UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)



**Miran Kirm**

# RAČUNALNIŠKO PODPRTO DISPEČERSTVO V NUJNI MEDICINSKI POMOČI

**Diplomska naloga**



Ljubljana, 2006

# ZAHVALA

Zahvaljujem se vsem, ki so mi na kakršen koli način pomagali, tako pri študiju kot tudi pri izdelavi diplomske naloge. Zahvaljujem se mentorju mag. Matiji Lokar za nadzor in koristne napotke. Hvala zunanjemu mentorju mag. Alešu Jelovšku za pomoč pri učenju tehnologij, ki so bile osnova za diplomsko delo. Zahvaljujem se tudi svoji teti Mili in njeni družini, ki me je v času študija sprejela medse, ter mi tako olajšala študij.

Diplomsko nalogo posvečam svojim staršem.

**Program dela**

V diplomski nalogi opišite, kako je potekal razvoj aplikacije za računalniško podprto dispečerstvo v nujni medicinski pomoči. Na kratko opišite tudi tehnologijo za razvoj aplikacije, torej samo programsko okolje Visual Basic in ustrezni jezik.

Mentor:
mag. Matija Lokar

Zunanji mentor:
mag. Aleš Jelovšek

**POVZETEK**

V diplomski nalogi predstavljam razvoj aplikacije za beleženje podatkov o nujnih reševalnih prevozih. V uvodnem delu predstavim, kakšen način dela je bil v veljavi pred uporabo aplikacije, ter njegove pomanjklivosti. V nadaljevanju predstavim glavne sklope podatkov, ki so potrebni za uspešno in strokovno izvedbo reševalnega prevoza. Predstavim tudi tehnologijo, s katero je bila aplikacija napisana. Opišem, kakšno je bilo načrtovanje podatkovnih struktur, ter zakaj smo se odločili za tehnologijo, ki je bila uporabljena za izdelavo aplikacije. Na koncu prikažem in opišem tudi nekaj najzanimivejših delov aplikacije.



**Math. Subj. Class. (2000): 68N15, 92C50, 62P10**
**Computing Review Class. System (1998): C.2.4, D.1.1, D.1.7, D.2.5, D.3.2, E.1, H.2.8, H.4, J.3**

**Ključne besede:** Nujna medicinska pomoč, dispečerstvo, Visual Basic, intervencija, reševalna ekipa
**Keywords:** Emergency medical service, dispatching, Visual Basic, intervention, rescue team

# 1 UVOD

Nujna medicinska pomoč (NMP) je izvajanje nujnih ukrepov zdravnika in njegove ekipe pri osebi, ki je zaradi bolezni ali poškodbe neposredno življenjsko ogrožena, oziroma pri kateri bi glede na bolezenske znake v kratkem času lahko prišlo do takšne ogroženosti bolnika.

Izvajanje NMP zagotavlja služba NMP, ki je sestavni del mreže javne zdravstvene službe. Organizirana je za zagotavljanje neprekinjene nujne medicinske pomoči obolelim in poškodovanim na območju države Slovenije s ciljem, da se kar najbolj skrajša čas od nastanka nujnega stanja do začetka dokončne zdravniške oskrbe. Naloga NMP je tudi zagotavljanje nenujnih prevozov bolnikov, ki bi jim javni prevoz škodoval, oziroma nimajo možnosti prevoza z njim.

Dispečerstvo je sestavni del službe NMP, ki skrbi za prvo fazo zagotavljanja NMP. To je sprejemanje podatkov o bolnikih oziroma pacientih ter dodeljevanje nastalih intervencij posameznim ekipam, ki potem opravijo prevoz pacienta iz začetne na končno lokacijo.

V diplomski nalogi bom predstavil, kako je bila razvita aplikacija, ki podpira dispečerski postopek. Predstavljen bo sam način načrtovanja aplikacije. Prikazan bo tudi razvoj nekaterih ključnih delov aplikacije, kot na primer priprava vnosnih polij, shranjevanje vnešenih podatkov o intervenciji ter dodelitev intervencije ekipi, ki bo prevoz opravila.

# 2 VSEBINA APLIKACIJE

## 2.1 Analiza obstoječega stanja

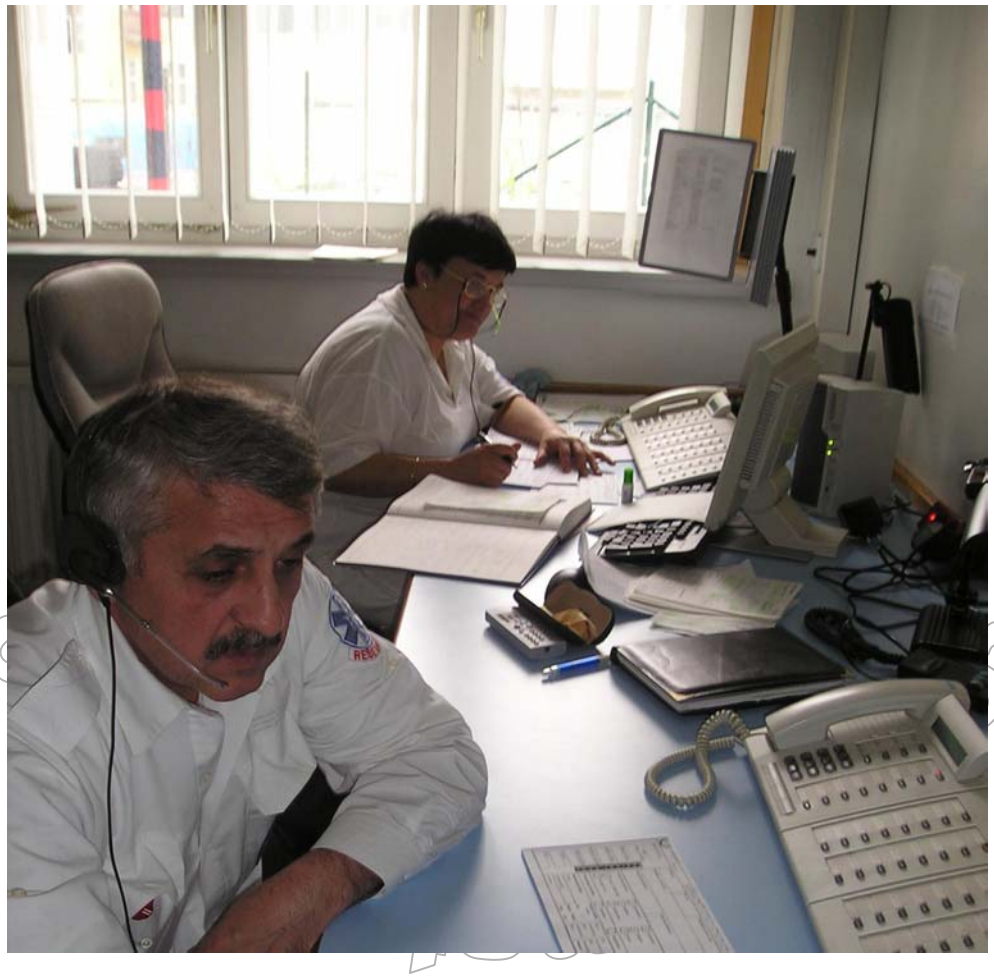
**Stanje v Sloveniji**

Stanje dispečerstva v nujni medicinski pomoči je v Sloveniji neurejeno. Zaradi tega prihaja do nepotrebnih napak tako v samem procesu sprejemanja intervencij, kot tudi pri izvajanju voženj. V večini primerov, razen v večjih organizacijah, nimajo oseb, ki bi bile zadolžene za sprejemanje klicev, ampak telefon dvigujejo sestre, zdravniki, ... skratka osebe, ki praviloma niso usposobljene za ustrezno obravnavo klica.

**Stanje na RP KC Ljubljana**

Na reševalni postaji kliničnega centra (RP KC) Ljubljana dispečerstvo izvajajo neprekinjeno, štiriindvajset ur dnevno. Izvajajo ga za to usposobljene osebe. Kljub temu pa prihaja do napak pri vnosu podatkov, tako pri sprejemu kot tudi pri oddaji intervencij. Ni sledenja, kdo je napako povzročil. Pogosto je tudi nemogoče ugotoviti, ali je do napake sploh prišlo, ali pa je klicatelj povedal napačne podatke.

Opišimo postopek pri obravnavi posameznega klica. Sprejem intervencij izvaja sprejemni dispečer. Ob telefonskem klicu zabeleži podatke na papirni obrazec. Izpolnjen obrazec skozi odprtino preda oddajnemu dispečerju.



**Slika 1**
**Sprejem intervencij**

Ta podatke o intervenciji zapiše v knjigo prevozov. To je knjiga, kjer se beležijo podatki o intervenciji in vožnji. Potem listek vtakne v prioritetno omarico, v sektor, ki je namenjen tej prioriteti. Ko je na voljo prosto vozilo, intervencijo odda ekipi tega vozila in to zabeleži v knjigo. Listek v prioritetni omarici premakne iz sektorja prioritet v sektor vozil. Tam ga uvrsti pod številko vozila, ki to intervencijo izvaja.

**Slika 2**
**Knjiga prevozov**


**Slika 3**
**Prioritetna omarica**

Pri takem načinu dela je veliko težav in prepisovanja podatkov. Že pri samem vnosu podatkov v knjigo prevozov prihaja do napak pri pisanju. Nekateri dispečerji tudi nimajo čitljive pisave in tisti, ki ga na delovnem mestu zamenja, vnesenih podatkov ne zna prebrati. Predvsem v kaotičnih stanjih, ko je veliko število intervencij, prihaja do pomot pri razvrščanju intervencij v predalčke. Zelo težko je imeti celostno sliko nad stanjem vseh intervencij. Ker se izvajaja tudi veliko rednih prevozov (prevozi bolnikov na dializo, ...), je veliko prepisovanja podatkov, kjer se podatki ponavljajo. Predvsem pa je ob takem načinu dela težavna in dolgotrajna analiza dela. Osebe zadolžene za letne analize za to delo potrebujejo mesec dni in več. Sami rezultati niso zelo zanesljivi, saj je velika verjetnost, da se pri prepisovanju podatkov v najrazličnejše programe zmotijo.

## 2.2 O aplikaciji in uporabnikih aplikacije

Da bi odpravili težave, ki izvirajo iz ročnega obravnavanja klicev, so se pri RP KC odločili, da naročijo izdelavo aplikacije. Ta naj bi dispečerjem omogočila lažje in hitrejše, predvsem pa natančnejše delo. Pomembno je tudi to, da se s pomočjo uporabe aplikacije dispečerja psihično razbremeni, saj mora včasih pomniti stanja tudi petih ali več intervencij.
Aplikacija je bila izdelana in predana v uporabo. Uporabnik aplikacije je trenutno le RP KC Ljubljana. V dogovoru glede uporabe smo tudi z RP Kranj, RP Maribor in RP Celje.

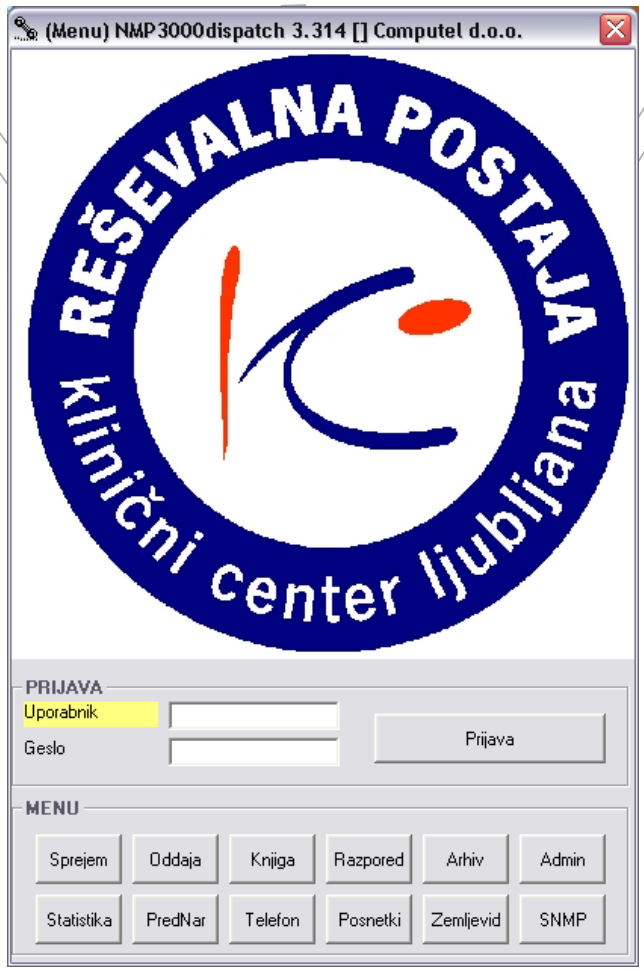
### 2.2.1 Uporabniki v organizaciji

Uporabniki aplikacije so:

- Sprejemni dispečerji (vnašajo podatke o intervencijah),
- Oddajni dispečerji (oddajajo sprejete intervencije ustreznim ekipam in spremljajo potek vožnje),
- Vodje izmen (kreirajo ekipe),
- Vodja dispečerjev (Je administrator aplikacije. Dodaja uporabnike aplikacije in člane ekip, lokacije,....)

## 2.3 Sklopi uporabe aplikacije

**Prijava v sistem**
Vsak uporabnik, ki želi uporabljati katerokoli funkcionalnost programa, se mora v program prijaviti. S tem smo vzpostavili zaščito pred tem, da bi nepooblaščene osebe znotraj organizacije dostopale do podatkov, za katere niso pooblaščene. Drugi razlog za uvedbo prijave pa je ta, da s tem sledimo, kdo je sprejel intervencijo ter kdo je podatke spreminjal oziroma dopolnjeval. S tem lahko ob morebitnih pritožbah hitro ugotovimo uporabnika, ki je podatke spreminjal.

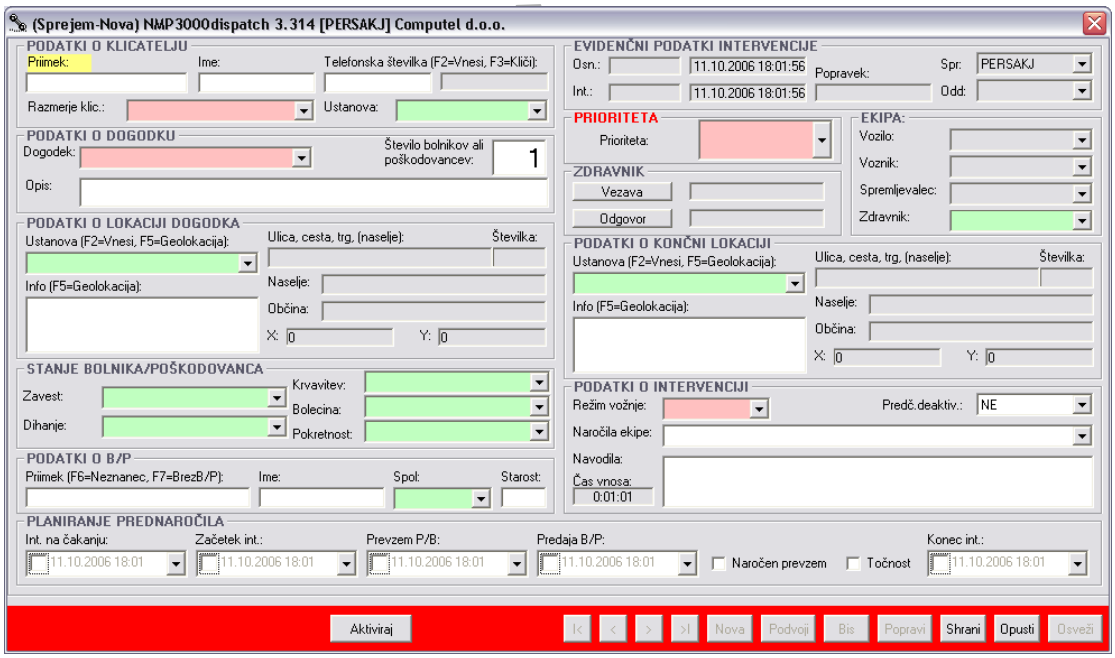



**Slika 4**
**Forma za prijavo in izbor funkcionalnosti**

**Sprejem intervencije**

Intervencije sprejemajo sprejemni dispečerji. Ob telefonskem klicu dispečer dvigne slušalko in ko ugotovi, da klic ni lažen ali kakorkoli drugače nepovezan z nastankom nove intervencije, odpre sprejemno formo. Sedaj ima tri možnosti. Lahko gre za novo intervencijo, dodatno intervencijo ali pa ponavljajočo se intervencijo. Pri novi intervenciji klic zahteva nov poseg nujne medicinske pomoči na terenu. Pri dodatni intervenciji se intervencija trenutno izvaja (na kraj dogodka je bila že poslana ena ali več ekip) in je potrebno poslati še dodatno ekipo ali celo več ekip. Pri ponavljajočih se intervencijah gre za ponavljajoče se dogodke, kot so recimo prevozi sladkornih bolnikov, ki jih je na določene dneve v tednu ali mesecu potrebno redno voziti na dializo. Dodatno intervencijo imenujemo tudi bis, ponavljajočo se intervencijo pa podvojena. Ker sta to v službi nujne medicinske pomoči zelo ustaljena izraza, ju bomo, čeprav sta žargonska, od sedaj naprej uporabljali.

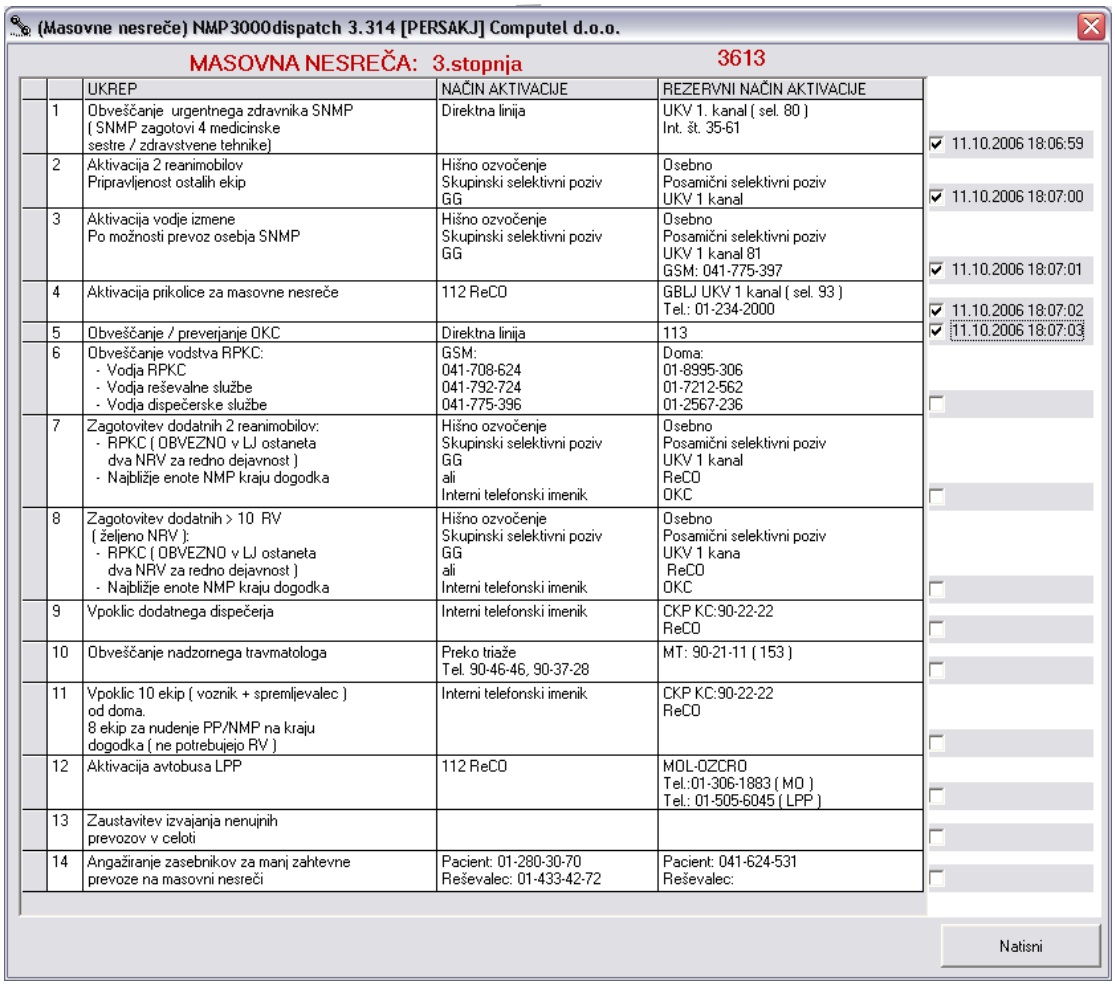
V primeru nove intervencije je sprejemna forma prazna in dispečer mora vnašati vse podatke. Podvojene intervencije in bisi se nanašajo na intervencije, ki so že vnesene v sam sistem. Zato pri teh dispečer najprej poišče ustrezno intervencijo. Tu se večina podatkov v vnosnih poljih lahko ohrani. Zato mora dispečer vnesti oziroma spremeniti le nekaj podatkov. Količina obveznih podatkov za vnos se v vseh treh primerih spreminja glede na vrsto dogodka. Obvezni podatki so prioriteta, stopnja nujnosti in vrsta dogodka. Glede na vrsto dogodka lahko postanejo obvezni vnosi še podatki, ki se nanašajo na stanje bolnika ali pacienta.




**Slika 5**
**Forma za sprejem intervencije**

V primeru težjih poškodb ali drugih nesreč dispečer telefonski klic preveže k zdravniku splošne nujne medicinske pomoči (SNMP). Ta se pogovori s kličočim in mu po potrebi poda nasvet, kako ravnati do prihoda reševalne ekipe na kraj dogodka. Na osnovi telefonskega pogovora s kličočim se zdravnik tudi odloči, ali bo osebno vključen v reševalno ekipo in to sporoči dispečerju.
Po končanem vnašanju podatkov mora dispečer podatke shraniti, da se ti zabeležijo v bazo. Če klikne na gumb prekliči, se vsi vneseni podatki brišejo in se ne shranijo v bazo.
V primeru masovnih nesreč se takrat, ko dispečer intervencijo shrani, odpre poseben obrazec za masovne nesreče. Ta dispečerja vodi skozi celoten proces obveščanja nadrejenih, aktiviranja dodatnih ekip in ostalih potrebnih ukrepov. Kot masovna nesreča je opredeljena tista, kjer kličoči ne ve števila poškodovancev, ve pa, da je le teh več ali pa tista nesreča, kjer je število poškodovancev večje od štiri.
Masovne nesreče se glede na število poškodovancev delijo v pet skupin:

- 1. delna stopnja (število poškodovancev je neznano)
- večja nesreča (v primeru, ko je med štiri in devet poškodovancev)
- 1. stopnja (število poškodovancev je med deset in devetnajst)
- 2. stopnja (poškodovancev je med dvajset in devetindvajset)
- 3. stopnja (ko je število poškodovancev trideset ali več)




**Slika 6**
**Obrazec za spremljanje masovne nesreče**

### Pregled prednaročil

Prednaročilo je nenujna intervencija. Naročilo zanjo je sprejeto vsaj dan, preden jo je potrebno izpeljati. Vnesena prednaročila vodjem izmen služijo za planiranje števila in sestave ekip.

Prednaročilo ima poleg vseh običajnih podatkov podana še dva osnovna časa. Prvi je čas »intervencija na čakanju«. To je čas, ko je intervencija vidna na seznamu intervencij, ki jih je potrebno še oddati. Drugi čas je »prevzem B/P« (bolnika ali pacienta) ali »predaja B/P«, odvisno od tega, kateri čas je pomembnejši. Seveda je izbrani čas ustrezno označen. Omenimo še to, da bomo kratico B/P v diplomi še pogosto uporabljali, saj se izraz bolnik ali pacient pojavi zelo pogosto. Prav tako je v sami aplikaciji povsod uporabljena ta kratica.

Poleg teh časov lahko sprejemni dispečer, vodja dispečerjev ali pa vodja izmene določijo še časa »začetek intervencije« in »konec intervencije«, torej čas, ko ekipa sprejme intervencijo in ko jo konča.

V primeru, da so dispečerji ob sprejemu prednaročila izpolnili vse čase, se ob poizvedovanju o intervencijah za določen dan, poleg osnovnih podatkov o vsakem prednaročilu izriše še časovni trak poteka intervencije.

Te vrste intervencij se lahko uredi po kraju dogodka, končni lokaciji, uri prevzema pacienta, uri predaje pacienta… S tem omogočamo vodji izmene, da po možnosti dve ali več intervencij združi v eno vožnjo.

**Oddaja vožnje ekipi in spremljanje poteka intervencije**
Oddajo vožnje, ki je lahko sestavljena iz ene ali več intervencij, izvaja oddajni dispečer. Oddajo izvede tako, da uporabi oddajno formo. Na njej so vidne vse intervencije na čakanju, intervencije v izvajanju in trenutno vzpostavljene ekipe NMP.
Intervencije na čakanju so tiste intervencije, ki so bile za to obdobje zabeležene v sistem in še niso bile izpeljane. V stolpcu se razvrščajo po stopnji nujnosti in po prioriteti (pomembnosti).
Intervencije v izvajanju so intervencije, ki jih ekipe že izvajajo, vendar še niso končane.
Ekipa je sestavljena iz vozila, voznika in spremljevalca. Vse tri komponente ekipe se lahko v času, ko ekipa obstaja, zamenjajo.
Oddaja poteka tako, da oddajni dispečer pogleda izbrano intervencijo in ugotovi, katera ekipa ima potrebna sredstva za izvedbo vožnje. Nato po radijskih zvezah izbrano ekipo obvesti, da je dobila intervencijo ter ji sporoči podatke o vožnji. V programu označi tako intervencijo kot ekipo ter klikne na gumb »Oddaj«. S tem je intervencijo oddal ekipi.
Delo oddajnega dispečerja je tudi, da spremlja potek intervencije. To delo mu olajša poseben program, ki je povezan z radijskimi zvezami in beleži statuse intervencije.
Statusi so stanja ekipe, ki so sledeči:

- prost (ekipa je prosta in lahko sprejme novo intervencijo)
- pogojno razpoložljiv (ekipa je pacienta že predala, vendar še ureja potrebno dokumentacijo, ali pa je na kosilu, ima odmor, … Ob nujni intervenciji je ekipa vseeno na razpolago.)
- na poti (ekipa je sprejela naročilo in je na poti na kraj dogodka)
- na kraju (ekipa je prišla na kraj dogodka)
- se vrača (ekipa gre s kraja dogodka in pelje B/P na končno lokacijo)
- na cilju (ekipa je pripeljala B/P na končno lokacijo)

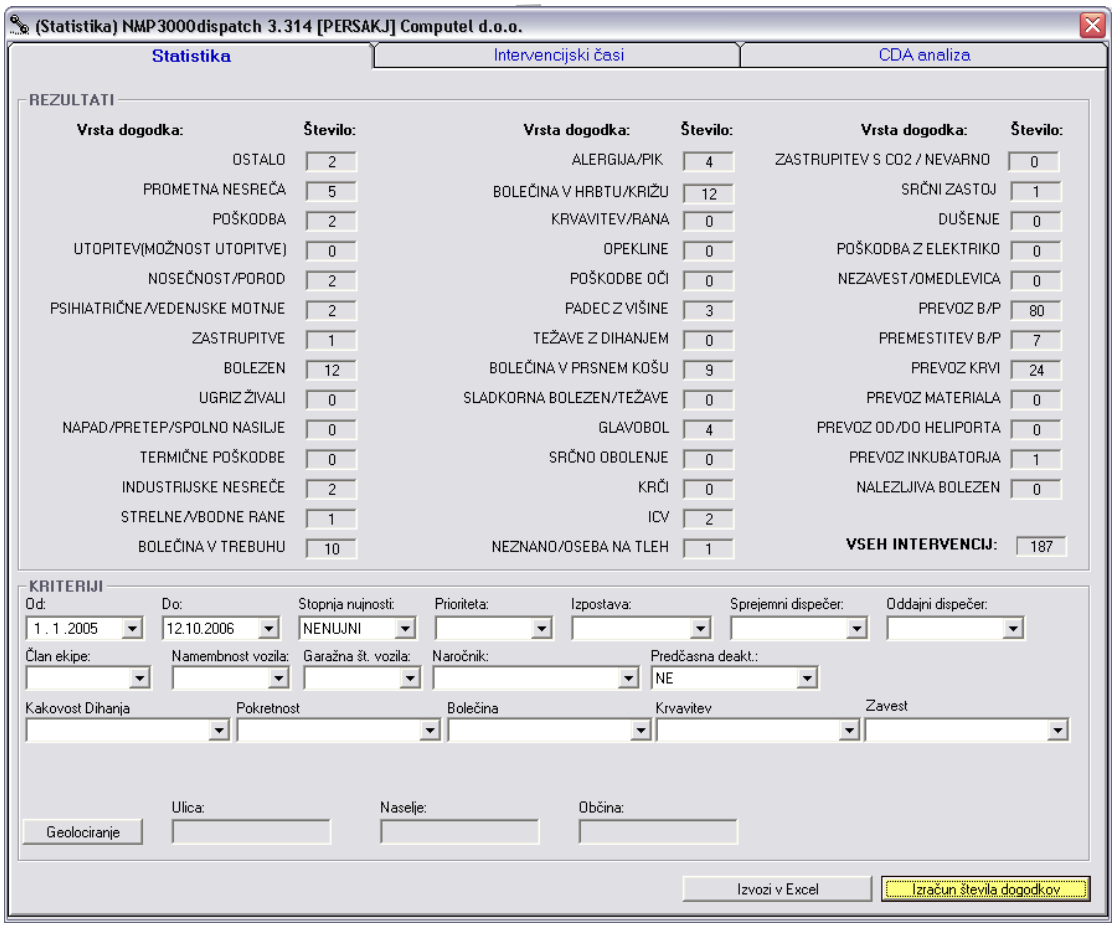
Oddajni dispečer mora statuse ekip spremljati tudi po radijskih zvezah. Statuse ekip mora vnesti k vsaki intervenciji, saj ekipa lahko sočasno izvaja dve ali več intervencij. K posameznemu statusu se beleži tudi čas, ko je bil ta status dosežen.

**Urejanje dokumentacije po končani intervenciji**
Po končani intervenciji morajo spremljevalci vodji dispečerjev prinesti napotnico ali nalog za prevoz, ki ga potrdi zdravnik SNMP. Vodja dispečerjev številko dokumenta vpiše v program in označi polje »Dokumentacija urejena«. To je potem skupaj z napotnico ali nalogom za prevoz podlaga za vnos podatkov v obračun.

**Sledenje in statistična obdelava**
Vse delo z aplikacijo se beleži. Sledenje poteka ob vsakršni spremembi podatkov. Dispečer mora, če želi podatke spremeniti, klikniti gumb »uredi«, vtipkati nove podatke ali popraviti obstoječe in klikniti »shrani«. Šele takrat se podatki shranijo, opremljeni z njegovim uporabniškim imenom. Novi podatki se shranijo tudi v evidenco popravkov, kjer so zapisana vsa stanja določene intervencije.
Sledenje omogoča nadzor dispečerjev in ogled razvoja posameznih intervencij. V primeru pritožb skupaj s posnetimi pogovori med zdravnikom, dispečerjem in kličočim, sledenje omogoča, da se ovržejo ali potrdijo morebitne napake delavcev reševalne postaje.

Statistična obdelava sledi po opravljenih vožnjah in je največkrat smiselna po daljšem časovnem obdobju. Statistiko delimo na tri dele.

- statistika, kjer v okviru kriterijev, ki jih določimo, izvemo, koliko intervencij je bilo izvedenih. Pregledi so urejeni po posameznih vrstah dogodkov.
- intervencijski časi. Tu izvemo, kakšni so bili posamezni maksimalni, minimalni ter povprečni časi trajanj intervencij v okviru kriterijev, ki jih določimo.
- CDA analiza. Call demand analiza nam pove ali posamezna organizacija, v našem primeru RP KC, dosega določene standarde.




**Slika 7**
**Statistika:**
**število intervencij glede na vrsto dogodka**

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

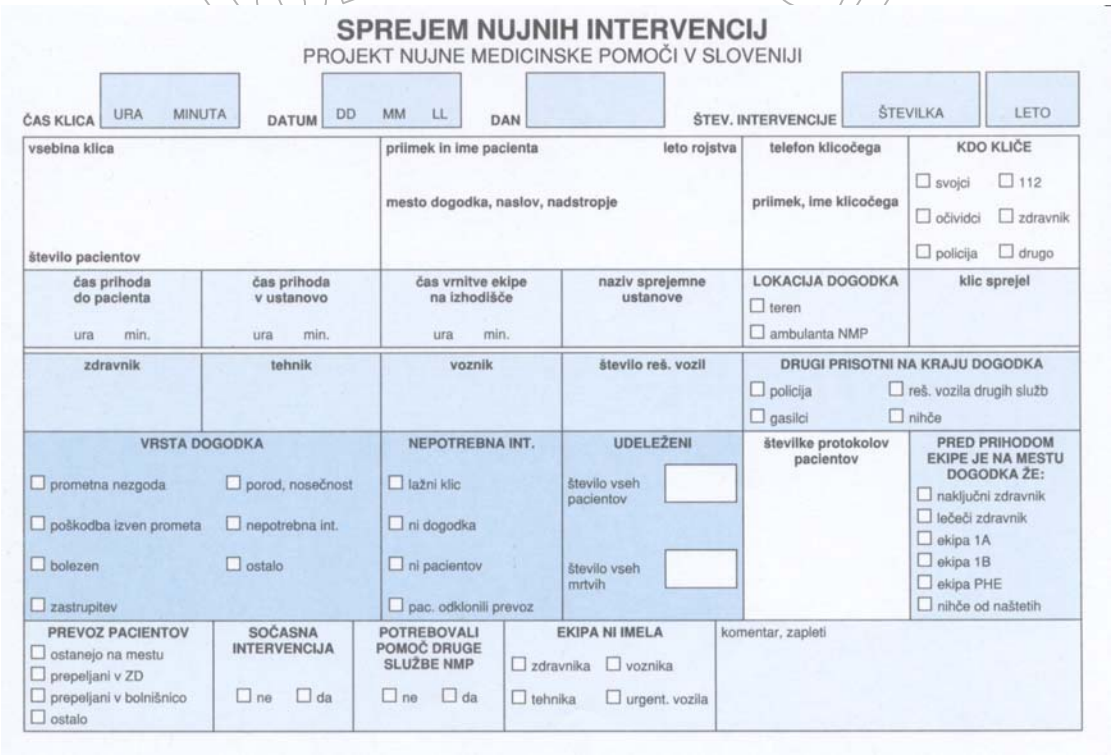
## 2.4 Opis podatkov

Osnova za pripravo baze podatkov sta bila dva obrazca. Prvi obrazec je predpisan s strani Ministrstva za zdravje in opisuje sprejem nujne intervencije (slika 8), drugega pa je predpisalo vodstvo Reševalne postaje Kliničnega centra in zajema podatke o prejemu telefonskega klica za nenujni prevoz B/P in spremljanje nadaljnih aktivnosti (slika 9).
Poleg teh podatkov smo med beležeče podatke dodali še različne evidenčne podatke o intervenciji. Ti nam pomagajo slediti razvoju intervencije. Tako je nastalo več sklopov podatkov, ki se skupaj imenujejo podatki o intervenciji in se zapisujejo v tabelo evidenca prevozov. Podatki se pridobivajo ali iz šifrantov ali pa jih mora uporabnik vpisati sam.

Glavni sklopi so:

- podatki o kličočem
- podatki o dogodku
- podatki o lokaciji dogodka in končni lokaciji
- stanje bolnika/poškodovanca (B/P)
- podatki o B/P

- prioriteta in nujnost
- evidenčni podatki intervencije
- ostali podatki



**SPREJEM NUJNIH INTERVENCIJ**

PROJEKT NUJNE MEDICINSKE POMOČI V SLOVENIJI

ČAS KLICA URA MINUTA | DATUM DD MM LL | DAN | ŠTEV. INTERVENCIJE ŠTEVILKA LETO

| vsebina klica<br>število pacientov | priimek in ime pacienta leto rojstva<br>mesto dogodka, naslov, nadstropje | telefon klicočega<br>priimek, ime klicočega | KDO KLIČE<br>☐ svojci ☐ 112<br>☐ očividci ☐ zdravnik<br>☐ policija ☐ drugo |
|---|---|---|---|

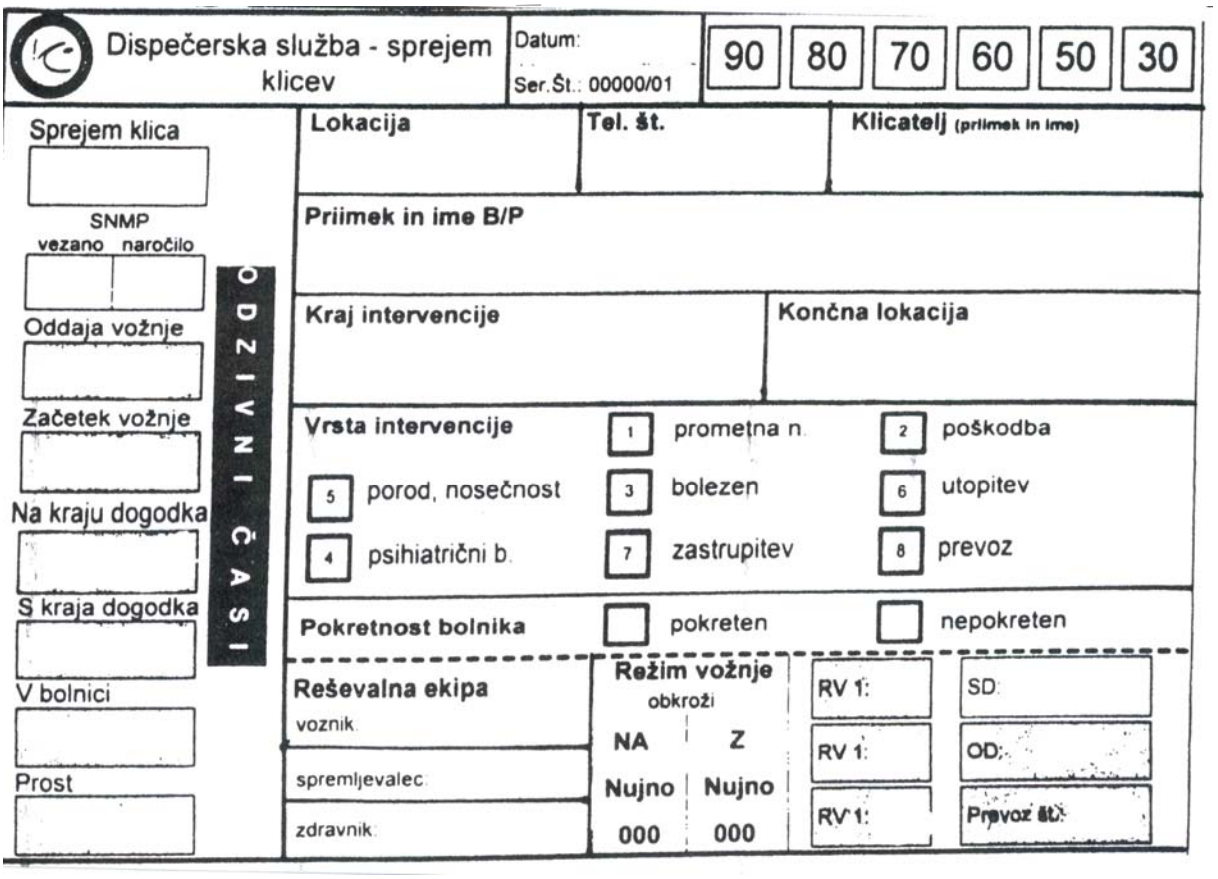
| čas prihoda do pacienta<br>ura min. | čas prihoda v ustanovo<br>ura min. | čas vrnitve ekipe na izhodišče<br>ura min. | naziv sprejemne ustanove | LOKACIJA DOGODKA<br>☐ teren<br>☐ ambulanta NMP | klic sprejel |
|---|---|---|---|---|---|

| zdravnik | tehnik | voznik | število reš. vozil | DRUGI PRISOTNI NA KRAJU DOGODKA<br>☐ policija ☐ reš. vozila drugih služb<br>☐ gasilci ☐ nihče |
|---|---|---|---|---|

| VRSTA DOGODKA | NEPOTREBNA INT. | UDELEŽENI | številke protokolov pacientov | PRED PRIHODOM EKIPE JE NA MESTU DOGODKA ŽE: |
|---|---|---|---|---|
| ☐ prometna nezgoda ☐ porod, nosečnost<br>☐ poškodba izven prometa ☐ nepotrebna int.<br>☐ bolezen ☐ ostalo<br>☐ zastrupitev | ☐ lažni klic<br>☐ ni dogodka<br>☐ ni pacientov<br>☐ pac. odklonili prevoz | število vseh pacientov<br>število vseh mrtvih | | ☐ naključni zdravnik<br>☐ lečeči zdravnik<br>☐ ekipa 1A<br>☐ ekipa 1B<br>☐ ekipa PHE<br>☐ nihče od naštetih |

| PREVOZ PACIENTOV | SOČASNA INTERVENCIJA | POTREBOVALI POMOČ DRUGE SLUŽBE NMP | EKIPA NI IMELA | komentar, zapleti |
|---|---|---|---|---|
| ☐ ostanejo na mestu<br>☐ prepeljani v ZD<br>☐ prepeljani v bolnišnico<br>☐ ostalo | ☐ ne ☐ da | ☐ ne ☐ da | ☐ zdravnika ☐ voznika<br>☐ tehnika ☐ urgent. vozila | |

**Slika 8**
**Obrazec ministrstva za zdravje**




**Slika 9**
**Interni obrazec RP Ljubljana**

### 2.4.1 Podatki o kličočem

To so podatki, ki jih dispečer zbere o kličočem Ta je lahko očividec ali pa B/P. Zabeleži se ime, priimek, telefonska številka, razmerje kličočega in ustanova.
Razmerje kličočega je razmerje, ki ga ima kličoči do B/P. Kličoči je lahko očividec, svojec, gasilec, kličoči iz centra za obveščanje in podobno.
Ustanova je zdravstvena organizacija ali pa oddelek zdravstvene organizacije, od koder prihaja kličoči. Ustanovo se določi le v primeru, ko je kličoči iz zdravstvene organizacije.
Ustanove vpisuje administrator preko posebnega obrazca v šifrant sistema. Poleg organizacije se določijo tudi ulica, hišna številka, naselje, občina ter koordinati X in Y.

### 2.4.2 Podatki o dogodku

Podatki o dogodku so najpomembnejši del aplikacije. Zato je potrebno ta sklop podatkov natančno izpolniti. Vsebujejo vrsto dogodka, število pacientov in opis dogodka.
Vrsto dogodka obvezno izberemo iz šifranta. Možne vrste dogodkov so:

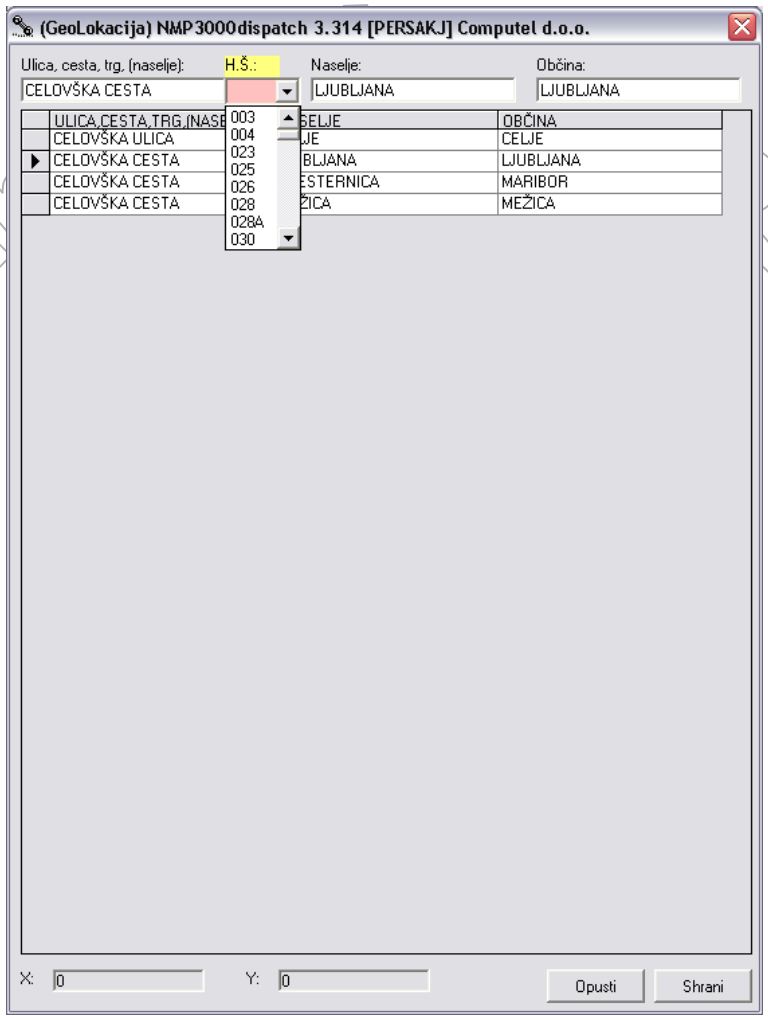
- Prometna nesreča
- Poškodba
- Utopitev ali možnost utopitve
- Nosečnost ali porod
- Psihiatrične ali vedenjske motnje
- Zastrupitve
- Bolezen
- Ugriz živali
- Pretep ali spolno nasilje
- Opekline
- Poškodbe oči

- Padec z višine
- Industrijske nesreče
- Neznana oseba na tleh
- Prevoz B/P
- Premestitev B/P
- Prevoz krvi
- Prevoz materiala
- Prevoz inkubatorja
- …

Na podlagi te izbire se določajo podatki, katerih vnos je obvezen (glej razdelek 2.4.4). Zelo pomemben podatek je tudi število pacientov. Na podlagi tega podatka se odloča, koliko vozil bo poslano na kraj dogodka, saj v večini primerov lahko eno vozilo pelje le enega pacienta naenkrat.
V opis se vpišejo še morebitne posebne okoliščine, kot so agresivnost B/P, morebitne dodatne oteževalne okoliščine in podobno.

### 2.4.3 Podatki o lokacij dogodka in končni lokaciji

Lokacija dogodka je lokacija, kamor je potrebno iti po pacienta. To je lahko nekje na cesti, če se je na primer zgodila prometna nesreča. Lahko je to hišni naslov pacienta, ki ga je potrebno peljati na pregled k specialistu, ali pa je to zdravstvena ustanova, od koder je potrebno peljati pacienta domov.
Končna lokacija je lokacija, kamor je potrebno pacienta peljati. To je ali hišni naslov pacienta ali pa ustanova, če se pacienta pelje na pregled oziroma zdravljenje.
Podatki o lokacijah vsebujejo ime ustanove, ulico, hišno številko, naselje, občino in Gauss-Krugerjeve koordinate lokacije. Kot pri ustanovi, od koder je kličoči, se tudi tu, če se lokacija nanaša na ustanovo, ustanova lahko izbira iz šifranta.
Če je lokacija na nekem naslovu, se naslov izbere preko posebne vnosne forme. Dispečer vtipka ime ulice ali njen del in pritisne tipko enter. Izpišejo se vsa naselja in občine, v katerih je ulica, ki vsebuje vneseno besedno zvezo. Dispečer izbere ustrezno in vsi podatki, tudi koordinate, se prepišejo v podatke o lokaciji.
V primeru, da se je intervencija zgodila nekje, kjer ni nobenih lokacij s hišnimi številkami, recimo na ljubljanski obvoznici, si dispečer pomaga z opisom kraja kličočega. Predpostavimo da kličoči pove, da se je zgodila prometna nesreča na ljubljanski obvoznici sto metrov pred izvozom za Vič. V tem primeru dispečer na sprejemni formi klikne gumb zemljevid. Odpre se forma z zemljevidom, kjer skupaj z tipko »shift« klikne z desnim gumbom miške na lokacijo, ki približno ustreza kraju dogodka. S tem se v sprejemno formo prenesejo koordinate dogodka.




**Slika 10**
**Obrazec za pridobivanje podatkov o lokaciji**

### 2.4.4 Stanje bolnika / poškodovanca (B/P)

Stanje B/P se razdeli v pet sklopov. To so zavest, dihanje, krvavitev, bolečina in pokretnost. Za vsak sklop obstaja šifrant, ki vsebuje opis stanja posameznega sklopa. Glede na vrsto dogodka je določeno, kateri od teh sklopov so obvezni za vpis. V evidenco prevoza se za vsak sklop zapiše šifra stanja.

Tako ima recimo pri dihanju dispečer na izbiro šest stanj, izmed katerih izbere tistega, ki najbolje opiše pacientovo dihanje. Na izbiro ima:

- Diha normalno
- Diha pospešeno
- Diha počasi oz. neredno
- Težko diha
- Ne diha
- Ni znano, ni možno ugotoviti

### 2.4.5 Podatki o B/P

So osnovni podatki o B/P. Vpišejo se ime, priimek in spol. Spol se vnaša iz šifrantov, zato se v evidenco prevoza zapisuje le šifra. Pogosto se zgodi, da v začetni fazi podatki niso vnešeni. Takrat je B/P voden kot neznanec.

### 2.4.6 Prioriteta in nujnost

Prioriteta in nujnost sta podatka, ki sta med sabo povezana. Uporabnik ju določi na podlagi podatkov, ki jih je dobil od kličočega. Tako prioriteta kot nujnost sta podatka v šifrantu in v evidenco prevoza se zapisuje le šifra.

### 2.4.7 Evidenčni podatki intervencije

To so podatki, na katere uporabnik nima vpliva. Delimo jih v dve skupini.

- Evidenčni podatki pred izvajanjem intervencije.
  V ta sklop štejemo zaporedno številko intervencije, čas nastanka intervencije, čas popravka intervencije, uporabnika, ki je intervencijo naredil, uporabnika, ki je intervencijo kakorkoli popravljal in čas, ko se je intervencija predala ekipi.
- Evidenčni podatki med izvajanjem intervencije.
  Ti podatki so časi, ki se beležijo med izvajanjem prevoza, kot so čas izvoza ekipe iz garaže, čas prihoda na kraj dogodka, čas odhoda s kraja dogodka in čas predaje pacienta. Vsi časi se beležijo avtomatsko. Ko član ekipe pritisne gumb radijske postaje, ki ustreza trenutnemu stanju ekipe, se v sistem zabeleži ustrezen čas.

### 2.4.8 Ostali podatki

Med ostale podatke štejemo podatke, ki jih ne moremo uvrstiti med zgoraj navedene podatke. Ti podatki so časi planiranja prednaročil, čas začetka zvonjenja telefona, čas dviga telefona. Časi o odzivnosti na telefonu se beležijo le v primeru, ko je nastala intervencija. Čase planiranja prednaročil se vpisuje ročno, medtem ko se časi povezani z telefonom, vpisujejo avtomatsko preko povezave s telefonsko centralo.
Oglejmo si sedaj vse podatke, zložene v tabelo.

| PODATEK | TIP PODATKA | NAČIN VNOSA |
|---|---|---|
| Zaporedna številka prevoza | številka | določi sistem |
| Številka osnovnega prevoza | številka | določi sistem |
| Šifra kličočega | številka | vnese uporabnik |
| Priimek kličočega | niz | vnese uporabnik |
| Ime kličočega | niz | vnese uporabnik |
| Šifra razmerja kličočega | številka | vnose uporabnik iz šifranta |
| Šifra vrste kličočega | številka | vnese uporabnik iz šifranta |
| Ustanova kličočega | številka | vnese uporabnik |
| Telefonska številka kličočega | niz | vnese uporabnik |
| Šifra Vrste dogodka | številka | vnese uporabnik iz šifranta |
| Opis dogodka | niz | vnese uporabnik |
| Število B/P | niz | vnese uporabnik |
| Priimek B/P | niz | vnese uporabnik |
| Ime B/P | niz | vnese uporabnik |
| Šifra Spola B/P | številka | vnese uporabnik iz šifranta |
| Starost B/P | število | vnese uporabnik |
| šifra zavesti | številka | vnese uporabnik iz šifranta |
| Šifra dihanja | številka | vnese uporabnik iz šifranta |
| šifra krvavitve | številka | vnese uporabnik iz šifranta |
| Šifra bolečine | številka | vnese uporabnik iz šifranta |
| Šifra pokretnosti | številka | vnese uporabnik iz šifranta |
| Prednaročilo | logična vrednost | vnese uporabnik |
| Prednaročilo točnost | logična vrednost | vnese uporabnik |
| Predvideno v oddajanje | čas | vnese uporabnik |
| Predviden začetek intervencije | čas | vnese uporabnik |

| | | | |
|---|---|---|---|
| Predviden prevzem pacienta | | čas | vnese uporabnik |
| Predvidena oddaja B/P | | čas | vnese uporabnik |
| Predviden konec intervencije | | čas | vnese uporabnik |
| Prednaročilo je začetek | | logična vrednost | vnese uporabnik |
| Ring | | čas | določi sistem |
| Dvig | | čas | določi sistem |
| Nastanek osnovnega naloga | | čas | določi sistem |
| Nastanek naloga | | čas | določi sistem |
| Končan vnos | | čas | določi sistem |
| Popravek naloga | | čas | določi sistem |
| Vezava SNMP | | čas | določi sistem |
| Trajanje sprejema | | čas | določi sistem |
| Odgovor SNMP | | čas | določi sistem |
| Oddaja vožnje | | čas | določi sistem |
| Začetek vožnje | | čas | določi sistem |
| Na kraju dogodka | | čas | določi sistem |
| S kraja dogodka | | čas | določi sistem |
| Na cilju | | čas | določi sistem |
| Prost | | čas | določi sistem |
| lokacija dogodka | Šifra ustanove | številka | vnese uporabnik iz šifranta |
| | ustanova | niz | vnese uporabnik iz šifranta / vnese uporabnik |
| | občina | niz | vnese uporabnik iz šifranta |
| | naselje | niz | vnese uporabnik iz šifranta |
| | hišni naslov | niz | vnese uporabnik iz šifranta |
| | hišna številka | niz | vnese uporabnik iz šifranta |
| | opis | niz | vnese uporabnik |
| | X | število | določi sistem |
| | Y | število | določi sistem |
| Končna lokacija | Šifra ustanove | številka | vnese uporabnik iz šifranta |
| | ustanova | niz | vnese uporabnik iz šifranta / vnese uporabnik |
| | občina | niz | vnese uporabnik iz šifranta |
| | naselje | niz | vnese uporabnik iz šifranta |
| | hišni naslov | niz | vnese uporabnik iz šifranta |
| | hišna številka | niz | vnese uporabnik iz šifranta |
| | opis | niz | določi sistem |
| | X | število | določi sistem |
| | Y | število | vnese uporabnik iz šifranta |
| Šifra prioritete | | številka | vnese uporabnik iz šifranta |
| Šifra ekipe | | številka | določi sistem |
| Šifra namembnosti | | številka | vnese uporabnik iz šifranta |
| Šifra izpostave | | številka | vnese uporabnik iz šifranta |
| Šifra vozila | | številka | vnese uporabnik iz šifranta |
| Šifra voznika | | številka | vnese uporabnik iz šifranta |
| Šifra spremljevalca | | številka | vnese uporabnik iz šifranta |
| Šifra zdravnika | | številka | vnese uporabnik iz šifranta |
| Šifra lokacija prost | | številka | določi sistem |
| Naročila ekipe | | niz | vnese uporabnik |
| Šifra režima vožnje | | številka | vnese uporabnik iz šifranta |
| Je deaktivirana | | številka | vnese uporabnik |
| Je vsa dokumentacija | | logična vrednost | vnese uporabnik |
| Številka naloga | | niz | vnese uporabnik |

| | | |
|---|---|---|
| Šifra vnosnega dispečerja | številka | določi sistem |
| Šifra izvirnega dispecerja | številka | določi sistem |
| Šifra oddajnega dispečerja | številka | določi sistem |

Za določene podatke se morda izbira tipa ne zdi logična. Tako je na primer tip podatka število pacientov kar niz. Razlog temu je, da s tem podatkom ne izvajamo nobenih računskih operacij (oziroma jih le redko in takrat pač podatek pretvorimo iz niza v število). Z uporabo niza omogočimo, da kadar je število pacientov neznano, v bazo zapišemo niz »?«

# 3 TEHNOLOGIJA ZA RAZVOJ APLIKACIJE

Glavni namen računalniško podprtega dispečerstva je, da se vsi podatki, ki jih uporabniki vnesejo, zabeležijo v podatkovni bazi. Do te baze potem dostopa več ljudi iz več računalnikov znotraj organizacije.
Za razvoj uporabniškega vmesnika, ki omogoča dostop do skupnih podatkov, v osnovi obstajata dva možna načina: razvoj spletne aplikacije in razvoj namizne aplikacije.
Spletna aplikacija je program, ki je nameščen na enem samem računalniku, povezanem v omrežje in do katerega lahko dostopa več računalnikov hkrati s poljubnega mesta znotraj omrežja. Na centralnem računalniku teče spletni strežnik in znotraj tega ustrezna aplikacija. Vsi podatki, ki jih aplikacija potrebuje za delovanje in tudi vsi vnosi uporabnikov, se hranijo v bazi podatkov, ki teče na tem centralnem računalniku. Uporabniški vmesnik je razvit kot spletna stran, omrežje za povezavo pa je internet. Zato so spletne aplikacije primerne predvsem takrat, kadar želimo do aplikacije dostopati preko javnega omrežja. Ker je aplikacija za računalniško podprto dispečerstvo namenjena uporabi le znotraj organizacije na določenih računalnikih, ta tehnologija ni primerna. Vprašljiva bi bila tudi sama hitrost in odzivnost tako pripravljenega programa.
Namizna aplikacija je program, ki je nameščen na uporabnikovem računalniku in se tam tudi v celoti izvaja. Ker je računalnikov, ki delajo z istimi podatki več, je tudi tu potrebno poskrbeti, da se podatki hranijo v centralni bazi podatkov. Ta je skupna in ni na uporabnikovem računalniku. Torej tudi tu obstaja centralni računalnik, ki nadzira uporabo te skupne baze. Računalnik, na katerem teče aplikacija, ima povezavo do baze. Program zahteva od baze določene podatke, jih dobi, obdela na lokalnem računalniku in obdelane pošlje nazaj v centralno bazo. V našem primeru je prednost namizne aplikacije tudi v tem, da če želijo uporabniki z novega računalnika dostopati do podatkov, moramo nanj namestiti ustrezni program. Tako imamo kontrolo nad dostopom in širjenjem programa. Dejstvo je tudi, da so namizne aplikacije praviloma funkcionalno bogatejše od spletnih. Izvajajo se v nadzorovanem okolju in tako nam ni potrebno upoštevati minimalnega skupnega imenovalca brskalnikov ter omejitev, ki jih postavljajo sami internetni protokoli.
Odločili smo se torej, da bomo našo aplikacijo razvili kot namizno aplikacijo, ki se bo povezovala s centralno bazo. Za razvoj same aplikacije bomo uporabili programski jezik Visual Basic in pripadajoče razvojno okolje. Podatke v bazi pa bomo upravljali s sistemom MSSQL. O samem jeziku Visual Basic si bomo nekaj najnujnejših stvari ogledali v naslednjem razdelku.

## 3.1 Visual basic 6.0

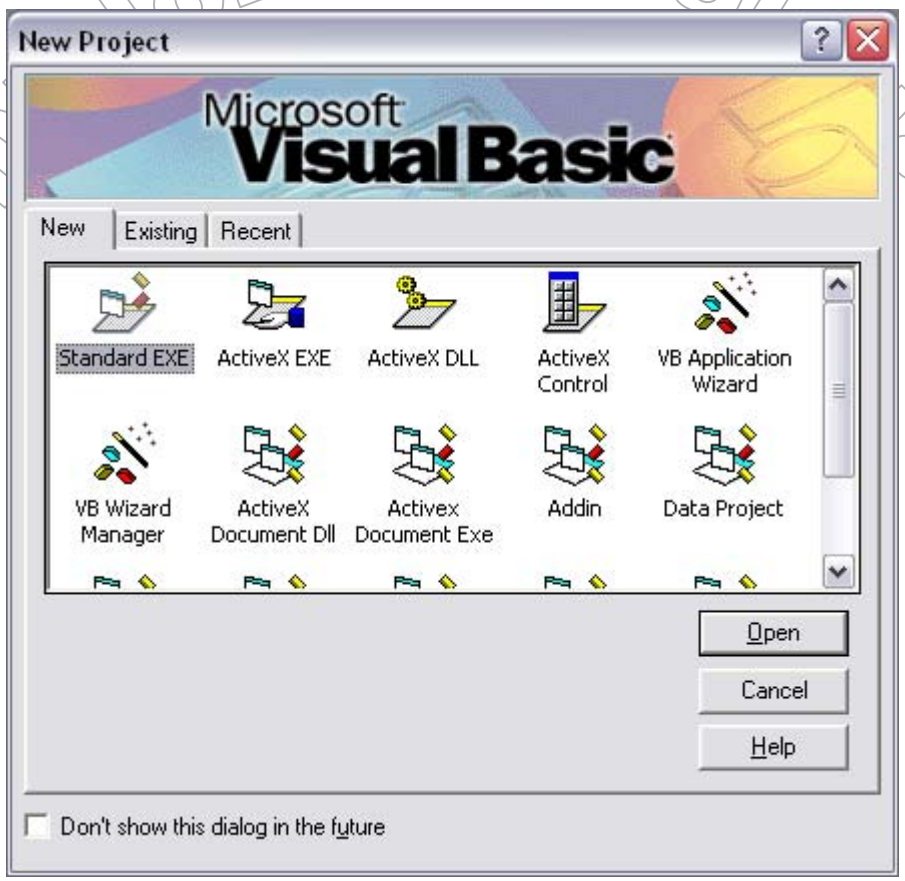
Visual basic (VB) je po eni strani programski jezik, po drugi strani pa okolje za hitro razvijanje aplikacij za operacijski sistem Windows. Razvili so ga pri družbi Microsoft. Sam program vsebuje veliko število knjižnic za enostavno gradnjo grafičnih vmesnikov. Le tega gradimo s pomočjo t.i.vizualnega programiranja.
Natančneje si bomo nekaj korakov gradnje uporabniškega vmesnika s pomočjo vizualnega programiranja ogledali v poglavju 4, ko bomo predstavili nekatere funkcionalnosti naše aplikacije. Tukaj pa le v grobem opišimo, za kaj gre pri tem procesu.
Ko oblikujemo uporabniški vmesnik, nanj polagamo gradnike. Gradnik izberemo iz palete vnaprej pripravljenih gradnikov in ga prenesemo na obrazec. Tam mu določimo lastnosti, ter sestavimo ustrezne podprograme, ki povedo, kako se mora gradnik obnašati ob določenem dogodku (npr. ob kliku miške nanj)

### 3.1.1 Razvojno okolje

Po zagonu razvojnega okolja Visual Basic 6.0 se najprej odpre pogovorno okno New Project, ki ima tri razdelke:

- New naredimo nov projekt

- Existing poiščemo in odpremo obstoječ projekt
- Recent prikažemo seznam projektov, ki smo jih nazadnje izdelali



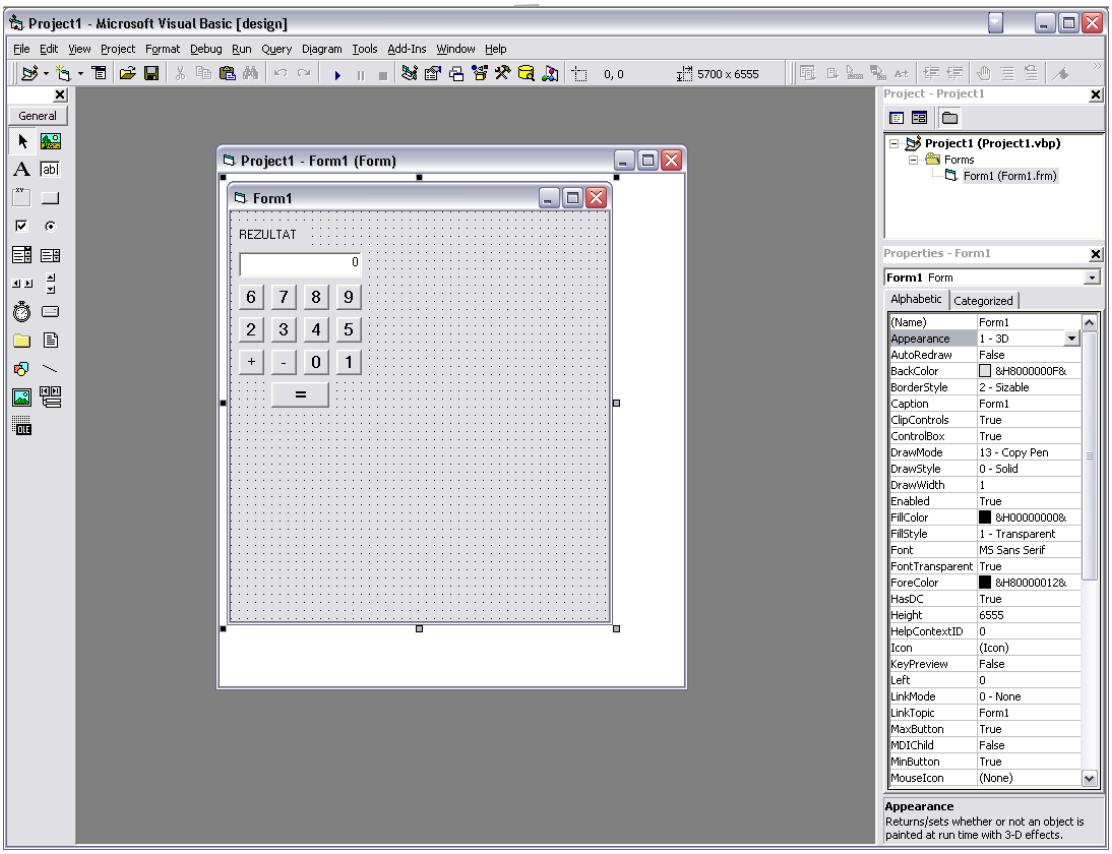

**Slika 11**
**Izbira projekta**

Vsak nov projekt je izdelan na osnovi predloge, za katero se odločimo v mapi New. Omeniti velja, da je število predlog odvisno od različice Visual Basica. Najpogosteje uporabljene predloge so:

- **Standard EXE** izdelamo navaden program, ki deluje v okolju Windows
- **VB Application Vizard** izdelamo ogrodje tipičnega programa, ki ga lahko potem nadgrajujemo
- **ActiveX Control** izdelamo kontrolo ActiveX, ki jo nato shranimo v datoteko s podaljškom .ocx. Te kontrole lahko uporabimo pri razvoju programov
- **ActiveX DLL** izdelamo avtomacijski strežnik, ki nastopa v obliki knjižnice DLL

Pri razvoju naše aplikacije bomo uporabili osnovno predlogo Standard EXE.

**Programsko okno**
Po zagonu novega projekta nas sistem postavi v programsko okno. Najprej si oglejmo njegov videz in osnovne elemente urejevalnika.



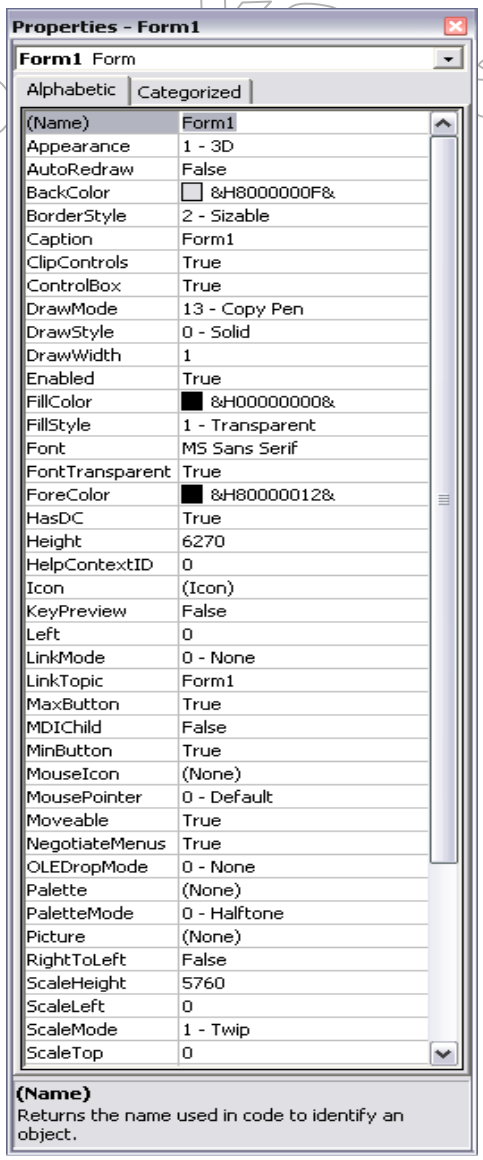

**Slika 12**
**Programsko okno**

Najpomembnejši sklopi programskega okna so:

- **Vrstica z meniji,** kjer najdemo vse ukaze, ki so nam na voljo v urejevalniku.
- **Orodna vrstica**, ki nam omogoča hiter dostop do najuporabnejših ukazov ki jih lahko poženemo z klikanjem na ikone.
- **Okvir z orodji** (skrajno levo na sliki 12), kjer se nahajajo raznovrstne kontrole, ki jih med pripravo vstavljamo v obrazec.
- **Obrazec** (osrednji del okna) predstavlja vmesnik med programom in uporabnikom. Nanj uporabnik nalaga gradnike.
- **Okno project** (na sliki 12 desno zgoraj) je zbirka različnih vrst datotek, ki skupaj tvorijo program.
- **Okno properties** ( slika 12 desno) prikazuje lastnosti obrazca in gradnikov, ki so na njem.
- **Okno s programsko kodo** (osrednji del okna – imenjuje se s prikazom obrazca) je okno, kamor zapisujemo kodo oziroma ukaze jezika Visual Basic. Vsak obrazec ima okno s pripadajočo kodo. Projekt lahko vsebuje tudi kodo, ki je zapisana v raznih modulih.

**Uporabniški vmesnik**

Ko oblikujemo uporabniški vmesnik, nanj polagamo gradnike. Gradniki so osnovni elementi uporabniškega vmesnika. V razvojnem okolju jih imamo na voljo celo vrsto. Gradnik izberemo v okviru z orodji, tako, da nanj kliknemo in ga povlečemo na obrazec. Tam nanj ponovno kliknemo ter mu z vlečenjem določimo ustrezno velikost. Z miško lahko gradnike premikamo ter raztezamo in krčimo v vse smeri, kar nam omogočajo oprijemke.
Izgled posameznega gradnika določajo njegove lastnosti (properties). Le-te so zapisane v posebnem oknu, ki se nam za posamezni gradnik pokaže ob kliku nanj. Takšne lastnosti so denimo ime (name), višina (height), širina (width), barva ozadja (backcolor) ...




**Slika 13**
**Okno z lastnostmi**

Lastnosti gradnika, ki jih nastavljamo in spreminjamo preko pregledovalnika lastnosti, predstavljajo tiste lastnosti, ki jih ima gradnik ob zagonu programa. Vse te lastnosti lahko spreminjamo tudi kasneje, med samim izvajanjem aplikacije. Za to napišemo ustrezno programsko kodo. Obstajajo pa tudi takšne lastnosti gradnikov, ki jih je mogoče spreminjati zgolj s programsko kodo.
Lastnosti, ki jih gradnik ima, so odvisne od tega, za kakšno vrsto gradnika gre. Tako ima gradnik slikovni okvir (picture box) lastnost picture, ki ji priredimo ustrezno datoteko na disku in predstavlja sliko, ki se prikaže znotraj tega okvira. Gradnik tekstni okvir (text box) te lastnosti nima, ima pa lastnost besedilo (text), ki je poljubna kombinacija znakov.
Pri gradnji uporabniškega vmesnika je osnovni gradnik obrazec (form). Nanj nalagamo vse ostale gradnike. Ko zaženemo sam Visual Basic, nam že sam program ponudi osnovni obrazec.

### 3.1.2 Gradniki in njihove lastnosti

Tu si bomo ogledali nekaj najpomembnejših gradnikov, ki smo jih uporabili pri tvorbi naše aplikacije. Navedli bomo tudi tiste ključne lastnosti teh gradnikov, ki so nam prišle najbolj

prav. Osnovne, kot so Name, Width, Height in podobne z jasnim pomenom, ne bomo opisovali.

**TextBox**

Okence za besedilo omogoča najpreprostejši način, da uporabnik vpiše neko besedilo. Uporabimo ga lahko tudi za prikaz podatkov, ki jih program vrne.

Poglejmo si nekaj lastnosti, ki sem jih uporabili v programu:

- **Text** je osnovna lastnost in predstavlja besedilo, ki je v okencu.
- **MultiLine** pove, da v okence lahko vnesemo več vrstic besedila.
- **Aligment** določa poravnavo besedila v okencu. Vrednost je moč spreminjati tako v času ustvarjanja uporabniškega vmesnika, kot tudi ob delovanju programa. Poravnava je možna le, če smo lastnosti MultiLine priredili vrednost True. Vrednosti, ki jih lahko določimo lastnosti Aligment, so:
  - 0 – besedilo je poravnano na levi rob okenca(privzeta vrednost),
  - 1 – besedilo je poravnano na levi rob okenca,
  - 2 – besedilo je poravnano na sredino okenca.

  Oblika zapisa v programski kodi je: `ImeGradnika.Aligment = Vrednost`
- **PasswordChar** določa znak, ki se namesto natipkanega znaka izpisuje v okencu, ko vanj vpisujemo neko besedilo. To je uporabno za pisanje gesel, ko želimo preprečiti, da bi drugi videli, kaj je uporabnik vtipkal. Običajno se uporabi znak *, čeprav lahko uporabimo tudi poljuben drug znak. Če lastnosti priredimo vrednost "" (privzeta vrednost), postane to običajno okence za pisanje besedila. Oblika zapisa je: `ImeGradnika.PasswordChar = Znak`
- **ScrollBars** določa, ali je okence z besedilom opremljeno z drsniki. To lastnost lahko določamo le med izdelavo uporabniškega vmesnika, med samim izvajanjem pa je ne moremo spreminjati. Če jo želimo uporabiti, moramo najprej prirediti lastnosti MultiLine vrednost True. Vrednosti, ki jih lahko priredimo lastnosti ScrollBars, so:
  - 0 – brez drsnikov (privzeta vrednost),
  - 1 – vodoravni drsnik,
  - 2 – navpični drsnik,
  - 3 – oba drsnika.

  Oblika zapisa je: `ImeGradnika.ScrollBars = Vrednost`

**CommandButton**

Ukazni gumb je morda najpomembnejši gradnik, ki je skoraj nepogrešljiv v vsakem programu. Z ukaznimi gumbi običajno prožimo razne postopke, ki so sestavni del programov.

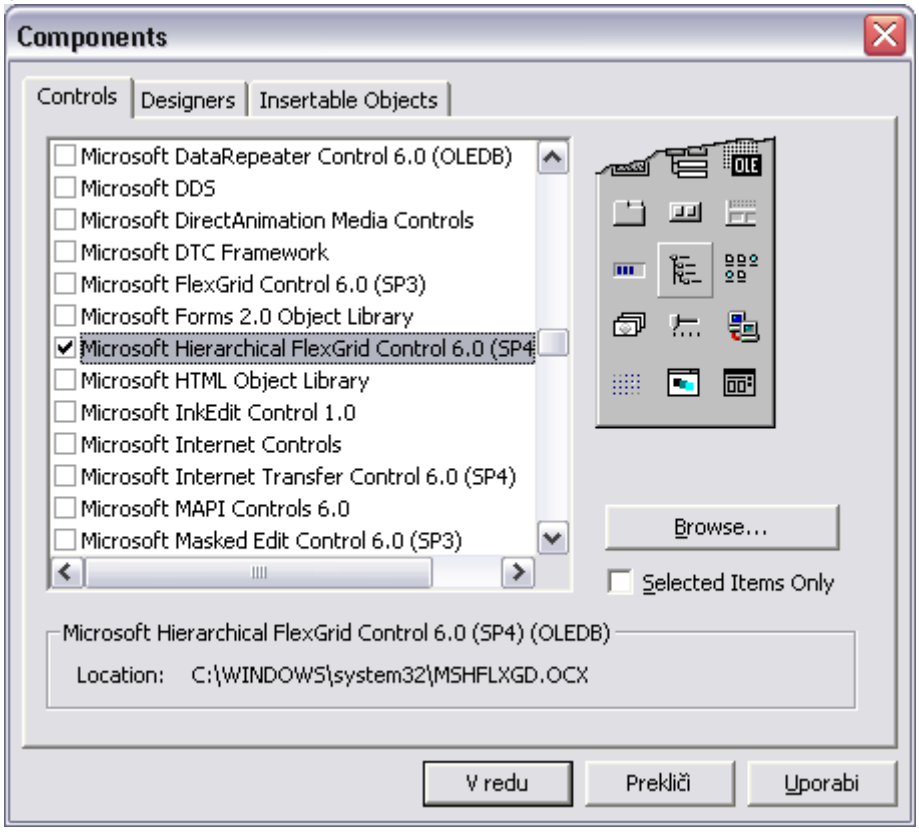
Poglejmo si nekaj lastnosti, ki sem jih uporabil v programu:

- **Style** določa, ali je na ukazni gumb moč vstaviti sliko ali ne. Te lastnosti ne moremo spreminjati med delovanjem programa. Vrednosti, ki ju lahko priredimo lastnosti Style, sta 0 – Na gumb ne moremo vstaviti slike, ampak le besedilo (privzeta nastavitev) in 1 – Na ukazni gumb lahko vstavimo sliko, kar lahko storimo z lastnostjo Picture. Oblika zapisa je `ImeGradnika.Style = Vrednost`. Če ima lastnost Style prirejeno vrednost 1, potem lahko gumbu določimo tudi barvo z lastnostjo **BackColor,** sicer pa ne.
- **Picture** določa sliko (navedemo datoteko, kjer slika je), ki se pojavi na ukaznem gumbu, ko je ta dostopen in ni pritisnjen. Sliko lahko izberemo med izdelavo programa ali pa med izvajanjem s funkcijo LoadPicture. Slika se prikaže na sredini gumba. Če pa gumb vsebuje napis (lastnost **Caption**), se slika pojavi nad napisom. Prevelike slike se samodejno obrežejo.

**MSHFlexGrid**

Ta gradnik je namenjen prikazu podatkov v tabelarični obliki. Omogoča razvrščanje in oblikovanje elementov, ki so lahko tekstovni ali pa slikovni. Gradnik je moč povezati s podatkovno zbirko, vendar ne omogoča urejanja, temveč le prikaz tovrstnih podatkov.

V samem okvirju z orodji tega gradnika privzeto ni. Če ga želimo uporabiti, ga moramo najprej naložiti v okvir. To storimo z ukazom Project, Components. V pogovornem oknu nato poiščemo in označimo gradnik, ali kot mu v terminologiji jezika Visual Basic tudi rečemo, kontrolo, Microsoft Hierarchical FlexGrid Control 6.0. Kontrola MSHFlexGrid je shranjena v datoteki Mshflxgd.ocx




**Slika 14**
**Dodajanje kontrole v okvir**

Kontroli MSHFlexGrid lahko veliko lastnosti določimo že med izdelavo programa. Potem, ko kontrolo postavimo na obrazec, nanjo kliknemo z desnim gumbom miške in izberemo ukaz Properties.

Poglejmo si nekaj pomembnejših lastnosti gradnika MSHFlexGrid:

- **CellPicture** določa sliko, ki se pojavi v celici. Posamezni celici lahko sliko določimo le s kodo. Sliko določimo s funkcijo LoadPicture ali kako drugače.
- **Col** določa stolpec, ki ga želimo označiti. V povezavi z lastnostjo **Row,** ki določa vrstico, ki jo želimo označiti, nam omogoča, da označimo poljubno celico. Oblika zapisa je `ImeGradnika.Col = StevilkaStolpca` oziroma `ImeGradnika.Row = StevilkaVrstice`
- **ColIsVisible** določa, ali je neki stolpec viden (vrednost True) ali ne (vrednost False). To pomeni, da lahko po potrebi določene stolpce skrijemo. Oblika zapisa je: `ImeGradnika.ColIsVisible(index) = Vrednost.` Indeks je številka stolpca, ki ga želimo skriti.
- **`RowIsVisible`** določa, ali je neka vrstica vidna (vrednost True) ali ne (vrednost False). To pomeni, da lahko po potrebi določene vrstice skrijemo.

Oblika zapisa je: `ImeGradnika.RowIsVisible(index) = Vrednost.` Indeks je številka vrstice, ki jo želimo skriti.

- **TextMatrix** določa vsebino poljubne celice, ne da bi morali pri tem spremeniti vrednosti lastnosti Col in Row. Lastnost TextMatrix ima dva argumenta. Prvi je zaporedna številka vrstice, drugi pa zaporedna številka stolpca, kamor bomo zapisali nek podatek, ali od koder ga želimo prebrati. Oblika zapisa je: `ImeGradnika.TextMatrix(vrstica, stolpec)`

### 3.1.3 Pisanje kode

Risanju uporabniškega vmesnika in nastavitvi osnovnih lastnosti sledi pisanje programske kode. To se dogaja v posebnem oknu, kjer sta na voljo dva seznama. V prvem so imena gradnikov, ki se nahajajo na vmesniku, v drugem pa dogodki (events), ki se nad izbranim gradnikom lahko zgodijo. Pogostejši dogodki so klik (Click), dvojni klik (DblClick), sprememba(Change), uporabnik je čez gradnik povlekel miško (DragOver) ... Po izbiri v obeh seznamih se na zaslonu pokaže koda v naslednji obliki:

```
Private Sub ImeGradnika_Dogodek ()

End Sub
```

Koda, ki jo vpišemo znotraj metode, se izvrši takrat, ko se nad gradnikom (`ImeGradnika`) zgodi ustrezen dogodek (`Dogodek`). Gradniku ukazni gumb (command button) in dogodku klik tako lahko priredimo kodo, ki se bo izvedla, ko bo uporabnik kliknil na ta gumb. Sam program je tako sestavljen iz posameznih podprogramov, ki se izvajajo ob ustreznih dogodkih na določenih gradnikih. Kodo (podprograme) pišemo s programskimi stavki. V njih uporabljamo rezervirane besede (if, else, next ...), konstante, spremenljivke, imena gradnikov z njihovimi ostalimi lastnostmi in metodami ... .

**Spremenljivke**

Programski jezik BASIC načeloma ne zahteva deklaracij (najav) spremenljivk. Ker pa nam najava spremenljivk pomaga pri iskanju napak, lahko v jeziku Visual Basic v vsakem modulu posebej zahtevamo obvezno najavo spremenljivk. Ta najava se zapiše v posebni deklaracijski del modula. Zahteva se glasi:

```
Option Explicit
```

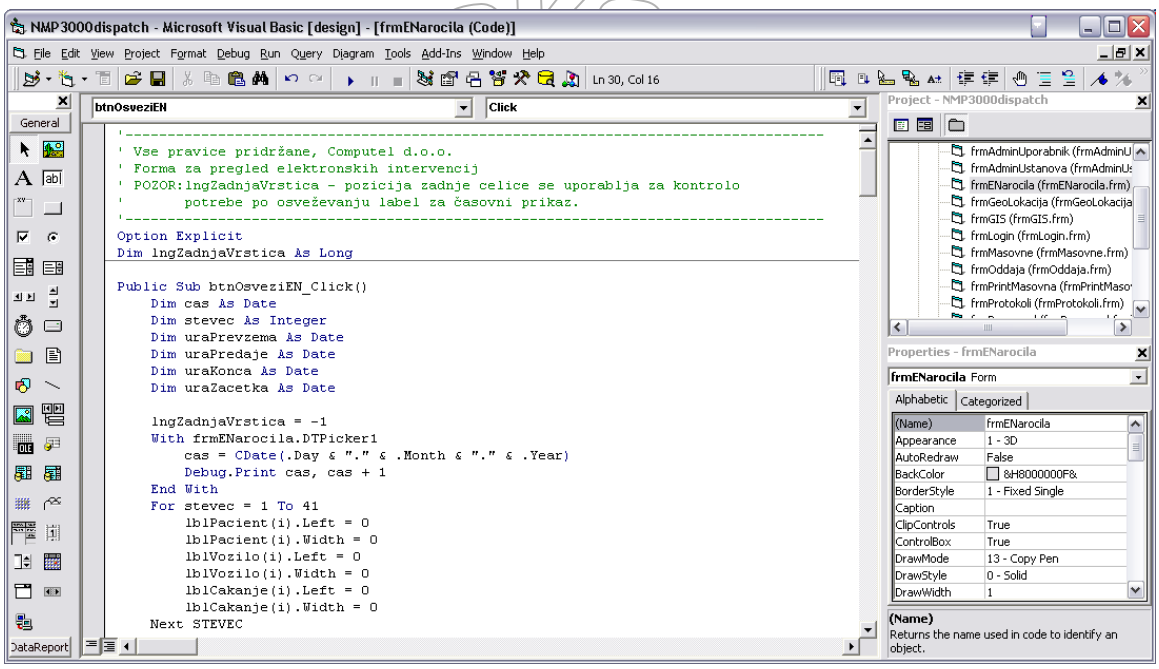
Če smo z Option Explicit zahtevali obvezno najavo spremenljivk, je spremenljivke pred prvo uporabo potrebno najaviti. Če je ne najavimo, prevajalnik javi napako. Kadar najave ne zahtevamo, se predvideva, da je bila spremenljivka najavljena na tistem mestu v kodi, kjer smo jo prvič uporabili.

Pri spremenljivkah sta poleg tipa pomembna še doseg in življenjska doba spremenljivke. Ti dve lastnosti sta odvisni od načina in mesta najave spremenljivke v kodi. Splošna oblika najave se glasi:

```
DeklaracijskaRezerviranaBeseda ImeSpremenljivke As TipSpremenljivke
```

Za ImeSpremenljivke veljajo podobna pravila kot pri večini progrmskih jezikov. Omenimo pa, da v nasprotju s programskim jezikom Java, Visual Basic ne razlikuje malih in velikih črk. Spremenljivko lahko najavimo v posebnem deklaracijskem delu, ki se nahaja na vrhu vsakega okna, kjer pišemo kodo, ali pa znotraj nekega podprograma. Pri najavljanju spremenljivke znotraj podprograma se držimo dogovora, da vse spremenljivke najavljamo na začetku podprograma. S tem omogočimo lažje in bolj razumljivo branje kode.

Do posebnega deklaracijskega dela pridemo preko že omenjenih seznamov gradnikov in lastnosti.




**Slika 15**
**Prikaz najavljanja spremenljivk**

Če bomo spremenljivko najavili v deklaracijskem delu, imamo dve možnosti. Ob uporabi rezervirane besede `Global` bo spremenljivka dostopna celotni aplikaciji. Njena vrednost se bo hranila, dokler se bo aplikacija izvajala. Deklaracija z `Global` v deklaracijskem delu torej pomeni, da je spremenljivka dostopna po celi aplikaciji, njena življenska doba pa je enaka času delovanja aplikacije. Zgled take deklaracije je

```
Global jazSemDostopnaVsem As integer;
```

Če spremenljivko v deklaracijskem delu najavimo z `Dim`, bo spremenljivka dostopna le v modulu, kjer je bila najavljena. Njena življenska doba je enaka času izvajanju kode v tem modulu. Ko zapustimo modul, v katerem smo spremenljivko najavili, se njena vrednost izgubi.
Ob najavi spremenljivke v podprogramu se zopet odpirata dve možnosti. Spremenljivko lahko deklariramo s `Static` ali pa z `Dim`.

```
Static doMeneSePrideLeVPodprogramu As double;
```

Najava s `Static` ali `Dim` pomeni, da je spremenljivka vidna le podprogramu. Razlika pri najavi je v življenjski dobi. Če najavimo spremenljivko z `Static`, je njena življenska doba čas trajanja aplikacije. Pri najavi z `Dim` pa je njena življenjska doba enaka izvajanju podprograma. Razložimo to na primeru:

```
Private Sub Povecuj_Click ()
      Dim stevec As Integer
      stevec = stevec + 1
End Sub
```

S to kodo smo želeli doseči, da bi se ob vsakem kliku na ukazni gumb `Povecuj` vrednost spremenljivke `stevec` povečala za 1. Toda podprogram ne deluje na ta način. Pri tovrstni najavi spremenljivke `stevec` njena življenjska doba traja zgolj do novega dogodka. Ker ni

bila uporabljena deklaracija s `Static`, bo vrednost ob vsakem kliku 1. Koda pa deluje tako, kot je bilo zamišljeno.

```
Private Sub Povecuj_Click ()
      Static stevec As Integer
      stevec = stevec + 1
End Sub
```

Pri opravilih s spremenljivkami uporabljamo operatorje. Najpogostejši so +, -, *, / ter &, ki ga uporabljamo za stikanje nizov, kot na primer:

```
Dim a As String
Dim b As String
a = "Tekstni"
b = "Okvir"
Napis.Text = a & b 'V lastnosti Text je sedaj besedilo TekstniOkvir
```

V zgledu smo uporabili tudi komentar. Znak `'` igra enako vlogo kot `//` v Javi. Z njim napovemo komentar, ki se konča s koncem vrstice.
V zgledu vidimo tudi primer, ko dostopamo do lastnosti gradnika. Ker želimo spremeniti lastnost `Text` gradnika `Napis`, uporabimo `Napis.Text` kot del prireditvenega stavka.

**Stavki**
Poleg že omenjenih deklaracijskih stavkov si poglejmo še nekaj najpogosteje uporabljenih stavkov.

**Prireditveni stavki**

```
Cilj = PoljubenIzraz
```

Cilj je lahko bodisi spremenljivka, bodisi lastnost nekega gradnika. Dejansko bi lahko namesto Cilj napisali kar Spremenljivka, saj so tudi lastnosti gradnikov v bistvu spremenljivke. Le njihova imena so vnaprej določena kot `ImeGradnika.Lastnost`.

**Metode**
Nad posameznimi vrstami gradnikov je mogoče izvajati metode. Le-te so že narejeni podprogrami nad nekim gradnikom. Skladnja stavka, kjer uporabljamo klic metode, je:

```
ImeGradnika.Metoda Argument
```

Če je argumentov več, jih med sabo ločimo z vejicami. Tako na primer v gradnik seznam (list box) novo postavko dodamo s klicem metode `additem`:

```
List1.Additem "Nov zapis"
```

**Odločitveni stavki**
Odločitvene stavke uporabljamo, kadar želimo, da se nek sklop programske kode izvede le, če je izpolnjen nek pogoj.

Najpogostejši stavek te vrste je stavek if, ki ima naslednjo obliko:

```
If pogoj then
      koda
End if
```

Pogoj se napiše v obliki izjave ali kombinacije izjav. Napišemo lahko:

```
If UporabnikovOdgovor.text <> "Kranj" then
      UporabnikovOdgovor.text = "Napačno"
else
      UporabnikovOdgovor.text = "Pravilno"
End if
```

Ob tako napisani kodi se bo v tekstni okvir UporabnikovOdgovor vpisalo "Pravilno" le, če bo prej v njem pisalo "Kranj".

Kot alternativa gnezdenim ali zaporednim stavkom `if...then` se v Visual Basicu uporablja tudi programski blok `Select Case`. Programska koda je z uporabo tega stavka bolj pregledna in tudi malenkost hitrejša.

Blok `Select Case` najprej ovrednoti izraz, potem pa na podlagi njegovega izida izbere mesto izvajanja programa. Sintaksa je naslednja:

```
Select Case izraz
Case prvi seznam izidov
      Prvi blok stavkov
Case drugi seznam izidov
      drugi blok stavkov
...
Case Else
      zadnji blok stavkov
end Select
```

Za primerjavo si poglejmo, kako napišemo zgornji primer z blokom Select Case.

```
Select Case UporabnikovOdgovor.text
Case "Kranj"
      UporabnikovOdgovor.text  =  "Pravilno"
Case Else
      UporabnikovOdgovor.text  =  "Napačno"
End Select
```

**Zanke**

Včasih želimo, da se določena koda izvrši večkrat. Tedaj uporabimo zanke. Najpogostejša je zanka `For...Next`. Za besedo `For` priredimo spremenljivki začetno vrednost. Spremenljivka je števec zanke. Določa, kolikokrat se zanka ponovi. Števec se povečuje ali zmanjšuje v korakih po ena, vendar se lahko odločimo tudi za kako drugo vrednost, ki jo zapišemo za besedo `Step`. Ko se zanka prvič izvede, se uporabi začetna vrednost. Pri naslednjem izvajanju zanke se števec poveča ali zmanjša za korak, ki smo ga podali s `Step`. Izvajanje zanke se ponavlja, dokler je števec manjši (oz. večji v primeru negativnega koraka) od končne vrednosti. Če je začetna vrednost manjša od končne, se zanka sploh ne izvede, razen seveda v primerih, ko se števec zmanjšuje.

```
For Spremenljivka=zacetnaVrednost to koncnaVrednost step korak
      koda
Next Spremenljivka
```

Če je `korak` enak 1, potem del `step` lahko izpustimo:

```
For Spremenljivka = zacetnaVrednost to koncnaVrednost
      koda
Next Spremenljivka
```

Poglejmo si delovanje zanke `For` še na primeru:

```
For Katera = 1 to 10
      OceneLestvica.additem katera
Next Katera
```

Ta koda doda v seznam `OceneLestvica` deset novih elementov, naravnih števil od 1 do 10.

Pogosto števila ponovitev ne moremo opredeliti s številom korakov zanke. Želimo, da se nekaj dogaja toliko časa, dokler je izpolnjen nek pogoj. Tedaj uporabimo zanko `While...Wend`. Ta ima obliko:

```
While pogoj
      stavki
Wend
```

Stavki se bodo izvajali, dokler bo pogoj resničen (izpolnjen). Paziti moramo, da imajo stavki v zanki določen vpliv na resničnost pogoja, sicer se bodo izvajali v nedogled. Primer take zanke, ki prešteje, kolikokrat lahko razpolovimo število 4.7, da je to manjše od števila 1, je:

```
Stevilo = 4.7
KolikokratRazpolovili = 0

While Stevilo > 1
      Stevilo = Stevilo/2
      KolikokratRazpolovili = KolikokratRazpolovili + 1
Wend
```

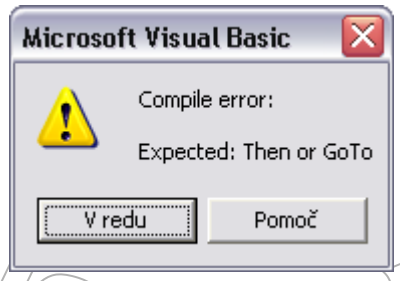
**Razhroščevanje**

Razhroščevanje oziroma odpravljanje napak je pomemben del programiranja, saj se pri pisanju programov vedno pojavijo večje ali manjše semantične (pomenske) napake. Zato program ne deluje, kot smo si zamislili, da bo.

Razvojno okolje Visual basic ponuja orodje za razhroščevanje, pa tudi orodja za sprotno odpravljanje napak. Tako se lahko odločimo za preverjanje skladnje ukazov, sproti spremljamo vrednosti spremenljivk, izvajamo posamezne vrstice kode ali posamezne procedure.

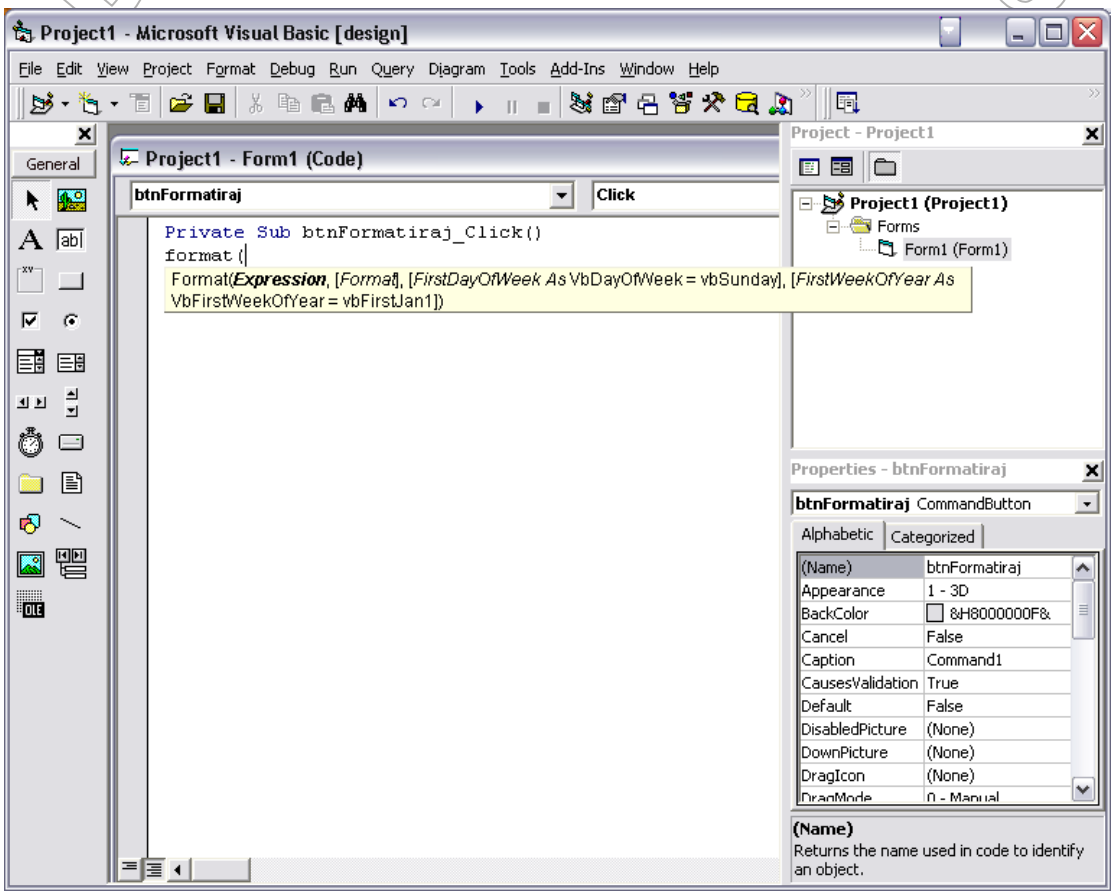
**Napake pri tipkanju**

Visual Basic pozna orodje, ki sproti preverja skladnjo izrazov, medtem ko jih zapisujemo. Če v neki vrstici naredimo napako, nas to orodje opozori nanjo, brž ko se premaknemo v novo vrstico.



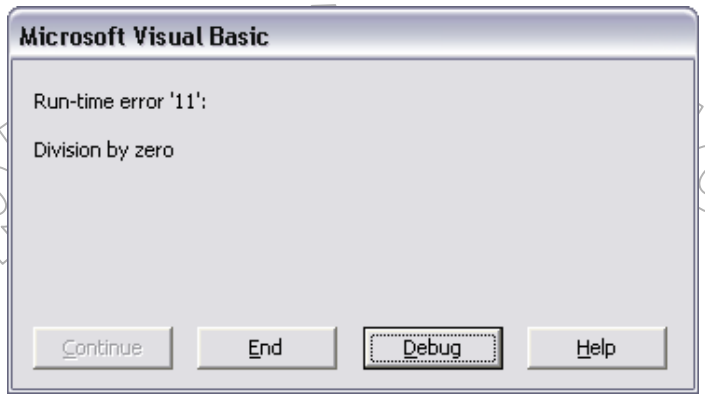

**Slika 16**
**Javljanje napake pri pisanju kode**

Med tovrstne napake sodijo napačno zapisane ključne besede, mankajoči deli izrazov, kot so oklepaji in podobno.
Orodje za preverjanje napak med tipkanjem je običajno aktivirano. Če ni, ga lahko aktiviramo z ukazom Tools, Options. V razdelku Editor označimo izbiro Auto Syntax Check.
Omeniti velja še eno koristno orodje, ki skrbi za pravilen vnos stavkov in funkcij, orodje za prikaz argumentov funkcij. Aktiviramo ga z ukazom Tools, Options in v razdelku Editor potrdimo polje Auto List Members. To orodje nam prikaže, kakšni morajo biti argumenti ob klicu posamezne funkcije.




**Slika 17**
**Pomoč pri pisanju argumentov funkcije**

Pri lovljenju tipkarskih napak nam pomaga tudi pravilo, ki se ga Visual Basic drži glede velikih črk. Načeloma prevajalnik za jezik Visual Basic ne dela razlike med velikimi in malimi črkami. Omogoča pa nam, da se spremenljivka, ki jo najavimo na primer kot `ImeSpremenljivke`, vedno izpiše v taki obliki (torej z velikima I in S). To je lahko zelo učinkovit način za prestrezanje napak, saj bi računalnik morebitno tipkarsko napako razumel kot uvedbo nove spremenljivke.
Za večjo preglednost kode se uporabljajo tudi barve. Visual Basic samodejno poskrbi, da so ključne besede, kot so imena zank, vgrajenih funkcij in podobno, zapisane z veliko začetnico in prikazane v modri barvi. Z eno barvo so označene tudi konstante, nizi, … . Tako že sam pogled na kodo programerju pomaga, da loči med posameznimi sestavnimi deli programa.
Če med izvajanjem programa pride do napake, se pojavi pogovorno okno s številko napake in kratkim opisom:




**Slika 18**
**Opis Napake**

Če je možno, lahko s klikom na gumb Continue, nadaljujemo z izvajanjem. Gumb Help nam nudi opis napake, ki se je zgodila, medtem ko z gumbom End končamo izvajanje programa. Če pritisnemo gumb Debug, se vrnemo v kodo. Visual Basic nam z rumeno barvo označi vrstico, v kateri je prišlo do napake.

**Odpravljanje napak**
Za iskanje in odpravljanje napak obstajajo orodja, ki jih aktiviramo z ukazom View, Toolbars, Debug.


**Slika 19**
**Orodna vrstica za odpravljanje napak**

Puščico kliknemo za začetek izvajanja programa in za nadaljevanje programa, če smo ga prekinili. Dve pokončni črti kliknemo, ko želimo začasno ustaviti izvajanje programa. S tem preidemo v način break. Kvadrat kliknemo, ko želimo končati z izvajanjem programa.
S klikom na ikono z roko trenutno vrstico kode označimo kot prekinitveno točko. V tej vrstici se bo program ob izvajanju ustavil in prešel v način break.
Okno z ikono puščica korak niže omogoča, da v načinu break izvedemo naslednjo vrstico kode, ne glede na to, če bomo s tem vstopili v naslednjo funkcijo ali proceduro.
Z ikono s puščico čez izvedemo naslednjo vrstico kode. Če je to funkcija ali procedura, se le ta izvede v celoti.
S puščico korak više se do konca izvede funkcija oziroma procedura v kateri smo, nato se izvajanje ustavi.
V primeru, da kliknemo na ikono okno, se odpre pogovorno okno Locals, v katerem spremljamo vrednost vseh spremenljivk, ki so definirane v trenutno aktivni proceduri.
Klik na okno s klicajem odpre okno Immediate, ki v načinu break omogoča ročno izvajanje posameznih vrstic kode.
Okno z očali omogoča spremljanje vrednosti vseh spremenljivk, ikona očal pa spremljanje vrednosti tiste spremenljivke, ki si jo izberemo.
S klikom na ikono z več okni dobimo vpogled v to, katere procedure in funkcije se trenutno izvajajo.

## 3.2 MS SQL

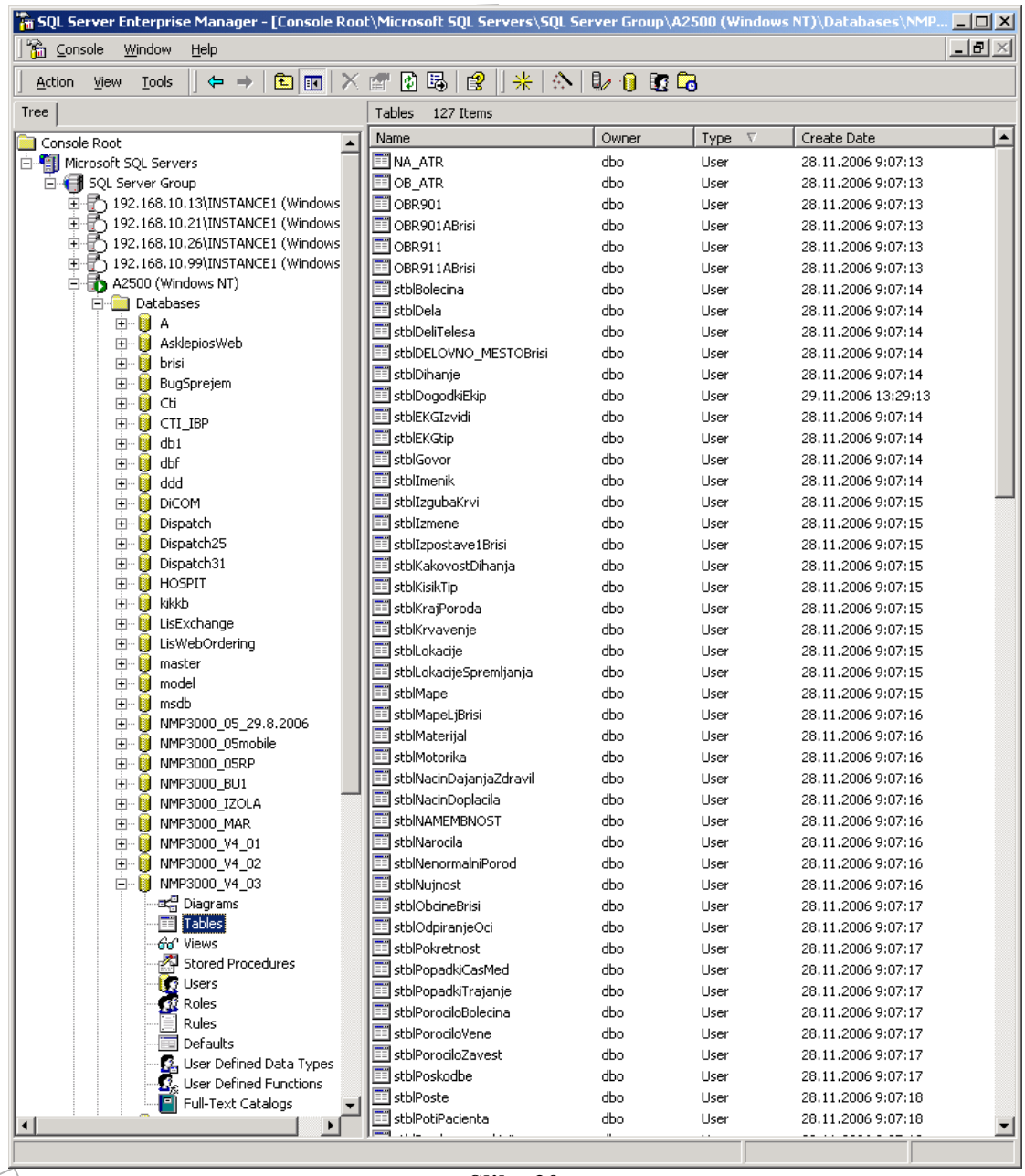
MS SQL je podatkovna zbirka, ki lahko deluje kot samostojen podatkovni strežnik ali pa v kombinaciji z drugimi tehnologijami.

Podatkovne zbirke so računalniško zapisane množice podatkov, ki so organizirane tako, da zagotavljajo sprotne, celovite in nasploh kakovostne informacije o sistemu podatkov. Podatkovna baza je sklop podatkov, medsebojnih sklicevanj na podatke in sistem za razvrščanje in urejanje podatkov v bazi. Podatkovne zbirke so sestavljene iz tabel, ki so med seboj povezane.

**SQL Server Enterprise Manager**

Za lažje kreiranje tabel smo uporabili orodje SQL Server Enterprise Manager, ki grafično prikaže strukturo podatkovne zbirke.

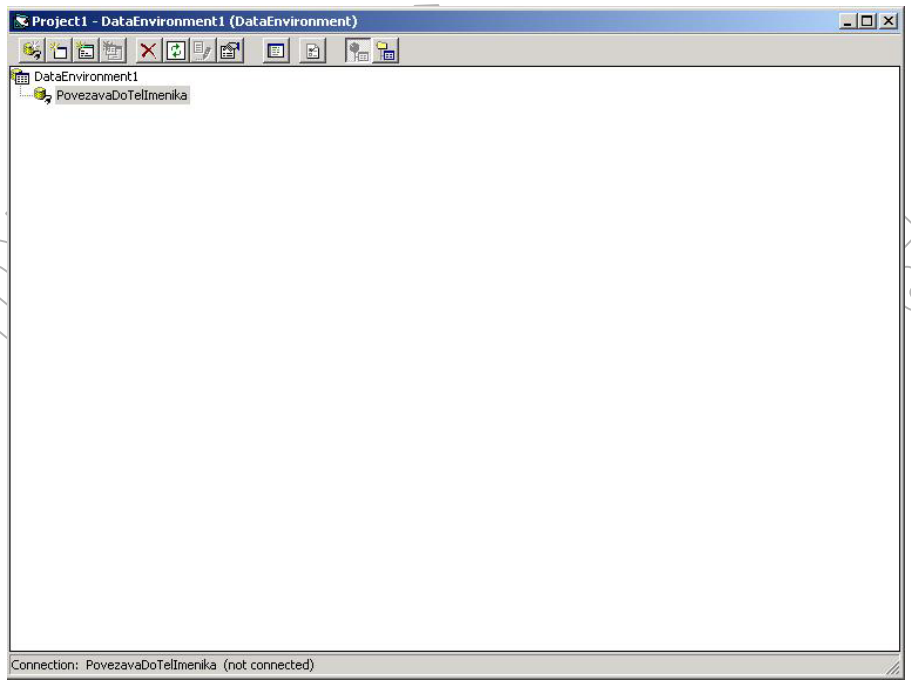
Strežnik MS SQL server lahko vsebuje več podatkovnih zbirk (katalogov). Katalog ustvarimo tako, da v Enterprise Managerju kliknemo z desnim miškinim gumbom in izberemo New Database. Odpre se pogovorno okno Database Properties, kjer vpišemo ime baze. Kliknemo gumb OK. S tem smo ustvarili novo podatkovno zbirko. Če dvokliknemo nanjo se prikaže pogovorno okno, kjer lahko pregledujemo tabele, uporabnike, pravila, … . Za delo z tabelami kliknemo na ikono Tables. Prikažejo se nam tabele, ki jih je skreiral sam MS SQL. Imenujemo jih sistemske tabele. V njih so zapisani podatki o podatkovni zbirki. Če želimo dodati svojo tabele, kliknemo z desnim miškinim gumbom in izberemo New table. Odpre se okno, kamor vpisujemo imena stolpcev, tip podatkov, ki bodo v ta stolpec zapisani in dolžino. Označimo lahko tudi, ali dovolimo, da v nek stolpec ne vpisujemo nobenih podatkov.




**Slika 20**
**SQL Server Enterprise Manager**

### Povezovanje podatkovne zbirke in VB

Za povezovanje podatkovne zbirke in Visual Basica v Visual Basicu obstaja poseben modul, imenovan Data Enironment. V njem lahko naredimo povezavo do baze. To storimo tako, da z desnim miškinim gumbom kliknemo na povezavo, ki se pojavi ob odprtju Data Environmenta.




**Slika 21**
**Data Environment**

Odpre se pogovorno okno z štirimi zavihki. Na zavihku Provider izberemo Microsoft OLE DB Provider for ODBC Drivers. Na zavihku Connection vpišemo ime serverja, ime kataloga ter uporabnika ter geslo. Slednja vpišemo le, če smo podatkovno strukturo zaščitili z geslom. Ko vpišemo podatke, lahko povezavo tudi preverimo. To storimo s klikom na gumb Test Connection. Da bi v program dobili podatke iz baze, moramo uporabiti tako imenovane poizvedbene stavke. Stavki se pišejo v jeziku SQL. Najosnovnejši ukazi za uporabo podatkovnih zbirk so:

- **create:** ustvari; s tem ukazom lahko kreiramo tabele
- **select:** izberi; izbiramo podatke iz podatkovnih struktur
- **insert:** vnesi; vstavljamo vrednosti v podatkovne zbirke
- **delete:** briši; lahko brišemo podatke iz podatkovnih zbirk

Oglejmo si osnovno strukturo select stavka. To je tudi najpogosteje uporabljen stavek jezika SQL:

```
SELECT seznam podatkov, ki jih želimo videti
FROM seznam tabel, kjer bomo podatke našli
WHERE pogoji, ki določajo, katere podatke želimo videti
```

Oglejmo si to še na primeru

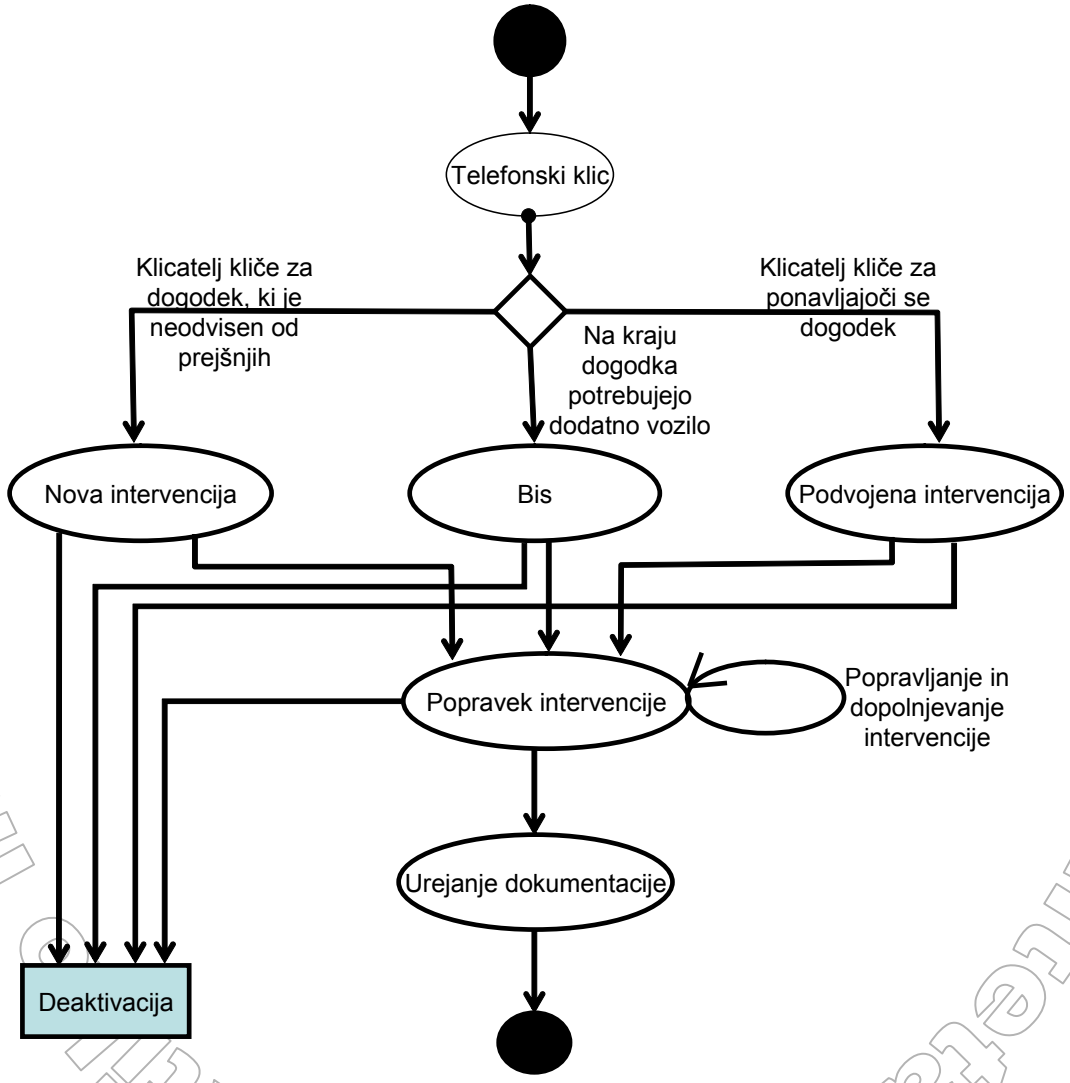
```
SELECT  Ime, Priimek, Tel_Stevilka
FROM tblPrijatelji
WHERE Ime LIKE 'Janez'
```

V tem primeru iz tabele `tblPrijatelji` izberemo vse prijatelje, ki imajo v stolpcu `Ime` vpisano besedo, podobno besedi Janez.

# 4 RAZVOJ APLIKACIJE

## 4.1 Načrtovanje sistema

Načrtovanje sistema je verjetno najbolj pomemben in odgovoren del razvoja aplikacije. Vsaka napaka v tem delu nam lahko zelo oteži samo implementacijo in podaljša razvoj aplikacije.
Pri načrtovanju sistema računalniško podprte nujne medicinske pomoči smo se najbolj posvetili načrtovanju baze podatkov in razvoju logike delovanja aplikacije. Pri tem smo tesno sodelovali z naročnikom, saj nam je le ta lahko najbolje razložil sam potek dogodkov od telefonskega klica do končanega vnosa dokumentacije. Pri načrtovanju smo prišli do spoznanja, da se nobenih podatkov ne sme brisati iz baze, ampak se jih lahko samo deaktivira. Uporabnik je podal tudi željo po načinu prikaza podatkov o intervenciji. Pomembno je bilo tudi načrtovanje sistemskega dovoljevanja pravic glede uporabe aplikacije.




**Slika 22**
**Načrt poteka spreminjanje podatkov**

**Intervencija - Dogodek**
Aplikacija za spremljanje razvoja intervencije mora omogočati natančen pregled nad spreminjanjem podatkov o vsaki intervenciji. Ni dovolj, da se hranijo samo tekoči podatki,

ampak morajo biti dostopna vsa stanja podatkov o intervencijah. Zato smo pri načrtovanju aplikacije vsako uporabnikovo željo po spreminjanju stanja zaklenili z gumboma uredi in shrani. Ko uporabnik želi podatke o intervenciji spremeniti, klikne gumb uredi. Šele potem lahko željene podatke spremeni in shrani. Ob shranjevanju se podatki poleg v evidenco trenutnega stanja intervencije zapisujejo tudi v zgodovino. S shranjevanjem postane to stanje intervencije osnovno za nadaljnje evidentiranje stanja.
Pregledovanje stanja intervencije je sestavljeno iz dveh delov:
Prvi del omogoča pregledovanje zadnjega stanja intervencije. Od tu je omogočeno nadaljnje evidentiranje podatkov o intervenciji, njeno podvajanje in izdelava tako imenovanih bisov.
Drugi del omogoča pregledovanje razvoja intervencije od prvega vnosa pa do zadnjega stanja, ter tiskanje posameznega stanja.

## 4.2 Opis in izvedba nekaterih funkcionalnosti aplikacije

### 4.2.1 Nastanek intervencije

Nastanek intervencije tehnološko ločimo na dva koraka. Prvi korak je priprava vseh vnosnih polj, drugi pa je beleženje podatkov v bazo.

**Priprava vnosnih polij**
Uporabnik v sprejemni formi v vsakem trenutku vidi podatke o določeni intervenciji. Ko klikne na gumb »nova«, se vsa polja izpraznijo. Pri kliku na gumb »bis« in »podvoji« določeni podatki iz trenutne sprejemne forme ostanejo. Zato mora uporabnik pri bisu in podvojeni intervenciji najprej poiskati (in v sprejemni formi izpisati) ustrezno intervencijo. Le tako bodo v sprejemni formi ostali pravilni podatki.

Preden si ogledamo, na kakšen način pripravimo vnosna polja, je potrebno pojasniti dve funkciji, ki jih bomo srečali v nadaljevanju:

- `DatumCasSql` je funkcija, ki nam vrne čas strežnika, na katerem je baza programa. S tem smo rešili problem nekonsistenčnosti časov na lokalnih računalnikih. Dogaja se, da sta časa na dveh računalnikih različna. Tako je prihajalo do tega, da je bil na primer zabeleženi čas oddaje intervencije manjši kot čas sprejema le te. Z uporabo časa strežnika pa je čas poenoten in težav ni več.
- Funkcija `DebugPrint` je funkcija, ki nam na posebno datoteko izpiše željen niz besed. Uporabljamo jo zato, da v datoteko zapišemo, kaj se dogaja. Na ta način imamo ob primeru napak vpogled na to, kaj se je v tistem trenutku v aplikaciji dogajalo.

Ker program izmenjuje podatke z bazo podatkov, je potrebno z njo vzpostaviti povezavo. To naredimo v tako imenovanem podatkovnem okolju (data environment), kjer vzpostavimo povezavo do podatkovne baze. Za to, da podatke pridobimo iz baze, moramo v aplikaciji napisati poizvedbene stavke (command). Poizvedbeni stavek nam v tabeli (recordset) vrne podatke, ki smo jih zahtevali. Potem lahko nad temi podatki izvajamo različne operacije. Lahko jih popravljamo, brišemo, dodajamo, ...
Tabela podatkov ima več stanj. Tako na primer, če je tabela odprta, na njej ne moremo izvesti poizvedbenega stavka, pač pa moramo prej to tabelo zapreti.

Podprogram `PripraviSprejem`, ki si ga bomo ogledali, kličemo iz različnih delov programa. Zato da vemo, iz katerega dela podprogram ga kličemo, uporabimo globalno spremenljivko `intNacinNastanka`. Ta nam pove, v katerem načinu sprejema intervencije se bo podprogram izvedel. Glede na vrednost spremenljivke `intNacinNastanka` se vnosna polja različno prednastavijo.

Poglejmo, kako pripravimo vnosna polja.

```
Private Sub PripraviSprejem()
      Dim datumCas As Date
      ZacetekPripraviSprejem = datumCasSql

      DebugPrint "subEditSprejem"
      'Vstavljanje vrednosti iz baze v kontrolnike (osveževanje) ----
      With deDispatch.rscmdSprejemRazmerjaKlicatelja
            ' če podatkovna tabela o kličočem ni zaprta, jo zapremo
            If Not (.State = adStateClosed) Then .Close
            ' pokličemo poizvedbeni stavek, ki nam vrne vsa razmerja
            ' kličočega
            deDispatch.cmdSprejemRazmerjaKlicatelja "%", True
      End With
      ' podatke iz podatkovne tabele prepišemo v gradnik
      With dcRazmerjeKlicatelja
            .RowMember = "cmdSprejemRazmerjaKlicatelja"
            Set .RowSource = deDispatch
      End With

       'Z vrednostmi iz baze napolnimo še vse ostale gradnike
       ' (šifrante), ki pa jih zaradi obsega ne bomo prikazovali


      'Konec vstavljanje vrednosti iz baze v kontrolnike -----------

      'Prekinemo povezavo med bazo in kontrolniki--------------------

      With Me.dcRazmerjeKlicatelja
            Set .DataSource = Nothing
      End With

       ' V tem delu, ki ga ne bomo prikazali,
       ' prekinemo povezavo še za vse ostale kontrolnike

      ' Konec Prekinemo povezavo med bazo in kontrolniki ------------

      dateTime = dateTimeSql
      '******
```

Znaki ***** na koncu prikazane kode pomenijo, da se prikazani podprogram nadaljuje in bo predstavljen v nadaljevanju. Če pa so znaki na začetku prikazane kode, se le ta nadaljuje od zadnjih znakov *****.
Vsi gradniki v Visual Basicu imajo prednastavljeno lastnost, do katere lahko dostopamo samo z imenovanjem gradnika. Tako ima gradnik oznaka (label) prednastavljeno lastnost Caption, textovni okvir pa lastnost Text.

Tako je v zgornji kodi stavek

```
      dateTime = dateTimeSql
```

enakovreden stavku

```
      dateTime.Text = dateTimeSql
```

v kodi, ki sledi, pa je npr.:

```
            lblYOd = 0
```

enakovredno

```
            lblYOd.Caption = 0

      '******
      Select Case intNaciniNastanka 'kako smo poklicali ta podprogram
      Case frmMode.SprejemNova 'gre za novo intervencijo
            'nastavimo vse vrednosti na začetno vrednost

            lblXOd = 0
            lblYOd = 0
            lblNastanek = dateTime
            lblPopravek = ""
            lblIDPrevoza = ""
            lblNastanekOs = dateTime
            lblIDPrevozaOs = ""
            dcRazmerjeKlicatelja.Text = ""
            ┌──────────────────────────────────────────────┐
            │ 'tega dela kode zaradi obsega ne bomo prikazali │
            └──────────────────────────────────────────────┘
      Case frmMode.SprejemPodvoji 'podvojena intervencija
            'nastavimo potrebne vrednosti na začetno vrednost
            'ostale pustimo nespremenjene

            'lblXOd in lblYOd koordinati kraja intervencije
            'ostaneta nespremenjena
            lblNastanek = dateTime
            lblPopravek = ""
            lblIDPrevoza = ""
            lblNastanekOs = dateTime
            lblIDPrevozaOs = ""
            'dcRazmerjeKlicatelja.Text ostane enak
            ┌──────────────────────────────────────────────┐
            │      čen del kode z ostalimi kontrolniki     │
            └──────────────────────────────────────────────┘
      Case frmMode.SprejemBis 'ta intervencija je Bis
            'lblXOd in lblYOd koordinati kraja intervencije
            'ostaneta nespremenjena
            lblNastanek = dateTime 'vnesemo sistemski datum in čas
            lblPopravek = ""
            lblIDPrevoza = ""
            'lblNastanekOs ostane enak
            'lblIDPrevozaOs ostane enak
            'dcRazmerjeKlicatelja.Text ostane enak
            ┌──────────────────────────────────────────────┐
            │                                              │
            └──────────────────────────────────────────────┘
      Case frmMode.SprejemPopravi
            'V primeru, če uporabnik želi popraviti intervencijo,
            'ostanejo vsi podatki nespremenjeni
      End Select
End Sub
```

Ko se bo podprogram izvedel, se bodo ustrezno izpraznila vnosna polja. Osvežili se bodo tudi vsi šifranti. To se zgodi ob povezovanju gradnikov in kontrolnikov. S tem preprečimo, da bi imel uporabnik v šifrantih stare podatke. Vnosna forma je s tem pripravljena za vnos podatkov.
Ko gradnike napolnemo z vrednostmi iz baze, vedno prekinemo povezavo z bazo, da pri vnosu v bazo ne pride do zavrnitve.

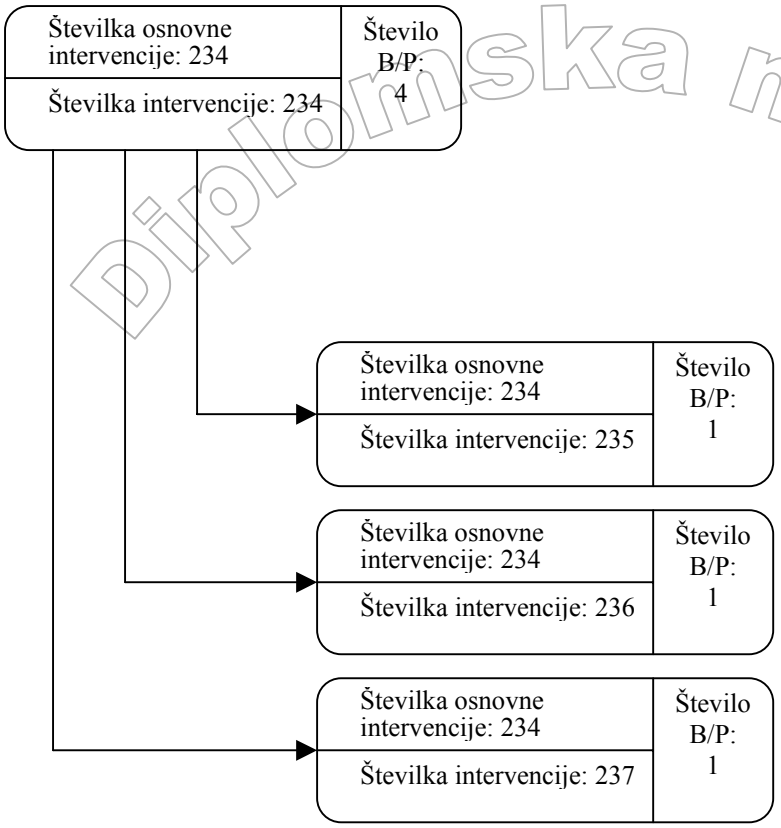
**Beleženje podatkov v bazo**

Potem, ko se je izvedla metoda `PripraviSprejem`, je vnosni obrazec pripravljen za beleženje podatkov. Podprogram `btnShraniIntervencijo_Click()`, ki ob kliku na gumb `btnShraniIntervencijo` zabeleži podatke v bazo, lahko razdelimo na dva osnovna procesa. Prvi proces preveri pravilnosti vnesenih podatkov. Drugi proces pa je samo zapisovanje podatkov v bazo.

Pravilnosti vnesenih podatkov moramo preveriti, da strežnik Sql ne zavrne vpisa podatkov v bazo. Preverjamo predvsem tipe vnesenih podatkov in to, ali so obvezni podatki vnešeni.

```
Private Sub btnShraniIntervencijo_Click()
      Dim i As Integer
      Dim datumCas As Date
      Dim lngID As Long
      Dim dodamoNovoInt As Boolean

      dodamoNovoInt = False
      datumCas = dateTimeSql
      DebugPrint "btnShraniIntervencijo_Click() " & Me.Caption

      'Če naredimo novo intervencijo
      Select Case intNaciniNastanka
      Case frmMode.SprejemNova, frmMode.SprejemPodvoji, _
      frmMode.SprejemBis
            dodamoNovoInt = True
      End Select
      '******
```

Pri vnašanju števila B/P je potrebno paziti, saj velja, da ima vsak pacient svojo intervencijo. Če ima nova intervencija več ali neznano število B/P, se naredijo bisi. Ta ima lahko samo enega pacienta. Bis ima svojo številko intervencije, vodi pa se še osnovna številka intervencije, ki je številka intervencija iz katere je bil bis narejen. S tem smo omogočili sledenje intervencijam istega dogodka.




**Slika 23**
**Nastanek bisov**

Problem vpisovanja števila pacientov smo rešili tako, da smo najprej preverili ali je vpisano število. Ali je vneseni podatek število, v Visual Basicu preverimo s funkcijo `IsNumeric(besedilo)`. Funkcija vrne vrednost `True`, če je besedilo `število` in vrednost `False`, če to ni.
Če število B/P ni numerično, potem je lahko nastala samo nova intervencija. To preverimo tako, da primerjamo številko osnovne intervencije s številko intervencije. Če se ujemata, mora biti v polju število B/P vnešen znak ?. V nasprotnem primeru uporabniku javimo napako pri vnosu. Če se številka osnovne intervencije ne ujema z številko intervencije, prav tako javimo napako pri vnosu.
Če je število B/P numerično, je lahko nastal bis, nova intervencija ali pa podvojana intervencija. Število pacientov je omejeno le v primeru bisa. Zato preverimo, če sta številka osnovne intervencije in številka intervencije različni. Če sta, število pacientov ne sme biti večje kot 1. V nasprotnem primeru uporabniku javimo napako.
Poglejmo si, kako je to implementirano v programu:

```
'******
'Preverba vnosa-----------------------------------------------
On Error Resume Next
'Vneseno število pacientov je lahko neznano
If Not IsNumeric(txtStevBP) Then
        ' Preverimo ali je številka osnovne intervencije enaka
        ' številki intervencije. To se zgodi le pri novi
        ' intervenciji
        If lblIDPrevozaOs = lblIDPrevoza Then
        ' če je številka intervencije enaka osnovni številki
        ' intervencije potem je lahko vnesen le znak ?, ki pomeni
        ' neznano število pacientov
             If Not txtStevBP = "?" Then
```

```
                  MsgBox "Doloci število B/P ali ? za neznano
                  število B/P!", vbExclamation
                  DebugPrint "Doloci število B/P ali ? za
                  neznano število B/P!"
                  txtStevBP.SetFocus
                  Exit Sub
            End If
      ' če številka osnovnega prevoza ni enaka
      ' številki prevoza in to ni število, uporabnika opozorimo
      ' s sporočilnim oknom
      Else
            MsgBox »Število B/P je 1 ali 0!", vbExclamation
            DebugPrint "Število B/P je 1 ali 0!"
            txtStevBP.SetFocus
            Exit Sub
      End If
' v primeru, da je vneseno število
Else
      ' če številka osnovnega prevoza ni enaka
      ' številki prevoza, kar se zgodi le v primeru bisa in je
      ' število pacientov več kot 1, uporabnika opozorimo na
      ' napako pri vnosu
      If lblIDPrevozaOs <> lblIDPrevoza Then
            If txtStevBP > 1 Then
                  MsgBox "Število B/P je 1 ali 0!", _
                  vbExclamation
                  DebugPrint "Število B/P je 1 ali 0!"
                  txtStevBP.SetFocus
                  Exit Sub
            End If
      End If
End If
On Error GoTo e
'******
```

Pri vpisovanju nekaterih podatkov o intervenciji ima uporabnik že vnaprej določene vrednosti, med katerimi lahko izbira. Tako na primer lahko uporabnik, ko določa prioriteto, izbere le eno izmed vrednosti 90, 80, 70, 60, 50, 40, 30, 20 ali 10. Če ne izbere nobeno od teh vrednosti, se barva gradnika (`ImeGradnika.BackColor`) nastavi na roza (`vbRoza`). Ko podatke shranjujemo, preverimo barvo tega gradnika. Če je roza, uporabnika opozorimo s sporočilnim oknom.

```
'******
If dcPrioritete.BackColor = vbRoza Then
      MsgBox "Doloci prioriteto!", vbExclamation
      DebugPrint "Doloci prioriteto!"
      dcPrioritete.SetFocus
      Exit Sub
End If
' Tu preverimo še ostale potrebne podatke, vendar zaradi
' dolžine kode tega dela ne bomo prikazali

'End Kontrola vnosa-----------------------------------------
'******
```

Ko smo pregledali pravilnost vnesenih podatkov, je potrebno osvežiti tabelo zapisov iz baze. To je potrebno narediti, ker ta podprogram uporabljamo tudi za popravljanje intervencij. Med tem, ko smo dopolnjevali podatke določene intervencije, je lahko nek drug uporabnik že popravil in v bazo vpisal podatke o isti intervenciji. Zato imamo mi v tabeli iz baze napačne

podatke. Torej, če imamo v trenutku, ko želimo v tabelo podatkov iz baze zapisati nek podatek, ki je drugačen kot je v bazi, nam strežnik SQL vnos zavrne. S tem strežnik SQL preprečuje, da bi uporabnik vnesel nove podatke ne, da bi videl najnovejše. Oglejmo si primer za lažje razumevanje:
Brata Miha in Tine uporabljata aplikacijo, ki v bazo podatkov zapisuje ime, priimek in telefonsko številko njunih prijateljev. Aplikacijo imata nameščeno vsak na svojem računalniku, bazo pa imata skupno, tako da lahko Miha vidi podatke, ki jih je vnesel Tine in obratno. Poleg vseh imata tudi prijatelja Janeza Novaka z telefonsko številko 031111222. Janez je zamenjal telefonsko številko zato jo morata popraviti. Miha ne ve, da jo želi zamenjati tudi Tine. Ko Miha odpre aplikacijo in želi popraviti Janezovo telefonsko številko so podatki za Janeza v bazi sledeči:

| Ime | Priimek | Telefonska številka |
|---|---|---|
| Janez | Novak | 031111222 |

Ko klikne gumb popravi, se podatki prepišejo v tabelo podatkov iz baze:

| Ime | Priimek | Telefonska številka |
|---|---|---|
| Janez | Novak | 031111222 |

Ker ga pokliče mama, podatkov ne spremeni in shrani. Med tem pa Tine na svojem računalniki popravi Janezovo telefonsko število, ki je 031333444. Sedaj so v bazi podatkov za Janeza sledeči podatki:

| Ime | Priimek | Telefonska številka |
|---|---|---|
| Janez | Novak | 031333444 |

Ko pride Miha nazaj, popravi Janezovo telefonsko število in želi shraniti podatke. Ker pa so v njegovi tabeli podatkov iz baze podatki za uporabnika Janez drugačni, kot v bazi podatkov mu strežnik SQL vnos zavrne. To lahko preprečimo tako, da pred beleženjem podatkov v bazo osvežimo podatke v tabeli podatkov iz baze.

```
'******
'Osvežitev, da ne pride do zavrnitve vnosa zaradi neažurnosti
'zapisa v bazi---------
If lblIDPrevoza = "" Then
      lngID = 0
Else
      lngID = lblIDPrevoza
End If
OsveziIntervencijo (lngID)

'End osvežitev, da ne pride do zavrnitve vnosa zaradi
neažurnosti zapisa v bazi-----------------------------------
'******
```

`OsveziIntervencijo(stevilkaIntervencije)` je podprogram, ki nam vrne iz baze sveže podatke o intervenciji s številko `stevilkaIntervencije`. Če dodajamo novo intervencijo, iz baze dobimo prazno tabelo podatkov.

**Transakcije**
Sedaj je vse pripravljeno za zapisovanje podatkov v bazo. Ker se lahko zgodi, da gre med zapisovanjem v bazo kaj narobe (prekine se povezava, …) ali se uporabnik premisli in želi obdržati stare podatke, to delamo s transakcijo. Najpomembnejša lastnost transakcije je

možnost preklica cele vrste operacij nad podatkovnimi strukturami, ki smo jih izvedli od njenega začetka pa do danega trenutka. Ta lastnost je najpomembnejša pri popravljanju ali spreminjanju zapisov. Transakcija je praviloma določena z štirimi lastnostmi:

- **atomarnost:** če se bo transakcija prekinila med delovanjem, pričakovano ali nepričakovano, to pomeni, da se bodo razveljavile tudi vse operacije, ki so sestavljale transakcijo.
- **konsistentnost:** transakcija mora vedno dati isti rezultat neglede ali jo izvedemo ta trenutek ali pa čez dve uri. To seveda velja, če se vhodni parametri za transakcijo ne spremenijo
- **izolacija:** če je transakcija sestavljena iz velikega števila operacij, morajo biti notranja stanja (spremenjeni podatki) ostalim sistemom nevidni. Šele, ko je transakcija potrjena, so lahko rezultati vidni ostalim sistemom.
- **trajnost:** pomeni, da so rezultati trajni in se nikoli ne izgubijo, razen če jih ne odstrani uporabnik. Poleg tega je rezultat transakcije vedno vrnljiv. To pomeni da obstaja postopek, ki lahko vrne podatke v stanje pred transakcijo.

Transakcijo pričnemo s stavkom `BeginTrans`. Ko jo zaključimo, jo moramo bodisi potrditi s stavkom `CommitTrans`, bodisi preklicati s stavkom `RollBack`. Če se program pred zaključkom transakcije konča (bodisi zaradi napake, bodisi da mi sami ali pa logika programa prekine izvajanje programa), se samodejno izvede preklic.

Transakcije lahko tudi gnezdimo. Zaključiti moramo vsako posebej. Zaključek transakcije se vedno nanaša na zadnjo aktivno.

```
'******
'Začetek Transakcija-------------------------------
DebugPrint "BeginTrans"
deDispatch.cnDispatch.BeginTrans
On Error GoTo rbt

Select Case intMode
Case frmMode.SprejemNova, frmMode.SprejemPodvoji, _
frmMode.SprejemBis,
      'dodamo nov zapis v
      DebugPrint "Nov Zapis"
      With deDispatch.rscmdSprejem
            .AddNew
            !OD_X = lblXOd
            DebugPrint " deDispatch.rscmdSprejem!OD_X=" & !OD_X
            !OD_Y = lblYOd
            DebugPrint "deDispatch.rscmdSprejem!OD_Y=" & !OD_Y
            !Popravek_naloga = dateTime
            DebugPrint _
            "deDispatch.rscmdSprejem!Popravek_naloga=" & _
            !Popravek_naloga

            If lblNastanekOs = "" Then
                  !Nastanek_osnovnega_naloga= dateTime
            Else
                  !Nastanek_osnovnega_naloga= lblNastanekOs
            End If
            DebugPrint "!Nastanek_osnovnega_naloga= " & _
            deDispatch.rscmdSprejem!Nastanek_osnovnega_naloga
            If lblIDPrevozaOs = "" Then
                  !ID_Osnovnega_prevoza = Null
            Else
                  !ID_Osnovnega_prevoza = lblIDPrevozaOs
            End If
```

```
                DebugPrint _
                "deDispatch.rscmdSprejem!ID_Osnovnega_prevoza = " _
                & !ID_Osnovnega_prevoza

                'Tu nastavimo še ostale vrednosti
            end With
        Case frmMode.SprejemPopravi
            ' Tu popravljamo zapis o intervenciji, ki pa ga
            ' zaradi podobnosti z novo intervencijo in obsega
            ' kode ne bomo pokazali
        End Select
        deDispatch.rscmdSprejem.Update

        If dodamoNovoInt Then
            'z funkcijo NatisniNovoInt besedilo izpišemo osnovne
            ' podatke o intervenciji na vrstični printer
            NatisniNovoInt ">Sprejem I:" & .Fields("ID_Prevoza") & _
            "/" & .Fields("ID_Prioriteta") & "/" & _
            .Fields("Nastanek_naloga")
            ' v funkciji NatisniNovoInt je samo nakazano na kakšen
            ' način se podatki izpisujejo saj je podatkov, ki se
            ' izpišejo mnogo več.
        End If
        PosodobiPopravkiIntervencij deDispatch.rscmdSprejem
        DebugPrint "CommitTrans"
        deDispatch.cnDispatch.CommitTrans
        'Potrditev Transakcije---------------------------------------

        ' Pokazemo obrazec za masovno nesreco
        If txtStevBP.Text = "?" or CLng(txtStevBP.Text) > 4 Then
            btnMasovna.Visible = True
            frmMasovne.mShow _
            deDispatch.rscmdSprejem!ID_Osnovnega_prevoza, _
            txtStevBP.Text,0
        End If
        'END masovna nesreča
        Exit Sub
        '******
```

`PosodobiPopravkiIntervencij tabelaIzBaze` je podprogram, ki beleži vsako spremembo podatkov v bazi. Deluje tako, da vsako polje v izvorni tabeli prepiše v tabelo popravkov.

**Lovljenje napak**

V zgornji kodi (in v kodi prej) smo lahko opazili stavke oblike `On Error` in sicer `On Error Resume Next`, `On Error GoTo 0` in `On Error GoTo Oznaka`. Gre za tako imenovane pasti. Pasti nastavljamo za primer, da pride do napake v podprogramu. Taka napaka bi drugače povzročila, da bi aplikacija prenehala z delovanjem (program bi se "sesul"). Pasti nastavimo v vsakem podprogramu posebej. Past je aktivna od stavka `On Error GoTo Oznaka` pa do konca podprograma. Oznaka mora biti neko ime. Past lahko izključimo tudi sredi podprograma s posebno obliko omenjenega stavka in sicer `On Error GoTo 0`. Z enako oznako in dvopičjem označimo vrstico tik pred začetkom prestreznika napak. Prestreznik napak običajno pišemo na koncu podprograma. Začne se po stavku `Exit Sub` ali `Exit Function`. To storimo zato, da se, v primeru, da napake ni, podprogram konča pred prestreznikom. Past `On Error Resume Next` navedemo v primeru, ko želimo, da se program klub napaki nadaljuje v naslednji vrstici.
Pasti ni mogoče gnezditi, lahko pa jih v enem podprogramu napišemo več. Ko navedemo naslednjo past, se prejšnja izklopi.

Prestreznika `e:` in `rbt:` se v naši kodi izvedeta v različnih primerih napake. Na prestreznik `e:` pridemo, ko se zgodi napaka pred zapisovanjem v bazo. Na prestreznik `rbt:` pridemo v primeru, ko se je pojavila napaka pri zapisovanju v bazo. V tem primeru prekličemo vse zapise v bazo, ki smo jih do sedaj uspešno naredili z to transakcijo.

```
'******
e:
      ' Če pride do napake, ko ni transakcije
      DebugPrint "e: Err.Source=" & Err.Source & " Err.Description="_
      & Err.Description
      Exit Sub
rbt:
      'Zavrnitev Transakcija-------------------------
      On Error GoTo 0
      DebugPrint "rbt: "
      ' če je podatkovna tabela sprejetih intervencij odprta,
      ' razveljavimo vnesene podatke
      With deDispatch.rscmdSprejem
            If (.State = adStateOpen) Then .CancelUpdate
      End With
      ' če je podatkovna tabela popravkov odprta,
      ' razveljavimo vnesene podatke
      With deDispatch.rscmdPopravkiIntervencij
            If (.State = adStateOpen) Then .CancelUpdate
      End With
      deDispatch.cnDispatch.RollbackTrans
      DebugPrint "RollbackTrans"
End Sub
```

### 4.2.2 Pregled intervencij na čakanju, v izvajanju ter razpoložljivosti ekip

Stanje intervencij, ki jih je potrebno še opraviti, stanje razpoložljivosti vozil in stanje intervencij v izvajanju so za oddajnega dispečerja zelo pomembne informacije. Oddajni dispečer se na podlagi teh stanj odloči, kdaj bo kateremu od razpoložljivih vozil oddal intervencijo, ki ni nujna. Nujne intervencije se vedno oddajajo takoj. Zato mora imeti oddajni dispečer v vsakem trenutku na razpolago nekaj vozil za nujne primere.

Prikaz teh stanj je na oddajni formi razporejen v treh sklopih:

- intervencije na čakanju
- ekipe
- intervencije v izvajanju

Vsak sklop poleg tabele,v kateri so v posamezni vrstici navedeni podatki o določeni intervenciji oziroma ekipi, vsebuje tudi nekaj funkcijskih gumbov, ki služijo preglednejšemu in lažjemu delu oddajnega dispečerja.

**Intervencije na čakanju**

Za to, da neko intervencijo poimenujemo »intervencija na čakanju«, obstajata dva kriterija. Kriterija se ločita po tem, ali je intervencija prednaročilo ali ne. Če ni prednaročilo, potem je pogoj sledeč:

- intervencija še ni bila predana ekipi
- intervencija ni deaktivirana

Če intervencija je prednaročilo, potem je na čakanju, kadar velja:

- intervencija še ni bila predana ekipi
- intervencija ni deaktivirana
- je ura že toliko, da je potrebno prednaročilo pokazati (čas, ko je prednaročilo potrebno pokazati v oddajni formi, nastavi dispečer ob sprejemu intervencije)

**Ekipe**

Ekipe, ki jih dispečer vidi v oddajni formi, je ustvaril vodja izmene. Ekipe so v tabeli razdeljene v štiri navidezne skupine. Skupine so ločene po statusu ekipe. Statusi ekipe so:

- razpoložljiva
- nerazpoložljiva
- pogojno razpoložljiva
- intervencija (v dispečerstvu NMP-ja se uporablja tudi izraz »Zasedena« )

V tabeli so ekipe glede na status ločene z barvo:

- ekipe, ki so razpoložljive, so obarvane zeleno. Pogoj, da je ekipa razpoložljiva, je sestavljen iz naslednjih pogojev:
    - Ekipa ni deaktivirana.
    - Čas začetka dela ekipe je manjši od trenutnega.
    - Čas, ko ekipa konča z delom, je večji od trenutnega.
    - Imajo status *razpoložljiv*.

- pogojno razpoložljive ekipe so obarvane rumeno. Ekipe so pogojno razpoložljive kadar:
    - Ekipa ni deaktivirana.
    - Čas začetka dela ekipe je manjši od trenutnega.
    - Čas, ko člani ekipe končajo delo, je večji od trenutnega.
    - Njihov status je *pogojno razpoložljiv.*

- ekipe, ki so zasedene, so obarvane rdeče. Ekipa je zasedena, če:
    - Ekipa ni deaktivirana.
    - Čas začetka dela ekipe je manjši od trenutnega.
    - Čas, ko člani ekipe končajo delo, je večji od trenutnega.
    - Imajo status *intervencija*.
- ne razpoložljive ekipe so obarvane sivo. Pogoj, da je ekipe ne razpoložljiva, je sledeč:
    - Ekipa ni deaktivirana.
    - Čas začetka dela ekipe je manjši od trenutnega.
    - Čas ko končajo delo je večji od trenutnega.
    - Ima status *nerazpoložljiv.*

Stanje ekip, intervencij na čakanju ter intervencij v izvajanju avtomatsko osvežujemo na določen čas, lahko pa si jih uporabnik osveži tudi ročno.

Poglejmo si, kako prikažemo ekipe. Za prikaz ekip je zadolžen dogodek klik na gumb btnOsveziEkipe. Ta dogodek se zgodi, ko uporabnik klikne na gumb Osveži ali pa dogodek sprožimo avtomatsko s pomočjo gradnika Timer.

```
Private Sub btnOsveziEkipe_Click()
    Dim lngStatus As Long
    Dim lngNamembnost As Long
    Dim i As Long

    DebugPrint "btnOsveziEkipe_Click()"
    cmdOddajaEkipe dateTimeSql 'zabeležimo ustrezni čas
    '******
```

Če je tabela ekip iz baze prazna (to je tabela rscmdOddajaEkipe), potem gradnik MSHFlexGrid, ki prikazuje ekipe, napolnimo s samimi naslovnimi vrsticami. S tem uporabniku povemo, da ni narejene nobene ekipe.

```
    '******
    With rscmdOddajaEkipe
```

```
        If .EOF And .BOF Then ' če v tabeli ekip ni aktivne ekipe
              ' potem nastavimo same prazne vrstice
              hfgEkipeCakanje.Clear
              hfgEkipeCakanje.FixedRow = 55
              hfgEkipeCakanje.ColWidth(0) = 10000
              hfgEkipeCakanje.Refresh
              Exit Sub
        End If
  End With
  '*****
```

Ker je uporabnik želel, da se informacija o razpoložljivosti ekip podaja tudi z barvnim kodiranjem, po povezovanju tabele iz baze in gradnika MSHFlexGrid pregledamo podatke in ustrezno pobarvamo celice

```
'*****
With hfgEkipeCakanje 'obdelujemo gradnik MSHFlexGrid z ekipami
      'iz baze vzamemo podatke
      .DataMember = rscmdOddajaEkipe.DataMember
      Set .DataSource = rscmdOddajaEkipe.DataSource

      'prekinemo povezavo med bazo podatkov in kontrolo
      Set .DataSource = Nothing
      .DataMember = ""

      For i = 1 To hfgEkipeCakanje.Rows - 1 'obdelamo vse ekipe
            .RowData(i) = 0 ' v .RowData lahko za vsako vrstico
            ' zapišemo poljubno vrednosti, za katero nimamo
            ' drugih ustreznih lastnosti.
            ' S to lastnostjo pri oddaji intervencije
            ' ekipi pregledamo, katero ekipo je uporabnik
            ' označil.

            .Row = i
            .col = 18 ' postavimo se v 18. stolpec
            ' V 18. stolpcu je koda statusa ekipe
            ' zapisana je v lastnosti .Text, z metodo Clng
            ' pa pretvorimo podatkovni tip String v Long,
            ' saj se vse vsi podatki, ki jih vstavimo v
            ' mshFlexGrid pretvorijo v String.
            lngStatus = CLng(.Text)

            .col = 17 ' postavimo se v 17 stolpec
            ' tu je zapisana namembnost vozila
            lngNamembnost = CLng(.Text)

            'barvamo prvi stolpec
            .col = 1
            'glede na status ekipe, le to ustrezno pobarvamo
            Select Case lngStatus
            Case 1
                  .CellBackColor = vbRoza
            Case 2
                  ' vbButtonFace je siva barva, ki je že
                  ' vgrajena v razvojno okolje VB
                  ' poleg barv, ki so že vgrajene ja jih lahko
                  ' ustvarjamo tudi sami
                  .CellBackColor = vbButtonFace
            Case 3
                  .CellBackColor = vbSvetloRumena
            Case 4
```

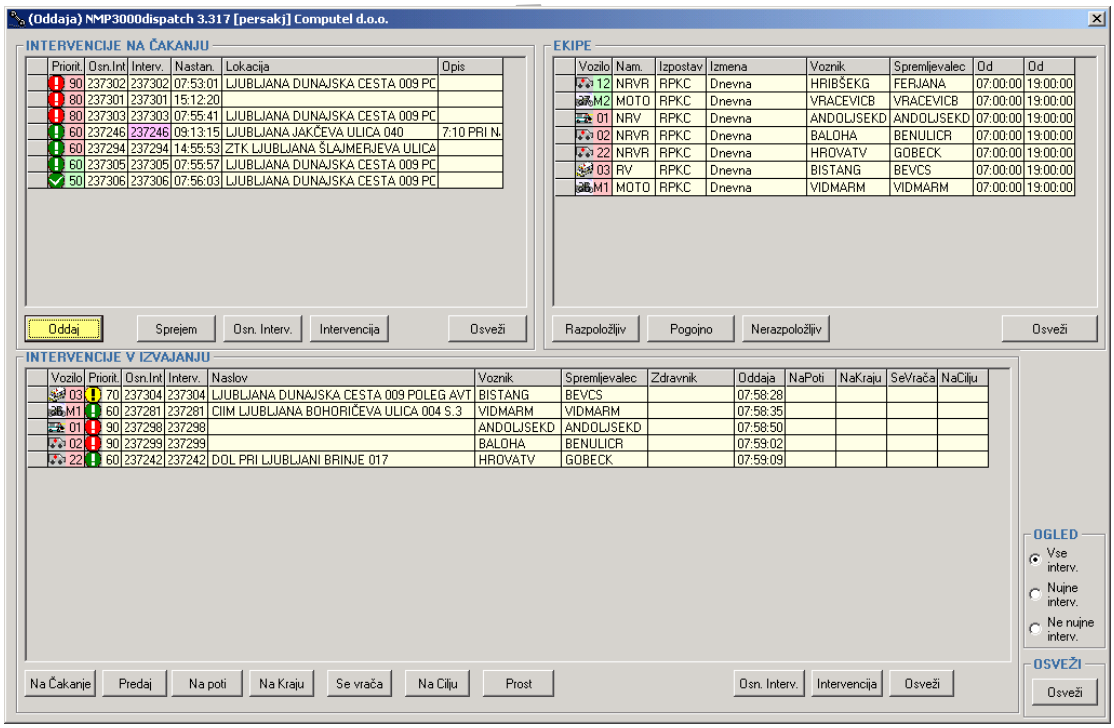
```
                .CellBackColor = vbSvetloZelena
            End Select

            'glede na namembnost prikažemo ustrezno sliko
            Select Case lngNamembnost
            Case 1
                Set .CellPicture = pNRVR
            Case 2
                Set .CellPicture = pNRV
            Case 3
                Set .CellPicture = pRV
            Case 4
                Set .CellPicture = pVUZ
            Case 5
                Set .CellPicture = pVI
            Case 6
                Set .CellPicture = pS
            Case 7
                Set .CellPicture = pMOTO
            End Select
        Next i
        .Refresh
    End With
    Exit Sub
    '****
```

pNRVR, pNRV,... pMOTO so slikovni okvirji, v katerih so ikone določenih vozil, ki jih postavimo v ustrezno celico.
Če je ob izvajanju prišlo do napake, jo ujamemo v past in ustrezno obdelamo.

```
    '****
e:
    MsgBox "btnOsveziEkipe_Click/e" & Chr(13) & "Napaka: " & _
    Err.Number & Chr(13) & Err.Description, vbCritical
    hfgEkipeCakanje.Refresh
End Sub
```

**Intervencije v izvajanju**
V tabeli intervencije v izvajanju prikazujemo podatke tako o intervenciji, kot tudi ekipi, ki to intervencijo izvaja. Pogoja, da intervencijo imenujemo intervencija v izvajanju, sta:

- Intervencija ima že dodeljeno ekipo.
- Intervencija še nima zabeleženega časa *prost*. Ko se nastavi čas prost, se intervencija na oddajni formi ne prikazuje več. Čas se zabeleži v bazo podatkov, na oddaji pa se ga ne vidi.




**Slika 24**
**Oddajna forma**

### 4.2.3 Oddaja intervencije na čakanju ekipi

Po tem, ko se dispečer odloči, katero intervencijo na čakanju bo predal določeni ekipi, mora to zabeležiti v programu. V panelu Intervencije na čakanju označi intervencijo, v panelu Ekipe pa ekipo, ki bo to intervencijo dobila. Nato klikne gumb Oddaj (rumeno obarvani gumb na sliki 22).
S tem, ko dispečer označi vrstico v intervencijah na čakanju, označeni vrstici nastavi vrednost, da je ta vrstica izbrana. Enako velja tudi za ekipe.

Poglejmo, na kakšen način je to implementirano s kodo. Spet je ustrezna metoda klik, tokrat na gumb btnOddano.

```
Private Sub btnOddano_Click()
      Dim datumCas As Date
      Dim i As Long
      Dim j As Long
      Dim k As Long
      Dim l As Long

'*****
```

Ko kliknemo na ustrezni gumb, moramo preveriti, če je tako v panelu Intervencije na čakanju, kot tudi v panelu Ekipe izbrana samo ena vrstica. Ko uporabnik klikne v prvi stolpec ene od intervencije na čakanju in v prvi stolpec ekipe, ki ji bo dodelil intervencijo, se s tem v obeh pokaže puščica. Ta označuje, da sta celici izbrani. Zato bomo le preverili, če sta obe ustrezni. To bomo storili tako, da bomo pogledali sliki v obeh celicah. Če sta določeni obe (imata kodo različno od 0), je vse v redu in lahko pričnemo s transakcijo (oddajo intervencije ekipi). Če pa katerakoli od obeh slik ni določena, intervencije ekipi ne moremo dodeliti.

```
'*****
      DebugPrint "btnOddano_Click() " & Me.Caption
```

```
If hfgECakanje.CellPicture <> 0 And _
hfgIntervencijeCakanje.CellPicture <> 0 Then
      'Begin Transakcija-----------------------------------
      DebugPrint "BeginTrans"
      deDispatch.cnDispatch.BeginTrans
      On Error GoTo e
      datumCas = dateTimeSql() ' zabeležimo čas oddaje
      hfgIntervencijeCakanje.col = 6 'Postavimo se v 6 stolpec.
                                     'V njem je zapisana
                                     'številka prevoza
      hfgECakanje.col = 3 ' Postavimo se v 3 stolpec. V njem je
                          ' zapisana številka ekipe.

      ' Poiščemo ekipo, ki bo dobila intervencijo
      For i  = 1 to hfgEkipeCakanje.Rows - 1
            if hfgEkipeCakanje.RowData(i) <> 0 Then
                  Exit For
            End If
      Next i
      ' poiščemo intervencijo, ki jo bomo opravili
      For j  = 1 to hfgIntervencijeCakanje.Rows - 1
            if hfgIntervencijeCakanje.RowData(j) <> 0 Then
                  Exit For
            End If
      Next j

      '*****
```

Ko imamo označeno tako ekipo kot tudi intervencijo, moramo v bazi podatkov poiskati ustrezno intervencijo, ki ji dopišemo nove podatke.

```
      '*****
      'Intervencije Cakanje
      With deDispatch.rscmdOddajaCakanje
            If Not (.State = adStateOpen) Then .Open
            ' če je tabela podatkov iz baze prazna potem javimo
            ' napako in opustimo oddajo intervencije ekipi
            If .BOF And .EOF Then
                  MsgBox "btnOddano_Click" & Chr(13) &
                  "rscmdOddajaCakanje EMPTY", vbExclamation
                  deDispatch.cnDispatch.RollbackTrans
                  'Rollback Transakcija--------------------
                  GoTo rbt
            End If
            ' postavimo kazalec na prvo vrstico v tabeli
            ' podatkov iz baze
            .MoveFirst
            ' poiščemo vrstico z ustreno številko intervencije
            .Find ("ID_Prevoza = " & _
            hfgIntervencijeCakanje.TextMatrix(i, 3))
            ' če ne najdemo intervencije javimo napako in
            ' opustimo oddajo intervencije ekipi
            If .EOF Then
                  MsgBox "btnOddano_Click" & Chr(13) & _
                  "Intervencije = " _
                  & hfgIntervencijeCakanje.TextMatrix(i, 6) & _
                  " NE NAJDEM", vbExclamation
                  'Rollback Transakcija------------------------
                  GoTo rbt
            End If
            ' V bazo zapišemo ustrezne podatke. !(ime stolpca
```

```
            ' v bazi) je okrajšava za metodo .Fields(ime
            ' stolpca v bazi), ki stolpcu v izbrani vrstici
            ' priredi ustrezno vrednost.
            !Popravek_naloga = dateNow
            !Oddaja_voznje = dateNow
            !ID_Ekipa = hfgECakanje.TextMatrix(l, 13)
            !ID_Vozilo = hfgECakanje.TextMatrix(l, 14)
            !ID_Voznik = hfgECakanje.TextMatrix(l, 15)
            !ID_Spremljevalec = hfgECakanje.TextMatrix(l, 16)
            !ID_Namembnost = hfgEkipeCakanje.TextMatrix(l, 17)
            !ID_Izpostave =   hfgEkipeCakanje.TextMatrix(l, 19)
            !ID_Oddajni_dispecer = lngIDUporabnika
            !ID_Vnos_dispecer = lngIDUporabnika
            .Update
            ' metoda LinePrint izpiše na tiskalnik
            ' besedilo ki ga podamo kot parameter
            ' V našem primeru izpišemo številko intervencije,
            ' njeno prioriteto, čas ko smojo oddali ekipi ter
            ' številko ekipe
            LinePrint ">Oddaja I:" & .Fields("ID_Prevoza") & _
            "/" & .Fields("ID_Prioriteta") & "/" & _
            .Fields("Oddaja_voznje") & " E:" & _
            hfgEkipeCakanje.TextMatrix(l, 1) _
      End With
      '******
```

Nato poiščemo še ustrezno ekipo, ki ji dopišemo spremembe v stanju ekipe.

```
      '******
      'Ekipe Zasedemo
      With rscmdOddajaEkipe
            If Not (.State = adStateOpen) Then .Open
            If .BOF And .EOF Then
                  MsgBox "btnOddano_Click" & Chr(13) &
                  "rscmdOddajaEkipe EMPTY", vbExclamation
                  'Rollback Transakcija-----------------------
                  GoTo rbt
            End If
            .MoveFirst
            .Find ("ID_Razporeda_ekipe = " &
            hfgEkipeCakanje.TextMatrix(l, 13))
            If .EOF Then
                  MsgBox "btnOddano_Click" & Chr(13) & _
                  "rscmdOddajaEkipe.ID_Razporeda_ekipe = " & _
                  hfgECakanje.TextMatrix(l, 13) & " NE NAJDEM"
                  'Rollback Transakcija-----------------------
                  GoTo rbt
            End If
            !ID_Status_ekipe = se.seIntervencija
            .Update
      End With
      DebugPrint "Vozilu " & hfgECakanje.TextMatrix(l, 1) & _
      " oddana intervencija " & _
      hfgIntervencijeCakanje.TextMatrix(i, 3)
   ' ZANKA PO hfgIntervencijeCakanje
      subUpdatePopravkiInt deDispatch.rscmdOddajaCakanje
      DebugPrint "CommitTrans"
      deDispatch.cnDispatch.CommitTrans
      'Commit Transakcija-------------------------------
End If
btnOsvezi_Click
```

```
      Exit Sub
e:
      DebugPrint "e: Err.Source=" & Err.Source & " Err.Description="_
      & Err.Description
    On Error GoTo 0
rbt:
'Rollback Transakcija-------------------------
      On Error GoTo 0
      DebugPrint "rbt: "
      With deDispatch.rscmdOddajaCakanje
            If (.State = adStateOpen) Then .CancelUpdate
      End With
      With rscmdOddajaEkipe
            If (.State = adStateOpen) Then .CancelUpdate
      End With
      With deDispatch.rscmdPopravkiIntervencij
            If (.State = adStateOpen) Then .CancelUpdate
      End With
      deDispatch.cnDispatch.RollbackTrans
      DebugPrint "RollbackTrans"
End Sub
```

# 5 ZAKLJUČEK

**Odzivi uporabnikov**
Z aplikacijo za beleženje podatkov o nujnih reševalnih prevozih je bil storjen prvi korak računalniškega podprtja dela dispečerskega dela službe nujne medicinske pomoči. Pred uvajanjem smo pričakovali, da bodo določeni uporabniki imeli odpor do uporabe aplikacije. To smo pričakovali predvsem zaradi tega, ker so nekateri starejši dispečerji povsem računalniško neosveščeni in so svoje delo opravljali tudi že po dvajset let. A to se ni zgodilo. Uporabniki (dispečerji) so aplikacijo hitro sprejeli kot pripomoček in do nje niso imeli odpora.
Na podlagi rezultatov, ki jih je prinesla ta aplikacija, se je naročnik odločil, da se razvije tudi aplikacija, ki bo v pomoč reševalcem na terenu.

**Vsebina aplikacije in njena izvedba**
V trenutku, ko je naročnik predstavil idejo aplikacije, smo pričakovali, da bo implementacija razmeroma enostavna. Skozi razvoj pa je kompleksnost aplikaciji rasla. Predvsem se kompleksnost kaže v povezovanju podatkov, ki velikokrat odločajo o nadaljnjem poteku intervencije.
Soočili smo se tudi z dejstvom, da se nobeni podatki ne smejo brisati iz baze podatkov in da je potrebno slediti razvoju intervencije. To smo dosegli na ta način, da smo naredili dve tabeli. V prvi vedno kažemo najnovejše stanje, v drugi pa so zabeležena vsa stanja intervencije.

**Prihodnost aplikacije**
Prihodnji razvoj aplikacije bo temeljil predvsem na povezavi z ostalimi sistemi (obračun opravljenih storitev, …). Omogočiti bo potrebno tudi povezavo s podatki, ki jih bodo vnesli reševalci na terenu, ter implementirati različne protokole (pravila o ravnanju). Protokoli bodo v pomoč dispečerju, pri odločanju, kaj storiti z sprejeto intervencijo. Poleg tega pa bodo v pomoč dispečerjem pri telefonskem vodenju očividca pri nudenju prve pomoči ponesrečencem.

**Moja izkušnja**
To je bila zame prva izkušnja s pisanjem tako obsežne aplikacije, saj sem do sedaj programiral le za potrebe študija. Povsem neznan mi je bil tudi programski jezik Visual Basic. Ob pomoči sodelavcev in primerov na internetu sem se kmalu naučil osnov programiranja v tem jeziku. Nekoliko težje je bilo z implementacijo naročnikovih idej. V zaključni fazi razvoja aplikacije sem se srečal tudi z izdelavo namestitvenega paketa aplikacije ter namestitvijo aplikacije v delovno okolje.

Aplikacija je bila predstavljena na 11. simpoziju urgentne medicine, ki je potekal na Brdu pri Kranju. Tam smo ugotovili, da smo prvi v Sloveniji, ki smo razvili tako podporo dispečerjem v nujni medicinski pomoči.

# 6 LITERATURA


1. ŠULER, ALEŠ, Spoznajmo Microsoft Visual Basic, Šempeter pri Gorici, Flamingo založba,1999
2. HALVORSON MICHAEL, Microsoft Visual Basic 6.0 professional step by step, Redmond, Microsoft Press, 1998
3. MESOJEDEC UROŠ, Visual Basic od začetkov do aplikacije, Izola, Desk, 1994
4. Interno gradivo podjetja Computel d.o.o.
5. Interno gradivo Reševalne postaje kliničnega centra Ljubljana

**Internetne strani**

6. ROCKY MOUNTAIN COMPUTER CONSULTING, VB Helper, http://www.vb-helper.com/index.html, 13.7.2006
7. MICROSOFT CORPORATION, Visual Basic Developer Center http://msdn2.microsoft.com/en-us/vbasic/default.aspx, 2005
8. JUPITERMEDIA CORPORATION, DevX.com The know-how behind application development, http://www.devx.com, 2006
9. JAGARINEC, DARKO, Transakcije v podatkovni zbirki, http://mojmikro.si/articles/42_43_podatkovne_zbirke.pdf, 10.10.2004