Dokumentacja projektu z przedmiotu „Programowanie Zdarzeniowe” realizowanym na 4. semestrze kierunku Informatyka na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej

Współdzielony edytor tekstowy z możliwością kolaboracji poprzez sieć

Prowadzący: dr inż. Jakub Janusz Koperwas

Zespół projektowy:
Kukawka, Aleksandra (300246)
Pokorzyński, Łukasz Sebastian (300251)
Sozański, Patryk Jan (300258)

Data ukończenia projektu: 08.06.2020

Spis treści

Cel projektu

Celem projektu jest utworzenie oprogramowania w języku JAVA, który dostarcza usługi współdzielonego dokumentu tekstowego umożliwiającego zdalną pracę poprzez sieć.

Jednym z głównych założeń projektowych był równomierny rozkład pracy w zespole liczącym od 2 do 3 osób. Współpraca między nimi powinna być organizowana przy użyciu systemu kontroli wersji.

Wymagania projektowe nie określały ściśle sposobu rozwiązania problemu, lecz pozostawiały duże pole do własnej interpretacji w zakresie wyboru metody realizacji interfejsu graficznego i komunikacji sieciowej między modułami oprogramowania.

Niemniej, zostały postawione przed nami oczekiwania o następującej treści:

1. system powinien zawierać graficzny interfejs użytkownika zrealizowany w sposób zdarzeniowy,
2. system powinien zawierać co najmniej dwa moduły komunikujące się ze sobą za pomocą sieci,
3. system powinien zawierać testy jednostkowe.

Opis rozwiązania

Graficzny interfejs użytkownika (GUI)

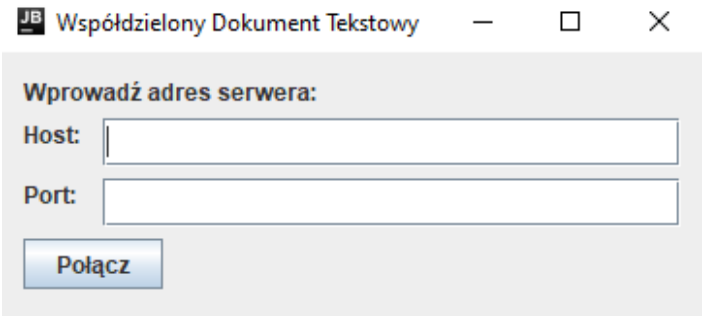
Fragment projektu nadzorowany przez Aleksandrę Kukawkę.

Ta część projektu jest zależna od pakietów *serwer* i *handlers*, dlatego prace nad tymi pakietami były prowadzone w tym samym czasie. Zawiera ona graficzny interfejs i praktyczne odwzorowanie tego, co dzieje się w tych dwóch wymienionych wcześniej pakietach.

Fragment został zrealizowany w sposób zdarzeniowy. Dwie główne biblioteki odpowiedzialne za wygląd i działanie interfejsu to AWT i Swing. Biblioteka AWT dostarczyła nam możliwości przechwytywania zdarzeń generowanych przez użytkownika (m.in. kliknięcie myszką) za pomocą mechanizmu słuchaczy zdarzeń. Biblioteka Swing jest rozwinięciem biblioteki AWT. Zdecydowaliśmy się akurat na te biblioteki, ponieważ znajdują się one w podstawowym API dostarczanym z JDK.

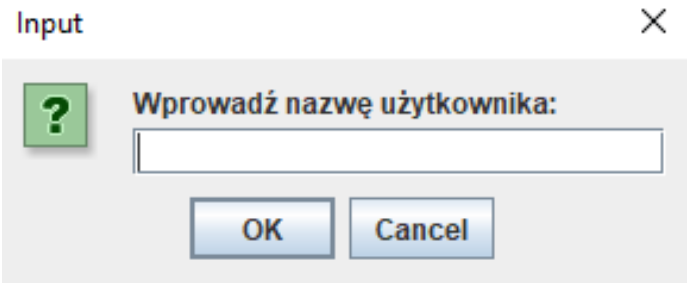
Pakiet *gui* zawiera kilka widoków. Wszystkie one są ze sobą połączone i opisane w klasie *MainWindow.* Gdy użytkownik łączy się do serwera, tworzy się dla niego nowa instancja klasy *MainWindow,* która stanowi podstawowy kontener dla wszystkich widoków oraz gromadzi wszystkie informacje o danym kliencie oraz obsługuje jego wątek. Dodatkowo zarządza przełączaniem między widokami. Teraz po kolei zostanie opisany każdy z nich w kolejności, w jakiej widzi je na ekranie użytkownik.

Na początku po uruchomieniu całego programu wyświetla się okno (rys. 1), w którym użytkownik proszony jest o wprowadzenie adresu serwera (hosta i portu). W chwili wciśnięcia przycisku „Połącz” sprawdzana jest poprawność wpisanych argumentów. Gdy użytkownik poda niepoprawne dane, wyświetlane jest okno błędu, a pola są czyszczone. W przeciwnym wypadku zostaje uruchomiony nowy wątek ConnectViewThread. Całość dotycząca tego fragmentu jest zaimplementowana w klasie *ConnectView*.



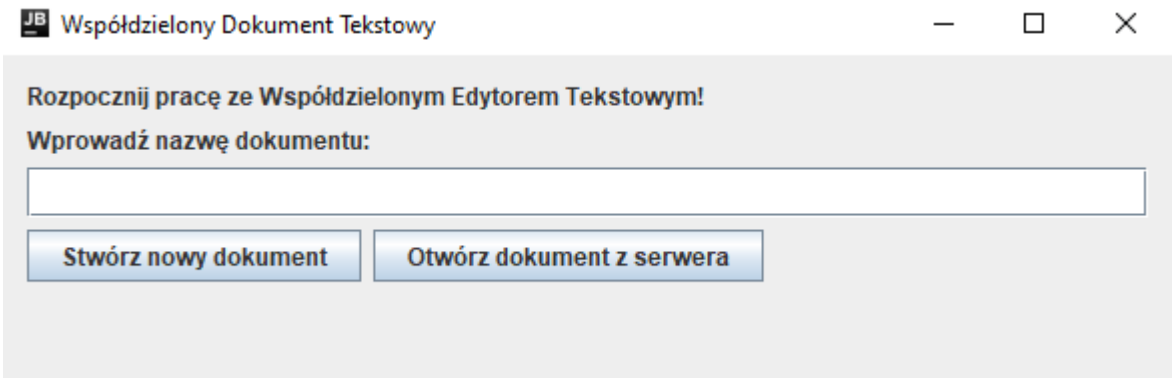
GUI – rys. 1

Po wpisaniu prawidłowych danych przez użytkownika, zostaje on przełączony do widoku, w którym musi podać swoją nazwę użytkownika (rys. 2). Nazwa ta jest następnie przechwytywana i przesłana do serwera.



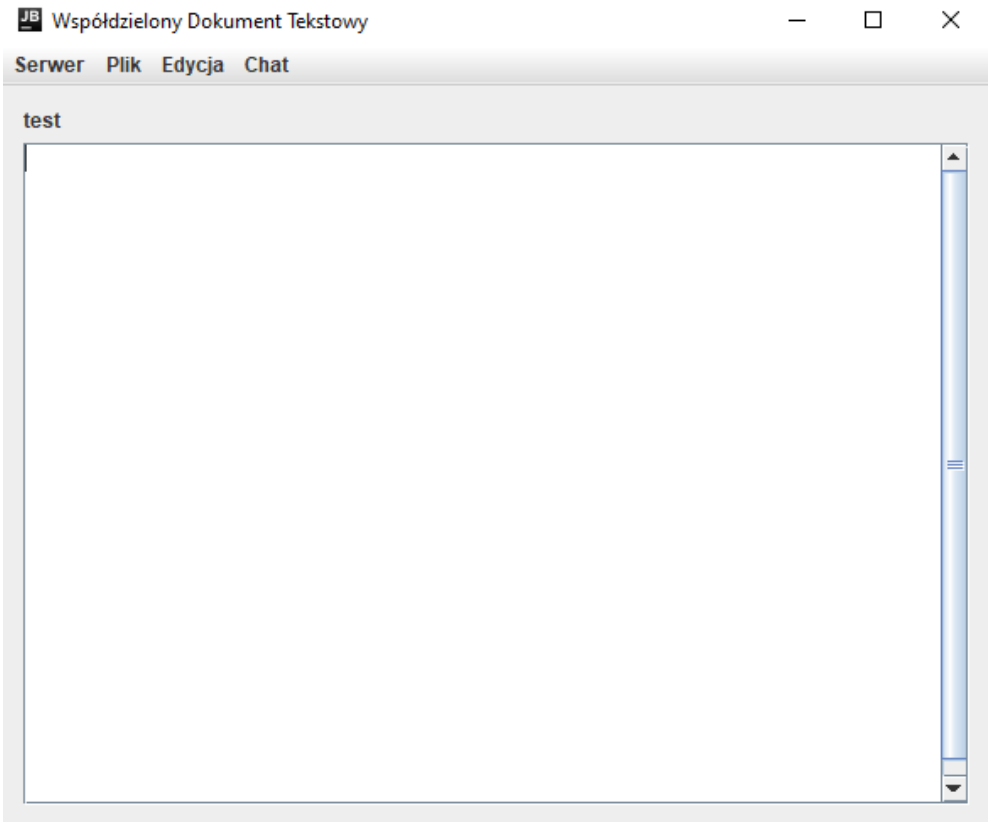
GUI - rys. 2

Następuje przełączenie użytkownika do okna z klasy *WelcomeView* (rys 3). Tutaj zostaje uruchomiony listener dla pola, w którym użytkownik wprowadza nazwę dokumentu. Sprawdza on poprawność podanej nazwy oraz tworzy nowy wątek WelcomeView dla nowego klienta, który przesyła wiadomość do serwera z informacją, że został utworzony nowy dokument tekstowy. W tym miejscu użytkownik otrzymuje także możliwość otwarcia istniejącego już dokumentu z serwera, jeśli jakiś został wcześniej na nim zapisany. Całość dotycząca tego fragmentu jest zaimplementowana w klasie *WelcomeView*.



GUI - rys. 3

Niezależnie od tego czy klient stworzy nowy dokument, czy też otworzy istniejący z serwera – zostaje przekierowany do widoku dokumentu zawartym w klasie *DocumentView* (rys. 4). To tutaj odbywa się edycja całego dokumentu.



GUI - rys. 4

Użytkownik otrzymuje szereg opcji, które mogą ułatwić mu współpracę z innymi oraz edytowanie tekstu. Opcje te zostały podzielone na cztery rozłączne kategorie:

1) Serwer – zawiera opcje dotyczące działań na serwerze, do którego jest podłączony klient.
    a. „Nowy” - utworzenie nowego pliku na danym serwerze (wysłanie wiadomości „new” do tegoż serwera),
    b. „Otwórz” – otworzenie istniejącego już pliku z danego serwera (wysłanie wiadomości „look” do serwera i wyświetlenie listy dokumentów, które znajdują się na serwerze),

Tutaj warto się chwilę zatrzymać, ponieważ do tego celu została utworzona oddzielna klasa *OpenDocumentDialog.* To właśnie ona obsługuje całe otwarcie dokumentu z serwera. Tworzony jest nowy obiekt klasy OpenDocumentDialog w postaci JOptionPane, z którego klient wybiera istniejący dokument. Wysyłana jest wiadomość „open” do serwera. Jeśli na serwerze nie istnieje jeszcze żaden dokument, to wyświetlony zostaje błąd w postaci komunikatu informującego o braku jakichkolwiek dokumentów w serwerze.

    c. „Wyjdź” – zakończenie pracy klienta i odłączenie się od serwera (dodatkowe wyświetlenie potwierdzenia wyjścia).

2) Plik – zawiera dwie opcje dotyczące działań na lokalnej maszynie.
    a. „Zapisz” – opcja umożliwiająca zapisanie obecnego stanu dokumentu na lokalnym komputerze,
    b. „Wczytaj – opcja umożliwiająca wczytanie pliku tekstowego do aplikacji i edytowania go w niej.

3) Edycja – podstawowe opcje dotyczące pracy z tekstem.
    a. „Kopiuj” – kopiowanie zaznaczonego fragmentu tekstu,
    b. „Wytnij” – wycinanie zaznaczonego fragmentu tekstu,
    c. „Wklej” – wklejanie wcześniej zaznaczonego tekstu we wskazane miejsce.

4) Chat – szereg opcji związanych z działaniem chatu (opisane w podrozdziale „Obsługa chatu”)
    a. „Chatuj” – uruchomienie chatu,
    b. „Rozłącz” – rozłączenie chatu.

Klasa, o której wcześniej nie było jeszcze mowy, to klasa *ExitWindow.* Jest to klasa słuchacza okien, która obsługuje każde zamknięcie. Wysyła do serwera wiadomość „bye” podczas zamykania serwera.

Więcej szczegółów (w tym między innymi szczegółowy opis zaimplementowanych klas i funkcji) znajduje się w plikach źródłowych pakietu *gui*, na które składają się: *ConnectView.java, DocumentView.java, ExitWindow.java, MainWindow.java, OpenDocumentDialog.java* i *WelcomeView.java*.

Komunikacja sieciowa

Fragment projektu nadzorowany przez Łukasza Pokorzyńskiego.

Komunikacja sieciowa w opisywanym projekcie była zdecydowanie największym wyzwaniem, szczególnie mając na uwadze niewielkie doświadczenie zespołu z implementacją funkcjonalności online.

Do tego modułu wykorzystane zostały istniejące w języku Java rozwiązania bazujące na socketach. Cała aplikacja jest zaprojektowana w modelu klient-serwer. Aby zacząć współpracę z innymi osobami, należy na jednej z maszyn uruchomić instancję serwera edytora tekstowego *ServerMain*, która automatycznie włącza się na porcie 50000 hosta. Po uruchomieniu serwer czeka na połączenie się pierwszego z użytkowników i operuje na ServerSocket z pakietu java.net języka Java. Służy do tego metoda *serve()* serwera, która stale nasłuchuje, czy nie pojawił się nowy klient chętny do połączenia.

Klasa *Server* w głównej mierze przetrzymuje dostępne dokumenty, ich wersje, listę wątków, czyli połączonych użytkowników, ich nazwy, zarządza wprowadzanymi zmianami do otwartego dokumentu tekstowego z użyciem klasy *EditManager*. Dodatkowo obsługuje on wysyłanie wiadomości do klientów, jeżeli takie dotyczą wszystkich podłączonych użytkowników.

Każdy użytkownik w serwerze ma przypisywany osobny wątek, który jest przedstawiony przez klasę *OurThreadClass*. Klasa ta głównie za zadanie ma utrzymywanie połączenia z serwerem i przetwarzaniem wiadomości protokołu, czyli obsługą zgłaszanych zdarzeń - odpowiednio dwie metody

handleConnection() i *handleRequest(),* z czego szczególnie ważna jest ta druga. To ona właśnie w głównej mierze jest odpowiedzialna za akceptowanie wprowadzanych zmian w pliku tekstowym, tworzenia nowych dokumentów. Ma ona zdefiniowane działania w przypadku:

- Dodania nazwy użytkownika do aktualnie połączonych, jeżeli dana nazwa użytkownika jest zajęta, nie jest ona dodawana i jest zgłaszany odpowiedni komunikat - zgłoszenie “name”,
- Podejrzenia aktualnie utworzonych dokumentów, jeżeli żaden nie jest utworzony, użytkownik jest o tym powiadamiany – zgłoszenie “look”,
- Otwarcia istniejącego pliku na serwerze - zgłoszenie “open”,
- Utworzenie nowego dokumentu tekstowego - zgłoszenie “new”,
- Wprowadzania zmian do otwartego u klienta pliku tekstowego, zgłoszenie “change” - ta sekcja dzieli się na dwie możliwości, które są obsługiwane w sekcji krytycznej:
    - insert – wprowadzanie nowych znaków do dokumentu,
    - remove – usuwanie tekstu z pliku tekstowego.

Do poprawnej obsługi zgłoszenia remove muszą być zdefiniowane: nazwa dokumentu, jej aktualna wersja oraz pozycja początkowa i końcowa offsetu. W przypadku zgłoszenia insert są to wprowadzane znaki i pozycja początkowa offsetu.

- Odłączenie się użytkownika - zgłoszenie “bye”.

Dodatkowo po stronie klienta, jeżeli użytkownik pracuje na nieaktualnym dokumencie, to dokonywana jest jego aktualizacja przed wysłaniem żądania.

Każda wprowadzona edycja jest zapisywana jako obiekt klasy *Edit* i następnie wysyłana do instancji klasy *EditManager* przetrzymywanej w serwerze. Przetwarza on edycję pobierając kolejno obiekty *Edit* z kolejki FIFO, zapisując ją w logach i gdy jest to potrzebne, koryguje offset. Przetwarzanie następuje dość szybko, zatem buforowanie obiektów *Edit* jest niezauważalne.

Integralna częścią obsługi sieciowej jest klient, który może połączyć się do instancji serwera. Klient jest uruchamiany przez plik *ClientMain*. Jedyne, co wspomniany plik robi, to tworzy obiekt klasy *MainWindow* i ustala go jako widoczny. Przebieg kolejnych okien został wytłumaczony w rozdziale wyżej na temat GUI, więc nie będzie omawiany szczegółowo w tym miejscu. Ważne, że w klasie *MainWindow* przetrzymywany jest obiekt klasy *Client, który jest dla nasz istotny podczas komunikacji sieciowej.*

Client jest podstawową klasą potrzebną do pracy, bowiem przetrzymuje ona informacje na temat aktualnie otwartego dokumentu i jego treści, ma określone gniazdo, jego port, IP hosta oraz ma swój własny obiekt *ClientActionServer*, który komunikuje się z serwerem. Działanie tej klasy jest proste – definiuje ona gniazdo dla użytkownika, które później posłuży do wysyłania wiadomości zgłoszeń do serwera. Innymi funkcjami jest ustawienie pseudonimu użytkownika, przypisanie głównego okna.

Z kolei *ClientActionServer* jest klasą odpowiedzialną za przetwarzanie wiadomości protokolarnych przekazywanych od serwera. Są to wiadomości o dokonaniu (lub nie) pewnych zgłoszeń wysłanych wcześniej przez klasę *Client*. Obsługiwane są odpowiednie działania:

- Powiadomienie użytkownika o wystąpieniu błędu - otwierane jest okno błędu wraz z przekazana od serwera wiadomością - powiadomienie “error”,
- Jeżeli użytkownik wyśle chęć wyświetlenia wszystkich utworzonych na serwerze dokumentów, to dostanie wiadomość zwrotną “alldocs” i wyświetlone zostanie okno wyboru dokumentu,
- Przy logowaniu klient otrzyma wiadomość zwrotną o poprawnym dodaniu pseudonimu do listy, a klient przypisze odpowiedni pseudonim do pola userName – powiadomienie “name”,

- Gdy klient utworzy nowy dokument na serwerze, to dostanie wiadomość zwrotną, że operacja powiodła się, a widok klienta zmieni się na widok edycji dokumentu – powiadomienie “new”,
- Przy otwarciu istniejącego dokumentu w kliencie zmieniana jest nazwa aktualnie edytowanego dokumentu, zostanie mu przekazana jego treść i otwarte zostanie okno edycji – powiadomienie “open”,
- Przy poprawnej edycji dokumentu klient otrzymuje potwierdzenie jaki tekst został wprowadzony, w jakim dokumencie i w którym miejscu nastąpiła zmiana oraz przez kogo została wprowadzona – powiadomienie “change”.

W budowie protokół ten przypomina protokół używany przy okazji obsługi zgłoszeń po stronie serwera.

Obsługa chatu

Fragment projektu nadzorowany przez Patryka Jana Sozańskiego.

Prace nad implementacją tego fragmentu kodu były wykonywane równolegle z tworzeniem funkcjonalności dokumentu tekstowego i niezależnie od niego, dlatego do pewnego stopnia pakiet obsługujący funkcjonalności czatu stanowi samodzielną jednostkę, która jest możliwa do wdrożenia przy innych projektach.

Podstawą do komunikacji między poszczególnymi użytkownikami chatu stanowi, podobnie jak w przypadku dokumentu tekstowego, architektura klient-serwer z wykorzystaniem gniazd (socket). W celu ustanowienia połączenia niezbędne jest wcześniejsze utworzenie instancji serwera, który zapewnia usługi dla klientów zgłaszających do niego żądania obsługi.

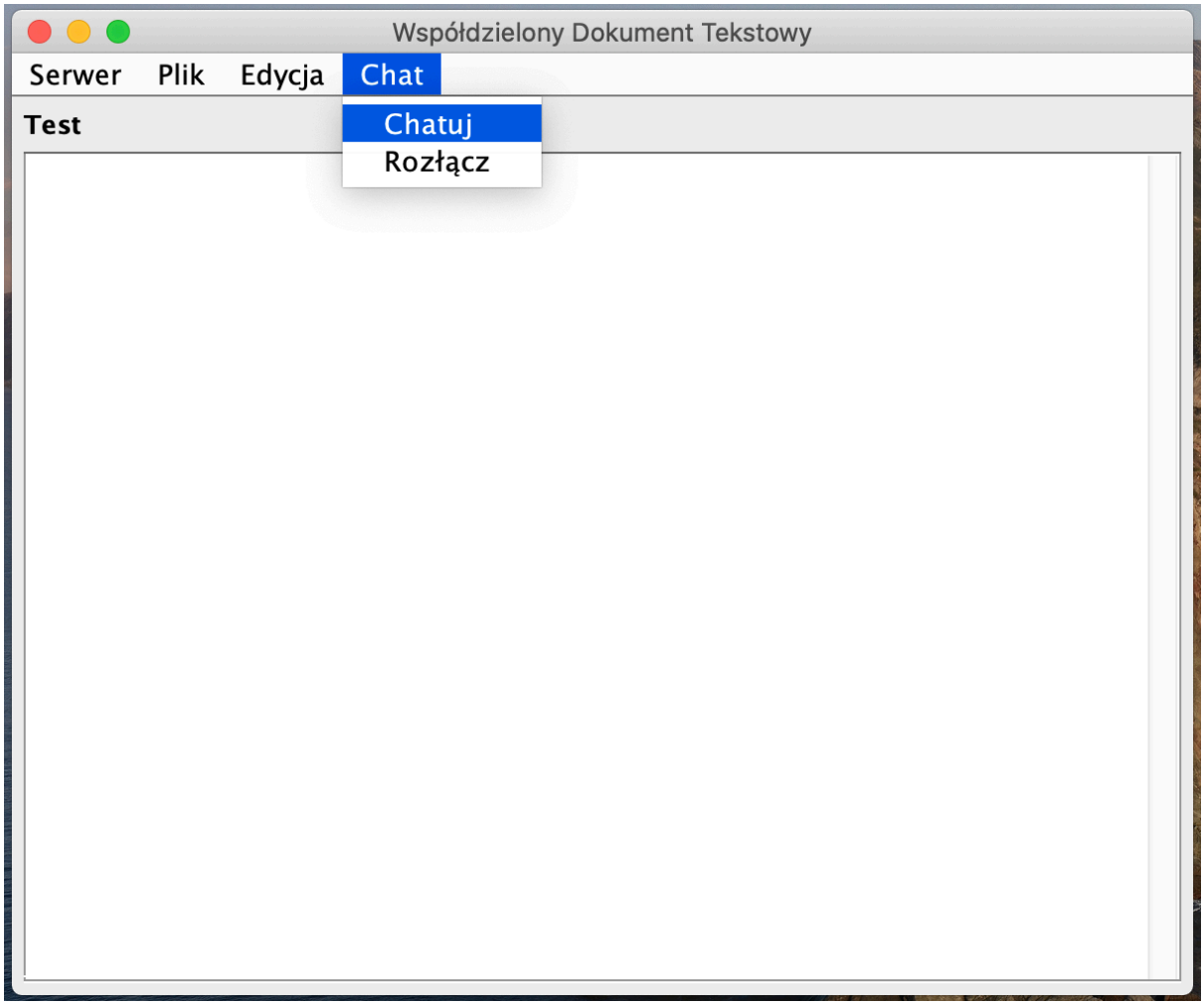
Obsługa użytkowników chatu została celowo odseparowana od obsługi użytkowników dokumentu tekstowego poprzez utworzenie dwóch odrębnych serwerów zapewniających usługi dla danego typu klientów. Takie rozwiązanie poniosło za sobą wiele korzyści.

Jedną z nich było zwiększenie przenośności kodu, jako że nie występują żadne ścisłe powiązania pomiędzy elementami pakietu *chat* i resztą kodu. De facto jedyne zależności zostały zaimplementowane w klasie *DocumentView* pakietu *gui*, gdzie dochodzi do utworzenia nowej instancji klasy *ChatView* pakietu *chat* dla konkretnego klienta dokumentu tekstowego.

Kolejnym pozytywnym aspektem tego rozwiązania była możliwość znacznego uproszczenia struktury kodu. Oba serwery zapewniają usługi dla klientów niezależnie od siebie, a więc forma zgłaszanych przez nich żądań obsługi mogła być całkowicie inna. Z tego powodu, że zarządzanie klientami chatu wymagało znacznie mniej skomplikowanej implementacji od zarządzania klientami dokumentu tekstowego, została wykorzystana całkowicie inna składnia i obsługa wiadomości wysyłanych i odbieranych przez elementy zapewniające komunikację sieciową tego pakietu.

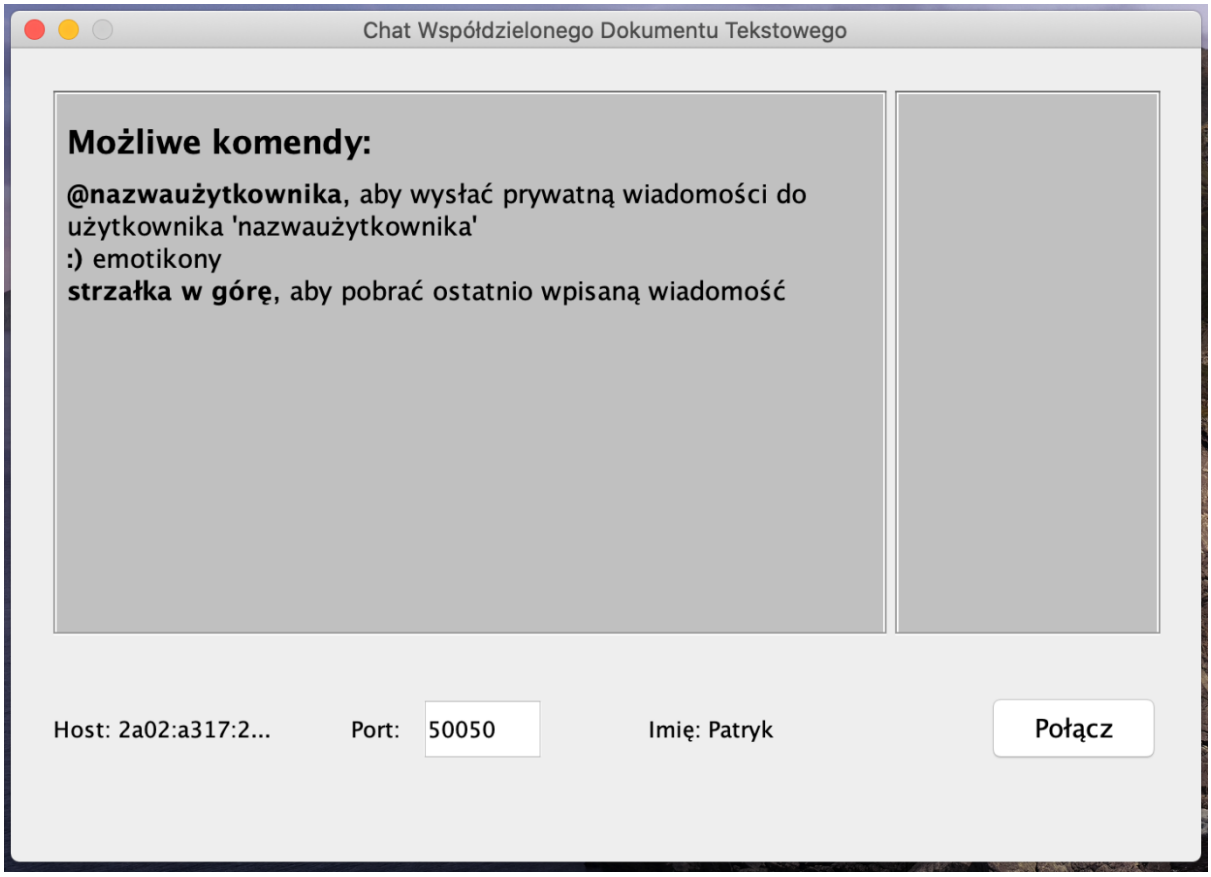
Dodatkowo usprawniło to funkcjonowanie programu, gdyż komunikacja między elementami tego pakietu nie jest zależna od stanu serwera dokumentu tekstowego. Dzięki temu, że oddzielny serwer zarządza ruchem sieciowym chatu, serwer dokumentu tekstowego został odciążony z tej części pracy. Tym samym komunikacja, która ma miejsce z wykorzystaniem jednego z serwerów, przebiega całkowicie bez wpływu na czynności wykonywane przez drugi serwer. Możliwe jest to do zaobserwowania na przykład, gdy serwer dokumentu tekstowego wykonuje wzmożoną pracę przy wczytywaniu dużego pliku, a komunikacja na chacie przebiega bez żadnych opóźnień.

Użytkownik, chcący skorzystać z funkcjonalności chatu, musi wybrać opcję „Chatuj” dostępną po najechaniu kursorem na pole „Chat” na pasku menu okna dokumentu tekstowego (rys. 1).



Obsługa chatu – rys. 1

Po wykonaniu tej operacji zostaje wyświetlone okno chatu z widokiem powitania (rys. 2).



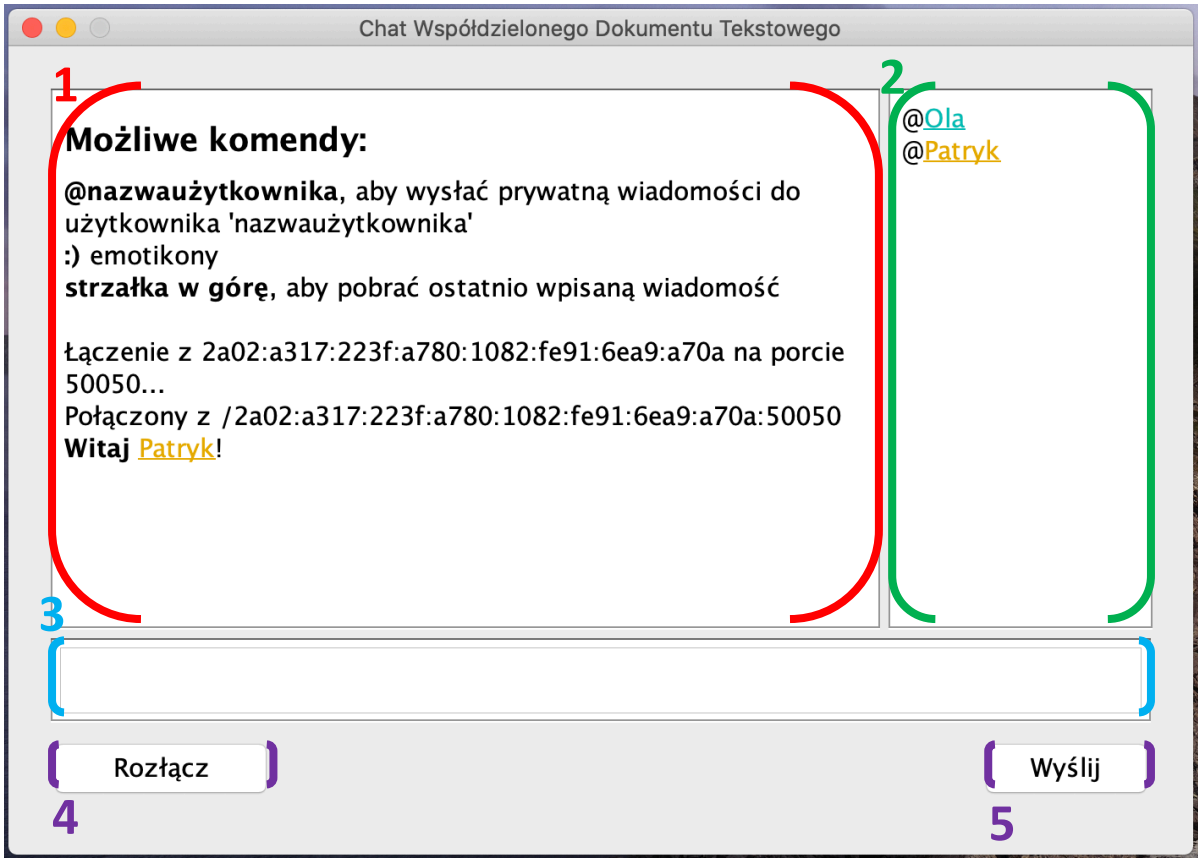
Obsługa chatu – rys. 2

Pozostaje ono nieaktywne (co reprezentowane jest przez szary kolor tła) do momentu, aż użytkownik nie ustanowi połączenia. Aby do tego doszło, niezbędne jest wprowadzenie odpowiednich danych serwera chatu w pola u dołu widoku.

Użytkownikowi prezentowane są 3 następujące pola:

- **Host** – stanowi adres IP serwera obsługującego chat, z którym użytkownik chce się połączyć. Zawartość tego pola nie jest możliwa do zmiany – przyjmuje ona wartość identyczną do tej, jaka została wprowadzona podczas łączenia do serwera dokumentu tekstowego;
- **Port** – stanowi numer portu serwera obsługującego chat, z którym użytkownik chce się połączyć. Zawartość tego pola domyślnie przyjmuje wartość 50050, ale jest możliwa do zmiany w zależności od tego, na jakim porcie został utworzony docelowy serwer chatu;
- **Imię** – stanowi imię/nick użytkownika, który chce się połączyć z chatem. Zawartość tego pola nie jest możliwa do zmiany – przyjmuje ona wartość identyczną do tej, jaka została wprowadzona podczas łączenia do serwera dokumentu tekstowego.

Zatwierdzenie zawartości powyższych pól następuje poprzez kliknięcie na przycisk „Połącz” widoczny obok, po czym następuje próba połączenia ze wskazanym serwerem. W przypadku sukcesu dochodzi do zmiany widoku i zostaje wyświetlona wiadomość powitalna (rys. 3).

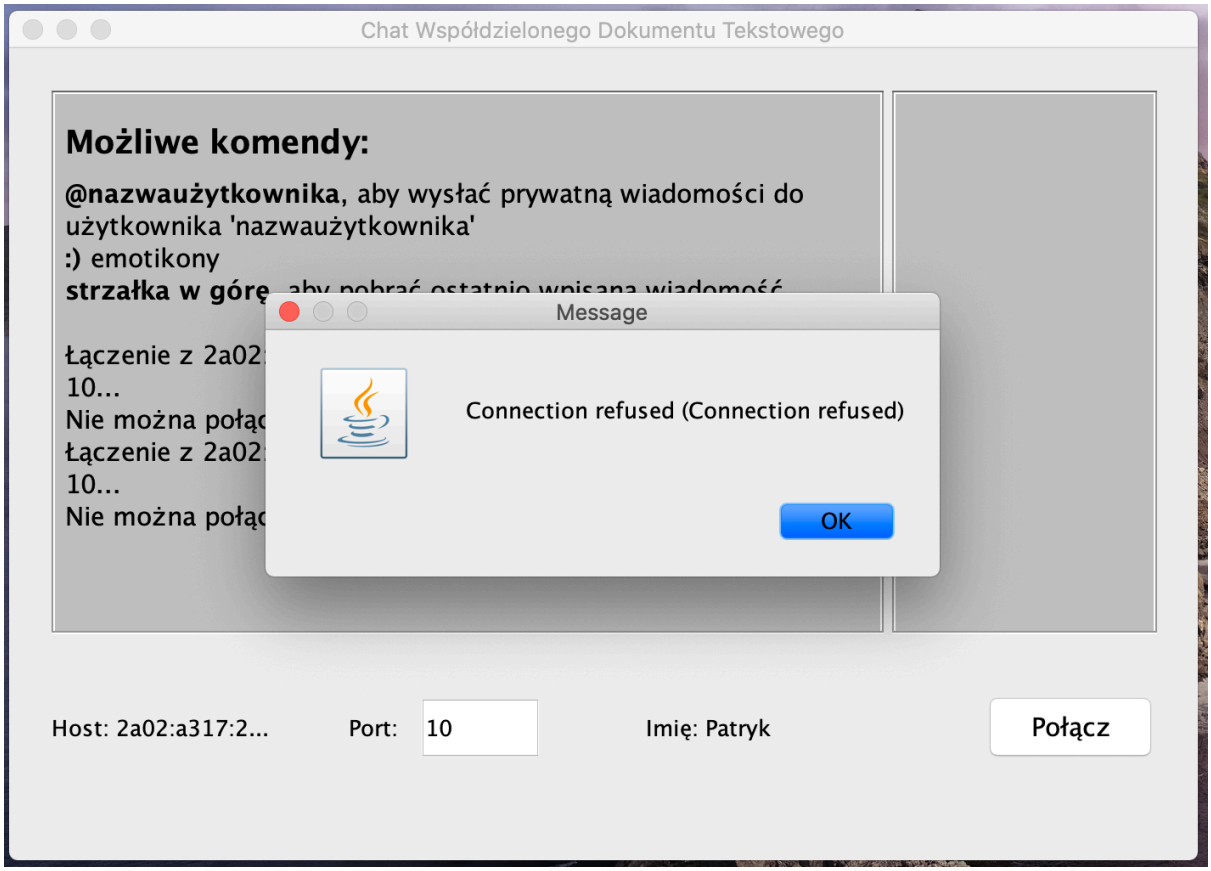


Obsługa chatu – rys. 3

Widok aktywnego okna chatu składa się z następujących elementów:

1. – okno wiadomości – wyświetla krótką instrukcję korzystania z chatu, informacje dotyczące połączenia oraz wiadomość powitalną,
2. – lista użytkowników połączonych z chatem – wyświetla listę użytkowników, którzy w danym momencie korzystają z funkcjonalności chatu,
3. – bufor wiadomości – stanowi pole, do którego użytkownik wpisuje treść swojej wiadomości dla innych użytkowników chatu,
4. – przycisk służący do rozłączenia z chatem – po jego wciśnięciu następuje przekazanie odpowiedniego żądania usługi do serwera i dochodzi do rozłączenia użytkownika,
5. – przycisk służący do wysłania wiadomości (alternatywnie klawisz ENTER) – po jego wciśnięciu zawartość bufora wiadomości zostaje przekazana do serwera z odpowiednim żądaniem usługi.

Natomiast w przypadku, gdy nie jest możliwe połączenie ze wskazanym serwerem, wyświetlany zostaje odpowiedni komunikat błędu (rys. 4). Powodem tej sytuacji może być wprowadzenie błędnego numeru portu przez użytkownika, próba połączenia z nieistniejącym serwerem lub problemy związane z połączeniem sieciowym pomiędzy użytkownikiem a serwerem.



Obsługa chatu – rys. 4

Więcej szczegółów (w tym między innymi opis zaimplementowanych klas i funkcji) znajduje się w plikach źródłowych pakietu *chat*, na które składają się: *ChatServer.java*, *ChatView.java*, *ColorInt.java*, *ReceivedMessagesHandler.java*, *User.java* oraz *UserHandler.java*.

Testowanie

Testy jednostkowe są oparte o sprawdzenie działania strony serwerowej, a dokładniej działania protokołu obsługującego zgłoszenia napływające do serwera. Wszystkie testy są uruchamiane przez włączenie pliku *TestMain,* korzystający z pliku *ServerTests*.

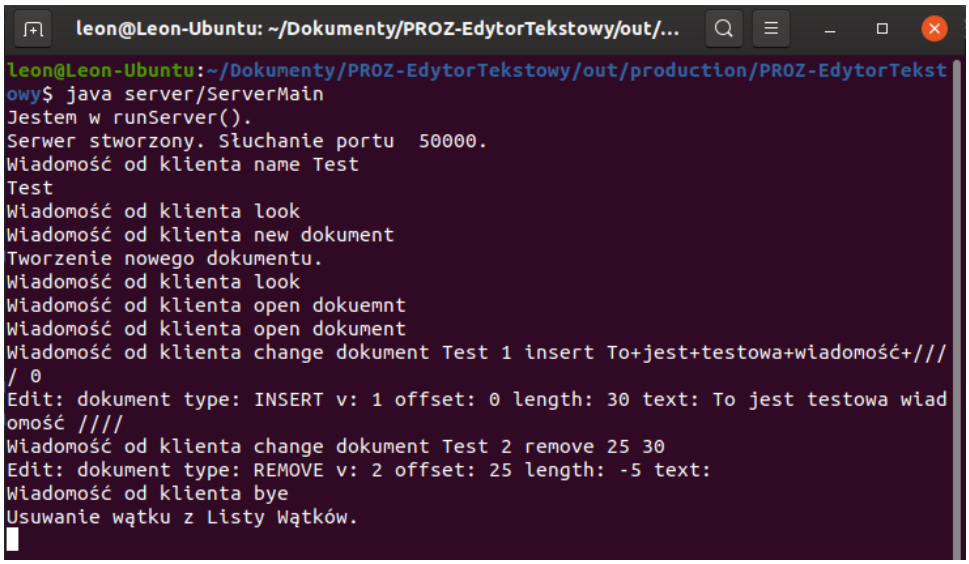
Scenariusz testów przebiega następująco:

1. Przede wszystkim przed uruchomieniem testów należy utworzyć instancję serwera na maszynie testującej.
2. Gdy uruchomienie serwera powiedzie się, należy utworzyć instancję *TestMain* i uruchomić ją w trybie "*server*". Od tej pory wszystkie testy przebiegają automatycznie.
3. Pierwszy test polega na próbie utworzenia wątku w serwerze przez zdefiniowanie nowego gniazda dołączającego się do serwera. W przypadku sukcesu wyświetla się komunikat o poprawnym utworzeniu gniazda, a co za tym idzie – utworzenia nowego wątku do obsługi napływających z tego gniazda komunikatów i zgłoszeń.
4. Następnie jest przeprowadzany prosty test dodania pseudonimu przez bezpośrednie wysłanie prośby "*name Test*". W przypadku niepowodzenia wyświetlany jest komunikat, że dana nazwa użytkownika już istnieje, bądź nie udało się go dodać do listy.

5. Następny test sprawdza działanie żądań “*look*” do podglądu utworzonych na serwerze dokumentów oraz “*new document*”, który przy powodzeniu powinien utworzyć nowy dokument tekstowy do edycji na serwerze.
6. Kolejny test sprawdza obsługę otwarcia dokumentu. Na początku sprawdzana jest obsługa wysłania niepoprawnej nazwy dokumentu “*open dokuemnt*”, która powinna zwrócić komunikat o nieistnieniu danego dokumentu. Dopiero potem sprawdzamy działanie “*open dokument*”, które z założenia, po uprzednim utworzeniu dokumentu, powinno się powieść.
7. Przedostatni test skupia się na obsłudze edycji wprowadzanych do dokumentu. Na początek wprowadzany jest tekst “*To jest testowa wiadomość ////*”, a potem frontslashe z końca wprowadzonego tekstu są usuwane zgłoszeniem “*remove*”. Są to odpowiednio zgłoszenia “*change dokument Test 1 insert To+jest+testowa+wiadomość+//// 0*” oraz "*change dokument Test 2 remove 25 30*"
8. Ostatni test jest prostym wysłaniem zgłoszenia “*bye*”, które kończy połączenie z serwerem i usuwa login z listy połączonych użytkowników.

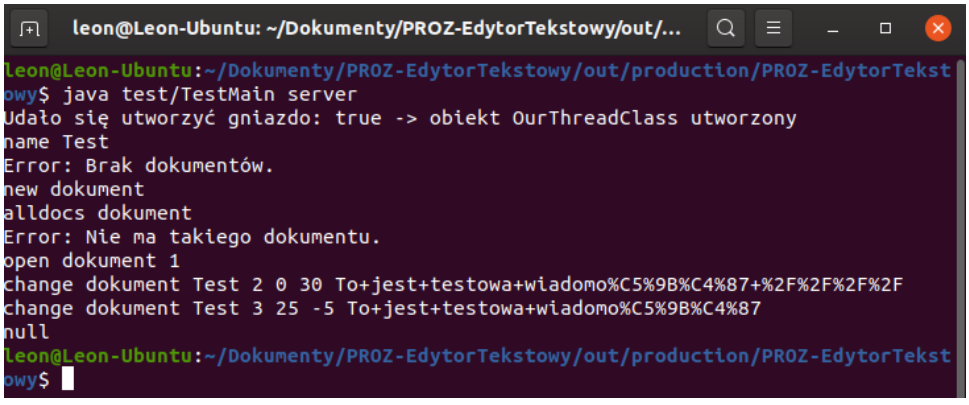
Wyniki powyższych testów można zobaczyć poniżej.

Terminal serwerowy:



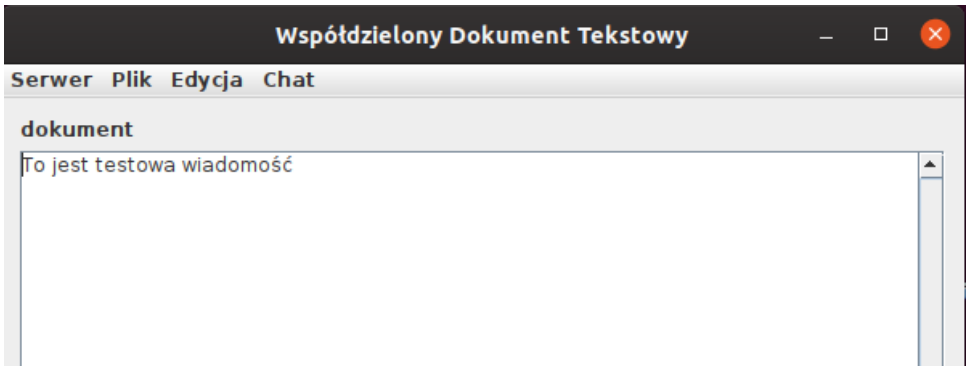
Testowanie - rys. 1

Terminal z przeprowadzanymi testami:



Testowanie - rys. 2

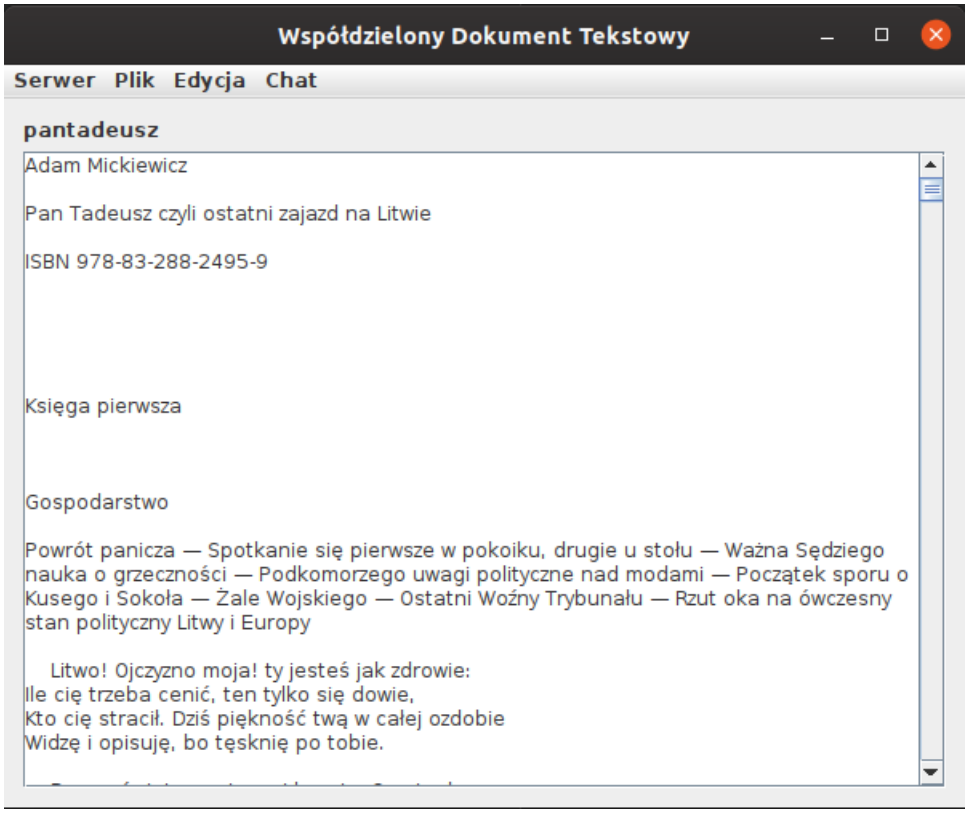
Wyniki widoczne po zalogowaniu do serwera:



Testowanie - rys. 3

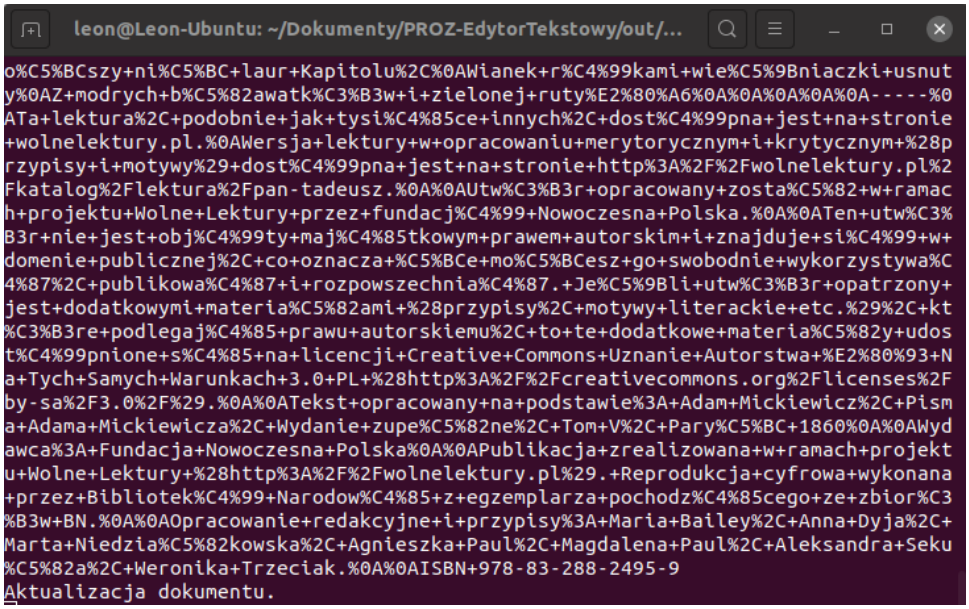
Testy GUI oraz klienckie postanowiliśmy przeprowadzić manualnie z powodu większego ich skomplikowania. Samo GUI w przeprowadzonych przez nas próbach działało prawidłowo, nic nie potrzebowało większych poprawek bądź reworków.

W innym teście postanowiliśmy sprawdzić przetwarzanie dużego objętościowo pliku podczas wczytywania. Do tego celu posłużył nam plik PanTadeusz.txt, który zawierał w sobie całą treść dzieła. Wgrywanie wspomnianego pliku do dokumentu tekstowego działo się mniej więcej przez 2-3 sekundy i zakończyło się powodzeniem – nie zostały zgubione żadne znaki, a wszystkie polskie wczytały się prawidłowo (tzn. nie wystąpiły żadne “krzaczki” w ich miejscu).



Testowanie - rys. 4

Widok w konsoli serwera:

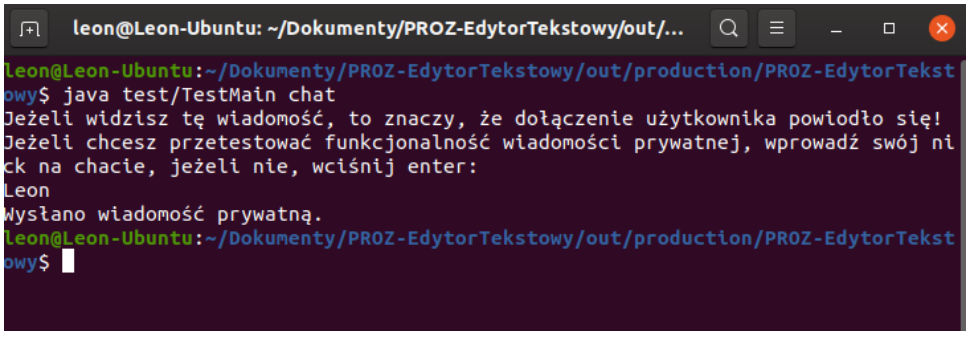


Testowanie - rys. 5

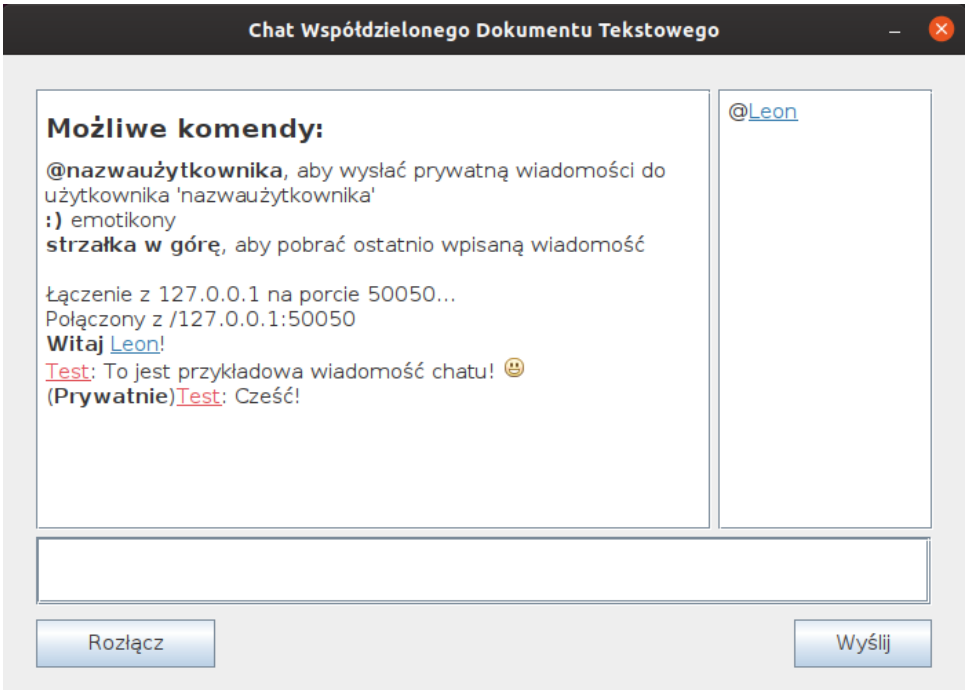
Testy chatu również zostały wykonane automatycznie, lecz do ich przeprowadzenia wymagane było włączenie instancji serwera, klienta oraz serwera chatu. W przypadku włączenia jedynie serwera chatu testy są przeprowadzane, ale nie widać ich w akcji w 100%. Scenariusz tych testów jest bardzo prosty:

1. Po włączeniu wszystkich wymaganych komponentów włączamy *TestMain* w trybie "*chat*".
2. Najpierw tworzony jest nowy użytkownik testowy. Jeżeli test dołączenia powiedzie się, wyświetlany jest odpowiedni komunikat.
3. Następny test skupia się na funkcjonalności wysyłania wiadomości. Najpierw jest wysyłana zwykła wiadomość do wszystkich. Następnie tester proszony jest o wprowadzenie nicku z chatu, do którego ma być wysłana wiadomość prywatna. Jeżeli nick nie zostanie wprowadzony/będzie wprowadzony nieprawidłowy, to wiadomość nie ukaże się na chacie.
4. Testy kończą się.

Efekty można zobaczyć na poniższych obrazach:



Testowanie - rys. 6



Testowanie - rys. 7

Wnioski, spostrzeżenia oraz analiza

Praca w grupie

Wszystkie nasze postępy zostały udokumentowane przy użyciu najpopularniejszego systemu kontroli wersji – GitHuba. Link do naszego repozytorium: https://github.com/justleon/PROZ-EdytorTekstowy/tree/master.

Ze względu na obecną sytuację kontaktowaliśmy się głównie poprzez komunikatory internetowe, co znacznie utrudniło i przedłużyło naszą pracę. Niemniej jednak udało się nam postępować zgodnie z harmonogramem, który ustaliliśmy na początku i każdy wykonywał swoje zadanie. Nie było możliwe jednaka, aby poprowadzić swoją część od początku do końca samodzielnie, ponieważ wszystkie z pakietów w mniejszym lub większym stopniu współpracują ze sobą. Dlatego często konsultowaliśmy się między sobą, aby każdy mógł dopasować poniekąd swoją część do pozostałych.

Dzięki temu nauczyliśmy się, że praca w zespole to ciągłe rozmowy, konsultacje i zmiany swojego kodu dla innym fragmentów tworzonych w tym samym czasie przez innych. Dodatkowo rozwinęliśmy w sobie umiejętność pójścia na kompromis, gdy któreś z nas chciało zmian, których pozostali nie akceptowali.

Rozwój

Projekt ten był dla nas wyzwaniem głównie dlatego, że nigdy wcześniej nie mieliśmy styczności z implementacją modułów komunikujących się ze sobą za pomocą sieci. Nie mieliśmy doświadczenia w programowaniu w języku Java, ale to nie stanowiło większego problemu, ponieważ wcześniej

mieliśmy do czynienia z wieloma innymi językami programowania. Dzięki temu projektowi nauczyliśmy się jak zrealizować graficzny interfejs użytkownika (GUI) w sposób zdarzeniowy, jak umożliwić komunikację sieciową oraz jak napisać podstawowe testy do tego typu aplikacji.

Możliwości dalszego rozwoju projektu

Jak już zostało wspomniane powyżej, chat i dokument tekstowy stanowią oddzielne segmenty. Można je dekomponować oraz łączyć z innymi. Taki chat można wykorzystać do zwykłej komunikacji sieciowej między użytkownikami, do współdzielonego arkusza kalkulacyjnego, współdzielonej tablicy do rysowania, itp. W drugą stronę – edytor tekstowy można odłączyć od sieci, aby działał samodzielnie jako zwykła, lokalna aplikacja.

Możliwości dalszego rozwoju projektu jest bardzo wiele. Można ulepszać go biorąc pod uwagę zarówno zaplecze techniczne, jak i interfejs graficzny całej aplikacji np. przy użyciu języka HTML. Jeśli chodzi o pierwszą kategorię śmiało można pokusić się o dodanie obsługi innych formatów tekstowych oraz przetwarzanie obrazów i linków. Od strony graficznej można dodać szereg opcji związanych z formatowaniem tekstu: wyrównanie do lewej lub prawej strony, wyśrodkowanie lub justowanie tekstu. Dodatkowo można ulepszyć wygląd i funkcjonalność całego edytora poprzez dodanie różnych szablonów zawierających całą paletę kolorów tekstu oraz ciekawe czcionki.