DOKUMENTACJA

Karol Orłowski, Marcel Jezierski, Maksymilian Krysa, Jakub Kochanowski
prowadzący: mgr inż. Konrad Grochowski

Opis funkcjonalności użytkowych:

1. Projekt powinien realizować proste funkcjonalności koedytora tekstu.
2. Możliwość tworzenia nowych dokumentów, tak aby inni użytkownicy mieli do nich dostęp.
3. Możliwość edycji dokumentów znajdujących się na serwerze przez wielu użytkowników naraz.
4. Poprawna synchronizacja dokumentów edytowanych przez użytkowników.
5. Składowanie dokumentów na serwerze przez czas określony przez administratora serwera.

Opis infrastruktury:

Projekt składa się z 3 współpracujących ze sobą modułów zaprojektowanych pod systemem Linux. Pierwszym z nich, i jednocześnie najważniejszym, jest moduł serwera. Serwer jest odpowiedzialny za składowanie i synchronizację dokumentów edytowanych przez klientów. Kolejnym z modułów jest klient „Desktopowy”. Obsługuje on możliwości wyświetlania dokumentów znajdujących się na serwerze i ich edycji. Udostępnia on również możliwość tworzenia nowych dokumentów, tak aby inni użytkownicy koedytora mogli z nich korzystać. Trzecim z modułów jest klient „Webowy” działający na większości przeglądarek webowych. Jego funkcjonalności są podobne jak w przypadku klienta Desktopowego.

Założenia programistyczne:

1. Każdy z klientów może edytować maksymalnie jeden plik naraz.
2. Komunikacja klientów z serwerem jest niezależna od typu klienta.
3. Jednorazowo można dodać do dokumentu 4kB znaków. (wklejanie dużych porcji tekstu)
4. Usuwanie zaznaczonego tekstu jest realizowane w ściśle określony sposób

Serwer - (C++)

Infrastruktura serwera jest dość złożona. Zaczynając od składowania danych są one przechowywane w plikach o rozszerzeniu „.txt” w folderze „Files”. Gdy klient prosi o dostęp do konkretnego pliku serwer buforuje dany plik do struktury DATABASE funkcją SHEET bufferFile() jeżeli jeszcze go tam nie ma. Po tym jak żaden z klientów nie modyfikuje już danego pliku wpisujemy ostatnią wersję pliku jaki stworzyli klienci i buforujemy ją na dysk serwera funkcją void updateFile(). Taki sposób komunikacji sprawia, że znacznie szybciej następuje rozesłanie aktualnego dokumentu do wszystkich klientów. Serwer, aby mieć pełną kontrole nad tym kiedy i jak przyjmować komunikaty od wielu użytkowników, dla każdego klienta uruchamia wątek. Dany wątek serwera nasłuchuje wiadomości od klienta podłączonego na konkretnym porcie.
Komunikacja klient-serwer następuje jak poniżej:
Wiadomości pomiędzy klientem, a serwerem mają formę ciągów znaków string. Klient wysyła wiadomość o aktualizacji dokumentu na socket wątku do którego jest przypisany, ten zaś dane zapytanie kieruje do kolejki wiadomości messageQueue. Proces "matka" serwera następnie analizuje pierwsze zapytanie z kolejki messageQueue , obsługuje to zapytanie i rozsyła wprowadzone zmiany do wszystkich zainteresowanych klientów (wszystkich działających na danym pliku). Innego typu zapytania niż aktualizacje dokumentu obsługują wątki "dzieci" bezpośrednio z klientami. Rodzaje komunikatów możliwych do obsługi przez serwer są opisane w poniższej tabeli.

Rodzaje wiadomości wysyłanych pomiędzy klientem, a serwerem		
Wiadomość klienta	Działanie serwera	Odpowiedź serwera
‘N’+nazwa pliku	Tworzenie nowego, pustego dokumentu tekstowego w folderze „Files”. Jeżeli plik o takiej nazwie już istnieje to brak działania.	-
‘UP’	Przygotowanie listy dokumentów w postaci „pierwszydocu.txt\ndrugidoc.txt\ntrzecidoc.txt ...”	Lista dokumentów do ubiegającego się klienta
‘G’+nazwapliku	Dodanie danego klienta jako korzystającego z pliku ‘nazwapliku’. Jeżeli nikt nie korzysta z danego pliku to go buforujemy do struktury DATABASE	Ciąg znaków reprezentujący zawartość pliku ‘nazwapliku’ na serwerze.
‘UG’	Usunięcie danego klienta jako korzystającego z pliku z którego aktualnie korzystał. Jeżeli był to ostatni klient korzystający z danego pliku to plik ten jest przenoszony ze struktury danych DATABASE na dysk do folderu Files	-
‘Z’+’poczatkowy index wierszy.poczatkowy index kolumn’+’końcowy index wierszy.końcowy index kolumn:’+’wprowadzona zmiana’	Wątek dziecko wysyła informacje o zmianie w danym pliku do kolejki obsługiwanej przez proces matki. Proces matka wprowadza daną zmianę w życie	Informacja o danej zmianie do wszystkich klientów(włącznie z tym, który daną zmianę wprowadził), którzy operują na danym pliku
Żadna z powyższych opcji	Brak działania	-

Opis struktury DATABASE
Struktura DATABASE jest strukturą wielowymiarową przechowującą większość potrzebnych serwerowi informacji. Na najwyższym wymiarze jest to mapa hashująca struktury DOCK gdzie kluczem jest nazwa pliku. Struktura DOCK jest to para struktury SHEET reprezentującej plik w której reprezentacją linii jest wektor znaków LINE, a nieuporządkowanym zbiorem LISTENERS reprezentującym podłączonych do danego pliku Klientów.

Opis funkcji serwera:

```
void readDocumentNames(const string&, deque<string>&);
```

funkcja pobierająca nazwy plików w Folderze Files i wynik wpisywująca do string'a

```
void *connection_handler(void *);
```

funkcja, która jest ciałem wątku 'dziecko'; obsługuje ona wiadomości od klienta

```
void *listening(void*);
```

funkcja, która jest ciałem wątku 'matka'; obsługuje ona wykonywanie kolejnych wiadomości w kolejce messageQueue

```
SHEET bufferFile(string);
```

funkcja buforująca plik o nazwie podanej jako jej argument i zwracająca strukture SHEET, która jest reprezentacją wnętrza tego pliku tekstowego

```
void updateFile(string, SHEET)
```

funkcja aktualizująca plik na serwerze o nazwie podanej jako argument(string) na podstawie aktualnej wersji w strukturze SHEET

```
void add(DATABASE &, string, string);
```

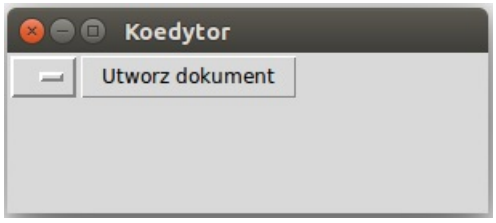
funkcja modyfikująca strukturę SHEET w DATABASE o nazwie pliku, która jest przekazywana jako drugi argument tej funkcji. Trzeci argument jest to informacja o zmianie jaką trzeba wprowadzić.

Aby poprawnie wyłączyć serwer, po naciśnięciu Ctrl+C, wykorzystując tablice aktualnie działających wątków „dzieci", powiadamiamy klientów o wyłączeniu serwera i zamykamy poprawnie wątki „dzieci".

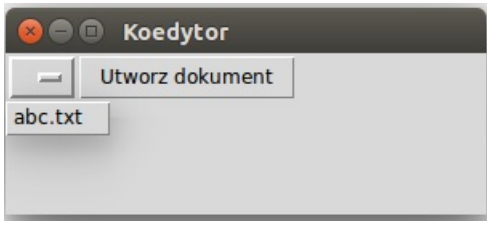
Klient Desktop(Python)

Obsługa:

Po uruchomieniu skryptu runDesktopClient.sh następuje wyświetlenie się następującego okna:

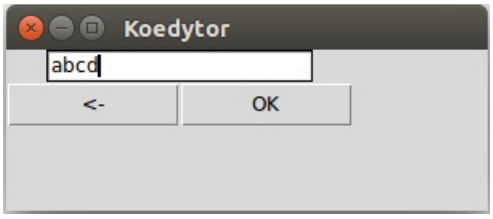


Widzimy w nim przycisk „Utwórz dokument” oraz rozwijaną listę dokumentów aktualnie znajdujących się na serwerze. Naciskając na rozwijaną listę widzimy następującą zmianę.

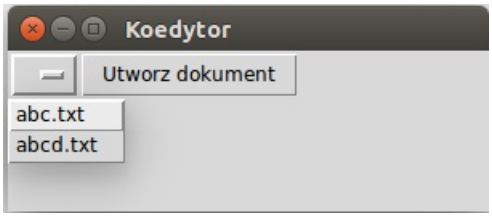


Okazuje się, że na serwerze znajduje się jeden plik „abc.txt” . Naciskając na jego nazwę możemy otworzyć jego zawartość i rozpocząć edycję.

Naciskając na czerwony krzyżyk w lewym, górnym rogu powrócimy do głównego menu. Teraz zobaczmy co robi przycisk „Utwórz dokument”



Widzimy dwa przyciski „←” oraz „OK” oraz pole tekstowe. Wpisując w nie nazwę nowego pliku oraz klikając „OK” na serwerze stworzy się nowy plik o nazwie, w naszym przypadku - „abcd.txt”.



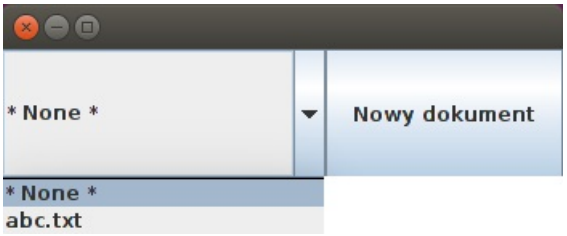
Widzimy, że nowo utworzony plik znalazł się na serwerze.

Jako bibliotekę graficzną Pythona wykorzystaliśmy Tkinter.

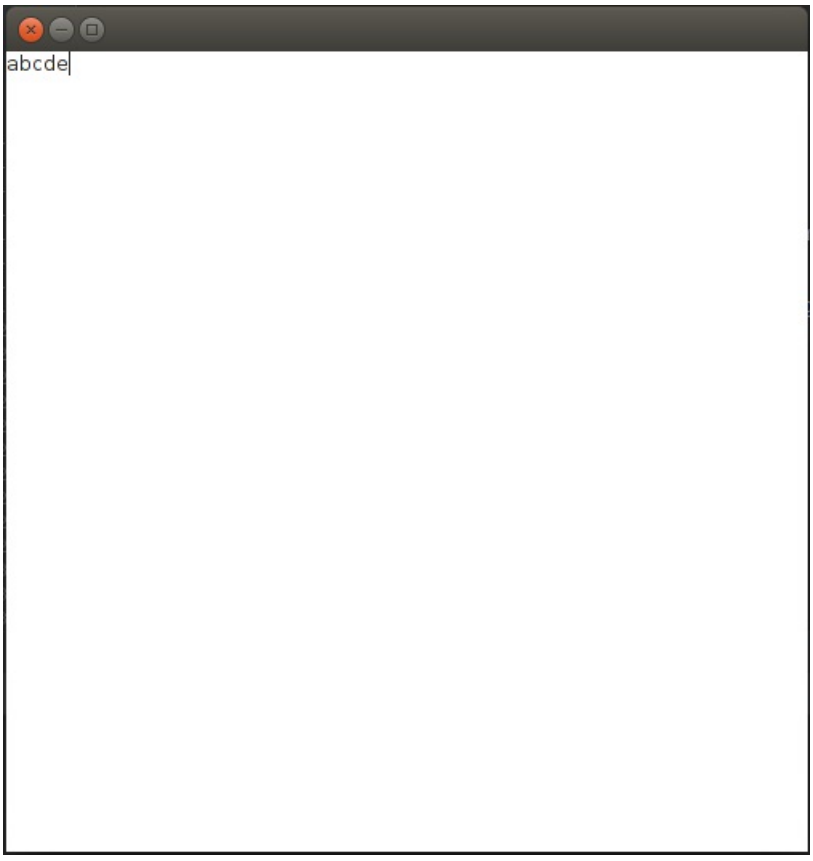
WebClient(Java+Webswing)

Po uruchomienia serwera tomcat, postawieniu na nim aplikacji, otwierając przeglądarkę z odpowiednim URL w oknie przeglądarki widzimy następujący rezultat.

Widać tutaj analogiczny widok do widoku klienta na tym etapie obsługi. Przedstawione okno składa się z rozwijanej listy dokumentów znajdujących się na serwerze oraz z przycisku nowy dokument, który tworzy nowy dokument na serwerze.



Po naciśnięciu na rozwijaną listę widzimy listę plików, które znajdują się na serwerze. W naszym przykładzie jest to abc.txt. Naciskając na element listy otwiera nam się pole tekstowe, w którym możemy dokonać edycji bieżącego dokumentu.



Widzimy, że w pliku abc.txt na serwerze znajduje się już jakiś ciąg znaków. Zamykając okno z dokumentem wracamy do naszego głównego menu. Naciskając przycisk „Utwórz Dokument” otwiera nam się poniższe okno.

Widzimy tutaj dwa przyciski oraz pole tekstowe. Wpisując w polu tekstowym nazwę pliku jaki chcemy stworzyć i przyciskając OK tworzy nam się plik (w naszym przypadku abcde.txt) na serwerze. Przycisk „←” sprawi, że wrócimy do głównego menu. Po automatycznym powrocie do głównego menu po naciśnięciu przycisku OK widzimy, że rzeczywiście na serwerze znajduje się nowo utworzony plik.

Testowanie

Stworzyliśmy testy jednostkowe testujące klienta Desktopowego, a jednocześnie poprawną reakcje serwera. Są to unit testy napisane w języku Python. Zarówno serwer jak i klient Desktop przeszły napisane przez nas testy. W przypadku aplikacji Web zostało przetestowane ogólne działanie aplikacji od strony użytkowej i wykryliśmy brak poprawnego działania dla skrajnych przypadków.