Факултет техничких наука

Универзитет у Новом Саду

Виртуелизација процеса

ЧИТАЊЕ И КЕШИРАЊЕ ПОДАТАКА О ПОТРОШЊИ

Аутори:	***Број индекса:***
Лука Влатковић	PR 37/2020
Бојан Куљић	PR 43/2020
Вук Огњановић	PR 51/2020
Дора Митић	PR 58/2020

.

Садржај

1 Опште информације о пројекту

Читање и кеширање података о потрошњи је пројекат из предмета „виртуелизација процеса“ који се слуша у VI семестру на Факултету техничких наука у Новом Саду.

Пројекат се састоји од клијентске и серверске апликације које комуницирају помоћу WCF-а. Апликација врши прикупљање података, из екстерних складишта , о прогнозираној и оствареној потрошњи електричне енергије. Апликација се састоји од XML и In-Memory базе података преко којих се постижу боље перформансе.

2 Техничка спецификација пројекта

2.1 Коришћење технологије у изради пројекта

- C# (.NET)
- WCF
- SQLite

2.2 Контекст проблема

- Проблем је у спором приступу екстерној XML бази података и великом утрошку ресурса .Када би стално приступали таквој бази података видно би се успорио рад апликације и била би могућа појава нежељених багова.

2.3 Решење проблема

- До решења проблема би довело увођење нове базе података која би се налазила ближе самој апликацији!
- In-Memory база би омогућила брже реализовање функција добављања података и брисала би те податке након одређеног временског периода. XML база је спорија али ипак поседује већи капацитет који нам је неопходан за потребе складиштења података.
  In-Memory компонентом се смањује број приступа XML бази података и побољшавају се перформансе апликације.

3 Циљеви пројекта и захтеви

3.1 Општи кориснички захтеви

- Захтеви које је изнео корисник су:
  1. Могућност читања података из базе
  2. Могућност примања повратне поруке о непостојећем датуму
  3. Могућност чувања резултата претраге у In-Memory бази података
  4. Могућност аутоматског брисања мерења из локалне In-Memory базе
  5. Могућност креирања CSV датотеке коју чувамо на изабраној локацији

3.2 Технички захтеви

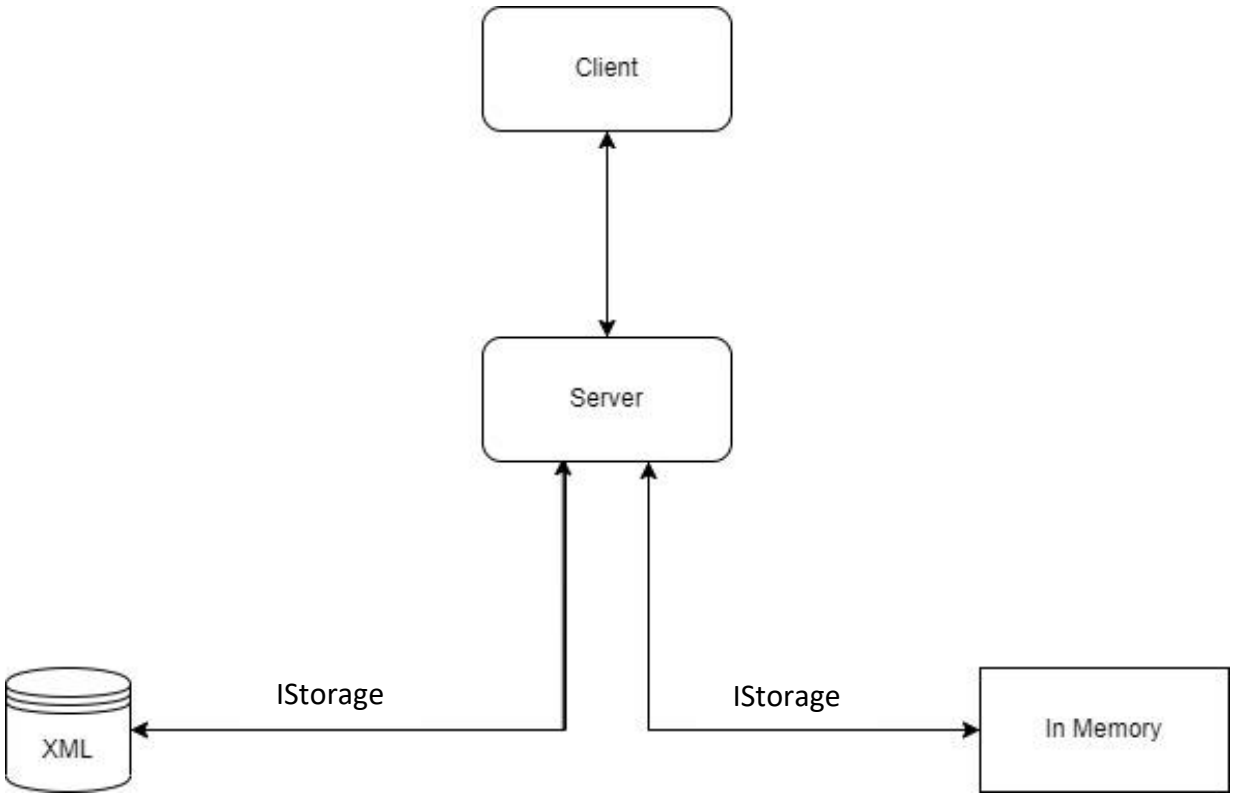
- Нормалан рад саме апликације је предвиђен уз искључиво поштовање одређених смерница пројектаната и ограничења дефинисаних у спецификацији задатака. Апликацију је потребно покретати на оперативном систему **Windows 10** или новије, и да на рачунару на ком се покреће апликација буде инсталиран **.NET Framework верзије 4.8** или новије. Са стране хардверских захтева потребан је минимално **двојезгарни процесор** и **2Gb** радне меморије за респонзиван рад апликације.

4 Дизајн и архитектура система

4.1 Опис и шема система

Дизајн или архитектура система је процес дефинисања елемената система као што су компоненте, интерфејси, повезници и подаци неопходни за систем на основу специфицираних захтева који сви заједно чине именовану компоненту.

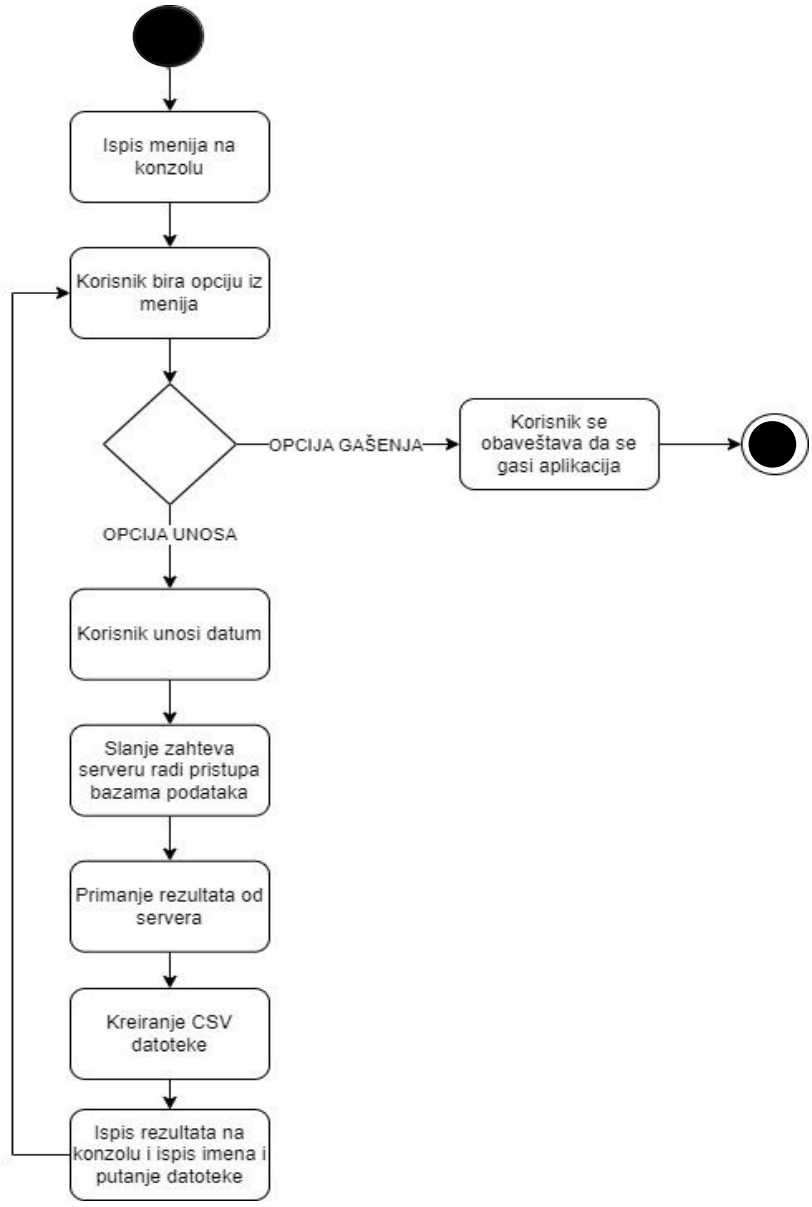
На приложеној слици је приказан дизајн система са дијаграмом компоненти за апликацију о читању и кеширању података о потрошњи :



4.2 Компоненте и дијаграми система

Овај систем се састоји од сервера који комуницира са XML и In-Memory базом података. Клијент представља конзолну апликацију преко које корисник комуницира са сервером и на тај начин врши комуникацију са базама података. Компоненте међусобно комуницирају преко **WCF (Windows Communication Protocol)** и потребних интерфејса. Модел података који се шаље између компоненти је имплементиран у **Class Library** под именом **Common** у фајлу Source. Модел података садржи класе **Load i Audit** које се састоје од поља (Id, Timestamp, ForecastValue, MeasuredValue, MessageType, Message)

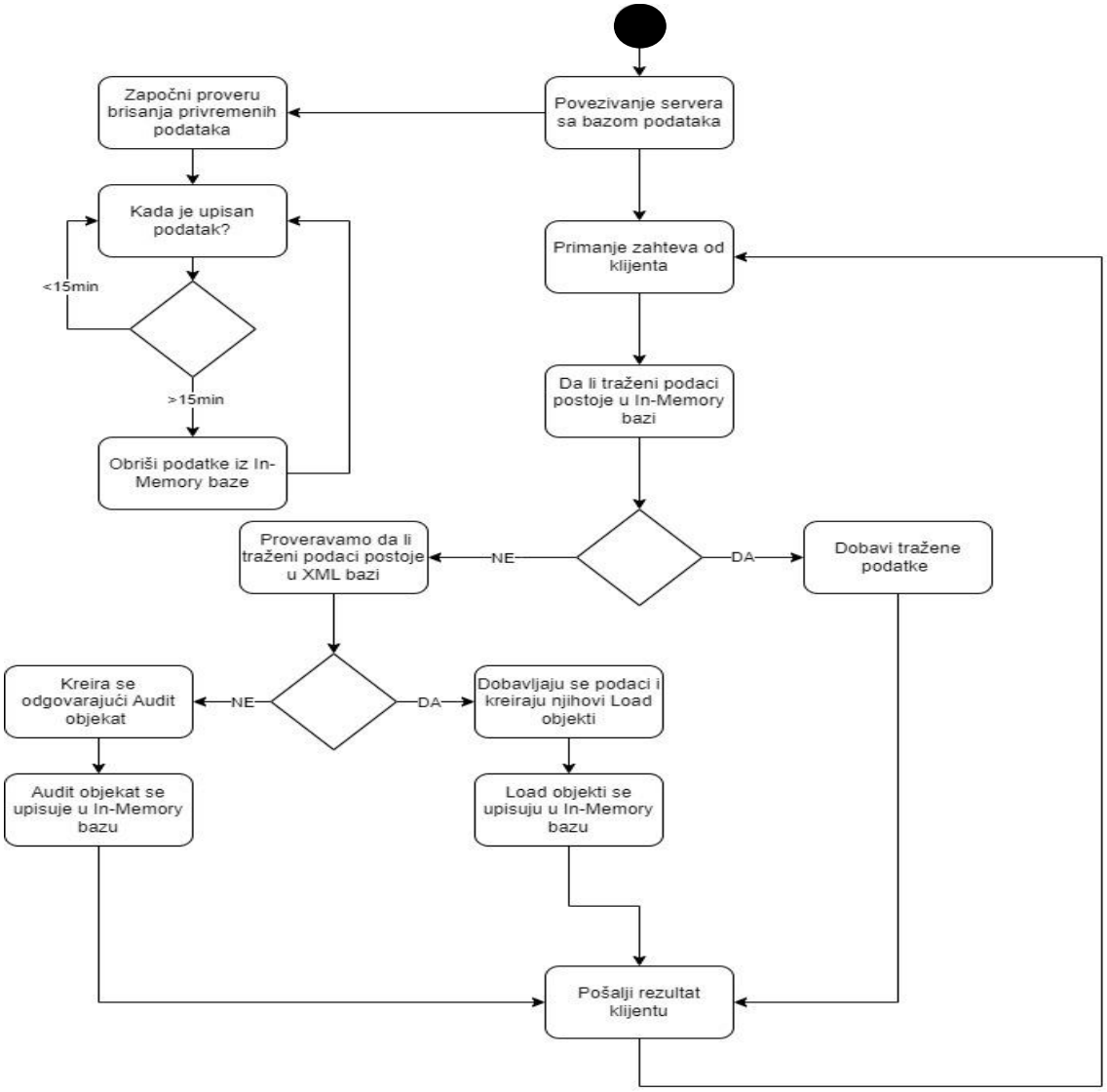
4.2.1 Компонента клијент



Слика 2. Дијаграм активности клијентске компоненте

Клијентска компонента служи за интеракцију са корисником.
Корисник преко клијента уноси датум који се шаље серверу ради добијања жељених мерења.
Прима се резултат претраге и исписује на конзоли, након тога корисник поново добија опцију да унесе датум претраге.
Након примљених Load објеката, на основу њих се креира CSV датотека на дефинисаној локацији и на конзоли се исписује порука о креираној датотеци и њеним подацима.
Корисник има могућност да угаси клијента.

4.2.2 Компонента сервер



. Слика 3. Дијаграм активности серверске компоненте

Компонента сервер комуницира са клијентом и од њега добија датум мерења које корисник захтева.
На основу тог датума сервер врши претрагу In-Memory базе података.
In-Memory чува најтраженије захтеве који се после одређеног времена, након 15 минута, бришу из базе.
Уколико подаци постоје, одговарајући Load објекти се враћају клијенту, а у супротном се врши претрага XML базе података. Уколико постоји мерење креира се Load објекат тог мерења и уписује у In-Memory базу података и шаље клијенту.
Уколико у бази не постоје мерења за унети датум креира се Audit објекат са одговарајућом поруком. Тај Audit објекат се затим уписује у обе базе података и прослеђује се клијентској апликацији као резултат уноса.

4.2.3 Ток података од складишта до серверске компоненте

Сервер има могућност приступа бази података која се налази у 2 облика XML i In-Memory. Приликом покретања сервера In-Memory база података је иницијално празна па се прво приступа XML бази.

Да би смо омогућили пренос података од сервера до складишта креирали смо класу Storage која имплементира интерфејс IStorage и омогућава чување и читање података из расположивих складишта. Креирали смо методе GetLastAudit(), Read(DateTime dt) и Write(Audit audit) које објашњавају функционалности класе.

Метода GetLastAudit() дохвата последње Audit поруке из XML i In-Memory извора.

Метода Read() врши читање Load објеката из InMemory и XML извора на основу датума, а метода Write() је задужена за чување података у нашим базама.

5 Интерфејс и основне функционалности

Интерфејси су кључни елементи у нашем пројекту помоћу којих нам је омогућена апстракција и уговор између класа.

Помоћу њих смо дефинисали скуп метода и својстава које класе могу имплементирати и искористили могућности полиморфизма.
Интерфејси нам пружају уговор између класа и омогућавају лако проширивање програма. Ако класа имплементира интерфејс, она мора пружити све методе и својства које интерфејс дефинише.

5.1 Istorage

Интерфејс IStorage дефинише и обезбеђује функционалности за складиштење и приступ подацима. Интерфејс се састоји од следећих метода:

Read(DateTime dt): Метода Read омогућава читање списка Load објеката на основу задатог датума dt. Ова метода треба да врати листу Load објеката који одговарају задатом датуму.

Write(Audit audit): Метода Write омогућава уписивање Audit поруке у складиште. Прима објекат типа Audit који треба да се запише у складиште.

GetLastAudit(): Метода GetLastAudit враћа последњу Audit поруку из складишта. Ова метода треба да врати објекат типа Audit који представља последњу записану поруку у складишту.

Овај интерфејс дефинише основне операције за читање, уписивање и добијање последње поруке у складишту. Класе које имплементирају овај интерфејс треба да пруже конкретну имплементацију ових метода у складу са захтевима интерфејса.

5.2 Iload

Интерфејс ILoad дефинише методу за листање података (Load).
Метода ListLoads(DateTime dt) пружа операцију за добијање списка Load објеката.
Овај интерфејс је део сервисног контракта (Service Contract) и дефинише операцију коју сервис пружа. Сервис који имплементира овај интерфејс треба да обезбеди имплементацију методе ListLoads, која ће бити доступна клијентским апликацијама које комуницирају са сервисом.

5.3 IAudit

Интерфејс IAudit дефинише операцију за добијање последње аудитне поруке (Audit).
Метода GetLastAudit() пружа операцију за добијање последње аудитне поруке из система. Ова метода треба да врати објекат типа Audit који представља последњу записану поруку. Сервис који имплементира овај интерфејс треба да обезбеди имплементацију методе GetLastAudit, која ће бити доступна клијентским апликацијама које комуницирају са сервисом.

6 Закључак

Вишедневним тимским радом омогућили смо да наша апликација која је рађена по клијент-сервер архитектури:

- комуницира путем WCF-а
- омогући унос датума за који се очекују подаци
- прима захтев од клијента за читање података на основу датума
- приступа XML и In-Memory базама података
- креира нови Audit објекат са одговарајућом поруком који се уписује у обе базе података.
- уписује Load објекте у In-Memory базу података ради бржег приступа у будућности.
- аутоматски брише податке из In-Memory базе података након дефинисаног времена (DataTimeout).
- креира CSV датотеку са подацима примљеним од серверске апликације
- исписује поруку на конзолу са информацијама о креираној датотеци.

5.1 Могућа будућа разматрања и проширења пројекта

Тим је заједничким разматрањем и дискусијом дошао на идеју да би наша апликација могла да има:

1. Подршку за различите формате база података:
   Омогућити апликацији да чита податке не само из XML и In-Memory база података, већ и из других формата база података, као што су релационе базе (RDBMS) или NoSQL базе
2. Употребу кеширања на нивоу клијента:
   Имплементирати механизам кеширања и на страни клијента, како би се додатно убрзало читање података. Клијентска апликација би могла да чува копију претходно прочитаних података и користи их уместо да увек шаље захтев серверу.
3. Додавање механизма аутентификације и ауторизације:
   Увођење система за аутентификацију и ауторизацију корисника како би се омогућило само овлашћеним корисницима да приступају подацима и извршавају операције.
4. Имплементацију система за праћење грешака и логовање:
   Додати механизам за праћење грешака и логовање догађаја у апликацију, како би се олакшала пријава и решавање грешака.