Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Katedra Inżynierii Oprogramowania

kierunek studiów: Informatyka

specjalność: Inżynieria Oprogramowania

Praca dyplomowa inżynierska

IMPLEMENTACJA APLIKACJI DO WSPÓŁTWORZENIA NOTATEK Z WYKORZYSTANIEM TECHNOLOGII PEER TO PEER

IMPLEMENTATION OF A COLLABORATIVE NOTE-TAKING APPLICATION USING PEER-TO-PEER TECHNOLOGY

Oskar Marcin Chybowski

nr albumu: **54941**

Opiekun:

dr inż. Mirosław Mościcki

Katedra Inżynierii Oprogramowania

Szczecin, 2026

Spis treści

Wstęp

Tu będzie dopiero treść :)

1 Zakres teoretyczny

1.1 Systemy współdzielenia dokumentów w czasie rzeczywistym

Budując rozwiązania związane z równoczesnym tworzeniem i modyfikacją tekstu przez więcej niż jednego użytkownika, musimy rozważyć wyzwania napotykane w aktualizacji tworzonego dokumentu, gdzie każdy klient posiada lokalną kopię i nanosi na nie własne zmiany, ale też w międzyczasie musimy nanieść zmiany od pozostałych klientów. W takim systemie mówimy wtedy o zbieżności danych[1] - czyli zapewnieniu tego samego stanu między każdym klientem. W przypadku edycji tekstu skupię się na ewentualnej zbieżności, która uwzględnia posiadanie rozbieżnych kopii tego samego źródła danych u każdego z klientów przez pewien czas. Dopiero gdy zostanie zakończona edycja tekstu, zmiany zostają propagowane i nanoszone do pozostałych klientów. Finalnie każdy klient po czasie posiada identyczną kopię dokumentu. Ze strony doświadczeń użytkowania jest to skuteczna strategia ze względu na możliwość zapewnienia płynności interfejsu graficznego oraz z pomocą złożonych mechanizmów umożliwia rozwiązywanie konfliktów między kopiami.

Wspomniany model nie jest bez wad. Największym problemem jest istnienie konfliktów, których rozwiązanie klienci muszą ustalić za pomocą dodatkowych strategii. Najeczęściej wykorzystywaną jest Last-Write-Wins (LWW). Rozstrzyga ona konflikty poprzez nanoszenie tylko tej zmiany, która jest uznawana jako ostatnia w kolejności zbioru konfliktujących operacji. Ustalanie kolejności nie jest jasno tutaj zdefiniowane. W systemach baz danych takich jak Cassandra[9] oraz SQL Server P2P[10] każdy zapis otrzymuje własny znacznik czasowy, na podstawie którego wybierany jest najmłodszy wpis i nim nadpisywane są zmiany w źródle danych. Zmiany ze starszymi znacznikami są porzucane. Zauważalną wadą LWW jest wysokie ryzyko utraty danych w czasie nanoszenia zmian, ponieważ wszystkie konfliktujące starsze zmiany nie są brane pod uwagę.

W przypadku tekstu jako typu danych, istnieje specjalny wariant ewentualnej zbieżności - silna ewentualna zbieżność. Ten model wykorzystuje specjalne struk-

tury danych, które zapewniają bezkonfliktowe nanoszenie zmian, a ich skuteczność opiera się na matematycznych dowodach[2].

Istnieje również model silnej zbieżności, gdzie każdy z klientów musi mieć tą samą kopię danych przez cały czas pracy systemu. Ze względu opóźnienia i potrzebę mechanizmu blokady klientów przed wprowadzaniem zmian, ten model został porzucony w dzisiejszych systemach, ponieważ wspomniane problemy skutkowały gorszą użytecznością w porównaniu do systemów wykorzystujących ewentualną zbieżność. Istnieją jednak prace, które wskazują na istnienie systemów opartych o model silnej zbieżności, które osiągają bardzo niskie czasy opóźnienia, co redukuje problem używalności takiego rozwiązania w rzeczywistych zastosowaniach.[11]

1.2 Algorytmy zapewniające silną ewentualną zbieżność

Podejście do rozwiązania problemu synchronizacji stanów między każdym klientem wymaga zaprojektowania algorytmu od podstaw skupionego na tym zagadnieniu. Przedstawię dwa najczęściej wykorzystywane algorytmy rozwiązujące opisany problem.

Operational Transformation (OT) polega na zamianie każdej wykonanej operacji na kodowalny obiekt, który może być propagowany i nanoszony na kopie w innych replikach. Większość znanych implementacji zakłada też, że istnieje scentralizowany serwer określający kolejność każdej operacji zgłaszanej przez wszystkich klientów i dystrybujący wspomniane zmiany do pozostałych replik. W przypadku wystąpienia konfliktu, operacje są sortowane przez serwer, a następnie każda z nich jest transformowana tak, by uwzględnić zmianę poprzedniej w kolejności operacji. Przykładowymi propozycjami algorytmów OT, których skuteczność nie została obalona dowodami, są: Jupiter[3], SOCT3/4[4] oraz TTF[5]. Jupiter oraz SOCT3/4 wymagają wcześniej wspomnianego scentralizowanego serwera. Google Docs opiera się na Operational Transformation[12]. Ze względu na złożność implementacji większość systemów nie korzysta dziś z tych algorytmów do synchronizacji danych, gdzie przykładem takiej decyzji jest Figma[13].

Następcą Operational Transformation są bezkonfliktowe replikowane typy danych (Conflict-free replicated data types - CRDT). Pozwalają one na nanoszenie zmian na własne kopie przez klientów, bez potrzeby uzgadniania tego z innymi klientami. Tak jak każdy algorytm ewentualnej zbieżności, pozwala na tymczasową rozbieżność między replikami, ale po otrzymaniu wszystkich zmian przez każdego klienta, dane zawsze pozostają zbieżne. Przykładowymi algorytmami są RGA[6], WOOT[7] oraz TreeDoc[8].

1.3 Architektura Peer-To-Peer w środowiskach mobilnych

Współcześnie smartfony i podobne urządzenia mobilne posiadają moduły wspierające różne standardy komunikacji bezprzewodowej. Najpopularniejszymi z nich są Bluetooth, Bluetooth Low Energy (BLE), Wi-Fi (IEEE 802.11) oraz GSM (Global System for Mobile Communications). Część z nich, np. Bluetooth, jest zaprojektowana tylko pod komunikację na niewielkie odległości (poniżej 100 metrów). Wi-Fi w nowoczesnych systemach jest rozszerzane o wsparcie protokołu Wi-Fi Direct, który umożliwia komunikację Peer-To-Peer z pomocą Wi-Fi, które najczęściej samo w sobie służy do komunikacji z innymi urządzeniami, ale wykorzystując do tego urządzenia infrastruktury sieciowej - bezprzewodowe punkty dostępu (wireless access points).

1.4 Architektura Peer-To-Peer w systemach iOS, iPadOS, macOS, tvOS.

Systemy operacyjne iOS, iPadOS, tvOS oraz macOS są rozwijane przez firmę Apple Incorporated i instalowane wyłącznie na urządzeniach przez nią produkowanych - iPhone'y, iPady, Apple TV, Macintoshe. Współcześnie wszystkie sprzedawane modele zawierają moduły oferujące Bluetooth i Wi-Fi. Dla programistów przygotowane są specjalne biblioteki do bezpośredniego wykorzystania tych technologii, jednak ich użycie wymaga specjalnych certyfikatów wraz z wyjaśnieniem ich wykorzystania, które jest weryfikowane w czasie przygotowania do dystrybucji oprogramowania wykoszytującego wspomniane technologie. Ze względu na wyraźną potrzebę zapewnienia bezpieczeństwa transportowanych danych, zalecanym przez Apple jest korzystanie nie z bezpośrednich narzędzi do komunikacji z wykorzystaniem Bluetooth czy Wi-Fi, ale z bibliotek, które oferują transport danych do pobliskich urządzeń. Są one abstrakcją na wcześniej wspomniane protokoły komunikacji bezprzewodowej jak i przewodowej, gdzie po stronie programisty pozostaje jedynie obsłużyć parowanie się z pobliskimi urządzeniami i przygotować kodowalne obiekty do transportu. Dziś możemy wyróżnić dwie takie biblioteki oferowane przez Apple - Multipeer Connectivity oraz Network.

Multipeer Connectivity to framework zapewniający komunikację oraz odkrywanie pobliskich urządzeń. Do komunikacji wykorzystuje protokoły komunikacji bezprzewodowej - w tym dostępne sieci Wi-Fi, Bluetooth oraz w przypadku Macintoshy oraz Apple TV - protokół komunikacji przewodowej - Ethernet. Interfejs dostarczany przez bibliotekę umożliwia transport danych w postaci niewielkich wiadomości, strumieniowania danych oraz transportu plików.

Architektura Multipeer Connectivity opiera się na sesjach - połączeniach między użytkownikami w pobliżu. Sesje są reprezentowane przez obiekty typu MCSession. Takie obiekty są tworzone przez programistę w dowolnym momencie oraz u nowo połączonego klienta od razu po dołączeniu do sesji do której został zaprosozny,

jednocześnie każda z kopii sesji zawiera informację o wszystkich połączonych klientach. Do odkrywania innych użytkowników używany jest protokół Bonjour.

Urządzenie ogłaszające u pobliskich urządzeń dostępną sesję jest nazywany odkrywcą. Jego rolę reprezentuje obiekt MCNearbyServiceBrowser. Możemy również wykorzystać obiekt MCBrowserViewController który zapewni interfejs graficzny, dzięki któremu użytkownik będzie mógł zaprosić klientów do swojej sesji. Urządzenie poszukujące dostępnych sesji jest nazywany nadawcą. Jego reprezentantem jest obiekt MCNearbyServiceAdvertiser.

Obiektem identyfikującym urządzenie miedzy sesjami jest MCPeerID, zawierający dane unikalne dla klienta w zakresie danej sesji.

W czasie odkrywania urządzeń w pobliżu, mamy jedynie dostęp do ograniczonej ilości informacji, które są dystrybuowane przez nadawców. Dopiero po wysłaniu zaproszenia przez odkrywcę, a następnie akceptacji przez nadawcę, możemy wykorzystać w pełni interfejs Multipeer Connectivity do komunikacji.

Network jest frameworkiem skupionym na ogólnej interakcji z połączeniami sieciowymi. Oprócz lokalnego dostępu do innych urządzeń, daje możliwość wykorzystania całej dostępnej infrastruktury sieciowej. Apple w przypadku tworzenia aplikacji w oparciu o połączenia peer to peer, zaleca łączenie Network z biblioteką DeviceDiscoveryUI, która zapewnia dodatkowy interfejs graficzny dla użytkownika aplikacji implementujący proces parowania. W ramach tego procesu Apple pozwala na wybór protokołu do komunikacji w czasie parowania. Mamy do wyboru Bonjour - starszy, należący do Apple protokół - oraz Wi-Fi Aware - otwarty, międzyplatformowy standard. DeviceDiscoveryUI jest nową biblioteką, dostępną od systemów operacyjnych wydanych w 2025 roku - iOS 26, iPadOS 26, macOS 26. Wyjątkiem jest wsparcie dla tvOS, tutaj Apple oferuje wsparcie od wersji 16, wydanej w 2022 roku. Wi-Fi Aware jest również dostępny dopiero od systemów wydanyc w 2025 roku. Network wraz z własnym protokołem ramkowania jest dostępny od wersji systemów operacyjnych wydanych w 2019 roku.

W przypadku oparcia aplikacji komunikujacej się peer to peer z innymi urządzeniami o Network, musimy przygotować własny protokół ramkowania `NWProtocolFramerImplementation`. W nim powinniśmy obsłużyć wszystkie wydarzenia związane z poprawnym zarządzaniem połączeniem - inicjację, powrót z uśpienia, zatrzymanie, proces czyszczenia prze dealokacją. Do nasłuchiwania na przychodzące połączenia wykorzystalibyśmy obiekt `NWListener`. Połączenie z innymi urządzeniem byłoby reprezentowane przez obiekt `NWConnection`, a jeśli chcielibyśmy zaimplementować własny proces odkrywania i negocjacji połączenia z pobliskimi urządzeniami, musielibyśmy dodatkowo wykorzystać obiekt `NWBrowser`.

1.5 Istniejące rozwiązania i ich problemy

Dwoma najpopularniejszymi aplikacjami do współtworzenia notatek w czasie rzeczywistym są Google Docs i Microsoft Word Online. Ich zakres funkcjonalności jest bardzo podobny - złożone formatowanie tekstu; możliwość dodawania załączników, tabel; historia wersji dokumentu; obecność publicznego API; wykorzystanie nowoczesnych standardów szyfrowania komunikacji; eksport dokumentu w postaci innego rodzaju pliku oraz wymagają konta do możliwości ich użycia. Google Docs korzysta ze swojego algorytmu Operational Transformation do synchronizacji zmian między użytkownikami, sposób przechowywania dokumentów nie jest jasny, nie wiemy w jakiej postaci są one przechowywane na serwerach Google, a dostęp do aplikacji jest darmowy. Microsoft Word Online nie udostępnia publicznie informacji o stosowanym podejściu do synchronizacji danych, dokumenty są przechowywane w postaci plików docx, a dostęp do aplikacji jest również darmowy.

Głównym problemem większości dzisiejszych narzędzi do tworzenia dokumentów jest uzależnienie od dostawcy. Każda z wymienionych aplikacji, jak i wiele innych dostępnych na rynku do skorzystania ze swoich usług wymaga założenia konta na platformie dostawcy, a w czasie współpracy wymagany jest ciągły dostęp do Internetu. W momencie gdy pracujemy nad jednym dokumentem z innymi użytkownikami w tym samym pomieszczeniu i nasze urządzenia są podłączone do tej samej sieci, opóźnienie w nanoszeniu zmian między użytkownikami zawsze uwzględnia dostęp do scentralizowanych serwerów. W momencie gdy zostaniemy odcięci od dostępu do Internetu, ale nadal będąc w tej samej sieci lokalnej, tracimy możliwość dalszej pracy. Szczególnie widać to w przypadku Google Docs, którego algorytm synchronizujący narzuca obecność serwera wybierającego kolejność nanoszenia zmian w dokumentach.

2 Wymagania systemowe

W poniższym rozdziale opiszę specyfikację wymagań dla projektowanej aplikacji do współtworzenia notatek w architekturze peer to peer. Celem jest określenie zakresu funkcjonalności oraz ograniczeń technologicznych, których będę się trzymać w ramach proponowanej implementacji. Wyodrębnię dwóch aktorów:

- Użytkownika - osoba zarządzająca notatkami przez interfejs graficzny zaimplementowany w ramach implementacji,
- Klient - instancja implementowanej aplikacji na urządzeniu fizycznym.

Same wymagania zostały podzielone odpowiednio na wymagania funkcjonalne i niefunkcjonalne.

2.1 Wymagania funkcjonalne

- System musi umożliwiać rozgłaszanie swojej obecności w sieci lokalnej, by móc zostać wykrytym przez innych klientów.
- System musi umożliwiać wyszukiwanie innych aktywnych klientów znajdujących się w zakresie lokalnego otoczenia oraz lokalnej sieci.
- System musi pozwalać na wysyłanie zaproszeń do połączenia się z wykrytymi klientami.
- System musi na bieżąco wyświetlać listę aktualnie połączonych klientów.
- System musi umożliwiać utworzenie nowej notatki tekstowej.
- System musi umożliwiać edycję treści istniejącej notatki.
- System musi umożliwiać trwałe usunięcie notatki.
- System musi automatycznie tworzyć unikalny identyfikator oraz znacznik czasowy dla nowych notatek.
- System musi pozwalać na zdefiniowanie tytułu notatki, będącego niezależnym parametrem od treści notatki.
- System musi enkodować strukturę danych notatki do formatu umożliwiającego przesył notatki do połączonych klientów.
- System musi dekodować otrzymane zakodowane dane o notatce i przekształcić je w natywny obiekt reprezentujący notatkę w systemie.
- System musi automatycznie rozsyłać zaktualizowaną treść notatki do wszystkich aktualnie połączonych klientów w jak najkrótszym czasie od wykrycia zmian.

- System musi nadpisać istniejącą notatkę nowo otrzymaną kopią, gdy obie posiadają ten sam identyfikator, ale otrzymana kopia zawiera nowszy znacznik czasowy.
- System musi zapisywać wszystkie notatki w trwałej pamięci urządzenia.
- System musi odświeżać interfejs z listą notatek w czasie rzeczywistym.

2.2 Wymagania niefunkcjonalne

- System musi wspierać urządzenia mobilne firmy Apple z zainstalowanymi systemami operacyjnymi iOS lub iPadOS w wersji 18.0 lub wyższej.
- Kod źródłowy powinien być napisany w języku Swift z wykorzystaniem deklaratywnego frameworka do budowy interfejsów graficznych - SwiftUI.
- Operacje zapisu do plików oraz procesy sieciowe muszą być wykonywane poza głównym wątkiem - wątkiem obsługującym interfejs graficzny aplikacji.
- Czas propagacji zmian w notatce do innego połączonego klienta nie powinien wynosić więcej niż 1 sekunda.
- System musi obsłużyć przypadek zerwania połączenia, bez uszkodzenia notatki oraz rzucania wyjątków uniemożliwiających dalsze funkcjonowanie systemu.
- System musi zapewnić szyfrowaną komunikację między klientami.
- Interfejs użytkownika musi być responsywny i dostosowywać się do różnych rozmiarów ekranów i ich orientacji w zakresie urządzeń tworzonych przez Apple.
- Wygląd aplikacji powinien spełniać oficjalne wytyczne projektowe Apple Human Interface Guideliens.
- System powinien automatycznie dostosowywać paletę kolorów interfejsu graficznego do aktualnie ustawionego motywu systemowego.

3 Implementacja

4 Testowanie i weryfikacja

4.1 Metodologia testowania

4.2 Scenariusze testowe

4.3 Analiza wydajności i zużycia zasobów

5 Podsumowanie i kierunki rozwoju

5.1 Osiągnięte cele

5.2 Ocena spełnienia wymagań

5.3 Możliwości dalszej rozbudowy

Spis literatury

Artykuły

[1] Werner Vogels. „Eventually consistent”. W: 52.1 (2009-01), s. 40–44.

[2] Victor B. F. Gomes, Martin Kleppmann, Dominic P. Mulligan i Alastair R. Beresford. „Verifying strong eventual consistency in distributed systems”. W: 1.OOPSLA (2017-10).

[3] David A. Nichols, Pavel Curtis, Michael Dixon i John Lamping. „High-latency, low-bandwidth windowing in the Jupiter collaboration system”. W: (1995), s. 111–120.

[4] Nicolas Vidot, Michelle Cart, Jean Ferrié i Maher Suleiman. „Copies convergence in a distributed real-time collaborative environment”. W: (2000), s. 171–180.

[5] Gérald Oster, Pascal Molli, Pascal Urso i Abdessamad Imine. „Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems”. W: (2006), s. 1–10.

[6] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim i Joonwon Lee. „Replicated abstract data types: Building blocks for collaborative applications”. W: 71.3 (2011), s. 354–368.

[7] Gérald Oster, Pascal Urso, Pascal Molli i Abdessamad Imine. „Data consistency for P2P collaborative editing”. W: (2006), s. 259–268.

[8] Nuno Preguica, Joan Manuel Marques, Marc Shapiro i Mihai Letia. „A Commutative Replicated Data Type for Cooperative Editing”. W: (2009), s. 395–403.

Źródła internetowe i inne

[9] Apache. *Data versioning - Apache Cassandra Documentation*. URL: `https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html#data-versioning`.

[10] Microsoft. *Peer To Peer Transactional Replication - Microsoft SQL Server documentation*. URL: `https://learn.microsoft.com/en-us/sql/relational-databases/replication/transactional/peer-to-peer-transactional-replication?view=sql-server-ver17`.

[11] Weixin Yu i Kaylee Xie. *A Paxos-Based Strongly Consistent Live Document Editor*. URL: `https://www.scs.stanford.edu/26wi-cs244c/proj/live_document_editor.pdf`.

[12] John Day-Richter. *What's Different About New Google Docs*. URL: `https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html`.

[13] Evan Wallace. *How Figma's Multiplayer Technology Works*. URL: `https://www.figma.com/blog/how-figmas-multiplayer-technology-works/`.