26.05.2020

Notice Board

Autorzy

Tomasz Jóźwik *(lider)*

Tomasz Mazur

Marcin Mozolewski

Marcin Różański

Spis treści

1. Projekt wstępny

1.1. Treść zadania - tablica ogłoszeń

Przedmiotem projektu jest stworzenie serwera i aplikacji do obsługi tablicy ogłoszeń aktualizowanej w czasie rzeczywistym. Wiadomość wysyłana przez użytkownika jest przekazywana przez serwer wszystkim innym użytkownikom „podpiętym" do odpowiedniej kategorii. Dodatkowo serwer przechowuje archiwum wiadomości w bazie danych umożliwiając przeglądanie starych wiadomości przez przeglądarkę WWW.

1.2. Założenia

1. Założenia funkcjonalne:
   - Logowanie i zarządzanie użytkownikami,
   - Wyświetlanie ogłoszeń,
   - Dodawanie ogłoszeń,
   - Archiwizacja ogłoszeń,
   - Tworzenie kategorii,
   - Przekazywanie ogłoszeń (wiadomości) do klientów.

2. Założenia niefunkcjonalne:
   - Całość komunikacji będzie zaimplementowana dla IPv4 i IPv6,
   - Aktualizacja tablicy w czasie rzeczywistym,
   - Kończenie wątku i zamykanie gniazd po utracie połączenia,
   - Intuicyjny interfejs użytkownika,
   - Obsługa sytuacji awaryjnych.

1.3. Podstawowe przypadki użycia

1. Przeglądanie ogłoszeń:
   a. Scenariusz główny:
      i. Użytkownik loguje się do systemu w aplikacji,
      ii. System wyświetla stronę główną zawierającą aktualne ogłoszenia,
   b. Scenariusz alternatywny 1:
      i. Jak w scenariuszu głównym,
      ii. System powiadamia o nieprawidłowości danych.
2. Dodanie ogłoszenia:
   a. Scenariusz główny:
      i. Użytkownik loguje się do systemu w aplikacji,
      ii. Użytkownik dodaje treść ogłoszenia oraz wybiera jego kategorie, po czym zatwierdza ogłoszenie,

iii. System zatwierdza ogłoszenie i wysyła je klientom, którzy mają podpiętą daną kategorię,

b. Scenariusz alternatywny 1:
   i. Jak w scenariuszu głównym,
   ii. System powiadamia o nieprawidłowości danych,

c. Scenariusz alternatywny 2:
   i. i.-ii. jak w scenariuszu głównym,
   ii. System powiadamia o nieprawidłowości ogłoszenia (np. zbyt duża ilość liter).

3. Przeglądanie archiwum:
   a. Scenariusz główny:
      i. Użytkownik loguje się do systemu w przeglądarce,
      ii. Aplikacja wyświetla główną stronę,
      iii. Użytkownik wybiera opcję przejrzenia archiwum,
      iv. System wyświetla wszystkie ogłoszenia z danej kategorii,
   b. Scenariusz alternatywny 1:
      i. Jak w scenariuszu głównym,
      ii. System powiadamia o nieprawidłowości danych.
4. Dodanie kategorii:
   a. Scenariusz główny:
      i. Użytkownik loguje się do systemu w przeglądarce,
      ii. Aplikacja wyświetla główną stronę,
      iii. Użytkownik wybiera opcję dodania nowej kategorii,
      iv. System wyświetla pole do wprowadzenia nazwy kategorii,
      v. Użytkownik podaje nazwę kategorii i zatwierdza wybór
      vi. System wprowadza nową kategorię
   b. Scenariusz alternatywny 1:
      i. Jak w scenariuszu głównym,
      ii. System powiadamia o nieprawidłowości danych.
   c. Scenariusz alternatywny 2:
      i. i-v. jak w scenariuszu głównym
      ii. System powiadamia o istnieniu kategorii z daną nazwą
5. Rejestracja do systemu
   a. Scenariusz główny:
      i. Użytkownik wybiera opcję rejestracji w przeglądarce
      ii. System wyświetla panel rejestracji
      iii. Użytkownik wprowadza dane
      iv. System rejestruje użytkownika

   b. Scenariusz alternatywny 1:
      i. i-iii jak w scenariuszu głównym

ii. System informuje o istnieniu użytkownika z danym Emailem

c. Scenariusz alternatywny 2:
   i. i-iii jak w scenariuszu głównym
   ii. System informuje o niezaakceptowaniu podanego hasła

1.4. Środowisko sprzętowo-programowe i narzędziowe

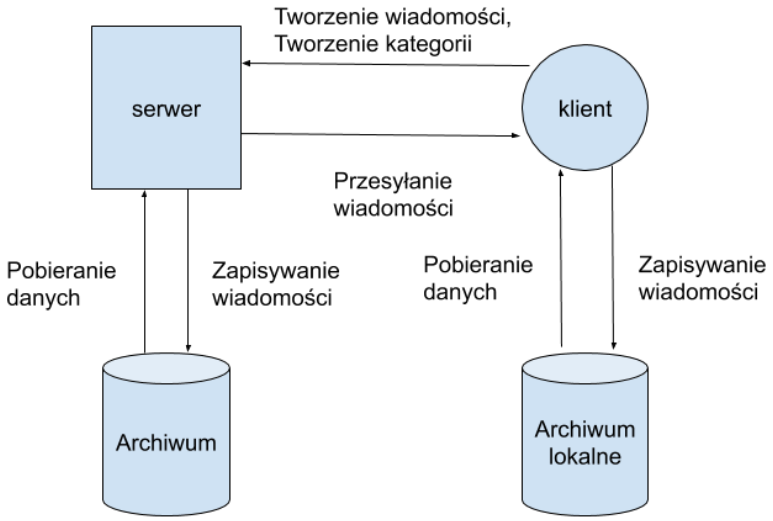
- System operacyjny Linux Ubuntu 18.04,
- Część komunikacyjna napisana została w języku C++ z wykorzystaniem API gniazd BSD i protokołu TCP,
- Do kompilacji kodu w języku C++ użyto kompilatora g++,
- Testy napisane zostały przy użyciu biblioteki Boost (C++), RSpec (Ruby on Rails),
- Do obsługi wielowątkowości wykorzystano bibliotekę pthread,
- Archiwalne wiadomości są przechowywane w bazie danych PostgreSQL,
- Do wymiany informacji między bazami danych i częścią komunikacyjną wykorzystana została bibliotaka libpqxx,
- Do napisania aplikacji internetowej do przeglądania archiwum wykorzystano framework Ruby on Rails,
- Do wyświetlania wiadomości przez klienta wykorzystywana aplikacja stworzona przy użyciu technologii JavaFX,
- Do debugowania użyto debuggera GDB,
- Do ciągłej integracji wykorzystano GitHub Actions oraz Travis CI.
- Pozostałe biblioteki: spdlog, nlohmann_json, postgresql(Java), lombok(Java).

1.5. Architektura rozwiązania: klient - serwer

1. Klient:
   - Aplikacja której zadaniem jest umożliwienie wysyłania wiadomości i wyświetlania ogłoszeń przypisanych do danego użytkownika.
     W tym celu łączy się z serwerem za pomocą API gniazd BSD,
   - Odbiera i aktualizuje w czasie rzeczywistym tablicę ogłoszeń, na której powinny pojawić się ogłoszenia związane z przypisanymi do użytkownika kategoriami.
2. Serwer:
   - Jest odpowiedzialny za obsługę żądań przychodzących z aplikacji klienta,

- Na podstawie informacji zawartych w bazie danych przesyła każdą otrzymaną wiadomość do odpowiednich użytkowników,
- Odpowiada za zarządzanie archiwum wiadomości. Każda nowa wiadomość powinna zostać przez serwer automatycznie dodana do bazy danych,
- Dodatkowo umożliwia przeglądanie archiwum poprzez aplikację internetową (przy pomocy przeglądarki).

3. Baza danych serwera:
    - Przechowuje informacje dotyczące użytkowników, kategorii oraz archiwum wiadomości.
4. Lokalna baza danych:
    - Przechowuje informacje na temat wiadomości i kategorii będących obiektami zainteresowania właściciela



1.6. Sposób testowania

- Testy jednostkowe,
- Oprogramowanie tworzone w TDD,
- Do testów regresyjnych wykorzystamy Continuous Integration: GitHub Actions (w przypadku kodu w C++) i Travis CI (w przypadku aplikacji Ruby on Rails),

- Jako bibliotekę do testowania wykorzystamy Boost (C++) i RSpec (Ruby on Rails),
- Testowanie integracyjne opierało się na testach empirycznych opartych o podstawowe przypadki użycia, weryfikujących poprawność działania.

1.7. Sposób demonstracji rezultatów

- W celu demonstracji rezultatów uruchomimy na jednej z maszyn serwer, a na innych aplikację klienta. W ten sposób będziemy w stanie zademonstrować wszystkie funkcjonalności projektu,

- Scenariusz 1: przesłanie wiadomości:
  - Przykładowa wiadomość zostanie przesłana przez jednego z klientów. Na komputerze z serwerem sprawdzimy (przez przeglądarkę) czy wiadomość została zapisana w archiwum. Na pozostałych klientach sprawdzimy, czy została zaktualizowana tablica ogłoszeń (przy założeniu, że klient był przypisany do tej samej kategorii co wiadomość),

- Scenariusz 2: tworzenie kategorii:
  - Użytkownik łączy się z aplikacją przeglądarkową. Następnie wprowadza nową kategorię do obecnej bazy. Po wykonaniu danej czynności inny użytkownik w przeglądarce podpina się pod daną kategorię. Następnie wchodzi do aplikacji klienta i sprawdza czy dana kategoria się pojawiła.

- Scenariusz 3: Dodawanie nowego użytkownika i logowanie:
  - Użytkownik łączy się z aplikacją przeglądarkową. Wybiera opcję utworzenia nowego konta użytkownika, wypełnia formularz z podstawowymi danymi użytkownika i zatwierdza utworzenie nowego konta,

- Scenariusz 4: Obsługa sytuacji awaryjnych:
  - Klient pobierze wiadomości przeznaczone dla niego i następnie rozłączymy połączenie pomiędzy klientem i serwerem. Na innym połączonym kliencie wyślemy wiadomość, która powinna trafić do odłączonego klienta. Po wysłaniu wiadomości połączymy ponownie klienta z serwerem i sprawdzimy czy dotarła zaległa wiadomość.

1.8. Podział prac

Zadanie	**Tomasz Jóźwik**	**Tomasz Mazur**	**Marcin Mozolewski**	**Marcin Różański**
API gniazd BSD - po stronie serwera		■	■	
API gniazd BSD - po stronie klienta	■			■
Obsługa sytuacji awaryjnych - po stronie serwera		■	■	
Obsługa sytuacji awaryjnych - po stronie klienta	■			■
Archiwum wiadomości	■			■
Tworzenie kont użytkowników	■			■
Wyświetlanie ogłoszeń danego klienta		■	■	
Aplikacja klienta (Java)		■	■	
Testy jednostkowe (C++)		■	■	■
Testy jednostkowe (RSpec)	■			
Testy integracyjne	■	■	■	■
Testy w zakresie bezpieczeństwa	■	■	■	■
Testy wdrożeniowe		■	■	
Testy regresji	■	■	■	■
Przygotowanie prezentacji końcowej	■	■	■	■

1.9. Harmonogram prac

	Estymacja terminu		
Zadanie	**Najlepszy scenariusz**	**Najbardziej prawdopodobny scenariusz**	**Ostateczny scenariusz**
Stworzenie wstępnej wersji koncepcji projektu	25.03.2020	28.03.2020	28.03.2020
Konsultacje stworzonej koncepcji	-	31.03.2020	31.03.2020
Stworzenie dokumentacji projektu wstępnego w formacie PDF	31.03.2020	03.04.2020	01.04.2020
Implementacja prototypu części komunikacyjnej	08.04.2020	23.04.2020	18.04.2020
Implementacja prototypu archiwum wiadomości	11.04.2020	26.04.2020	22.04.2020
Implementacja prototypu tablicy ogłoszeń	15.04.2020	26.04.2020	22.05.2020
Zgłoszenie częściowej realizacji projektu	05.05.2020	09.05.2020	06.05.2020
Stworzenie wersji finalnej części komunikacyjnej	14.05.2020	20.05.2020	24.05.2020
Stworzenie wersji finalnej archiwum wiadomości	16.05.2020	21.05.2020	24.05.2020
Stworzenie wersji finalnej tablicy ogłoszeń	16.05.2020	21.05.2020	25.05.2020
Testy bezpieczeństwa aplikacji	16.05.2020	23.05.2020	25.05.2020
Testy wdrożeniowe	19.05.2020	24.05.2020	26.05.2020
Przygotowanie prezentacji projektu	17.05.2020	25.05.2020	31.05.2020
Prezentacja projektu	01.06.2020	-	05.06.2020

1.10. Linki do repozytoriów

https://github.com/tjozwik/TIN_Tablica_ogloszen

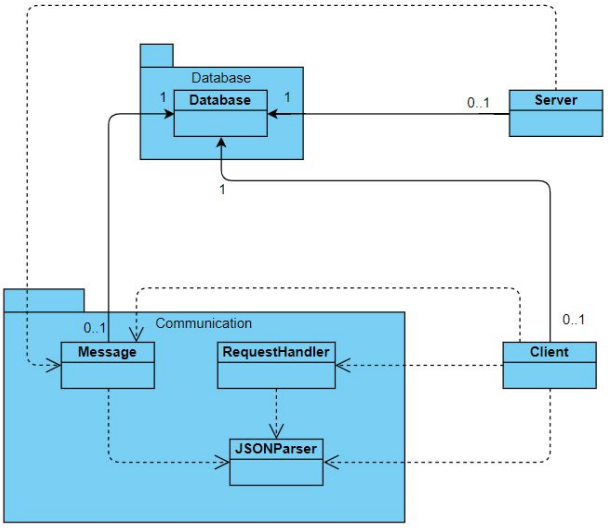
https://github.com/tjozwik/TIN_Archiwum

https://github.com/marcjanek/TIN-client-GUI

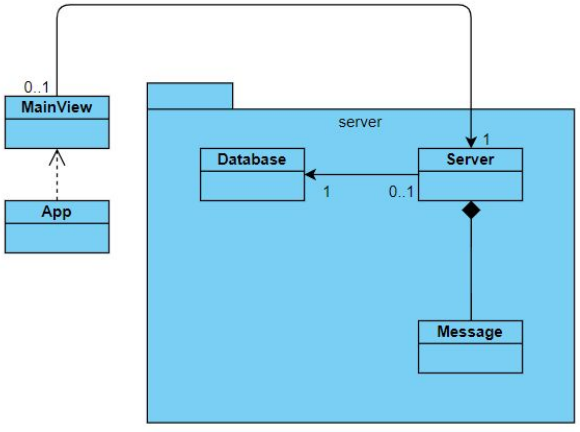
2. Projekt końcowy

2.1. Struktura implementacyjna systemu

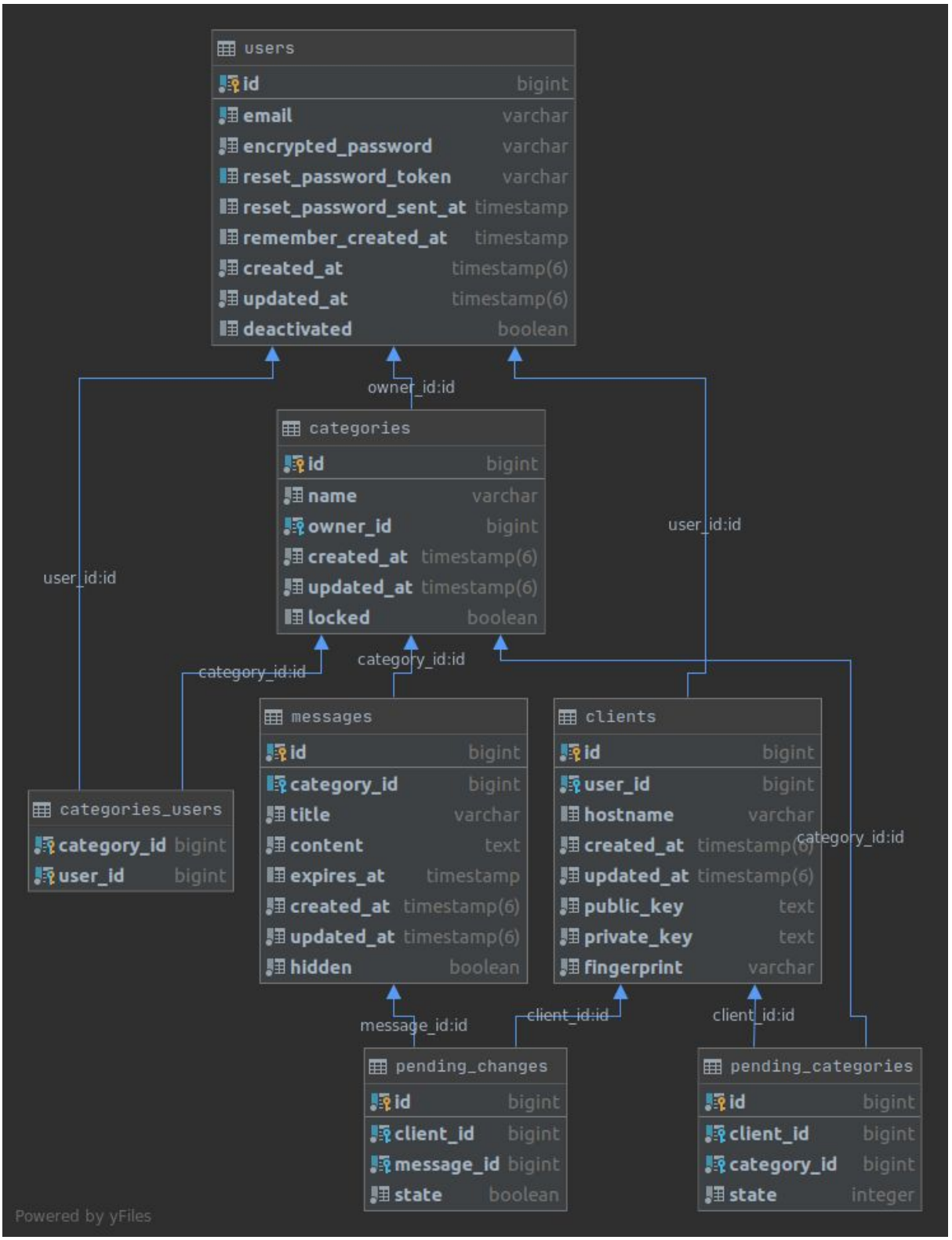
2.1.1. Struktura głównej aplikacji



2.1.2. Struktura graficznego interfejsu użytkownika

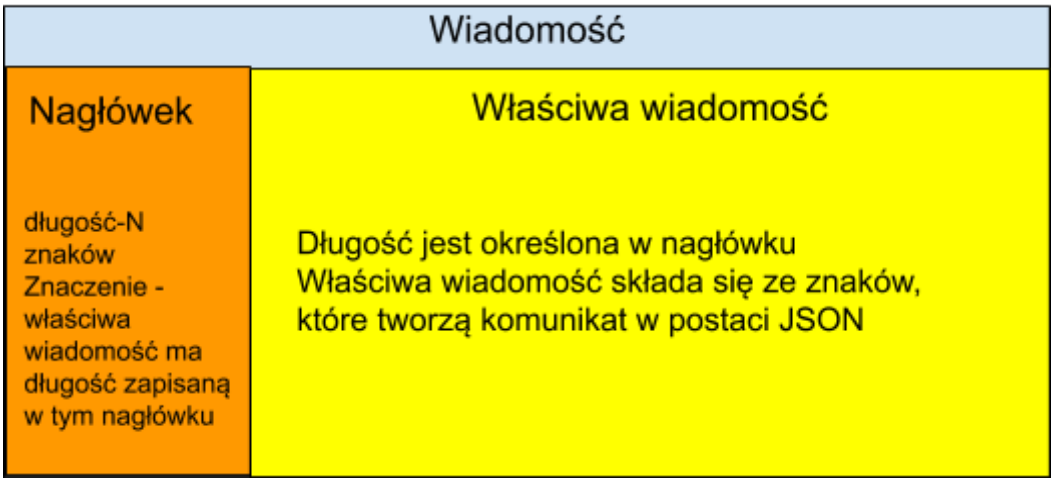


2.1.3. Struktura bazy danych serwera



2.2. Definicja komunikatów

Komunikacja w systemie polega na wysyłaniu żądań przez klienta do serwera, ich obsłudze przez serwer i wysyłaniu odpowiedzi zawierającej wynik obsłużenia żądania. Wszystkie komunikaty przekazywane w systemie przekazywane są jako łańcuchy znaków, wykorzystywane są do tego odpowiednie funkcje związane z gniazdami.



Nagłówek: może składać się tylko z cyfr. W przypadku wystąpienia znaku innego niż cyfra, wystąpi błąd i zostanie zwrócony wyjątek z metody odczytującej wiadomość z deskryptora.

Właściwa wiadomość: po odczytaniu długości wiadomości z nagłówka następuje odczytanie właściwej treści wiadomości. Czytanie jest wyłącznie tyle znaków ile zostało zgłoszone w nagłówku wiadomości. Jeżeli wysłana zostanie inna ilość znaków zostanie to zauważone i zgłoszone jako wyjątek (pozwala to na obsługę sytuacji związanych np. z wysyłaniem wiadomości o zbyt dużej długości).
Wiadomości wysyłane są jako ciągi znakowe w postaci JSON. W klasach obsługujących tworzenie komunikatów zdefiniowano wszystkie możliwe komunikaty. Zadaniem klienta jest wpisanie odpowiednich wartości do atrybutów, które potem zostaną odczytane przez serwer. Serwer zwraca łańcuch znaków w postaci JSON zawierający odpowiedź na żądanie klienta. Każda komunikat zawiera atrybut code - liczbę całkowitą, która oznacza rodzaj żądania/kod odpowiedzi oraz własną listę atrybutów związanych z przekazywaną wiadomością (może to być np. atrybut token, body, title, category i inne).
Znaczenie atrybutu code:

- 1-Autoryzacja klienta,
- 2-Utworzenie nowej wiadomości,
- 3-Pobranie nowej wiadomości,
- 4-Usunięcie wiadomości,

- 5-Pobranie nowej kategorii,
- 0-Komunikat o błędzie.

2.3. Opis zachowania podmiotów komunikacji

2.3.1. Serwer

Serwer po uruchomieniu otwiera pasywnie gniazda BSD oczekujące na połączenie z klientem. W momencie ustanowienia tego połączenia od klienta pobierany jest komunikat, który jest następnie parsowany. Po stwierdzeniu poprawności przesłanego komunikatu wysyłane jest adekwatne żądanie do bazy danych, przechowującej archiwum wiadomości. Na podstawie otrzymanej stamtąd odpowiedzi generowany jest komunikat zwrotny dla klienta, który następnie zostaje wysłany przy użyciu przypisanego mu gniazda BSD. Po wykonaniu takiej operacji gniazdo jest zamykane. Gniazdo jest zamykane również jeśli przypisany mu zegar przekroczy zadany czas. W wypadku błędu związanego z bazą danych, zarówno powiązanego bezpośrednio z połączeniem jak i przetworzeniem wysłanego żądania, odpowiedni komunikat jest zwracany klientowi. Efekty wykonanych działań bez względu na efekt są dokumentowane w dzienniku programu.

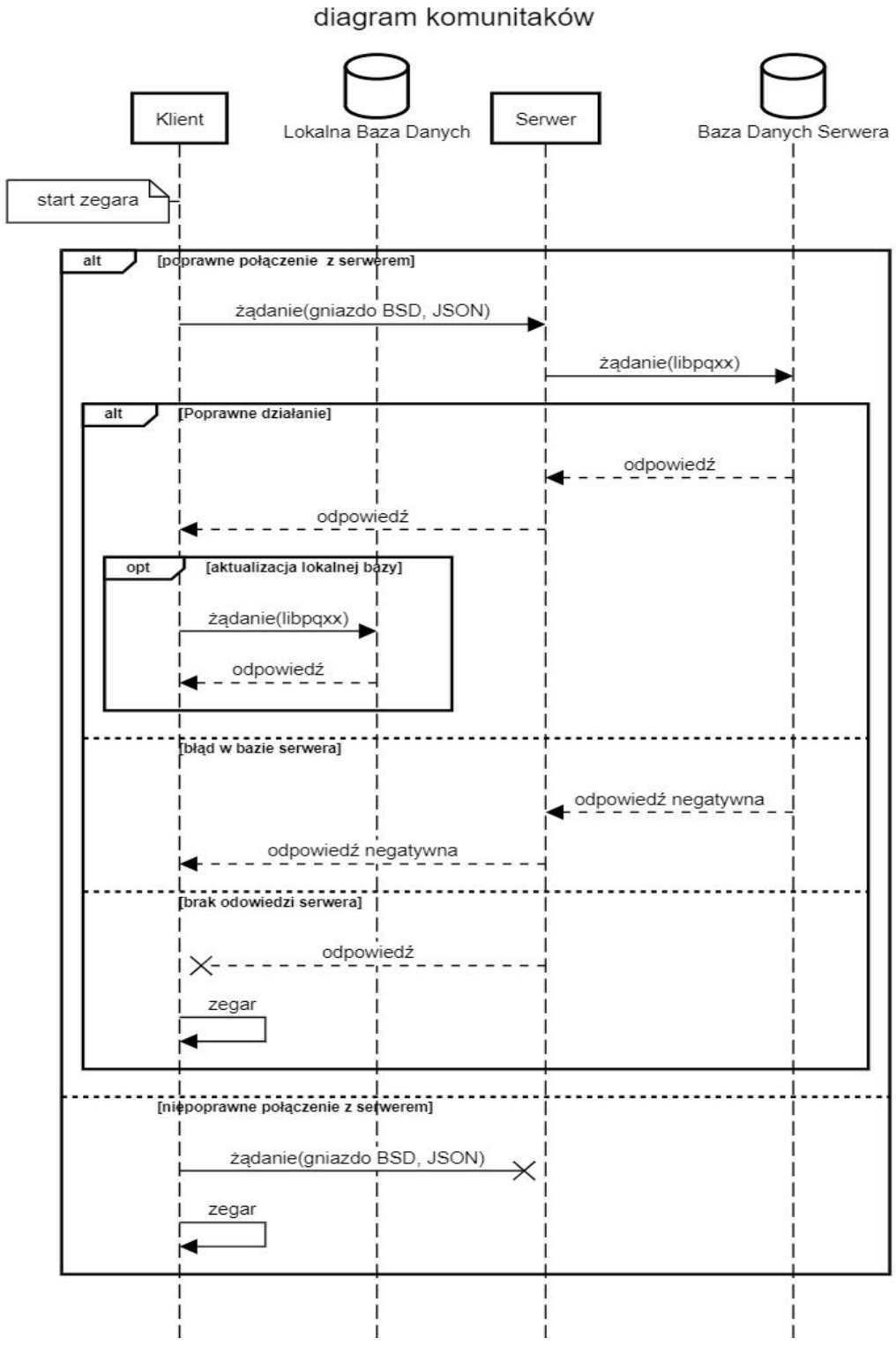
2.3.2. Klient

Klient po uruchomieniu łączy się z serwerem poprzez gniazdo BSD. Wczytane przez niego żądanie, pochodzące z lokalnej bazy danych lub wykonywane okresowo (w przypadku synchronizacji), jest odpowiednio przekształcane na pożądany ciąg znakowy w postaci JSON i przekazywane do serwera. Po otrzymaniu poprawnej odpowiedzi przeprowadzana jest aktualizacja odpowiednich rekordów w lokalnej bazie danych, co synchronizuje jej stan z główną bazą danych, wykorzystywaną przez serwer. W wypadku wystąpienia problemów związanych z komunikacją z serwerem, lokalną bazą danych lub po otrzymaniu informacji o nieudanym przetworzeniu żądania, nastąpi przerwanie próby nawiązania połączenia przez zegar. Informacja na temat przeprowadzonego procesu jest przesyłana do dziennika programu.

2.3.3. Archiwa wiadomości

Baza danych obsługująca archiwum wiadomości wykonuje zapytania otrzymane od serwera, który wysyła je przy pomocy biblioteki libpqxx. Stan archiwum wiadomości jest aktualizowany na podstawie przesyłanych przez klientów zapytań, za pośrednictwem serwera.
Baza danych umożliwia także synchronizację aktualnych wiadomości na klientach. W tym celu tworzone są przy pomocy wyzwalaczy odpowiednie rekordy w tabelach intersekcji - właściwych dla wiadomości oraz kategorii. Wyzwalacze zapewniają, że

synchronizacja odbędzie się tylko dla wiadomości dotyczących bezpośrednio danego użytkownika. W przypadku pozostałych wiadomości synchronizacja zostanie przeprowadzona dopiero w momencie podjęcia działania związanego z daną kategorię i użytkownikiem (np. w momencie zapisania lub wypisania się użytkownika z danej kategorii).



2.4. Wnioski z testowania

Na podstawie obserwacji dokonanych w trakcie przeprowadzania testów na aplikacji, możemy stwierdzić, że poszczególne podmioty komunikacji, jak i cały system, działają zgodnie z pierwotnymi założeniami projektu, a więc spełniają postawione przez prowadzącego wymagania. Zauważalne może być opóźnienie w graficznym interfejsie użytkownika przy dodaniu naraz wielu nowych wiadomości na serwer. Wynika to z faktu, że nowe wiadomości pobierane są pojedynczo z pewną przerwą między kolejnymi pobraniami. Nie przekreśla to jednak poprawnego działania systemu jako całości.

2.5. Analiza podatności bezpieczeństwa

Zarówno po stronie serwera, jak i klienta, odbierane komunikaty podlegają parsowaniu w celu znalezienia niespójności między informacjami na temat wiadomości podanymi w nagłówku i samą treścią. W wypadku potwierdzenia spójności komunikatu jest on, na podstawie informacji z nagłówka, odpowiednio dzielony i czytany przez konkretny podmiot. W przeciwnym wypadku komunikat jest odrzucany i nie jest poddawany dalszym działaniom. Ponieważ komunikaty są sprawdzane przez oba podmioty, nie istnieje ryzyko wystąpienia przepełnienia bufora nawet przy odebraniu komunikatu z niepowołanego źródła. Wiadomości stworzone w graficznym interfejsie użytkownika są poddawane wstępnej analizie pod kątem poprawności jeszcze przed ich wysłaniem.

Biblioteka libpqxx użyta do komunikacji z bazą danych przesyła wysyłane żądania jako normalne transakcje. Zapewnia to brak ryzyka naruszenia powiązań występujących obecnie w bazie. W wypadku próby wysłania żądania, którego skutkiem byłoby naruszenie tych powiązań, żądanie nie jest akceptowane i zwracany jest wyjątek.

W przypadku aplikacji przeglądarkowej wykorzystywana jest walidacja dla pól dostępnych dla użytkownika. Hasło do konta użytkownika w aplikacji przeglądarkowej, które umożliwia generowanie kluczy dostępu do serwera, jest przechowywane w formie zaszyfrowanej. Istnieje możliwość unieważniania kluczy dostępu dla poszczególnych użytkowników poprzez archiwum wiadomości. Informacja o nazwie komputera, który jako ostatni skorzystał z danego klucza dostępu, jest zapisywana i wyświetlana w aplikacji przeglądarkowej.

2.6. Instalacja systemu

System składa się z następujących części:

- klienta (bazy danych PostgreSQL, aplikacji w języku C++ oraz aplikacji graficznej w języku Java),
- serwera (aplikacji w języku C++),
    - archiwum wiadomości (bazy danych PostgreSQL - dostępnej również dla serwera, aplikacji w języku Ruby on Rails).

Poszczególne części systemu mogą zostać zainstalowane w sposób następujący:

- klient
    - baza danych PostgreSQL:
        - zalecane wykorzystanie kontenera postgres dla dockera. Utworzenie wymaganego obrazu wraz z zainicjalizowaną strukturą możliwe jest poprzez komendę: “docker build -f Initializers/Dockerfile.pg .”,
    - aplikacja C++:
        - kompilacja przy wykorzystaniu CMakeLists.txt. Uruchomienie utworzonego pliku wykonywalnego “client”,

- serwer
    - aplikacja C++:
        - kompilacja przy wykorzystaniu CMakeLists.txt. Uruchomienie utworzonego pliku wykonywalnego “server”,
    - archiwum wiadomości
        - baza danych PostgreSQL:
            - zalecane skorzystanie z usługi PaaS (np. heroku),
            - w przypadku własnej infrastruktury wskazane użycie kontenera postgres dla dockera,
    - aplikacja przeglądarkowa:
        - wymagany interpreter języka Ruby w wersji 2.6.6,
        - instalacja zależności poprzez wydanie komendy: “bundle”,
        - konfiguracja bazy wymaga podania szczegółów w pliku config/database.yml i wydania komendy “rails db:migrate”,
        - oznaczanie wiadomości wygasłych (ich usuwanie z klientów) możliwe jest poprzez wywołanie komendy uruchamiającej zadanie rake: “rake messages:hide_expired”. Podczas testowania do automatyzacji tego procesu został wykorzystany heroku Scheduler.

Zmienne środowiskowe wykorzystane do konfiguracji systemu:

Zmienna	Opis	Wartość domyślna
Klient		
SERVER_NAME	Adres serwera	127.0.0.1
SERVER_PORT	Port serwera	57076
DATABASE_NAME	Nazwa bazy danych	postgres
DATABASE_USER	Użytkownik bazy danych	postgres
DATABASE_PASSWORD	Hasło bazy danych	docker
DATABASE_PORT	Port bazy danych	5432
DATABASE_HOST	Adres serwera bazy danych	0.0.0.0
CLIENT_TOKEN	Klucz dostępowy użytkownika	/brak wartości domyślnej/
Serwer		
DATABASE_NAME	Nazwa bazy danych	noticeboard
DATABASE_USER	Użytkownik bazy danych	noticeboard
DATABASE_PASSWORD	Hasło bazy danych	noticeboard
DATABASE_PORT	Port bazy danych	5432
DATABASE_HOST	Adres serwera bazy danych	0.0.0.0

2.7. Podsumowanie

2.7.1. Doświadczenia wyniesione z projektu

1. Należy uważać na wersję systemu operacyjnego wykorzystywanego do tworzenia aplikacji, gdyż różnice w wersjach niektórych zależności między członkami zespołu prowadziły do problemów z działaniem aplikacji,

2. Skorzystanie z usługi PaaS (w naszym przypadku heroku) znacznie przyspieszyło utworzenie produkcyjnej instalacji aplikacji przeglądarkowej, stanowiącej część projektu,
3. Kluczowy był początkowy podział obowiązków, który podlegał ewentualnym korektom. Dzięki temu zespół nie przeznaczał nadmiernej ilości czasu na ustalanie swoich obowiązków na bieżąco,
4. Należy możliwie często i regularnie informować członków zespołu o dokonanych zmianach w kodzie lub koncepcji, które mogą mieć wpływ na działanie tworzonego przez nich kodu,
5. Wspólna praca nad danym fragmentem projektu pozwala na wyeliminowanie potencjalnych błędów i niespójności w koncepcji. Jednocześnie efektywność pracy spada gdy więcej niż 2 osoby pracują w tej samej chwili nad tą samą funkcjonalnością,
6. Warto dokładnie przestrzegać terminów wyznaczonych przez prowadzącego - ocena podczas częściowego oddania ma znaczący wpływ na motywację do dalszej realizacji projektu.

2.7.2. Statystyki określające rozmiar stworzonych plików

Kod projektu (w części klienta i serwera) zajmuje łącznie ok. 2000 linii.
Ilość linii kodu w głównych plikach:

nazwa pliku	**liczba linii w .cpp**	**liczba linii w .h**
database	~450	~70
client	~380	~70
Message	~230	~40
server	~170	~40
RequestHandler	~90	~40
JSONParser	~80	~50

Statystyki dotyczące tworzonych linii kodu C++ w zależności od czasu:

Statystyki dotyczące usuwanych linii kodu w zależności od czasu:

Statystyki dotyczące ilości commitów w zależności od czasu:

Wnioski dotyczące przyrostu kodu:

1. Początkowy wolniejszy przyrost ilości nowych i usuniętych linijek kodu spowodowany jest przeznaczeniem większej ilości czasu na prace koncepcyjne,
2. Wraz ze wzrostem zaawansowania projektu wzrosła też liczba usuwanych linijek. Spowodowane było to m.in. przez drobne zmiany w koncepcji projektu,
3. Praca przebiegała systematycznie, natomiast widoczne jest zwiększenie zaangażowania wraz ze zbliżaniem się poszczególnych terminów odbioru częściowego.

W przypadku GUI i aplikacji przeglądarkowej, uzyskane statystyki są nieadekwatne ze względu na wykorzystanie do tego frameworków generujących znaczną liczbę linii kodu. Przykładowo, w aplikacji przeglądarkowej utworzone zostało ok. 37 tysięcy linijek kodu.

2.7.3. Oszacowanie czasu poświęconego na realizację projektu

Największa ilość czasu, przeznaczonego na projekt, została wykorzystana na implementację oraz naprawę błędów w projekcie. Część czasu przeznaczonego na implementację realnie była przeznaczona na testowanie - sprawdzanie działania funkcjonalności po utworzeniu zmian. Szacowany łączny czas, poświęcony przez wszystkich członków zespołu na realizację projektu, wyniósł około 280 godzin. Oszacowanie podziału na poszczególne typy zadań przedstawia się następująco: