

**Razvoj višeslojnih aplikacija u elektroenergetskim sistemima**

**Projektna dokumentacija**

**Autor:** Božidar Marić PR32/2016

**Asistent:** Mirko Mikać

**Predmetni profesor:** Darko Čapko

Table of Contents

[**Kratak opis aplikacije** 3](#_Toc19493135)

[**Tehnologije i obrasci korišćeni u projektu** 5](#_Toc19493136)

[***MVVM*** 5](#_Toc19493137)

[***Fabrika i Proxy*** 6](#_Toc19493138)

[***Strategija*** 7](#_Toc19493139)

[***Singleton*** 7](#_Toc19493140)

[***Komanda*** 8](#_Toc19493141)

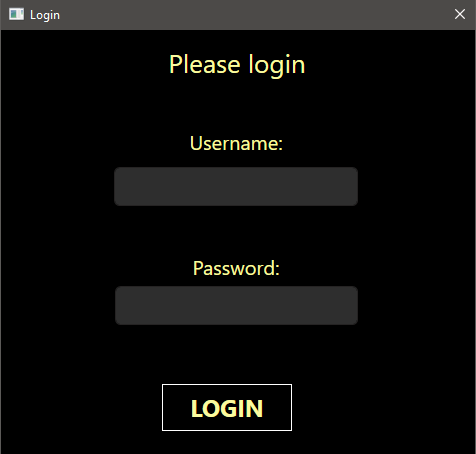
[***Muva*** 9](#_Toc19493142)

# **Kratak opis aplikacije**

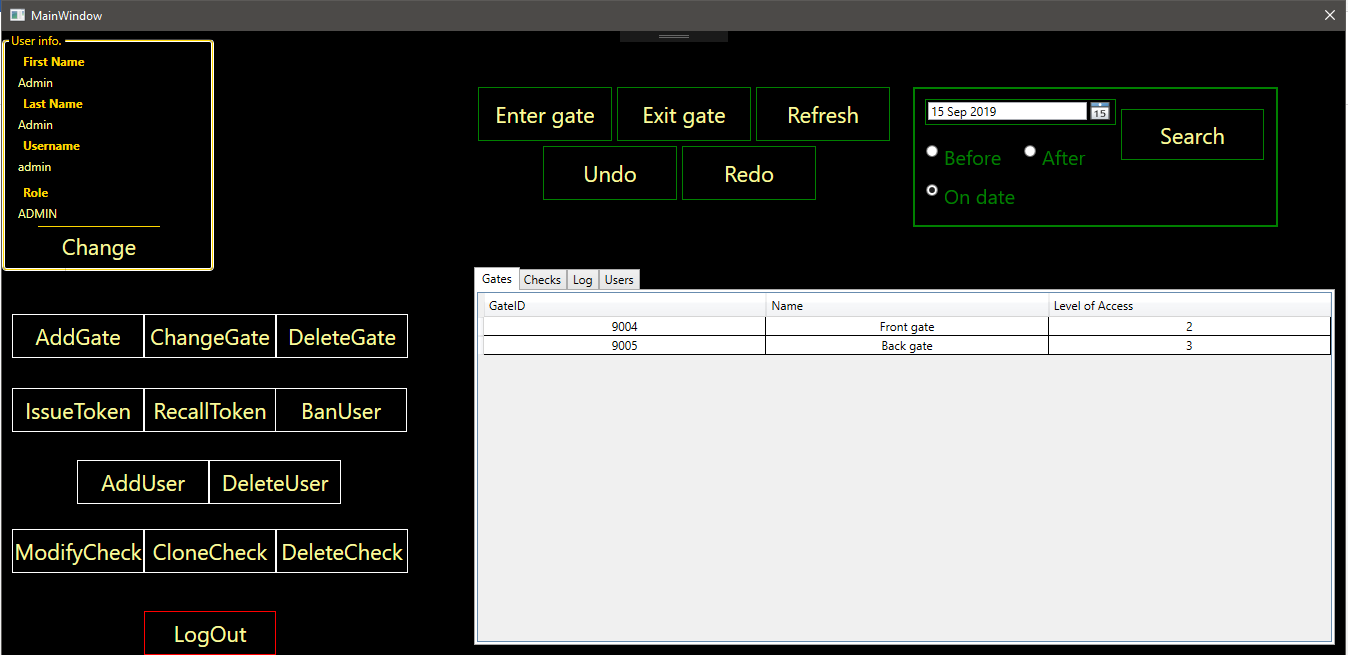
Prema opštem tekstu zadatka i smernicama asistenta aplikacija je osmišljena kao sistem koji prati prolaske korisnika kroz kapije. Svaki korisnik može da se uloguje na sistem pregleda svoje podatke, izmeni i sačuva ih, ima uvid u svoju aktivnost na sistemu i omogućena mu je pretraga i izmena primarnih podataka (u mom slučaju Čekova). Admin ima neke dodatne mogućnosti, kao što su dodavanje i brisanje korisnika, kapija, dodeljivanje Token-a (na osnovu kojih korisnik može da dobije dozvolu da prođe kroz kapiju), povlačenje tog Token-a, i zabranu prolaska konkretnom korisniku kroz određenu kapiju.

Trebalo bi napomenuti da se Čekovi generišu sistemski na osnovu nekoliko kriterijuma. Naime, korisnik selektovanjem kapije i klikom na jedna od dva dugmeta (*Enter gate* ili *Exit gate)* inicira akciju prolaska kroz kapiju, sistem zatim proverava da li je za tog korisnika izdat Token i koji mu je nivo pristupa, tim Token-om, dodeljen, dalje se proverava da li je korisniku zabranjeno da prođe kroz tu kapiju kao i da li je već ušao na neku kapiju, a da nije kroz nju izašao (u slučaju da pokuša da uđe) ili da li je izašao kroz sve kapije na koje je ušao (u slučaju da pokuša da izađe kroz neku kapiju). Na kraju ukoliko je korisnik prošao kroz sve provere izdaje se Ček i korisnik prolazi kroz kapiju.

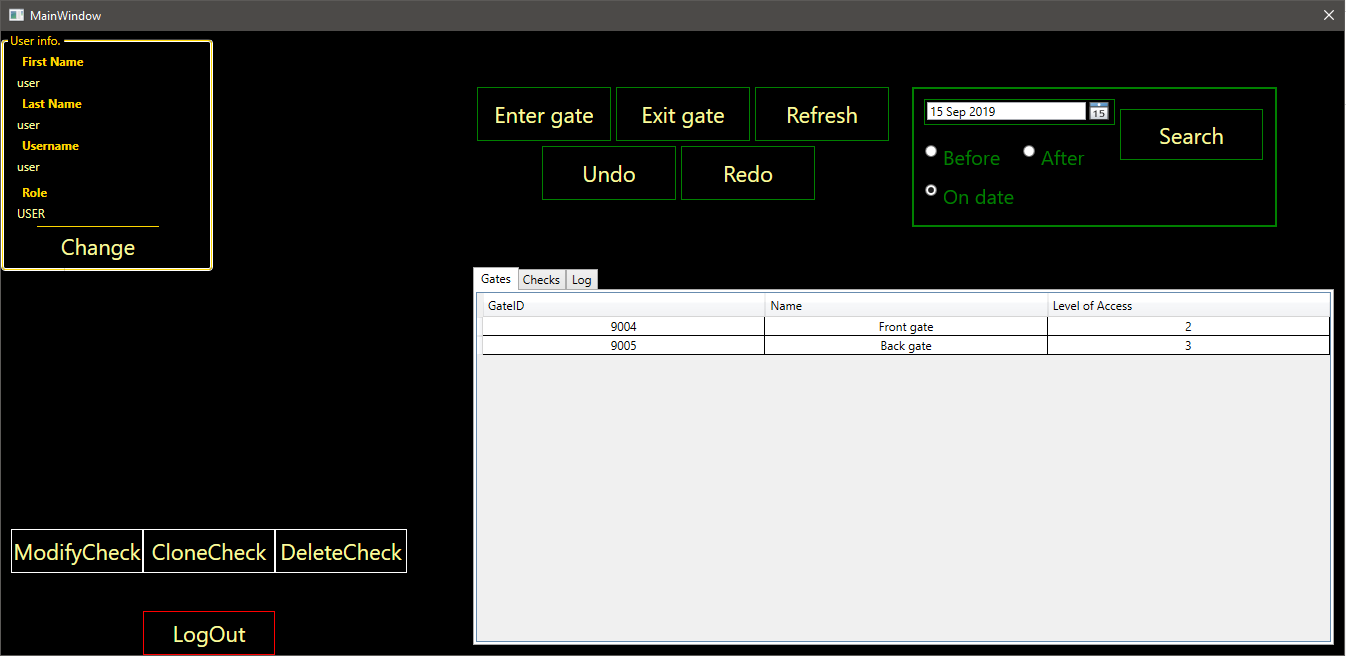
Na slikama ispod možete videti kako izgleda prozor za prijavu korisnika (Slika 1.1) kao i izgled Main prozora za dva različita tipa korisnika (Slike 1.2 i 1.3).



***Slika 1.1*** Prozor za prijavu korisnika



***Slika 1.2*** Izgled Main prozora sa svim opcijama administratora



***Slika 1.3*** Izgled Main prozora sa opcijama običnog korisnika

# **Tehnologije i obrasci korišćeni u projektu**

Pri izradi podataka od tehnologija je korišćeno:

* WPF
* WCF
* Entity Framework Core
* log4net biblioteka
* Data i Command binding

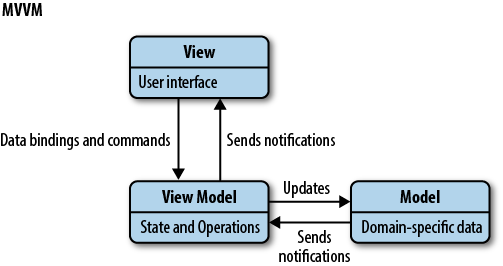
A od obrazaca je implementirano:

* MVVM
* Fabrika i Proxy
* Strategija
* Muva
* Singleton
* Komanda

## ***MVVM***

Model-View-ViewModel je obrazac koji služi za separaciju zaduženja u aplikaciji i koji se sastoji iz tri dela:

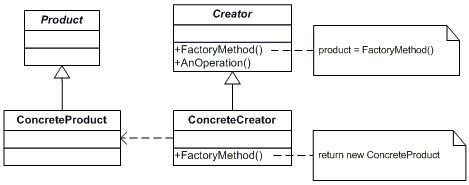
1. Model – definiše podatke i njihove tipove
2. View – definiše izgled korisničkog interfejsa
3. ViewModel – posreduje između Model-a i View-a, zadužen je za prezentaciju podataka i obradu korisnikovih interakcija



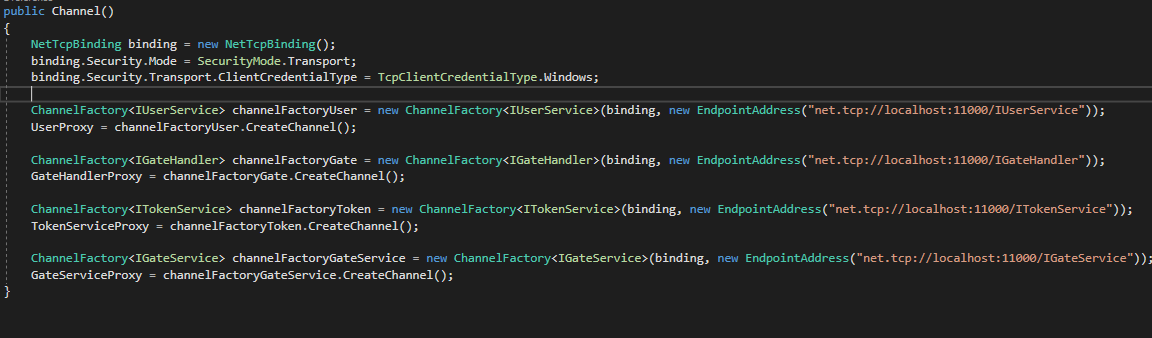
***Slika 2.1*** MVVM

## ***Fabrika i Proxy***

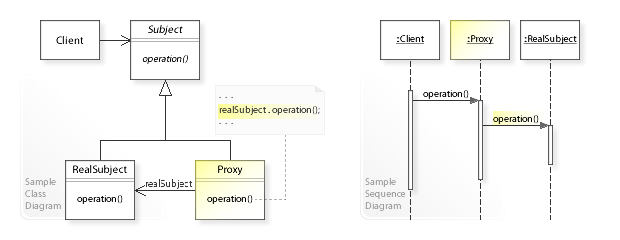
Komunikacija klijenta i servera preko WCF tehnologije se odvija tako što se zahtev za kreiranje objekta prosleđuje Factory metodi. Factroy klasa definiše interfejs za kreiranje objekta, dok je sama implementacija prepuštena podklasama.



***Slika 2.2*** Fabrika



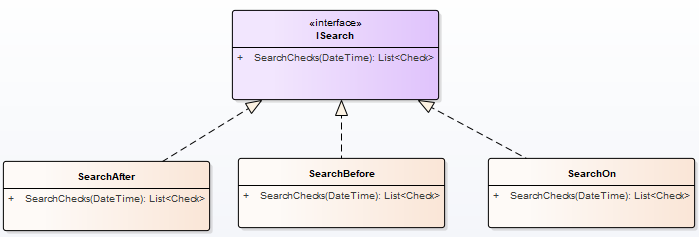
Proxy predstavlja posrednika za pristupanje drugom objektu (obično ako je neophodna kontrola pristupa). On održava referencu koja mu omogućava pristup stvarnom objektu i obezbeđuje interfejs identičan interfejsu subjekta.



***Slika 2.3*** Proxy

## ***Strategija***

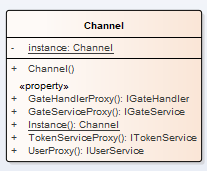
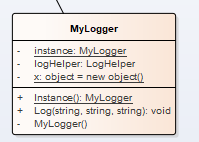
Definiše grupu algoritama, enkapsulirajući ih i čineći ih međusobno zamenljivim. Omogućava laku zamenu algoritama tokom izvršavanja programa.



***Slika 2.4*** Strategija

## ***Singleton***

Unikat ili Singleton obezbeđuje da klasa ima samo jednu instancu i daje globalni pristup toj instanci. Odgovorna je za kreiranje i rad sa svojom sopstvenom jedinstvenom instancom. U aplikaciji je ovaj obrazac implementiran više puta od čega su na slikama dole prikazana dva.

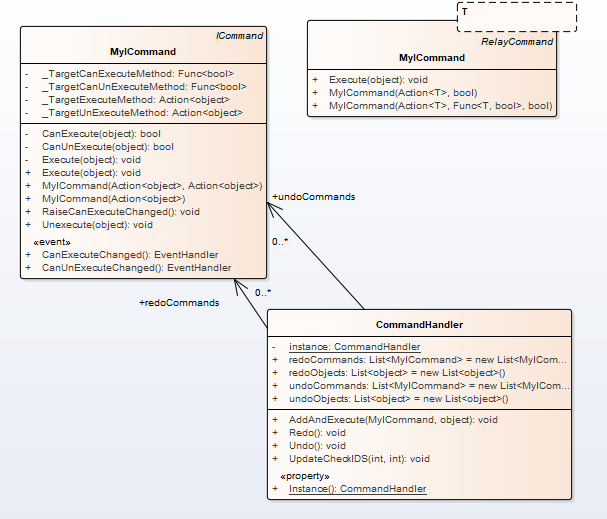


***Slika 2.5*** Singleton

## ***Komanda***

Enkapsulira zahtev za izvršenjem određene operacije u jedan objekat. Umesto da se direktno izvrši određena operacija, kreira se objekat-komanda, koji se potom prosleđuje na izvršenje. Konkretne klase koje implementiraju komande obično imaju mnogo zajedničkih osobina koje se lokalozuju u osnovnu apstraktnu klasu.

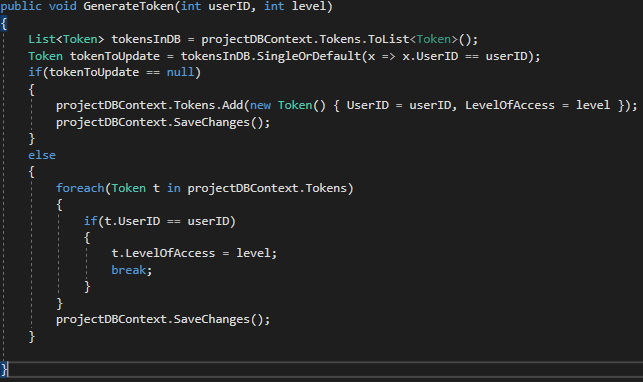
Implementirana komanda je delimično izmenjena upotrebom generičke klase zbog mogućnosti te klase da placeholder-e za tipove polja, metode i parametre zameni odgovarajućim tipovima tokom kompajliranja. Takođe je komanda kombinovana sa Singleton obrascem radi jednostavnijeg pristupa i lakše realizacije Undo/Redo funkcionalnosti.



***Slika 2.6*** Komanda + Singleton

## ***Muva***

Muva je objekat koji minimizuje upotrebu memorije, deljenjem podataka sa drugim, sličnim objektima. Predstavlja način za upravljanje velikom količinom objekata, u situacijama gde konstantno kreiranje novih objekata nije prihvatljivo zbog memorijskih ograničenja. Muva pokušava da maksimizuje ponovnu iskoristivost sličnih objekata njihovih skladištenjem. Kreira nove objekte samo kada ne može da nađe odgovarajući među već postojećim. Implementacija muve je takođe izmenjena u smislu da ne pravi instance klase u programu već pokušava da smanji broj skladištenih podataka u bazi. Implementirana je prilikom kreiranja i skladištenja Token-a jer su ti podaci jedinstveni po korisniku za kog su izdati i postojanje više Tokena koji su dodeljeni jednom korisniku može izazvati probleme u sistemu.



***Slika 2.7*** Muva