UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jurij Rožanec

# APLIKACIJA ZA NADZOR UPORABNIKOVE AKTIVNOSTI PRI UPORABI RAČUNALNIKA

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2013

# I Z J A V A  O  A V T O R S T V U

## diplomskega dela

Spodaj podpisani/-a **Jurij Rožanec**,

z vpisno številko **63090327**,

sem avtor diplomskega dela z naslovom:

**APLIKACIJA ZA NADZOR UPORABNIKOVE AKTIVNOSTI PRI UPORABI RAČUNALNIKA**

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom

  **viš. pred. dr. Igorja Rožanca**

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 14.1.2013 Podpis avtorja:

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

*Tržaška 25*
*1001 Ljubljana, Slovenija*
*telefon: 01 476 84 11*
*01 476 83 87*
*faks: 01 426 46 47*
*01 476 87 11*
*www.fri.uni-lj.si*
*e-mail: dekanat@fri.uni-lj.si*

Št. naloge: 00217/2012
Datum: 04.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JURIJ ROŽANEC**

Naslov: **APLIKACIJA ZA NADZOR UPORABNIKOVE AKTIVNOSTI PRI UPORABI RAČUNALNIKA**

**AN APPLICATION FOR SUPERVISING USER COMPUTER ACTIVITIES**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Nadzor uporabnikove aktivnosti pri uporabi računalnika je učinkovit pristop za preprečevanje različnih zlorab uporabe računalniških virov v okoljih, kjer je drugačen nadzor otežen ali celo nemogoč.

V diplomski nalogi preverite nekatere obstoječe rešitve in zasnujete učinkovito tehnično rešitev za aplikacijo, ki bo spremljala, omejevala in prikazovala uporabnikovo aktivnost pri delu na osebnem računalniku. Aplikacija naj omogoča prikrito spremljanje izvajanja računalniških procesov in uporabe tipkovnice, omejevanje izvajanja prepovedanih procesov, zbiranje podatkov o izvedenih aktivnostih po obdobjih, njihovo analizo in nazoren prikaz.

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

# Zahvala

Zahvaljujem se mentorju viš. pred. dr. Igorju Rožancu za strokovno pomoč in nasvete pri izdelavi diplomske naloge.

Poleg tega pa se zahvaljujem tudi vsem ostalim, ki so mi stali ob strani v času študija in kakorkoli pripomogli pri izdelavi diplomskega dela.

# Kazalo

# Kazalo slik:

# Povzetek

Diplomsko delo predstavlja opis samostojno izdelane aplikacije Nadzornik, ki je namenjena spremljanju oziroma nadzorovanju uporabnikove aktivnosti na osebnem računalniku. Funkcionalnost dosegamo z uporabo različnih tehnik pri zbiranju podatkov o aktivnosti ter prikazu le-teh.

Aplikacija Nadzornik se deli na dva dela. Odjemalec je namenjen zbiranju podatkov o uporabnikovi aktivnosti, administratorska aplikacija pa je namenjena pregledovanju in analiziranju podatkov pridobljenih s strani odjemalca in predstavlja večji del naše rešitve.

Glavne funkcionalnosti in lastnosti odjemalca so zbiranje informacij o procesih v ospredju, beleženje aktivnosti tipkovnice, blokiranje prepovedanih procesov, pošiljanje zbranih podatkov oddaljeni podatkovni bazi, komuniciranje z administratorsko aplikacijo, prikrito delovanje in preprečevanje zaprtja s strani neavtorizirane osebe.

Glavne funkcionalnosti administratorske aplikacije so predstavitev zbranih podatkov na različne načine: vizualno v kontrolah Urnik (tedenski, dnevni podatki) in Histogram (mesečni podatki) ter tekstovno v tabeli. Sledijo analiza podatkov (dnevna, tedenska, mesečna), neposredni nadzor uporabnika (prenašanje zaslonske slike in aktivnosti tipkovnice), upravljanje z uporabniškimi in administratorskimi računi ter procesi in shranjevanje zgodovine zadnjih odprtih analiz in njihovo ponovno odprtje.

Funkcionalnosti administratorske aplikacije so dostopne preko nadgrajenega uporabniškega vmesnika, ki omogoča nastavitev različnih barvnih tematik in stilov.

V prvem delu diplomskega dela predstavimo sama orodja, ki smo jih uporabili pri izdelavi ter podobne obstoječe aplikacije. Nato pa sledi obširen opis razvoja same aplikacije Nadzornik, ki je namenjen predvsem predstavitvi pomembnejših programskih rešitev. V zadnjem delu analiziramo našo aplikacijo in s sklepnimi ugotovitvami zaključimo diplomsko nalogo.

Ključne besede: nadzorna aplikacija, nadzor aktivnosti tipkovnice, analiza uporabe, Microsoft Visual Studio, Microsoft SQL Server.

# Abstract

The thesis represents the description of the application Nadzornik, its purpose is to monitor and supervise all user activities on a personal computer.

Nadzornik consists of two parts. The client gathers the data on the user activity, and the administration application reviews and analysis the data sent from the client. The administration application is a major part of our solution.

Main functions of the Client are: information collecting of process on the foreground; logging of keystrokes; blocking certain forbidden processes; relaying this data to a remote database; retrieving updates from the administration application; hiding from the user and preventing unauthorized closure of the client program.

Main functions of the administration application are: representation of collected data (from the client) visually in controls Urnik (daily and weekly data) and Histogram (monthly data) or displayed as text inside table; data analysis (daily, weekly and monthly); remote user control (streaming the monitor, screen imaging and keyboard activity logs); user, administrator and process management and saving recent history of recently run analysis and their reopenings.

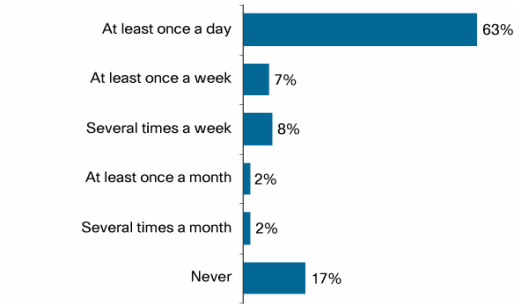
Functions of the administration application are available to user by upgraded user interface, which allows us to set different color schemes and styles.

First part of this dissertation describes the tools we used during creation of the application. The detailed description of the application Nadzornik is presented last. This part describes the most important programming solutions. In the last part we analyze our application and conclude the thesis with conclusions.

Keywords: supervision application, keystroke logging, activity analysis, Microsoft Visual Studio, Microsoft SQL Server.

# 1 Uvod

Osebni računalnik je dandanes nepogrešljiva naprava, ki se že dlje časa uporablja kot eno najbolj pogostih administrativnih oziroma delovnih orodij tako v podjetjih, kot pri posameznikih. Pri tem je mogoče uporabljati različne aplikacije, tudi take, ki niso namenjene (ali so celo prepovedane) za uporabo v podjetjih, vendar zaposleni vseeno včasih posegajo po njih. Raziskave [1] so pokazale, da v povprečju več kot polovica zaposlenih v podjetjih vsakodnevno uporablja delovni računalnik v zasebne namene, kar lahko vidimo na sliki 1.




**Slika 1: Rezultat raziskave o uporabi delovnega računalnika v zasebne namene**

Posledica ni samo očiten padec produktivnosti. Takšno in podobno kršenje pravil delovanja v podjetju (na primer uporaba nedovoljene programske opreme) lahko privede do tveganega izpostavljanja občutljivih podatkov podjetja, kar je po mnenju IT strokovnjakov razlog za kar polovico takšnih incidentov.

Ta dejstva so temeljno izhodišče za izdelavo naše diplomske naloge. Njen namen je izdelati učinkovito nadzorno aplikacijo, s katero bomo sledili uporabi računalnika zaposlenega. Pri tem ne želimo posegati na pravni vidik tega vprašanja, temveč je naš namen preveriti možnosti za učinkovito tehnično izvedbo. Po pregledu literature in tehnologije, ki je na voljo, smo ugotovili, da zastavljeni cilj najbolje dosežemo, če sledimo odprtim aplikacijam uporabnika, uporabe prepovedanih aplikacij sporočamo administratorju ter njihov zagon po potrebi blokiramo.

# 2 Predstavitev uporabljenih orodij in tehnologij

Pri izbiri tehnologije, smo se odločili za uporabo pretežno Microsoftove razvojne opreme, saj je tudi cilj naloge spremljati uporabnike v okolju Windows. Kot razvojno orodje smo uporabili Microsoft Visual Studio 2010. Za hranjenje podatkov skrbi strežnik Microsoft SQL Server 2008 R2, ki ga upravljamo preko orodja Microsoft SQL Server Management Studio. Podatkovno bazo pa smo načrtovali z orodjem PowerDesigner verzije 12.5.

## 2.1 Razvojno okolje Visual Studio

Visual Studio [2] je integrirano razvojno okolje (ang. integrated development environment - IDE), razvito v podjetju Microsoft.

Uporablja se za razvoj :

- konzolnih aplikacij,
- aplikacij z grafičnim vmesnikom,
- Windows Forms aplikacij,
- spletnih strani,
- spletnih aplikacij in
- spletnih storitev.

Podprt razvoj na platformah :

- Microsoft Windows,
- Windows Mobile,
- Windows CE,
- .NET Framework,
- .NET Compact Framework in
- Microsoft Silverlight.

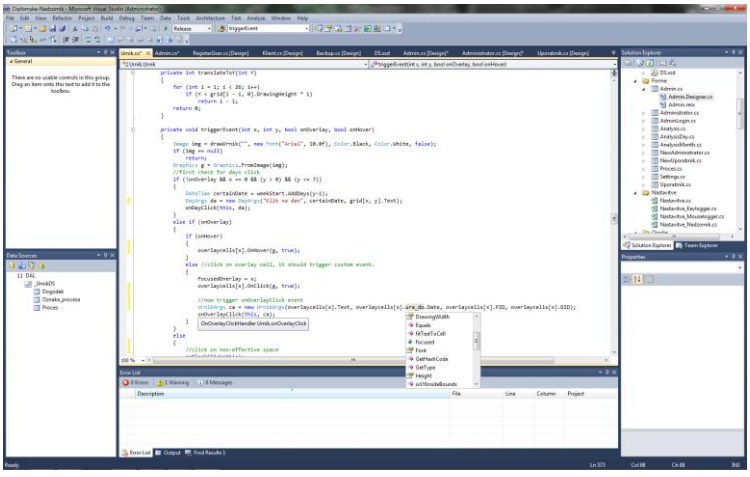
Podprti programski jeziki:

- Visual C#,
- Visual Basic .NET (VB.NET)
- C/C++,
- F#,
- HTML/XHTML,
- JavaScript,
- CSS in
- preko servisov Python, Ruby, M.

Visual Studio za razvoj aplikacij uporablja posebno ogrodje (Microsoft .NET Framework [3]), ki vsebuje zelo veliko zbirko objektno usmerjenih knjižnic (ang. libraries), ki omogočajo poenostavljeno programiranje. Poleg tega skrbi za skupno izvajalno okolje (ang. run environment), kjer se posamezni deli aplikacije, ne glede na programski jezik, združijo v univerzalen vmesni jezik, ki se kasneje lahko nemoteno pretvori v strojni jezik.

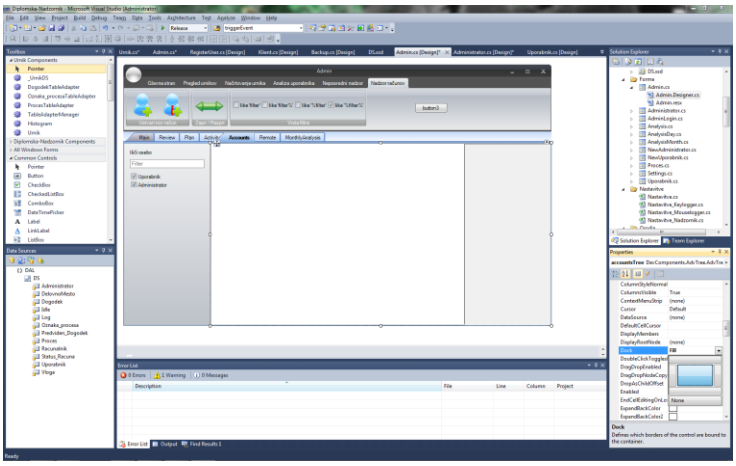
Za naš projekt smo uporabili aplikacijo Windows Forms, ki teče na operacijskem sistemu Microsoft Windows z uporabo programskega jezika C#.

Slika 2 prikazuje kakšno je naše programsko okolje v načinu pogleda programiranja kode. Pri programiranju nam je v veliko pomoč orodje IntelliSense [4], saj nam pomaga pri pregledu nabora funkcij, metod, spremenljivk ter lastnosti uporabljanih objektov ter omogoča samodokončanje ter predlaga elemente zahtevnejših podatkovnih struktur.



**Slika 2: Izgled orodja Visual Studio v načinu pogleda programiranja kode**

Na sliki 3 pa vidimo kakšno je v pogledu vizualnega načrtovanja. Orodje nam omogoča uporabo različnih kontrol, kot tudi spreminjanje njihovih lastnosti [5]. Naša aplikacija ni vezana na samo na določeno resolucijo oziroma velikost, saj nam Visual Studio z uporabo lastnosti kontrol `Dock` (spodaj desno) pomaga skrbeti za velikost kritičnih kontrol in prilagajati njihov izris glede na podan prostor. Deluje tako, da poravna svoje robove s kontrolo s katero je omejen. Omogočena je izbira poravnave na levo, desno, zgoraj, spodaj ter polnjenje praznega prostora.



**Slika 3: Izgled orodja Visual Studio v načinu pogleda vizualnega načrtovanja**

Kot večina razvojnih okolij vsebuje tudi napreden razhroščevalnik, ki pride pogosto prav pri pregledovanju stanj objektov med samim izvajanjem.

Poleg ostalih pomembnih funkcionalnosti bi izpostavili še nadzor nad datotekami v projektu in njihovimi verzijami (zgodovina sprememb) z uporabo orodja Visual SourceSafe [6], oziroma bolj naprednega orodja Team Foundation Server [7], ki zraven poskrbi še za sinhronizacijo in nadzorom datotek med večjo ekipo programerjev.

## 2.2 Microsoft SQL Server in SQL Server Management Studio

### 2.2.1 Microsoft SQL Server

Microsoft SQL Server [8] (MS-SQL) je sistem za upravljanje relacijskih zbirk podatkov – podatkovnih baz.

Osnovna naloga sistema je shranjevanje in prenašanje podatkov glede na zahteve povezane zunanje aplikacije. Povezane aplikacije so lahko na istem računalniku kot strežnik (ang. server), na istem omrežju ali pa povezane preko internetne povezave. Število povezanih aplikacij oziroma uporabnikov ni omejeno.

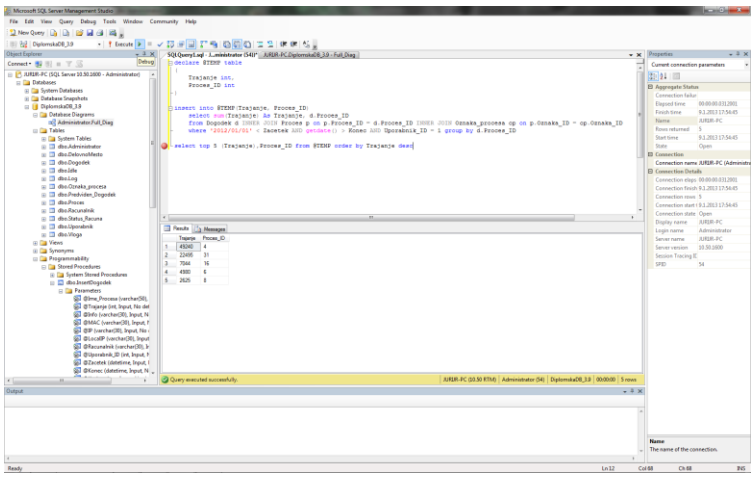
Za upravljanje s podatkovnimi bazami, procedurami in podatki v njih se uporablja programski jezik Transact-SQL (T-SQL) [9] in ANSI SQL. Omogočata podporo shranjevanj osnovnih podatkovnih formatov, vključno s slikami, zvokom, videom ter drugimi multimedijskimi datotekami.

### 2.2.2 SQL Server Management Studio

SQL Server Management Studio [10] je orodje, ko omogoča lažje delo s sistemom Microsoft SQL Server.

Uporablja se za konfiguracijo, urejanje in administriranje vseh komponent omenjenega sistema. Torej omogoča nam na primer izvajanje SQL skript in njihovo razhroščevanje, nastavljanje uporabniških vlog in pravic, upravljanje s podatkovnimi tabelami in funkcijami ter celoten nadzor in upravljanje podatkovnih baz.

Slika 4 prikazuje primer uporabe orodja za izvajanje poizvedbe.



**Slika 4: Primer uporabe orodja Microsoft SQL Server Management Studio**

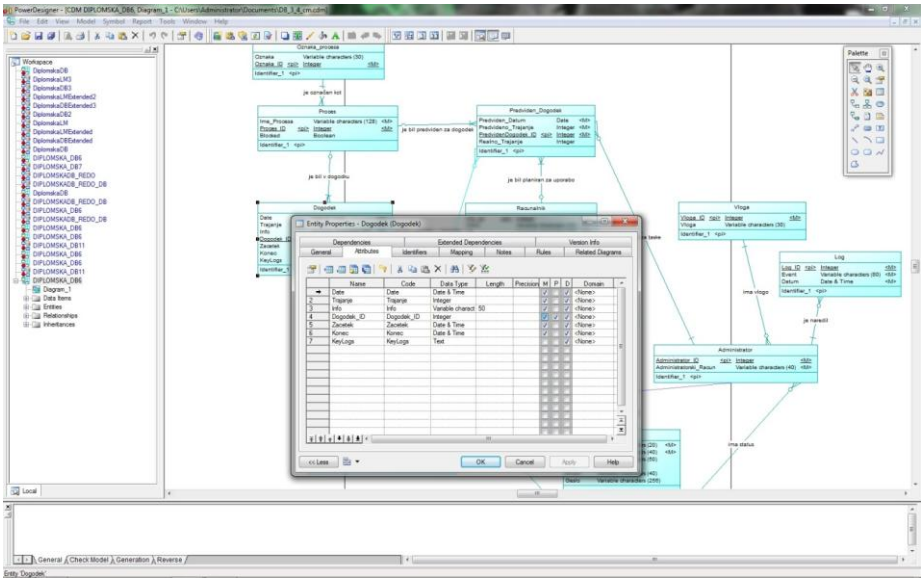
## 2.3 Power Designer

Power Designer [11] je orodje za modeliranje, ki uporabniku omogoča, da lahko podatke in podatkovne sheme vizualizira, načrtuje in analizira.

Vključuje podporo za :

- modeliranje poslovnih procesov (ang. business process modeling),
- modeliranje podatkovnih baz (ang. data modeling),
- modeliranje podatkovnih skladišč (ang. data warehouse modeling),
- objektno modeliranje (ang. object modeling),
- XML modeliranje (XML modeling) in
- samodejno generiranje programske kode za različne programske jezike.

Orodje je bilo za nas zelo uporabno, saj je podatkovno bazo mnogo lažje načrtovati, če interakcija poteka preko uporabniškega vmesnika in je predstavljena vizualno, kot vidimo na sliki 5. Poleg tega smo uporabili možnost samodejnega generiranja SQL kode, kar nam prihrani dosti časa.



**Slika 5: Primer uporabe orodja PowerDesigner**

# 3 Obstoječe aplikacije za nadzor uporabe

## 3.1 Aplikacija All In One Keylogger

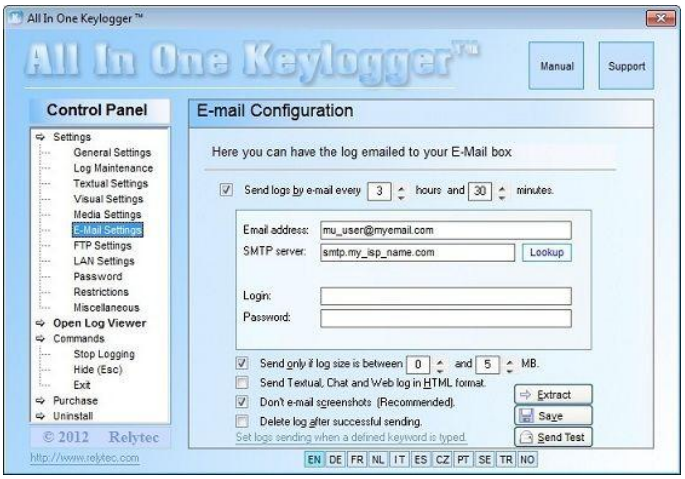
All In One Keylogger [12] je plačljiva aplikacija, ki je namenjena prikritemu beleženju aktivnosti na računalniku.

Omogoča beleženje (ang. logging) :

- uporabe tipkovnice (ang. keystroke logging),
- uporabe aplikacij (ang. application monitoring),
- uporabe mikrofona (ang. microphone logging) in
- uporabe tiskalnika (ang. printer logging).

Poleg funkcionalnosti beleženja omogoča shranjevanje trenutnega pogleda namizja (ang. screenshot logging), ki se proži glede na nastavljen časovni interval in shranjuje na disk.

Datoteke logov se lahko na določen interval prenašajo do nadzorne osebe preko elektronske pošte, FTP ali pa se prekopirajo na USB ključek. Primer nastavitev pošiljanja preko elektronske pošte lahko vidimo na sliki 6.




**Slika 6: Aplikacija All In One Keylogger**

Vsebuje pregledovalnik logov (ang. log viewer), kjer so zbrani podatki predstavljeni s pomočjo tabele.

Odlikuje se po zmožnosti zajemanja pogovorov socialnih orodij. Manjka pa predvsem boljša predstavitev in analiza zbranih podatkov.

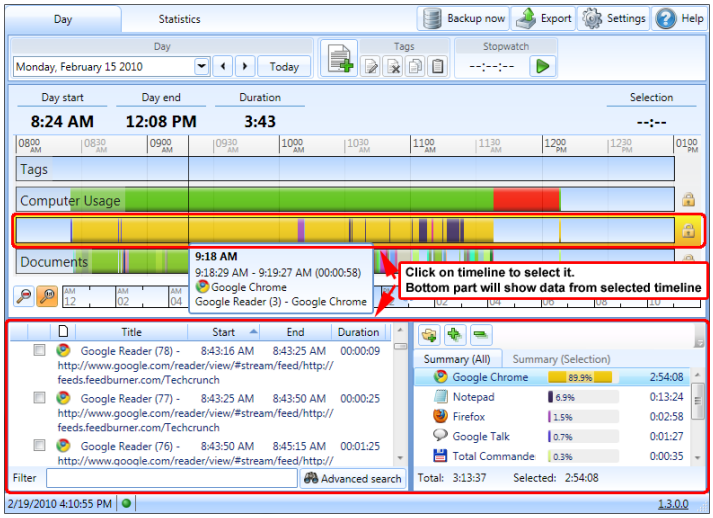
## 3.2 Aplikacija ManicTime

ManicTime [13] je prosto dostopna aplikacija, katere namen je sledenje času, ki smo ga porabili za delo.

Omogoča beleženje uporabe aplikacij in obiskanih spletnih strani, kot tudi čas odsotnosti od računalnika.

Zbrani podatki se shranjujejo na lokalno podatkovno bazo.

Pregledovalnik podatkov je hkrati analiza podatkov in je prikazan na sliki 7. Vsebuje urnik, ki predstavlja čas odsotnosti od računalnika, časovni razpon uporabe aplikacij in časovni razpon posamezne odprte spletne strani.




**Slika 7: Aplikacija ManicTime**

Aplikacija ManicTime ima pregleden in učinkovit način predstavitve zbranih podatkov. Merjenje opravljenega dela je uporabna funkcionalnost za ljudi, ki zaračunavajo svoje storitve glede na porabljen čas. Njen namen je sledenje in analiza lastne aktivnosti, torej ne potrebuje dodatnih funkcionalnosti. S tem zaključimo, da nam aplikacija ponudi vse potrebne informacije, ki si jih želimo.

# 4 Izdelava aplikacije Nadzornik

## 4.1 Ideja

Ideja za aplikacijo Nadzornik je nastala iz vprašanja, kako bi lahko sledili uporabi računalnika. Delni odgovor je prišel s strani prikritih aplikacij, to so tako imenovani »Keyloggerji«. Te vrste aplikacij izrabljajo možnost prestrezanja pritiskov tipk na tipkovnico, odtujevanje informacij iz spletnih brskalnikov in drugimi tehnikami z ciljem, da pridobijo oziroma ukradejo čim več uporabnikovih podatkov. Osredotočajo se na primer na osebne podatke, bančne podatke ali pa uporabniške račune z gesli aplikacij in spletnih strani. Protivirusne (ang. antivirus) in protivohunske(ang. anti-spyware) aplikacije večino teh aplikacij zaznavajo kot zlonamerne (ang. virus). Ni pa vsaka uporaba takšnega programa zlonamerna, saj se lahko uporabi tudi v preventivne namene, ali za nadzor. Primer take uporabe bi lahko bil preventivni nadzor nad otroci, ki brskajo po internetu po potencialno nevarnih straneh, ali pa kontrolni nadzor zaposlenih, ki med delom preveč radi brskajo po internetu in uporabljajo aplikacije, ki ne spadajo k delu.

Idejni načrt:

- Izognemo se detekciji zlonamerne kode, zato raje uporabljamo bolj pasivne načine sledenja aktivnosti uporabnika. Aplikacijo lahko razvijemo kot bolj ali manj pasivno sledenje aplikacijam, ki so trenutno v ospredju.
- Oblikovali bomo listo nedovoljenih procesov in ob detekciji le-teh primerno odreagirali.
- Aplikacija mora biti tudi ustrezno zavarovana pred nepooblaščenim zaprtjem, saj drugače nebi imela smisla.
- Dostop do pridobljenih podatkov mora biti sproten, kar nam omogoča takojšen odziv na dogodke.

Torej glavna ideja je izdelati aplikacijo, ki bi uporabniku dovoljevala pregled in nadzor nad uporabo računalnika ter uporabljenimi aplikacijami.

## 4.2 Funkcionalne zahteve

Natančnejše specifične funkcionalne zahteve so:

- Želimo razviti aplikacijo za pregled in nadzor aktivnosti na osebnem računalniku delavcev v podjetju.
- Aplikacijo naj bo možno uporabljati na vseh računalnikih podjetja, ki imajo nameščen operacijski sistem Windows.
- Aplikacije mora omogočati pregled aplikacij v obliki urnika, ki jih uporablja delavec in opozarjanje na aplikacije, ki na delu niso dovoljene. V tem primeru naj bo možno pridobiti časovni razpon in verodostojen dokaz o pojavitvi dogodka.
- Poleg tega pa naj se za vsako pojavitev aplikacije posebej beleži tudi aktivnost tipkovnice – besedilo, ki ga uporabnik vpisuje.
- Administrator mora poleg splošnega pregleda aktivnosti v koledarju imeti možnost pregleda analize podatkov za vsak dan posebej, celoten teden in celoten mesec. Posamezne analize naj bo možno shraniti in predstaviti izven aplikacije.
- Obravnava podatkov naj bo za vsakega uporabnika individualna.
- Podatki določenega uporabnika naj bodo administratorju dostopni v realnem času, prav tako pa naj aplikacija dovoljuje (ang. real-time) pogled na monitor uporabnika v realnem času in njegovo aktivnost.

## 4.3 Specifikacija analize zahtev aplikacije

Specifikacija omejitev:

- Aplikacijo bo možno uporabljati na osebnih računalnikih in prenosnikih, ki imajo nameščeno eno izmed novejših izdaj operacijskega sistema Microsoft Windows.

  Poleg tega pa mora biti nameščena še novejša verzija ogrodja .NET Framework.

- Aplikacija bo za shranjevanje podatkov uporabljala strežnik SQL Server 2008 R2, ki bo omogočal shranjevanje podatkov tudi v primeru, ko zaposleni uporablja službeni prenosnik, in dela od doma, z vzpostavljeno internetno povezavo.
- Aplikacija za osnovno delovanje ne potrebuje nujno internetne povezave (razen izjeme v prejšnji točki). Deluje lahko tudi na lokalnem omrežju, vendar morajo biti vanj povezani vsi računalniki ter podatkovni strežnik.

Specifikacija delovanja:

- Aplikacija bo razdeljena na dva dela:
  - odjemalec nameščen pri nadzorovani osebi in bo pošiljal podatke strežniku,
  - administratorska aplikacija, ki služi pregledu zajetih podatkov.

- Odjemalce bo na posamezne računalnike namestil administrator, pri čemer bo tudi dodelil določen računalnik specifični osebi.
- Delovanje odjemalca bo skrito pred nadzorovano osebo, in tudi odjemalec sam bo primerno zaščiten pred nepooblaščenim izklopom.
- Glavna naloga odjemalca bo beleženje uporabljanih aplikacij in podatkov povezanih z njo (aktivnost tipkovnice). Podatke takoj po zajetju posreduje strežniku, od koder so takoj na voljo za uporabo. V primeru, da pride do detekcije nedovoljenega programa, se administratorju pošlje obvestilo, ki vsebuje slikovni dokaz. Odjemalec bo omogočal administratorski aplikaciji, da se poveže in prenaša sliko zaslona nadzorovanega uporabnika.

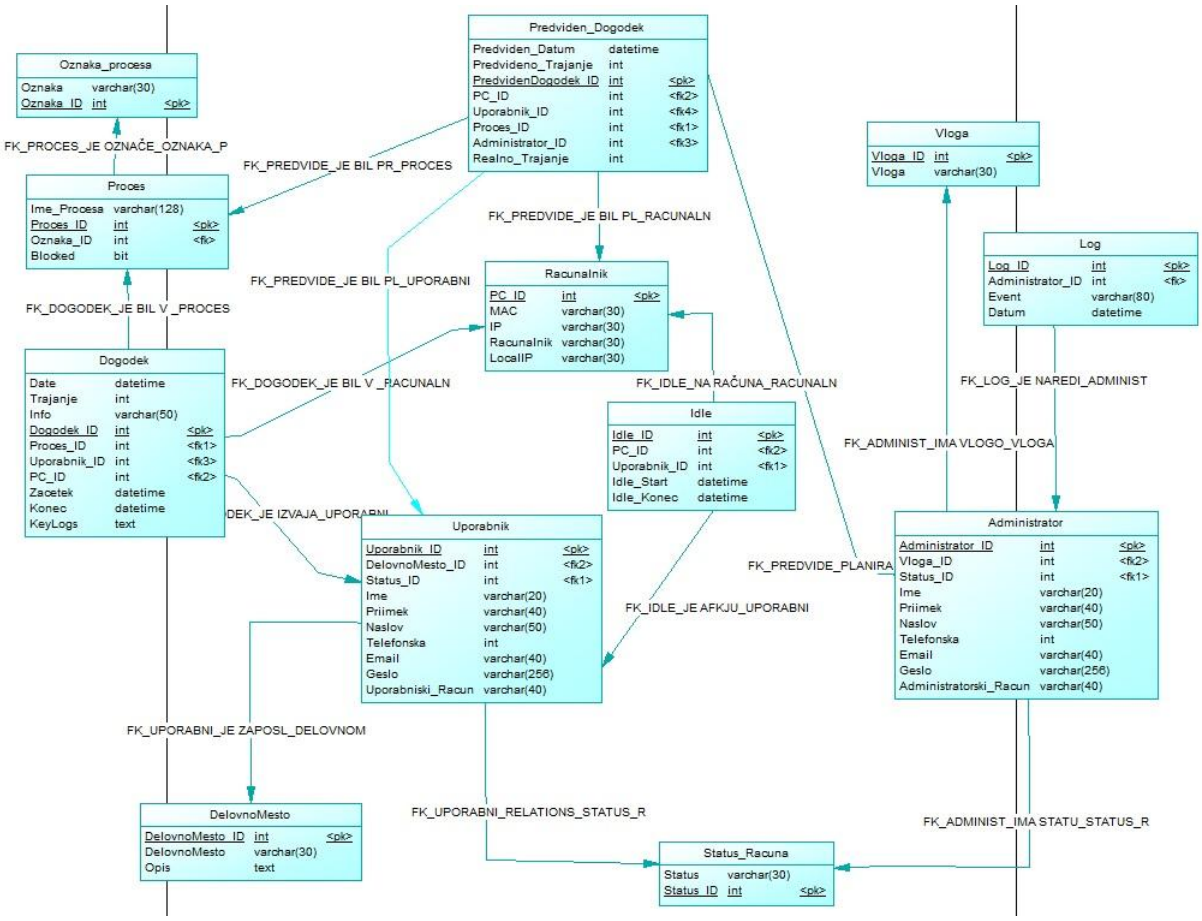
- Administratorska aplikacija bo imela ustrezno zaščito pred nepooblaščenim dostopom.
- Administratorska aplikacija bo administratorju omogočala izbiro nadzorovane osebe. Po izbiri bo omogočeno brskanje po zbranih podatkih o osebi. Izdelana bo nova kontrola, ki bo vizualno predstavljala urnik osebe. Ob pregledu urnika bo ta kontrola prikazovala celoten urnik osebe za tekoči teden. Izbira iz koledarja bo omogočala spreminjanje tedna, oziroma dneva pregledovanja. Aktualni urnik za določen dan bo predstavljen analogno v tabeli. Na voljo naj bodo različni filtri za enostavnejše iskanje določenih procesov. Omogočeno bo označevanje procesov, in tudi blokiranje le-teh. Pregled urnikov bo omogočal tudi analizo za dan oziroma teden, tako za vse procese osebe, ali pa samo določen proces. Pri analizi bodo vidni tudi tekstovni logi, ki so bili zajeti med posameznimi trajanji procesov. Posebej se lahko izvede še analiza za celoten mesec, ki bo vsebovala posebno kontrolo za prikaz grafa uporabe procesov določenih dni in filter najbolj uporabljanih procesov. Vsaka analiza se lahko izvozi v PDF format. Izvedena bo tudi funkcionalnost neposrednega nadzora, kjer bo lahko administrator videl, katere osebe so trenutno aktivne, in se po želji povezal na Klient enega od njih ter v realnem času prejemal zaslonsko sliko nadzorovane osebe, kot tudi njeno aktivnost tipkovnice.
- Administratorska aplikacija bo imela možnost urejanja in dodajanja informacij o računih oseb, ki so nadzorovane, in vseh administratorjev.
- Administratorska aplikacija bo tudi imela možnost personalizacije uporabniškega vmesnika ter seznam nazadnje odprtih kartic oziroma analiz.

## 4.4 Načrtovanje podatkovne baze

Glede na specifikacijo analize zahtev aplikacije smo z orodjem Power Designer najprej zasnovali podatkovni model (ang. conceptual data model - CDM) baze, ki bo omogočala shranjevanje vseh potrebnih podatkov. V ta namen smo ustvarili več tabel, ki jih bomo bolj podrobno predstavili v nadaljevanju.

Power Designer ponuja avtomatsko generiranje fizičnega podatkovnega modela (ang. physical data model - PDM) in pripadajoče kode za generiranje podatkovne baze iz konceptualnega modela. Tako smo pridobili našo MS-SQL podatkovno bazo.

Slika 8 prikazuje celoten logični model naše podatkovne baze.




**Slika 8: Orodje PowerDesigner - Logični model podatkovne baze**

**Tabela `PROCES`**

Tabela se uporablja za shranjevanje vseh procesov (aplikacij), ki se med izvajanjem Nadzornika pojavijo v ospredju. Torej za vsak nov zabeležen proces, katerega zapis v tabeli še ne obstaja, se ustvari nov zapis.

Podatkovna polja:

- `Proces_ID` ( INT, primarni ključ, obvezen ) – identifikacijska številka procesa;
- `Ime_Procesa` ( VARCHAR, obvezen ) – ime procesa (ni dejansko ime aplikacije) in je edinstveno glede na aplikacijo;
- `Blocked` ( BIT ) – atribut, ki označuje ali naj Nadzornik blokira določen proces;
- `Oznaka_ID` ( INT , tuji ključ, obvezen) – označuje proces;

**Tabela `OZNAKA_PROCESA`**

Tabela se uporablja za shranjevanje različnih oznak procesa:

- Neznan,
- Nedovoljen in
- Dovoljen.

Podatkovna polja:

- `Oznaka_ID` ( INT, primarni ključ, obvezen) – identifikacijska številka oznake;
- `Oznaka` ( VARCHAR, obvezen ) – tekst, ki predstavlja oznako procesa;

**Tabela `DOGODEK`**

Tabela `DOGODEK` je najpomembnejša za izvajanje aplikacije. Vanjo se shranjujejo vsi zapisi o pojavitvah procesa in relativni podatki kot je na primer, katera izmed nadzorovanih oseb ga uporablja in s katerega računalnika so prišli podatki.

Podatkovna polja:

- `Dogodek_ID` ( INT, primarni ključ, obvezen) – identifikacijska številka dogodka;
- `Date` (DATETIME, obvezen ) – datum dogodka;
- `Trajanje` ( INT, obvezen ) – trajanje dogodka v sekundah;
- `Zacetek` ( DATETIME, obvezen ) – čas začetka pojavitve dogodka;
- `Konec` ( DATETIME, obvezen ) – čas konca pojavitve dogodka;
- `Info` ( VARCHAR, obvezen ) – tekst, ki predstavlja naslov(ang. title) okna procesa;
- `Keylogs` ( TEXT ) – tekstovni log, ki je bil zajet med uporabo procesa;

- `Proces_ID` ( INT , tuji ključ, obvezen ) – označuje proces;
- `Uporabnik_ID` ( INT , tuji ključ, obvezen ) – označuje uporabnika;
- `PC_ID` ( INT , tuji ključ, obvezen ) – označuje računalnik;

**Tabela `RACUNALNIK`**

Tabela `RACUNALNIK` služi shranjevanju podatkov o računalnikih, ki so jih uporabljali nadzorovani uporabniki.

Podatkovna polja:

- `PC_ID` ( INT, primarni ključ, obvezen) – identifikacijska številka računalnika;
- `Racunalnik` ( VARCHAR, obvezen ) – Windows označba imena računalnika;
- `MAC` (VARCHAR, obvezen ) – MAC ( ang. Media Access Control ) naslov;
- `IP` (VARCHAR, obvezen ) – javni IP ( ang. Internet Protocol ) naslov;
- `LocalIP` (VARCHAR, obvezen ) – lokalni IP naslov ( znotraj omrežja );

**Tabeli `UPORABNIK` in `ADMINISTRATOR`**

Tabeli omogočata shranjevanje podatkov nadzorovanih oseb in administratorjev.

Podatkovna polja:

- `Uporabnik_ID` / `Administrator_ID` (INT, primarni ključ, obvezen) – identifikacijska št. osebe;
- `Uporabniski_Racun` / `Administratorski_Racun` ( VARCHAR, obvezen ) – uporabniško ime osebe;
- `Ime` ( VARCHAR, obvezen ) – ime osebe;
- `Priimek` ( VARCHAR, obvezen ) – priimek osebe;
- `Naslov` ( VARCHAR ) – naslov osebe;
- `Telefonska` ( INT ) – telefonska številka osebe;
- `Email` ( VARCHAR ) – spletni poštni račun osebe;
- `Geslo` ( VARCHAR ) – kriptirano geslo računa;
- `Status_ID` ( INT, tuj ključ, obvezen ) – označuje status računa;
- `DelovnoMesto_ID` ( INT, tuj ključ ) – označuje delovno mesto osebe;
- (`ADMINISTRATOR`) `Vloga_ID` ( INT, tuj ključ, obvezen ) – označuje administratorsko vlogo;

**Tabela `STATUS_RACUNA`**

Tabela se uporablja za shranjevanje različnih statusov računa:

- Aktiven,
- Neaktiven.

Podatkovna polja:

- `Status_ID` ( INT, primarni ključ, obvezen ) – identifikacijska številka statusa računa;
- `Status` ( VARCHAR, obvezen ) – tekstovna predstavitev statusa računa;

**Tabela `VLOGA`**

Tabela omogoča shranjevanje različnih vlog v katerih lahko dela administrator.

Podatkovna polja:

- `Vloga_ID` ( INT, primarni ključ, obvezen ) – identifikacijska številka vloge;
- `Vloga` ( VARCHAR , obvezen ) – tekstovna predstavitev vloge;

**Tabela `DELOVNOMESTO`**

Tabela omogoča shranjevanje različnih delovnih mest znotraj podjetja v katerih delajo nadzorovane osebe.

Podatkovna polja:

- `DelovnoMesto_ID` ( INT, primarni ključ, obvezen ) – identifikacijska številka delovnega mesta;
- `DelovnoMesto` ( VARCHAR, obvezen ) – naziv delovnega mesta;
- `Opis` (TEXT ) – opis delovnega mesta;

**Tabela `LOG`**

Tabela omogoča shranjevanje zapisov (ang. logs) o aktivnostih administratorja v aplikaciji.

Podatkovna polja:

- `Log_ID` ( INT, primarni ključ, obvezen ) – identifikacijska številka zapisa;
- `Datum` ( DATETIME, obvezen ) – datum in čas zapisa;
- `Event` ( VARCHAR, obvezen ) – tekstovna predstavitev zapisa;

**Tabela `PREDVIDEN_DOGODEK`**

Tabela omogoča shranjevanje planiranih procesov s predvidenim trajanjem za določen dan.

Podatkovna polja:

- `PredvidenDogodek_ID` ( INT, primarni ključ, obvezen ) – ident. št. predvidenega dogodka;
- `Predviden_Datum` ( DATE, obvezen ) – datum predvidenega procesa;
- `Predvideno_Trajanje` ( INT, obvezen ) – predvideno trajanje procesa v sekundah;
- `Proces_ID` ( INT, tuji ključ, obvezen ) – označuje predviden proces;
- `Uporabnik_ID` ( INT, tuji ključ, obvezen ) – označuje uporabnika, ki mu planiramo proces;
- `Administrator_ID` ( INT, tuji ključ, obvezen ) – označuje administratorja, ki je planiral proces;
- `PC_ID` ( INT, tuji ključ, obvezen ) – označuje računalnik, kjer naj bi se proces izvajal;

**Tabela `IDLE`**

Tabela omogoča shranjevanje časovnih razponov, ko je bil nadzorovan uporabnik zaznan kot odsoten, čeprav se je določen proces izvajal.

Podatkovna polja:

- `Idle_ID` ( INT, tuji ključ, obvezen ) – identifikacijska številka odsotnosti;
- `Idle_Start` ( DATETIME, obvezen ) – začetek odsotnosti uporabnika;
- `Idle_Konec` ( DATETIME, obvezen ) – konec odsotnosti uporabnika;

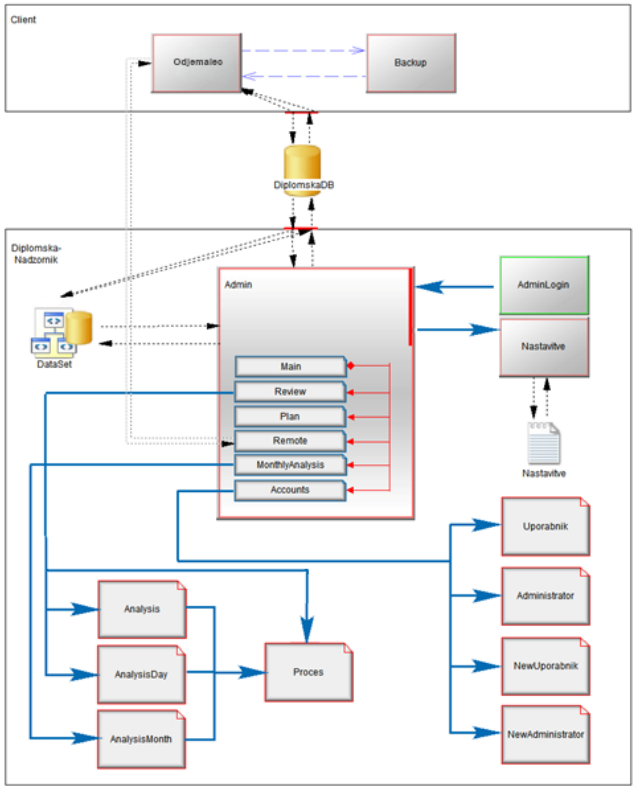
## 4.5 Predstavitev posameznih delov aplikacije in njihove izvedbe

Ker je aplikacija Nadzornik precej obsežna, se bomo posebej lotili razlage posameznih delov aplikacije. Najprej predstavimo delovanje delov aplikacije kot celote.

Na sliki 9 so prikazani prehodi med formami, karticami in zavihki. Poleg tega pa lahko vidimo pretok podatkov znotraj aplikacije, ki je označen s črnimi črtanimi črtami.

Odjemalec in Backup medsebojno sodelujeta, da ju ni mogoče zapreti, kar bomo spoznali pri opisu delovanja projekta Client. Administratorska aplikacija pridobi večino podatkov odjemalca preko podatkovne baze, prvi del preko neposrednih poizvedb, drugi pa z uporabo lokalne entitete za shranjevanje iz podatkov baze – strukturo Dataset [14]. Obstaja pa izjema, kjer uporabimo protokol TCP za vzpostavitev povezave med odjemalcem in zavihkom neposrednega nadzora uporabnika.

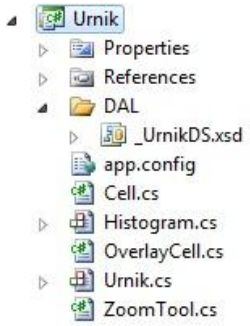
Na tej sliki lahko opazimo še prehode med glavnimi elementi v administratorski aplikaciji. Vidimo, da se glavna aplikacija, kjer so vsi zavihki medsebojno povezani, odpre iz forme za vpis administratorja. Preko zavihkov pa nam omogoča dostop do vseh kartic.




**Slika 9: Prikaz prehajanja procesov in podatkov**

V programskem orodju Visual Studio smo celotno rešitev (ang. solution) razdelili na tri projekte, ki jih bomo sedaj predstavili bolj podrobno.

### 4.5.1 Projekt Urnik




**Slika 10: Datoteke v projektu Urnik**

Cilj projekta Urnik je bila izdelava dveh uporabniških Windows Forms kontrol, ki se uporabljata v glavni aplikaciji.

#### *4.5.1.1 Kontrola »Urnik«:*

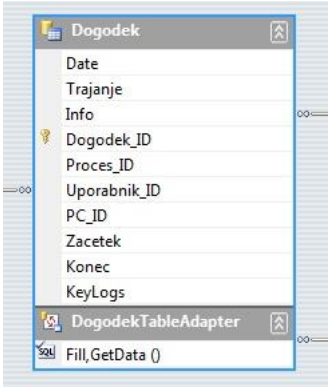
Kontrola Urnik se uporablja za interaktivno vizualno predstavitev urnika uporabnika.

Zanimivo dejstvo je, da smo za osnovo kontrole vzeli kontrolo za prikaz slik – PictureBox, ki smo jo dedovali v delni razred (ang. partial class). S tem nam je bila prihranjena izdelava nekaterih osnovnih dogodkov (ang. eventov).

**Delo s podatki**

Kontrola bere podatke iz podatkovne baze. Ker bi bilo branje podatkov neposredno iz baze ob vsakem ponovnem izrisu precej potratno, smo uporabili strukturo Dataset.

Dataset nam omogoča, da vanj lokalno shranimo podatke iz baze. Na ta način se podatki tabel (`DOGODEK`, `PROCES`, `OZNAKA_PROCESA`) prenesejo le enkrat. Kot vidimo na sliki 11, je vizualna predstavitev tabele v tej strukturi zelo podobna dejanski v bazi. Z nastalo strukturo lahko izvajamo le bolj preproste poizvedbe, vendar so te več kot dovolj za naše potrebe znotraj kontrole Urnik.




**Slika 11: Dataset Dogodek**

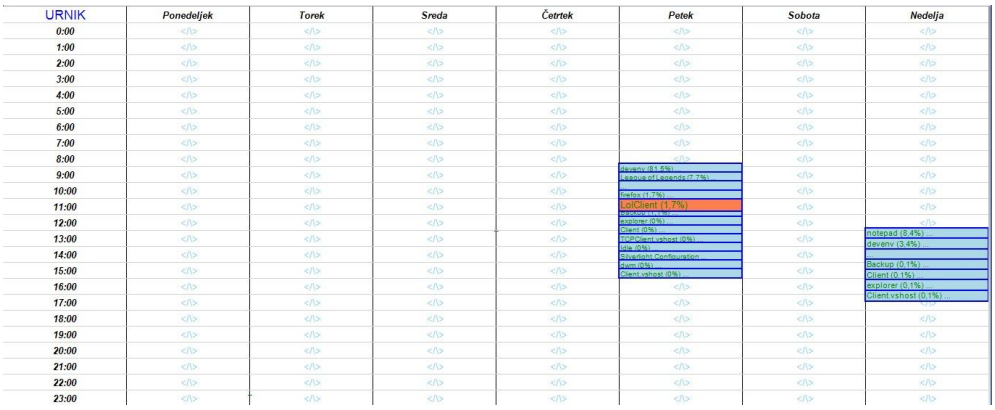
Ko so podatki pripravljeni na uporabo, iz njih po filtriranju in procesiranju glede na izbrano obliko prikaza pridobimo potrebne podatke za izris urnika. Podatke potrebne za vizualiziranje procesa shranimo v namensko programsko strukturo, od koder so na voljo za izris.

### Izris kontrole Urnik

Deluje tako, da s pomočjo razreda za risanje objektov (`Graphics`) dobesedno narišemo celotno sliko, ki jo nato prikažemo v kontroli za prikaz slik.

Začeli smo z risanjem glavnih črt in dodali posameznim objektom tekst z dnevi in urami. Nato smo se sprehodili čez tabelo objektov `Cell` (predstavljajo osnovne celice v naši kontroli) in `OverlayCell` (predstavljajo celice s pojavitvami dogodkov) ter na vsakem objektu klicali funkcijo za izris.

Pri izrisu lahko predstavimo urnik za tekoči teden, kar vidimo na sliki 12, lahko pa tudi samo posamezen dan.




**Slika 12: Kontrola Urnik - prikaz aktivnosti celotnega tedna**

Ker želimo, da kontrola podpira uporabo funkcionalnosti samodejnega prilagajanja velikosti elementov (ang. docking), smo morali poskrbeti, da se kontrola pravilno izriše pri katerikoli velikosti. To smo dosegli z risanjem glede na višino in širino, ki nam je na voljo. Glede na leto smo prilagajali velikost objektov `Cell` in `OverlayCell`. V teh objektih smo nato prilagajali velikost besedila (ang. font size), da se ujema v mejah normale. Ob dosegu kritične meje izrisa, smo omejili količino besedila, ki je vidna. Problem smo nato dokončno rešili s posebnim dogodkom (ang. custom event), ki bo opisan v nadaljevanju.

### Interakcija s kontrolo Urnik

Za interakcijo je poskrbljeno z uporabo različnih dogodkov, ki so zaključek spreminjanja velikosti forme - `ResizeEnd`, pritisk levega gumba - `MouseDown` in premik miške - `MouseMove`. Ti dogodki sami po sebi niso dovolj, ampak služijo proženju dogodkov po meri, ki smo jih izdelali sami in se uporabljajo tako znotraj same kontrole kot tudi pri proženju posebnih dogodkov izven nje.

Dogodki po meri:

- `onOverlayClick` – predstavlja klik na celico, ki predstavlja dogodek (proces);
  `onDayClick` – predstavlja klik na celico, ki predstavlja besedilo dneva;
  `onBlankClick` – predstavlja klik v »prazno«;

  Ti dogodki se prožijo ob pritisku levega guma miške. Kontrola najprej za vsako celico, ki predstavlja dogodek, preveri, ali je koordinata klika znotraj njegovih izrisanih mej, drugače pa izračuna, na katero celico se je zgodil klik miške in temu primerno sproži dogodek.

  Ti dogodki imajo implementiran prožilec (ang. event handle) zunaj kontrole, tako smo lahko na primer klik celice dogodka uporabili, da smo v aplikaciji Nadzornik odprli kartico tega procesa. Seveda pa smo za vsak dogodek morali še napisati svojo instanco argumentov dogodka (ang. event arguments), ki nam posreduje potrebne informacije za nadaljnjo uporabo.

- `onHover` – predstavlja dogodek, ko se z miško premaknemo nad celico, ki predstavlja dogodek;

  Dogodek se preverja ob premiku miške in vsebuje nekaj optimizacijske kode, da ne porabi več pomnilnika, kot je v mejah normale. Sproži se, ko je miška nad celico dogodka.

  Dogodek služi kot rešitev, če v celici ni dovolj prostora za prikaz celotnega besedila. Deluje tako, da ob sprožitvi izriše okno primerne velikosti s celotnim besedilom.
- `redrawOnResize` – predstavlja dogodek znotraj kontrole, ki skrbi za ponovni izris kontrole, ko spremenimo velikost same kontrole;

#### *4.5.1.2 Kontrola »Histogram«:*

Kontrola Histogram se uporablja za grafično predstavitev aktivnosti uporabnika po posameznih dnevih. V obliki histograma predstavimo aktivnost, os x predstavlja določen dan v mesecu, os y pa skupno število minut v določenem dnevu, ko je bil analiziran uporabnik aktiven.

Podobno kot pri Urniku smo si prihranili nekaj dela in za osnovo naše kontrole vzeli kontrolo za prikaz slik.

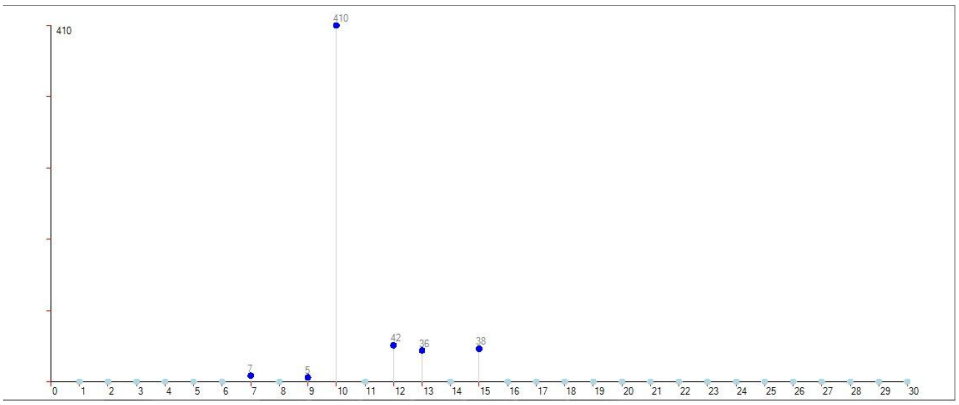
### Delo s podatki

Kontrola sprejme podatke iz aplikacije v obliki tabele tipa `Integer`. Število dni v histogramu je enako številu dni v mesecu, ki pa je določeno z velikostjo vhodne tabele. Dejanska aktivnost dni je podana v poljih tabele.

### Izris kontrole Histogram

Kontrolo izrišemo na podoben način kot pri Urniku.

Začnemo z izračunom točk osi in dolžine osi glede na podan prostor, ki ga imamo na voljo za izris in iz vrednosti dobimo maksimalno vrednost. Glede na pridobljene podatke se določi razmerje za izris. Nato izrišemo obe osi in vmesne točke za merila, kot tudi oznake z vrednostmi teh točk. Ko imamo podlago izrisano, se lotimo izrisa vrednosti, ki smo jih pridobili iz vhodne tabele. Glede na že prej izračunana razmerja z lahkoto pridobimo potrebne koordinate, kamor izrišemo manjši krogec ter nad njo dejansko vrednost točke. Končen videz lahko vidimo na sliki 13.

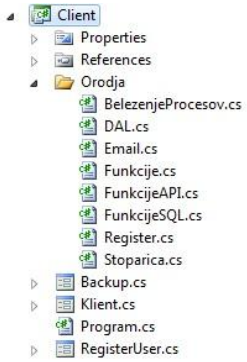



**Slika 13: Kontrola Histogram - prikaz aktivnosti meseca**

### Interakcija s kontrolo Histogram

Interakcija pri tej kontroli sicer ni potrebna, ampak je bilo kot pri kontroli Urnik potrebno poskrbeti še za primerno funkcionalnost prilagajanja izgleda kontrole ob spremembah velikosti, torej ponoven izris, glede na novo velikost.

## 4.5.2 Projekt Client




**Slika 14: Datoteke v projektu Client**

Cilj projekta Client je aplikacija, ki se izvaja na strani uporabnika. Torej glavni namen projekta je zbiranje podatkov o uporabnikovi aktivnosti in posredovanju le-teh oddaljeni podatkovni bazi.

#### 4.5.2.1 Razred Funkcije

Vsebuje 3 javne statične metode s katerimi pridobimo informacije o samem računalniku.

- `GetLocalIP`() nam vrne IP naslov znotraj mreže (ang. local internet protocol). Zapis z oznako lokalnega naslova poiščemo v strežniku domenskih imen (ang. Domain Name Server - DNS).

- `GetPublicIP`() vrača javni IP naslov (ang. public internet protocol) računalnika. Poveže se na spletno stran, in iz prejetega odziva (ang. response) v html obliki, enostavno izluščimo IP naslov.

- `GetMacAddress`() vrne MAC naslov mrežne kartice. Najdemo ga na listi omrežnih vmesnikov. Preverimo kateri od njih ima najvišjo hitrost in kot rezultat vrnemo njegov fizični naslov.

#### 4.5.2.2 Razred FunkcijeAPI

Vsebuje javne statične metode, ki uporabljajo funkcije sistemske knjižnice `user32.dll` .

V njih za referenco na trenutno aktivno okno uporabljamo funkcijo `GetForegroundWindow`(), ki je prav tako del te knjižnice.

- `aktivnoOkno`() vrača tekst, ki je v naslovu (ang. title) procesa s pomočjo funkcije `GetWindowText`().

- `dobiAktivenProces`() vrne objekt tipa `Process` s pomočjo funkcije `GetWindowThreadProcessId`().

- `casBrezEventov`() vrne spremenljivko tipa uint . Rezultat v milisekundah je izhod funkcije `GetLastInputInfo`(), odštet od preteklega časa zagona sistema.

#### 4.5.2.3 Razred FunkcijeSQL

Vsebuje javne statične metode, ki skrbijo za prenos podatkov med aplikacijo in bazo.

Metoda `dodajDogodek`() vrne spremenljivko tipa bool, katere vrednost je odvisna od uspešnosti izvedbe.

SQL proceduri `InsertDogodek` pošljemo več parametrov v zvezi z dogodkom in uporabnikom.

Namenjena je vnosu podatkov o posamezni pojavitvi dogodka v podatkovno bazo.

Metoda `validateAccount`() vrne spremenljivko tipa bool, katere vrednost je odvisna od uspešnosti izvedbe.

SQL proceduri `ValidateAdminAccount` pošljemo administratorsko ime in geslo, ki nato vrne rezultat validacije, ter v primeru pozitivnega odgovora tudi identifikacijsko številko administratorja.

Metoda `RegisterUserAccount`() vrne spremenljivko tipa bool, katere vrednost je odvisna od uspešnosti izvedbe.
SQL proceduri `RegisterUserAccount` pošljemo več parametrov v zvezi z uporabnikom in računalnikom. Namenjena je potrjevanju računa in registraciji določenega računalnika za nadzorovanega uporabnika.

Metoda `getBlockList`() vrne tabelo tipa string, ki preko poizvedbe v vrne imena procesov, ki jih je potrebno blokirati.

Metoda `whoAmI`() vrne spremenljivko tipa string, ki preko poizvedbe vrne priimek in ime uporabnika glede na podano uporabniško identifikacijsko številko.

#### 4.5.2.4 Razred Register

Omogoča pisanje - `Write`(), branje - `Read`() in brisanje - `DeleteKey`() ključev sistemskega registra.

Prav tako pa omogoča možnost nastavitve naše aplikacije, naj se samodejno zažene skupaj z operacijskim sistemom.

#### 4.5.2.5 Razred Stoparica

Omogoča merjenje časa. Deluje na principu preteklega časa od zagona sistema.
Vsebuje javne metode: `Zazeni()`, `Ustavi()`, `Pavziraj()`, `TrajanjeMs()`, `TrajanjeSec()`, `TrajanjeString()`, `TrenutnoTrajanjeMs()`, `TrenutnoTrajanjeSec()`, `TrenutnoTrajanjeString()` .

#### 4.5.2.6 Razred Beleženje procesov

Razred `BelezenjeProcesov` skrbi za zbiranje in posredovanje podatkov podatkovni bazi.

Glavna funkcionalnost razreda je metoda `preverjanje_Tick()`, ki se požene na določen interval.

Najprej kliče metodo `FunkcijeAPI.Okno.dobiAktivenProces()`, ki nam vrne instanco razreda `Process` . Nato preverimo, če je proces v listi blokiranih procesov, ki jo dobimo s klicem metode `FunkcijeSQL.getBlockList()` . V primeru da je, proces uničimo s klicem metode `Kill()`.

Nato ugotovimo, ali gre še vedno za enak proces kot prej, in glede na to izvajamo aktivnosti. Mednje spada merjenje trajanja procesa in beleženje aktivnosti tipkovnice za vsako pojavitev procesa posebej. Ko se proces v ospredju zamenja, se merjenje časa ustavi in zbrani podatki se prenesejo v podatkovno bazo preko metode `FunkcijeSQL.dodajDogodek`.

Za beleženje aktivnosti tipkovnice se uporablja instanca razreda `KeyboardHookListener`, kateremu dodamo poslušalca dogodkov (ang. event listener) `tipkovnica_KeyPress` za dogodek `KeyPress` , ki se sproži ob pritisku tipke. Ob dogodku iz argumentov vzamemo lastnost (ang. property) `e.KeyChar`, ki vsebuje referenco na pritisnjeno tipko, oziroma črko, katero predstavlja. Črko nato vstavimo na konec (ang. append) spremenljivke `keylogs` tipa `StringBuilder`, ki predstavlja tekstovni log vseh zapisov za trenutno pojavitev procesa in jo ob zamenjavi procesa v ospredju prenesemo z ostalimi podatki.

#### 4.5.2.7 Razred Client

Predstavlja našo aplikacijo za zajem podatkov.

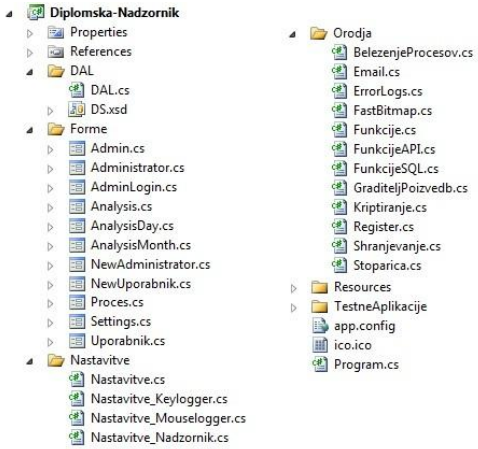
Aplikacijo najprej naredimo uporabniku nevidno, tako da jo naredimo prosojno in je ne prikazujemo v upravitelju opravil. Zagon aplikacije je samodejen ob zagonu sistema za kar smo poskrbeli s klicem metode `Register.setStartUpLaunch()`.

Ob poskusu nedovoljenega zaprtja aplikacije, katerega razlog ni `CloseReason.WindowsShutDown`, torej zaustavitev sistema, administratorja o tem takoj obvestimo preko spletne pošte in poskusimo prekiniti zaprtje.

V primeru, da nam ne uspe prekiniti zaprtja, imamo na voljo še eno rešitev. Poleg aplikacije Client smo implementirali še aplikacijo Backup, ki je prav tako skrita. Obe aplikaciji imata še dodatno nalogo, da poskrbita, da nobene od njiju ni mogoče zapreti. Aplikaciji medsebojno preverjata, ali je bila katera od njiju zaprta in pri pozitivnem odgovoru nemudoma ponovno zažene zaprto aplikacijo.

Za funkcionalnost sledenja procesom uporabimo instanco objekta `BelezenjeProcesov` , katerega delovanje smo že spoznali, ter `listenServer`, ki se uporablja pri prenašanju zaslonske slike uporabnika administratorju in ga bomo predstavili pri administratorski aplikaciji.

### 4.5.3 Projekt Diplomska-Nadzornik




**Slika 15: Datoteke v projektu Diplomska-Nadzornik**

Ta projekt predstavlja končno aplikacijo namenjeno administratorjem in je precej obsežen. Glavna funkcionalnost je predstavitev zajetih podatkov.

#### *4.5.3.1 Delo s podatki*

Delo s podatkovno bazo v administratorski aplikaciji poteka preko Dataseta ter direktnih poizvedb.

Dataset smo uporabljali že pri projektu Urnik. Tudi tukaj naredimo shemo tabel, ki se polnijo iz baze. Definirali smo njihove instance in vmesnike za njihovo polnjenje. Razred deluje na način, da uporabljamo javne nastavitve (ang. properties), preko katerih lahko enostavno dostopamo do njihovih instanc. Te instance lahko napolnimo s podatki iz baze oziroma osvežimo kadar to želimo. Če dostopamo do instance, ki še ni bila napolnjena, naš razred sam poskrbi, da se ta instanca napolni. V primeru, da podatke spreminjamo in želimo spremembe potrditi in shraniti nazaj v podatkovno bazo imamo tudi za vsako instanco posebej metodo, ki to naredi.

Pri direktnih poizvedbah pa v večini primerov najprej vzpostavimo povezavo s podatkovno bazo, ustvarimo novo poizvedbo s pomočjo razreda `SqlCommand` in preberemo odziv podatkovne baze s pomočjo metode `ExecuteReader()` in razreda `SqlDataReader`.

#### *4.5.3.2 Kriptiranje gesel*

V razredu `Kriptiranje` smo napisali enkripcijsko metodo `ROT13`, ki se uporablja za kriptiranje gesel. `ROT13` je preprost primer Cezarjeve šifre [15]. Deluje tako, da črko kodiramo z zamikanjem 13 mest v desno po abecedi, ki ima 26 znakov. Algoritem je torej enak za kodiranje in dekodiranje besedila.

#### *4.5.3.3 Beleženje napak*

Za beleženje napak (ang. exceptions) uporabljamo naš razred `ErrorLogs`. Podatke o napakah shranjujemo v tekstovno datoteko na disku.

#### *4.5.3.4 Obvestila preko elektronske pošte*

Za pošiljanje opozoril smo naredili razred `Email`, ki omogoča definiranje, generiranje in pošiljanje različnih vrst obvestil.

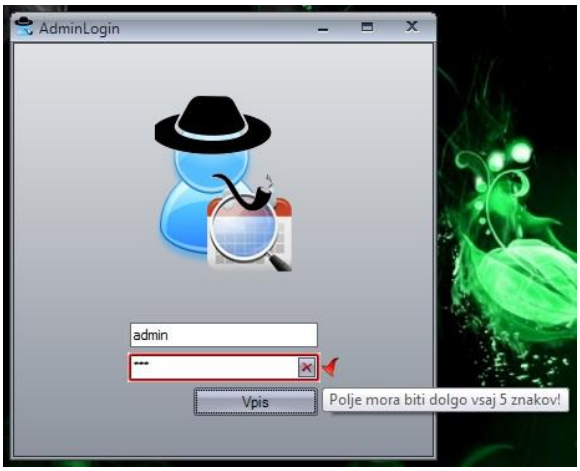
Vsaka vrsta obvestila ima svojo predlogo (ang. template), kamor se napolnijo podatki, ki jih pridobimo iz posredovane identifikacijske številke, katere pomen je odvisen od vrste obvestila. Možno je tudi nalaganje posnetka ekrana kot priloge.

Za izvajanje aktivnosti uporabljamo `BackgroundWorker` [16] kar pomeni, da se bo proces izvajal vzporedno v ločeni niti (ang. thread) in pri tem ne bo motil samega izvajanja aplikacije.

#### *4.5.3.5 Forme*

### AdminLogin

Forma je namenjena vpisu in preverjanju dostopa administratorja do glavne aplikacije.




**Slika 16: Forma AdminLogin**

Kot lahko vidimo na sliki 16, forma vsebuje dve vnosni polji za administratorski račun in geslo. Za preverjanje teh polj se uporabljajo regularni izrazi, za javljanje napak pa poskrbimo preko razreda `DevComponents.DotNetBar.Validator.SuperValidator`, ki s pomočjo razredov `ErrorProvider` in `Highlighter` predstavi lokacijo in kontekst same napake.

Ob zagonu forme se iz registra preberejo nastavitve oblikovanja, kot sta na primer `STYLE` in `COLORTINT`. Prvega bomo spoznali kasneje pri nastavitvah oblikovanja, `COLORTINT` pa predstavlja barvo, s katero bomo prepojili (ang. tint) sliko ozadja in samo barvno temo aplikacije. Obdelava ozadja poteka s pomočjo metode `Orodja.FastBitmap.TintImage()` in prav tako poteka v ločeni niti.

Po kliku gumba za vpis se v primeru uspešnega preverjanja polj kliče metoda `FunkcijeSQL.ValidateAccount()`, ki nam v primeru uspešne avtorizacije računa vrne status ter identifikacijsko številko tega računa. Nato naredimo instanco glavne forme, ki v primeru, da nalaganje podatkov in procesiranja ozadja še vedno poteka, počaka na dokončanje in se nato odpre.

## Admin

Administratorska forma predstavlja osrednji del projekta Diplomska-Nadzornik.

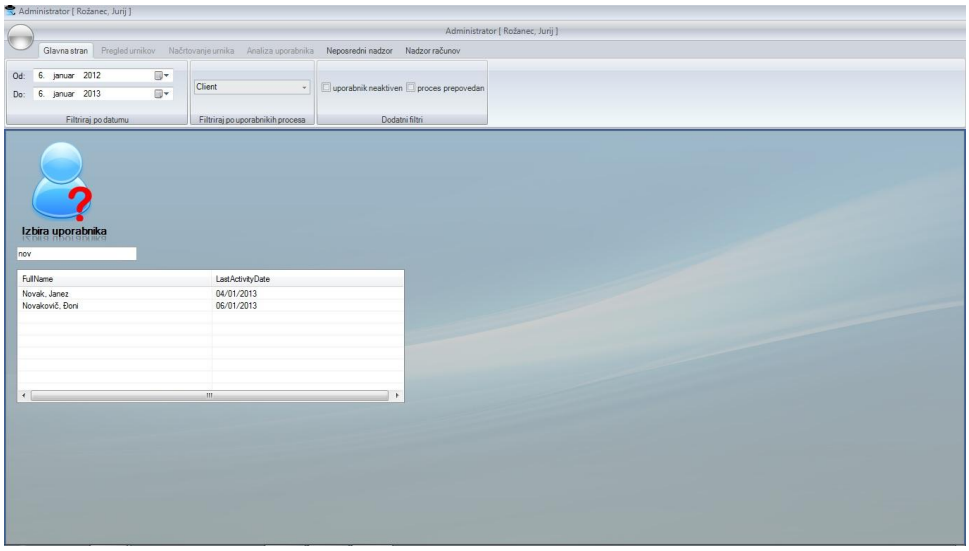
Za obogatitev grafičnega vmesnika, smo uporabili zbirko komponent `DotNetBar` [17]. Tako smo na primer za navigacijo po funkcionalnostih forme uporabili kontrolo `RibbonControl`, ki omogoča podoben izgled in način delovanja kot večina Microsoftovih aplikacij. Torej poleg načina delovanja zavihkov nam omogoča uporabo `RibbonPanel`, katerega smo uporabili za postavitev raznih možnosti manipulacije s podatki, ki so tematsko povezane s trenutno odprtim zavihkom.

Forma je tematsko razdeljena je na več zavihkov:

- glavna stran,
- pregled urnikov,
- načrtovanje urnika,
- analiza uporabnika,
- neposredni nadzor in
- nadzor računov.

## Glavna stran

Funkcionalnost glavne strani je predvsem izbira uporabnika, katerega podatke želimo pregledovati. Izbira poteka preko kontrole `ListView`, ki vsebuje seznam uporabnikov z datumom njihove zadnje aktivnosti, kot lahko vidimo na sliki 17.




**Slika 17: Zavihek Glavna stran**

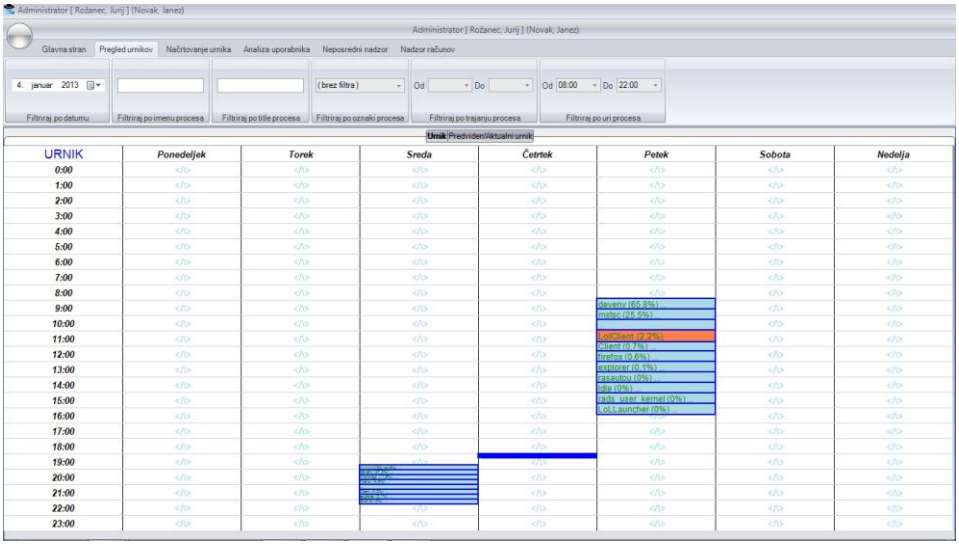
Filtri te strani obsegajo:

- delno tekstovno iskanje (ang. wild card) po imenu in priimku,
- določanje datumskega razpona,
- iskanje uporabnikov, ki so uporabljali določeno aplikacijo,
- iskanje po aktivnih / neaktivnih uporabnikih,
- uporabnikih, ki so uporabljali prepovedane aplikacije.

Ob izbiri uporabnika se omogoči dostop do aktivnosti in možnost menjave izbranega uporabnika, prav tako pa nam jih v grafični obliki ponudi sam zavihek.

### Pregled urnikov

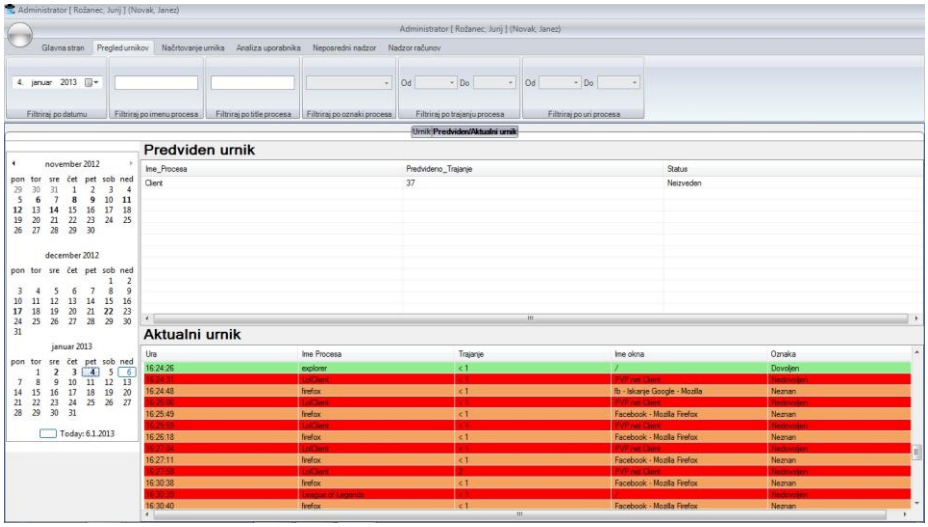
Zavihek Pregled urnikov je eden pomembnejših delov aplikacije in nam omogoča predvsem pregled nad aktivnostjo uporabnika po posameznih tednih z uporabo naše kontrole Urnik, kot lahko vidimo na sliki 18.




**Slika 18: Zavihek Pregled urnikov - Vizualna predstavitev aktivnosti tedna**

Urnik je nastavljen tako, da prikazuje aktivnost celega tedna naenkrat, vidimo lahko razpon aktivnosti za posamezen dan, ki so prikazane v deležu skupne dnevne aktivnosti. Kontrola nam omogoča, da s klikom na določen proces odpremo kartico analize, ki vsebuje bolj podrobno analizo izbranega procesa v tistem dnevu in jo bomo spoznali kasneje med opisom posameznih kartic. Ob kliku na polje, kjer se nahaja ime dneva, pa nam kontrola odpre kartico dnevne analize, ki predstavlja bolj podrobno analizo celotnega dneva, ki jo bomo prav tako spoznali kasneje v razdelku opisov kartic.

Ob menjavi notranjega zavihka, kar je prikazano na sliki 19 se nam pregled na urnikom spremeni v analogen prikaz, kjer lahko brskamo po posameznih dneh ter primerjamo izvedbo predvidenih/načrtovanih procesov proti dejanskim. Tukaj ob kliku na določen proces odpremo kartico Proces, ki bo predstavljena kasneje. Za lažji pregled imamo na voljo koledar, kjer so dnevi, ki vsebujejo zapise o aktivnostih, odebeljeni.



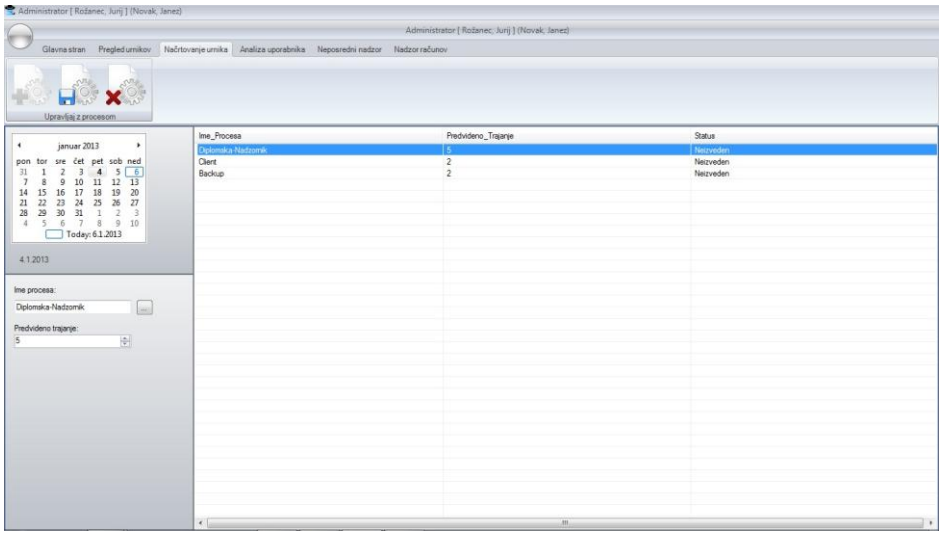

**Slika 19: Zavihek Pregled urnikov - analogen prikaz v tabeli**

Oba načina prikaza sta sinhronizirana in tudi uporabljata enake možnosti filtriranja:

- izbira datuma/tedna,
- po imenu procesa,
- po naslovu procesa,
- po oznaki procesa,
- po trajanju procesa in
- po uri procesa.

## Načrtovanje urnika

Zavihek je namenjen načrtovanju procesov, ki naj bi jih nadzorovani uporabnik moral uporabljati določen dan. Načrtujemo lahko katerikoli kdaj zabeležen proces s predvidenim časom trajanja.



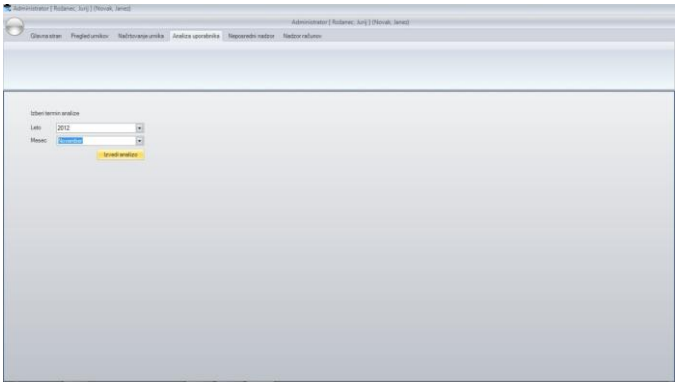

**Slika 20: Zavihek Načrtovanje urnika**

Kot je vidno na sliki 20, nam je omogočeno pregledovanje, dodajanje, urejanje in brisanje teh zapisov. Po uspešnem dodajanju jih lahko vidimo tudi kot odebeljene dneve na koledarju. Pri pregledu lahko vidimo tudi njihov Status izvedbe.

Primerjava med aktualnim urnikom in načrtovanimi procesi pa se pojavi tudi na zavihku Pregled urnikov.

### Analiza uporabnika

Zavihek Analiza uporabnika sicer sam po sebi nima nobene funkcionalnosti, ampak je namenjen odpiranju kartice AnalysisMonth, torej mesečne analize uporabnika, ki bo predstavljena v nadaljevanju.

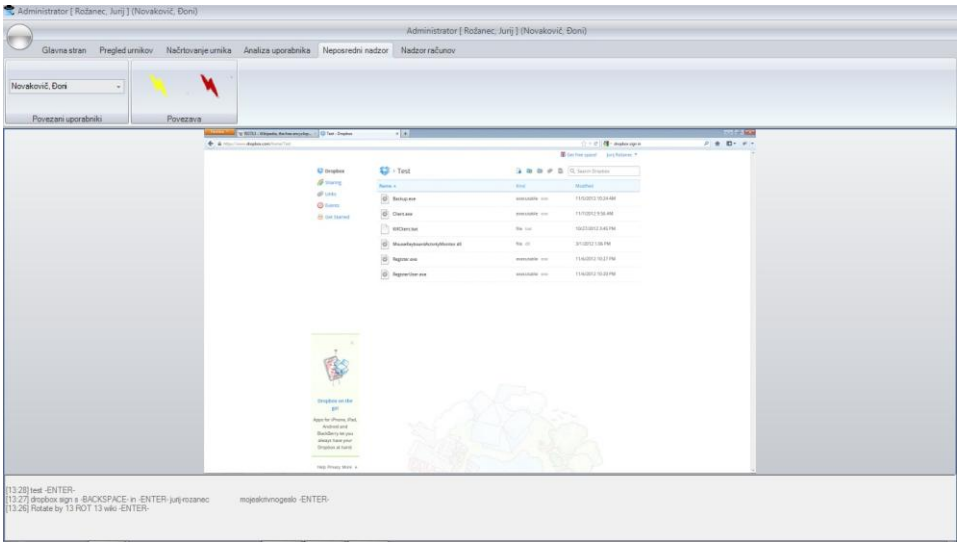


**Slika 21: Zavihek Analiza uporabnika**

Na sliki 21 lahko vidimo da nam omogoča izbiro termina analize, to je izbira leta in meseca.

### Neposredni nadzor

Zavihek Neposredni nadzor omogoča pregled vseh trenutno povezanih uporabnikov in možnost neposrednega povezovanja na njihov odjemalec, ki omogoča posredovanje slike zaslona uporabnika v realnem času, prav tako pa tudi sledenje aktivnosti njegove tipkovnice, kar je razvidno na sliki 22.




**Slika 22: Zavihek Neposredni nadzor - prikaz posredovanja slike in aktivnosti tipkovnice**

Naloga je bila implementirana v obliki razredov `Server` in `listenServer`, ki skupaj skrbita za nemoteno povezavo in delovanje.

Razred `Server` se izvaja na strani administratorske aplikacije.
Najprej pride na vrsto povezovanje z uporabnikovim odjemalcem. V instanci razreda se izvaja metoda `ListenForClients`(), ki sprejema vhodne povezave uporabnika na vratih (ang. port) številka 8080. Povezanega odjemalca predstavimo kot instanco razreda `TcpClient`, ki jo dodamo v seznam povezanih uporabnikov `listClient` tipa `HashTable`. Ob vsaki vzpostavitvi povezave kličemo metodo `HandleClientComm`, ki se izvaja v svoji niti.

V tej metodi najprej naredimo instanco objekta `NetworkStream`, ki služi branju toka podatkov. Prve dva prejeta bajta podatkov vsebujeta posebno številko, ki služi kot tip obvestila, kar nam pove kako procesiramo preostali del podatkov.

Glavni tipi obvestil :

- odjemalec se povezuje - 0,
- odjemalec se zapira - 1,
- odjemalec pošilja tekstovno sporočilo - 2,
- odjemalec pošilja slikovno sporočilo - 3,
- odjemalec sporoča, da je še dosegljiv - 9.

V primeru prejetih podatkov oziroma na tip obvestila, podatkovne bajte pretvorimo v ustrezen format. V primeru tekstovnega sporočila, jih s pomočjo razreda `ASCIIEncoding`, pretvorimo nazaj v obliko `String`. V nasprotnem primeru, da gre za slikovno sporočilo, pa poskusimo podatke pretvoriti v sliko formata `Bitmap`.

Na določen interval preverjamo, ali so odjemalci na našem seznamu še dosegljivi - `pokeToAllClients`(). Za ta namen za vsak zapis v našem seznamu uporabimo metodo `pokeToClient`(), ki pošlje na odjemalec poskusno sporočilo. V primeru, da pride pri prenosu sporočila do napake v povezavi, odjemalec označimo kot nedosegljiv in ga odstranimo z liste.

Vsebuje tri dogodke:

- `OnPicRecieved` – predstavlja uspešno prejeto sliko
  Dogodek se sproži, ko prejmemo tip obvestila za prejem slikovnega sporočila. Uporablja se za posodabljanje prikaza zaslonske slike v administratorski aplikaciji, istočasno pa osveži tekst aktivnosti tipkovnice.
- `OnNewClientConnected` – predstavlja povezavo novega odjemalca
  Dogodek se sproži, ko med sprejemanjem vhodnih povezav naletimo na odjemalec, ki ga ni na našem seznamu.
  Uporablja se za osveževanje našega seznama povezanih uporabnikov v administratorski aplikaciji.
- `OnClientDisconnected` – predstavlja prekinitev povezave z odjemalcem
  Dogodek se sproži, ko odjemalec med preverjanjem ni dosegljiv. Prav tako je namenjen osveževanju seznama v administratorski aplikaciji.

Pri postopku identifikacije, kateremu uporabniku pripada odjemalec, se zanašamo na naše podatke iz baze. Najbolj preprost način, brez dodatne poizvedbe nad odjemalcem je uporaba IP naslova, preko katerega že komuniciramo z njim. Z uporabo te informacije enostavno ugotovimo, kateri računalnik je v uporabi. Nato samo še med dogodki preverimo, kateri uporabnik ga je nazadnje uporabljal.

Razred `listenServer` se torej izvaja na uporabnikovem odjemalcu.
Komunikacija in povezovanje deluje na podoben način. Ker bi se lahko zgodilo, da sta Server in odjemalec na istem računalniku, tukaj nastavimo poslušalca na vrata 8081, da ne pride do konfliktov.

Tipa obvestil:

- začni prenos slik in
- končaj prenos slik.

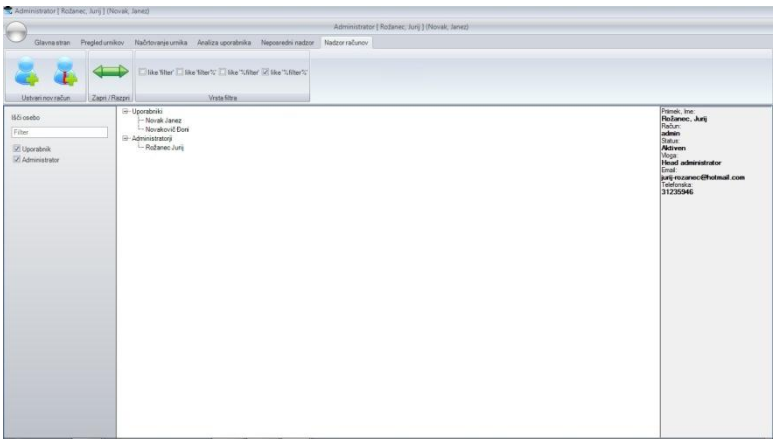
Odjemalec se najprej poveže na Server in nato poleg njegove osnovne funkcije ne dela ničesar, dokler ne dobi sporočila za začetek prenosa slik. V tem primeru ustvarimo novo nit, ki se izvaja dokler je ne prekinemo s sporočilom za konec prenosa. V tej zanki zajamemo sliko ekrana, pretvorimo v podatkovne bajte, ter jo nato z uporabo razreda `NetworkStream` in `TcpClient` pošljemo Serverju preko protokola TCP v manjših paketih poljubne velikosti. Nato naredimo enako še z besedilom, ki pripada aktivnosti tipkovnice. Da se izognemo napakam pri prenosu posameznih črk, raje pošiljamo kar tekstovni log za določeno časovno enoto naenkrat. Če odjemalec izgubi povezavo, se poskusi na določen interval ponovno povezati.

### Nadzor računov

Zavihek nam omogoča nadzor nad uporabniškimi in administratorskimi računi.

Ustvarjanje novega računa osebe odpre kartico NewUporabnik oziroma NewAdministrator, kjer se nahaja omenjena funkcionalnost.

Na sliki 23 lahko vidimo prikaz računov, kjer uporabljamo drevesno strukturo s pomočjo kontrole `AdvTree`. Korenska elementa drevesa sta elementa 'Uporabniki' in 'Administratorji'. Izbira določene osebe iz drevesa odpre kartico Uporabnik, oziroma kartico Administrator.



**Slika 23: Zavihek Nadzor računov**

Posamezni elementi znotraj drevesa upoštevajo filtre:

- delno tekstovno iskanje (ang. wild card) z izbiro vrste filtra po imenu in priimku,
- samo uporabniki,
- samo administratorji.

#### *4.5.3.6 Kartice*

Besedna oznaka kartica v naši aplikaciji označuje instanco objekta `Form`, torej novo okno. V Administratorski aplikaciji kartice uporabljamo za prikaz bolj specifičnih vsebin, katere imajo skupno vrsto informacij, ampak lahko vsebujejo različne vhodne podatke za procesiranje.

Za lažjo predstavitev - primer kartice je kartica Proces, ki vsebuje polje z informacijo o statusu procesa. Vsaka kartica tipa Proces ima to polje, njegova vrednost pa je odvisna od podane identifikacijske številke procesa – `Proces_ID` v konstruktorju kartice.

Način delovanja kartic pride prav pri oblikovanju zgodovine uporabe aplikacije – funkcionalnost seznama Zadnjih odprtih kartic. Torej shranjujemo seznam nazadnje odprtih kartic, ki nam nato omogoča preprosto ponovno odpiranje le-teh, pri čemur potrebujemo le oznako vrste kartice in podane vhodne parametre.

Ob odprtju vsake kartice najprej prilagodimo stil in barvo forme. Te nastavitve se že nahajajo shranjene v glavnem razredu `Program` kot spremenljivke, ki so tako dostopne povsod po aplikaciji. Nato ustrezno prilagodimo velikost in stanje. Kot smo že omenili se preko konstruktorja forme prenesejo argumenti, ki se nato uporabijo pri nadaljnjem procesiranju.

**Delovanje seznama zadnjih odprtih kartic**

Na glavni formi imamo svojo metodo za osveževanje podatkov, ki se kliče, kadar iz kartice želimo izvesti akcijo v glavni formi – večinoma sinhronizacijo. Referenca na glavno formo obstaja v razredu Program, ki nam omogoča, da lahko metodo kličemo iz kartic. Za funkcionalnost dodajanja na seznam se uporablja ključna beseda `AddToRecents`, kjer poleg imena kartice lahko dodajamo parametre. Primer takega klica je pri kartici mesečne analize :

```
Program.adminForm.Reload(String.Format("AddToRecents:AnalysisMonth#{0}_{1}_{2}", UID, Year, Month));
```

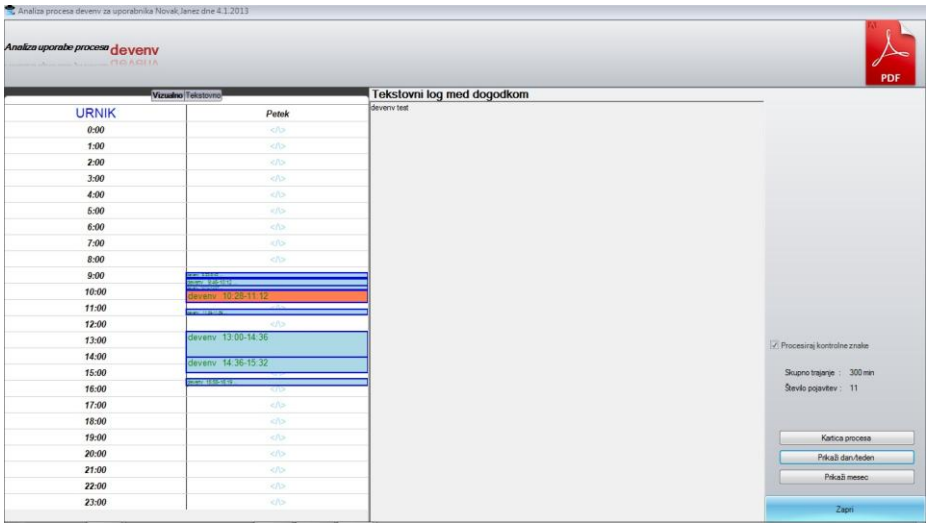
Ti podatki se shranijo v objekt tipa `Hashtable` razreda `Nastavitve_Nadzornik`, preko katerega skrbimo za pravilen vrstni red seznama in da se ti podatki ustrezno shranijo na disk ob zaprtju aplikacije v obliki tekstovne datoteke, katere struktura je podobna XML.

Metoda `readSettings()` se uporablja za prikaz imen kartic v obliki posebnih gumbov `ButtonItem` v kontroli `GalleryContainer`. Gumbe se ustvarimo in dodamo na generičen način. Takrat poleg nastavljanja teksta gumba in poslušalca za klik na gumb, v polje opisa

zapišemo parametre kartice. Iz podanih parametrov ob kliku ugotovimo katero kartico in s kakšnimi vhodnimi parametri je potrebno odpreti.

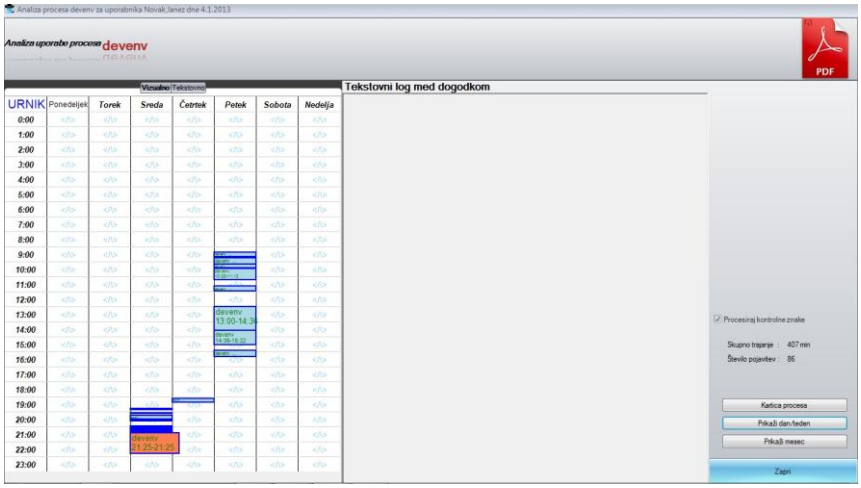
### Kartica Analysis

Kartica nam omogoča bolj podrobno analizo izbranega procesa določenega dne oziroma tedna. Za razliko od uporabe kontrole Urnik v zavihku Pregled urnika, ki prikazuje delež skupne dnevne aktivnosti, tukaj prikazujemo dejanski urnik, glede na posamezne pojavitve procesa, kot lahko vidimo na sliki 24.




**Slika 24: Kartica Analysis - prikaz dne**

Na enak način je omogočen tudi prikaz celotnega tedna, ki ga prikazuje slika 25.



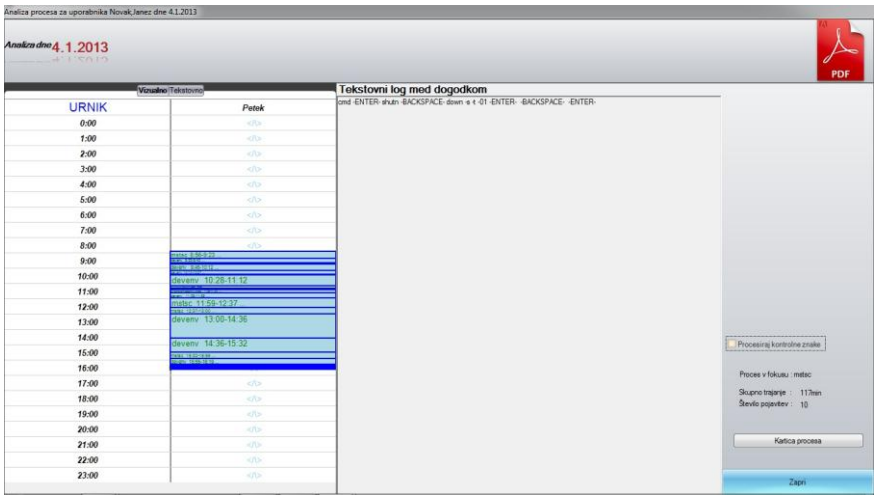

**Slika 25: Kartica Analysis - prikaz tedna**

Kartica prav tako omogoča tekstoven prikaz urnika v tabeli.

Ob kliku na posamezno pojavitev procesa se nam prikaže tekstovni log, ki je bil zabeležen med trajanjem te pojavitve. Tekstovni log lahko prikaže v dveh načinih. Prvi procesira kontrole znake, kot je na primer Backspace. V tem primeru se iz loga odstrani zadnji vpisan znak. V drugem primeru pa tega kontrolnega znaka ne procesiramo, kar pomeni, da ga zapišemo kot niz znakov ' - BACKSPACE - '.

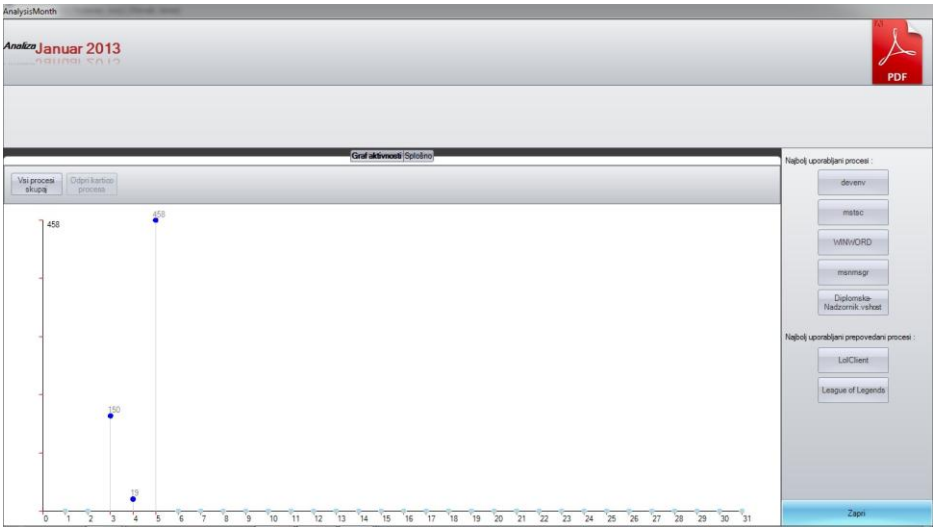
### Kartica AnalysisDay

Na sliki 26 lahko vidimo, da je kartica je zelo podobna prejšnji. Omogoča nam pregled določenega izbranega dne, z vsemi dejanskimi pojavitvami dogodkov v kontroli Urnik oziroma tekstovni predstavitvi v tabeli.




**Slika 26: Kartica AnalysisDay**

### Kartica AnalysisMonth



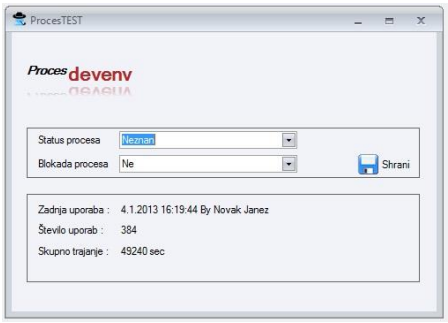

**Slika 27: Kartica AnalysisMonth**

Kartica, ki jo vidimo na sliki 27 je namenjena predstavitvi skupne aktivnosti uporabnika, za vsak dan celotnega meseca. Za namen vizualizacije uporabljamo kontrolo Histogram. Kartica nam nudi seznam petih najbolj uporabljanih procesov in petih najbolj uporabljanih prepovedanih procesov. S klikom na izbranega, filtriramo vizualizacijo aktivnosti v kontroli Histogram, da prikaže samo podatke v zvezi z izbranim procesom.

Vse tri kartice za podrobnejšo analizo pa omogočajo tudi izvoz podatkov kartice v PDF format.

### Kartica Proces

Kartica Proces nam omogoča enostavno urejanje statusa in blokade določenega procesa. Kot vidimo na sliki 28, imamo pregled, kdaj in kdo je proces nazadnje uporabljal, pogostost uporabe, in skupno rabo.

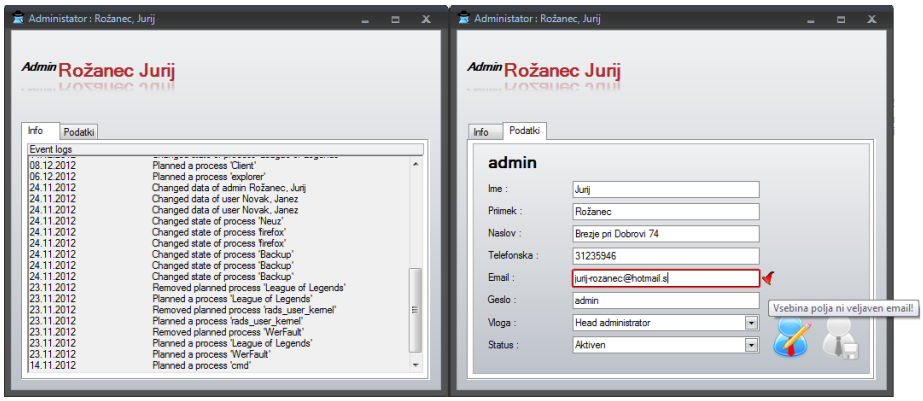



**Slika 28: Kartica Proces**

### Kartici Uporabnik in Administrator

Kot vidimo na sliki 29 sta kartici namenjeni nadzoru uporabniških in administratorskih računov. Obe omogočata pregled, urejanje in shranjevanje podatkov v povezavi z osebo in njenim računom.

Kartica Administrator omogoča tudi pregled nad celotno zgodovino dejavnosti določenega administratorja.




**Slika 29: Kartica Administrator**

### Kartici NewUporabnik in NewAdministrator

Funkcionalnost teh kartic je predvsem registracija, torej vpis podatkov novih uporabnikov in administratorjev v sistem.

### Kartica Settings

Kartica Settings omogoča personalizacijo uporabniškega vmesnika. Na izbiro imamo več različnih tem (ang. Themes), katerih osnovno barvo lahko zamenjamo s poljubno. Nato primerno procesiramo še ozadje glavne strani, kar poteka s pomočjo metode `Orodja.FastBitmap.TintImage()`.

Te nastavitve se shranjujejo v sistemski register, torej veljajo samo na trenutnem računalniku.

# 5 Analiza aplikacije

## 5.1 Primerjava s podobnimi aplikacijami

| | Aplikacija Nadzornik | Aplikacija All In One Keylogger | Aplikacija ManicTime |
|---|---|---|---|
| Beleženje uporabe aplikacij | ✓ | ✓ | ✓ |
| Beleženje aktivnosti tipkovnice | ✓ | ✓ | ✗ |
| Beleženje obiskanih spletnih strani | ● | ✗ | ✓ |
| Beleženje odsotnosti od računalnika | ● | ✗ | ✓ |
| Beleženje uporabe zunanjih naprav | ✗ | ✓ | ✗ |
| Enolična identifikacija aplikacije | ✓ | ✗ | ✓ |
| Shranjevanje besedila okna procesa | ✓ | ✓ | ✓ |
| Predstavitev podatkov v tabeli | ✓ | ✓ | ✓ |
| Vizualna predstavitev podatkov | ✓ | ✗ | ✓ |
| Možnost podrobnejšega filtriranja | ✓ | ✗ | ✓ |
| Izvajanje analiz | ✓ | ✗ | ✓ |
| Takojšnja dostopnost podatkov | ✓ | ✗ | ✓ |
| Možnost ekranskih posnetkov | ✓ | ✓ | ✗ |
| Neposreden pogled na uporabnika | ✓ | ✗ | ✗ |
| Blokiranje aplikacij | ✓ | ✗ | ✗ |
| Označevanje aplikacij | ✓ | ✗ | ✓ |
| Pošiljanje obvestil | ✓ | ✓ | ✗ |
| Izvoz podatkov v drug format | ✓ | ✓ | ✓ |
| Možnost nadzorovanja več uporabnikov | ✓ | ✓ | ✗ |
| Avtorizacija dostopa do aplikacije | ✓ | ✗ | ✗ |
| Prikrito delovanje | ✓ | ✓ | ✗ |
| Uporaba podatkovne baze | ✓ | ✗ | ✓ |
| Preprečevanje zaprtja | ✓ | ✗ | ✗ |
| Upravljanje z uporabniškimi računi | ✓ | ✗ | ✗ |

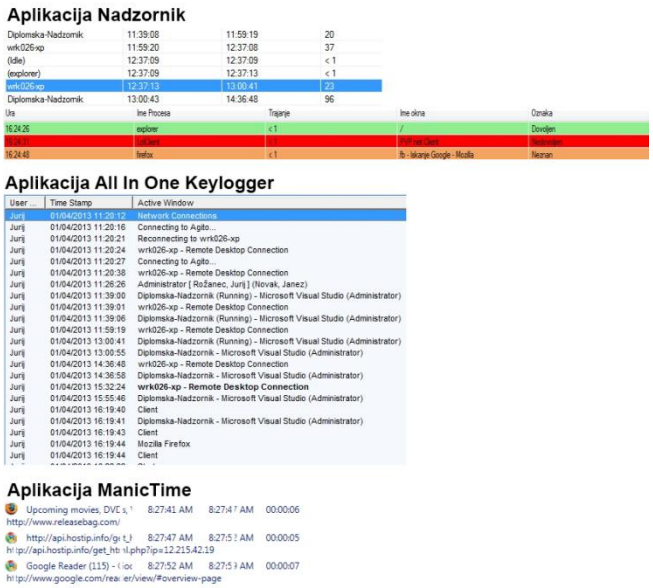
Legenda:
✓ vsebuje
✗ ne vsebuje
● načrtovano za implementacijo v naslednjih verzijah

**Slika 30: Primerjava s podobnimi aplikacijami**

Na sliki 30 vidimo primerjavo funkcionalnosti in lastnosti naše aplikacije z All In One Keylogger ter ManicTime, ki sta bili predstavljeni v podpoglavjih 3.1 in 3.2.

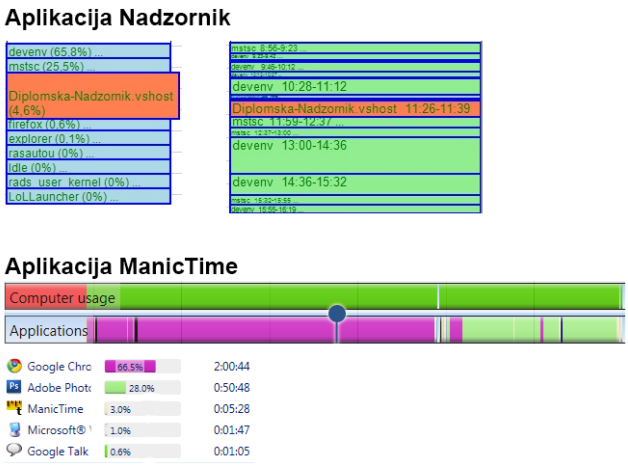
Prva verzija aplikacije Nadzornik ponuja večino funkcionalnosti in možnosti načinov beleženja različnih vrst podatkov, ki jih ponujata ostali aplikaciji. Naši aplikaciji na primer manjkata beleženje uporabe mikrofona in tiskalnika, ki pa v našem kontekstu nista bistvenega pomena.

Vse aplikacije ponujajo predstavitev podatkov v tabeli, njihovo primerjavo lahko vidimo na sliki 31.




**Slika 31: Primerjava predstavitev podatkov v tabeli**

Vizualno predstavitev, ki jo vidimo na sliki 32, ponujata le aplikacija Nadzornik in ManicTime, vendar slednja ne ponuja možnosti pregleda celotnega tedna.




**Slika 32: Primerjava vizualne predstavitve**

Aplikacije so po funkcionalnosti beleženja dokaj enakovredne glede na njihov cilj. Večja razlika se pojavi v prikazu podatkov in možnostmi manipulacije z njimi, v čemur sta aplikaciji Nadzornik in ManicTime izboljšani v primerjavi z aplikacijo All In One Keylogger. Največja prednost Nadzornika je prenos podatkov v realnem času, kljub temu, da se podatki zbirajo na drugem računalniku. Tako so podatki za analizo dostopni takoj.

# 6 Sklepne ugotovitve

V okviru diplomske naloge smo izdelali aplikacijo Nadzornik, katere osnovna funkcionalnost je nadzorovanje in zbiranje podatkov o uporabniški aktivnosti s pomočjo odprtih aplikacij ter njihova predstavitev v različnih oblikah. Osnovno funkcionalnost smo nato razširili na zaokroženo celoto, ki nam omogoča še beleženje aktivnosti tipkovnice med samim izvajanjem procesa in preprečevanje zagona določenih procesov. Glede nadzora smo dodali takojšnje posredovanje slikovnega dokaza o uporabi prepovedane aplikacije administratorju preko elektronske pošte in možnost, da se administrator neposredno poveže na uporabnikov odjemalec in spremlja sliko njegovega zaslona ter aktivnost tipkovnice. Prav tako smo obogatili predstavitev zbranih podatkov na način, da smo izdelali lastni kontroli, ki prikazujeta podatke na različne načine.

Aplikacijo bi lahko nadgradili z manjši deli funkcionalnosti, ki pa so zaradi pomanjkanja časa delno implementirani in še niso vključeni v aplikacijo. Primera takšnih funkcionalnosti sta beleženje odsotnosti in beleženje obiskanih spletnih strani, ki pa sta skupaj z odpravo manjših napak v aplikaciji že načrtovani za naslednjo verzijo aplikacije.

Med razvojem aplikacije smo pridobili veliko novih izkušenj in znanj na področjih načrtovanja in programiranja. V aplikaciji, predvsem pri kodiranju, se pozna pozitiven in negativen vpliv sočasnega dela kot razvijalca ASP.NET spletnih aplikacij. Pozitiven vpliv je opaziti predvsem pri načinu kodiranja, poimenovanju struktur ter preventivnemu izogibanju napakam, kar je vodilo tudi do hitrejšega razvoja. Negativen vpliv se pozna pri razliki med začetnim in končnim stilom programiranja, torej način kodiranja ni povsod skladen. Predvsem pa so izkušnje pripevale k zmanjšanju količine časa, ki smo ga lahko namenili za razvoj diplomske naloge.

Cilj diplomske naloge, to je razvoj in predstavitev aplikacije za spremljanje uporabnikove aktivnosti na osebnem računalniku je bil torej dosežen. S tehničnega vidika smo raziskali, kakšne vrste informacij lahko beležimo, na kakšen način jih lahko predstavimo ter večji del tudi implementirali v aplikaciji. Glede pravnih vidikov uporabe tovrstnih opcij bi se bilo potrebno posvetovati s pravniki.

# Literatura in viri

[1] (2013) Cisco¸ »Data Leakage Worldwide: Common Risks and Mistakes Employees Make«. Dostopno na : http://www.cisco.com/en/US/solutions/collateral/ns170/ns896/ns895/white_paper_c11-499060.pdf.

[2] (2013) Wikipedia, »Microsoft Visual Studio«. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio.

[3] (2013) Wikipedia, ».NET Framework«. Dostopno na: http://en.wikipedia.org/wiki/.net_framework.

[4] (2013) Wikipedia, »IntelliSense«. Dostopno na: http://en.wikipedia.org/wiki/IntelliSense.

[5] Bruce Johnson, Mike Snell, Shawn Wildermuth, »Designing and Developing Windows-Based Applications using Microsoft .NET FRAMEWORK«, Washington: Microsoft Press, 2007, poglavje 3.

[6] (2013) Wikipedia, »Microsoft Visual SourceSafe«. Dostopno na: http://en.wikipedia.org/wiki/Visual_SourceSafe.

[7] (2013) Wikipedia, »Team Foundation Server". Dostopno na: http://en.wikipedia.org/wiki/Team_Foundation_Server.

[8] (2013) Wikipedia, »Microsoft SQL Server«. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_SQL_Server.

[9] (2013) Wikipedia, »Transact-SQL«. Dostopno na: http://en.wikipedia.org/wiki/Transact-SQL.

[10] (2013) Wikipedia, »SQL Server Management Studio«. Dostopno na: http://en.wikipedia.org/wiki/SQL_Server_Management_Studio.

[11] (2013) Wikipedia, »PowerDesigner«. Dostopno na: http://en.wikipedia.org/wiki/PowerDesigner.

[12] (2013) RelyTec, »All In One Keylogger«. Dostopno na: http://www.relytec.com/.

[13] (2013) ManicTime, »ManicTime«. Dostopno na: http://www.manictime.com/.

[14] (2013) MSDN, »DataSet Class«. Dostopno na: http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx.

[15] (2013) Wikipedia, »Cezarjeva šifra«. Dostopno na: http://sl.wikipedia.org/wiki/Cezarjeva_šifra.

[16] (2013) MSDN, »BackgroundWorker Class«. Dostopno na: http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx.

[17] (2013) DevComponents, »DotNetBar«. Dostopno na: http://www.devcomponents.com/dotnetbar/.