Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Uroš Knapič

Združitev avtomatiziranih procesov v podjetju

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Ko neka programska rešitev, ki uporablja različne programske procese in vrši opravila nad podatkovno bazo, odkrivanje in reševanje napak postane zelo časovno zahtevno. Preučite možnosti za avtomatizacijo nadzora nad temi procesi in predlagajte primerna orodja za tovrstni nadzor. Rešitev ovrednotite na konkretnem primeru znotraj podjetja.

Za pomoč pri izdelavi diplomskega dela se zahvaljujem mentorju doc. dr. Luki Šajnu in sodelavcem podjetja Ceneje d. o. o.

Kazalo

Seznam uporabljenih kratic

HTML	hypertext markup language	jezik za označevanje nadbesedila
XML	eXtensible Markup Language	razširljiv označevalni jezik
CSS	cascading style sheets	predloge, ki določajo videz spletnih strani
URL	Uniform Resource Locator	enolični krajevnik vira
SQL	structured query language	strukturiran povpraševalni jezik
MVC	Model–view–controller	arhitektura MVC
JSON	JavaScript Object Notation	oblika zapisa označevanja objektov JavaScript
IP	Internet Protocol	internetni protokol

Slike

Seznam algoritmov

Povzetek

Naslov: Združitev avtomatiziranih procesov v podjetju

Avtor: Uroš Knapič

Namen diplomskega dela je opis in razvoj programa za nadzor avtomatiziranih procesov v podtjetju. V podjetju se soočajo z veliko količino procesov na različnih mestih. Proces v podjetju predstavlja avtomatizirano nalogo, ki podatke osvežuje ali spreminja. Cilj je rešiti problem velikega nenadzorovanega števila procesov z zbiranjem informacij o procesih na enem mestu. Implementirati je treba program, ki bo zbrane procese zaganjal in ustrezno zapisoval. Diplomsko delo vsebuje tehnično vsebino z odseki izvorne programske kode. Na začetku je treba izbrati ustrezne tehnologije za implementacijo rešitve, ki je implementirana v okolju .NET Core. To omogoča delovanje programa na operacijskem sistemu Linux. Za hranjenje procesov uporablja podatkovno bazo Microsoft SQL Server, za beleženje zgodovine procesov pa dokumentno bazo MongoDB. Glavni program in spletno aplikacijo se implementira v razvojnem okolju Visual Studio s programskim jezikom C# ter spletnimi tehnologijami HTML in CSS. Implementacija glavnega programa omogoča zaganjanje procesov na enem mestu. Pri zaganjanju procesov se upošteva format cron [1]. Spletna aplikacija omogoča enostaven pregled in urejanje procesov. Opravljeno je bilo testiranje rešitve in prikazana učinkovitost končne rešitve.

Ključne besede: spletna aplikacija, C#, .NET Core, HTML, CSS, cron, crontab, MongoDB, SQL, Microsoft SQL Server.

Abstract

Title: Unifying automated processes in the company

Author: Uroš Knapič

The purpose of the thesis is to describe and develop a program to supervise automated processes in a company. They deal with a vast number of processes at different places. The process in the company represents an automated task which updates or changes the process. Goal is to solve the problem with collecting data at single place. We need to develop the program which runs processes and saves any information about it. Thesis contains technical content with sections of source code. First of all, we need to choose the right technologies to implement our solution. The solution is implemented in .NET Core, which allows the program to run on Linux. The Microsoft SQL Server will be used to store process information and MongoDB to record the history. The main program and the web application are developed using Visual Studio with the C# programming language and web technologies, such as HTML and CSS. The final solution allows the processes to run from one place. However, the web application provides a simple overview and allows the users to edit the processes easily. We also tested the final solution and demonstrated its effectiveness.

Keywords: web application, C#, .NET Core, HTML, CSS, cron, crontab, MongoDB, SQL, Microsoft SQL Server.

Poglavje 1

Uvod

1.1 Motivacija in cilji

Projekt je nastal na pobudo sodelavcev zaradi večanja števila avtomatiziranih procesov. Proces predstavlja shranjeno poizvedbo SQL ali program, ki osvežuje ali posodablja pomembne podatke. Težava je nastopila pri zamudnem sledenju in odpravljanju napak, saj ni bilo razvidno, kje se proces nahaja in ali pravilno deluje. Namen projekta je združitev procesov na enem mestu. Končni cilj je zmanjšati čas iskanja in najti način za lažje odpravljanje napak.

1.2 Zgradba diplomskega dela

V drugem poglavju je predstavljena izbira tehnologij za izvedbo projekta, njihov opis in razloge za izbiro. Na koncu poglavja je predstavljen diagram poteka glavnega programa, ki je osnova za razvoj programa. Iz diagrama je možno razbrati glavne naloge, ki jih je treba implementirati v program. V tretjem poglavju je predstavljena tehnična implementacija glavnega programa. Pogledali si bomo združitev in namen vseh izbranih tehnologij v končni rešitvi. V četrtem poglavju je predstavljena spletna aplikacija za pregled procesov, urejanje procesov in pregled napak, ki so povzročile, da proces ni deloval.

Poglavje 2

Zahteve projekta

V tem poglavju se bomo osredotočili na načrtovanje projekta. Pogledali si bomo izbiro tehnologij in razloge za to.

2.1 Izbira tehnologij

Arhitektura sistema v podjetju nam je omogočila pestro izbiro tehnologij. Pred izbiro smo morali odgovoriti na naslednja vprašanja:

1. Na katerem operacijskem sistemu se bo izvajal program?
2. Kako pogosto in na kakšen način bomo zaganjali program?
3. Kje bodo shranjeni procesi?
4. Kam bomo zapisovali stanja procesov?

Projekt zahteva nemoteno in neprekinjeno delovanje glavnega programa. Združiti je treba procese na enem mestu in omogočiti enostaven ter hiter pregled. Predvsem so pomembne točne in jasne informacije o napakah.

2.1.1 Operacijski sistem

Za operacijski sistem smo izbrali operacijski sistem Linux. Glavni razlog za izbiro je bil časovni razporejevalnik procesov crontab [5]. Zapis novega

procesa v seznam procesov razporejevalnika je zelo preprost. Operacijski sistem Windows ima podoben program, ki se imenuje časovni razporejevalnik procesov za izvajanje avtomatiziranih nalog (angl. Task Scheduler) [13, 17].

2.1.2 Podatkovna baza

Za hranjenje informacij o procesih smo se odločili za relacijsko podatkovno bazo Microsoft SQL Server [8]. V tej podatkovni bazi bomo hranili ime procesa, zaporedno številko, čas naslednjega zagona, pogostost zaganjanja procesa v obliki cron, naslov URL [3] do glavne funkcionalnosti procesa in ali je proces aktiviran ali ne. Za hranjenje stanj procesov bomo uporabili nerelacijsko dokumentno bazo MongoDB [2]. Razlog je predvsem minimalističen in preprost zapis dokumentov. Dokument je zapisan v formatu JSON [6]. Iz izkušenj v podjetju je MongoDB prostorsko bolj učinkovita kot podatkovna baza Microsoft SQL Server. Namreč, v dokumentni bazi bomo hranili zgodovino stanj o procesih. Zapisov bo zelo veliko, zato je pomembno, da se z nepotrebnimi metapodatki ne porabi preveliko prostora.

2.1.3 Razvojno okolje in programski jezik

Za glavno logiko programa smo se odločili uporabiti novost podjetja Microsoft. Želeli smo preizkusiti okolje .NET Core, ki omogoča neodvisnost programov od operacijskega sistema [12]. Okolje .NET je do sedaj omogočalo razvoj aplikacij in programov le za operacijski sistem Windows. Za razvojno okolje bomo uporabili Microsoft Visual Studio 2017. Zadnja različica okolja omogoča razvoj aplikacij v okolju .NET Core. Za implementacijo rešitve smo izbrali programski jezik C#.

2.1.4 Spletna aplikacija in arhitektura

Kot zadnji korak projekta je izdelava spletne aplikacije za prikaz, pregled in urejanje procesov. S spletno aplikacijo bo mogoče vklopiti ali izklopiti procese ter preveriti napake, do katerih je prišlo ob neuspešni izvedbi procesov. Tu

smo se odločili za razvojno okolje ASP.NET [16]. Pri razvoju bomo upoštevali arhitekturo MVC [10].

2.1.5 Verzioniranje in hranjenje kode

Za sprotno verzioniranje kode in shranjevanje napredka bomo uporabili spletno storitev Bitbucket. Omogoča ustvarjanje novih projektov, dodajanje kode, sledenje spremembam in dodeljevanje pravic uporabnikom nad projekti. Pri lažjem delu s spletno storitvijo bomo uporabili namizno aplikacijo SourceTree. Ta aplikacija je grafični vmesnik za delo s shranjenimi podatki, ki omogoča podobne funkcionalnosti kot Bitbucket.

2.2 Opis tehnologij

2.2.1 Podatkovni bazi Microsoft SQL Server in MongoDB

SQL je strukturiran povpraševalni jezik za delo s podatkovnimi bazami. Omogoča vračanje, dodajanje, spreminjanje in brisanje podatkov ter tudi ustvarjanje podatkovnih baz, dodajanje novih relacijskih shem, dodajanje shranjenih procedur in dodajanje pravic [8]. Za delo z jezikom SQL potrebujemo sistem za upravljanje s podatkovnimi bazami. Relacijska shema je zbirka sorodnih podatkov, sestavljena iz vrstic in stolpcev [8]. Vrstica predstavlja zapis enega podatka. V tem primeru gre za relacijsko podatkovno bazo. Primer tabele je prikazan spodaj (glej sliko 2.1).

2.2.2 Cron, crontab in format crontab

Cron je dodaten program na operacijskem sistemu Linux, ki omogoča avtomatično zaganjanje ukazov ali programov v ozadju [5]. Omogoča zaganjanje programov po rednih časovnih zamikih. Operacijski sistem Windows ima za to nalogo časovni razporejevalnik procesov. Program cron uporablja bese-

dilne datoteke crontab, v katerih so zapisani ukazi, ki se poženejo na določen časovni interval. Vsak uporabnik posebej ima lahko svojo besedilno datoteko crontab. Program cron bo vsako minuto pregledal besedilne datoteke vseh uporabnikov z zapisanimi ukazi in jih zagnal, če časovni interval ustreza pogoju za zagon [1]. Crontab je ukazni poziv in omogoča dodajanje ter pregled ukazov, ki se bodo izvajali na časovnem intervalu. Z ukazom crontab odpremo začasno datoteko dokler datoteke ne shranimo. Čisto na dnu datoteke se za primer nahaja ukaz s formatom crontab.

Vsaka vrstica posebej je ukaz za program cron. Ukaz sestavljata dva sklopa. Prvi sklop pove časovni interval izvajanja ukaza, drugi pa nam pove, kateri ukaz naj izvrši (glej sliko 2.2). Spodaj so navedeni trije primeri časovnega izvajanja datoteke helloWorld:

- 0 5 1 * * * datoteka bo izvedena vsak prvi dan v mesecu ob 5.00 uri [1].
- * 17-20 * * * datoteka bo izvedena dnevno vsako minuto med 17.00 in 20.00 uro [1].
- */2 * * * * datoteka bo izvedena vsaki dve minuti [1].

2.2.3 Linux

Gre za operacijski sistem, podoben operacijskemu sistemu Windows. Upravlja z viri, ki jih ponuja strojna oprema, in jih povezuje s programsko opremo. Sestavljen je iz zagonskega nalagalnika, jedra, gonilnikov, programov, lupine za izvajanje ukazov, grafičnega dela za izris ikon in namiznega okolja [9].

2.2.4 C# in razvojno okolje Visual Studio

Programski jezik C# je visoko nivojski, objektno usmerjen, programski jezik. Sintaksa jezika je zelo podobna programskim jezikom C, C++ in Java. C# omogoča v primerjavi s programskim jezikom Java uporabo praznih vrednosti, naštevne tipe, lambda izraze in neposreden dostop do pomnilnika [14].

Program Visual Studio je interaktivno razvojno okolje, ki omogoča osnovno podlago za pregled in urejanje kakršne koli kode. Omogoča odkrivanje napak, razvoj in objavljanje spletnih aplikacij, spletnih strani in aplikacij za operacijske sisteme Android, iOS in Windows [15].

2.2.5 ASP.NET, HTML, CSS

ASP.NET je modul za razvoj celovitih spletnih aplikacij in del programskega ogrodja .NET. Medtem, ko razvijamo aplikacije ASP.NET, imamo dostop do ostalih razredov programskega ogrodja .NET [16]. HTML je jezik za označevanje nadbesedila, ki opisuje strukturo spletne strani, dokument pa sestavljajo značke. Brskalnik uporabi značke za pravilen prikaz besedila vendar jih ne prikaže [7, 17].

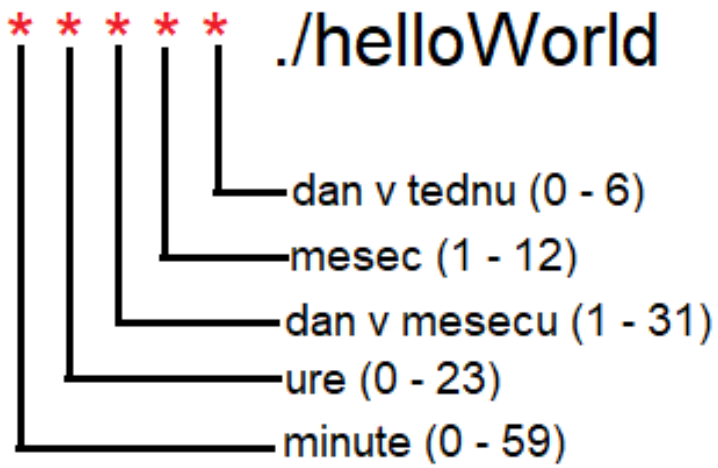
2.3 Potek glavnega programa

Po izbiri tehnologij je sledil načrt poteka glavnega programa. V kratkem opisu spodaj je naveden preprost potek. Za lažje razumevanje pod opis prilagamo sliko diagrama poteka programa (slika 2.3).

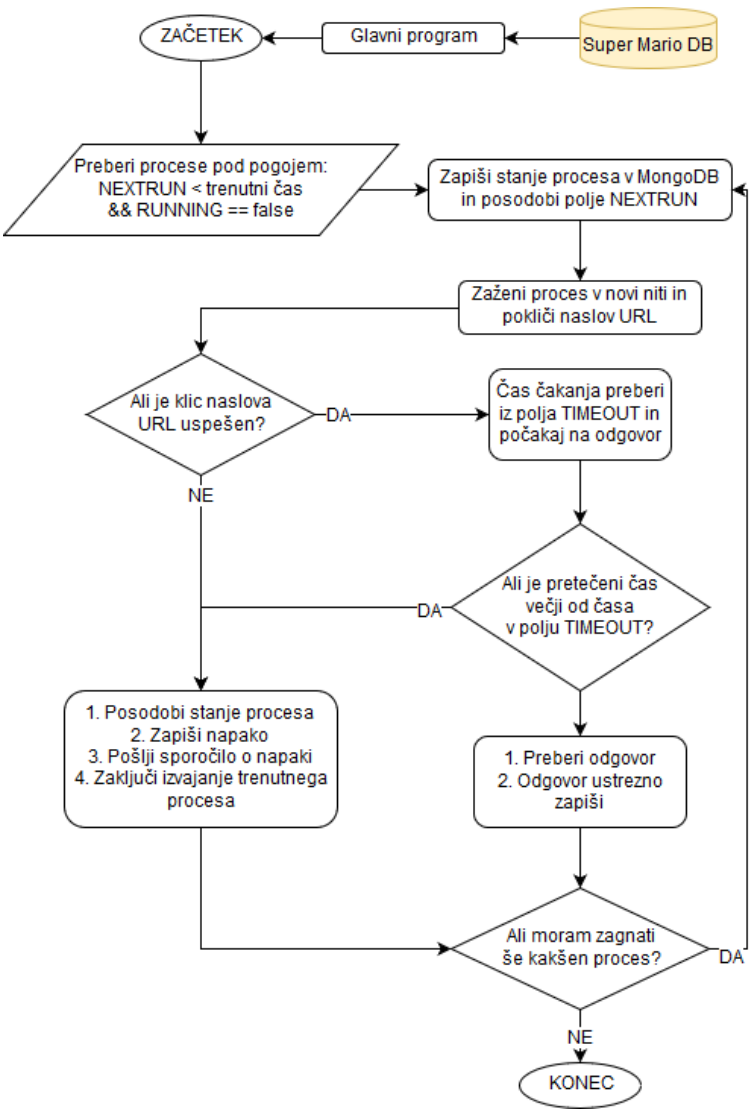
1. Ob zagonu programa preberi procese, ki so veljavni za zagon.
2. Zapiše začetno stanje procesa v MongoDB.
3. Izvede zahtevo na shranjen naslov URL in počakaj na odgovor.
4. Pridobi odgovor in ustrezno zapiše končno stanje procesa.

TASKS	
PK	IdTask INT
Country INT (att.ChildDB) FK	
Name VARCHAR(50)	
Cron VARCHAR(10)	
Enabled BOOL	
NextRun DATETIME	
EstimatedRunTime INT	
Url VARCHAR(100)	
IsRunning BOOL	

Slika 2.1: Primer relacijske sheme SQL s poimenovanimi stolpci.



Slika 2.2: Primer ukaznega poziva v besedilni datoteki crontab.



Slika 2.3: Diagram poteka glavnega programa.

Poglavje 3

Implementacija glavnega programa

V tem poglavju bomo predstavili implementacijo glavnega programa. Pogledali si bomo povezavo do podatkovne baze, branje podatkov iz podatkovne baze in sledenje dogodkov med izvajanjem procesa. Omenili bomo pomen nastavitvene datoteke in na koncu predstavili nekaj težav pri razvoju.

3.1 Nastavitvena datoteka

Nastavitvena datoteka appsetting.json vsebuje globalne nastavitve. Namesto da nastavitve vnesemo neposredno v kodo programa, uporabimo datoteko appsetting.json. S tem načinom se izognemo spreminjanju glavne kode ob spremembi nastavitev. Tako lahko spremenimo samo vrednosti v datoteki in program nemoteno deluje naprej. Vsebina datoteke je navedena spodaj (glej algoritem 3.1).

```
{
  "MongoServer": "192.168.1.10",
  "SqlConn": "Data␣Source=192.168.1.2;database=mother;
     Trusted_Connection=False;Max␣Pool␣Size=1000;User␣ID=
```

```
    project;Password=123456",
  "MongoPortNum": 12345,
  "MongoUserName": "MainUser",
  "MongoPassword": "123qweewq321",
  "MongoCollection": "Processes",
  "MongoDatabase": "Main",
  "MongoAuth": "admin",
  "PathToLogFiles": "//project//log",
  "DateFormat": "yyyy-MM-dd␣HH:mm"
}
```

Algoritem 3.1: Datoteka appsetting.json.

Nastavitvena datoteka vsebuje:

- MongoServer – naslov IP do strežnika z dokumentno bazo MongoDB,
- SqlConn – nastavitve za povezavo s podatkovno bazo na strežniku Microsoft SQL Server,
- MongoPortNum – številka vrat,
- MongoUserName, MongoPassword, MongoAuth – podatki za avtentikacijo,
- MongoCollection – ime glavne zbirke dokumentov,
- MongoDatabase – ime podatkovne baze,
- PathToLogFiles – pot za shranjevanje beleženja,
- DateFormat – oblika zapisa datuma.

Datoteka je zapisana v obliki JSON. Vsebuje ime ključa in vrednost ključa. Glavni razlog za izbiro zapisa oblike JSON je minimalističen zapis. Nastavitve, zapisane v datoteki, preberemo s pomočjo razreda Config.cs. Pri branju

datoteke smo si pomagali z uporabo metode razreda JsonConverter, ki nastavitveno datoteko pravilno prebere in izlušči tiste nastavitve, ki jih želimo prebrati. Metoda SearchAppSettings(string searchKey) nam poenostavi iskanje nastavitev. Za vhodno vrednost metode prejme ime ključa in vrne vrednost, ki ji pripada. Spodaj je navedena vsebina datoteke (glej algoritem 3.2).

```
public string Conn = SearchAppSettings("Conn");
public string Mongo1 = SearchAppSettings("Mongo1");
public int MongoPort = SearchAppSettings("MongoPortNum").
   AnyToInt();
public string MongoUserName = SearchAppSettings("MongoUserName"
   );
public string MongoPassword = SearchAppSettings("MongoPassword"
   );
public string MongoCollection = SearchAppSettings("
   MongoCollection");
public string MongoDatabase = SearchAppSettings("MongoDatabase"
   );
public string MongoAuth = SearchAppSettings("MongoAuth");
public string PathToLogFiles = SearchAppSettings("
   PathToLogFiles");
public string DateFormat = SearchAppSettings("DateFormat");

public string SearchAppSettings(string searchKey)
{
 var json = JsonConvert
       .DeserializeObject<Dictionary<string, string>>
       (
         File.ReadAllText(
           Path.Combine(
             AppDomain.CurrentDomain.BaseDirectory, "appsetting
```

```
                .json"
            )
          )
        );
 json.TryGetValue(searchKey, out var returnValue);
 return returnValue ?? string.Empty;
}
```

Algoritem 3.2: Datoteka Config.cs.

3.2 Povezava s podatkovno bazo

3.2.1 Microsoft SQL Server

Za vzpostavitev povezave s podatkovno bazo Microsoft SQL Server uporabimo nadrazred BaseDataAccess. Zgledovali smo se po članku [11]. Rešitev nam je zelo ustrezala, ker smo se izognili vsakokratnemu ponavljanju kode pri vzpostavitvi povezave s podatkovno bazo. V metodi GetDataReader() nadrazreda se nahaja implementacija povezave do podatkovne baze. Nahaja se na enem mestu in jo lahko uporabimo večkrat brez nepotrebnega ponavljanja vzpostavljanja povezave. Metoda sprejme dve spremenljivki. Prva spremenljivka je namenjena imenu iskane shranjene procedure, druga pa seznamu vhodnih spremenljivk, če jih shranjena procedura zahteva.

```
private static readonly string ConnectionString = Config.Conn;

    private static SqlConnection GetConnection()
    {
      SqlConnection connection = new SqlConnection(
          ConnectionString);
      if (connection.State != ConnectionState.Open)
        connection.Open();
```

```
    return connection;
  }
```

Algoritem 3.3: Implementirana povezava do podatkovne baze.

3.2.2 MongoDB

Povezavo do dokumentne baze MongoDB potrebujemo za zapis končnih stanj procesov. Razred MongoDb() smo naredili tako, da je v konstruktorju logika za vzpostavitev povezave (glej algoritem 3.4). Spodaj je navedena implementacija povezave. Na ta način smo dosegli vzpostavitev povezave na enem mestu, zato jo lahko večkrat uporabimo. Ta način bo prišel prav pri metodah, ki bodo dodajale in spreminjale podatke v podatkovni bazi.

```
private readonly IMongoCollection<MongoTaskHistory> _collection
   ;
public MongoDb()
{
var creditential = MongoCredential
.CreateCredential(
   Config.MongoAuth,
   Config.MongoUserName,
   Config.MongoPassword
);

var servers = new List<MongoServerAddress>
{
 new MongoServerAddress(Config.Mongo1, Config.MongoPortNum)
};

var settings = new MongoClientSettings
{
```

```
 Credentials = new[] { creditential },
 Servers = servers,
 ReadPreference = ReadPreference.Primary
};

 _client = new MongoClient(settings);
 var db = _client.GetDatabase(Config.MongoDatabase);
 _collection = db.GetCollection<MongoTaskHistory>(
 Config.MongoCollection);
}
```

Algoritem 3.4: Implementacija povezave do podatkovne baze MongoDB.

3.3 Branje procesov iz podatkovne baze

Ustrezne procese za zagon preberemo s pomočjo shranjene procedure. To je shranjena datoteka v podatkovni bazi sistema Microsoft SQL Server, ki vsebuje poizvedbo SQL. Shranjeno proceduro uporabimo namesto pisanja poizvedbenega stavka v kodo glavnega programa. V tabeli procesov (glej sliko 2.1) se nahaja polje NextRun. Polje vsebuje datum, ki nam pove, kdaj se bo proces ponovno zagnal. S pomočjo shranjene procedure report.GetAllTasks pridobimo vse tiste procese, ki so vključeni in po vrednosti v polju NextRun starejši od trenutnega časa. Branje kode je implementirano v spodnjem primeru kode.

```
public static IEnumerable<CenejeTask> GetTasks()
{
  using (DbDataReader dataReader = GetDataReader
  ("report.GetAllTasks", null))
    if (dataReader != null && dataReader.HasRows)
      foreach (DbDataRecord row in dataReader)
```

```
        yield return new CenejeTask(row);
}
```

Algoritem 3.5: Uporaba metode GetDataReader() nadrazreda za branje procesov.

3.4 Zapisovanje končnih stanj in rezultatov procesov

Po uspešnem ali neuspešnem zagonu bi radi sledili dogodkom procesa. V ta namen ustvarimo dve metodi, ki posodabljata podatke v dokumentni bazi MongoDB. Metoda InsertInHistory() prejme objekt tipa MongoTaskHistory in ga shrani kot dokument.

```
public ObjectId InsertInHistory(MongoTaskHistory mth)
{
  _collection.InsertOne(mth);
  return mth._id;
}
```

Algoritem 3.6: Metoda za vstavljanje zapisa v MongoDB.

Objekt vsebuje pomembne informacije procesa. Zanima nas predvsem spremenljivka Response. V tem polju shranimo statusno kodo izvedenega procesa. Za statusne kode uporabljamo kodo 200, ki označuje uspešno zaključen proces, ali kodo 500, ki sporoči, da je bilo s procesom nekaj narobe. Podrobnejši opis napake lahko najdemo v dodatnih spremenljivkah. V spremenljivki TimeElapsedSeconds lahko preberemo čas v sekundah, ki ga je proces porabil od začetka do konca izvajanja. Če želimo pri procesu shraniti poljubne informacije, ki bi bile pomembne, uporabimo spremenljivko CustomData.

```
public class MongoTaskHistroy{
    public ObjectId _id { get; set; }
    public int IdTask { get; set; }
    public string NameTask { get; set; }
    public string Url { get; set; }
    public string TaskStatus { get; set; }
    public DateTime StartDateTime { get; set; }
    public DateTime? EndDateTime { get; set; }
    public StatusResponse Response { get; set; }

}
public class StatusResponse{
    public int Code { get; set; }
    public string Message { get; set; }
    public string ExceptionStackTrace { get; set; }
    public string ExceptionMessage { get; set; }
    public int TimeElapsedSeconds { get; set; }
    public object CustomData { get; set; }

}
```

Algoritem 3.7: Osnovna razreda za dokumentno bazo MongoDB.

Poleg zapisovanja končnih stanj procesov, smo dodali sprotno beleženje dogodkov. Sprotne dogodke beležimo v navadno besedilno datoteko, ki jo shranimo na strežniku programa. V primeru, da nas zanima podrobnost izvajanja določenega procesa, to storimo tako, da preberemo zapise v tej besedilni datoteki.

```
public static void LogMessage(
    CenejeTask task,
    string message,
```

```
    string path
)
{
  string logFileName = string.Format(
    @"{0}/{1}",
    path,
    task.Name + ".log"
  );
  Console.WriteLine(message);
  File.AppendAllText(logFileName, message + Environment.NewLine
     );
}
```

Algoritem 3.8: Metoda za zapisovanje stanj v besedilni dokument.

3.5 Težave pri razvoju

Sam razvoj celotnega projekta nam je predstavljal največji izziv. Pri razvoju smo naleteli na sledeče izzive in težave:

- Prvo težavo pri razvoju nam je predstavljala glavna zanka zagona procesov. Pomembno nam je bilo, da se vsi procesi zaženejo in ne čakajo drug na drugega. Zato smo se odločili za uporabo vzporednega izvajanja procesov. To smo izvedeli z uporabo večnitenja. Vsak proces smo zagnal v novi niti.

- Druga težava se je pojavila pri klicu procesa na naslov URL. Pri klicu smo morali zagotoviti, da se povezava ne zapre in da pridobi ustrezen odgovor ne glede na pretečeni čas.

Poglavje 4

Spletna aplikacija za pregled procesov

V tem poglavju je predstavljena izdelava spletne aplikacije za urejanje in pregled procesov. Povezave do podatkovne baze ne bomo ponavljali, ker je narejena na enak način, kot je predstavljeno v tretjem poglavju. Pogledali si bomo njeno uporabo. Zaradi varovanja informacij smo resnične podatke zbrisali ali zamenjali. Pri razvoju smo uporabil spletne tehnologije HTML, CSS in ogrodje za oblikovanje strani Bootstrap.

4.1 Nastavitvena datoteka

V primerjavi z nastavitveno datoteko iz tretjega poglavja je razlika velika. Datoteka je zapisana v formatu XML [4]. Zaradi razvoja spletne aplikacije v okolju ASP.NET nismo želeli spreminjati oblike zapisa datoteke. Datoteka, poleg poljubnih nastavitev, vsebuje nastavitve, pomembne za delovanje spletne aplikacije. Podobno kot v prejšnjem poglavju, smo tu ustvarili razred Config.cs, ki je namenjen branju poljubnih nastavitev.

```
<appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
```

```
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
    <add key="Mongo1" value="192.168.1.10" />
    <add key="MongoPort" value="12345" />
    <add key="MongoUsername" value="Mainuser" />
    <add key="MongoPassword" value="123qweewq321" />
    <add key="MongoCollection" value="Processes" />
    <add key="MongoDatabase" value="Main" />
    <add key="MongoAdmin" value="admin" />
</appSettings>
```

Algoritem 4.1: Nastavitvena datoteka web.config.

4.2 Funkcionalnosti spletne aplikacije

4.2.1 Pregled procesov

Glavna spletna stran vsebuje pregled uspešno in neuspešno izvedenih procesov (glej sliko 4.1). Na strani je možno prebrati podatke, kako dolgo je proces tekel, kdaj se je zagnal in kdaj zaključil. Stran je razdeljena na dva dela. Na zgornji polovici se nahajajo informacije o zadnjih desetih procesih, ki so se uspešno zaključili. Na spodnji polovici pa vidimo vse procese, ki so se izvedli neuspešno. Na desni strani neuspešnega procesa se nahaja gumb, ki nam odpre informacije o napaki (glej sliko 4.4). To napako lahko ustrezno odpravimo in odobrimo. Odobrena napaka se odstrani s seznama neuspešnih procesov.

4.2.2 Urejanje procesov

Naslednja pomembna stran v aplikaciji je namenjena urejanju procesov (glej sliko 4.2). Na tej strani je možno po imenu poiskati ustrezen proces. Konzola

izpisuje naslednje stolpce:

- Id Task – zaporedna številka procesa. Id preberemo iz podatkovne baze SQL. Gre za pomemben podatek pri iskanju zgodovine iz dokumentne baze MongoDB,
- Country – informacija, kateri državi pripada proces,
- Name – ime procesa,
- Cron – zapis, ki pove, kako pogosto se izvaja proces. Več o formatu zapisa se nahaja v drugem poglavju,
- Enabled – omogoča vklop ali izklop procesa,
- Next Run – čas, ko se bo proces ponovno zagnal,
- End Date – čas, po katerem se bo proces avtomatično izklopil,
- Estimated Run Time – čas v minutah. Pove nam, kako dolgo naj bi tekel proces. Po pretečenem času se proces ustavi in je označen kot neuspešno izveden,
- Url – naslov procesa,
- Is Running – stanje, ki nam pove, ali proces teče ali ne,
- Edit button – ob kliku na gumb se nam pojavi okno za urejanje procesa (glej sliko 4.3).

4.2.3 Pregled neuspešnih procesov

Pri pregledu napake se nam odpre okno s podrobnostmi (glej sliko 4.4). Po pregledu in popravku napake lahko potrdimo proces s pritiskom na gumb Approve failed task. S tem dejanjem proces odstranimo s seznama neuspešnih procesov.

4.2.4 Pomoč pri dodajanju procesov

Zaradi uporabe spletne aplikacije s strani drugih uporabnikov smo ustvarili stran z navodili (glej sliko 4.5). Na tej strani so podana navodila za lažje dodajanje novih procesov.

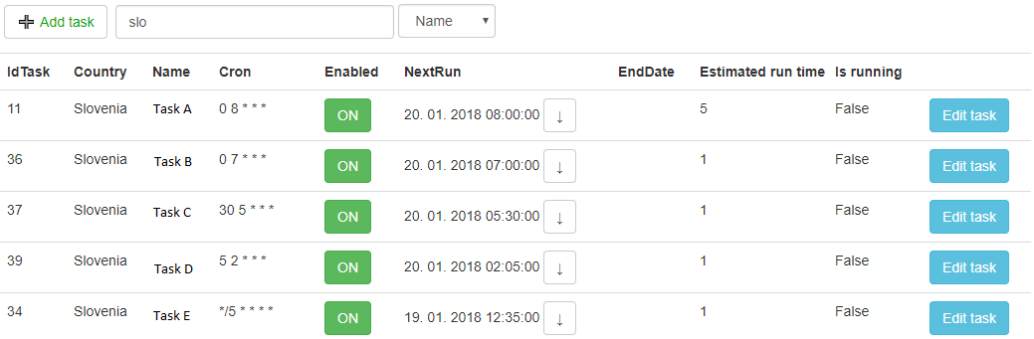
SUCCESSFUL tasks

ID	Name	Status	Start time	End time	Elapsed time (s)
71	Error report SR	Success	19. 01. 2018 08:30:03	19. 01. 2018 08:30:04	1
180	Product SelectiveOffers Calculate Cro	Success	19. 01. 2018 08:30:03	19. 01. 2018 08:30:05	2
82	Cro Regenerate products	Success	19. 01. 2018 08:30:03	19. 01. 2018 08:30:30	27
68	Error report SLO	Success	19. 01. 2018 08:30:03	19. 01. 2018 08:30:04	1
70	Error report CRO	Success	19. 01. 2018 08:30:03	19. 01. 2018 08:30:04	1

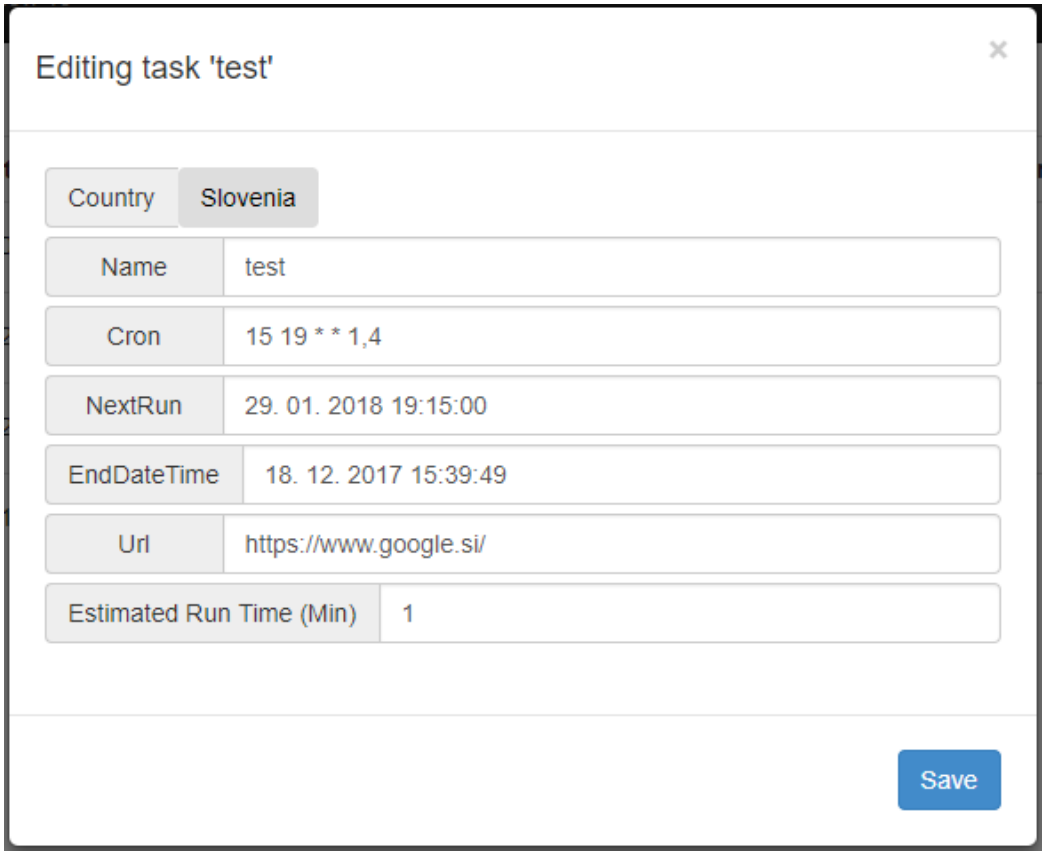
FAILED and WARNING tasks

ID	Name	Status	Start time	End time	Run time (s)	Total errors: 70
63	Addwords sheet update SLO	Failed	19. 01. 2018 08:00:03	19. 01. 2018 08:00:09	6	Error details
63	Addwords sheet update SLO	Failed	19. 01. 2018 07:00:03	19. 01. 2018 07:00:10	7	Error details
63	Addwords sheet update SLO	Failed	19. 01. 2018 06:00:03	19. 01. 2018 06:00:10	6	Error details
63	Addwords sheet update SLO	Failed	19. 01. 2018 05:00:03	19. 01. 2018 05:00:09	6	Error details
63	Addwords sheet update SLO	Failed	19. 01. 2018 04:00:03	19. 01. 2018 04:00:10	7	Error details
83	Rs Regenerate products	Failed	19. 01. 2018 03:27:03	19. 01. 2018 03:27:34	31	Error details
83	Rs Regenerate products	Failed	19. 01. 2018 03:21:03	19. 01. 2018 03:21:33	31	Error details

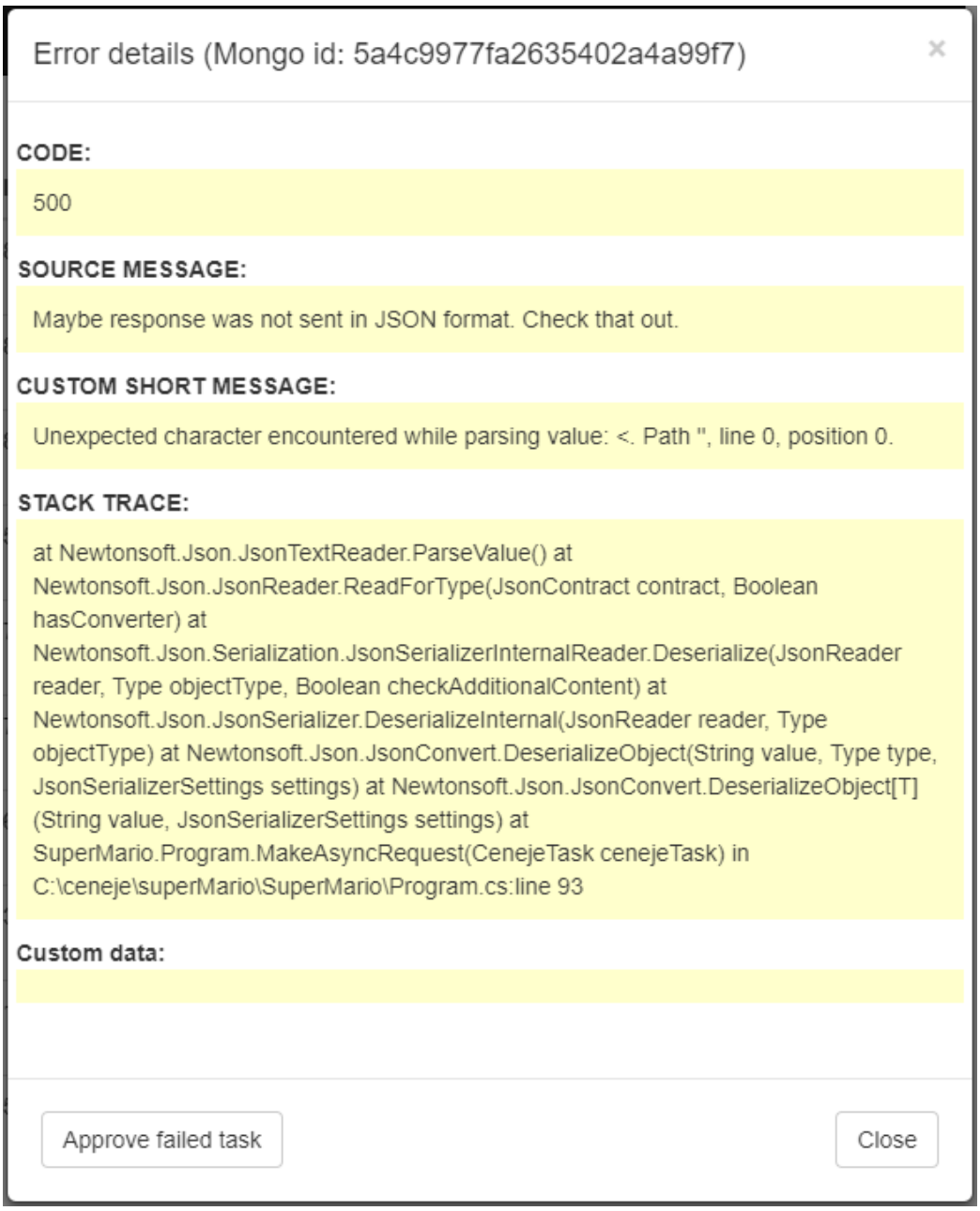
Slika 4.1: Pregled uspešnih in neuspešnih procesov.



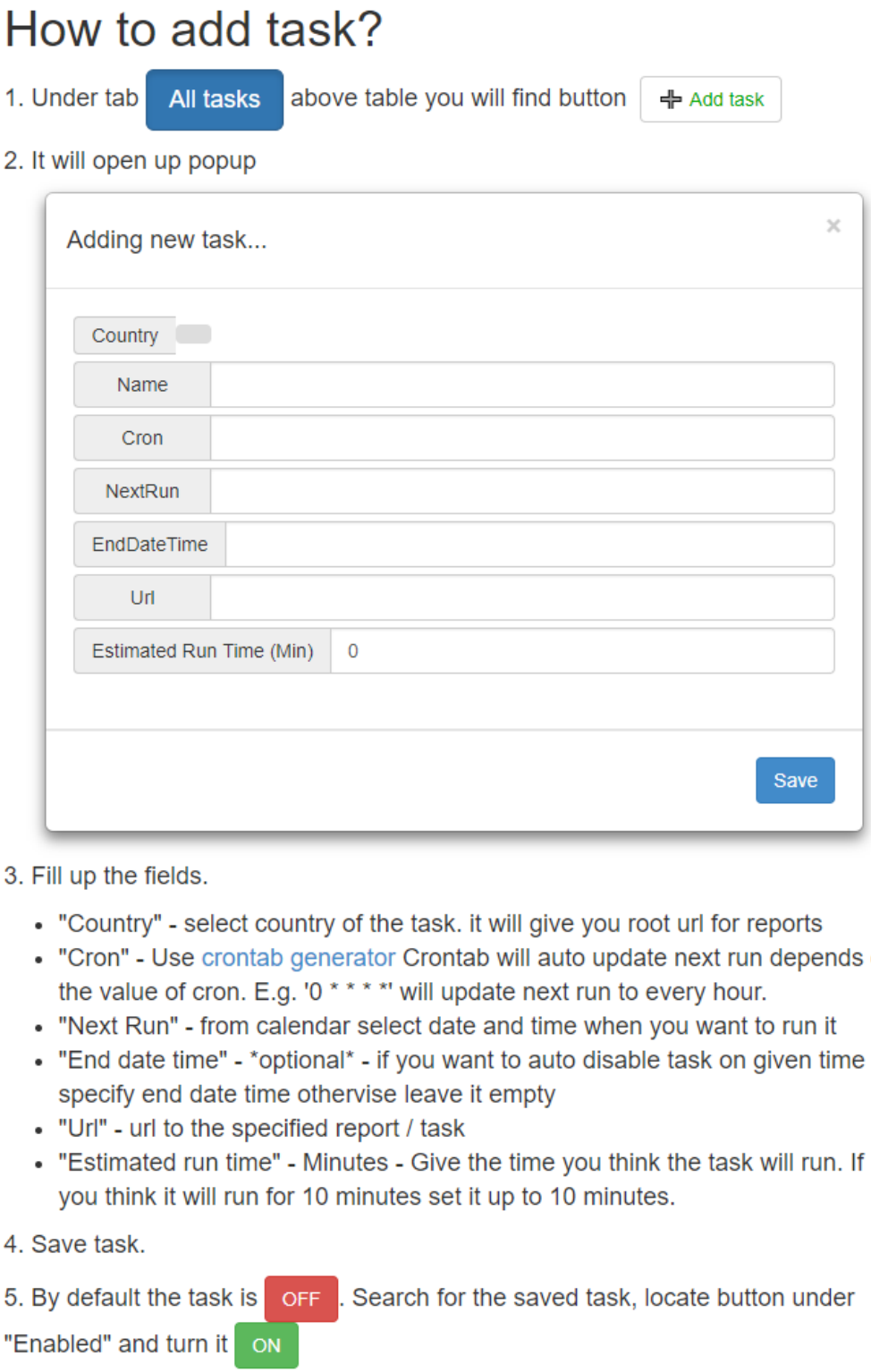
Slika 4.2: Pregled podrobnih informacij procesov.



Slika 4.3: Okno za urejanje procesa.



Slika 4.4: Okno za pregled napake ob neuspešnem procesu.



Slika 4.5: Pomoč pri dodajanju novega procesa.

Poglavje 5

Testiranje in analiza

Implementirano rešitev smo testirali tako, da smo 50 procesov prenesli v spletno aplikacijo. Pri testiranju smo želeli preveriti pravilnost delovanja celovite rešitve. Testiranje je potekalo sedem dni. V tem času smo s pomočjo novega sistema uspeli najti nekaj procesov, ki niso pravilno delovali. Nadaljnja analiza je potekala mesec dni, pri čemer so se pokazale sledeče prednosti projekta:

- krajši čas iskanja in odpravljanja napak (glej sliko grafa 5.1),
- hitreje pridobljene informacije o napakah,
- enostavno urejanje procesov,
- podatek o lokaciji procesa.

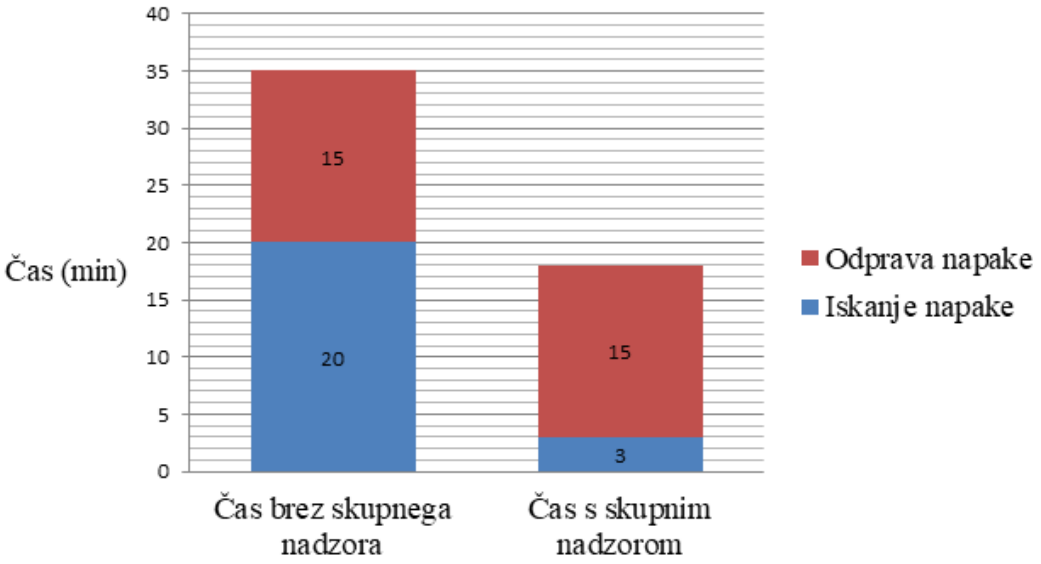
Pri analizi zgodovine izvedenih procesov je mogoče ugotoviti, kdaj je prišlo do kakšne spremembe. Veliko projektov znotraj podjetja zahteva novosti in spremembe. Določene spremembe lahko povzročijo nedelovanje procesa. Tako lahko hitreje odkrijemo, katera sprememba je vplivala na spremenjeno delovanje procesa.

V zadnjem delu testiranja in analize so sodelovali sodelavci. Testiranje z njihove strani je potekalo mesec dni. V tem času so vnašali nove procese,

urejali obstoječe in odpravljali napake. Izpostavili so nekaj pozitivnih mnenj in pripomb.

- Pozitivna mnenja:
  - enostaven pregled nad tekočimi procesi,
  - učinkovitejše odkrivanje napak,
  - hitrejše odpravljanje napak,
  - enostavno dodajanje novih procesov.
- Pripombe in predlogi za izboljšavo:
  - počasnejša stran pri večjem številu procesov – optimizacija poizvedb,
  - včasih opisi napak ne opišejo točne napake – dodati več informacij o napaki.

Vse pripombe in mnenja bodo v nadaljevanju razvoja pripomogla k izboljšanju končne rešitve.



Slika 5.1: Graf, ki prikazuje manjšo porabo časa pri iskanju napake.

Poglavje 6

Zaključek

V diplomskem delu smo predstavili problem, ki se je pojavil znotraj podjetja. Predstavili smo potek izdelave projekta, in sicer od načrtovanja, implementacije in testiranja rešitve. Predstavili smo ključne dele implementacije, ki so bolje pripomogle k nadaljnjemu vzdrževanju in spreminjanju kode. Testiranje in analiza sta pokazala velike prednosti projekta in potrdila osnovne zahteve projekta. Med najpomembnejše naloge končne rešitve spada:

- zbiranje procesov na enem mestu,
- enostaven pregled nad procesi,
- podrobne informacije o napakah.

Ugotavljamo, da je projekt zelo pozitivno pripomogel k nadzoru procesov. Menimo in vidimo, da je veliko možnosti za izboljšave pri spletni aplikaciji. Zaradi vse večjega števila procesov predvidevamo nadgradnjo s preprosto uporabniško prijavo. Spremljali bi lahko, kdo in kaj je spreminjal.

Literatura

[1] Linux Academy. Opis zaganjalnega programa cron na operacijskem sistemu linux. Dosegljivo: `https://linuxacademy.com/blog/linux/the-cron-daemon/`. [Dostopano: 28. 12. 2017].

[2] Kyle Banker. *MongoDB in action*. Manning Publications Co., 2011.

[3] Roy in Masinter Larry Berners-Lee, Tim in Fielding. Rfc 3986, uniform resource identifier (uri): Generic syntax, 2005. *Dosegljivo: http://www. faqs. org/rfcs/rfc3986. html*, 2014. [Dostopano: 5. 01. 2018].

[4] Jean in Sperberg-McQueen C Michael Bray, Tim in Paoli and François Maler, Eve in Yergeau. Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, 16:16, 1998.

[5] Admin's Choice. Opis crontaba. Dosegljivo: `http://www.adminschoice.com/crontab-quick-reference`. [Dostopano: 27. 12. 2017].

[6] Refsnes Data. Opis oblika zapisa json. Dosegljivo: `https://www.w3schools.com/js/js_json_intro.asp`. [Dostopano: 5. 01. 2018].

[7] Refsnes Data. Opis označevalnega jezika html. Dosegljivo: `https://www.w3schools.com/html/html_intro.asp`. [Dostopano: 28. 12. 2017].

[8] Refsnes Data. Opis povpraševalnega jezika sql. Dosegljivo: https://www.w3schools.com/sql/sql_intro.asp. [Dostopano: 28. 12. 2017].

[9] The Linux Foundation. Opis operacijskega sistema linux. Dosegljivo: https://www.linux.com/what-is-linux. [Dostopano: 28. 12. 2017].

[10] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[11] Microsoft. Implementacija povezave s podatkovno bazo sql z uporabo strežnika microsoft sql (ang. microsoft sql server). Dosegljivo: https://social.technet.microsoft.com/wiki/contents/articles/35974.exploring-net-core-net-core-1-0-connecting-sql-server-database.aspx. [Dostopano: 06. 01. 2018].

[12] Microsoft. Opis okolja .net core. Dosegljivo: https://docs.microsoft.com/en-us/dotnet/framework/get-started/net-core-and-open-source. [Dostopano: 27. 12. 2018].

[13] Microsoft. Opis operacijskega sistema windows. Dosegljivo: https://en.wikipedia.org/wiki/Microsoft_Windows. [Dostopano: 05. 01. 2018].

[14] Microsoft. Opis programskega jezika c#. Dosegljivo: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. [Dostopano: 03. 01. 2018].

[15] Microsoft. Opis razvojnega okolja visual studio. Dosegljivo: https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-ide. [Dostopano: 03. 01. 2018].

[16] Microsoft. Podrobnejši opis okolja asp.net. Dosegljivo: https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx. [Dostopano: 03. 01. 2018].

[17] Matjaž Gams (ured.). DIS slovarček, slovar računalniških izrazov, verzija 2.1.71. Dosegljivo: `http://dis-slovarcek.ijs.si`. [Dostopano: 3. 01. 2018].