UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Andrej Šifrer

SPLETNA APLIKACIJA ZA VODENJE EVIDENCE NESTABILNOSTI TAL

Diplomska naloga

Ljubljana, 2005

Zahvaljujem se vsem, ki so kakorkoli pripomogli k nastanku moje diplomske naloge. Zahvala mentorju Matiji Lokarju za nadzor in koristne napotke pri izdelavi diplomske naloge. Zahvala zunanjemu mentorju Milošu Peganu za pomoč in usmerjanje pri učenju različnih informacijskih tehnologij. Zahvaljujem se tudi celotni skupini iz podjetja IGEA d.o.o, ki me je seznanjala z razvojnim okoljem in vsebino geografskih informacijskih sistemov.

Diplomsko nalogo posvečam svojim staršem.

Vsebina

Program dela

V diplomski nalogi na primeru razvoja spletne aplikacije za evidenco plazov prikažite način razvoja tovrstne aplikacije – od načrtovanja same aplikacije do odločitve glede uporabe določenih tehnologij. Prikažite tudi uporabo internetnih tehnologij kot so JSP (Java Server Pages), povezava z bazami podatkov in podobno.

Mentor:
mag. Matija Lokar

Zunanji mentor:
Miloš Pegan, univ. dipl. inž. str.

Povzetek

V diplomski nalogi sem opisal svojo izkušnjo z načrtovanjem in razvojem spletne aplikacije s področja geografskih informacijskih sistemov. Podrobno sem opisal razvoj spletne aplikacije za vodenje evidence nestabilnosti tal od načrtovanja do izbire tehnologij in izvedbe.

V razvojni skupini, ki je sistem razvijala, smo naredili analizo stanja podatkov o plazovih v Sloveniji in na podlagi dodatnih zahtev naročnika pripravili predlog za postavitev enotnega sistema za vodenje evidence plazov. Iz obstoječih evidenc smo sestavili seznam in strukturo podatkov, ki se bodo vodili, dodali geografske podatke o lokaciji in podatke povezali s podatki zemljiškega katastra in katastra stavb. Pri načrtovanju smo upoštevali raznolikost in število uporabnikov, ki bodo sistem uporabljali, in se odločili, da sistem razvijemo kot spletno aplikacijo, do katere lahko uporabniki dostopajo preko internetnega omrežja. Za razvoj spletne aplikacije smo izbrali tehnologijo Java Server Pages (JSP) in njeno nadgradnjo, ki jo ponuja Oracle, imenovano Application Development Framework (ADF). Baza podatkov, ki vsebuje tudi kompleksne geografske podatke, temelji na podatkovni bazi Oracle, sistem pa vključuje tudi grafični prikaz geografskih podatkov, ki je razvit kot Java Applet.

V diplomski nalogi sem poleg načrtovanja opisal spletni tehnologiji JSP in ADF in na praktičnih primerih ponazoril, kako poteka razvoj aplikacije v omenjenem okolju. Pri tem sem se omejil na posamezne zanimivejše dele. Tako sem opisal, kako zaradi zagotavljanja sledljivosti sprememb podatkov prijavo v aplikacijo izvedemo s pomočjo digitalnih certifikatov. Opisal sem glavne elemente ADF okolja, to so Business komponente za komunikacijo z bazo in UIX komponente za grajenje uporabniškega vmesnika. Zanimivo je tudi posredovanje podatkov med evidenco in grafičnim pregledovalnikom, ki delujeta kot dve ločeni aplikaciji. Tako sem na primeru opisal postopek, ko uporabnik na zemljevidu s klikom izbere parcelo, nato pa z enim klikom v evidenco uvozi opisne podatke o parceli in jih shrani kot del podatkov o plazu. Zanimiv problem je bil zagotavljanje ustrezne odzivnosti spletne aplikacije. Tehnologija JSP pri razvoju spletnih aplikacij med drugim ponuja objekt tipa Session, ki predstavlja t.i. sejo med odjemalcem in strežnikom. Pri razvoju evidence plazov smo ta objekt izkoristili za začasno shranjevanje podatkov pred zapisom v bazo.

Sistem Evidenca nestabilnosti tal smo predstavili tudi na 9. strokovnem srečanju uporabnikov programske opreme Oracle (SIOUG 2004) v Portorožu, kjer smo spoznali, da smo bili prvi, ki smo se odločili razvijati spletne aplikacije v okolju Oracle ADF.

Razvoj aplikacije od začetka do postavitve na strežnik je bila zame bogata izkušnja s področja informacijskih sistemov, tako njihovega razvoja kot tudi delovanja.

Math. Subj. Class. (2000): 68P20, 68U35, 68N15, 68N19

Ključne besede: Zemeljski plaz, evidenca, geografski informacijski sistem (GIS), Java Server Pages (JSP), Oracle Application Development Framework, Session.
Key words: Landslide, evidence, geographical information sistem, Java Server Pages, Oracle Application Development Framework, Session.

1. UVOD

Ocenjujejo, da je v Sloveniji trenutno preko 2800 zemeljskih plazov in sorodnih pojavov nestabilnosti tal. Kot posledica delovanja narave se pojavljajo praktično vsak dan. Večina plazov zahteva določeno sanacijo. S tem so običajno povezani veliki stroški, kjer mora znatna sredstva običajno prispevati tudi država. Zato je bilo nujno potrebno vzpostaviti centralno evidenco plazenja tal. Ažurna baza plazenja tal in drugih pojavov je namreč osnova za delovanje Ministrstva za okolje, prostor in energijo (v nadaljevanju MOP) na tem področju. MOP potrebuje ustrezne podatke tako v fazi pridobivanja podatkov o neposrednih nevarnostih, v fazi planiranja ukrepov in tudi med potekom sanacije. Prav tako so podatki potrebni za izdelavo ocen ogroženosti določenih področij. Z ažurno bazo je z ustreznimi analizami moč predvideti in zagotoviti ustrezne ukrepe za preprečevanje nastopa tovrstnih pojavov. S tem se tudi zmanjša potreben obseg sanacijskih ukrepov.

Poleg Ministrstva za okolje, prostor in energijo bodo informacijski sistem za vodenje evidence nestabilnosti tal v prvi vrsti uporabljale občine. Te se prve neposredno spopadajo s tovrstnimi naravnimi nesrečami. Občine bodo kot zunanji uporabniki aplikacije prijavljale novo nastale premike tal in vnesene podatke obnavljale. MOP kot skrbnik podatkov bo te vnose in spremembe preverjal ter jih potrjeval. Uporabnikov sistema bo torej veliko. Zato je najprimernejša tehnologija za implementacijo spletna tehnologija, ki omogoča centralno vzpostavitev informacijskega sistema. Strežnik z ustrezno aplikacijo je na enem mestu, uporabniki pa dostopajo do tega informacijskega sistema preko internetne povezave in spletnega pregledovalnika.

V diplomski nalogi sem predstavil, kako je bila tovrstna aplikacija razvita. Obravnavan bo način samega načrtovanja aplikacije, prikazan pa bo tudi razvoj nekaj ključnih delov same aplikacije, kot na primer prevzem podatkov iz grafike ter način prikazovanja in shranjevanja podatkov.

2. VSEBINA APLIKACIJE

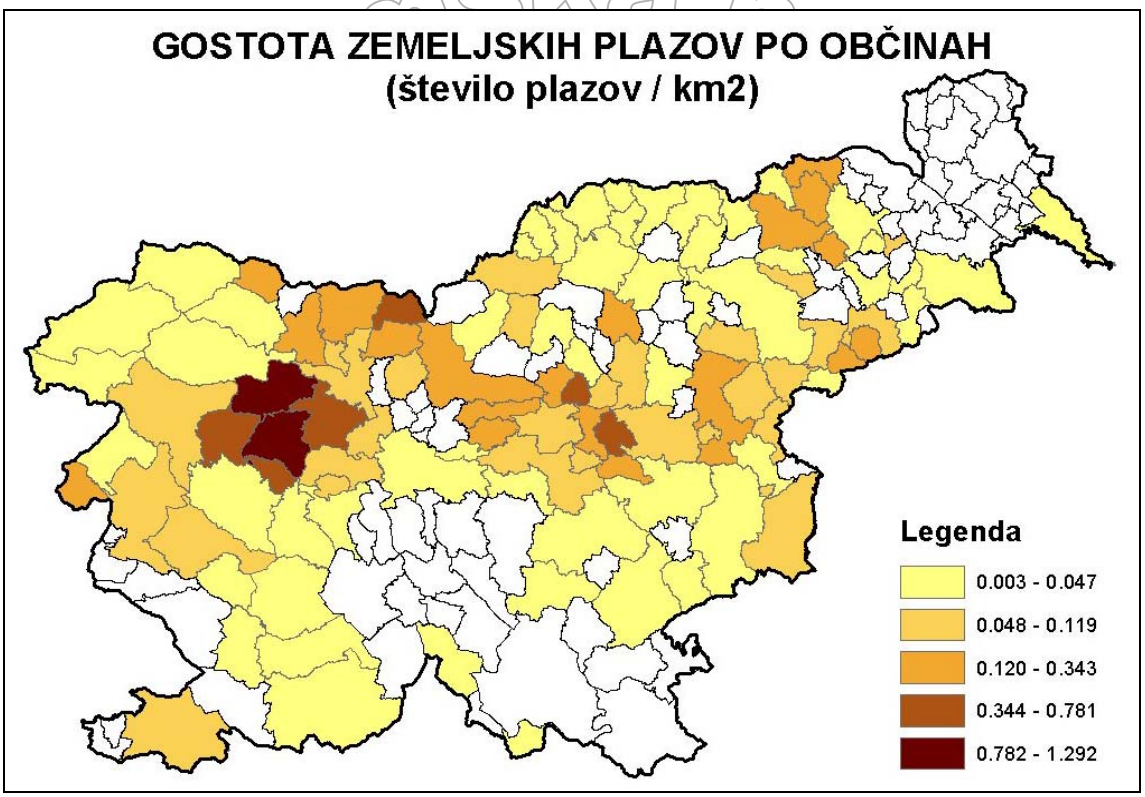
2.1. Analiza obstoječega stanja

Podatki o plazenju in drugih pojavih nestabilnosti tal se trenutno zbirajo na več mestih. MOP sicer razpolaga z evidenco največjega števila teh pojavov, a ne sodeluje pri sanaciji vseh. Zato ni nujno, da so vsi pojavi plazov in nestabilnosti tal zajeti v evidencah MOP.

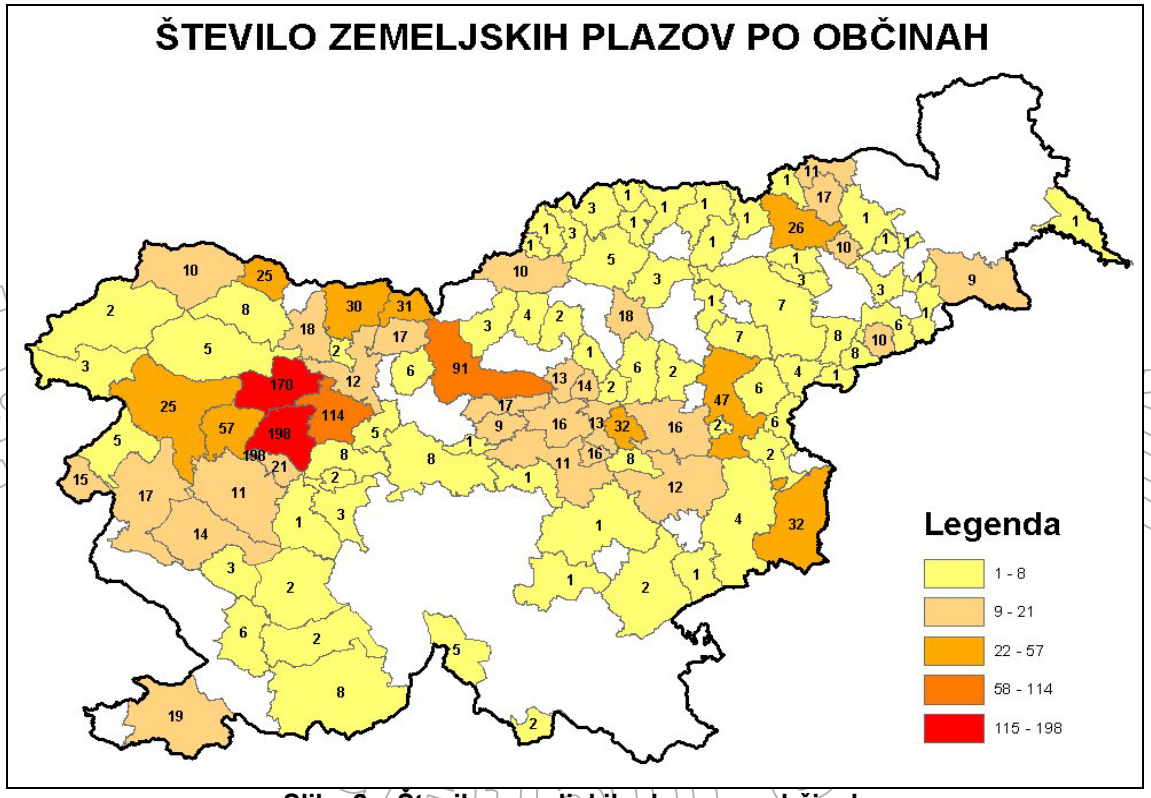
Glede na analizo ugotovimo, da so podatki o plazovih trenutno zajeti v naslednjih evidencah:

- Geografski informacijski sistem (GIS) UJME je digitalna baza plazov, v kateri je zajetih nekaj več kot 1300 plazov. Vsi vsebujejo tudi podatke o lokaciji. Zadnje ažuriranje baze je bilo izvedeno leta 1997, torej manjkajo podatki za zadnjih osem let.
- Oddelek za sanacijo naravnih in drugih nesreč v okviru MOP razpolaga z različnimi evidencami plazenja tal in drugih pojavov nestabilnosti tal. Te evidence so delno v digitalni, delno v papirni obliki. Veliko pomanjkljivost predstavlja predvsem dejstvo, da je geolokacija (zemljepisne koordinate položaja) podana le v posameznih primerih. V zadnjem obdobju je za prijavo teh pojavov predpisan standardni obrazec, ki vsebuje vse osnovne podatke, vključno z geolokacijo.
- Občine imajo lastne evidence plazenja tal in drugih pojavov nestabilnosti.
- Podatke o plazovih vodijo še na državnih cestah, železnicah, itd.

Tovrstnih evidenc je torej veliko. Podatki, ki so v njih zajeti, niso poenoteni, določeni (tudi ključni, kot je na primer geolokacija) pa večkrat manjkajo.



Slika 1 – Gostota zemeljskih plazov po občinah



Slika 2 – Število zemeljskih plazov po občinah

2.2. Uporabniki aplikacije

Uporabnike, ki bodo uporabljali aplikacijo, delimo na dve skupini:

Notranji uporabniki
MOP – ARSO[1] uporabnik ureja in skrbi za evidenco plazov. Ima pravico pregledovanja in spreminjanja podatkov vseh občin (ima skrbništvo nad vsemi podatki). Seveda lahko tudi vnaša podatke o novih plazovih.
Administrator sistema skrbi za pravilno delovanje sistema, ureja šifrante ter dodeljuje nova uporabniška imena in gesla.

Zunanji uporabniki
Občina prijavlja nove plazove. Na ta način si lahko zagotovi sredstva za odpravo posledic plazu. Občina vidi podatke ostalih občin, popravlja pa lahko samo lastne vnesene podatke.
Ostali uporabniki imajo možnost pregledovanja podatkov, ne morejo pa jih vnašati in spreminjati.

2.3. Glavni sklopi uporabe aplikacije

Prijava v sistem
Prijava v sistem je možna le s pomočjo digitalnega potrdila, ki ga izdaja SIGEN-CA[2]. Tako imamo popoln nadzor nad tem, kdo je vnašal, spreminjal ali pregledoval podatke. S tem preprečimo zlorabo sistema.

Iskanje in pregled podatkov
Aplikacija omogoča iskanje in pregled zbranih podatkov. Te delimo na atributne in grafične. Atributni podatki so opisni podatki o plazu, ki jih uporabniki vnašajo v sistem preko tipkovnice ali izbirajo iz šifrantov. Grafične podatke predstavlja zemljevid Slovenije, ki grafično prikazuje ceste, stavbe, parcele, plazove, itd.

Prijava plazu
Prijava plazu pomeni prvo omembo plazu v evidenci in prvi vnos podatkov. Vnašanje podatkov je možno na štiri načine:

- Ročno vnašanje podatkov s pomočjo vnosnih polj (ime plazu, ime lokacije plazu, stroški sanacije, itd)
- Izbiranje iz šifrantov, ki jih določa skrbnik podatkov, s pomočjo padajočega seznama (stanje plazu, hitrost plazenja, itd)
- Izbiranje iz šifrantov večjega obsega (ime in šifra občine, ime in šifra naselja, nabor parcel iz zemljiškega katastra)
- Izbiranje v grafičnem delu in uvoz podatkov v atributni del aplikacije (koordinate plazu, ogrožene ceste, ogrožene stavbe, prizadete parcele)

[1] ARSO – Agencija Republike Slovenije za okolje
[2] SIGEN-CA (Slovenian General Certification Authority) je izdajatelj kvalificiranih digitalnih potrdil na Centru Vlade RS za informatiko.

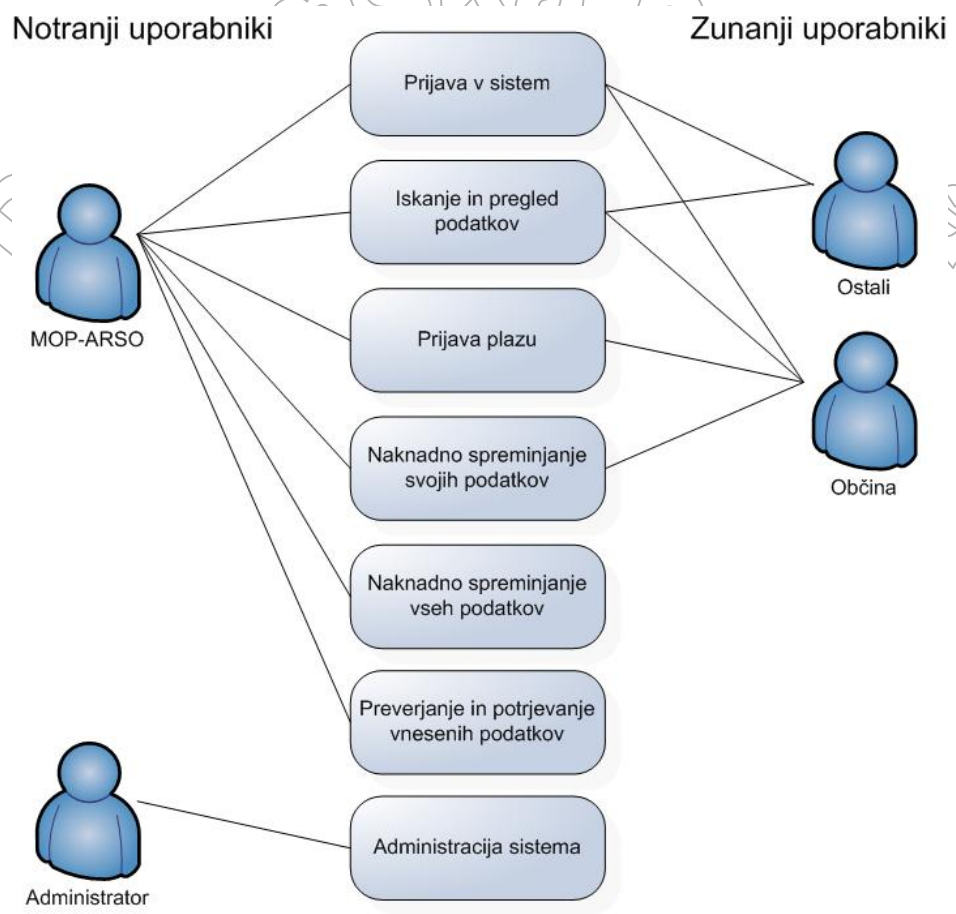
Spreminjanje podatkov
Možnost spreminjanja podatkov je opredeljena z vrsto uporabnika, glede na dodeljene pravice. Vsaka občina lahko spreminja le podatke, ki jih je sama vnesla, notranji uporabnik pa lahko spreminja vse podatke. Vsaka sprememba podatkov mora biti zabeležena, zato je potrebno voditi zgodovino sprememb (od prijave plazu do zadnje spremembe podatkov).

Preverjanje in potrjevanje vnesenih podatkov
Notranji uporabnik je skrbnik podatkov, zato mora imeti popoln nadzor nad njimi. Vsak vnos ali spremembo mora pregledati in potrditi. Ko so podatki potrjeni, postanejo veljavni in lahko služijo kot osnova za pridobivanje sredstev za odpravo posledic plazu in podobno. V primeru, da nove oziroma spremenjene podatke skrbnik zavrne, se le-ti ne izbrišejo iz baze, ampak se jim določi status neveljavnosti. To pomeni, da so podatki še vedno dostopni tako kot vsi ostali, le označeni so kot neveljavni (napačni) in jih zato ne smemo uporabljati za pridobivanje sredstev za sanacijo in podobno. Za vsak plaz imamo torej trenutno veljavne podatke ter celotno zgodovino vseh popravkov podatkov o plazu. Ti so bili bodisi neveljavni bodisi kasneje spremenjeni.

Administracija sistema
Skrbnik sistema (administrator) ima nadzor nad dostopom do aplikacije. Dodeljuje nova uporabniška imena, jim določa pravice in jih po potrebi briše. Poleg tega skrbi tudi za šifrante, ki se uporabljajo v sistemu. Šifrantom lahko dodaja vrednosti, jih spreminja ali briše.

Povezava med uporabniki in sklopi uporabe
Povezave med uporabniki in sklopi uporabe (pravice uporabnikov), so prikazane na Sliki 3 v obliki diagrama po standardu UML[1]. Več o tem standardu si bomo ogledali v nadaljevanju pri opisu načrtovanja. Diagram vsebuje uporabnike, sklope uporabe ter povezave med njimi. Vsaka povezava med uporabnikom in sklopom pomeni, da ima uporabnik pravico uporabljati ta sklop.

[1] UML – Unified Modeling Language je jezik za specifikacijo in vizualno načrtovanje informacijskih sistemov, predvsem namenjen za gradnjo poslovnih sistemov. V njem so zbrani določeni inženirski pristopi, ki so se izkazali za uspešne pri modeliranju velikih in kompleksnih sistemov.



Slika 3 - Diagram uporabnikov in sklopov uporabe po standardu UML

2.4. Opis podatkov

Osnova za podatke, ki se vodijo v evidenci plazov, je "Evidenčni list plazu". Ta list je že pred časom sestavil MOP. Podatke s tega lista smo razširili in jih prilagodili elektronskemu vnašanju. Zaradi potreb naročnika sistema smo dodali tudi precej novih podatkov, ki so povezani tako s samim stanjem plazu, kot tudi s sanacijo in odpravljanjem posledic plazu.

V splošnem so podatki razdeljeni na dve vrsti:

- Podatki, ki so s plazom v relaciji ena – ena (1 – 1). To pomeni, da vsak plaz vsebuje samo en podatek določene vrste. Primer takega podatka je ime plazu, saj ima vsak plaz lahko samo eno ime.
- Podatki, ki so s plazom v relaciji ena - več (1 – *). Plaz takrat lahko vsebuje več podatkov določene vrste. Primer takega podatka je vzrok nastanka plazu. Na nastanek plazu je lahko vplivalo več vzrokov, zato je potrebno, da plaz lahko vsebuje več vzrokov nastanka plazu.

Podatki, ki so s plazom v relaciji ena – več, so pogosto sestavljeni podatki. Podani so v obliki tabele, ki vsebuje več stolpcev. Tako je na primer vsak vzrok nastanka plazu (posamezna vrstica v tabeli) opisan s štirimi podatki. Ti so odstotek, tip vzroka, sprožitelj in opomba.

Spodnja tabela prikazuje podatke, ki se vodijo v evidenci, njihovo vrsto ter način vnosa. Načini vnosa so lahko:

- *določi sistem* (zaporedne številke določa sistem sam),
- *vnos* (ročni vnos s tipkovnico),
- *izbor v grafiki (*vnos preko grafičnega vmesnika),
- *izbor iz šifranta* (izbor iz šifranta, ki se odpre v novem oknu in vsebuje iskalnik) ali
- *šifrant* (izbor iz šifranta neposredno na vnosni formi s pomočjo padajočega seznama).

PODATEK	VRSTA PODATKA	NAČIN VNOSA
zaporedna številka plazu	1 – 1	določi sistem
šifra dogodka	1 – 1	določi sistem
ime plazu	1 – 1	vnos
ime lokacije	1 – 1	vnos
občina	1 – 1	izbor iz šifranta občin
naselje	1 – 1	izbor iz šifranta naselij
centroidi		
– X	1 – 1	vnos ali izbor v grafiki
– Y	1 – 1	vnos ali izbor v grafiki
– Z	1 – 1	vnos
vir podatkov	1 – 1	šifrant
vzroki nastanka	1 – *	vnos in šifrant
poškodovani in ogroženi objekti:		
– ceste	1 – *	vnos ali izbor v grafiki
– stavbe	1 – *	vnos ali izbor v grafiki
– javna infrastruktura	1 – *	vnos
– zemljišča	1 – *	vnos

– poškodbe in ogroženosti (za vsako cesto, stavbo, javno infrastrukturo ali zemljišče) – ukrepi na objektu (za vsako cesto, stavbo, javno infrastrukturo ali zemljišče)	1 – * 1 – *	vnos in šifrant vnos in šifrant
prizadete parcele	1 – *	izbor iz šifranta parcel ali izbor v grafiki
dimenzije – dolžina – širina – globina – volumen	 1 – 1 1 – 1 1 – 1 1 – 1	 vnos vnos vnos izračuna sistem
stanje	1 – 1	šifrant
hitrost	1 – 1	šifrant
oblika drsne ploskve	1 – 1	šifrant
vrsta premikanja	1 – 1	šifrant
smer plazu – azimut	1 – 1	vnos
nagib plazenja	1 – 1	vnos
oddaljenost med mestom proženja in odlaganja plazu	1 – 1	vnos
prioriteta reševanja	1 – 1	šifrant
dokumentacija	1 – *	vnos, šifrant in izbira iz lokalnega diska (upload)
fotografije	1 – *	vnos in izbira iz lokalnega diska (upload)
URL (spletni naslovi)	1 – *	vnos
opis	1 – 1	vnos
škoda	1 – *	vnos in šifrant
sanacija	1 – *	vnos in šifrant
ukrepi	1 – *	vnos in šifrant
slojevitost podlage	1 – 1	šifrant
orientiranost sloja	1 – 1	šifrant
sestava plazine	1 – 1	šifrant
razširjanje plazu	1 – 1	vnos
vrsta plazu	1 – 1	šifrant
globina drsne ploskve	1 – 1	vnos
litološki opis / sestavina plazu	1 – *	vnos in šifrant
hidrogeološki opis	1 – *	vnos in šifrant
geomehanska klasifikacija	1 – *	vnos in šifrant
vsebnost vode	1 – *	vnos in šifrant
verjetnost širitve plazu	1 – 1	šifrant
tip širitve plazu	1 – 1	šifrant
položaj plazu	1 – 1	šifrant
oblika terena plazu	1 – 1	šifrant
povprečen nagib plazu	1 – 1	vnos

3. TEHNOLOGIJA

3.1. Izbira tehnologije

Glavni namen postavitve centralne evidence nestabilnosti tal je na enem mestu v podatkovni bazi združiti vse podatke o vseh plazovih v Sloveniji. Do te baze naj bi potem dostopalo večje število uporabnikov iz vseh koncev Slovenije. Do ene same baze naj bi bil torej mogoč dostop iz večjega števila računalnikov.

Uporabniški vmesnik za dostop do centralne evidence je v splošnem možno razviti na dva načina: v obliki namizne aplikacije ali v obliki spletne aplikacije. Namizna aplikacija je program, ki je nameščen na uporabnikovem računalniku in se tam tudi v celoti izvaja. Ker uporabnik želi delati s centralno bazo podatkov, ki ni na njegovem računalniku, program potrebuje tudi povezavo v omrežje. Program zahteva od baze določene podatke, jih dobi, obdela na lokalnem računalniku in obdelane pošlje nazaj v centralno bazo. Problem pri tovrstnih aplikacijah je zlasti v vzdrževanju celotnega sistema. Skrbeti moramo, da je na vseh računalnikih, kjer se prikazujejo in obdelujejo podatki, nameščen ustrezen program. Če (ko) pride do spremembe programa, je popravke potrebno namestiti na vse računalnike. Če želimo z novega računalnika dostopati do centralne evidence, moramo nanj namestiti ustrezni program. To je pogosto nepraktično, saj večkrat potrebujemo z določenega računalnika le začasni dostop do centralne evidence.

Ker centralna evidenca v vsakem primeru zahteva stalno povezavo v omrežje, se pojavi alternativa namiznim aplikacijam: spletne aplikacije. Če vsi uporabniki uporabljajo eno bazo podatkov, zakaj ne bi uporabljali tudi vsi samo ene aplikacije. Ta bi se izvajala na centralnem računalniku in bi uporabnikom vračala samo rezultate, ki bi jih želeli. Uporabniški vmesnik naj ne bi bil samostojen program, ampak naj bi uporabnik uporabljal orodje, ki je na praktično vsakem računalniku s pristopom do omrežja - spletni brskalnik. Preko spletnih strani naj se izvaja celotna komunikacija med uporabnikom in centralno bazo. Na ta način delujejo spletne aplikacije.

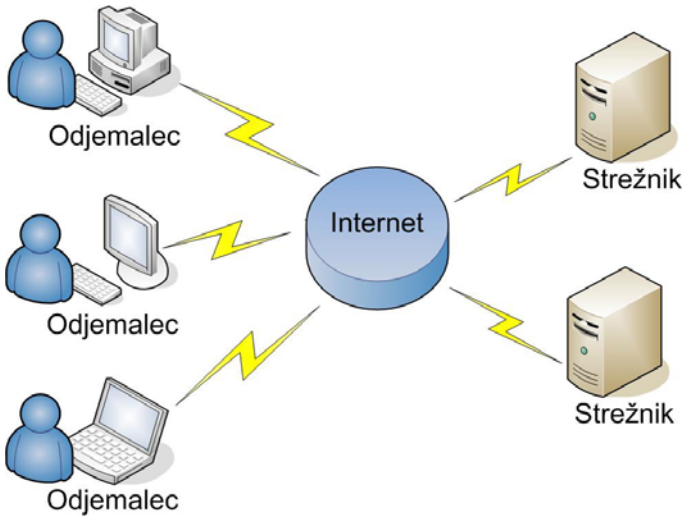
Spletna aplikacija je program, ki je nameščen na enem samem računalniku (strežniku), ki je povezan v omrežje in do katerega lahko dostopa več računalnikov (odjemalcev) hkrati s poljubnega mesta znotraj omrežja. Uporabniški vmesnik je razvit kot spletna stran. Takšen način postavitve aplikacije je veliko lažje vzdrževati, omrežje za povezavo pa je že postavljeno in zelo razširjeno: internet.

Ker tehnologija za razvoj spletnih aplikacij ponuja dovolj varnosti in zaščite podatkov v javnem omrežju, smo se pri implementaciji evidence plazov odločili za spletno aplikacijo. Slabost tega pristopa je le v tem, da zahteva ustrezno zmogljiv strežnik, ki se zmore ustrezno odzivati vsem sočasnim uporabnikom, ki pa jih v našem primeru ne bo veliko.

3.1.1. Odjemalec - strežnik

Spletna tehnologija deluje na principu odjemalec - strežnik, kjer odjemalec pošilja zahteve, strežnik pa te zahteve obdela in odjemalcu vrne zahtevano vsebino (spletno stran). Vsa poslovna logika aplikacije (prebiranje zahtev, priprava podatkov iz baze, itd) se torej odvija na strežniku, odjemalec le oblikuje zahteve in sprejema končne rezultate. S tem se večina bremena, ki ga pri namiznih aplikacijah nosi osebni računalnik, prenese na strežnik.

Odjemalec pošilja na strežnik zahteve, ki so v svoji sestavi precej kratke in enostavne in tako ne obremenjujejo povezave. Večje breme za povezavo predstavlja odgovor s strani strežnika. Ta v večini primerov pošilja do odjemalca večje količine podatkov, kot so slike in video posnetki. Ker smo še v času, ko večina uporabnikov nima širokopasovnega dostopa do interneta, moramo pri razvoju spletnih aplikacij upoštevati tudi to dejstvo.



Slika 4 - Način delovanja spletne tehnologije

3.1.2. Statične – dinamične spletne strani

Spletne strani, ki jih odjemalcu vrača strežnik, so lahko statične ali pa se generirajo dinamično. Statične strani so tiste, ki so postavljene na strežnik in se ne spreminjajo. Ob zahtevi po taki strani strežnik vsem uporabnikom v vsakem primeru vrne enako stran – torej iste podatke. Uporabniki ne morejo vplivati na vsebino prikazane spletne strani.

Dinamične spletne strani se za razliko od statičnih generirajo glede na uporabnikove vnose in parametre. Uporabnik preko uporabniškega vmesnika vstavlja in izbira podatke. Strežnik te podatke sprejme, jih obdela in na podlagi njih generira novo spletno stran. To kot odgovor na vnesene podatke pošlje nazaj odjemalcu.

Poglejmo si preprost primer razlike med statičnimi in dinamičnimi spletnimi stranmi. Denimo, da bi želeli, da ima uporabnik na voljo različne barve ozadja.

- Pri statičnih spletnih straneh je barva ozadja točno določena in enaka za vse odjemalce. Če želimo torej imeti spletne strani z različnimi ozadji, moramo vnaprej pripraviti več spletnih strani. Te bodo enake, razlikovale se bodo le po barvi ozadja. Uporabnik si v seznamu potem izbere stran z določeno barvo ozadja. Problem nastopi takrat, ko je strani veliko. Za vsako od ponujenih strani bi morali imeti različice z vsemi barvami ozadja. Če bi kasneje želeli spremeniti vsebino neke strani, bi popravek morali narediti na vseh različicah te strani. Del težav bi sicer lahko omilili z uporabo skriptnih jezikov, vendar je lažje, če se ustrezne spletne strani generirajo sproti glede na dane parametre.
- Pri dinamično ustvarjenih spletnih straneh lahko uporabniku ponudimo izbirno polje, kjer sam izbere barvo ozadja. Izbrano barvo si lahko znotraj spletne aplikacije tudi zapomnimo in vse spletne strani znotraj aplikacije bodo imele barvo ozadja tako, kot jo je izbral uporabnik. Ustrezna koda HTML z izbrano barvo bo generirana vsakič sproti, zato ne bo težav tudi, če bo uporabnik naknadno spet spremenil želeno barvo.

3.1.3. Format zapisa za prikaz vsebine spletne strani

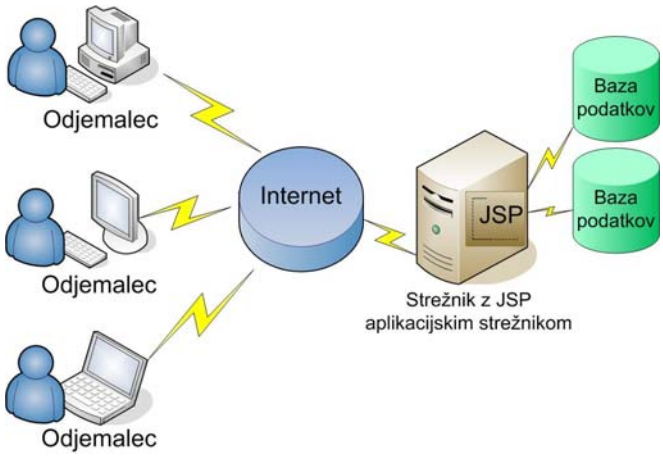
Format zapisa podatkov, ki ga kot rezultat zahtevka pošilja strežnik in ga razumejo spletni pregledovalniki, se imenuje HTML (Hyper Text Markup Language) in že od samega začetka interneta predstavlja osnovo za grajenje spletnih strani. Za oblikovno raznolikost skrbi jezik CSS (Cascading Style Sheet), za odzivnost in dinamičnost spletne strani na strani odjemalca pa se je uveljavil skriptni jezik imenovan JavaScript. Ti trije jeziki so trenutno najbolj razširjeni in jih podpirajo vsi spletni pregledovalniki. Skupaj predstavljajo dovolj svobode pri oblikovanju in so se uveljavili kot standardni jeziki za oblikovanje spletnih strani. Zato je tudi naša spletna aplikacija zasnovana na uporabi teh standardnih jezikov. Uporaba le-teh omogoča, da uporabniki ne bodo imeli težav z uporabo programa, saj bo zadoščal že standardni brskalnik.

3.1.4. Baza podatkov

Za bazo podatkov smo izbrali Oracle Database. Ta je trenutno med boljšimi na področju shranjevanja podatkov in nudi dovolj ustreznih rešitev za shranjevanje geografskih podatkov. Evidenca plazov bo v končni verziji delovala na Centru vlade za informatiko, kjer je že postavljena baza podatkov o katastru, ki jo bo evidenca plazov v veliki meri tudi uporabljala. Ker je omenjena baza Oracle baza, ni bilo dvoma o izbiri baze za evidenco plazov, saj smo se tako izognili usklajevanju razlik med različnimi bazami.

3.1.5. Programski jeziki za dinamično generiranje spletnih strani na strani strežnika

Za razliko od enostavnosti in majhnega števila programskih jezikov za prikaz spletnih strani, je na področju strežnikov, ki te spletne strani generirajo, stanje prav nasprotno. Poslovna logika aplikacije na strani strežnika je veliko bolj kompleksna. Obstaja veliko med seboj nezdružljivih tehnologij kot so JSP (Java Server Pages), ASP (Active Server Pages), PHP, CGI (Common Gateway Interface), itd.
Te tehnologije združujejo jezike za generiranje uporabniškega vmesnika (izgleda) aplikacije in programske jezike za implementacijo poslovne logike (prebiranje parametrov iz zahtevka, komuniciranje z bazo podatkov, itd). Za razliko od jezikov za prikaz strani, ki jih znajo interpretirati že sami spletni pregledovalniki, programski jeziki na strani strežnika potrebujejo ustrezen aplikacijski strežnik, na katerem se izvajajo.



Slika 5 - Način delovanja spletne aplikacije razvite v JSP tehnologiji

3.1.6. JSP – Java Server Pages

Kaj je to Java Server Pages?
Java Server Pages (JSP) je tehnologija za razvijanje spletnih strani z dinamično vsebino. Za razliko od statične HTML strani, katere vsebina je vedno enaka, JSP stran gradi vsebino glede na poljubno število različnih parametrov, kot so npr. podatki o uporabniku, informacije, ki jih uporabnik vnaša ali izbira, itd. Ta tehnologija omogoča spletno nakupovanje, bančno poslovanje, pregled podatkov o katastru, itd. ter vsebino spletnih strani, ki je lahko v več jezikih in jo uporabniki priredijo svojim željam.
JSP stran vsebuje standardne HTML elemente ter posebne JSP elemente, ki strežniku omogočajo vstavljanje dinamičnih vsebin na spletne strani. JSP elementi lahko prebirajo podatke iz baze, shranjujejo podatke o uporabniku, vodijo evidenco dostopa, itd. Ko uporabnik pošlje zahtevo po JSP strani, server izvede JSP elemente, rezultat uskladi s statičnim delom strani in brskalniku vrne dinamično ustvarjeno stran.

Zakaj smo izbrali JSP?
V začetku svetovnega spleta je bila edina tehnologija za kreiranje dinamičnih spletnih strani Common Gateway Interface (CGI), ki pa danes ni več tako učinkovita. Skozi leta so se razvile nove tehnologije, ki imajo v večini primerov eno pomanjkljivost: spletne strani gradijo tako, da HTML elemente vstavljajo neposredno v programsko kodo svojega jezika. S tem je celotno breme razvoja spletne aplikacije na razvijalcu.
JSP tehnologija deluje ravno obratno. Namesto da bi vstavljali HTML elemente v programsko kodo, vstavljamo posebne JSP elemente v HTML kodo. Ti elementi so podobni HTML elementom, vendar v ozadju nosijo različne programe v programskem jeziku Java. Te strežnik izvede, ko uporabnik zahteva stran.

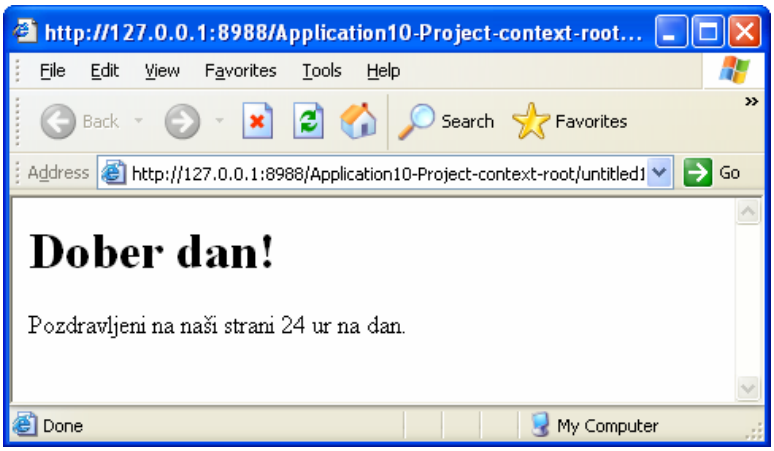
Primeri JSP strani
Poglejmo si enostaven primer JSP strani (JSP elementi so odebeljeni, ostalo je HTML koda):

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<html>
  <body bgcolor="white">
    <jsp:useBean id="ura" class="java.util.Date" />
    <c:choose>
      <c:when test="${ura.hours < 12}">
        <h1>Dobro jutro!</h1>
      </c:when>
      <c:when test="${ura.hours < 18}">
        <h1>Dober dan!</h1>
      </c:when>
      <c:otherwise>
        <h1>Dober Večer!</h1>
      </c:otherwise>
    </c:choose>
    Pozdravljeni na naši strani 24 ur na dan.
  </body>
</html>
```

Ta stran vrne uporabniku sporočilo, ki je odvisno od časa, ko je uporabnik zahteval stran: "Dobro jutro", če je čas pred 12. uro, "Dober dan", če je čas med 12. in 18. uro ter "Dober večer" v ostalih primerih.
Vidimo, da so JSP elementi strukturirani kot običajne značke v jeziku HTML, kar omogoča, da te strani brskalniki pravilno tolmačijo. Ko uporabnik zahteva stran, aplikacijski strežnik izvede kodo, ki je predstavljena z odebeljenimi JSP elementi in ustvari HTML stran, ki jo nato pošlje nazaj uporabniku. Če je npr. trenutni čas 16:25, je HTML stran, ki jo ustvari strežnik, videti tako:

```
<html>
  <body bgcolor="white">
    <h1>Dober dan!</h1>
    Pozdravljeni na naši strani 24 ur na dan.
  </body>
</html>
```

Slika 6 prikazuje omenjeno HTML stran tako, kot jo vidi uporabnik v brskalniku.



Slika 6 – Rezultat enostavne JSP strani

Poleg JSP elementov lahko JSP stran vsebuje tudi programsko kodo v Javi, ki jo vstavljamo v t.i. *skriptne elemente (scripting elements)*. Ta način pisanja JSP kode obstaja že od samega začetka tehnologije in je primeren za enostavnejše pisanje pogojnih stavkov, definiranje enostavnih metod, prebiranje parametrov, itd. Skriptni elementi so trije:

- <%! … %> - definicija metod in razredov
- <% … %> - logika pisana v Java kodi
- <%= … %> - izpis Java spremenljivke v HTML

Primer:

```
<html>
  <%!
    int sestej(String a, String b) {
      if(a != null && b != null)
      return Integer.parseInt(a) + Integer.parseInt(b);
      else return -1;
    }
  %>
  <body bgcolor="white">
    <%
      String a = request.getParameter("a");
      String b = request.getParameter("b");
      if(sestej(a,b) >= 0) {
    %>
      Vsota je pozitivna!
    <% } else { %>
    <form>
      Vsota je negativna! Popravite števili!<br>
      število a: <input type="text" name="a" value="<%=a%>"/><br>
      število b: <input type="text" name="b" value="<%=b%>"/><br>
      <input type="submit" value="Preveri vsoto"/>
    </form>
    <% } %>
  </body>
</html>
```

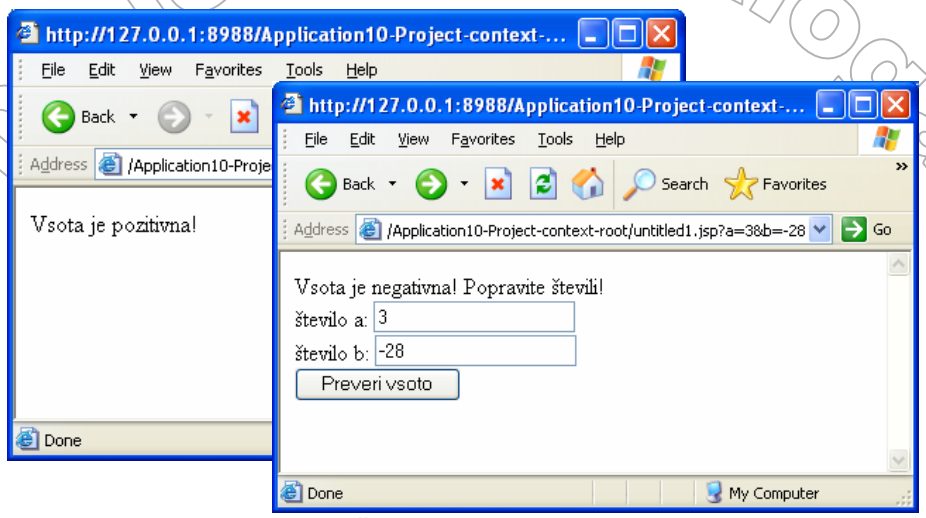
Zgornja JSP stran na podlagi prejetih parametrov `a` in `b` vrne HTML stran, ki ima lahko dve različni vsebini. V primeru `a = 3`, `b = 28`, ko je vsota parametrov pozitivna, vrne:

```
<html>
  <body bgcolor="white">
    Vsota je pozitivna!
  </body>
</html>
```

V primeru `a = 3`, `b = -28`, ko je vsota negativna, vrne:

```
<html>
  <body bgcolor="white">
    <form>
      Vsota je negativna! Popravite števili!<br>
      število a: <input type="text" name="a" value="3"/><br>
      število b: <input type="text" name="b" value="-28"/><br>
      <input type="submit" value="Preveri vsoto"/>
    </form>
  </body>
</html>
```

Slika 7 prikazuje rezultat JSP strani, kot ga vidi uporabnik v brskalniku.

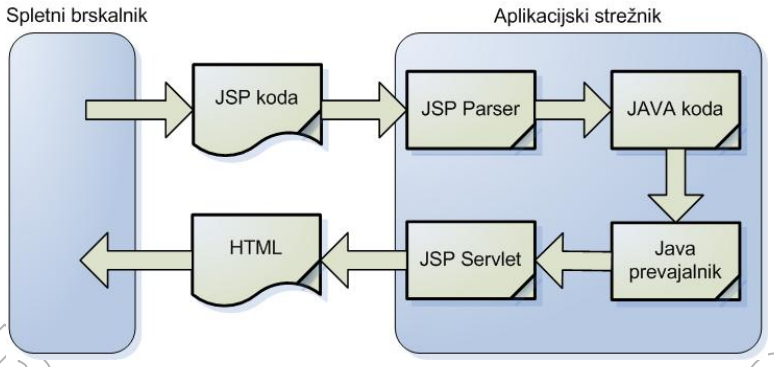


Slika 7 - Dva različna HTML rezultata ene JSP strani

Prevajanje in izvajanje JSP strani

Prednost JSP tehnologije je tudi v njenem načinu izvajanja na aplikacijskem strežniku. Za razliko od nekaterih drugih spletnih tehnologij, se JSP stran prevede samo enkrat. Ko po postavitvi določene JSP strani na aplikacijski strežnik prvi uporabnik zahteva to stran, se ta prevede v izvajalno obliko. Ta se potem uporabi pri vseh naslednjih zahtevah po tej strani.

Spodnja slika prikazuje izvajanje kode JSP ob prvem klicu po postavitvi programa na aplikacijski strežnik. Ob prvem klicu se koda prevede v izvajalno obliko. *JSP parser* najprej kodo, ki je sestavljena iz ukazov v jeziku HTML in v programskem jeziku Java, prevede v standardni program v jeziku Java. Tega *prevajalnik* za programski jezik J*ava* prevede v izvajalno obliko imenovano *Servlet.* Ta ni nič drugega kot datoteka class, torej javanska vmesna koda.



Slika 8 - Izvajanje JSP kode ob prvem klicu

Pri nadaljnjih klicih te kode prevajanje ni več potrebno. Ko spletni brskalnik pošlje zahtevek, se na aplikacijskem strežniku takoj izvede ustrezni Servlet, ki generira odgovor strežnika brskalniku.

3.2. Oracle ADF tehnologija

Pri evidenci plazov smo za razvoj programa uporabili nadgradnjo tehnologije JSP,- *Application Development Framework (ADF)*. To nadgradnjo razvija podjetje Oracle. Gre za tehnologijo, ki v celoti temelji na Javi. S pomočjo razvitih objektov in orodij za delo z njimi nam olajša in skrajša razvoj spletnih aplikacij.

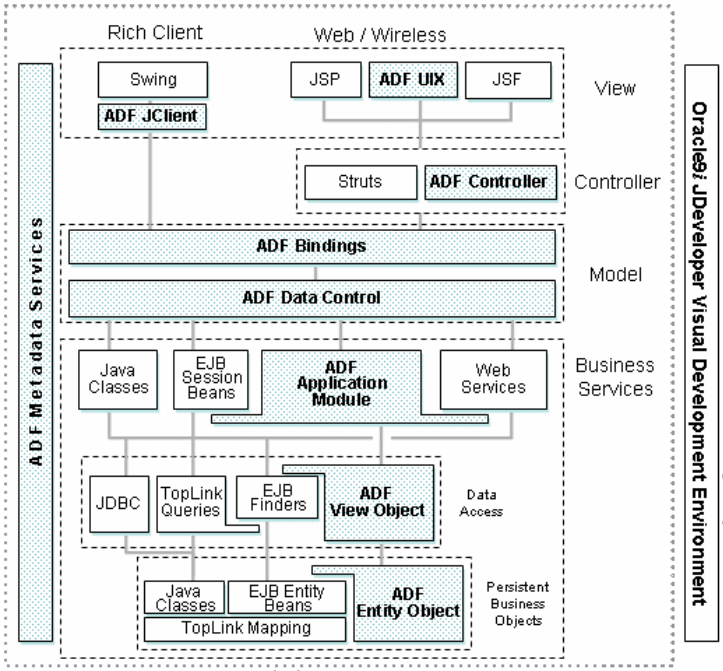
ADF sledi arhitekturi *Model - View - Controller (MVC)*. To pomeni da razvoj aplikacije razdeli na tri logično ločene dele:

- Model – predstavlja komunikacijo z bazo. V tem delu predvidimo način izvajanja operacij kot so prebiranje podatkov iz baze, oblikovanje podatkov, vnašanje v bazo, itd
- View – predstavlja uporabniški vmesnik. V tem delu določimo izgled prikazane spletne strani
- Controller – predstavlja poslovno logiko, ki se izvaja na strani aplikacijskega strežnika. V tem delu načrtujemo, kako se prebirajo zahteve in dinamično (glede na parametre) oblikujemo spletne strani. Ta del povezuje dela Model in View.

Prednost tehnologije ADF je v tem, da podpira več različnih načinov razvoja posameznega logičnega dela aplikacije. Tako se razvijalec lahko sam odloči, kateri najbolj ustreza njegovim zahtevam. Na posameznem nivoju nam Oracle omogoča uporabo naslednjih načinov, razdeljenih po prej predstavljenih logičnih delih:

- Model: EJB (*Enterprise Java Beans*), TopLink in BC4J (*Business Components for Java*)
- View: JSP (HTML + Java), UIX (Oracle-ov produkt) in UIX JSP (kombinacija prejšnjih dveh)
- Controller: Jakarta Struts

Slika 9 prikazuje arhitekturo in tehnologije razvoja aplikacij v ADF okolju.



Slika 9 – ADF aplikacijska arhitektura

Tehnologija Oracle ADF razvoj aplikacije razdeli na tri povsem ločene nivoje, ki znajo med seboj pametno komunicirati. Srce aplikacije je nivo Controller, ki glede na zahtevo pokliče ustrezni objekt na nivoju Model. Ta nivo mu vrne podatke, Controller pa te podatke posreduje nivoju View, ki poskrbi za uporabniku prijazni prikaz.

Koda aplikacije, ki je razvita na ta način, je lahko zelo učinkovita, pregledna in olajša kasnejše spreminjanje. Vendar pri razvoju aplikacije za evidenco nestabilnosti tal nismo v celoti uporabili tega pristopa. Za celovito uporabo te tehnologije v podjetju IGEA, kjer smo to aplikacijo razvijali, še ni bilo dovolj znanja in izkušenj. Za razumevanje in učinkovito uporabljanje vseh treh nivojev tehnologije je potrebno veliko izobraževanja in testiranja, to pa zahteva veliko časa. Zaradi časovnih okvirov, ki jih je postavil naročnik, smo se zato pri razvoju aplikacije oprijeli Oracle-ove možnosti izbire in v razvoj vključili le nekatere lažje razumljive tehnologije. Poglavitna odločitev je bila, da združimo nivoja Model in Controller in se na ta način izognemo uporabi sestavnega dela Jakarta Struts, s katerim nismo imeli dovolj izkušenj.

Osnova razvoja je bila tehnologija JSP, ki nam je bila že prej dobro poznana. Na nivoju Model smo za komunikacijo z bazo uporabili BC4J, na nivoju uporabniškega vmesnika pa UIX JSP. Tako je razvoj aplikacije ostal na dveh nivojih. Nivoja View in Controller smo združili v JSP straneh in skupaj dinamično oblikujeta rezultat na podlagi zahtevka s strani odjemalca.

BC4J

Business Components for Java (v nadaljevanju poslovne komponente) je sistem javanskih razredov, s katerimi predstavimo nivo baze v aplikaciji. Z nekaj različnimi objekti nam ustvarijo simulacijo baze na strani aplikacijskega strežnika in potem lahko iz aplikacije komuniciramo s temi objekti, ki nam vračajo podatke. Tako nam ni potrebno skrbeti za povezavo na bazo in vsakič znova pisati SQL stavke za poizvedbe po podatkih. Z dobrim poznavanjem BC4J tehnologije in njene uporabe si zelo poenostavimo komunikacijo z bazo.

BC4J okolje skrbi za to, da podatkovni nivo aplikacije razvijamo ločeno od ostalih. Pri kreiranju poslovnih komponent se lahko osredotočimo samo na to, kakšne in katere podatke bomo uporabljali v aplikaciji, kako jih bomo nabirali, kaj bomo z njimi počeli, kakšen mora biti format podatkov, itd.

Vzemimo naš primer evidence plazov. Podatki o plazu, ki jih želimo prikazati na eni spletni strani, se nahajajo v več kot dvajsetih tabelah. Osnovna tabela, ki predstavlja plazove, vsebuje le nekaj neposrednih podatkov o samem plazu (šifra, ime, lokacija, datum evidentiranja, itd). Večina podatkov je posrednih in so vezani na druge tabele. Primer takih podatkov so podatki, ki imajo predpisane vrednosti in jih imenujemo šifranti. Ko nabiramo podatke o plazu, dobimo iz osnovne tabele samo šifro (ključ), s katero nato poiščemo dejansko vrednost podatka v drugi tabeli. Pri enem šifrantu to ne predstavlja problema. Ko pa so podatki o plazu vezani na dvajset šifrantov, nam delo olajša rešitev, kot jo ponujajo poslovne komponente.

Pri ustvarjanju BC4J komponent lahko nabor podatkov iz dvajset in več tabel združimo v eno samo poslovno komponento in se nato nanjo sklicujemo kot na običajen Java objekt, ki vsebuje določene vrednosti. JDeveloper omogoča celoten razvoj poslovnih

komponent grafično. Preko vgrajenih čarovnikov izdelamo BC4J objekte in jim določimo tabele, iz katerih naj nabirajo podatke. V večini primerov znajo ti čarovniki iz baze prebrati tudi povezave med tabelami in sami oblikujejo SQL stavke, s katerimi poizvedujejo po podatkih. Tako lahko z nekaj kliki ustvarimo objekt, ki nabere vse podatke o plazu iz več kot dvajset tabel.
Arhitektura poslovnih komponent je sestavljena iz treh glavnih objektov:

- Objekt vrste Entity predstavlja točno določen objekt v bazi. V večini primerov je to tabela. Z boljšim poznavanjem BC4J okolja lahko ustvarimo tudi objekt Entity, ki je vezan na podatke, ki so shranjeni v datoteki v obliki XML ali TXT. Podatki, s katerimi upravljajo objekti vrste Entity, so predstavljeni kot atributi. Ko je tovrstni objekt vezan na tabelo, vsak stolpec v tabeli pomeni en atribut, ki ima vse lastnosti stolpca: tip podatka, dolžina podatka, povezava z drugimi podatki, itd.
- Objekti View predstavljajo poizvedbo nad podatki, ki so v enem ali več objektih tipa Entity. Objekt vrste View je tisti, ki oblikuje ustrezni SQL stavek in nabira podatke v obliki, ki smo jo določili ali pa podatke shranjuje v bazo. Tako kot objekt vrste Entity vsebuje atribute, ki predstavljajo podatke iz baze.
- Objekti vrste Application Module predstavljajo povezavo med objekti View in uporabniškim vmesnikom. Objektov vrste Application lahko ustvarimo več in jih ločimo po namembnosti. Vsakemu takemu objektu dodelimo objekte View, ki jih želimo uporabljati. V primeru evidence plazov smo ustvarili dva objekta tipa Application Module: prvi skrbi za pripravo in vnos podatkov o plazu, drugi pa za urejanje šifrantov.

Več o uporabi poslovnih komponent sledi v nadaljevanju, kjer bo predstavljeno delo z njimi.

UIX JSP

Razvoj uporabniškega vmesnika v tehnologiji UIX JSP temelji na navadnih straneh JSP (na straneh, ki poleg zapisa v jeziku HTML vsebujejo še programe v programskem jeziku Java), kamor uvozimo knjižnico komponent UIX. Te nam poenostavijo vstavljanje določenih grafičnih komponent.

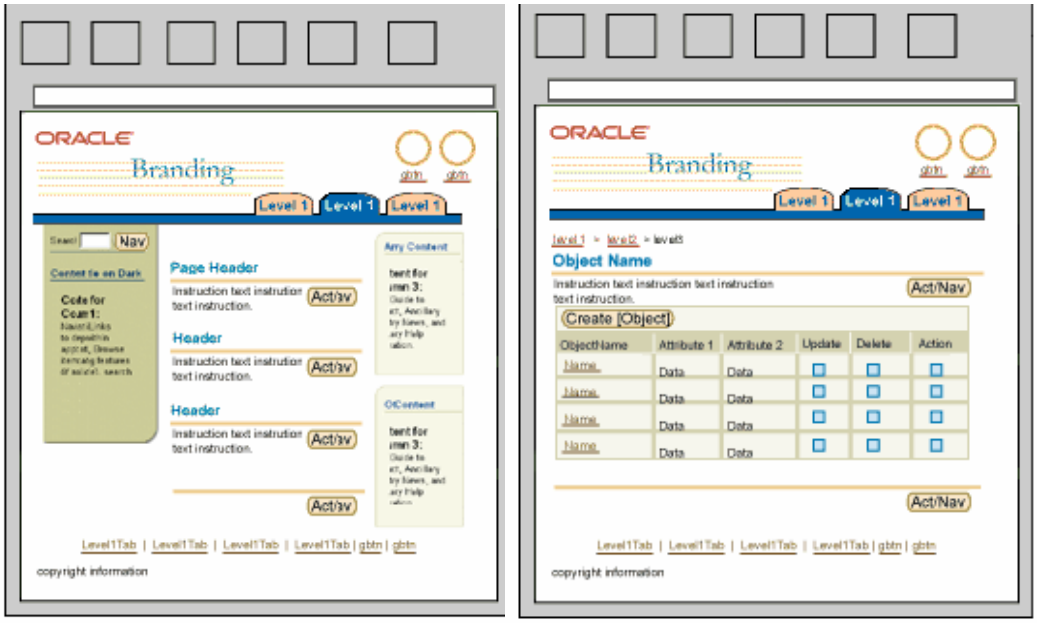
Komponente UIX pri oblikovanju uporabniškega vmesnika temeljijo na standardu Oracle BLAF (Browser Look And Feel). Standard Oracle BLAF opisuje in določa, kako naj bodo spletne aplikacije videti, kako naj delujejo ter kako naj bodo zgrajene. S tem želi Oracle vzpostaviti enoten izgled in način upravljanja s poslovnimi aplikacijami, ki so razvite znotraj njegovega okolja.

Standard BLAF nam narekuje in svetuje, kako naj gradimo spletne aplikacije. Hkrati nam ponuja bogato knjižnico grafičnih in organizacijskih komponent (UIX), s katerimi lahko hitro in uspešno sledimo standardu. Standard opisuje in določa naslednje nivoje spletnih aplikacij:

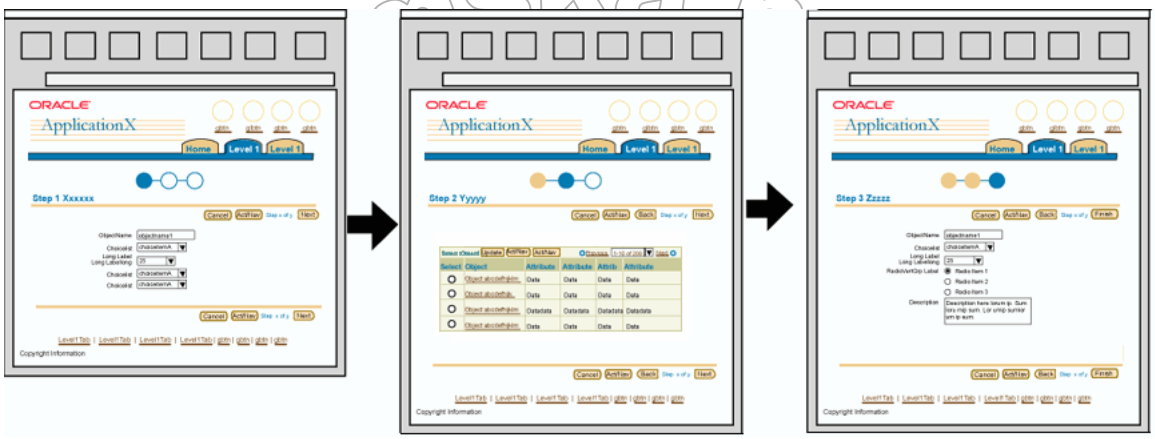
- Grafične komponente ali kombinacija grafičnih komponent: gumbi, tabele, ikone, itd.
- Izgled strani: kombinacija in razporeditev grafičnih komponent na posamezni strani.

- Zaporedje strani: kombinacija spletnih strani, ki v določenem zaporedju predstavljajo zaključen postopek: izpolnjevanje obrazcev, vnos osebnih podatkov, itd.
- Kombinacija zaporedij strani, ki skupaj predstavljajo ločen del aplikacije: iskanje artiklov, vnos novih podatkov, brisanje zapisov, itd.

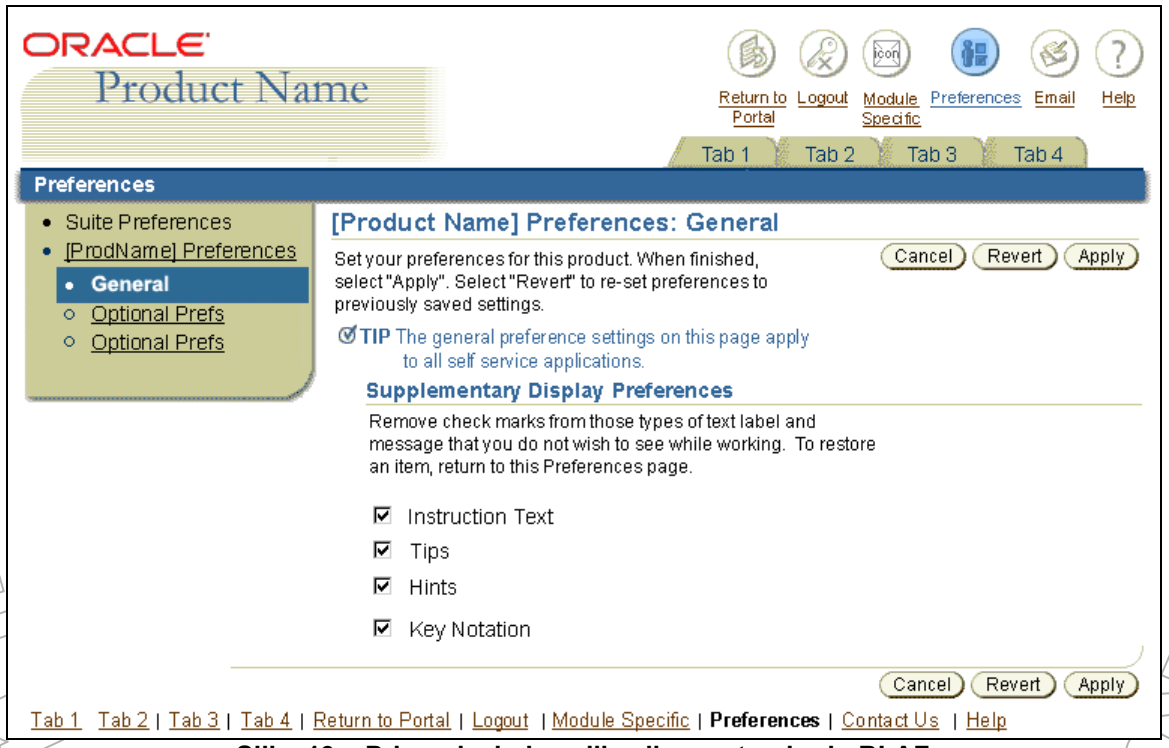
Slika 10 – Primeri grafičnih komponent po standardu BLAF



Slika 11 – Primera izgleda strani po standardu BLAF



Slika 12 – Primer zaporedja strani po standardu BLAF



Slika 13 – Primer izgleda aplikacije po standardu BLAF

Komponente UIX so zelo podobne HTML komponentam, vendar se za njimi skriva veliko več. Združujejo HTML, CSS in JavaScript razvojno kodo. Ko aplikacijo razvito s komponentami UIX postavimo na strežnik, vsaka komponenta ustvari HTML, CSS in JavaScript kodo, ki jo poznajo spletni brskalniki. Več o UIX komponentah je napisano v poglavju Implementacija aplikacije, Opis in izvedba nekaterih funkcionalnosti aplikacije.

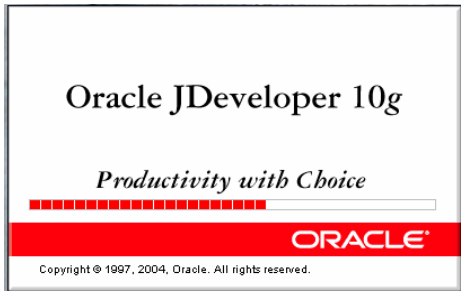
Pri razvoju aplikacije o plazovih smo v okviru uporabniškega vmesnika izkoristili enostavno postavitev zunanjega izgleda spletnih strani, ki jo omogočajo komponente UIX in v določenih primerih sledili standardu BLAF: celoten izgled strani, postavitev in izgled iskalnikov, uporaba ikon, uporaba zavihkov, itd.

3.3. Orodje JDeveloper

Za razvoj aplikacij v okolju ADF ponuja Oracle učinkovito orodje, imenovano JDeveloper. Že »J« v imenu nam da vedeti, da gre za orodje, ki temelji na Javi in je namenjeno razvoju, kjer kot programski jezik prvenstveno uporabljamo Javo.

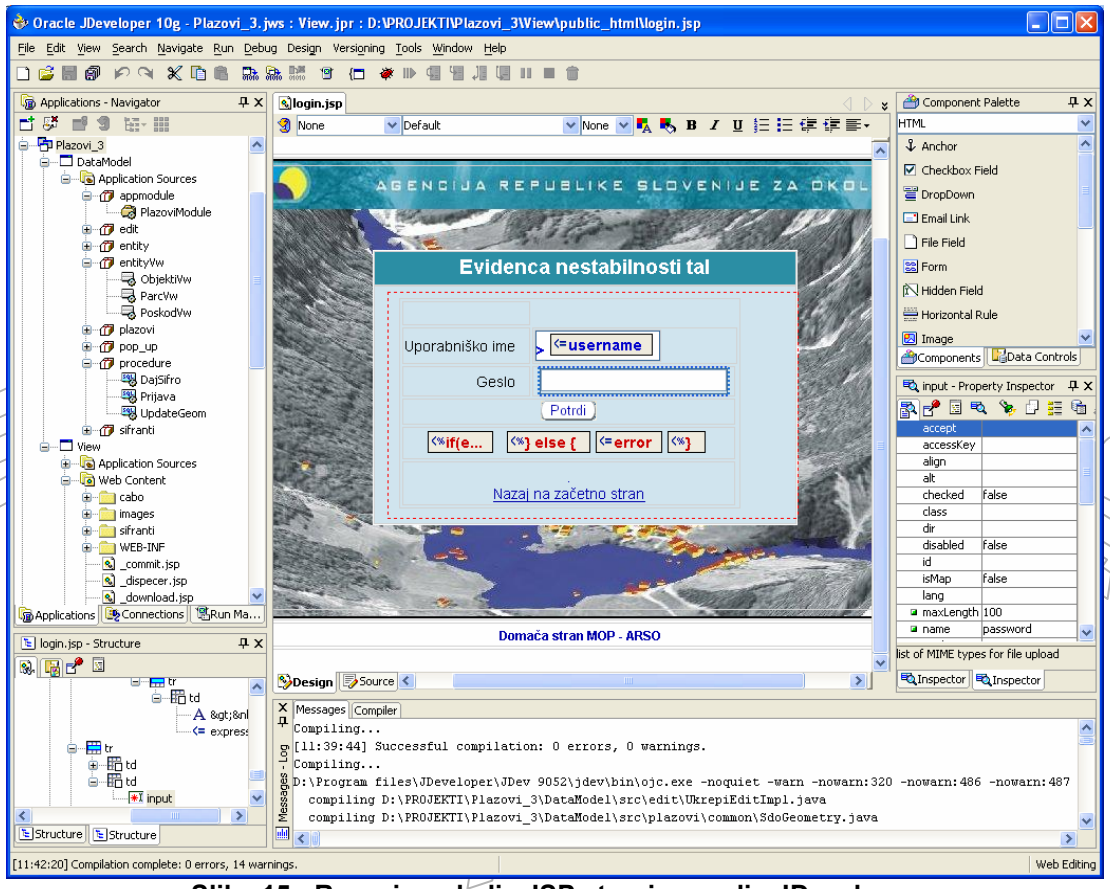
JDeveloper vsebuje različne sestavine, ki jih potrebujemo za hiter in učinkovit razvoj aplikacije:

- Grafično prikazuje arhitekturo aplikacije
- Kodo lahko v večini primerov razvijamo tudi grafično z dodajanjem komponent z miško in določanjem lastnosti preko oken in ne le neposredno v kodi
- Vsebuje obsežno pomoč in razlago okolja ADF
- Vsebuje prevajalnik kode
- Vsebuje svoj aplikacijski strežnik, tako da lahko med razvojem poženemo aplikacijo neposredno iz delovnega okolja



Slika 14 - Oracle JDeveloper 10g

Pri razvoju evidence plazov smo uporabili takrat zadnjo produkcijsko verzijo JDeveloper 9.0.5.2. Primer, kako poteka razvoj v tem orodju, je opisan kasneje v poglavju Implementacija aplikacije, Opis in izvedba nekaterih funkcionalnosti aplikacije.



Slika 15 - Razvojno okolje JSP strani v orodju JDeveloper

4. IMPLEMENTACIJA APLIKACIJE

4.1. Načrtovanje

Načrtovanje sistema je verjetno najbolj pomemben in odgovoren del razvoja aplikacije. Vsaka napaka v tem delu nam lahko samo implementacijo zelo oteži in podaljša razvoj aplikacije. Pri načrtovanju sistema evidence plazu smo se najbolj posvetili načrtovanju baze podatkov in razvoju poslovne logike spletne aplikacije. Poslovna logika določa, kako bomo predstavili podatke o plazu, kako bomo podatke spreminjali, pove, kdaj podatke lahko brišemo, kako bomo vodili zgodovino podatkov, itd.

Plaz – Dogodek
Spletna aplikacija za vodenje evidence plazov mora omogočati natančen pregled nad spreminjanjem podatkov o vsakem plazu. Ni dovolj, da se hranijo samo tekoči podatki, ampak morajo biti dostopna vsa stanja podatkov o plazu. Zato smo pri načrtovanju aplikacije vsak plaz razdelili na več dogodkov, kjer vsak dogodek predstavlja plaz v točno določenem stanju podatkov. Ko uporabnik želi spremeniti podatke o plazu, odpre nov dogodek s podatki prejšnjega dogodka, želene podatke spremeni in dogodek shrani v zgodovino izbranega plazu. Spremembo podatkov (dogodek nad plazom) lahko uporabnik spreminja toliko časa, dokler se ne odloči, da je končal in dogodek pošlje na potrditev. Skrbnik podatkov dogodek pregleda in ga sprejme ali zavrne. Če ga sprejme, postane ta dogodek izhodišče za nadaljnje spremembe, če pa ga zavrne, se aktivno stanje podatkov o plazu ne spremeni, zavrnjeni dogodek pa ostane v evidenci kot del zgodovine plazu in se ne more več spreminjati. Prav tako del zgodovine plazu postane prejšnje stanje, če je bil dogodek sprejet.
Seznam oziroma pregled plazov bo torej razdeljen na dva dela:

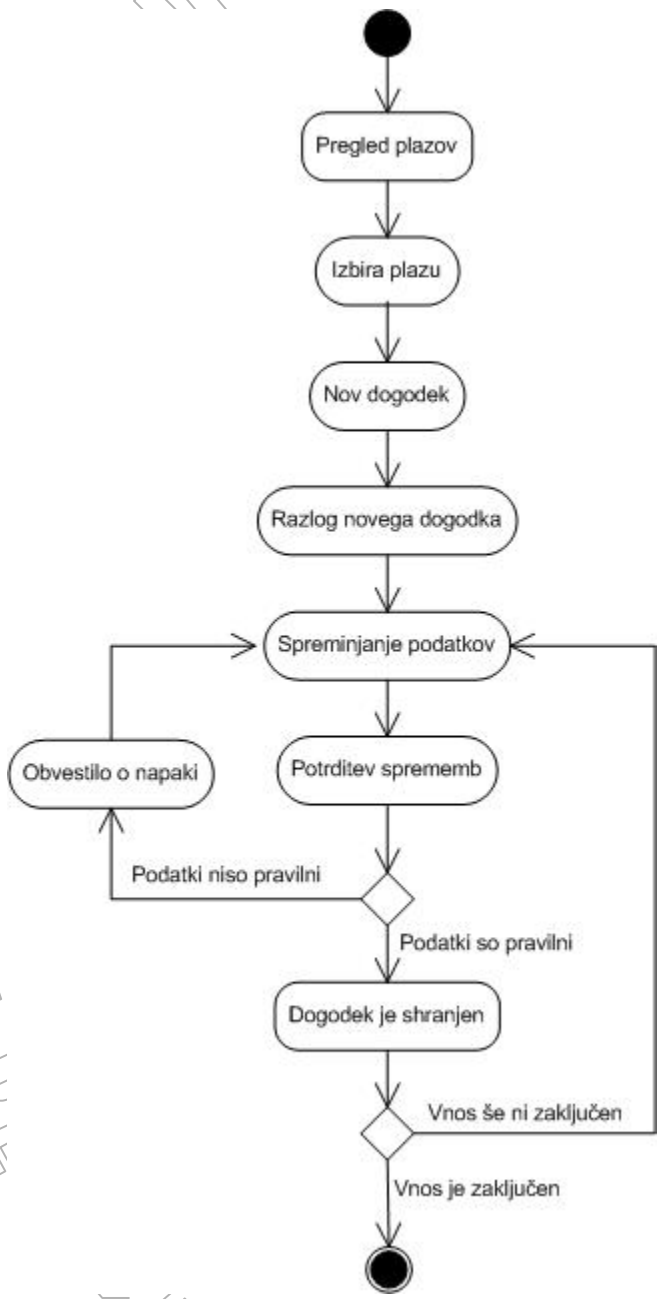
- Pregled vseh plazov, kjer bodo prikazani zadnji podatki o plazu, ki jih je skrbnik potrdil. Od tu naprej bo možnost ogleda podatkov, dodajanja novega dogodka (spreminjanje podatkov), prikaz plazu v grafičnem pregledovalniku ter skok na zgodovino podatkov.
- Zgodovina podatkov o posameznem plazu, kjer bo seznam vseh dogodkov plazu. V tem delu bo možen ogled starih podatkov, ogled in spreminjanje podatkov, ki še niso bili poslani v potrditev ter pošiljanje podatkov v potrditev in potrjevanje le-teh.

UML
Pri načrtovanju smo sledili standardu UML, ki predpisuje vrsto in način risanja diagramov. UML je jezik za specifikacijo in vizualno načrtovanje informacijskih sistemov, predvsem namenjen za gradnjo poslovnih sistemov. V njem so zbrani inženirski pristopi, ki so se izkazali za uspešne pri modeliranju velikih in kompleksnih sistemov. UML združuje različne že uveljavljene metode in je namenjen za objektno orientirano analizo in načrtovanje. Obstaja velika verjetnost, da bo v bližnji prihodnosti UML standardni jezik za modeliranje.

Primer UML diagrama smo že videli pri opisu vsebine aplikacije. To je bil diagram primerov uporabe, ki predstavlja povezavo med različnimi uporabniki in sklopi aplikacije. Ker bi bila predstavitev vseh diagramov celotnega načrta aplikacije preobširna, si kot zgled oglejmo le še tri primere diagramov aktivnosti. Diagrame smo risali z orodjem *Microsoft Visio*.

Sprememba podatkov o plazu – nov dogodek

Diagram aktivnosti:



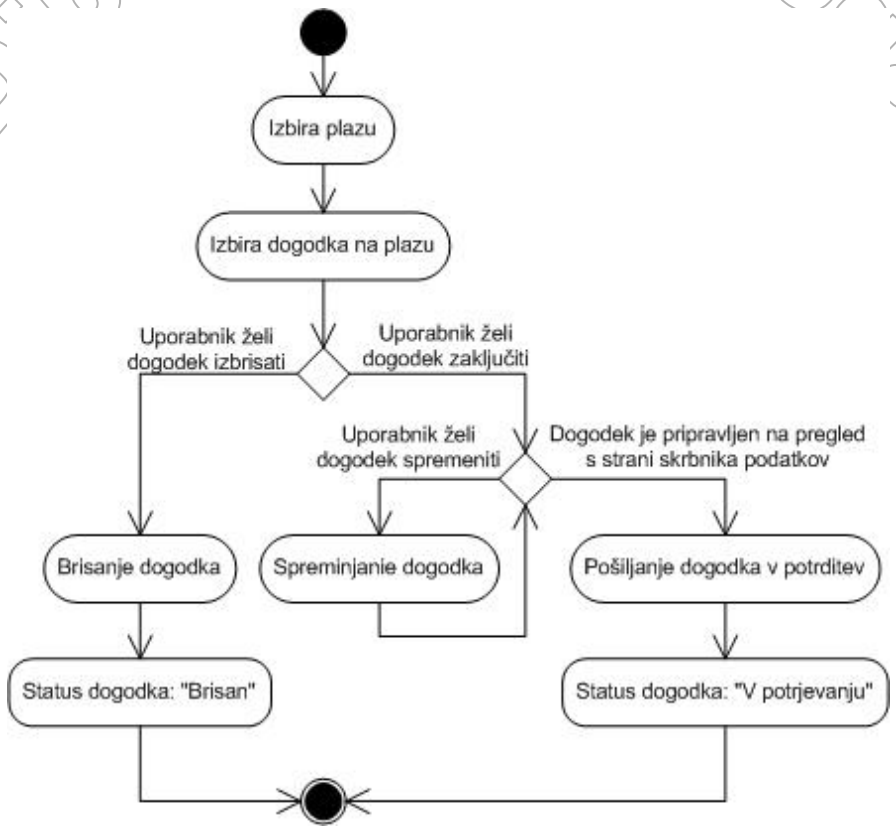
Slika 16 - Diagram aktivnosti za spremembo podatkov o plazu (nov dogodek) po standardu UML

Opis aktivnosti:

1. V tabeli plazov uporabnik izbere tistega, kateremu želi spremeniti podatke.
2. S klikom na določeno ikono sproži nov dogodek nad izbranim plazom.
3. Preden začne vnašati in popravljati podatke, vnese razlog spreminjanja podatkov.
4. Dodaja spreminja in briše podatke o plazu.
5. S klikom na gumb "Shrani" potrdi spremembe.
6. Sistem preveri pravilnost podatkov.
7. Če podatki niso pravilni, sistem uporabnika opozori na napake in ima možnost, da jih popravi.
8. Če (ko) so podatki pravilni, jih sistem shrani v bazo podatkov.
9. Podatke uporabnik lahko še spreminja in jih shrani.
10. Ko je vnesel vse podatke, je postopek spremembe podatkov zaključen.

Sprememba podatkov o plazu – zaključek dogodka

Diagram aktivnosti:



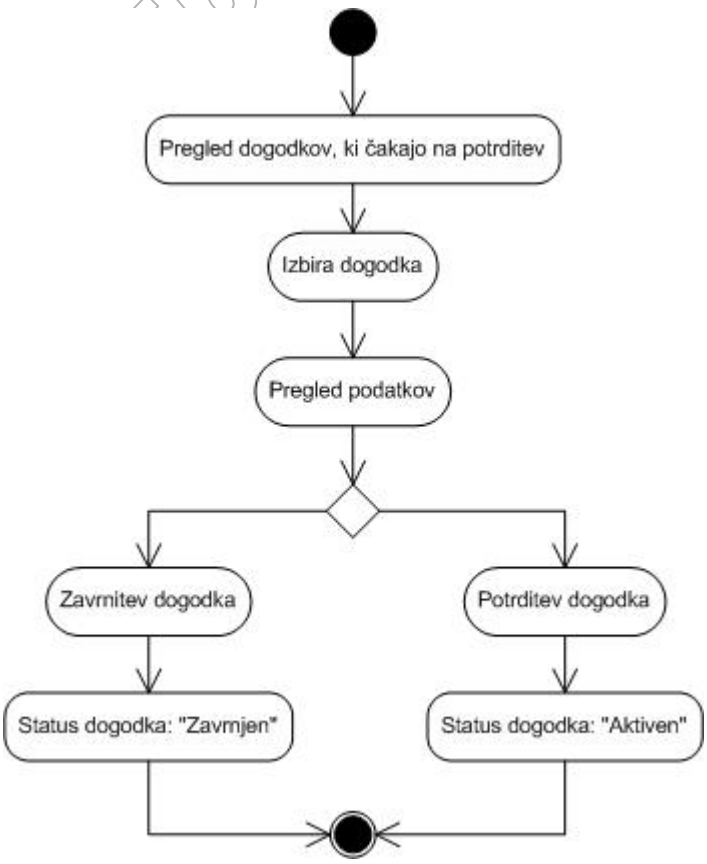
Slika 17 - Diagram aktivnosti za spremembo podatkov o plazu (zaključek dogodka) po standardu UML

Opis aktivnosti:

1. V tabeli plazov uporabnik izbere tistega, ki mu želi zaključiti dogodek.
2. Odpre seznam vseh dogodkov nad plazom in izbere določenega.
3. Če želi dogodek izbrisati, klikne ikono "Briši". Dogodek je v tem primeru zaključen in ima status "Brisan".
4. Če želi dogodek še spreminjati, klikne ikono "Spremeni", spremeni podatke in dogodek shrani.
5. Ko je dogodek končan, uporabnik klikne na ikono "Pošlji dogodek v potrditev". V tem primeru dogodka ne more več spreminjati, dogodek pa čaka na potrditev s strani skrbnika podatkov.

Potrjevanje novih dogodkov s strani skrbnika podatkov

Diagram aktivnosti:



Slika 18 - Diagram aktivnosti za potrjevanje dogodkov s strani skrbnika podatkov po standardu UML

Opis aktivnosti:

1. Skrbnik podatkov izbere dogodek iz tabele vseh dogodkov, ki čakajo na potrditev.
2. Odpre podatke o dogodku in jih pregleda.
3. Če se s podatki strinja, dogodek potrdi s klikom na ikono "Potrdi". Dogodek s tem postane aktiven in osnova za nove dogodke nad plazom.
4. Če se skrbnik s podatki ne strinja, jih s klikom na ikono "Zavrni" zavrne. Dogodek dobi status Zavrnjen.

4.2. Opis in izvedba nekaterih funkcionalnosti aplikacije

4.2.1. Prijava v aplikacijo

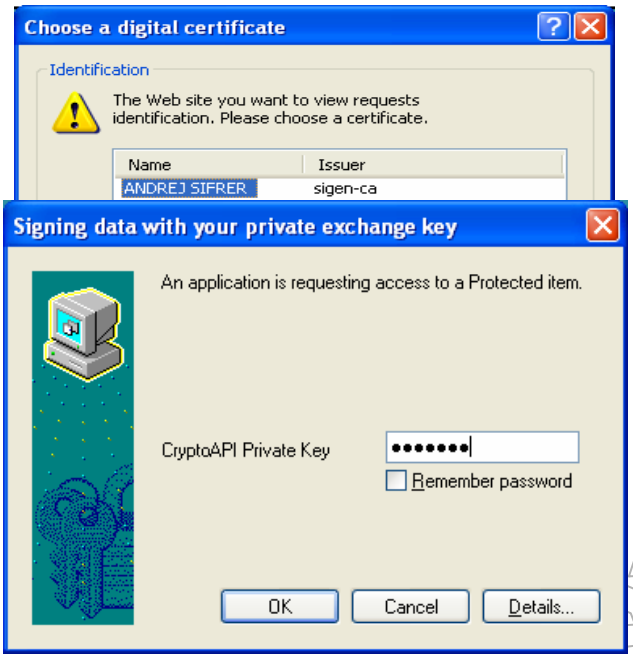
Opis

Spletna aplikacija za vodenje evidence nestabilnosti tal dostopa do zaupnih podatkov, ki javno niso dostopni (kataster stavb, zemljiški kataster, itd). Zato je velik poudarek na varnosti in omejenosti dostopa. Prijava v aplikacijo je možna le preko digitalnega potrdila, ki ga izdaja SIGEN-CA ali SIGOV-CA. S tem se učinkovito nadzoruje, kdo je dostopal do podatkov. Po prijavi z digitalnim potrdilom se na podlagi številke potrdila izvede preverjanje v bazi uporabnikov. Če obstaja uporabnik aplikacije s serijsko številko izbranega digitalnega potrdila, se uporabniku odpre prijavna stran, kjer mora vnesti še geslo za dostop do aplikacije. Ko uporabnik vnese pravilno geslo, je postopek preverjanja končan in uporabnik vstopi v aplikacijo.

Izvedba

Prebiranje podatkov iz digitalnega potrdila smo dosegli z že razvitimi metodami, ki vključujejo pravilno namestitev aplikacijskega strežnika *Apache* (uporaba *protokola SSL*[4]) ter Javine razrede in metode za delo s certifikati (*X509Certificate*).

Evidenca plazov bo v končni fazi nameščena v skupnem okolju s še nekaterimi aplikacijami, ki so že nameščene in uporabljajo enak način dostopa ter skupno bazo uporabnikov. Za branje podatkov iz digitalnega potrdila smo v primeru evidence plazov uporabili Java Servlet, ki ga uporabljajo tudi ostale omenjene aplikacije. Implementacija tako zahteva le pravilen klic URL naslova, kjer je omenjeni Servlet nameščen.



Slika 19 – Identifikacija uporabnika z digitalnim potrdilom

Klic URL naslova v primeru evidence plazov je

https://…pot do servlet-a.../**readCert**?**rUrl**=...pot do evid. plazov.../login.jsp

Zgornji klic URL naslova izvede Java Servlet z imenom **readCert.** Ta z odjemalca prebere podatke o digitalnem potrdilu in odpre URL naslov, ki je v zgornjem klicu podan kot parameter z imenom *rUrl*. Novi URL naslov vsebuje še dva parametra: serijsko

[4] SSL (Secure Sockets Layer) je protokol, ki omogoča šifrirano povezavo med strežnikom in odjemalcem

številko digitalnega potrdila z odjemalca in oznako izdajatelja, ki mora biti v primeru evidence plazov SIGEN-CA ali SIGOV-CA.
URL naslov, ki ga vrne omenjeni *Servlet,* ima naslednjo obliko

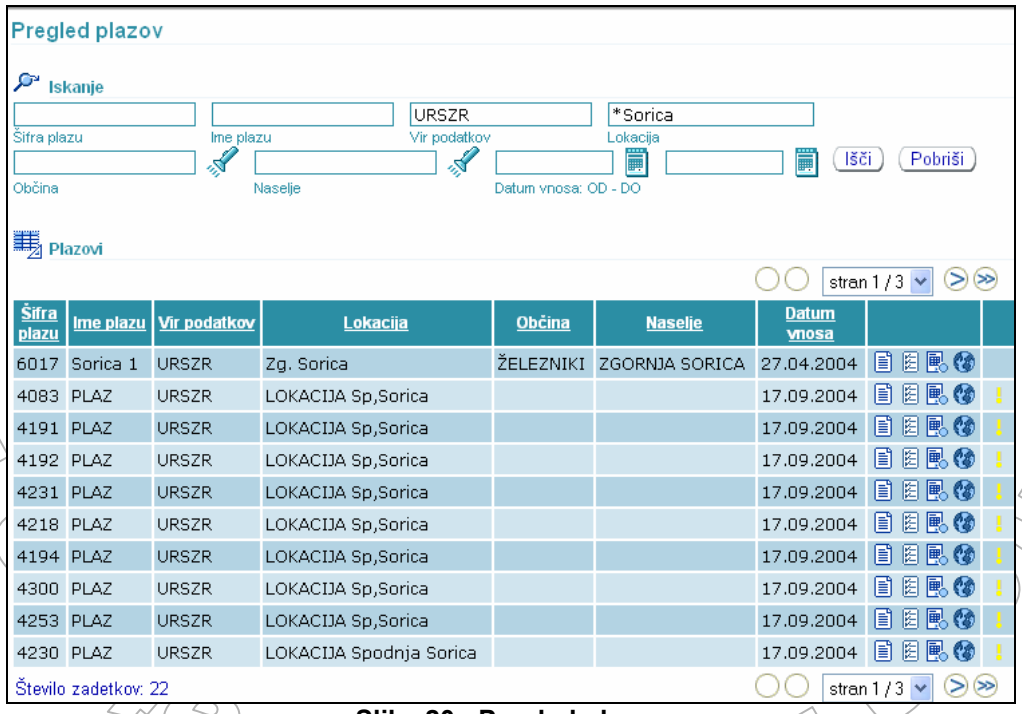
https://…pot do evid. plazov.../login.jsp?**xxx**=...serijska št. potrdila...&**ca**=…izdajatelj potrdila…

Z zgoraj opisanim postopkom od odjemalca dobimo serijsko številko uporabnikovega digitalnega potrdila in z odjemalcem vzpostavimo zaščiteno SSL sejo.

Za dokončno identifikacijo in preverjanje uporabnika nato na podlagi številke digitalnega potrdila iz baze podatkov preberemo podatke o uporabniku. Za uspešno prijavo v sistem od uporabnika zahtevamo še geslo za vstop in prijava je končana.

4.2.2. Pregled plazov

Opis
Seznam vseh plazov v Sloveniji je osnovna stran v aplikaciji in se odpre takoj po uspešno opravljeni prijavi v sistem. Dostopen je vsem uporabnikom aplikacije in predstavlja izhodišče za nadaljnje dogodke. Poleg tabele plazov osnovna stran vsebuje tudi iskalnik po tabeli, saj je vseh plazov v Sloveniji preko 2800. Iskalnik, ki omogoča iskanje po vseh osnovnih podatkih, je zato obvezen pripomoček za hitro in učinkovito delo z evidenco.



Šifra plazu	Ime plazu	Vir podatkov	Lokacija	Občina	Naselje	Datum vnosa
6017	Sorica 1	URSZR	Zg. Sorica	ŽELEZNIKI	ZGORNJA SORICA	27.04.2004
4083	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4191	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4192	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4231	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4218	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4194	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4300	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4253	PLAZ	URSZR	LOKACIJA Sp,Sorica			17.09.2004
4230	PLAZ	URSZR	LOKACIJA Spodnja Sorica			17.09.2004

Slika 20 - Pregled plazov

Osnovni podatki o plazu, ki se izpisujejo v tabeli, so šifra plazu, ime plazu, ime lokacije, občina in naselje kjer se plaz nahaja ter datum evidentiranja. Vsaka vrstica vsebuje poleg osnovnih podatkov še ikone, preko katerih prožimo aktivnosti, ki so možne nad

plazom in imamo zanje pravice. Vsi podatki in aktivnosti v tej tabeli se nanašajo bodisi na plazove, ki so v stanju aktivnosti (so bili sprejeti s strani skrbnika podatkov) ali pa predstavljajo prijavo novega plazu, ki je skrbnik še ni pregledal. Oglejmo si, kakšne aktivnosti lahko izvedemo nad plazom:

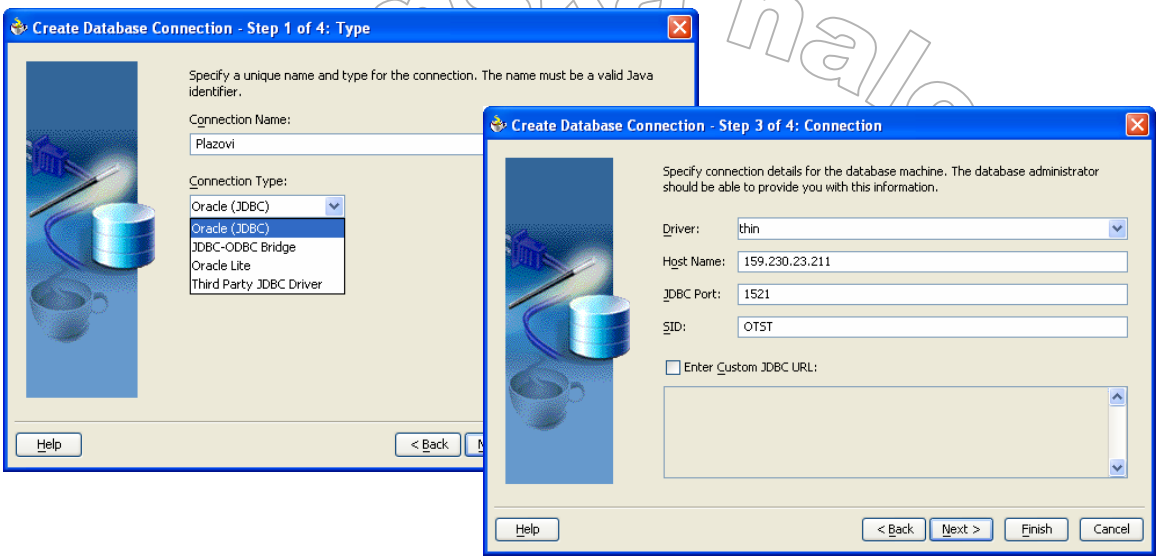
-  Prva možnost, ki je omogočena vsem uporabnikom, je pregled vseh podatkov o plazu. S klikom na to ikono se odpre okno, ki na petih straneh ponuja pregled vseh podatkov, ki se vodijo v evidenci plazov.
-  Z izbiro te ikone odpremo stran z vsemi dogodki, ki so se zgodili nad izbranim plazom. Seznam dogodkov je dostopen vsem uporabnikom in vsebuje vsa stanja podatkov od prijave do zadnje spremembe. Dokler uporabnik, ki vnaša spremembo plazu, le-te ne odda v potrditev, lahko preko te strani spremembo odpre in jo še popravlja.
-  Ta ikona omogoča spreminjanje podatkov o obstoječem plazu. To možnost imajo samo uporabniki, ki imajo pravico do spreminjanja in so prijavljeni znotraj občine (enote), ki je plaz prijavila. Podatek o občini (enoti), kateri pripada uporabnik in podatek o občini, v kateri se nahaja plaz tehnično nista povezana. To pomeni, da npr. uporabnik, ki pripada občini Kranj, lahko prijavi plaz, ki se nahaja v občini Železniki. Podatke pa nato lahko spreminjajo vsi uporabniki, ki pripadajo občini, ki je plaz prijavila, v tem primeru Kranju. Izjema so uporabniki iz skupine MOP – ARSO, ki imajo pravico spreminjati podatke za vse plazove.
-  S klikom na to ikono si ogledamo grafični prikaz lokacije plazu. Ta možnost je ponujena samo v primeru, ko plaz vsebuje podatke o koordinatah, ki sicer niso obvezni.

Izvedba

Pregled plazov predstavlja osnovni del aplikacije. Gre za izpis nekaj podatkov iz določene tabele v bazi. Poglejmo si, kako smo v okolju ADF dosegli izpis in iskanje po tabeli plazov. Opisan je postopek dela z orodjem JDeveloper od začetka, ko še nimamo povezave na bazo, do izpisa.

Prva stvar, ki jo potrebujemo za izpis podatkov, je povezava na bazo. Le-to hitro in preprosto izvedemo s čarovnikom, ki nam ga nudi JDeveloper:

- V zavihku *Connections* desno kliknemo na *Database* in izberemo *New Database Connection…*
- Odpre se nam prva stran čarovnika, kjer določimo ime in tip povezave
- Na naslednji strani določimo uporabniško ime in geslo za dostop do baze
- Na koncu vnesemo še parametre naslova, kjer se baza nahaja
- Povezavo lahko še testiramo in zaključimo postopek



Slika 21 - Čarovnik za povezavo na bazo

Naslednji korak je izdelava poslovnih komponent (*Business Components).* Uporabljamo torej nivo Model v arhitekturi MVC. Preko teh komponent bomo prišli do podatkov iz aplikacije. Ustvariti moramo tri komponente: Entity Object za povezavo na tabelo, View Object za izbiro in nabor prikazanih podatkov ter Application Module za dostop do objekta View iz programa v JSP.

- V meniju izberemo *File* in *New…*
- Med kategorijami poiščemo *Business Components* in na desni izberemo *Entity Object.*
- V čarovniku za izdelavo objekta Entity se izpiše seznam tabel v bazi.
- Objektu določimo ime in izberemo ustrezno tabelo iz baze. V našem primeru je to tabela *PLAZOVI_VW.*
- Izberemo tiste podatke iz tabele, ki jih bomo izpisovali.
- V naslednjem koraku podatkom lahko določimo nekaj lastnosti (npr. ali je podatek primarni ključ).
- V nadaljevanju še določimo, naj se istočasno ustvari še *View Object* in zaključimo postopek.
- Zdaj imamo tako Entity Object kot View Object. Potrebujemo še Application Module.
- Podobno kot prej med kategorijami izberemo izdelavo novega primerka *Application Module.*
- V čarovniku najprej določimo ime modula.
- Nato modulu dodamo prej ustvarjeni objekt View, ki ga poiščemo iz seznama objektov vrste View in zaključimo postopek.

Zdaj imamo pripravljene poslovne komponente, ki dostopajo do podatkov iz baze in jih pripravijo za izpis. Naslednji korak je izpis podatkov, ki mora delovati v povezavi z iskalnikom. V tem opisu ne bomo razlagali, kako je potrebno iskalnik sestaviti in dodati v aplikacijo, ampak bomo predpostavili, da pred vsakim izpisom plazov dobimo tudi nekaj parametrov, ki jih moramo upoštevati pri naboru podatkov. Primer: izpisati se morajo

samo plazovi, katerih ime se začne na črko »L« in so bili vneseni v evidenco leta 2004 ali kasneje.

Za dostop do objekta View moramo najprej na JSP stran postaviti modul Application Module, kar v JDeveloperju najlažje storimo preko palete komponent (*Component Palette*), ki v naboru *Business Components Connections* vsebuje komponento *Application Module*. Klik nanjo nam v kodi ustvari naslednji rezultat:

```
<jbo:ApplicationModule
   id="PlModule"
   definition="View.PlModule"
   releasemode="Stateful"
/>
```

Zdaj imamo na naši strani omogočeno neposredno povezavo do vseh objektov View, ki jih ta modul Application Module vsebuje. Do posameznih objektov vrste View nato dostopamo preko komponent DataSource, ki jih najdemo v že prej omenjenem delu palete komponent. S klikom na to vrsto komponent se nam odpre čarovnik. V njem določimo ime komponente na naši JSP strani ter še nekaj uporabnih parametrov za nabor podatkov. V našem primeru ustvarimo naslednjo komponento razreda DataSource:

```
<jbo:DataSource
   id="plazovi"
   appid="PlModule"
   viewobject="PlazoviSelect1"
   rangesize="-1"
   whereclause="IME LIKE <%=ime%>% AND DATUM >= <%=datum%>"
   orderbyclause="IME ASC"
/>
```

Pomen posameznih parametrov:

- *id*: ime, s katerim bomo na strani dostopali do objekta DataSource
- *appid*: ime modula Application Module, v katerem se nahaja objekt View
- *viewobject*: ime objekta View znotraj modula Application Module
- *rangesize*: število zapisov iz tabele, ki jih vrne posamezna poizvedba: -1 pomeni vse zadetke
- *whereclause*: stavek s pogoji, ki se upoštevajo pri poizvedbi. V zgornjem primeru smo sestavili *where* stavek tako, da smo vrednosti spremenljivk *ime* in *datum* iz java kode v JSP kodo vstavili s pomočjo skripte.
- *orderbyclause*: stavek s pogoji, ki določa vrstni red pri izpisu

S tem smo na naši strani dobili objekt tipa DataSource, ki dostopa do podatkov, ki ustrezajo naštetim parametrom. Mislimo si lahko, da objekt vsebuje vse podatke o vseh plazovih, ki ustrezajo pogoju, ki ga določa stavek where in sicer tiste podatke o posameznem plazu, ki smo jih določili pri sestavljanju objekta View. Za izpis teh podatkov imamo dve možnosti: lahko uporabimo nekaj komponent BC4J UIX, ki same oblikujejo tabelo za izpis, ali pa »na roke« z Javo in HTML-jem izpišemo podatke iz omenjenega objekta. Ker nam prva možnost nekoliko omejuje svobodo pri izpisu, smo se pri razvoju naše aplikacije odločili za drugo možnost.

Izpis podatkov iz objekta DataSource izgleda takole:

```
<%
RowSet rowSet = plazovi.getRowSet();                    #1
rowSet.setRangeSize(10);                                #2
rowSet.setRangeStart(0);                                #3
Row[] rows = rowSet.getAllRowsInRange();                #4
%>
<table>
  <tr class="tableHeader">
    <th>Šifra plazu</th>
    <th>Ime plazu</th>
    <th>Občina</th>
    <th>Naselje</th>
    <th>Datum vnosa</th>
  </tr>
  <%
  for(int i = 0; i < rows.length; i++) {
    %>
    <tr>
      <td><%= rows[i].getAttribute("Sif")%></td>        #5
      <td><%= rows[i].getAttribute("Ime")%></td>
      <td><%= rows[i].getAttribute("Obcina")%></td>
      <td><%= rows[i].getAttribute("Naselje")%></td>
      <td><%= rows[i].getAttribute("Datum")%></td>
    </tr>
    <%
  }
  rowSet.closeRowSet();                                 #6
%>
</table>
```

Za lažje razumevanje smo JSP kodo pisali odebeljeno HTML kodo pa normalno.

Razložimo, zakaj gre pri kodi JSP:
#1: iz objekta tipa DataSource (plazovi) ustvarimo objekt razreda RowSet, ki vsebuje vrstice iz tabele. Posamezna vrstica ustreza zapisu v bazi.
#2: določimo, koliko vrstic (zapisov) bomo izpisali (v tem primeru 10)
#3: določimo, pri kateri vrstici bomo začeli izpisovati
#4: sestavimo tabelo z zapisi (vrsticami), določenimi s prejšnjimi ukazi
#5: v posamezne stolpce (<td>) izpišemo ustrezne podatke iz vrstice
#6: zapremo objekt rowSet

Če smo v spremenljivki ime imeli zapisano "L" in v spremenljivki datum "1.1.2004", smo torej s to kodo sestavili tabelo v HTML, ki vsebuje podatke o prvih 10 (ali manj) plazovih, katerih ime se prične z L in so bili vnešeni leta 2004 ali kasneje. V tabeli so v stolpcih šifra plazu, ime plazu, občina, naselje in datum vnosa.

Nekaj parametrov, ki jih določimo objektu iz razreda DataSource, lahko določimo tudi objektu tipa RowSet. Denimo, da smo parameter *whereclause* v objektu DataSource določili takole:

```
whereclause="DATUM >= <%=datum%>"
```

Še vedno pa bi radi v objektu rowSet imeli le zapise, ki ustrezajo pogoju, kot ga je določal prejšnji stavek *where*. Zato v sami JSP kodi na objektu rowSet iz razreda RowSet izvedemo metodo *setWhereClause* (`rowSet.setWhereClause("IME LIKE <%=ime%>");`). S tem smo v objektu rowSet spet dobili iste vrstice, kot prej.

Razlika med tem, ali določimo stavek where že objektu DataSource, ali pa kasneje na objektu tipa RowSet izvedemo omenjeno metodo, je v tem, da v prvem primeru že iz baze dobimo omejen izpis, v drugem primeru pa izpis omejimo šele v aplikaciji. Drugi primer uporabimo samo, kadar želimo na isti strani imeti več različnih izpisov istega objekta tipa DataSource. Takrat iz baze črpamo tiste zapise, ki predstavljajo unijo vseh izpisov, samo izbiro pri posameznem izpisu pa potem dosežemo z uporabo metode setWhereClause na objektu tipa rowSet

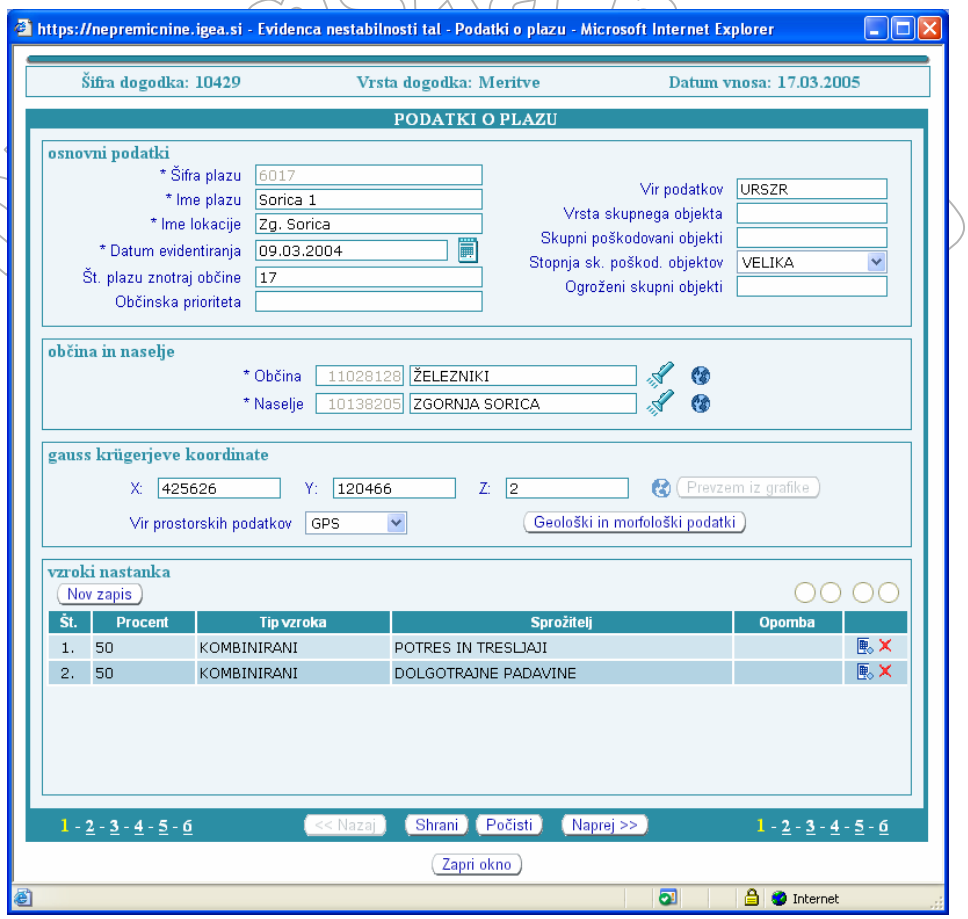
S tem smo končali izpis tabele plazov v okolju ADF. Za popolno uporabnost takega pregledovanja moramo seveda dodati še ustrezni iskalnik, s katerim določamo iskalne parametre, ter tabeli izpisov dodati gumbe za navigacijo po tabeli naprej in nazaj, vendar se pri tem tukaj ne bomo ustavljali.

4.2.3. Podatki o plazu

Opis

Obrazec za vnašanje in pregled podatkov o plazu je povzet iz standardiziranega obrazca »Evidenčni list prijave plazu«. Ker je podatkov preveč, da bi jih prikazovali vse na eni strani, so razdeljeni na šest podstrani in okno, ki ločuje geološke in morfološke podatke od ostalih.

- Prva stran vsebuje osnovne podatke o plazu: šifra, ime, lokacija, datum evidentiranja plazu, občina in naselje, kjer se plaz nahaja, koordinate plazu, vzroke nastanka, vir podatkov, itd.
- Druga stran vsebuje seznam poškodovanih in ogroženih objektov. Te delimo na štiri skupine: ceste, stavbe, javna infrastruktura ter zemljišča. Izpisujejo se v tabelah. S klikom na posamezno vrstico (z izbiro ustreznega objekta) se v tabeli pod objekti izpišejo vse poškodbe in ogroženosti izbranega objekta. Vsak objekt vsebuje tudi tabelo ukrepov. Do slednje pridemo preko ikone na koncu vsake vrstice v tabeli.
- Tretja stran vsebuje prizadete parcele ter podatke o stanju plazu, hitrosti plazenja, dimenzijah plazu, vrsti plazenja, itd. Prizadete parcele se izpisujejo v tabeli. Vsaka parcela vsebuje seznam ukrepov. Do seznama pridemo preko ikone na koncu vrstice.
- Četrta stran je namenjena dokumentaciji in fotografijam v zvezi z izbranim plazom. Podatki so izpisani v dveh tabelah in omogočajo shranjevanje dokumentov in fotografij v digitalni obliki.
- Peta stran vsebuje seznam spletnih naslovov, ki so povezani z izbranim plazom ter polje za vnos dodatnega opisa plazu.
- Šesta stran vsebuje podatke o škodi, sanacijah in ukrepih na plazu.
- Do okna, ki vsebuje geološke in morfološke podatke kot so slojevitost podlage, sestavina plazu, hidrogeološki opis, vsebnost vode, itd., pridemo preko gumba na prvi strani.



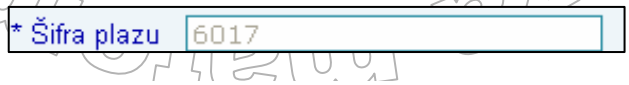
Slika 22 - Prva stran podatkov o plazu

Izvedba

Vnos podatkov je možen na več različnih načinov. Ti so odvisni od tega, kakšni ti podatki so, če so del šifranta ter v kakšnem razmerju so s plazom.

Najbolj osnovni so podatki, ki so s plazom v razmerju ena – ena (en plaz vsebuje en podatek) in nimajo predpisanih vrednosti. Za vnos teh podatkov smo uporabili komponente UIX. Uporaba teh nam v določenih primerih zelo skrajša pisanje kode. Oglejmo si nekaj primerov teh komponent, ki smo jih uporabili:

```
<uix:messageTextInput
    prompt="Šifra dogodka"
    name="sifraDog"
    text="<%=plaz.getSifraDgk()%>"
    required="no"
    disabled="true"
    styleClass="message240"
    ...
/>
```



Slika 23 - uix:messageTextInput

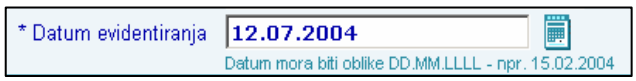
Komponenta *messageTextInput* predstavlja navadno vnosno polje z oznako pred njim. Nekatere lastnosti:

- prompt: napis, ki se izpiše pred poljem,
- name: ime, pod katerim se prenese podatek,
- text: vrednost polja,
- required: ali je polje obvezno za vnos,
- disabled: ali je trenutno polje aktivno,
- styleClass: izgled komponente,

Obvezno moramo navesti vsaj lastnost name, ostale pa lahko spustimo. V lastnosti lahko vrednosti vnesemo tudi z uporabo JSP ukazov. V našem primeru smo pri lastnosti text nad objektom plaz uporabili metodo getSifraDgk(), ki je vrnila avtomatično generirano šifro dogodka. Ker smo z nastavitvijo lastnosti disabled na no uporabo komponente onemogočili, uporabnik šifre plazu ne more sam spremeniti. To je vidno tudi tako, da je izpisano besedilo sivkasto obarvano.

Če bi želeli isto doseči z običajno kodo HTML, bi za to potrebovali vsaj dve komponenti in nekaj dodatnih JavaScript ukazov.

```
<uix:messageDateField
   prompt="Datum evidentiranja"
   name="datumVnosa"
   value="<%=plaz.getDatumVnosa()%>"
   columns="25"
   tip="Datum mora biti oblike DD.MM.LLLL - npr. 15.02.2004"
   styleClass="message240" >
            <uix:onSubmitValidater>
               <uix:date pattern="dd.MM.yyyy" />
            </uix:onSubmitValidater>
</uix:messageDateField>
```



Slika 24 - uix:messageDateField

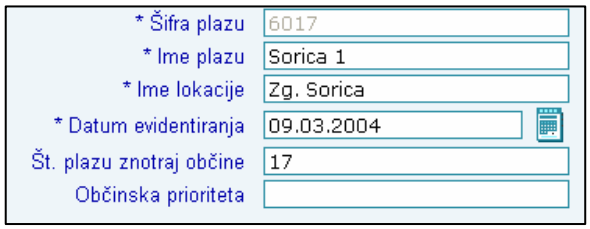
Komponento *messageDateField* uporabljamo za vnos datuma. Pred datumskim vnosnim poljem je ustrezen napis, za njim pa ikona, ki v novem oknu odpre koledar, kjer lahko izberemo datum, ki se nato prenese v vnosno polje. Komponenta *messageDateField* ima vse lastnosti, ki jih ima komponenta *messageTextInput*. Poleg tega imamo možnost, da znotraj komponente lahko definiramo format, v katerem naj se vnaša datum. Če je tako vnosno polje del forme, se pred pošiljanjem podatkov format tudi preveri, Če ni pravilen, nas brskalnik o tem obvesti.

V tem primeru lahko vidimo, da so komponente UIX zelo podobne elementom v HTML. Lahko so samostojne, lahko jim določamo lastnosti, lahko pa tudi vsebujejo druge komponente. V zgornjem primeru je komponenta *onSubmitValidator* del komponente *messageDateField*, sama pa vsebuje še komponento *date*. Vse tri skupaj s formo tvorijo polje za vnos datuma in preverjanje le-tega. Struktura komponent UIX je torej takšna:

```
<uix:ImeKomponente
    lastnost1=vrednost1
    lastnost2=vrednost2
   ...
 >
   <..."podrejene" komponente.../>
</uix:Razred>
```

V nadaljevanju sledi še nekaj primerov komponent UIX, ki so t.i. "zbirniki" in vsebujejo ter razporejajo nekatere ostale komponente.

```
<uix:labeledFieldLayout
   columns="1"
   labelWidth="200"
   fieldWidth="250"
   ...
>
... komponente ...
</uix:labeledFieldLayout>
```



Slika 25 - uix:labeledFieldLayout

Komponenta labeledFieldLayout služi temu, da znotraj nje navpično postavimo druge komponente. Najbolj je uporabna skupaj s komponentami namenjenimi vnosu podatkov. Določimo lahko v koliko stolpcev naj se vsebina postavi, koliko naj bodo široki napisi, koliko vnosna polja, itd.

Posebna vrsta podatkov so šifranti. To so podatki, ki imajo točno določeno zalogo vrednosti. Za izpis vrednosti šifranta v padajočem seznamu imamo v okolju UIX dve komponenti. To sta messageChoice in option. Komponenta messageChoice je zbirnik, v katerega vpišemo več komponent tipa option.

```
<uix:messageChoice
   prompt="Vrsta dogodka"
   name="vrstaDgkSif"
   styleClass="message240"
>
   <% for(int i = 0; i < dgkRows.length; i++) { %>
      <uix:option
         text="<%=dgkRows[i].getAttribute("Sif")%>"
         value="<%=dgkRows[i].getAttribute("Opis")%>"
      />
   <% } %>
</uix:messageChoice>
```

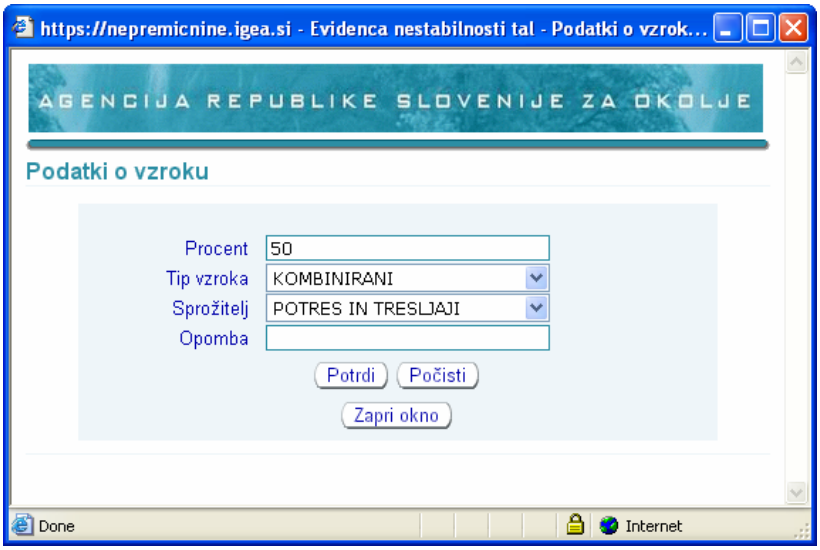
messageChoice predstavlja padajoči seznam z besedilom pred njim, imenom, izgledom, itd. Komponenta *option* predstavlja eno od vrednosti seznama, ki ji določimo ime, ki se izpiše ter vrednost, ki se prenese po potrditvi vnosa. V zgornjem primeru smo vrednosti padajočega seznama prebrali iz baze in si pomagali s kodo v Javi in skriptnimi ukazi.

Podatki o plazu vsebujejo tudi podatke, ki so s plazom v razmerju ena – več (en plaz vsebuje več podatkov določene vrste). Ti podatki so v večini primerov sestavljeni iz več drugih podatkov. To je implementacijo evidence plazov naredilo veliko bolj zapleteno. Takšno razmerje med podatki imenujemo *Master – Detail* oziroma, ker gre za razmerje

med tabelami podatkov, *tabele Master – Detail*. Tak podatek je na primer vzrok nastanka plazu. Ker ima en plaz lahko več vzrokov nastanka, so ti vzroki shranjeni v svoji tabeli. Vsak vzrok je povezan s plazom preko enolične številke plazu.

Pri implementaciji takih vrst podatkov smo na osnovno masko postavili tabelo, v katero lahko dodajamo in spreminjamo podatke preko novega okna in nove maske. Omenjeni vzrok nastanka plazu je potem videti takole:

Slika 26 – Primer vnosa Master – Detail podatkov (tabela vzrokov nastanka)



Slika 27 - Primer vnosa Master – Detail podatkov (vnos novega vzroka)

Sama implementacija je podobna seznamu plazov, kjer imamo plazove izpisane v tabeli in jih lahko dodajamo in spreminjamo preko nove maske.

4.2.4. Spreminjanje statusa

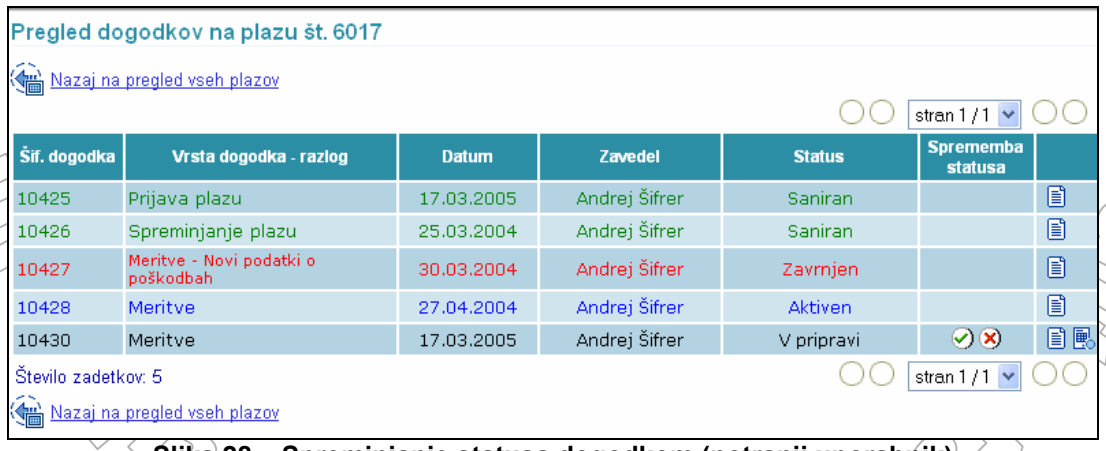
Opis

Vsako spremembo podatkov mora skrbnik podatkov (MOP - ARSO) potrditi, zato smo dogodkom nad plazovi dodali polje status, ki določa v katerem stanju so podatki. Dogodek je lahko v naslednjih stanjih:

- V pripravi: podatki so shranjeni v bazi, vendar uporabnik še ni zaključil z vnašanjem podatkov
- Brisan: uporabnik je shranil nove podatke v bazo nato pa se je odločil, da podatki niso v redu in preklical nadaljnje spreminjanje
- V potrjevanju: uporabnik je zaključil z vnašanjem ali spreminjanjem podatkov in jih je poslal v potrditev s strani skrbnika
- Aktiven: v tem statusu je lahko največ en dogodek nad istim plazom, saj predstavlja zadnjo spremembo, ki jo je odobril skrbnik; ta dogodek je izhodišče za nove spremembe podatkov
- Saniran: podatki so bili sprejeti, vendar obstajajo novejši podatki, ki so v stanju aktivnosti
- Zavrnjen: sprememba je bila poslana v potrditev, vendar jo je skrbnik zavrnil

Izvedba

Spreminjanje statusa dogodkom smo dodali med pregled dogodkov. Vsakemu dogodku smo dodali dve ikoni, s katerima uporabnik dogodek potrdi ali zavrne, kar pa ima za različne uporabnike različen pomen. Če je to zunanji uporabnik (npr. občina), klik na potrditev pomeni, da je zaključil z vnosom podatkov in s tem pošilja dogodek (podatke) na potrditev skrbniku podatkov. Če izbere zavrnitev podatkov, pomeni, da je zaključil z vnosom, podatki pa niso pravilni in se jim določi status Brisan. Notranjemu uporabniku (skrbniku podatkov) pa ikoni služita za sprejemanje oziroma zavračanje dogodkov (podatkov), ki jih je v potrditev poslal zunanji uporabnik. Če je avtor podatkov sam skrbnik podatkov, ni pošiljanja v potrditev, ampak lahko svoje podatke takoj sprejme ali zavrne.

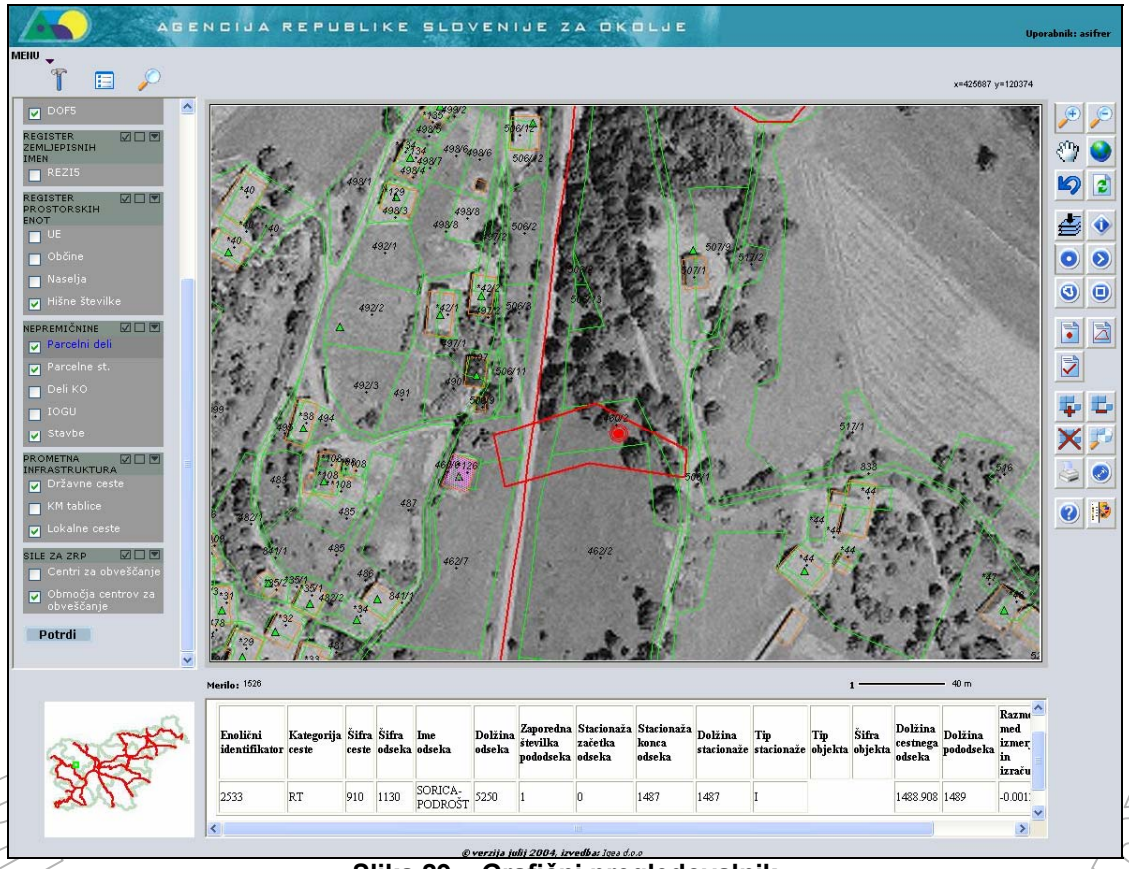


Slika 28 – Spreminjanje statusa dogodkom (notranji uporabnik)

4.2.5. Grafični pregledovalnik, komunikacija z evidenco

Grafični pregledovalnik omogoča prikazovanje lokacije plazu, objektov, parcel, itd na zemljevidu in se odpre v ločenem oknu takoj po uspešno opravljeni prijavi v aplikacijo. Če grafični del kadarkoli zapremo, ga lahko odpremo z gumbom v desnem zgornjem kotu vsake strani.

Glavni (osrednji) del grafičnega pregledovalnika je razvit kot *Java Applet* in je postavljen v HTML stran. Legenda, iskalniki in gumbi so v celoti del spletne strani in z applet-om komunicirajo s pomočjo jezika JavaScript. Pregledovalnik omogoča ogled zemljevida preko več različnih slojev, ki se lahko prekrivajo. Seznam slojev v grafiki je izpisan v levem meniju in uporabnik ima možnost vključevanja in izključevanja le-teh.



Slika 29 – Grafični pregledovalnik

Grafični pregledovalnik deluje ločeno od evidence in je tehnološko povsem samostojna spletna aplikacija. Zato neposredna komunikacija med njima ni mogoča. Zaradi varnosti delovanja spletnih aplikacij, na strani strežnika ni možno iz ene aplikacije vstopati v drugo. Edina povezava je možna na strani uporabnika, kjer z JavaScript jezikom lahko dostopamo do vseh oken, ki so odprta na odjemalcu. Vendar tudi v tem primeru varnostne nastavitve brskalnikov vedno bolj omejujejo komunikacijo med posameznimi okni brskalnika. Do vsebine drugega okna lahko s funkcijami v JavaScriptu dostopamo le, če je bila vsebina poslana iz istega strežnika.

Komunikacija med evidenco plazov in njenim grafičnim pregledovalnikom poteka v obe smeri:

- vsak plaz, cesta, stavba, parcela, občina ali naselje v evidenci vsebuje povezavo do grafičnega prikaza. To pomeni, da moramo iz evidence sprožiti ustrezno funkcijo v grafičnem pregledovalniku. To smo dosegli tako, da s funkcijami, napisanimi v skriptnem jeziku JavaScript kličemo funkcije applet-a v drugem oknu.
- Izbira cest, stavb in parcel za vnos v evidenco je možen tudi iz grafičnega pregledovalnika. Tukaj smo si pomagali z bazo. Vsak izbor objektov v grafiki se zapiše v bazo, od koder potem podatke uvozimo v evidenco.

Primer: Potek nabora parcel iz grafičnega pregledovalnika v evidenco plazov:

- Ko iz evidence odpremo grafični pregledovalnik, obe aplikaciji shranita podatek o uporabniku in enolični identifikator za trenutno sejo.
- Na tretji strani podatkov o plazu kliknemo na ikono za nabor parcel iz grafike  . V ospredje se nam postavi okno grafičnega pregledovalnika. Slika se poveča in postavi tako, da je plaz na sredini ekrana.
- Med ikonami na desni strani pregledovalnika izberemo ikono za nabor objektov  .
- Z miško kliknemo na parcele, ki jih želimo nabrati.
- S klikom na ikono  potrdimo nabor.
- Nabor v grafičnem delu aplikacije je s tem končan. Podatki o nabranih parcelah so se začasno shranili v bazo skupaj s podatkom o uporabniku in enoličnim identifikatorjem za trenutno sejo.
- Postavimo se nazaj v evidenco plazov in kliknemo na gumb *Prevzem iz grafike*. V tabelo prizadetih parcel se dodajo vsi podatki o nabranih parcelah. S klikom na gumb *Potrdi* se ti podatki tudi shranijo na pravo mesto v bazi.

Parcele, stavbe ali ceste, ki so del podatkov o plazu, so v grafičnem pregledovalniku označene in se ločijo od ostalih. Tako imamo pregled nad nabranimi objekti tudi v grafičnem delu. Če v evidenci določen objekt odstranimo, se sprememba pozna tudi v grafičnem pregledovalniku.

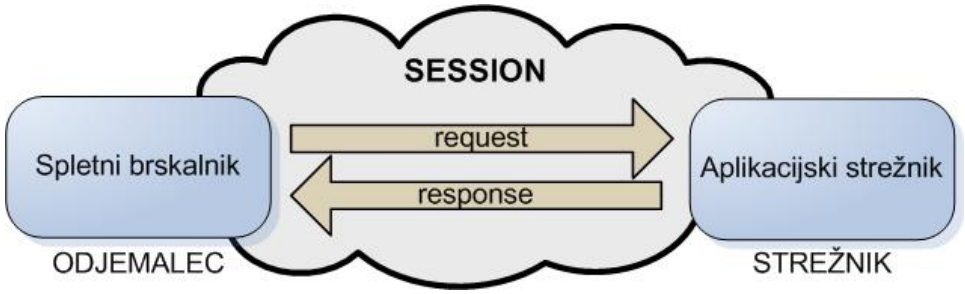
4.3. Shranjevanje tekočih sprememb v objektu tipa Session

Ko uporabnik vnaša ali popravlja podatke o plazu, se posamezni podatki o plazu ne smejo shranjevati v bazo samodejno in sproti, ampak mora uporabnik, ko konča z vnašanjem, sam potrditi vnos in s tem vse vnesene podatke naenkrat shraniti v bazo. S tem preprečimo, da bi se v bazo zapisovali nepopolni podatki. Uporabnik se med samim vnašanjem podatkov namreč lahko odloči, da podatki niso pravilni in postopek vnašanja prekine. Prav tako lahko med vnašanjem pride do tehničnih težav (izpad elektrike, okvara operacijskega sistema, itd), ki postopek prekinejo, ko še ni končan.

Ker je forma za vnašanje podatkov o plazu razdeljena na več strani, se morajo vneseni podatki pred prehodom na naslednjo stran začasno shraniti na neko mesto. Najbolj primerno mesto za začasno shranjevanje podatkov je javanski objekt tipa *Session*.

4.3.1. O objektu Session

Ko uporabnik vstopi v aplikacijo, začne t.i. sejo (*session*), ki predstavlja komunikacijo med odjemalcem in strežnikom. Seja je sestavljena iz zahtevkov (*request*), ki jih pošilja odjemalec strežniku in odgovorov (*response*), ki jih pošilja strežnik odjemalcu.



Slika 30 – Seja med spletnim brskalnikom in aplikacijskim strežnikom

V Javi vse tri omenjene pojme predstavimo z objekti tipa *Session*, *Request* in *Response*: te ustvari aplikacijski strežnik avtomatično. Objekt tipa Session (v nadaljevanju objekt session) se ustvari za vsakega uporabnika posebej, ko ta vstopi v aplikacijo. Obstaja toliko časa, dokler uporabnik aplikacije ne zapre ali pa poteče čas veljavnosti objekta session. Objekta Request in Response predstavljata po en zahtevek in odgovor. Z vsakim novim zahtevkom se ustvari nov objekt, podatki o prejšnjem pa niso več dostopni.

V objekt tipa Request se shranijo podatki (parametri), ki jih odjemalec pošlje v zahtevku strežniku. Če npr. odjemalec pošlje zahtevek oblike …/*stran.jsp?parameter=vrednost* se v Request shrani podatek *'vrednost'* (tipa String) pod imenom *'parameter'*. Do njega na strani strežnika pridemo z metodo *request.getParameter("parameter")*.

V objekt tipa Response strežnik shrani podatke (parametre), ki jih želi poslati odjemalcu.

Objekt session obstaja tekom celotne seje. Tako imamo na voljo objekt, v katerega lahko kadarkoli med sejo shranimo podatke in jih iz njega prebiramo. V objekt session shranjujemo podatke, ki jih želimo imeti na voljo kjerkoli znotraj aplikacije. Med najbolj

pogostimi tovrstnimi podatki so podatki o uporabniku (naziv uporabnika, uporabniško ime, geslo, itd). Ti nam omogočajo, da za vsako akcijo v aplikaciji vemo, kdo jo je sprožil. Samodejno ustvarjanje objekta session lahko tudi preprečimo, če vemo, da ga ne bomo potrebovali. S tem nekoliko pridobimo na hitrosti delovanja aplikacije.

Objekta tipa Request in Response lahko hranita samo podatke tipa String, v objekt tipa Session pa lahko shranimo objekte poljubnega tipa, zato smo ga izkoristili za shranjevanje podatkov o plazu pred končnim zapisom v bazo.

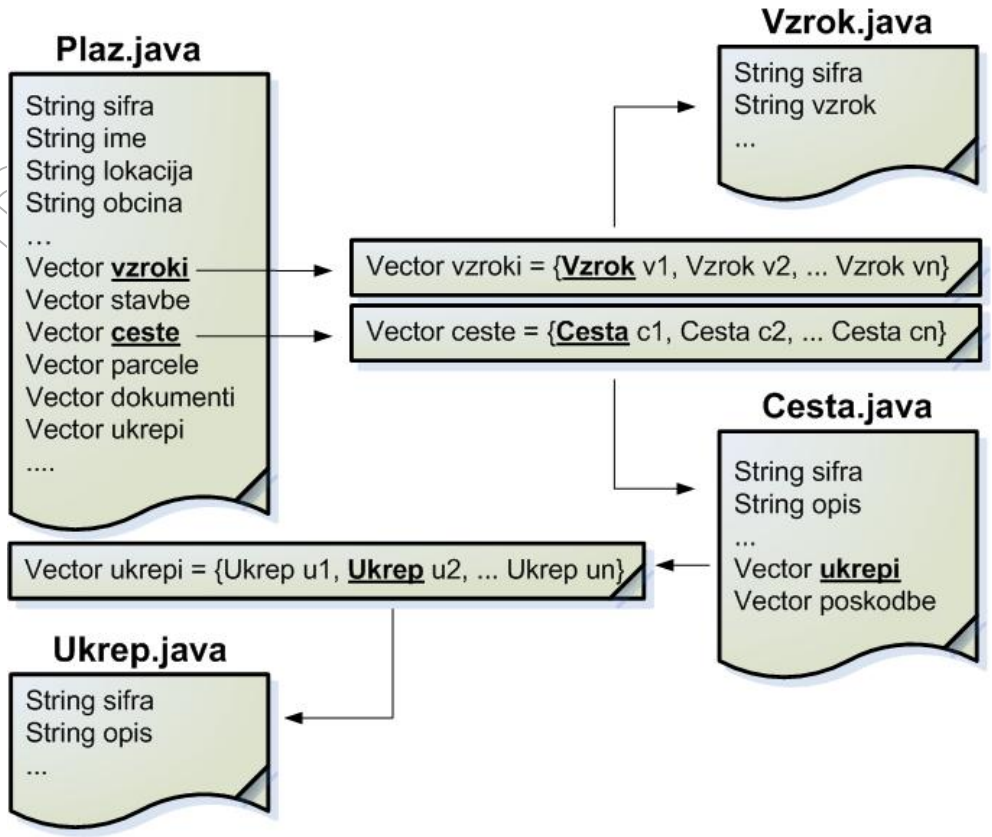
4.3.2. Struktura objektov s podatki o plazu

Za shranjevanje podatkov o plazu v objekt session smo sestavili objekt tipa Plaz, ki ima podobno sestavo kot tista tabela v bazi, ki predstavlja podatke o plazu. Tako kot so v bazi povezane tabele, so v objektu session povezani objekti. Dejansko smo v objektu session predstavili praktično kompletno bazo.

Primer:
Vsak plaz lahko vsebuje več vzrokov nastanka plazenja. Torej v bazi obstaja tabela vzrokov, ki je vezana na tabelo plazov (vsakemu plazu lahko pripada več vzrokov). Zato v objektu session obstaja objekt tipa *Vzrok* in vsak objekt tipa *Plaz* lahko vsebuje več objektov tipa *Vzrok*.

Objekti razreda *Plaz* torej vsebujejo dve vrsti podatkov (spremenljivk):

- Podatke, ki so s plazom v relaciji 1 - 1 (en plaz - en podatek). Taki so na primer ime plazu, ime lokacije, občina, naselje, itd. Ti podatki so v objektu tipa Plaz predstavljeni kot spremenljivke tipa String.
- Podatke, ki so s plazom v relaciji 1 - več (en plaz - več podatkov istega tipa). To so na primer vzroki nastanka plazu, poškodovani in ogroženi objekti, prizadete parcele, itd. Ti podatki so v objektu tipa Plaz predstavljeni s standardnim Javanskim tipom Vector iz objektov tipa Vzrok, Cesta, Stavba, Parcela, itd.



Slika 31 – Del zgradbe objekta Plaz.java

Vse spremenljivke razredov so privatne (*private String sifra, ...*) in so zato neposredno vidne samo znotraj objekta. Dostop do spremenljivk izven objektov je mogoč le preko metod, ki jih delimo na t.i. metode *'set'* in *'get'*. Metode *set* nastavljajo, metode *get* pa vračajo vrednosti spremenljivk.

4.3.3. Prebiranje in vstavljanje podatkov

Takoj, ko uporabnik začne delati z izbranim plazom, se ustvari objekt plaz tipa Plaz. Če gre za popravljanje obstoječega plazu se v ustrezne spremenljivke prenesejo podatki iz baze. Objekt plaz se zapiše v objekt session. Vsa komunikacija (spreminjanje, vpisovanje, ...) nato poteka z objektom plaz (ki se sproti prebira in zapisuje iz/v objekt session) in ne z bazo, vse dokler uporabnik ne zaključi z delom na tem plazu.

Dostop do podatkov iz objekta session poteka preko zgoraj omenjenih metod *get*. Najprej iz objekta session preberemo objekt tipa Plaz, nato pa iz objekta tipa Plaz preberemo izbrani podatek. Ta je lahko niz (String) ali objekt (tipa Vzrok, Cesta, ...) .

Vstavljanje novih podatkov o plazu v session objekt poteka podobno kot branje, le v nasprotnem vrstnem redu in z metodami *set*.

Primer: popravljanje podatka o opisu poškodbe na cesti v objektu tipa *Session*

```
// preberemo objekt plaz iz objekta session
Plaz plaz = (Plaz)session.getAttribute("plaz");
// preberemo ceste iz plazu
Vector ceste = plaz.getCeste();
// preberemo prvo cesto v seznamu
Cesta cesta = (Cesta)ceste.elementAt(0);
// preberemo poškodbe na cesti
Vector poskodbe = cesta.getPoskodbe();
// preberemo prvo poškodbo
Poskodba poskodba = (Poskodba)poskodbe.elementAt(0);

// definiramo nov opis poškodbe
String opis = "Poškodovane ograje";
// poškodbi nastavimo opis
poskodba.setOpis(opis);
// spremenjeno poškodbo zapišemo nazaj v seznam poškodb
poskodbe.setElementAt(poskodba, 0);
// spremenjene poškodbe zapišemo v objekt cesta
cesta.setPoskodbe(poskodbe);
// spremenjeni prvi podatek v seznamu cest zapišemo nazaj v seznam cest
ceste.setElementAt(cesta, 0);
// spremenjeni seznam cest zapišemo v objekt plaz
plaz.setCeste(ceste);
// popravljeni objekt plaz shranimo v objekt session
session.setAttribute("plaz", plaz);
```

Seveda bi lahko del stavkov združili. Denimo zadnja dva v

```
session.setAttribute("plaz", plaz.setCeste(ceste));
```

Z zgornjim postopkom smo spremenili opis prve poškodbe prve poškodovane ceste v "Poškodovane ograje".

5. ZAKLJUČEK

Prihodnost sistema Evidenca nestabilnosti tal

S postavitvijo sistema evidence nestabilnosti tal je Ministrstvo za okolje in prostor naredilo prvi korak k vzpostavitvi enotnega sistema za vodenje podatkov o plazovih. Sistem, ki smo ga zgradili v tej prvi fazi, omogoča t.i. produkcijo podatkov o plazovih. To pomeni, da smo vzpostavili sistem, s pomočjo katerega lahko vnašamo, urejamo in sledimo podatkom o plazovih.

Naslednji korak v razvoju sistema, ki ga ministrstvo načrtuje v bližnji prihodnosti, je podpora načrtovanju dodeljevanja finančnih sredstev za odpravo posledic in sanacijo plazov. Obstoječim podatkom se bodo dodali določeni indeksi, ki bodo predstavljali nujnost in višino finančnega posredovanja za sanacijo. Vsak plaz bo s tem vseboval tudi podatke o tem, koliko sredstev je bilo že namenjenih za sanacijo, koliko bi jih bilo še potrebno nameniti ter kako nujna je sanacija. Ministrstvo bo imelo tako pregled nad tem, koliko in kdaj mora zagotoviti sredstva za sanacijo te vrste naravnih nesreč.

Ko bo evidenca nestabilnosti tal že utečena in bo vsebovala dovolj podatkov, bodo le-ti dostopni tudi širši javnosti. Sistem se bo v tem primeru nadgradil podobno, kot so se že nekateri tovrstni sistemi pred njim. Tehnično se bosta ločili produkcijska in distribucijska baza podatkov. Produkcijska, ki smo jo s tem projektom postavili, bo namenjena ažuriranju podatkov. Distribucijska baza bo posnetek produkcijske. Posnetek se bo izvajal dnevno. Distribucijska baza bo namenjena zgolj pregledovanju podatkov o nestabilnosti tal. S takšnim pristopom vpogleda v podatke razbremenimo glavno (produkcijsko) bazo podatkov, ki mora biti ves čas ažurna in dostopna ter strogo varovana. Javno dostopna (in s tem bolj izpostavljena morebitnim vdorom) bo distribucijska baza, kjer nima smisla, da se podatki osvežujejo v realnem času.

Vsebina aplikacije in njena izvedba

Sistem evidence nestabilnosti tal se je pred implementacijo na prvi pogled zdel povsem klasičen. Gre za postavitev baze podatkov in aplikacije, v tem primeru spletne, ki omogoča pregled in polnjenje te baze. Vendar pa smo pri načrtovanju vedno bolj spoznavali kompleksnost sistema, kljub temu, da smo se ga trudili čim bolj poenostaviti.

Prva manjša ovira je bila predpostavka, da se noben vnesen in veljaven podatek nikoli ne odstrani iz baze. To je pomenilo, da je bilo potrebno vzpostaviti zgodovino podatkov. To smo dosegli s tem, da smo podatke razdelili v dogodke nad plazom.

Naslednja stvar, ki je prispevala h kompleksnosti sistema, je tehnična in vsebinska raznolikost podatkov, ki se vodijo v evidenci. Vodijo se podatki, ki jih uporabnik vnaša preko tipkovnice, podatki, ki so določeni s šifranti, podatki, ki so s plazom v razmerju ena – več, shranjujejo se datoteke, itd. Prav tako je dostop (vidnost) do posameznih podatkov različen. Vsebinsko so podatki razdeljeni v več skupin, kjer so nekateri vidni samo uporabnikom s posebnimi pravicami. Poleg raznolikosti podatkov določen zaplet predstavlja tudi dejstvo, da so podatki o enem plazu sestavljeni iz več kot 200 različnih atributov.

Tehnologija

Spletna aplikacija za vodenje evidence nestabilnosti tal je bila prva spletna aplikacija, ki smo jo zgradili v Oracle ADF okolju. Pred tem smo v razvojni skupini dovolj dobro poznali Oracle podatkovno bazo in JSP spletno tehnologijo.

Poznavanje JSP tehnologije je služilo le kot osnova za razumevanje ADF razvojnega okolja. Skozi razvijanje aplikacije smo morali podrobno spoznati BC4J (Business Components for Java) in UIX okolje, kar je od nas zahtevalo vsaj četrtino časa, ki smo ga porabili za implementacijo. Splošen vtis o okolju Oracle ADF, ki smo ga dobili med razvojem aplikacije, je, da zahteva veliko časa in nudi premalo pomoči za osvojitev znanja. Kljub temu smo s tem pridobili veliko znanja in izkušenj s področja gradnje spletnih aplikacij v okolju, ki ponuja že razvite rešitve za standardne aktivnosti in jih ni potrebno vedno znova programirati. Hkrati je poznavanje Oracle ADF okolja dober uvod v standard JSF (Java Server Faces), ki prihaja v ospredje na področje razvoja spletnih aplikacij.

Moja izkušnja

Poleg pridobivanja zgoraj naštetih izkušenj lahko pri sebi dodam še splošno izkušnjo načrtovanja in izvedbe takšnega sistema. Pred tem še nisem imel nikakršnih izkušenj s tega področja. Poznal sem le programski jezik Java in razvojno okolje JSP (HTML, CSS, JavaScript) ter nekaj osnov dela z bazami podatkov. Skozi načrtovanje evidence plazov sem se najprej naučil uporabljati jezik za načrtovanje UML in z njim predstaviti svoje zamisli in ideje drugim ter sprejemati in razumeti njihove zamisli in izkušnje. Kasneje sem spoznal tudi delo na področju vzpostavitve baze podatkov in se naučil programirati v programskem jeziku PL/SQL. Največ časa sem posvetil razvoju poslovne logike aplikacije in okolju Oracle ADF. Pri načrtovanju poslovne logike sem moral upoštevati vsebinske zahteve, tehnične zmožnosti in izkušnje drugih. Od vsega naštetega je bilo najtežje osvojiti okolje Oracle ADF . Začel sem z osnovami in s pomočjo redkih primerov na internetu simuliral evidenco plazov. Sčasoma sem pridobil dovolj znanja, da sem lahko sestavil zaključene dele aplikacije. Ob zaključku implementacije sem se srečal tudi z namestitvijo takšne aplikacije na aplikacijski strežnik.

Sistem Evidenca nestabilnosti tal smo predstavili tudi na 9. strokovnem srečanju uporabnikov programske opreme Oracle (SIOUG 2004) v Portorožu, kjer smo spoznali, da smo bili prvi, ki smo se odločili razvijati spletne aplikacije v okolju Oracle ADF.

Razvoj aplikacije od začetka do postavitve na strežnik je bila zame bogata izkušnja s področja informacijskih sistemov, tako njihovega razvoja kot tudi delovanja.

LITERATURA

1. Hans Bergsten, JavaServer Pages, 2nd Edition, O'Reily & Associates, 2002
2. Duane K. Fields, Mark A. Kolb, Web Development with JavaServer Pages, Manning Publications Co., 2000
3. Oracle JDeveloper, http://www.oracle.com/technology/products/jdev/index.html
4. Oracle Technology Network Forum, JDeveloper, ADF, http://forums.oracle.com/forums/forum.jsp?forum=83
5. Interno gradivo podjetja IGEA d.o.o.