Факултет техничких наука
Универзитет у Новом Саду

Читање и кеширање података о потрошњи

Виртуализација процеса
Предметни пројекат

Аутори:
PR 83/2020 Бојана Михајловић
PR 92/2020 Урош Ивановски
PR 77/2020 Никола Куравица
PR 80/2020 Зоран Тадић

30. мај 2023.

Садржај

1. Увод

Сврха апликације је читање података о прогнозираној и остреној потрошњи електричне енергије, уз уважавање захтева перформанси. Апликација има екстерно складиште података у облику XML базе, као и In-Memory базу података, са циљем побољшања перформанси.

2. Техничке спецификације

2.1. Коришћене технологије:

- C# (.Net Framework)

2.2. Опис проблема

Апликације које користе базе података као складиште података приликом добављања података користе методе које су временски дуге и захтевне за ресурсе. Ако су захтеви за читање података из базе чести, апликација ће имати смањене перформансе.

2.3. Решење проблема

Како би се смањио број читања нових података из базе података, уводи се кеш меморија у серверску апликацију. Она у себи складишти податке успешних претрага на одређено време, након чега се ти подаци бришу.

Уколико се у том периоду док су подаци у кешу понови исти захтев, сервер ће моћи да прочита податке из кеша и да их врати као резултат претраге, без потребе приступа екстерном складишту података.

Уколико сервер у својој кеш меморији нема тражене податке, сервер врши претрагу у XML бази. Ако пронађе тражене податке, њих ће кеширати, подесити им време „истицања“, и вратити те податке кориснику. Једном подешено време за брисање података се касније не мења, без обзира колико често или ретко се том кешираном податку приступа.

Овим додатком серверкој апликацији се смањује број приступа бази података и побољшавају се перформансе апликације.

3. Циљеви пројекта и захтеви

3.1. Општи кориснички захтеви

- Могућност претраге потрошње из XML базе података на основу датума
- Чување резултата претраге у екстерној CSV датотеци
- Кеширање резултата претраге у серверу
- Аутоматско брисање мерења из серверског кеша након истека одређеног периода
- Обавештавање корисника о успешности претраге
- Чување свих порука о успешности у засебну XML базу података

3.2. Додатни захтеви

Апликација треба да буде сигурна по питању управљања меморијом, дакле, треба да користи Dispose pattern и Finalize методе приликом рада са екстерним ресурсима, попут датотека.

Компоненте система треба да међусобно комуницирају преко WCF (Windows Communication Foundation).

Корисник треба да има могућност да преко конфигурационог фајла измени одређене параметре у систему, попут крајњих тачака, локација на којима се налазе XML базе и у којима ће се чувати нове CSV датотеке.

3.3. Технички захтеви

Апликацију је потребно покренути на рачунару са инсталираним Windows 10 (или новијим) оперативним системом, као и .Net Framework 4.8 (или новијим).

4. Дизајн и архитектура система

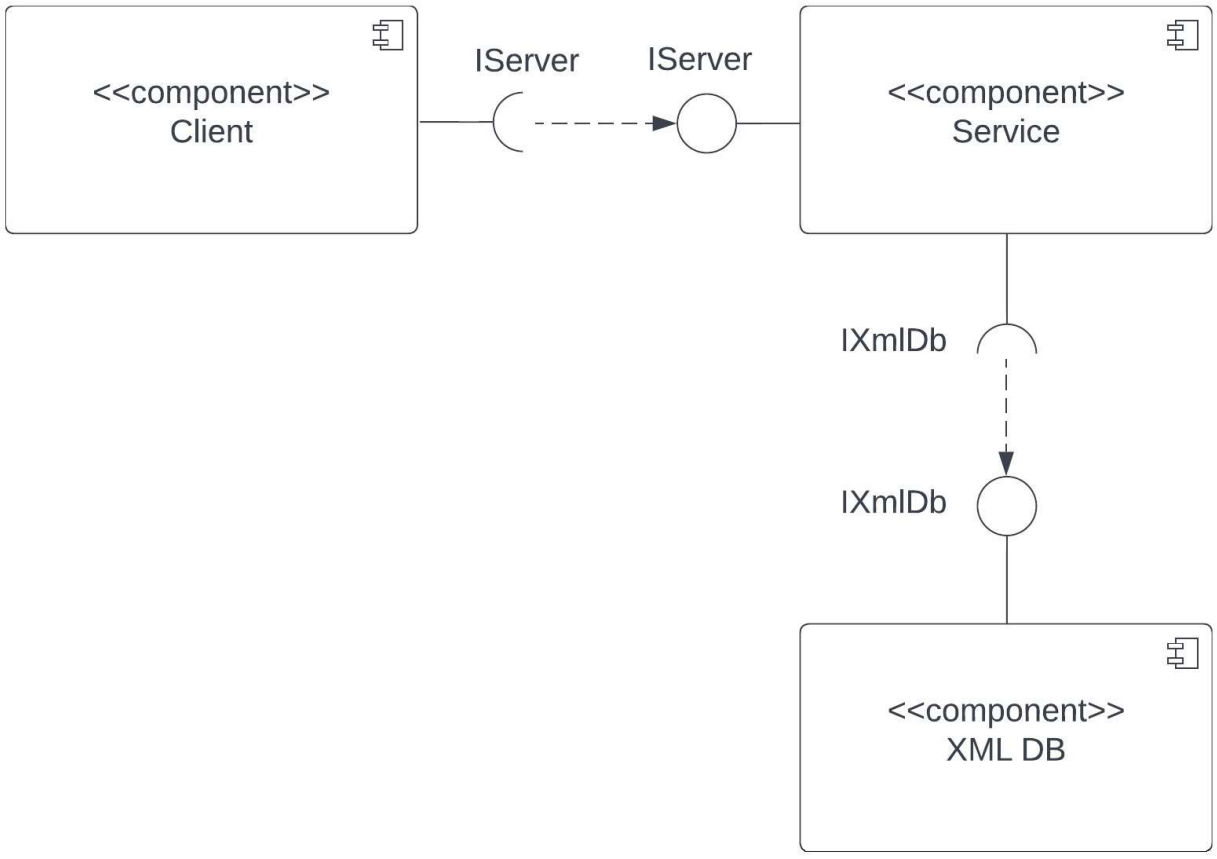
4.1. Компоненте система

Систем се састоји из три компоненте – клијента, сервера и базе података.

Клијент и сервер међусобно комуницирају преко порта 8001. Клијент се обраћа серверу са захтевом за претрагу у ком му шаље датум за који жели да добије резултате.

Сервер и база података комуницирају на порту 8002. Сервер се обраћа бази са неким од могућих захтева – за читање података за тражени датум из Лоад базе података, за највећу вредност ИД у Аудит бази података, или му шаље Аудит објекат за упис.

На слици 1 је приказан дијаграм компоненти система.



Слика 1. *Дијаграм компоненти*

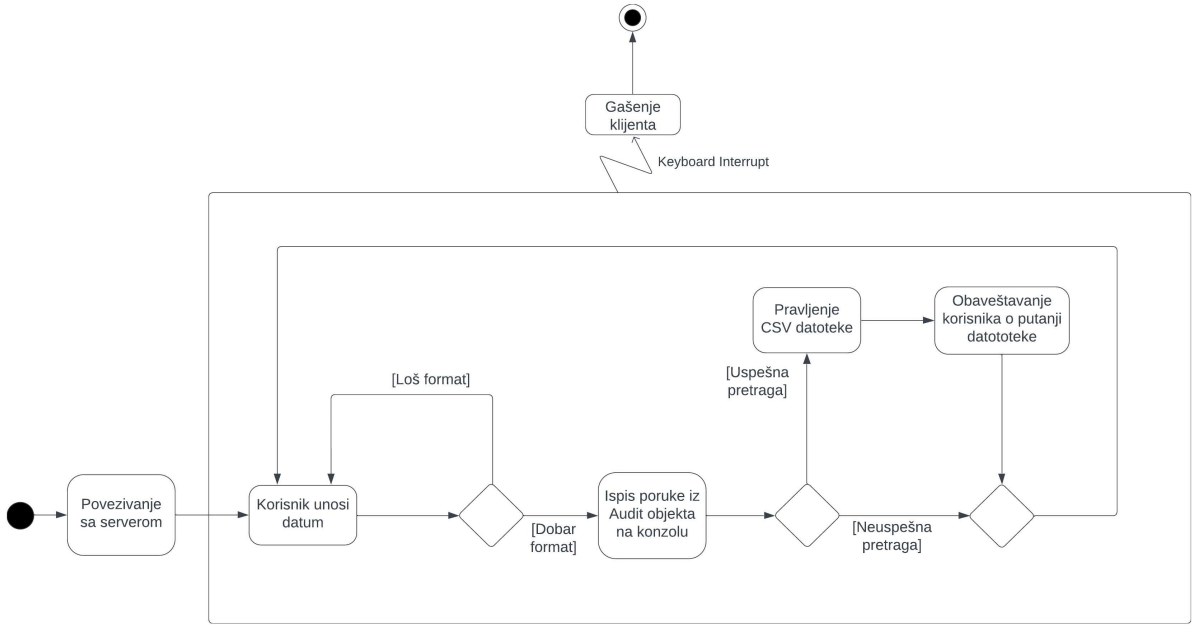
4.1.1. Клијент

Клијент је реализован као CLI (Console Line Interface) апликација. Кориснику се по покретању приказује промпт за унос датума који је од интереса за претрагу. Након уноса датума, контактира се сервер од ког се траже потребни подаци.

По пријему резултата претраге, клијент на конзоли исписује поруку коју је добио од сервера, у којој се обавештава о успешности претраге, и одакле су добављени подаци - ако су пронађени. Ако су подаци пронађени, клијентска апликација их чува у датотеку са резултатима, и кориснику исписује путању до те датотеке.

Након тога, кориснику се поново нуди промпт за унос датума за претрагу.

На слици 2 је приказан дијаграм активности за клијентску апликацију.



Слика 2. *Дијаграм активности за клијентску апликацију*

4.1.2. Сервер

Сервер прима захтеве од клијента за претрагу. Садржи локалну кеш меморију реализовану као два речника, у којима је кључ ИД објекта, а вредност објекат класе Аудит или ЛоадСервис.

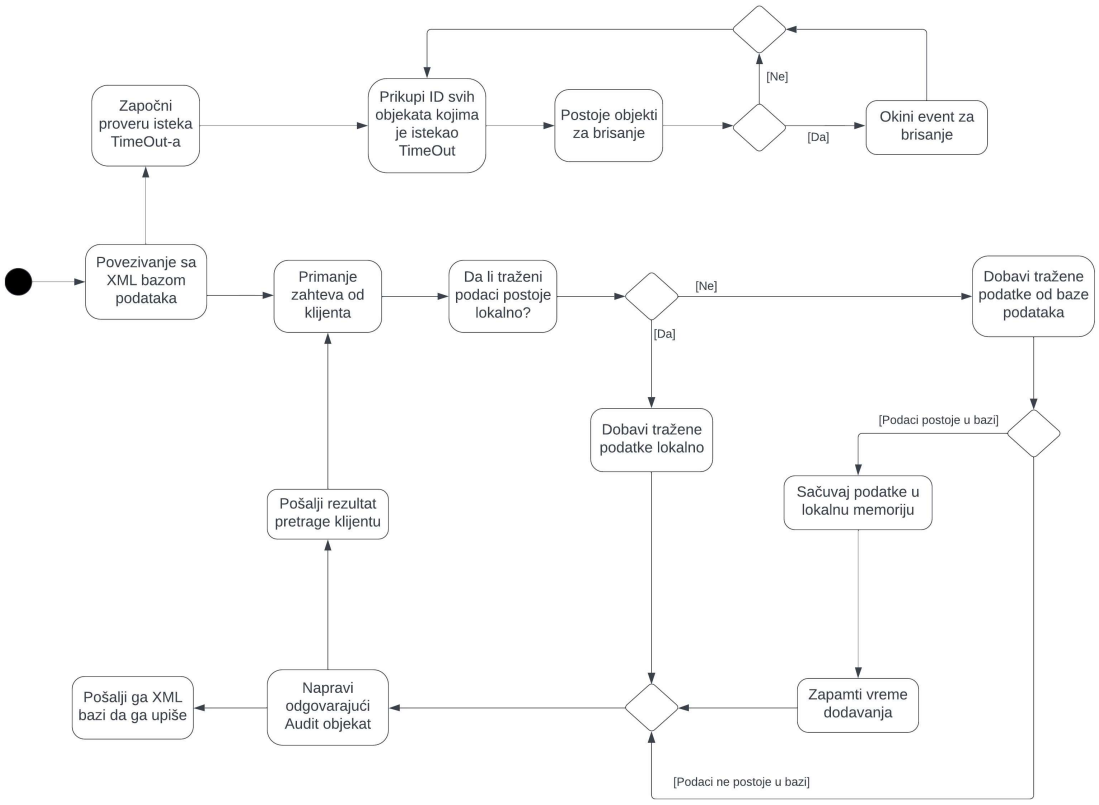
За потребе брисања података после истека одређеног периода, класа Лоад је проширена временском значком када је тачно додата у кеш сервера, и преко ње се проверава да ли је податак „истекао“. При покретању, кеш сервера је празан.

Када прими захтев од клијента, сервер ће прво претражити локалан кеш. У случају да не пронађе тржене податке ту, слаће захтев бази. Ако база пронађе податке, сервер ће те податке прво сачувати у локалној меморији, па ће их тек касније вратити клијенту, заједно са одговарајућим Аудит објектом.

Приликом креирања Аудит објекта, сервер од базе тражи информацију о највећем ИД у бази података за Аудит објекте. На основу тог податка одређује ИД новог објекта. Тај податак се прво шаље у базу како би се трајно сачувао, а затим се заједно са резултатом претраге враћа клијенту.

Сервер има једну позадинску нит која проверава да ли постоје подаци који су „истекли“. У случају да постоје такви подаци, окида догађај који покреће извршавање функције за брисање тих података из кеша. Овако ослобађамо ресурсе потребне за рад серверске апликације.

На слици 3 је приказан дијаграм активности за серверску апликацију.



Слика 3. *Дијаграм активности за серверску апликацију*

4.1.3. База података

База података врши читање из и упис у XML датотеке. База података се састоји из две табеле, једне за Лоад, а друге за Аудит податке.

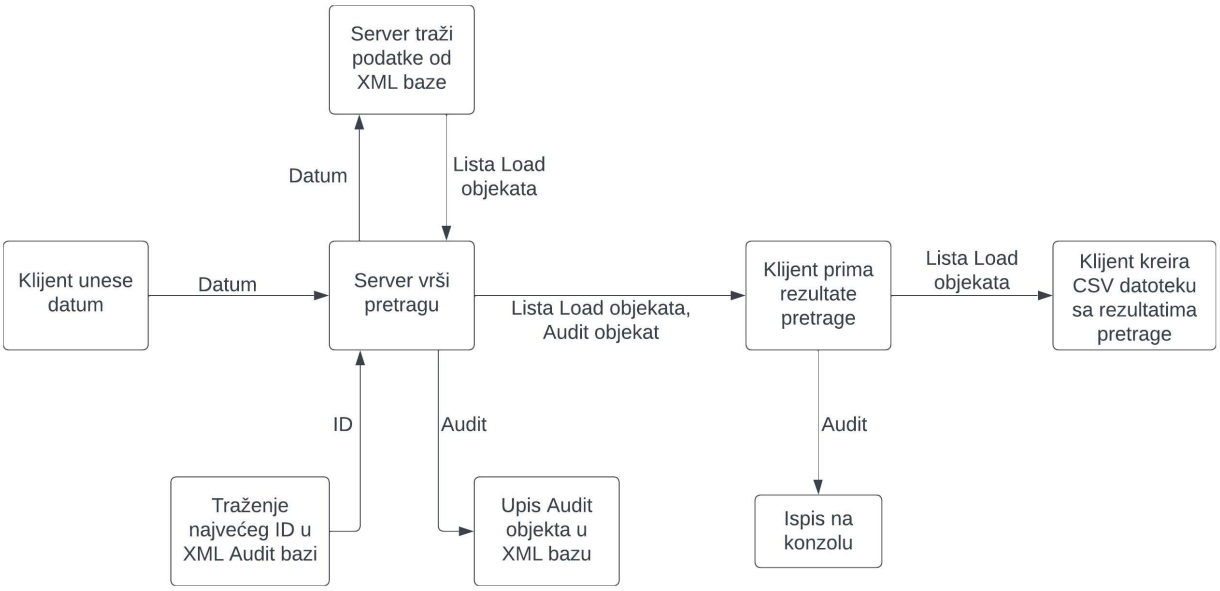
База података од сервера прима датум за који врши претрагу у табели са Лоад подацима. Као резултат претраге враћа листу са траженим подацима.

Такође, од сервера може да прими Аудит објекат који ће уписати у табелу са Аудит подацима.

База података ради са фајловима, тј. са unmanaged ресурсима. Зато је било неопходно имплементирати рад са фајловима на начин који се брине о меморији.

4.2. Ток података

На слици 4 је приказан дијаграм тока података у систему.



Слика 4. *Дијаграм тока података*

4.3. WCF

Компоненте система комуницирају преко WCF. Подешавање WCF-а се састојало из писања интерфејса са функцијама које ће се користити, моделирања података који ће се преносити кроз канале, и из отварања канала при покретању апликација.

4.3.1. Крајње тачке

Сервер се хостује на адреси:

- net.tcp://localhost:8001/Server

База се хостује на адреси:

- net.tcp://localhost:8002/XmlBaza

4.3.2. Интерфејси

Сервер имплементира интерфејс IServer који има само једну методу:

- `Tuple<List<Load>, Audit> UpitOdKlijenta(DateTime datum);`

База имплементира интерфејс IXmlDb који има три методе:

- `List<Load> ProcitajIzBazePodataka(DateTime trazeniDatum);`
- `int NajveciAudit();`
- `void UpisUBazuPodataka(Audit auditi);`

4.3.3. Модели података

У овом систему се користе два модела података: Аудит и Лоад.

Класа Аудит садржи следећа поља:

- `int Id`
- `DateTime Timestamp`
- `MessageType MessageType`
- `string Message`

Класа Лоад садржи следећа поља:

- `int Id`
- `DateTime Timestamp`
- `Double ForecastValue`
- `Double MeasuredValue`

5. Закључак

5.1. Могућа побољшања

Систем има релативно велик број приступа бази података. Највећи број приступа бази је везан за Аудит податке. За сваки креиран Аудит податак се дешавају два приступа бази – један је да се добави ИД, а други је за упис новокреираног податка.

Ови приступи се дешавају и у случају успешне и у случају неуспешне претраге.

Могуће решење за смањење броја приступа бази је кеширање и ових података, и њихов периодичан упис у базу у пакетима. То исто би се могло применити и за читање ИД за Аудит податке, где би се прочитала највећа вредност приликом покретања апликације, па би се та вредност периодично ажурирала/синхронизовала са подацима из базе.

5.2. Даљи развој и истраживање

Тема која нам се чини занимљива је тема делегата и догађаја. У овој имплементацији пројекта није било могућности да се реализује као прави Pub-Sub систем са више класа. То би било нешто на чему би могли да порадимо и више истражимо, поготово што изгледа као нешто што много олакшава позивање само оних метода које нам требају, баш онда када нам требају.