development life cycle. Main function of developed application is calculating and presentation of consumption costs for electrical energy and natural gas. Whole information system which is necessary for the smooth functioning of the application itself is implemented by using multiple development tools and frameworks. It consists of closed subsystems and database, which are well described in final work.

Firstly, the application development background is presented in the work, therefore all phases of the system development life cycle and general development models are described in detail. Secondly, the analysis of all user and system requirements are described and the whole system design is created on the basis of the analysis. Therefore the described theory is applied to the development of the final product. In addition to designing system in this phase it was decided which technologies to use for development and justified our selection of tools, furthermore the database frame was designed during this phase. After whole planning and selection of tools was justified, the application, subsystems and database were implemented and described in the next chapter. Finally the real implementation of subsystems, databases and user interface applications are composed to final system and whole implementation is described in detail.

Due to the frequency of changing data required by the calculation of costs and fragmentation of the current solution for this purpose in the company, this information system is very suitable solution for unifying operations and relieving work of end users.

Keywords

System development life cycle, application, calculator, user interface, Kohana, MooTools, electric energy, gas

KAZALO

KAZALO SLIK

KAZALO TABEL

SEZNAM KRATIC

SDLC	System development life cycle (Življenjski cikel razvoja aplikacij)
PHP	PHP: Hypertext Preprocessor (Skriptni programski jezik PHP)
HTML	HyperText Markup Language (Označevalni jezik HTML)
CSS	Cascading Style Sheets (Kaskadne stilske podloge)
T-SQL	Transact Structured Query language (Strukturiran poizvedovalni jezik)
RAD	Rapid application development (Hiter razvoj aplikacij)
MVC	Model-view-controller (Model-pogled-krmilnik način programiranja)

1. UVOD

Vsaka organizacija se spopada z vsakodnevnimi problemi v informacijskih sistemih in posledično stremi k optimizaciji obstoječih oziroma razvoju novih. Uporabniki upravljajo informacijski sistem preko vmesnikov oziroma aplikacij, ki so posledično zelo pomemben del organizacije. V trenutnem delovnem okolju se je pojavila potreba in povpraševanje po novem informacijskem sistemu, ki bo pripomogel k lažjemu razumevanju in računanju stroškov porabe električne energije ter zemeljskega plina.

Z nalogo smo predstavili razvoj aplikacije za računanje in prikazovanje stroškov porabe električne energije in zemeljskega plina v podjetju GEN-I d.o.o. skozi vse faze razvoja aplikacij. Primarni cilji so bili definiranje vseh uporabniških in sistemskih zahtev ter predvidevanje in sprotno odpravljanje problemov, do katerih bi lahko prišlo ob neupoštevanju le-teh. Aplikacijo smo načrtovali v skladu z uporabniškimi željami ter omejili glede na sistemske zahteve in specifikacije.

V teoretičnem delu smo uporabili strokovno literaturo ter internetne vire, s pomočjo katerih smo predstavili faze razvoja življenjskega cikla. Obravnavali smo različne modele življenjskih ciklov in posledično izbrali enega, po katerem smo se orientirali pri razvoju aplikacije. Celotno pridobljeno znanje smo aplicirali na samo izdelavo aplikacije v tehničnem delu.

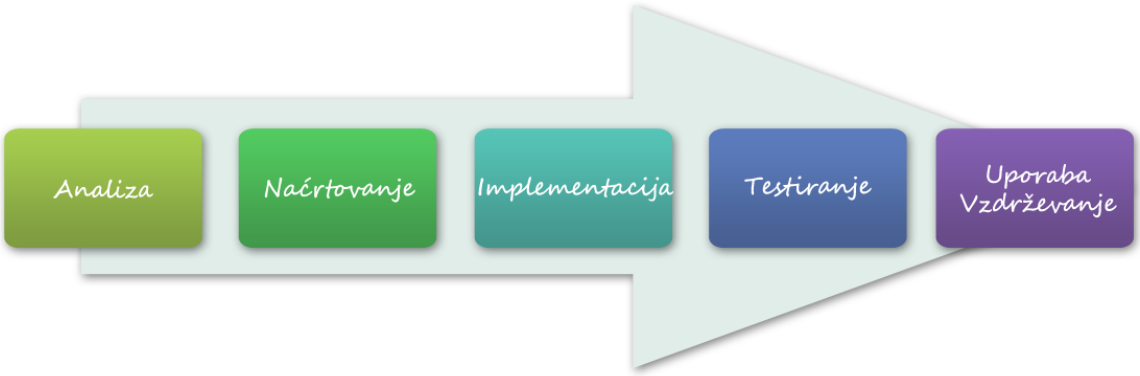
V tehničnem delu so na podlagi uporabniških želj in zahtev ter sistemskih specifikacij in omejitev izdelani diagrami, ki prikazujejo delovanje sistema, in relacijski diagram, ki predstavlja strukturo podatkovne baze. Na podlagi diagramov so implementirane programske definicije in rešitve, predstavljene s psevdokodo. Sistem je sestavljen iz dveh vmesnikov in podatkovne baze. Osrednja aplikacija je vmesnik za kalkulator, izdelan z uporabo markirnega jezika HTML in skriptnega jezika JavaScript. Za pravilno delovanje pa primarni vmesnik potrebuje še spletno storitev, implementirano z uporabo skriptnega jezika PHP, ki dostopa do

Microsoftove podatkovne baze. Vmesnik za vzdrževanje cen je realiziran s skriptnim jezikom PHP in sicer z uporabo MVC ogrodja Kohana Framework in JavaScript ogrodja MooTools. Tako implementirani sistem poenoti delovanje kalkulatorjev v organizaciji in olajša delo končnim uporabnikom.

2. ŽIVLJENJSKI CIKEL RAZVOJA APLIKACIJE

SDLC[1] se osredotoča na razvoj aplikacij skozi potrebne faze razvoja. Vsaka faza ima svoj pomen in namen ter zagotavlja povečano kvaliteto končne rešitve. Faze so podrobno opisane v poglavju 2.1.

2.1. Faze



Slika 1: Faze življenjskega cikla razvoja aplikacije [14]

2.1.1. Analiza zahtev in specifikacija sistema

V prvi fazi identificiramo vse uporabniške zahteve in omejitve, na katere moramo biti pazljivi pri kasnejšem načrtovanju aplikacije. Ko so zahteve in omejitve določene, pregledamo vse možnosti informacijskih in tehnoloških rešitev, ki lahko pridejo v poštev pri implementaciji, ter na podlagi ustreznosti določimo tehnologije in programska orodja, s katerimi bo realizirana končna aplikacija. Po določitvi skupka orodij za implementacijo preverimo, če se zahteve

[1] System development life cycle v prevodu življenjski cikel razvoja aplikacij

skladajo s programsko in strojno opremo, ki jo imamo na voljo, v nasprotnem primeru določimo drugi nabor orodij. Prepričamo se, da korist tako zasnovane rešitve sovpada z omejitvijo stroškov implementacije ter pravno zakonodajo. Rezultate študije izvedljivosti po končani analizi zberemo v poročilo, na podlagi katerega odgovorni sprejmejo odločitev o odobritvi ali zavrnitvi izvedbe aplikacije, torej je končni produkt prve faze izčrpno poročilo analize zahtev oziroma problema in specifikacija sistema. [14]

2.1.2. Načrtovanje

Ko je izvedba projekta odobrena, preidemo v fazo načrtovanja. V tej fazi, na podlagi predhodne analize, podrobno načrtujemo vse funkcije, operacije ter delovanje samega sistema in njegovih podsistemov. Celoten sistem iterativno razčlenimo na njegove konsistentne komponente, dokler niso dovolj majhne, da dobimo nabor manjših, zaključenih celot – modulov. Za vsak del sistema z algoritmi, funkcijami ter vhodnimi in izhodnimi podatki definiramo njegovo funkcionalnost in delovanje oziroma njegovo vlogo v celotnem sistemu, ter s pomočjo različnih diagramov in psevdokode vse podrobno dokumentiramo. S pomočjo relacijskega diagrama predstavimo celotno strukturo podatkov oziroma podatkovne baze. Končni produkti te faze so razbitost sistema na module in podsisteme, definicija strukture podatkov ter opisi algoritmov in funkcij.[4]

2.1.3. Implementacija

Po fazi načrtovanja sledi faza implementacije. Najprej zagotovimo, da so vsa potrebna programska orodja in okolja pripravljena in pravilno nastavljena za uporabo, torej pripravimo potrebne programske vmesnike in vmesnike za urejanje podatkovne baze, nato pa s pomočjo le-teh celoten načrt prelijemo v delujoč sistem. V tej fazi implementiramo vse dokumentirane algoritme in funkcije, kreiramo in inicializiramo podatkovno bazo ter oblikujemo uporabniške vmesnike namenjene za interakcijo s sistemom. Med programiranjem sproti popravljamo in čistimo napake ter optimiziramo delovanje posameznih modulov, specificiranih v fazi

načrtovanja. Med samo implementacijo sistem posledično tudi testiramo, tako programerski kot vsebinski del. Končni produkt te faze je prototip oziroma verzija sistema z vključno vsemi podsistemi, pripravljena na vsebinsko testiranje.[19]

2.1.4. Testiranje

Po končani fazi testiranja je sistem pripravljen za uporabo končnim uporabnikom in samo trženje aplikacije, zato je ta faza med najbolj kritičnimi. Pri testiranju moramo predvideti vse možne situacije in jih temeljito testirati, ker so posledice napak v sistemu zelo neprijetne. Sistem se mora odzivati natanko tako kot je bilo to predvideno oziroma načrtovano, v nasprotnem primeru se razvoj vrne na eno izmed prejšnjih faz, kjer se razišče in popravi razlog neprimernega odziva sistema. Ker sta fazi implementacije in testiranja tesno povezani tako v tej fazi kot v predhodni, koderji sproti odpravljajo hrošče in napake. Sodelovanje končnih uporabnikov in IT-strokovnjakov je ključnega pomena pri določanju, ali razviti sistem izpolnjuje predvidene zahteve in v kolikšni meri je sistem dejansko uporaben. V fazo testiranja pa spada tudi premik starih podatkov v novo podatkovno bazo, če le-ti obstajajo, in usposabljanje končnih uporabnikov za uporabo novega sistema. Faza je zaključena, ko se IT-strokovnjaki in končni uporniki strinjajo, da sistem izpolnjuje vse določene zahteve in pričakovanja. Končna produkta te faze sta prototip oziroma dodelana verzija aplikacije, pripravljene na uporabo končnih uporabnikov, in trženje.[20]

2.1.5. Prenos v ciljno okolje, uporaba in vzdrževanje

Kot že sam naslov pove v zadnji fazi razvoja aplikacijo prenesemo v ciljno okolje, uporabnikom omogočimo uporabo aplikacije in sproti vzdržujemo in popravljamo obstoječ sistem. Napake in neželeno obnašanje aplikacije sproti popravljamo in nadgrajujemo, za večje popravke oziroma spremembe delovanja sistema pa se vračamo v začetno fazo razvoja. Končna produkta te faze sta začetek uporabe aplikacije in njeno vzdrževanje.[4]

2.2. Modeli

Modeli življenjskega cikla aplikacije opisujejo faze kreiranja aplikacije in vrstni red, v katerem se faze odvijajo [1]. Na voljo je veliko različnih modelov in veliko podjetij uporablja svoj edinstven model [8], vendar so si vsi zelo podobni in sledijo istim smernicam oziroma uporabljajo iste vzorce.

2.2.1. Splošni model – linearno sekvenčni modeli[2]

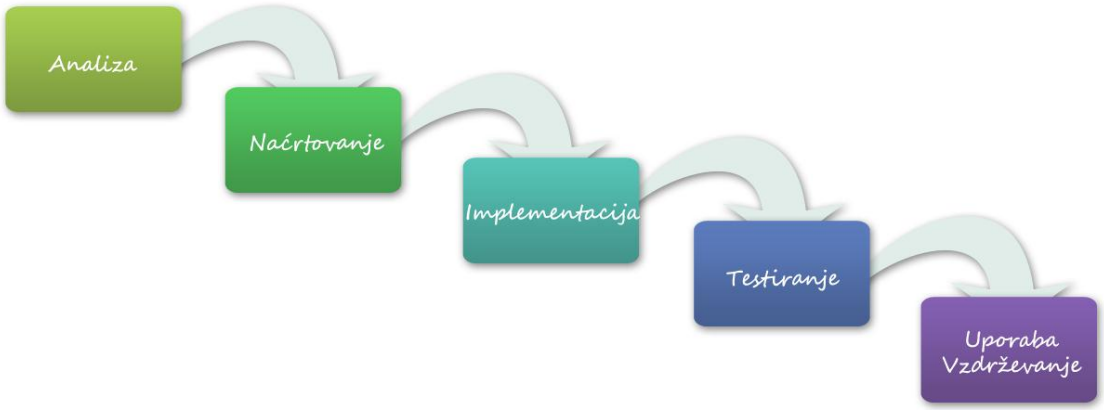
Splošni model smo definirali v poglavju 2.1., kjer smo podrobno opisali faze razvoja in njene aktivnosti ter vrstni red, v katerem si sledijo. Splošni življenjski cikel je linearno sekvenčni cikel, torej si faze sledijo zaporedno in se razvoj praviloma ne vrača na prejšnje že zaključene faze. Linearno sekvenčni življenjski cikli so najbolj uporabljani modeli za razvoj aplikacij, čeprav so najmanj prilagodljivi in najstarejši modeli. Prvi tak definirani model je bil slapovni model[3], med drugimi pa je predstavnik linearno sekvenčnih modelov tudi model V.

2.2.1.1. Slapovni model

Ta model porabi veliko časa v začetnih fazah, kar preprečuje napake v nadaljnjih in s tem poveča verjetnost uspešnosti in zmanjša morebitne stroške projekta [18].

Tak model uporabljamo, ko so zahteve zelo dobro poznane, ko je definicija produkta stabilna, ko poznamo vsa potrebna programska orodja in tehnologije, ko delamo nadgradnjo obstoječega produkta ali le premik obstoječega produkta na novo platformo.

[2] (angl.) Linear Sequential models
[3] (angl.) Waterfall model



Slika 2: Slapovni model [18]

Prednosti slapovnega modela so med drugimi tudi [15]:

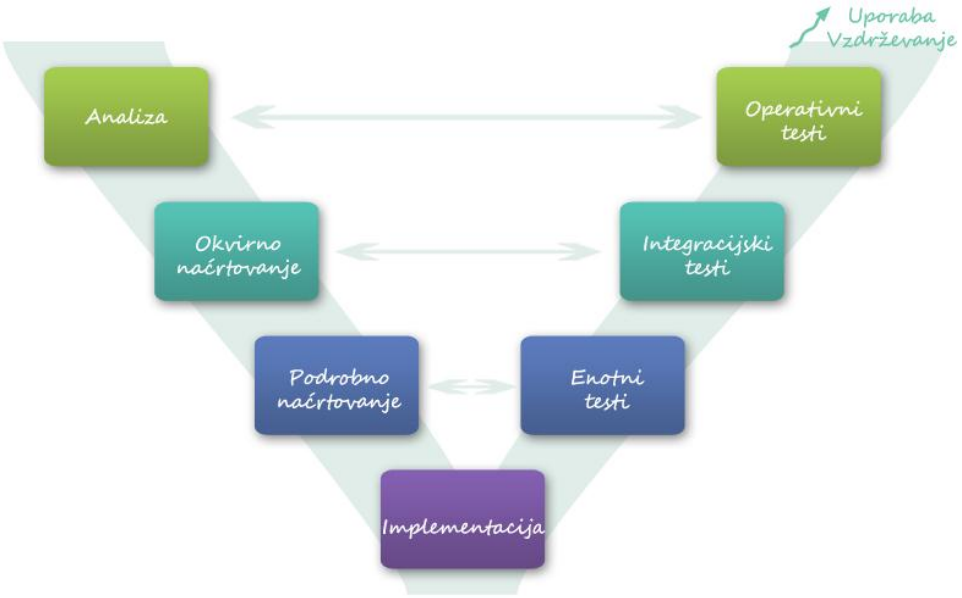
- preprosta in enostavna uporaba,
- vsaka faza ima končne rezultate in revizijski postopek,
- izvajanje je zaporedno, torej vsako fazo sproti obdelamo in zaključimo,
- vsaka faza je zelo dobro dokumentirana, torej imamo v najboljšem primeru zelo dobro definiran in dokumentiran projekt.

Slabosti slapovnega modela [15]:

- nimamo delujoče aplikacije do predzadnje oziroma zadnje faze razvoja, torej naročniku ne moremo prikazati napredka,
- dolgotrajen proces z veliko tveganja in negotovosti,
- slaba definiranost lahko prinese velike posledice,
- ni možnosti premikanja po fazah,
- slab model za kompleksne in objektno orientirane projekte, dolgotrajne projekte, za projekte, kjer so pričakovane spremembe.

2.2.1.2. Model V

Model V je nadgradnja slapovnega modela, torej ima podobne lastnosti s tem, da ima večji poudarek na testiranju, preverjanju in potrjevanju. Testni primeri so razviti že v zgodnjih fazah še preden izvedemo fazo implementacije in jih izvajamo vzporedno s fazami. V fazi analize izvedemo teste, ki preverijo možnost doseganja želenih in predvidenih ciljev. V fazi okvirnega načrtovanja testiramo, če programska orodja in tehnologije sovpadajo in imajo možnost skupnega delovanja ter če dosegajo želeno interakcijo s podatkovno bazo, medtem ko v podrobnem načrtovanju in implementaciji izvajamo enotne teste[4], torej testiramo programerske rešitve na nivoju kodiranja [3].



Slika 3: Model V [1]

Tak model uporabljamo, ko je ena najpomembnejših zahtev varnost oz. stabilnost sistema in je majhna verjetnost za velike spremembe (npr. bolnišnice, policija …), ko so vse zahteve in vsa programerska orodja poznana vnaprej.

Prednosti modela V [15]:

[4] (angl.) Unit tests

- večja možnost uspeha v primerjavi s slapovnim modelom, ker sproti testiramo celoten sistem od začetne faze,
- dober za manjše projekte, kjer je celotno testiranje izvedljivo,
- sproti odkrijemo napako in jo posledično odpravimo, torej zmanjšamo velike napake v sistemu.

Slabosti modela V [15]:

- veliko časa porabimo za izdelavo testov,
- za vsako manjšo spremembo moramo spreminjati, nadgrajevati teste.

2.2.2. Postopno razvijajoči se modeli[5]

Pri postopno razvijajočem se modelu razvoj začnemo še preden imamo definirane vse zahteve in specifikacije. Razvoj poteka rekurzivno čez vse faze razvoja in tako dobimo v vsaki iteraciji bolj definiran in dokončan produkt. Take modele posledično imenujemo tudi multi slapovni cikli[6]. Po končani prvi iteraciji že dobimo delujoč produkt, torej v primerjavi z linearnimi modeli lahko že zgodaj v razvoju uporabnikom in naročnikom pokažemo prototip aplikacije.

2.2.2.1. Prototipni model in hiter razvoj aplikacij[7]

Prototipni model in model hitrega razvoja aplikacij sta v mnogih pogledih zelo podobna. Ker sta oba postopno razvijajoča modela, hitro pridemo do prototipa aplikacije, ki ga skozi iteracije nadgrajujemo, popravljamo in dopolnjujemo.

Pri prototipnem modelu razvijalci kodirajo prototip že v fazi definiranja zahtev, na podlagi že znanih zahtev in predvidevanj. Ko je prototip izdelan, ga uporabniki testirajo in ocenijo ter razvijalcem vrnejo podrobne informacije o popravilih in spremembah, na podlagi katerih razvijalci nadaljujejo kodiranje. Ta postopek rekurzivno ponavljamo dokler uporabniki niso zadovoljni s končnim produktom.

[5] (angl.) Incremental/Iterative models [13]
[6] (angl.) Multi-Waterfall model
[7] (angl.) Prototype model and Rapid application development - RAD

Model hitrega razvoja temelji na ideji, da lahko dobre izdelke razvijamo hitreje in je kombinacija prototipnega in slapovnega modela. V samem procesu uporabniki, naročnik in razvijalci tesno sodelujejo pri definiranju zahtev, specifikaciji sistema in samem testiranju. Po vsaki iteraciji je produkt aplikacija z zahtevanimi popravki in nadgradnjami. Uporabniki aplikacijo ponovno testirajo oziroma uporabljajo dokler ne pride do napake ali novih zahtev, ki jih posredujejo razvijalcem in tako se postopek ponovi. Model je prvič dokumentiral Martin James (1991).

Take modele uporabljamo, ko so zahteve nestabilne, nepotrjene oziroma razmeroma znane in že v naprej vemo, da se bodo zahteve spreminjale oziroma dodajale skozi čas, pri razvoju uporabniških vmesnikov, pri razvoju kratko živečih in trenutno potrebnih aplikacij, ko so uporabniki zelo dostopni, saj so nujno potrebni pri celotnem razvoju, ko hitrost delovanja aplikacije ni nujno potrebna, saj je aplikacijo zelo težko optimizirati pri tako hitro spreminjajočem razvoju, ko sistem lahko razbijemo na module in ga implementiramo na tak način, da je vsak nadaljnji razvoj po neki iteraciji svoj modul, ki ga zelo enostavno dodamo v aplikacijo ipd. Tabela 1 prikazuje prednosti in slabosti obeh opisanih modelov.

	Prednost	Slabost
Hiter razvoj	V kratkem času že imamo prototip aplikacije, ki jo lahko uporabniki začnejo uporabljati, torej se vidi napredek razvoja	Uporablja se "hiter in grd" način programiranja, torej je implementirana rešitev običajno hitra in delujoča, ne pa tudi optimizirana, kar je na dolgi rok slaba rešitev, tak razvoj poveča površnost, število napak in zelo malo ali nič dokumentacije
	Krajši čas razvoja, višja produktivnost in krajši čas doseganja ciljev pomeni manjše stroške	Ker uporabniki hitro definirajo zahteve, se pričakuje tudi hiter razvoj (četudi l- ta potrebuje več časa), kar pomeni večji pritisk na razvijalce in posledično manj kvalitetne produkte

Začetek razvoja brez dokončno definiranih zahtev	Razvoj se začne veliko prej, kar posledično pomeni, da bo tudi sistem veliko prej dokončan	Obstaja verjetnost, da po določen času pridemo v "slepo ulico" oziroma do nesovpadanja neobhodnih zahtev ali nezmožnosti implementacije zahtev in celotno delo propade
Tesno sodelovanje razvijalcev in uporabnikov	Razvijalci so seznanjeni z vsebinskim delom, kar razvijalcem olajša delo, posledično je končni produkt bolj natančen in dovršen	V primeru ,ko trenutni razvoj ni edino delo razvijalcev in uporabnikov (zelo velika verjetnost) je konstantno prekinjanje dela zelo moteče, torej povečuje verjetnost napak in povzroča izgubo časa med čakanjem na naslednji korak
Velika fleksibilnost zahtev in razvoja	Uporabniki lahko sproti menjajo izgled in delovanje aplikacije, ker sproti naletijo na težave in definirajo napačno delovanje	Ker se zahteve sproti spreminjajo, je to zelo moteče za razvijalce in je velikokrat že implementirana rešitev zavržena, med drugim je to dejstvo v nasprotju s pomembnim pravilom programskega inženirstva, da se zahteve med samim razvojem ne sme spreminjati
	Sistem se razvija dinamično in omogoča lažjo nadgrajevanje in dodelavo	Projekti se pri takem načinu redko kdaj končajo, torej ne pride do zadnje faze "prenos v okolje, uporaba in vzdrževanje", ampak se sistem ves čas načrtuje in nadgrajuje, posledično pa sistem postane prevelik in nestabilen ter težko vzdržljiv

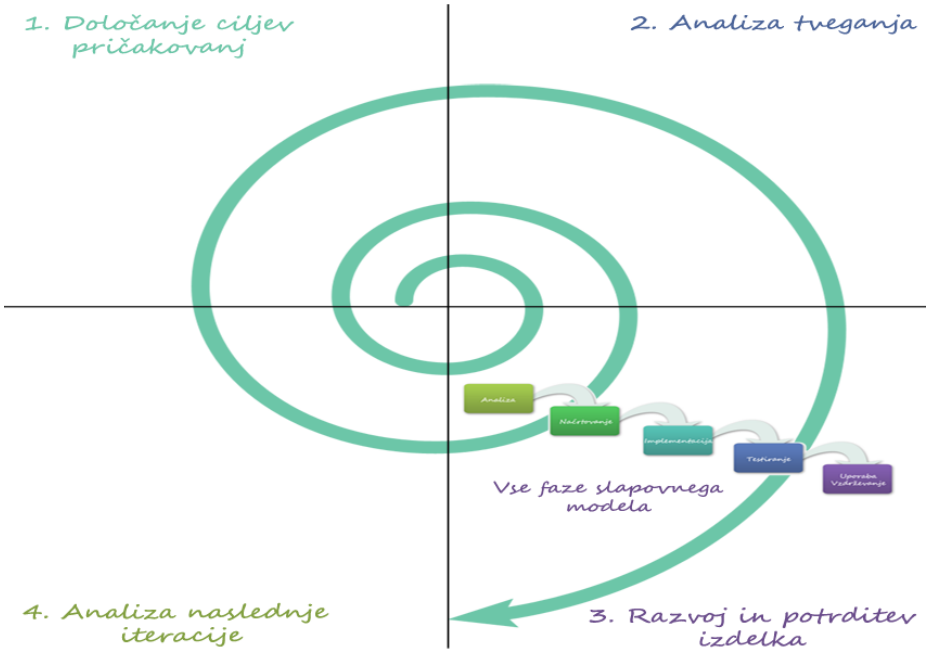
Tabela 1: Prednosti in slabosti prototipnega modela in modela hitrega razvoja [15]

2.2.2.2. Spiralni model

Spiralni model je kombinacija prototipnega in slapovnega modela z iterativnim razvojem vendar za razliko od modela hitrega razvoja ne temelji na hitrosti, ampak se osredotoča na temeljito planiranje in definiranje razvoja ter na analizo in obvladovanje tveganj. Prepoznavanje glavnih tveganj in omejevanje oziroma odprava tveganj ohranja proces razvoja programske opreme pod nadzorom. Zaradi dodatne analize tveganj in dodatnega planiranja je razvoj, ki temelji na spiralnem modelu, počasnejši vendar bolj zanesljiv od ostalih in je zato

primeren za velike, kompliciranе in dražje sisteme. Kot je značilno za postopno razvijajoče modele je produkt z vsako iteracijo bolj dovršen oziroma je nadgradnja trenutnega sistema [2]. Spiralni model je prvi predstavil Barry Boehm (1986).

Spiralni model uporabljamo, ko je vrednotenje stroškov in tveganj smiselno, ko razvijamo zelo drage ter visoko tvegane aplikacije, ko so zahteve zelo kompleksne, ko uporabniki oziroma naročniki niso prepričani v svoje zahteve, ko pričakujemo velike spremembe v času razvoja, ko začnemo novo linijo produktov ipd.



Slika 4: Spiralni model [2]

Prednosti spiralnega modela [15]:

- zagotavlja spoznanja nesprejemljivih tveganj v zgodnjih fazah,
- ker imamo vpogled v visoko rizične predele sistema, lahko celotno planiranje priredimo le-tem,
- zelo dobro ocenjeni končni stroški razvoja,
- povečana verjetnost uspešnosti projekta zaradi analize tveganj.

Slabosti spiralnega modela [15]:

- čas potreben za analizo tveganj je prevelik v določenih primerih, oziroma model je primeren samo za velike, rizične projekte.
- model je kompleksen,
- za dodatno analizo potrebujemo dodatne strokovnjake,
- v primeru, ko je to edini aktiven projekt lahko razvijalci ostanejo brez dela v ne razvijalskih fazah.

Poleg naštetih ima spiralni model tudi nekatere prednosti in slabosti prototipnega modela.

Ker ima vsak model svoje prednosti in slabosti, ni modela, ki je popolna izbira za vsako aplikacijo, torej model življenjskega cikla izberemo na podlagi količine poznanih zahtev in verjetnosti njihovega spreminjanja, na podlagi zahtev samih, na podlagi časa in ekonomskih sredstev, ki jih imamo na voljo, na podlagi tehnologij in orodij, med katerimi lahko izbiramo ipd.

Za razvoj našega sistema smo uporabljali prototipni model oziroma model hitrega razvoja. Ker zahteve niso bile dokončno definirane pred začetkom razvoja so uporabniki na podlagi razvitega prototipa dodatno definirali zahteve, ter pripomogli k testiranju same aplikacije, torej je bil razvoj tesno povezan s sodelovanjem razvijalcev in uporabnikov.

3. NAČRTOVANJE APLIKACIJE

V tem poglavju sta združeni prvi dve fazi razvoja, in sicer analiza zahtev in načrtovanje sistema. V fazi analize podrobno pregledamo vse uporabniške zahteve in želje, podobno preverimo še za sistemske omejitve in na podlagi vseh ugotovitev začnemo načrtovati sam sistem. Na podlagi analize, izmed vseh razvijalskih orodij in okolij, ki so nam na voljo, izberemo najbolj primerna, glede na sovpadanje z zahtevanim. Samo izbiro orodij predstavimo in utemeljimo. V fazi načrtovanja izberemo tudi primerno bazo in jo predstavimo s podatkovnim diagramom, samo delovanje celotnega sistema in podsistemov pa predstavimo z diagramom stanj, diagramom zaporedij in diagramom primera uporabe. Diagrami v veliki meri prikažejo želeno delovanje sistema in s tem razvijalcem olajšajo delo.

3.1. Uporabniške in sistemske zahteve

3.1.1. Uporabniške zahteve

Temeljne uporabniške zahteve želenega sistema so naslednje:

- Celoten sistem je namenjen uporabnikom zaposlenim v podjetju [5], kot tudi zunanjim sodelavcem. Potrebujemo dva vmesnika, enega za vzdrževanje cen in šifrantov, drugega za kalkulacije in prikaz okvirnih stroškov za kombinacijo izbranih kriterijev in vnesenih podatkov. Vmesnik za kalkulator je namenjen tako notranjim kot zunanjim uporabnikom, medtem ko do vmesnika za urejanje cen lahko dostopajo samo notranji uporabniki in sistemski vzdrževalci.
- Vmesnik za vzdrževanje cen ima dva glavna modula, in sicer modul »*Ceniki*« in modul »*Šifranti*«, ter modul za nastavitve, ki je dostopen samo vzdrževalcem. Na modulu

»*Ceniki*« uporabniki urejajo cenike in časovno usklajujejo cene dobrin, na modulu »*Šifranti*« pa upravljajo s šifranti.

- Sistem mora biti zelo dinamičen zaradi dodajanja artiklov in produktov, ki se navezujejo na primerne kalkulatorje, torej ko se pojavi potreba po novem kalkulatorju, uporabniki preko vmesnika za cenike dodajo primerne podatke, ki se aplicirajo v delujoč kalkulator na vmesniku za kalkulacije. S tem omogočimo delovanje sistema, ki je v majhni meri odvisen od vzdrževalcev in ga v celoti vzdržujejo uporabniki sami.
- Vmesnik za kalkulator mora biti pregleden in enostaven za uporabo, ker ga uporablja veliko različnih uporabnikov, ki niso deležni razlage delovanja samega sistema, ter lahko dostopen in hitro pripravljen za uporabo takoj po prejetju.

3.1.2. Sistemske zahteve

Sistemske zahteve in omejitve [6], na katere moramo biti pozorni, so:

- V organizaciji [5] so podatkovne baze vidne le v notranjem omrežju, zatorej zunanji sodelavci nimajo dostopa do podatkovnih baz in podatkov od katerih je aplikacija odvisna. Podatki, ki jih aplikacija potrebuje za natančno preračunavanje, se nepredvidljivo spreminjajo in morajo biti vedno posodobljeni za pravilno delovanje. Posledično se pojavi potreba po spletni storitvi, ki lahko dostopa do podatkovne baze in je dostopna vmesniku za kalkulacije.
- Vmesnik za kalkulator nima namestitve in mora biti lahko dostopen, potrebuje pa lokalno podatkovno bazo, v kateri so zadnji podatki iz krovne podatkovne baze.
- Vmesnik za kalkulator mora biti podprt na vseh operacijskih sistemih.
- Vmesnik za cenike je aplikacija na strežniku, dostopna le v notranjem omrežju in njegov glavni cilj je interakcija s podatkovno bazo.
- Oba vmesnika imata identifikacijo uporabnika.

Po analizi uporabniških in sistemskih zahtev lahko začnemo s fazo načrtovanja in na podlagi analize pridemo do zaključkov, ki se jih držimo pri načrtovanju in kasneje pri implementaciji. Preverimo še vse možnosti informacijskih in tehnoloških rešitev in ugotovimo, da so nam na voljo naslednja programska orodja in okolja ter tehnologije:

- Microsoft Excel (Visual Basic)
- PHP (Kohana Framework)
- HTML, CSS, JavaScript (MooTools)
- Matlab
- Microsoft podatkovna baza (Microsoft SQL Server)

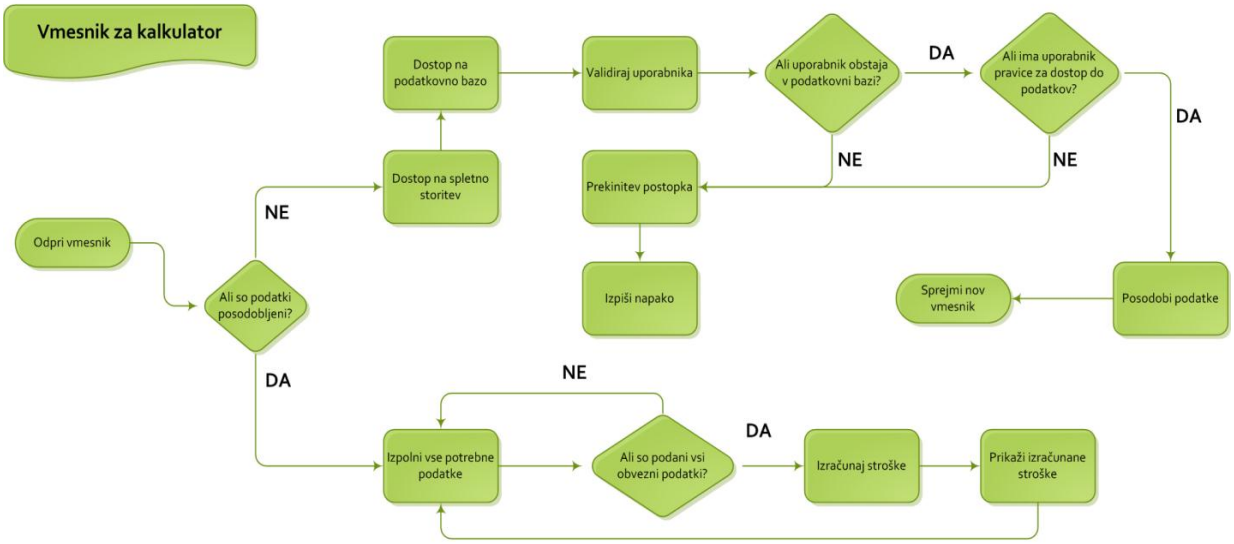
Temeljni sklepi, do katerih pridemo, so naslednji:

- Celoten sistem je sestavljen iz dveh vmesnikov, spletne storitve in podatkovne baze, od katerih delujejo vmesnik za cenike, spletna storitev in podatkovna baza na strežniku, medtem ko deluje vmesnik za kalkulacije lokalno na računalniku.
- Zahteve, kot so podpora vmesnika na vseh operacijskih sistemih, dostopnost in delovanje brez namestitve, nas pripeljejo do sklepa, da je celoten vmesnik implementiran v enem dokumentu, ki je dostopen preko spletne storitve. Kombinacija tehnologij, ki najbolj ustrezajo naštetim omejitvam je HTML, skriptni jezik JavaScript in CSS, podrobneje opisani v poglavju 3.2. Tak vmesnik nima namestitve, ampak deluje takoj po prenosu, podprt je na vseh operacijskih sistemih, ker njegovo vsebino predvajajo spletni brskalniki, ki so vsebovani v vseh operacijskih sistemih.
- Vmesnik za kalkulator potrebuje nabor podatkov, ki se shranijo lokalno na računalniku tako, da je aplikacija uporabna tudi, ko uporabnik nima dostopa do spleta. V ta namen lahko uporabimo spletni orodji piškotki[8] in lokalno skladišče[9]. Piškotki imajo omejitev velikosti podatkov 4 kB, kar je premalo za podatke potrebne za delovanje vmesnika, lokalno skladišče pa je podprto le na manjši skupini spletnih brskalnikov, kar v veliki

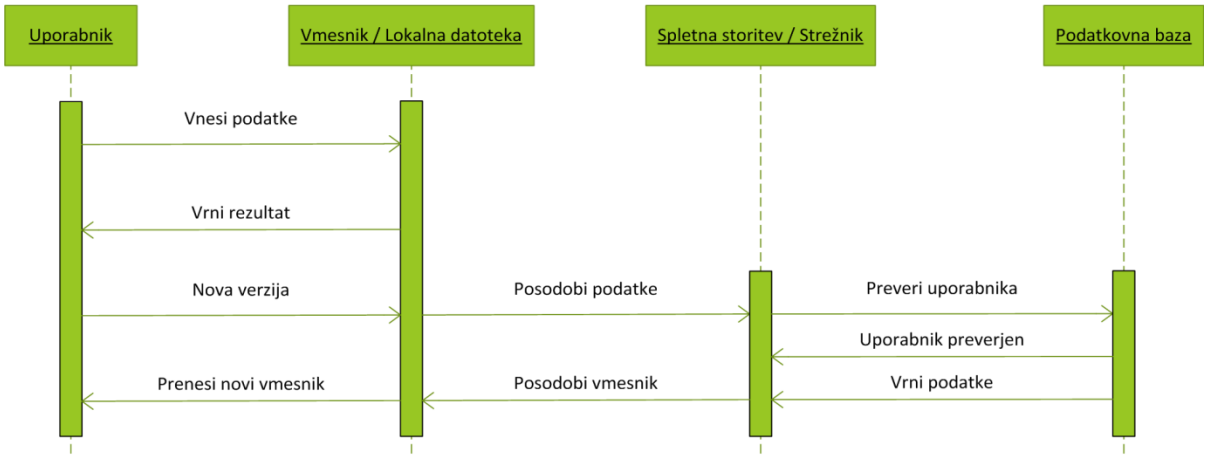
[8] (angl.) Cookies
[9] (angl.) Local Storage

meri omeji uporabo vmesnika. Podatke torej vkodiramo v samo rešitev vedno, ko uporabnik dostopa do dokumenta na spletno storitev na naslednji način: Uporabnik dostopa do spletne storitve s svojo unikatno kodo, ki jo spletna storitev dekodira in preveri obstoj uporabnika v podatkovni bazi. Ko se zaključi verifikacija uporabnika, spletna storitev prebere podatke iz tabel potrebnih za delovanje vmesnika in jih vkodira v dokument, ki je na strežniku. Po uspešni zaključitvi uporabniku ponudi nov dokument in s tem zadnjo različico aplikacije in osvežene podatke. Delovanje vmesnika je z diagrami prikazano v sliki 5, ki prikazuje diagram stanj vmesnika, ter v sliki 6, ki prikazuje diagram zaporedja.

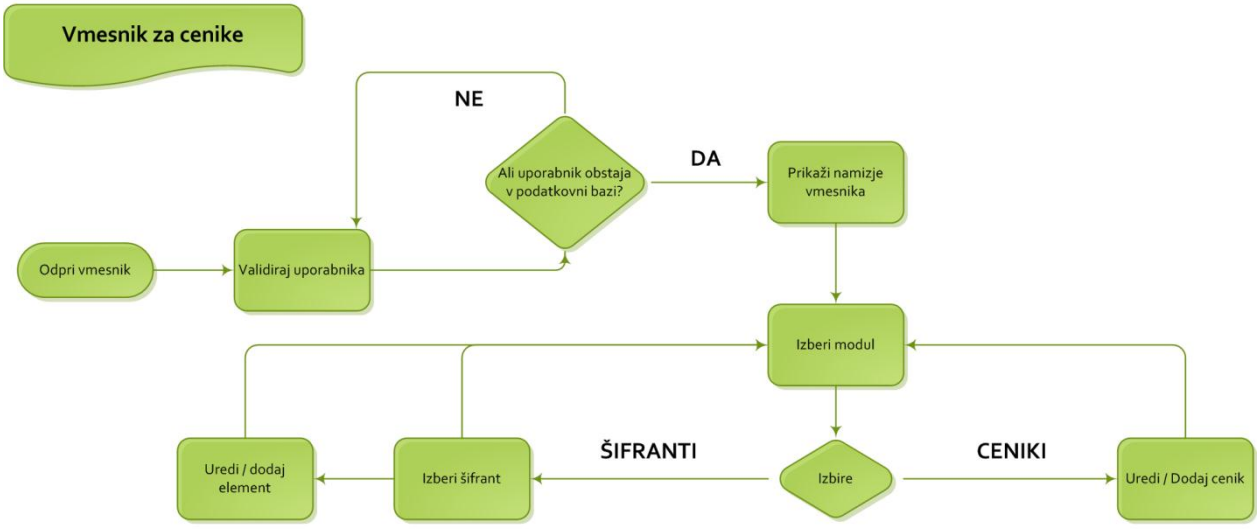
- Vmesnik za cenike je implementiran na strežniku v notranjem omrežju, torej lahko za kodiranje uporabimo skriptni jezik PHP z ogrodjem Kohana. Obe tehnologiji sta podrobneje opisani v 3.2. Uporabniki imajo na vmesniku dva modula, in sicer modul »*Šifranti*« ter modul »*Ceniki«*. Na modulu »*Šifranti*« uporabniki upravljajo preko vmesnika s podatkovno bazo in dodajajo ter urejajo šifrante, medtem ko je modul »*Ceniki«* zelo dinamičen in je uporabniški vmesnik za spreminjanje in vzdrževanje cenikov različnih kalkulatorjev. Samo delovanje vmesnika je prikazano z diagrami na slikah 7 in 8, ki predstavljata diagram stanj in diagram zaporedja.



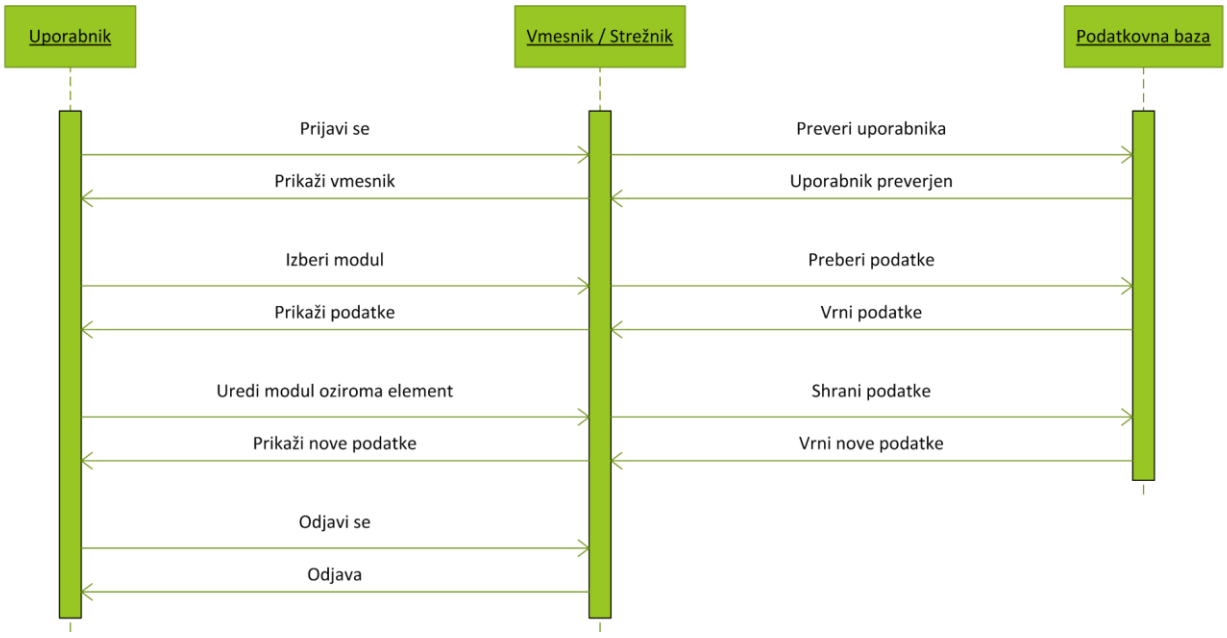
Slika 5: Diagram stanj vmesnika za kalkulator



Slika 6: Diagram zaporedja za kalkulator



Slika 7: Diagram stanj vmesnika za cenike



Slika 8: Diagram zaporedja za cenike

3.2. Programerske definicije

Orodja in jeziki, s katerimi je implementiran sistem, so naslednji:

- HTML
- CSS
- JavaScript (MooTools ogrodje)
- PHP (Kohana ogrodje)
- Podatkovna baza (Microsoft SQL Server)

Medtem ko lahko aplikacijo, zgrajeno s tehnologijami HTML, CSS in JavaScript, poganjamo na lokalnem računalniku v spletnem brskalniku, za zagon PHP-kode potrebujemo strežnik, ker se za razliko od prej naštetih izvaja na strežniku. Vmesnik za cenike se odvija na strežniku in uporablja vse zgoraj naštete tehnologije, medtem ko vmesnik za kalkulator poganjamo lokalno in je posledično programiran le s HTML, CSS in JavaScript.

3.2.1. HTML, CSS

HTML[10] je označevalni jezik za izdelavo spletnih strani. S HTML-jem postavimo na stran vse elemente, kot so napisi, vnosna polja, gumbi ipd., ter jim določimo poljubne atribute. HTML je torej osnovni in temeljni gradnik vsake spletne aplikacije. HTML-datoteke lahko razvijamo v poljubnem urejevalniku besedil kot je beležka ali emacs in prikazujemo v poljubnem spletnem brskalniku, tudi če nismo povezani na splet. Gradniki HTML-datoteke so HTML-elementi, ki so predstavljeni z začetno in končno značko. Primer:

```
<label id="helloLabel" class="greetingText">
       Hello
</label>
```

Zgornji zapis predstavlja HTML-element label, s katerim izpisujemo besedilo na vmesnik, ima dva atributa »id« in »class«, »id« služi za njegovo unikatno identifikacijo, z atributom »class« pa povemo, katere stile naj element uporablja. Stili so definirani v CSS-datoteki.

[10] (angl.) HyperText Markup Language [13]

CSS[11] je preprost slogovni jezik, s katerim oblikujemo prej opisane HTML-elemente. S CSS določimo, kako naj se elementi na strani prikažejo ter skrbimo za estetiko spletne strani. Elementom lahko določamo lastnosti kot so barva, oblika in velikost pisave, barva ozadja, širino in višino elementa, pozicijo, obrobe ipd. CSS lahko poleg estetike skrbi tudi za urejenost razvoja, saj na preprost način loči strukturo elementov in njihovo obliko. Elementu določamo stile preko prej omenjenih atributov »class« in »id«. »Class« je sklop večih slogov in lastnosti, ki jih preko atributov apliciramo na sam element. Isti atribut »class« lahko uporabljamo na večih elementih hkrati. Določanje oblik preko atributa »id« deluje enako kot preko atributa »class«, vendar pa lahko samo en element uporablja te lastnosti, kej je »id« unikaten, torej lahko na celotni strani le enkrat uporabimo isti »id«. CSS-datoteko izdelamo v poljubnem urejevalniku besedil ali primernem programerskem okolju. HTML in CSS sta vedno v paru in določata osnovno statično obliko vsake spletne strani.

3.2.2. JavaScript, ogrodje MooTools

JavaScript je objektni skriptni jezik, ki omogoča ustvarjanje interaktivnih spletnih strani. JavaScript v sodelovanju s HTML in CSS omogoča sprotno spreminjanje stilov in oblike HTML-elementov ter izvajanje funkcij, s čimer naredi spletne strani veliko bolj dinamične. Vso kodo JavaScript vgradimo v HTML kot svojo značko in z njo omogočimo izvajanje dinamičnih procesov, kot so odpiranje novih oken, preverjanje pravilnosti vnesenih podatkov, sprotne izračune ipd.

```
<script type="text/javascript">
       JavaScript koda …
</script>
```

Podobno kot HTML in CSS ga podpirajo vsi spletni brskalniki, vendar pa ga različni brskalniki različno interpretirajo. Za pravilno delovanje v vseh brskalnikih je posledično potrebno napisati več podobnih funkcij oziroma funkcije prirediti tako, da dosežemo povsod isto delovanje, kar pa lahko postane dolgotrajen in zapleten proces. Vsi razvijalci se običajno

[11] (angl.) Cascading Style Sheets

odločijo za izbiro enega izmed JavaScript ogrodij pri razvoju, saj le-ta olajšajo in pohitrijo delo. JavaScript ogrodja so skupek predefiniranih funkcij za delo z osnovnimi graditelji jezika ter za lažjo interakcijo s HTML-elementi. Nekatere knjižnice celo zagotavljajo pravilno in enotno delovanje JavaScript kode v vseh brskalnikih. Nekatere trenutno najbolj znane in uporabljane JavaScript knjižnico so Jquery, MooTools, DOJO, Prototype, JayData ipd. Oba vmesnika uporabljata knjižnico MooTools.

MooTools[12] je kompaktno, modularno objektno orientirano JavaScript ogrodje zasnovano za srednje do napredne razvijalce JavaScript.[11] Ogrodje je prvotni avtor Valerio Proietti izdal septembra 2006 [12]. MooTools se deli na dva krovna dela, in sicer na jedro in ostalo[13], katera vsebujeta funkcije za delo z razredi, osnovnimi in sestavljenimi graditelji JavaScript, kot so števila, besedila, polja, objekti, za lažjo in bolj izpopolnjeno interakcijo s HTML-elementi, za AJAX zahteve, delo z JSON knjižnico ipd. Ogrodje omogoča uporabniku veliko prednosti, kot so:

- razširljivo in modularno ogrodje omogoča uporabniku izbiro komponent in dodajanje obstoječih modulov,
- uporablja objektno orientirane prijeme,
- oplemenitena uporaba HTML-elementov,
- omogoča uporabo lokalnega skladišča,
- celotno ogrodje je brezplačno na voljo na spletu,
- dobra dokumentacija,
- zagotovljeno delovanje v brskalnikih Safari, Internet Explorer, Mozilla Firefox, Opera, Google Chrome.

[12] (angl.) My Object-Oriented Tools v prevodu moja objektno orientirana orodja
[13] (angl.) Core and More

3.2.3. PHP, ogrodje Kohana

PHP[14] je odprtokodni programski jezik, namenjen izdelavi dinamičnih spletnih strani z uporabo in obdelavo osnovnih programerskih tipov in objektov ter kreiranjem in uporabo funkcij, omogoča pa tudi enostavno delo s podatkovnimi bazami [16]. Programski jezik za delovanje potrebuje strežnik s primerno podporo, na katerem se izvaja programska koda in v sodelovanju s HTML in CSS generira spletno stran, med drugim pa se uporablja tudi za implementacijo spletnih storitev. Prvotni avtor PHP-ja je Rasmus Lerdorf [17], trenutno pa za razvoj skrbi podjetje The PHP Group [17]. PHP obstaja že od leta 1995 [17] in se ga še danes uporablja v veliki meri oziroma je en izmed najbolj razširjenih programskih jezikov. PHP-koda je vgrajena v HTML-datoteko s specifično značko:

```
<?php
        PHP koda …
?>
```

Podobno kot JavaScript tudi za PHP obstaja veliko ogrodij, katera olajšajo delo razvijalcem, nekatera med njimi so CakePHP, Zend Framework, CodeIgniter, Framewire, Kohana framework ipd.

Kohana je eleganten HMVC[15] okvir, ki deluje po PHP5 standardih in zagotavlja bogat nabor komponent za izgradnjo spletnih aplikacij [7]. Ogrodje razvijalcem omogoča MVC način programiranja, ki v veliki meri olajša delo ter naredi pregledno strukturo aplikacije. Struktura MVC je struktura, ki poskrbi, da so modeli, pogledi in krmilniki ločeni, ogrodje samo pa poskrbi za lažji dostop in povezanost posameznih datotek. Modeli predstavljajo razrede za delo s tabelami iz podatkovne baze, pogledi služijo za prikaz podatkov na vmesniku, medtem ko krmilnik povezuje modele in poglede ter omogoča implementacijo funkcij in algoritmov, po katerih aplikacija deluje.

[14] (angl.) PHP: Hypertext Preprocessor
[15] Hierarchical model–view–controller v prevodu Hierarhični model-pogled-krmilnik [13]

3.2.4. Microsoft SQL Server

Microsoft SQL Server je okolje z zelo širokim naborom orodij za obdelavo podatkov iz podatkovne baze oziroma je orodje za upravljanje z relacijskimi podatkovnimi bazami, torej temelji na relacijskem modelu in relacijski podatkovni bazi [10]. SLQ Server združuje hitrost in zanesljivost z varnostjo in prilagodljivostjo drugim okoljem, torej je zelo primeren vmesnik med uporabnikom in podatki oziroma med drugimi sistemi in podatki. Primarne naloge samega okolja so SQL poizvedbe, kot so iskanje s pogoji, sortiranje, filtriranje ter brisanje podatkov, medtem ko omogoča še veliko drugih funkcionalnosti, kot so razne analize podatkov, priprava poročil, beleženje zgodovine ukazom, izvoz celotne podatkovne baze ipd. SQL Server za izvajanje ukazov uporablja T-SQL oziroma Transact-SQL sintakso. Obstaja veliko različic okolja Microsoft SQL Server, od takih primernih za lokalne aplikacije z lokalno podatkovno bazo ter manjših spletnih aplikacij do aplikacij z milijoni uporabnikov in velikimi količinami podatkov.

3.3. Načrt aplikacije in podatkovne baze

3.3.1. Podatkovna baza

V fazi načrtovanja oblikujemo strukturo podatkovne baze na podlagi analize sistema in pravnih definicij. Izdelamo okvir podatkovne baze s primarnimi tabelami, ki jo sproti dopolnjujemo in spreminjamo. Osrednje tabele aplikacije so:

- tabela za kalkulatorje,
- tabela za dimenzije,
- tabela za labele,
- tabela za artikle,
- tabela za cenike,
- tabela za uporabnike.

Tabela za kalkulatorje, ki so definirani s kombinacijo dobrine, države in segmenta uporabnikov, vsebuje podatke o vseh možnih kombinacijah za izračun stroškov. Primer vnosa:

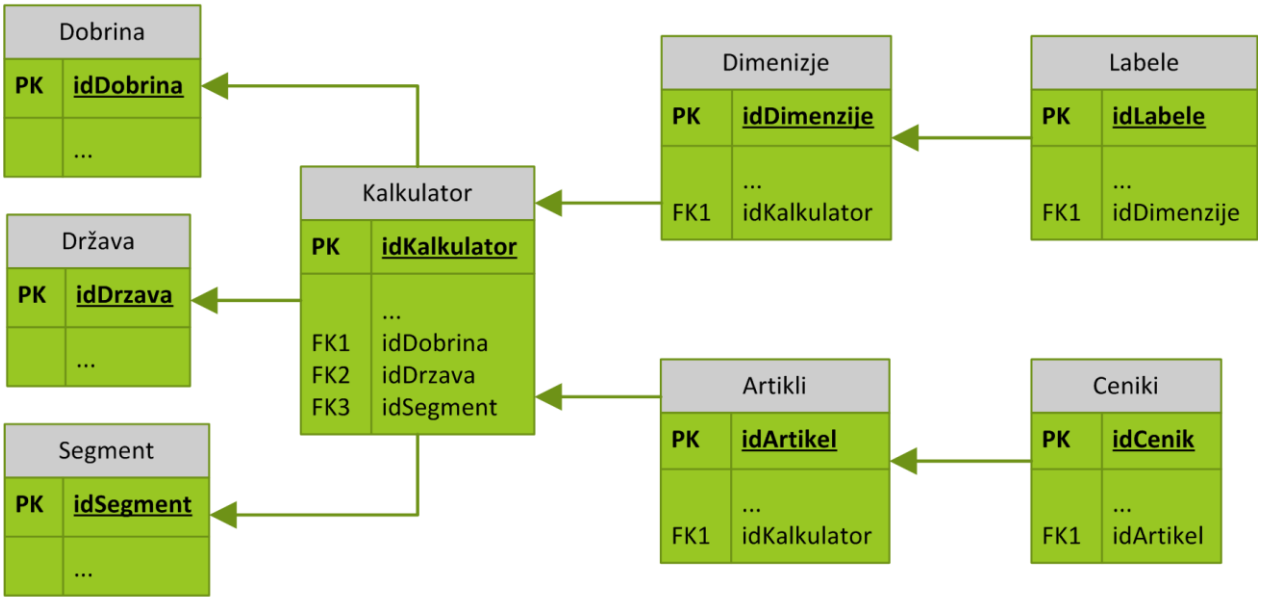
kalkulator za državo Slovenija, za dobrino elektrika in segment uporabnikov gospodinjstva, torej s tem kalkulatorjem računano stroške električne porabe gospodinjstev v Sloveniji.

Tabela za dimenzije predstavlja vse možnosti izbire in vnosnih polj za kalkulator, kjer v vnosna polja vnesemo neko vrednost, možnosti izbire pa so labele, shranjene v tabeli za labele.

Tabela za artikle vsebuje artikle, ki ponazarjajo postavko, ki se jo zaračunava porabnikom energije, odvisna pa je od kombinacije izbranih label in vnosnih podatkov na dimenzijah. Vsak artikel ima cenike določene prek kombinacij vseh dimenzij in label od katerih je odvisen artikel.

Tabela za uporabnike vsebuje splošne podatke o uporabnikih, njihove pravice in statuse.

Slika 9 predstavlja načrt podatkovne baze, medtem ko je celoten, končni podatkovni model predstavljen v poglavju 4.2.



Slika 9: Načrt podatkovne baze

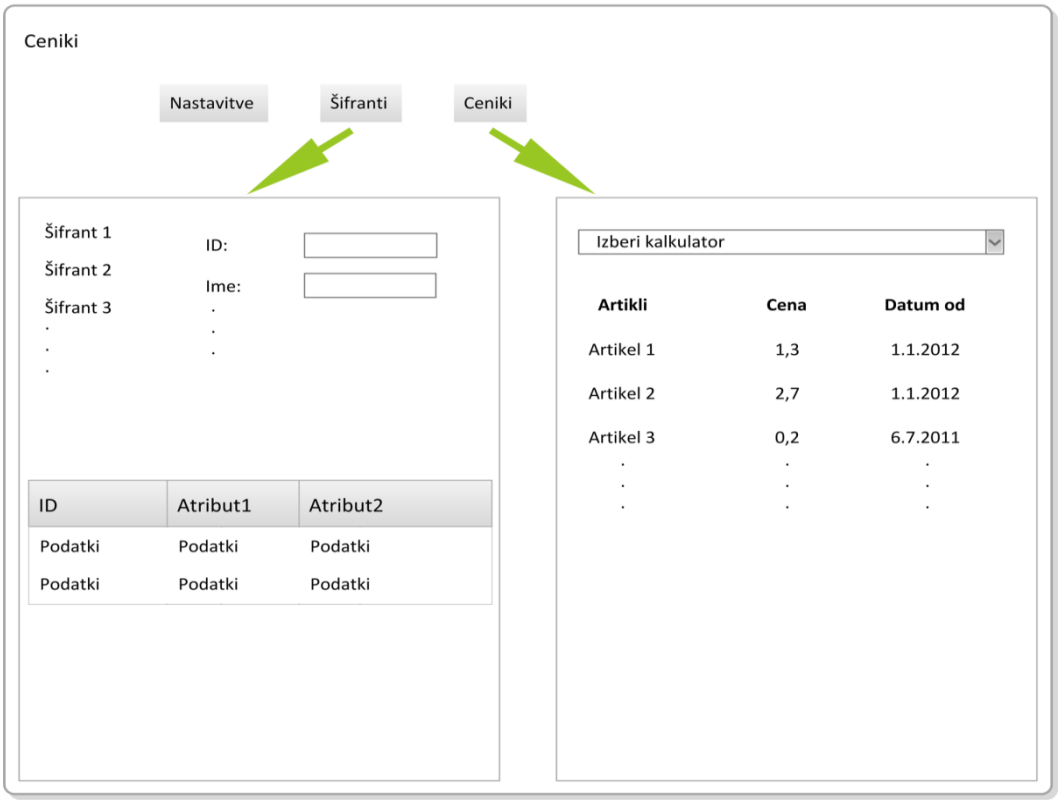
3.3.2. Vmesnik za kalkulator

Vmesnik za kalkulator uporabniku omogoča izračun stroškov porabe za izbrani kalkulator. Sistem je v celoti dinamičen in odvisen od izbire kalkulatorja, saj se na podlagi izbranega sestavi celotna forma in prikaže le pripadajoče dimenzije, artikle ter labele. Vmesnik razdelimo na dva dela, in sicer na selektivni del, ki zavzema zgornji del vmesnika (na sliki 10 označen s številko 1), ter na prikazovalni del, ki zaseda spodnji del vmesnika in je na sliki prikazan s številko 2. V selektivnemu delu izberemo in vnesemo vse zahtevane podatke, ki jih ob pritisku na gumb »Izračun« uporabimo za dostop do pripadajočih cen in izračunane vrednosti prikažemo v spodnjem delu vmesnika. Podrobnejši opis delovanja in implementacije je opisan v poglavju 4.1.

Slika 10: Načrt vmesnika za kalkulator

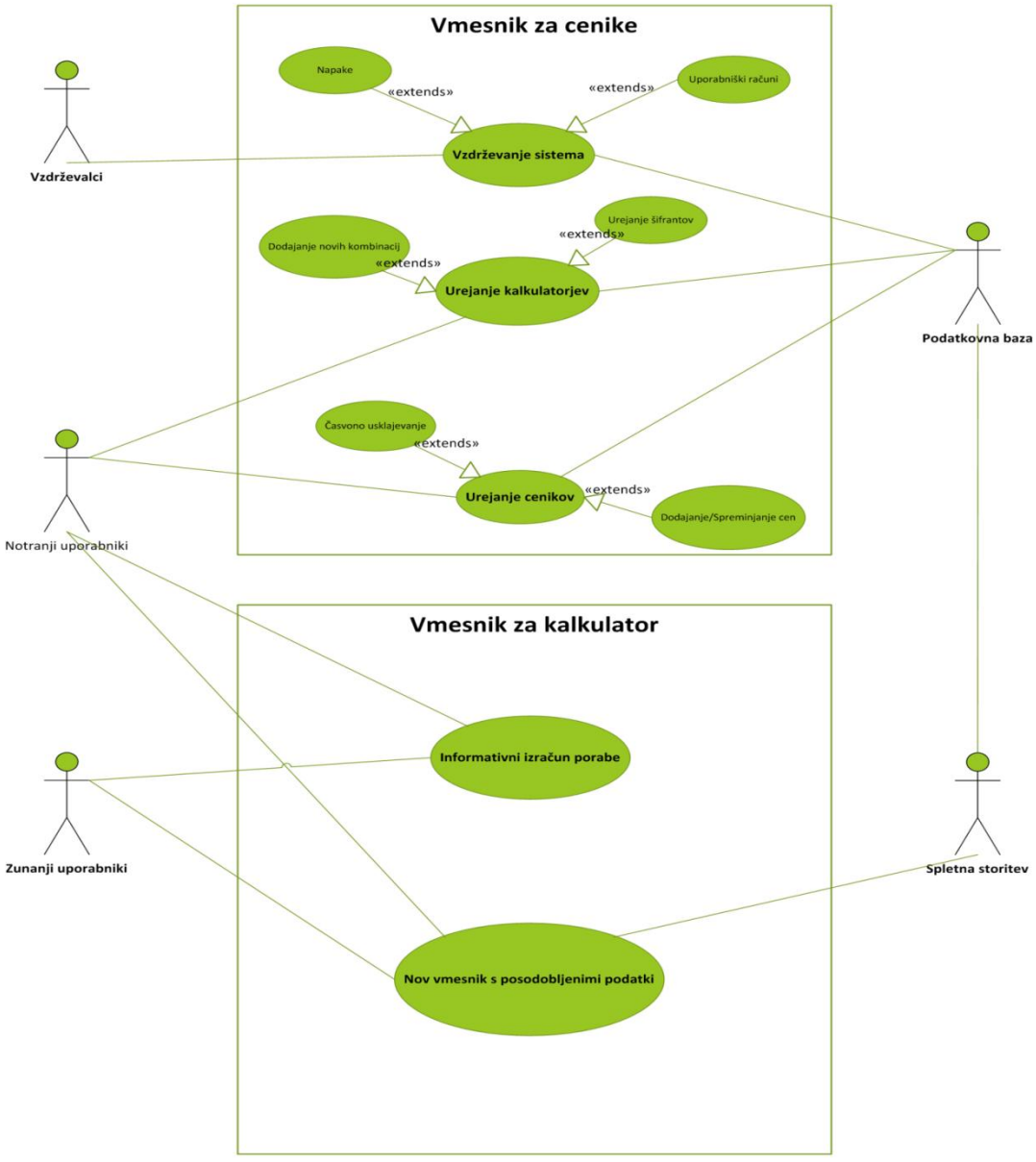
3.3.3. Vmesnik za cenike

Vmesnik za cenike je sestavljen iz treh modulov, in sicer modula Nastavitve, ki je na voljo samo vzdrževalcem ter modulov »*Šifranti*« in »*Ceniki*«. Modul »*Nastavitve*« vsebuje nastavitve za uporabnike, določanje vidljivosti modulov uporabnikom, beleženje zgodovine in napak ipd. Modul za šifrante vsebuje vse gradnike sistema, šifrante, in omogoča urejanje podatkov v tabelah. V modulu Šifranti dodajamo kalkulatorje ter njim pripadajoče dimenzije, labele in artikle. Modul za cenike deluje podobno dinamično kot vmesnik za kalkulator, torej na podlagi izbranega kalkulatorja spreminjamo celotno formo in prikažemo le pripadajoče artikle, katerim določimo cenike in datum od katerega začnejo vpisani ceniki veljati. Podrobnejši opis delovanja in implementacije je opisan v poglavju 4.3.



Slika 11: Načrt vmesnika za cenike

Slika 12 prikazuje diagram primera uporabe celotnega sistema in povezavo med podatkovno bazo, spletno storitvijo in obema vmesnikoma ter način, kako se celoten sistem uporablja.



Slika 12: Diagram primera uporabe

4. PROTOTIP SISTEMA

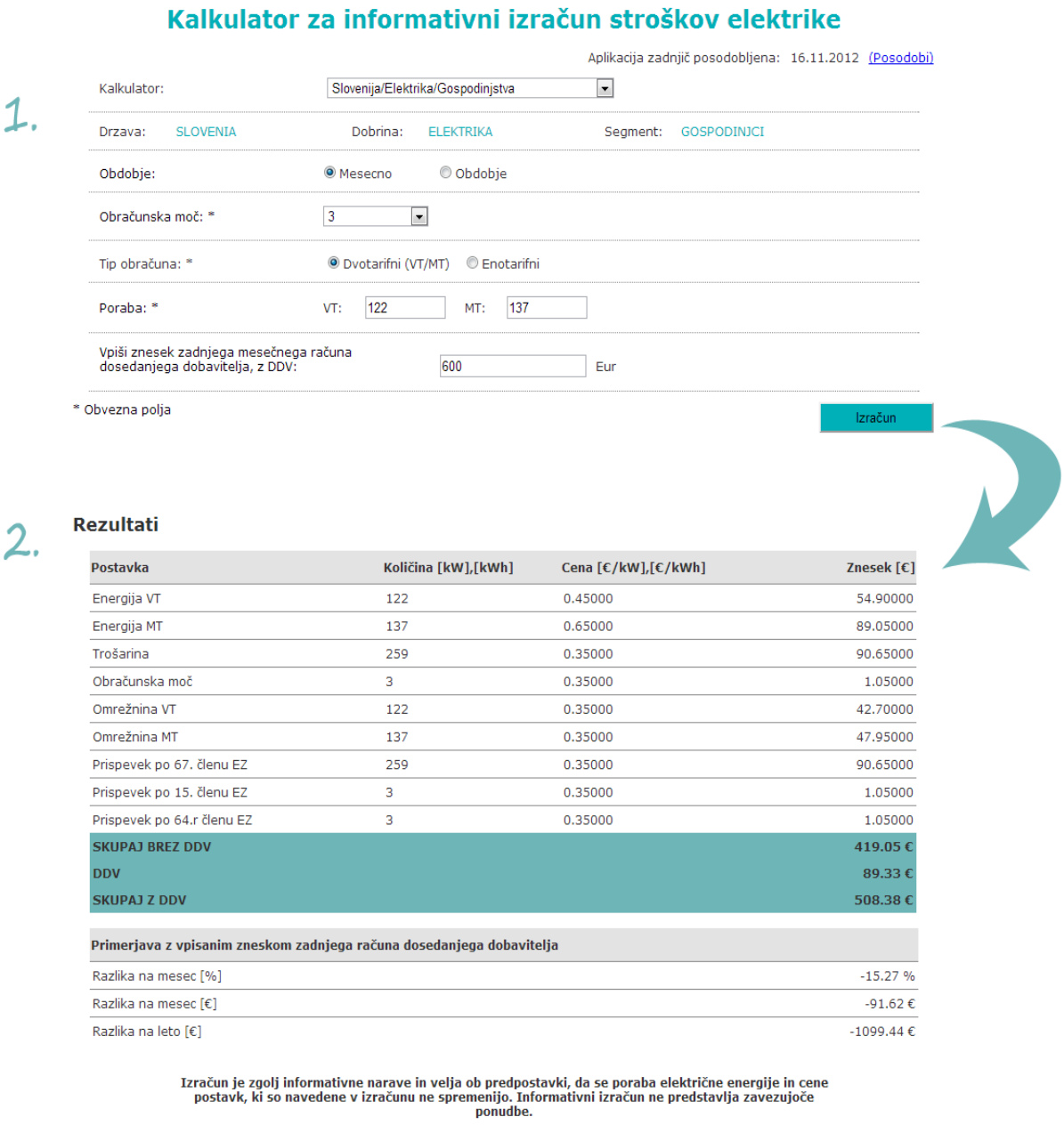
4.1. Vmesnik za kalkulator in spletna storitev

4.1.1. Vmesnik za kalkulator

Vmesnik za kalkulator je v celoti implementiran v eni datoteki z uporabo HTML, CSS in JavaScript tehnologij in posledično lahko vmesnik poganjamo na lokalnem računalniku v spletnem brskalniku kot spletno stran. Obe celoti ogrodja MooTools, Core in More, sta v enovrstični obliki vključeni v datoteko. Vmesnik je razdeljen na dva dela, zgornji del, ki vsebuje celotno formo in spodnji del, ki prikazuje rezultate izračuna. Na sliki 13 sta označena s številkama 1 za zgornji del in 2 za spodnji. Na zgornjem delu so prikazane vse dimenzije in labele vezane na posamezno dimenzijo. Večji del forme se generira dinamično na podlagi izbire kalkulatorja v prvem izbirnem meniju in posameznih izbir, vezanih na dimenzijo, manjši del pa je statičen. Spodnji del vmesnika se prikaže po kliku na gumb »Izračun« in se generira dinamično na podlagi kalkulatorja, izbranih label in vnesenih podatkov ter prikazuje vse postavke oziroma artikle, od katerih so odvisni stroški porabe, količino za določen artikel ter samo ceno za isti artikel. Pravilne količine in cene za artikle so določene po parametrih *acFormula* in *anBitMaska* v povezovalni tabeli za artikle, ki je podrobneje opisana v poglavju 4.2. Cene preberemo iz tabele za cenike preko polja *acFilter4Cenik*, same količine pa iz vpisanih podatkov in izbirnih menijev na zgornjem delu vmesnika.

Končni znesek dobimo po množenju količine s ceno artikla in sproti beležimo davek za določen artikel. V zbirniku prikažemo vsoto vseh zneskov, vsoto vseh davkov in končno vsoto vseh stroškov. Na koncu je še zbirnik za prikaz primerjave stroškov z zneskom zadnjega mesečnega računa dosedanjega dobavitelja.

Pravilno identifikacijo izbranih label in dimenzij zagotovimo z prenosom le teh preko HTML-atributov *»id«* in *»rel«*, katera premore vsak HTML-element in funkcij *handleSelect()*, *handleRadio()* in *handleInput()*.



Slika 13: Končna oblika vmesnika za kalkulator

4.1.2. Spletna storitev

Spletna storitev omogoča uporabnikom prenos zadnje posodobljene različice vmesnika, ki deluje po naslednjem principu:

Najprej preverimo uporabniško šifro, s katero uporabnik dostopa do storitve, in če ta ni prava, postopek prekinemo. Vsak uporabnik ima svojo šifro zapisano v podatkovni bazi, ki jo primerjamo z uporabniško šifro, s katero uporabnik poskuša dostopati na spletno storitev. Če obstaja v podatkovni bazi in ima pravice dostopanja do kalkulatorjev, poženemo metodi »*createNewCalculatorFile*« in »*returnNewCalculatorFile*«, v nasprotnem primeru na zaslon izpišemo, da prenos ni bil uspešen. Osrednji del programske kode spletne storitve:

```
if(checkUserHash()){
	createNewKalkulatorFile($kalkulatorFileName);
	returnNewKalkulatorFile($kalkulatorFileName);
} else {
	Echo 'Uporabnikška šifra ni veljavna. Preverite Vašo šifro ali kontaktirajte sistemske vzdrževalce.';
}
```

Programska koda funkcije *createNewKalkulatorFile*:

```
function createNewKalkulatorFile($kalkulatorFileName){
	$foundData	= false;
	$wholeHTML	= '';

	if ($calcFile = fopen($kalkulatorFileName,'r')) do {
		$line = fgets($calcFile);
		if($foundData){
			if(stringContains($line,'</script>')){
				$wholeHTML .= createNewDataStrings();
				$foundData = false;
			}
		}else{
			if(stringContains($line,'DATA_PART')) $foundData = true;
			else $wholeHTML .= $line;
		}
	} while (!feof($calcFile));

	fclose($calcFile);
	unlink($kalkulatorFileName);

	if ($calcFile = fopen($kalkulatorFileName,'w+')){
		fwrite($calcFile,$wholeHTML);
		fclose($calcFile);
	}
}
```

Vrstico za vrstico preberemo obstoječo datoteko in na mestu za podatke pokličemo funkcijo »*createNewDataStrings*«, ki iz določenih tabel podatkovne baze generira vse potrebne podatke za delovanje vmesnika in jih vrne nazaj, krovna funkcija pa poskrbi, da te podatke zapiše v datoteko. Ko je datoteka pripravljena, staro izbrišemo, novo pa ponudimo uporabniku za prenos preko funkcije »*returnNewKalkulatorFile*«.

Celotna spletna storitev je implementirana s programskim jezikom PHP.

4.2. Podatkovna baza

Pri izdelavi podatkovne baze se držimo nekaterih pravil poimenovanja, ker s tem ohranjamo urejenost podatkovne baze in lažjo preglednost. Vse tabele in atributi imajo določeno predpono, in sicer tabelam določamo predpono glede na sektor, v katerem jo uporabljamo, atributom pa določamo predpono glede na tip atributa v podatkovni bazi.

Predpone za tabele sestavimo tako, da na začetku zapišemo malo črko t, kar pomeni, da gre za tabelo, nato pa dodamo dvočrkovno oznako področja, kot so SK – skupna za vse, SI – šifrant, CC – ceniki ipd. Primeri poimenovanja tabel:

1. tSK_Uporabniki - skupna tabela za celoten sistem za uporanike
2. tSI_Artikli - tabela za modul šifranti za artikle
3. tCC_Ceniki - tabela za modul ceniki za cene

Predpone za atribute v tabeli pa sestavimo preprosto iz črke a in črke, ki označuje tip atributa, kot so n – število, c – besedilo, d – datum in b – logična vrednost[16]. Primeri poimenovanja atributov:

1. anIDDimenzije - število, ki predstavlja dimenzije ID
2. acNazivDimenzije - besedilo za naziv dimenzije

[16] da/ne oziroma 1/0

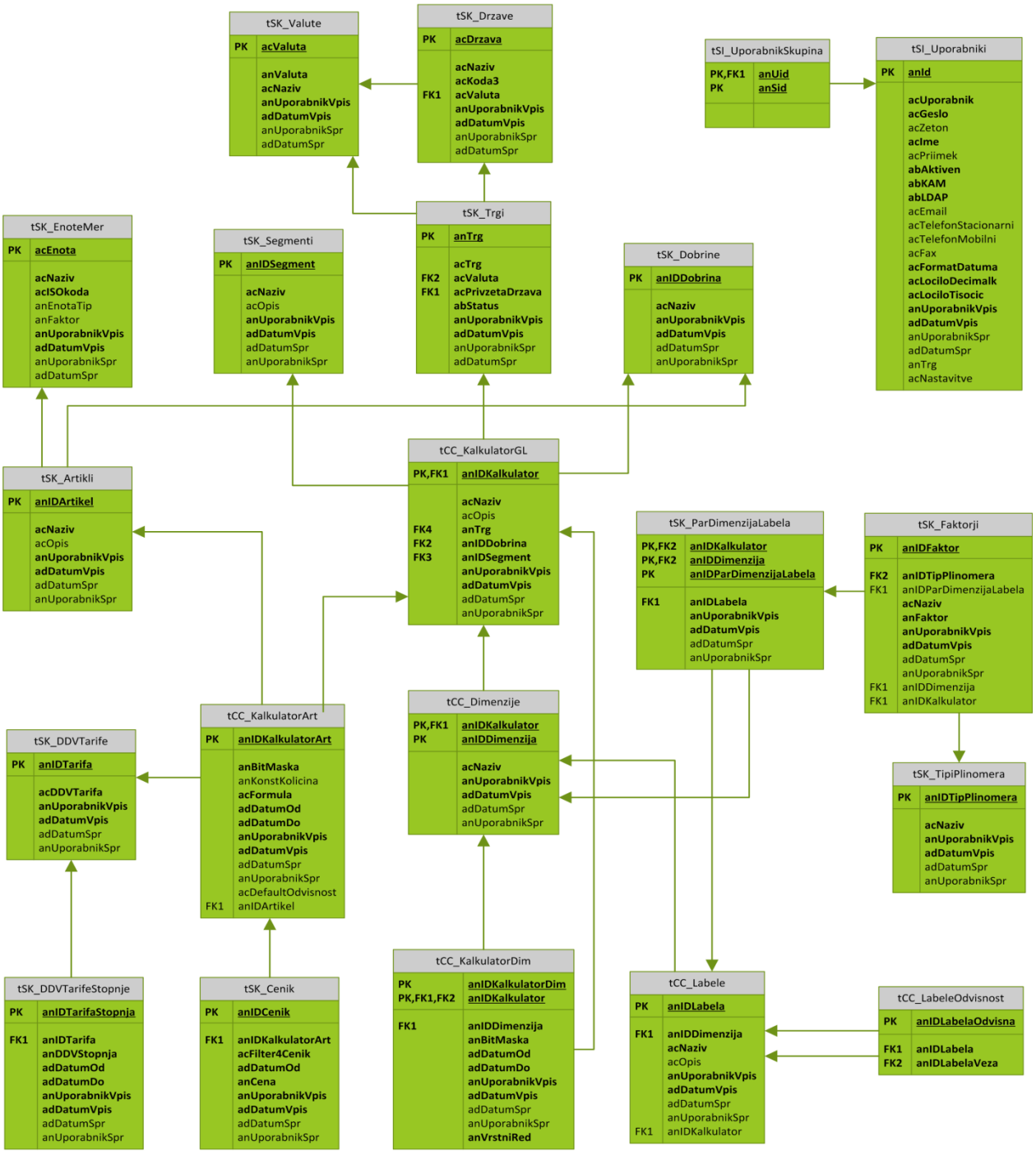
3. adDatumOd - datum, od katerega naprej velja dimenzija
4. abAktivna - logična vrednost aktiven ali neaktiven

Poleg tabel, prikazanih na sliki 13, ki prikazuje podatkovni model celotnega sistema, vsebuje podatkovna baza še nekatere tabele za beleženje sprememb in zgodovine dogodkov ter beleženje napak. Razlaga posamenih tabel je opisana v tabeli 2.

Tabela	Razlaga	Atributi
Kalkulator glave	Tabela za določanje kalkulatorjev, odvisna od trga, dobrine in segmenta, na podlagi teh treh atributov definiramo nov kalkulator, ki ga poimenujemo in mu dodamo opis	**PK**: anIDKalkulator, **FK**: anTrg, anIDDobrina, anIDSegment, acNaziv, acOpis
Trgi	Tabela vsebuje trge, na katerih organizacija posluje, na posamezen trg vežemo denarno valuto, preko atributa abStatus pa lahko določen trg skrijemo oziroma postavimo status na neaktiven	**PK**: anTrg, **FK**: acValuta, acTrg, acPrivzetaDrzava, abStatus
Dobrine	Tabela vsebuje vse dobrine, s katerimi organizacija posluje	**PK**: anIDDobrina, acNaziv
Segmenti	Tabela vsebuje vse segmente odjemalcev za vse dobrine, vendar ni vezana na dobrine, ker obe tabeli določata glavo kalkulatorja in posledično ne smeta biti odvisni	**PK**: anIDSegment, acNaziv, acOpis
Kalkulator dimenzije	Povezovalna tabela veže določene dimenzije na pripadajoč kalkulator, vsebuje tudi atribut anBitMaska, ki je potreben za natančno definiranje odvisnosti cen (Algoritem predstavljen v 4.3)	**PK**: anIDKalkulatorDim, **FK**: anIDKalkulator, anIDDimenzija, anBitMaska, anVrstniRed, adDatumOd, adDatumDo
Dimenzije	Tabela vsebuje vse obstoječe dimenzije v sistemu, ki jih v kalkulator dimenzije vežemo na primeren kalkulator, vsaka dimenzija pa je odvisna od dobrine	**PK**: anIDDimenzija, **FK**: anIDDobrina, acNaziv, acSifra, anVnos
Labele	Tabela vsebuje vse labele, ki jih vežemo na posamezno dimenzijo	**PK**: anIDLabela, **FK**: anIDDimenzija, acNaziv, acOpis

Kalkulator artikli	Povezovalna tabela veže določene artikle na kalkulator, vsebuje atribute anBitMaska, pove katera je vsota vseh anBitMaska dimenzij, od katerih je artikel odvisen, anKonstKolicina, ki pove, če se artikel obračunava konstantno ali dinamično, anIDTarifa pove odstotek davka na količino, acFormula pove, od katerih dimenzij je odvisna količina za artikel, acDefaultOdvisnost pa pove, če ima artikel kakšno specifično privzeto odvisnost	**PK**: anIDKalkulatorArt, **FK**: anIDKalkulator, anIDArtikel, anBitMaska, anKonstKolicina, anIDTarifa, acFormula, acDefaultOdvisnost, adDatumOd, adDatumDo
Artikli	Tabela vsebuje vse obstoječe artikle v sistemu, ki jih v kalkulator artikli vežemo na primeren kalkulator, vsak artikel pa je odvisen od dobrine	**PK**: anIDArtikel, **FK**: anIDDobrina, acNaziv, acEnota, acOpis
Ceniki	Tabela vsebuje vse cenike v sistemu, veže pa jih na kalkulator artikle oziroma se do njih dostopa preko acFilter4Cenik (Algoritem predstavljen v 4.3)	**PK**: anIDCenik, **FK**: anIDKalkulatorArt, acFilter4Cenik, adDatumOd, anCena
Uporabniki	Tabela vsebuje vse uporabnike in podatke o njih	**PK**: anId, acUporabnik, acGeslo, ..., acNastavitve
	Vse tabele poleg omenjenih atributov vsebujejo še:	anUporabnikVpis, adDatumVpis, adDatumSpr, anUporabnikSpr

Tabela 2: Razlaga tabel



Slika 13: Podatkovni model

4.3. Vmesnik za vzdrževanje cen

4.3.1. Modul šifranti

Na modulu šifranti so vsi šifranti, preko katerih urejamo podatke v tabelah. Vsi šifranti so dedovani iz glavnega šifranta, ki vsebuje funkciji *save()* in *values()*. Funkcija *values()* omogoča uporabo tabele in view-a na istem šifrantu, medtem ko funkcija *save()* skrbi za pravilno formatiranje datumov in shranjevanje v podatkovno bazo. Programska koda funkcije *save()*:

```
public function save(){
	// Tweaking: anTrg, all dates
	$allCols	= $this->_table_columns;
	$tableName	= $this->_table_name;

	foreach($allCols as $property => $data){
		if($property == 'anTrg' && $tableName != 'tCC_KalkulatorGL')
			 $this->anTrg = $_SESSION['trg'];

		switch($data['data_type']){
			case 'date':
			$this->$property = Date::format($this->$property,'Y-m-d');
			break;
			case 'datetime':
			$this->$property = Date::format($this->$property,'Y-m-d H:i:s');
			break;
			case 'datetime2':
			$this->$property = Date::format($this->$property,'Y-m-d H:i:s');
			break;
		}
	}

	if($this->loaded()){
		// update
		$this->anUporabnikSpr = $_SESSION['user']['anId'];
		$this->adDatumSpr = Date::format('now','Y-m-d H:i:s');
	}else{
		// insert
		$this->anUporabnikVpis = $_SESSION['user']['anId'];
		$this->adDatumVpis = Date::format('now','Y-m-d H:i:s');
	}
	parent::save();
}
```

Akcije, ki jih lahko izvajamo na posameznem šifrantu so kreiranje novega vnosa, urejanje vseh podatkov obstoječega vnosa razen primarnega ključa in brisanje obstoječih vnosov.

Seznam vseh šifrantov:

- Trgi
- Dobrine
- Segmenti
- Kalkulator Glave
- Kalkulator Dimenzije
- Kalkulator Artikli
- Artikli
- Dimenzije
- Labele
- Labele – odvisne
- Par dimenzija – labela
- Faktorji
- Tarife DDV
- Stopnje tarif DDV
- Merske enote
- Tip plinomera

Šifranta *kalkulator dimenzije* in *kalkulator artikli* imata implementirane posebnosti, in sicer oba v izbirnih menijih prikazujeta podatke šele po izbiri kalkulatorja in le podatke, ki pripadajo izbranem kalkulatorju.

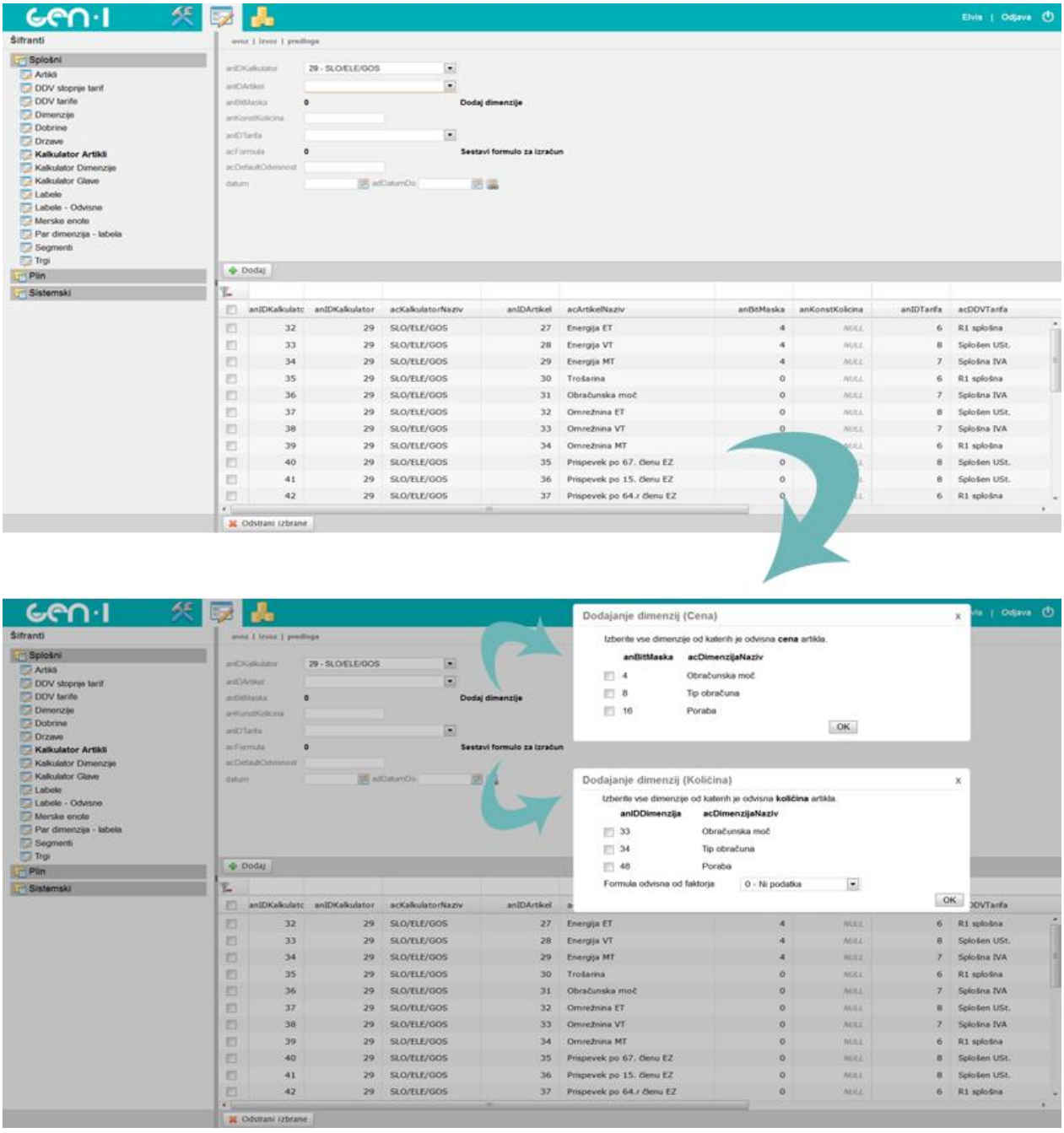
Postopek sestavljanja novega kalkulatorja je sledeč:

Najprej definiramo dobrino, trg in segment odjemalcev, na podlagi katerih ustvarimo nov kalkulator. Ko je kalkulator kreiran, naredimo dimenzije, ki vplivajo na stroške porabe in jim dodamo pripadajoče labele. Primer dimenzij in label:

- Dimenzije: obračunska moč, tip obračuna, poraba itd.
- Labele: Tip obračuna: enotarifni, dvotarifni itd.

Na šifrantu kalkulator dimenzije dodajamo posamezne dimenzije na določen kalkulator in jim sproti določimo parameter anBitMaska, za kar poskrbi šifrant sam. Naslednji korak je kreiranje novih artiklov, ki se obračunavajo za stroške porabe določene dobrine v izbrani državi. Primer artiklov: trošarina, obračunska moč, prispevek po 67. členu itd. Ko so artikli kreirani, jih preko šifranta kalkulator artikli dodamo na posamezen kalkulator in jim določimo skupno anBitMasko, ki predstavlja vsoto vseh *anBitMask* dimenzij, od katerih je cena artikla odvisna, *acFormulo*, preko katere določimo, od katerih dimenzij je odvisna količina za obračun, ter *anKonstKolicina*, ki pove konstantno količino za obračun, če jo artikel ima. Ko določimo vse odvisnosti artiklov, je kalkulator pripravljen za dodajanje cen na modulu ceniki.

Slika 14 prikazuje končni izgled modula šifranti in podrobno dodajanje artikla na šifrantu kalkulator artikli.



Slika 14: Končni izgled modula šifranti

4.3.2. Modul Ceniki

Modul je implementiran zelo dinamično in se spreminja glede na izbiro kalkulatorja. Na modulu so prikazani vsi artikli, ki pripadajo določenemu kalkulatorju. Na podlagi vsote *anBitMask*, od katere je artikel odvisen, določimo število nivojev potrebnih za dodajanje cenikov. Artikli, ki nimajo odvisnosti od dimenzij, so prikazani že v prvem koraku, medtem ko imajo ostali artikli na mestu za vnos cene gumb, ki odpre modalno okno z vsemi dimenzijami oziroma labelami, ki pripadajo dimenziji, od katere je artikel odvisen. Celoten končni izgled s primerom modalnega okna je prikazan v sliki 15. Ko dodajamo cene, nastavimo tudi datum, od katerega dalje velja vnesena cena in zaradi preprečitve napak nimamo parametra za datum, do katerega cena velja, ampak iz podatkovne baze vedno preberemo ceno, katere parameter adDatumOd je največji. Primer dodajanja cene:

Cene so zapisane v tabeli za cenike in do pravilne cene dostopamo preko polja *acFilter4Cenik*. Vsaka dimenzija ima atribut anBitMaska, ki je potenca števila 2. Polje *acFilter4Cenik* je sestavljen iz 10 mest ločenih z vejico, kjer vsako zapolnjeno mesto, razen prvih dveh, predstavlja id labele, medtem ko indeks mesta predstavlja potenco števila 2, s katero moramo število 2 potencirati, da dobimo bitMasko dimenzije. Prvo mesto je rezervirano za id kalkulatorja, drugo pa za id artikla. Filter po vejici razbijemo v polje in iz prvih dveh indeksov preberemo, za kateri kalkulator in artikel veljajo vpisani podatki, naprej pa preko indeksov iščemo točno ceno za izbrano kombinacijo label, na pripadajočih dimenzijah in jo prikažemo v tabeli rezultatov. Tabela 3 prikazuje primer z izmišljenimi podatki.

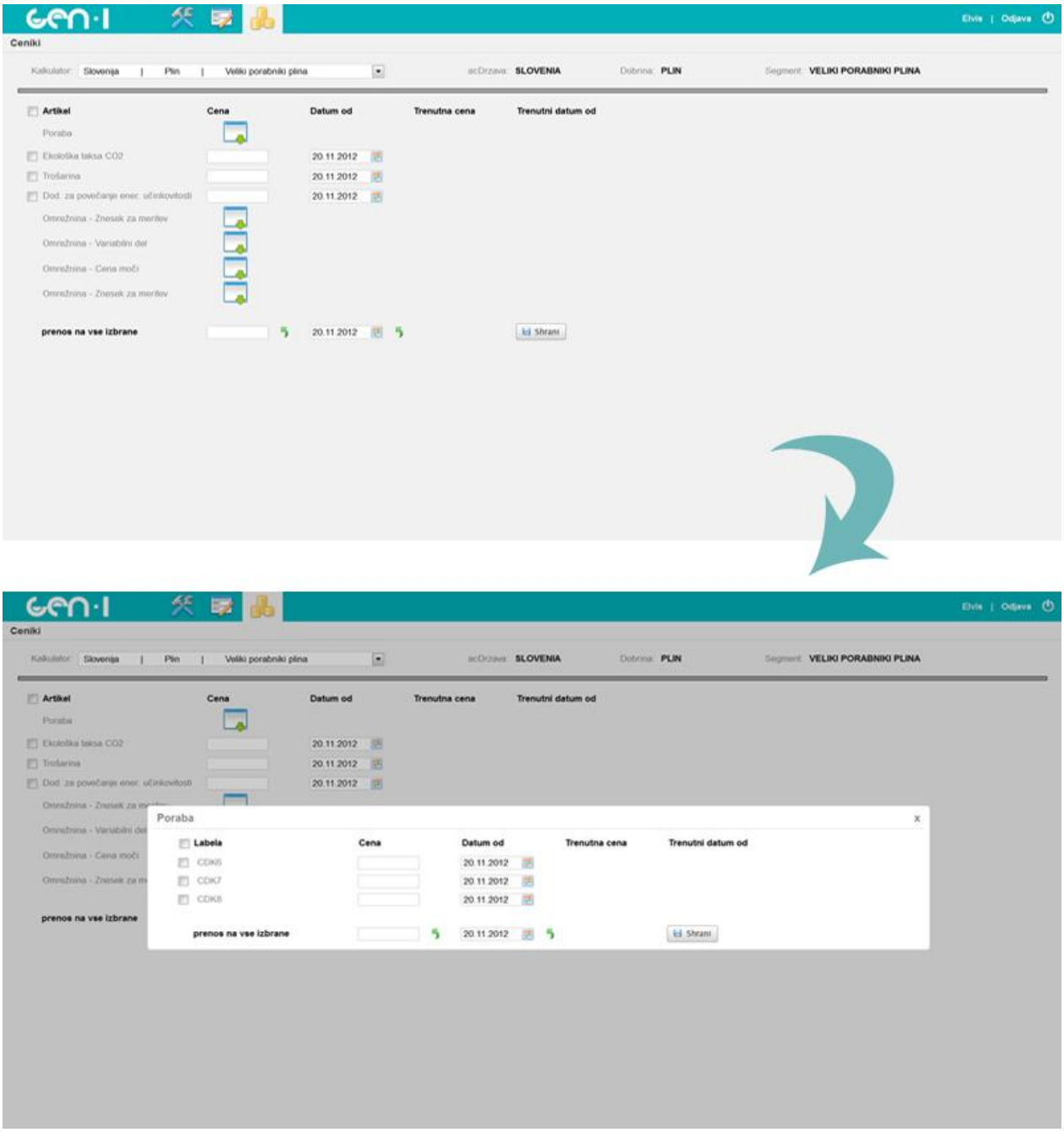
acFilter4Ceniki	31,42, , ,12, ,21,33, , ,									
acFilter4Cenik po razbitju	**31**	**42**			**12**		**21**	**33**		
indeks v polju po razbitju	**0**	**1**	2	3	**4**	5	**6**	**7**	8	9
potenca števila dva, kjer je eksponent enak mestu v polju => **anBitMaska**	**1**	**2**	4	8	**16**	32	**64**	**128**	256	512

Tabela 3: Primer določitve pravilne cene za izbrano kombinacijo

Vnos v tabeli ceniki, ki ima atribut *acFilter4Ceniki* enak »31,42,,,12,21,33,,« določa ceno in datum, od katerega naprej je cena aktivna za naslednjo kombinacijo:

Kalkulator z ID-jem 31, artikel z ID-jem 42, labelo z ID-jem 12, ki pripada dimenzij z anBitMasko 16, labelo z ID-jem 21, ki pripada dimenziji z *anBitMasko* 64 ter labelo z ID-jem 33, ki pripada dimenziji z *anBitMasko* 128.

Preko istega parametra *acFilter4Ceniki* cene določamo in po istem postopku, vendar v obratni smeri cene tudi preberemo pri računanju stroškov na vmesniku za kalkulator.



Slika 15: Končni izgled modula ceniki

5. PREDLOGI IN ZAKLJUČEK

Cilj zaključne naloge je bil do potankosti predstaviti sam razvoj in končno razvito aplikacijo, ter preostale podsisteme potrebne za pravilno delovanje aplikacije. V organizaciji se je uporabljalo več različnih rešitev za pridobivanje enakih rezultatov, kot jih ponuja novo razviti sistem in z implementacijo le tega smo končnim uporabnikom olajšali delo ter poenotili delovanje kalkulatorjev.

Z dodatnim načrtovanjem in s primernimi viri, kot so spletne storitve, bi lahko sistem dodatno nadgradili tako, da bi se podatki o cenikih v podatkovni bazi, za posamezne dobrine samodejno posodabljali na dnevni ali celo urni bazi. Možnost nadaljnje nadgradnje je sinhronizacija lokalne podatkovne baze s krovno podatkovno bazo celotnega sistema ob zagonu same aplikacije. Tako nadgrajen sistem bi omogočal samostojno delovanje procesov, brez interakcije vzdrževalcev in uporabnikov.

Z implementacijo sistema smo dosegli zastavljene cilje in zadovoljili potrebo ter povpraševanje po novem informacijskem sistemu, posledično pa pripomogli k izboljšanju notranjih procesov v podjetju in zadovoljstvu končnih uporabnikov.

6. LITERATURA IN VIRI

[1] Bhattacharjee, Supriyo (03. Januar 2008). Software Development Life Cycle (SDLC). *College of Agricultural Banking, RBI, PUNE.* Dostopno 30.11.2012 na internetu: http://www.cab.org.in/Lists/Knowledge%20Bank/Attachments/83/SDLC.pdf

[2] Boehm, Barry (1986). A Spiral Model of Software Development and Enhancement. *TRW Defense System Group*. Dostopno 30.11.2012 na internetu: http://weblog.erenkrantz.com/~jerenk/phase-ii/Boe88.pdf

[3] Bucanac, Christian (04. Januar 1999). The V-Model. University Of Karlskrona/ Ronneby. Dostopno 30.11.2012 na internetu: http://www.bucanac.com/documents/The_V-Model.pdf

[4] Burback, Ron (1998). Software engineering methodology: The watersluice. *Stanford University*. Dostopno 30.11.2012 na internetu: http://infolab.stanford.edu/~burback/watersluice/watersluice.pdf

[5] GEN-I d. o. o. Dostopno 30.11.2012 na internetu: http://www.gen-i.si/

[6] Hafmann , Torsten (17. oktober 2006). Software Development Process Models. *Computer science, University of Toronto*. Dostopno 30.11.2012 na internetu: http://www.cs.toronto.edu/~torsten/THahmann_CSC444_Tutorial1_SWDevProcesses.pdf

[7] Kohana Framework. Dostopno 30.11.2012 na internetu: http://kohanaframework.org/

[8] Lbaume (26. september 2007). System Development Life Cycle Checklists. *National Archives*. Dostopno 30.11.2012 na internetu: http://www.archives.gov/records-mgmt/initiatives/sdlc-checklist.pdf

[9] Martin, James (1991). Rapid application development. *Macmillan Publishing Co.* ISBN:0-02-376775-8

[10] Microsoft SQL Server. Dostopno 30.11.2012 na internetu: http://www.microsoft.com/sqlserver/en/us/product-info.aspx

[11] MooTools. Dostopno 30.11.2012 na internetu: http://mootools.net/

[12] Newton, Aaron (2008). MooTools Essentials: The Official MooTools Reference for JavaScript and Ajax Development(1st ed.). *Apress Berkely*. ISBN 1-4302-0983-6.

[13] Pelican engineering . *Glossary of Software Engineering Terms*. Dostopno 30.11.2012 na internetu: http://www.pelicaneng.com/DevDocs/seglossary.pdf

[14] Pelican engineering. *The Software Development Life Cycle (SDLC*). Dostopno 30.11.2012 na internetu: http://www.pelicaneng.com/DevDocs/sdlc.pdf

[15] Petlicki, John (2003). Software Development Life Cycle (SDLC). *De Paul Universit.* Dostopno 30.11.2012 na internetu: http://condor.depaul.edu/jpetlick/extra/394/Session2.ppt

[16] PHP. Dostopno 30.11.2012 na internetu: http://www.php.net/

[17] PHP history. Dostopno 30.11.2012 na internetu: http://si1.php.net/manual/en/history.php.php

[18] Satheesh (29. oktober 2009). SDLC Models. *OSQA*. Dostopno 30.11.2012 na internetu: http://www.onestopqa.com/resources/SDLC%20Models.pdf

[19] Texas Department of Information Resources (30. maj 2008). *System Development Life Cycle Guide*. Dostopno 30.11.2012 na internetu: https://www.dir.texas.gov/SiteCollectionDocuments/IT%20Leadership/Framework/Framework%20Extensions/SDLC/SDLC_guide.pdf

[20] Varun, Ramanujam (12. januar 2012). Testing Manual. *Learners Paradise*. Dostopno 30.11.2012 na internetu: http://www.learnersparadise.com/courses/ramanvar/public/doc/Testing%20Manual.doc