Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa magisterska

TEAMSYNC — SYSTEM WYMIANY DANYCH I KOMENTARZY W SYSTEMACH P2P

Filip Rachwalak

Promotor
dr inż. Anna Kobusińska

Poznań, 2015

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wstęp

W dobie coraz łatwiejszego i szybszego dostępu do sieci globalnej oraz rosnącej liczby urządzeń przetwarzających dane zachodzi coraz pilniejsza potrzeba sprawnej i bezpiecznej wymiany informacji. Twórcy obecnych narzędzi oferują coraz więcej dodatkowych funkcjonalności towarzyszących wymianie danych. Mogą się one różnić w zależności od architektury stworzonego systemu, ze względu na ograniczenia, które architektura ze sobą niesie. Głównym aspektem rozpatrywanym w ramach niniejszej pracy jest wymiana danych w postaci plików różnego typu, udostępnianych wzajemnie pomiędzy użytkownikami systemu.

Jedną z ważnych dodatkowych funkcji systemów umożliwiających wymianę danych jest możliwość komentowania zamieszczanych plików przez użytkowników, którzy udostępniają swoje dane. W przypadku systemów scentralizowanych jest to bardzo często dodawana usługa, ponieważ implementacja oraz utrzymanie takich systemów są zdecydowanie łatwiejsze. W centralnym systemie komunikaty oraz pliki można przechowywać i odczytywać w łatwy sposób w centralnej części systemu. Jednak dane umieszczane w takich systemach są mniej bezpieczne i narażone na brak dostępu w przypadku awarii części centralnej.

Chcąc wymieniać (i przechowywać) dane bezpieczniej, można wybrać rozwiązania o rozproszonej architekturze. Aplikacja *TeamSync* zaimplementowana w ramach niniejszej pracy magisterskiej jest narzędziem, które łączy funkcjonalność synchronizacji danych z możliwością wymiany opinii na ich temat poprzez system komentarzy pogrupowanych w sposób ułatwiający ich umieszczanie oraz czytanie. W swojej ostatecznej postaci umożliwia wygodne wprowadzanie swoich opinii przez użytkowników oraz łatwe i szybkie wyszukiwanie informacji wewnątrz nich. Poprzez wygodny interfejs graficzny umożliwiona została nawigacja po synchronizowanych katalogach i możliwość wybrania pliku, o którym można rozpocząć dyskusję. Funkcjonalność systemu obejmuje również bardziej zaawansowane operacje wykonywane na danych — filtrowanie oraz sortowanie wątków i funkcje wyszukiwania wpisywanych fraz wewnątrz komentarzy.

System *TeamSync* do komunikacji oraz wymiany danych wykorzystuje narzędzie *BitTorrent Sync*. Po uruchomieniu aplikacji *TeamSync* w tle rozpoczyna działanie system *BitTorrent Sync* i obsługuje wymianę danych pomiędzy użytkownikami.

Struktura pracy jest następująca. W rozdziale 2 przedstawiono zestawienie oraz krótki opis kilku istniejących rozwiązań, które swoją funkcjonalnością zbliżone są do aplikacji zaimplementowanej w ramach niniejszej pracy. Wśród nich znajduje się wspomniany powyżej *BitTorrent Sync*. Rozdział 3 zawiera opis modelu systemu — przyjęte założenia, nałożone ograniczenia oraz podstawowe obiekty i struktury opisywane w pracy — komentarze, wątki, foldery. Omawiana jest również wewnątrz niego spójność synchronizowanych danych.

Rozdział 4 poświęcony jest architekturze systemu. Opisana zostały ogólna idea działania, protokół komunikacyjny z aplikacją *BitTorrent Sync* oraz dokładne dane, które są przechowywane w plikach komentarzy oraz wątków. W rozdziale 5 omówiono działanie systemu — jak wyglądają tworzone struktury folderów w systemie plików oraz pliki konfiguracyjne przechowujące dane aplikacji.

Rozdział 6 zawiera opis zastosowanych w projekcie technologii oraz sposób, w jaki część kliencka i serwerowa komunikują się pomiędzy sobą. Dodatkowo w tym rozdziale wyjaśnione zostały przykładowe polecenia wykonywane przez serwer oraz sposób działania użytych technologii. Rozdział 7 stanowi podsumowanie pracy wraz z wnioskami oraz możliwymi kierunkami rozwoju aplikacji.

W dodatku A znajduje się opis graficznego interfejsu użytkownika wraz ze szczegółami dotyczącymi trybu jego pracy oraz wskazówki dotyczące stosowania filtrów wewnątrz aplikacji. W dodatku znajdują się również przykłady użycia wybranych funkcji interfejsu graficznego.

Rozdział 2

Przegląd istniejących rozwiązań

W niniejszym rozdziale przedstawiono istniejące rozwiązania, które oferują funkcjonalność zbliżoną do zaproponowanego w pracy magisterskiej.

Do zestawienia wybrano rozwiązania reprezentujące grupę systemów o bardzo zbliżonych wymaganiach — pominięto usługi, które powielają funkcjonalność swojego reprezentanta. Każdy przedstawiciel grupy różni się od przedstawicieli pozostałych grup architekturą, sposobem przechowywania danych lub dodatkową funkcjonalnością. W zestawieniu wzięto pod uwagę rozwiązania prezentujące różne podejście do tego samego problemu i aby je porównać będą rozważane przede wszystkim następujące właściwości każdego z nich:

- architektura (centralna/rozproszona),
- lokalizacja przechowywanych danych (dane dostępne lokalnie/zdalnie),
- możliwość współdzielenia plików z innymi użytkownikami (tak/nie),
- możliwość komentowania plików (tak/nie).

Podczas tworzenia zestawienia pominięto kontekst odpłatności za usługi, gdyż nie jest on związany z właściwościami systemów (nie wpływa na funkcjonalność, tylko na ograniczenia bądź udogodnienia świadczone przez dostawców usług). Po omówieniu wszystkich systemów wymienione wyżej parametry każdego z rozwiązań zostaną zebrane w jednej tabeli porównawczej.

Dropbox

Najpopularniejszym serwisem oferującym usługi przechowywania oraz zaawansowanego współdzielenia plików jest Dropbox [1]. Firma pojawiła się na rynku we wrześniu 2008 roku, a już w maju 2014 roku korzystało z jej usług ponad 300 milionów użytkowników. Obecność serwisu Dropbox w poniższym zestawieniu tłumaczy nie tylko jego popularność, ale też fakt skupiania w sobie wszystkich funkcji oferowanych przez większość systemów umożliwiających przechowywanie danych w Internecie.

Dropbox jest usługą, która umożliwia zsynchronizowanie lokalnego folderu ze zdalnym serwerem — w momencie lokalnej zmiany lub gdy zmieni się plik na serwerze (np. inny użytkownik wprowadzi zmianę), rozpoczyna się natychmiastowa synchronizacja. Dodatkowo użytkownik ma możliwość przeglądania zawartości folderu poprzez przeglądarkę internetową na dowolnym komputerze z dostępnej do sieci globalnej. Nie musi więc lokalnie posiadać plików, aby mieć do nich dostęp.

Działanie aplikacji klienckiej systemu Dropbox można przedstawić w następujący sposób: program nasłuchuje, czy któryś z plików lokalnych się zmienił, a następnie (w przypadku, gdy tak

było) uspójnia dane z serwerem. Jednocześnie aplikacja odpytuje serwer, czy w międzyczasie nie zaszły zmiany w zawartości plików na serwerze i w przypadku zajścia zmian, uspójnienie następuje po stronie aplikacji klienta — na lokalnym dysku dane zostają odświeżone do najnowszej wersji. Dane wymieniane są przy pomocy protokołu SSL i składowane na serwerach Amazon S3[1] poprzez szyfrowanie algorytmem AES256 [23] — niestety kluczem „należącym" do usługodawcy, nie użytkownika.

Dodatkowo Dropbox umożliwia współdzielenie folderów między różnymi użytkownikami — właściciel folderu może udostępnić innym cały folder albo poszczególne pliki. System oferuje też przywracanie danych — w przypadku chęci odtworzenia stanu sprzed jakiegoś czasu można to zrobić, ale nie później niż 30 dni wstecz. Aktualnie w narzędziu Dropbox nie ma możliwości zamieszczania komentarzy przez użytkowników współdzielących folder.

Wśród systemów o identycznej funkcjonalności (pod względem wybranych cech, których dotyczy zestawienie), znajdują się takie narzędzia jak: SugarSync [2], SkyDrive [3] oraz MediaFire [4].

box

System box [5] jest pierwowzorem systemu Dropbox. Jest jednak mniej popularny ze względu na odmienną grupę docelową użytkowników, którą twórcy wybrali na początku działania systemu. Box kierowany był głównie dla firm, przez co stracił na popularności, gdy zorientowany na prywatnych użytkowników Dropbox pojawił się na rynku. Box został włączony do tego zestawienia ze względu na swoją dodatkową (względem Dropboxa) funkcjonalność — wysoko rozwinięte narzędzia ułatwiające współpracę wielu użytkowników nad plikami, w tym przede wszystkim umieszczanie komentarzy.

Box — podobnie jak jego popularniejszy odpowiednik — ma centralną architekturę. Klient może umieścić swoje dane na serwerze usługodawcy i zsynchronizować je pomiędzy tym serwerem a swoim lokalnym folderem. W ten sposób użytkownik ma dostęp do danych zarówno lokalnie, jak i zdalnie — poprzez przeglądarkę z każdego komputera podłączonego do Internetu lub aplikację kliencką dostępną na komputery stacjonarne oraz urządzenia mobilne.

Aplikacja kliencka działa bardzo podobnie do aplikacji zaimplementowanej w ramach systemu Dropbox — sprawdzane są zmiany w plikach (lokalnie oraz zdalnie poprzez odpytywanie serwera) i w przypadku wykrycia zmian następuje uspójnienie danych. System udostępnia rozbudowaną funkcjonalność w obszarze współpracy użytkowników. Poza współdzieleniem oraz udostępnianiem danych jest również możliwość komentowania plików, co powoduje, że usługa box wyróżnia się na tle serwisów takich jak Dropbox i jemu pokrewnych.

Istnieją aplikacje niemal identyczne (oczywiście w kontekście wybranych cech porównawczych), które jednak wprowadzone do powyższego zestawienia tylko powieliłyby rodzaj architektury i funkcjonalność reprezentowane przez system box. Są to np. Google Drive [6], która integruje w sobie wiele rozwiązań i oprócz powielonych funkcji systemu box ma możliwość bardzo zaawansowanej pracy nad dokumentami, albo Mega [7] — usługa, która wprowadza możliwość rozmowy audio/video między współdzielącymi dane.

[1] System Dropbox nie zapewnia fizycznej infrastruktury swoim klientom — korzysta z oferty firmy Amazon o nazwie Amazon S3. Strona internetowa usługi: `http://aws.amazon.com/s3/`

sher.ly

Sher.ly [8] jest systemem, który przy zachowaniu architektury (centralna) różni się od poprzednich lokalizacją udostępnianych danych. Usługa w swojej podstawowej funkcjonalności pozwala na udostępnienie innym użytkownikom danych lokalnych. Klient wybiera plik bądź folder z lokalnego dysku, za pomocą aplikacji przeglądarkowej tworzy grupę, do której zaprasza innych użytkowników, a następnie udostępnia wybrany wcześniej plik/folder. Współpracownicy pobierają dane i mogą je edytować. Dane nie są domyślnie kopiowane do lokalnego systemu plików, dzięki czemu oszczędzane jest miejsce po stronie klienta. Tym samym zwiększone jest bezpieczeństwo danych — w momencie np. zakończenia współpracy nad umową z którymś prawnikiem można usunąć go z grupy, przez co nie będzie miał już dostępu do danych.

Centralny element w tym systemie nie jest w stałej lokalizacji. Użytkownik oprócz udostępniania lokalnych danych może również udostępniać pliki ze zdalnego serwera — własnego serwera NAS (Network Area Storage), wykupionego serwera z miejscem do składowania danych, albo nawet usługi Dropbox. W takiej sytuacji centrum systemu nie będzie skupione na komputerze użytkownika, tylko na zdalnym serwerze.

Sher.ly to usługa zorientowana na współpracę pomiędzy użytkownikami, dlatego też posiada dużo udogodnień z tym związanych. Użytkownik będący właścicielem grupy ma pełną kontrolę nad swoimi współpracownikami i danymi, które im udostępnia. Kolejną funkcjonalnością, której nie mają wszystkie pozostałe usługi w zestawieniu, jest możliwość umieszczania komentarzy o danych plikach. Stanowi to podstawę pracy wewnątrz tego systemu, w którym komunikacja jest najważniejsza.

Do transportu danych, sher.ly nie korzysta (jak większość podobnych systemów) z protokołów CIFS [9]/SAMBA [10], lecz ze swojego protokołu, który znacznie lepiej wykorzystuje przepustowość łącza (50% CIFS/SAMBA przeciwko 94% protokołu twórców sher.ly). Dodatkowo, sher.ly łączy się bezpośrednio między źródłem danych, a odbiorcą — oznacza to, że jeśli użytkownik udostępnia pozostałym użytkownikom dane ze swojego lokalnego dysku, transportowane są bezpośrednio do nich, bez żadnego komputera pośredniczącego. Twórcy protokołu porównują go do tworzenia połączenia VPN do każdego pliku lub folderu, który ma zostać udostępniony.

Symform

Symform [11] jest projektem najbardziej odbiegającym architekturą i sposobem działania od pozostałych systemów. Jest to w pełni rozproszone współdzielenie danych poprzez ich podział i rozdystrybuowanie ich pomiędzy innych użytkowników sieci. Odbywa się to w następujący sposób:

1. Plik dzielony jest na bloki.
2. Każdy blok szyfrowany jest za pomocą algorytmu AES256.
3. Zaszyfrowany blok dzielony jest na 64 równe części.
4. Do każdego bloku dodawane są 32 fragmenty parzystości obliczone za pomocą algorytmu kodowania korekcyjnego Reeda-Solomona [25].
5. Uzyskane 96 fragmentów rozdzielamy losowo do 96 różnych urządzeń podłączonych do sieci (ze względu na bezpieczeństwo jeden węzeł nie może posiadać więcej niż jednego fragmentu z konkretnego bloku).

Po tych krokach, aby uzyskać dostęp do danych (a więc zrekonstruować blok) wystarczy 64 fragmenty z uzyskanych wcześniej 96. Jest to bardzo bezpieczne składowanie danych, ponieważ aby je

złamać, należałoby znaleźć 64 fragmenty rozproszone po całej sieci, połączyć je w jeden zaszyfrowany blok, rozszyfrować blok (AES256), a następnie powtórzyć tę samą operację dla wszystkich bloków, z których składa się plik.

Symform to rozwiązanie, które nie jest ukierunkowane do udostępniania danych innym użytkownikom. Jest systemem służącym jako bezpieczna, rozproszona kopia zapasowa danych. Dostępna dla użytkownika przestrzeń to suma przestrzeni udzielonych mu przez innych użytkowników — może również sam udostępniać innym klientom swoje miejsce na dysku, dzięki czemu zyskuje dodatkową przestrzeń w sieci na składowanie plików. Za miejsce w sieci, którym użytkownik może dysponować, nie płaci pieniędzmi, tylko własnym miejscem udostępnianym innym klientom.

Z racji charakterystyki systemu oraz braku możliwości współdzielenia folderów Symform nie umożliwia komentowania plików pomiędzy użytkownikami. Dostęp do danych — podobnie jak w klasycznych rozwiązaniach centralnych — może odbywać się zarówno lokalnie, jak i zdalnie, z wykorzystaniem przeglądarki internetowej.

BitTorrent Sync

BitTorrent Sync [12] to system umożliwiający zdalne synchronizowanie folderów, który transportuje dane pomiędzy rozproszonymi użytkownikami za pomocą protokołu BitTorrent [27]. Protokół ten wykorzystuje do komunikacji zewnętrzny serwer — tzw. *tracker*, który odpowiada za odsyłanie do klientów informacji z adresami pozostałych węzłów w sieci. Architektura aplikacji BitTorrent Sync jest rozproszona, a brak centralnego serwera w systemie (nie licząc *trackera*) oraz szyfrowanie komunikacji zapewnia bezpieczniejszy i szybszy transport — w momencie synchronizacji fragmenty danych pobierane są z wielu źródeł jednocześnie.

Ze względu na zastosowaną architekturę, system BitTorrent Sync nie oferuje zdalnego dostępu do synchronizowanych danych. Aby uzyskać dostęp do plików, użytkownik musi najpierw uspójnić dane z innym użytkownikiem, który w danym momencie ma uruchomiony komputer i łączność z globalną siecią. Nie ma możliwości — tak jak w przypadku pozostałych omawianych systemów — przeglądania swoich danych z przeglądarki dowolnego komputera, który jest podłączony do sieci globalnej.

W celu synchronizacji folderów użytkownik posługuje się 32-znakowym ciągiem cyfr oraz liczb, określanym jako *secret*. Za jego pomocą aplikacje dwóch klientów odnajdują się w sieci i mogą uspójnić dane. Możliwa jest zmiana sposobu wykrywania się wzajemnie przez węzły — poprzez *DHT* [13] [29] [30], *trackera* lub tylko w lokalnej sieci.

Poza częścią aplikacji, która zarządza wymianą danych (wykrywanie zmian w plikach, wyszukiwanie nowych węzłów, transport danych) dostępna jest również aplikacja przeglądarkowa, która umożliwia dodawanie nowych i zarządzanie już zsynchronizowanymi folderami oraz administrowanie użytkownikami podłączonymi do folderu. Twórcy narzędzia umożliwili też komunikację z systemem za pomocą wystawionego API, nasłuchującego przychodzących żądań, za pomocą których możliwe jest sterowanie aplikacją z poziomu kodu programisty.

BitTorrent Sync jest systemem zorientowanym na współdzielenie danych, mimo to nie posiada funkcjonalności umożliwiającej umieszczanie komentarzy w plikach. Usługa skupia się na szybkiej wymianie dużej ilości danych między wieloma użytkownikami, aby maksymalnie wykorzystać zalety protokołu BitTorrent.

Podsumowanie

W tej części rozdziału opisane wcześniej rozwiązania zostaną krótko podsumowane pod kątem ich podstawowych cech, a następnie zostaną zestawione w tabeli.

System Dropbox jest rozwiązaniem o centralnej architekturze oferującym miejsce do przechowywania plików oraz aplikację przeglądarkową, za pomocą której każdy użytkownik może przeglądać swoje dane. Dostęp do plików jest zarówno lokalny (po zsynchronizowaniu z serwerem), jak i zdalny. Pomimo rozwiniętej kooperacji użytkowników (współdzielenie oraz udostępnianie danych w trybie „read only"), nie ma możliwości wymiany komentarzy dotyczących danych.

Box jest narzędziem bardzo podobnym do systemu Dropbox — również posiada centralną architekturę oraz zarówno zdalny, jak i lokalny dostęp do synchronizowanych danych. Przewagą usługi Box jest możliwość zamieszczania przez użytkowników współdzielących dane komentarzy wewnątrz aplikacji przeglądarkowej. Jest to system jeszcze bardziej zorientowany na współpracę niż Dropbox.

Narzędzie sher.ly jest również rozwiązaniem centralnym, z tą różnicą, że użytkownik ma możliwość wskazania źródła danych, które chce udostępnić — może to być plik na dysku lokalnego komputera, albo zewnętrznej lokalizacji np. wewnątrz własnego konta systemu Dropbox. Usługa bardzo ułatwia pracę wielu osób nad plikami i umożliwia kontrolę dostępu do udostępnianych danych. Zawiera również bardzo dobrze rozwinięte funkcje dzielenia się uwagami w postaci komentarzy.

Symform nie jest aplikacją, której priorytetem jest współpraca użytkowników i udostępnianie danych. Jego rozproszona architektura i zaawansowane algorytmy umieszczania danych w sieci umożliwiają bardzo bezpieczne przechowywanie danych. Narzędzie nie oferuje możliwości udostępniania plików innym użytkownikom, co w konsekwencji nie pozwala na wymianę komentarzy.

BitTorrent Sync o rozproszonym modelu skupia się głównie na szybkości wymiany danych oraz bezpieczeństwie komunikacji (ze względu na brak punktów pośredniczących w wymianie danych). Narzędzie Nastawione jest na współdzielenie plików, ale nie umożliwia wymiany komentarzy. Użytkownik nie ma też możliwości przeglądania danych przed pełnym ich zsynchronizowaniem do swojego lokalnego folderu, a synchronizacja jest możliwa tylko wtedy, gdy inny użytkownik z aktualnymi danymi jest obecnie podłączony do sieci.

Poniższa tabela podsumowuje omówione wcześniej ważniejsze — w kontekście niniejszej pracy — cechy istniejących rozwiązań.

System	**Architektura**	**Źródło danych**	**Współdzielenie danych**	**Komentarze**
Dropbox	centralna	lokalne/zdalne	tak	nie
sher.ly	centralna	lokalne/zdalne	tak	tak
box	centralna	lokalne/zdalne	tak	tak
Symform	rozproszona	lokalne/zdalne	nie	nie
BitTorrent Sync	rozproszona	lokalne	tak	nie

Tablica 2.1: Tabela zawierająca obecne rozwiązania wraz z ważniejszymi elementami funkcjonalności.

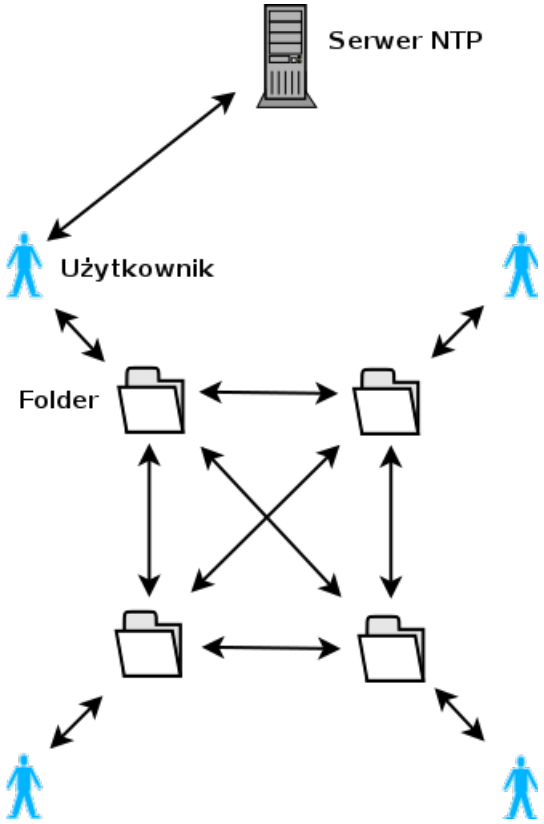
Zaprezentowane narzędzia pomimo wąskiego zakresu funkcjonalności („tylko" zdalne składowanie danych, udostępnianie ich i praca nad nimi z innymi użytkownikami), są bardzo różnorodne pod względem architektury oraz dodatkowych funkcji, które oferują. Jednak według najlepszej wiedzy autora, obecnie nie istnieje na rynku rozwiązanie, które — przy rozproszonej wymianie danych — umożliwiałoby wymianę komentarzy między współpracownikami synchronizującymi dane.

Rozdział 3

Model systemu

Aplikacja *TeamSync* działa w środowisku o rozproszonym charakterze — wszyscy użytkownicy w systemie są postrzegani jako odrębne i niezależne węzły (stacje robocze) w pewnej określonej lokalizacji w sieci. Każdy z użytkowników posiada unikalny identyfikator jednoznacznie wskazujący na niego. Model zakłada, że istnieje pomiędzy wszystkimi parami użytkowników łączność, która może być przerywana i wznawiana w dowolnych momentach.

Dodatkowo, w systemie znajduje się element centralny — serwer NTP [14] [26], który odpowiada na komunikaty użytkowników, odsyłając im znaczniki czasowe serwerowego czasu w momencie odebrania żądania. Serwer NTP zapewnia swoją dostępność za pomocą replikacji wewnątrz własnej ustrukturyzowanej sieci i w powyższym modelu zakładana jest jego bezawaryjność.



Rys. 3.1: Model systemu *TeamSync*.

Komunikacja i wymiana danych pomiędzy użytkownikami zachodzi tylko i wyłącznie poprzez

aplikację BitTorrent Sync. Wymiana danych odbywa się automatycznie w momencie umieszczenia przez użytkownika nowych plików we wskazanym wcześniej folderze, lub w chwili modyfikacji już istniejących danych wewnątrz niego. Na poziomie transportu danych, jest on realizowany za pomocą protokołu BitTorrent.

Głównym obiektem, wewnątrz którego dane są obustronnie synchronizowane, jest folder — w dalszej części pracy zastępowany również słowem „katalog" o tożsamym znaczeniu. Po synchronizacji folderów każdy z węzłów posiada pełną kopię uwspólnianych danych wewnątrz swojego lokalnego folderu. Użytkownicy łączą wzajemnie swoje katalogi poprzez podanie ciągu 32 znaków (jest to tzw. *secret*), za pomocą którego narzędzie BitTorrent Sync na jednym węźle jest w stanie odszukać u innego użytkownika zdalny folder, który oznaczony jest tym samym *secretem*.

System BitTorrent Sync gwarantuje natychmiastowe rozpoczęcie synchronizacji wszystkich danych wewnątrz folderu pomiędzy grupą połączonych ze sobą użytkowników. Wymiana danych pomiędzy dwoma węzłami odbywa się bezpośrednio — bez serwera — i nie jest możliwa, jeśli nie mają one możliwości nawiązania połączenia. Dodatkową konsekwencją takiej sytuacji jest fakt, że aby rozpropagowanie modyfikacji danych w sieci mogło mieć miejsce, przynajmniej dwaj użytkownicy muszą być do niej podłączeni.

Katalog synchronizowany przez użytkowników jest folderem zawierającym dowolną ilość danych, ograniczoną wyłącznie przez ilość dostępnej pamięci w systemie operacyjnym użytkownika. Pliki i katalogi wewnątrz współdzielonego folderu mogą być dowolnego typu, a dowolność ich nazw ogranicza jedynie system operacyjny.

Spójność wymienianych danych

Użytkownicy synchronizujący ze sobą dane wewnątrz folderów — ze względu na częstą zmianę dostępności każdego z węzłów — narażeni są na wiele sytuacji grożących brakiem spójności wymienianych danych. Każdy z użytkowników posiada swoją lokalną, pełną kopię uwspólnionych plików. Podczas modyfikacji jednego z nich, zmiany rozsyłane są do pozostałych *obecnych w danym momencie w sieci* węzłów — w przypadku nieobecności dowolnego z użytkowników, zmiana zostanie odłożona w czasie do momentu, w którym będzie on dostępny w sieci. Wówczas — o ile dowolny z węzłów posiadający dane po zmianie również jest obecny w sieci — zmiany zostają natychmiast rozpropagowane.

Dodatkowo im więcej węzłów posiadających zmodyfikowane dane jest w sieci, tym szybciej dane będą się synchronizować. Dzieje się tak dlatego, że do transportu użyto protokołu *BitTorrent*, za pomocą którego pobierane są drobne fragmenty danych od wszystkich dostępnych węzłów je posiadających, a dopiero po pobraniu fragmentów następuje łączenie ich w całość. Pozwala to na znacznie szybszą i bezpieczniejszą wymianę danych pomiędzy użytkownikami wewnątrz systemu *BitTorrent Sync*, ponieważ fragmenty danych pobierane są od pozostałych węzłów równolegle.

Jednak pomimo cennych korzyści, podstawową trudnością jest zachowanie spójności w rozproszonym systemie. *BitTorrent Sync* gwarantuje, że po uspójnieniu danych przez wszystkich użytkowników w systemie — niezależnie od wcześniejszych różnic w przechowywanych lokalnie danych — wszyscy użytkownicy ostatecznie będą posiadali *jednakową* kopię danych. Ich lokalne zmiany, które nie zostały rozpropagowane w sieci zostaną dodane do katalogu `.sync` wewnątrz synchronizowanego katalogu, aby nie utracili oni swoich wersji zmodyfikowanych danych. Zawartość katalogu `.sync` nie jest przesyłana w sieci — każdy węzeł ma inną jego zawartość.

Komunikacja między użytkownikami

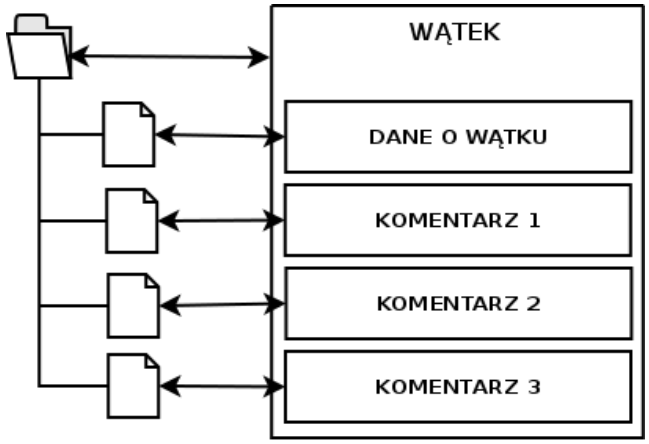
Podstawowymi obiektami, za pomocą których użytkownicy mogą się porozumiewać, są komentarze. Komentarz można umieścić w systemie *TeamSync* za pomocą interfejsu graficznego, który dodaje go w odpowiedni sposób do systemu plików synchronizowanego folderu i rozsyła do pozostałych węzłów w sieci. Komentarz może zostać umieszczony w systemie przez dowolnego użytkownika, natomiast fizycznie w systemie przechowywany jest jako plik (na jeden komentarz w systemie przypada jeden plik) zawierający w sobie takie informacje jak:

- treść wypowiedzi,
- data i czas zamieszczenia,
- identyfikator autora,
- listę osób, które odczytały komentarz,
- historię zmian (poprzednia treść oraz data i czas zmiany).

Komentarze — w dalszej części pracy nazywane również „wypowiedziami", lub „postami" — zgrupowane są w większe struktury nazwane wątkami. Jeden wątek — w dalszej części pracy nazywany też „dyskusją" — może zawierać w sobie nieograniczoną ilość wypowiedzi i może zostać oznaczony w systemie jako dyskusja na temat konkretnego (wybranego przez autora wątku) pliku znajdującego się w folderze.

W systemie wątek reprezentowany jest przez specjalnie oznaczony (za pomocą jego nazwy) katalog, wewnątrz którego znajdują się pliki z komentarzami. Dodatkowo, wewnątrz folderu przechowującego wątek, pomiędzy plikami z komentarzami znajduje się plik zawierający podstawowe informacje o wątku, między innymi:

- tytuł dyskusji,
- identyfikator autora,
- data i czas rozpoczęcia watku,
- plik, którego wątek dotyczy.



Rys. 3.2: Odwzorowanie fizycznego składowania plików z komentarzami w wątku na logiczną ich reprezentację w systemie.

Na powyższym rysunku zaprezentowano, w jaki sposób fizycznie przechowywane są komentarze i w jaki sposób logicznie udostępniane są użytkownikom w systemie. Na jego podstawie można opisać kolejne założenie systemu, stanowiące o tym, że nie ma możliwości, aby w systemie komentarze istniały bez wątku — przynależność wypowiedzi do dyskusji jest obligatoryjna.

Użytkownik może mieć wiele (maksymalnie 10) folderów współdzielonych, każdy z innymi węzłami. Z tego powodu w modelu uwzględniono tzw. „tożsamości". Tożsamość to nazwa użytkownika wpisywana przez niego podczas dołączania do folderu, która jest wyświetlana innym użytkownikom. Jedna tożsamość obowiązuje w jednym folderze i nie ma ograniczeń dotyczących powtarzalności — użytkownik może w każdym uwspólnianym folderze mieć identyczną wyświetlaną nazwę.

Model systemu zakłada, że pliki raz zamieszczone w folderze nie będą zmieniać swojej lokalizacji wewnątrz niego. Założenie to może nie być przestrzegane tylko w przypadku, gdy żaden z wątków nie dotyczy przenoszonego pliku. W przeciwnym wypadku — jeśli plik zmieni lokalizację (nawet pozostając nadal w uwspólnionym folderze) albo nazwę — dyskusja dotycząca pliku nie zostanie wyświetlona. Powyższe założenie nie wnosi negatywnego działania w odniesieniu do synchronizowanych danych — dotyczy wyłącznie funkcjonalności komentowania.

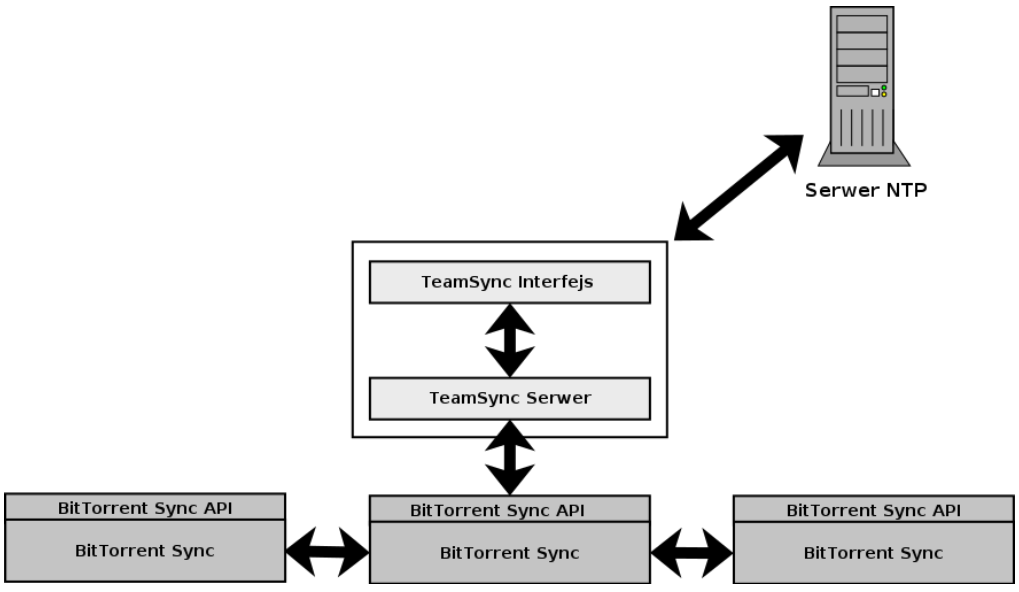
Wszystkie dane służące do obsługi systemu *TeamSync* — komentarze, pliki z danymi wątku, pliki przechowujące informacje o użytkownikach, pliki konfiguracyjne — zapisywane są w formie słownika według formatu `JSON`. Każdy ze słowników zawiera klucze (ciągi znaków określające daną właściwość) oraz ich wartości.

Rozdział 4

Ogólna koncepcja i architektura

Aplikacja *TeamSync* zrealizowana w ramach niniejszej pracy ma strukturę dwuwarstwową. Strony serwera oraz klienta, na wzór serwisów internetowych, są od siebie odizolowane, a komunikacja między nimi zachodzi z wykorzystaniem protokołu `HTTP` [15] [28]. BitTorrent Sync jest narzędziem współpracującym z aplikacją, które jest odpowiedzialne za wymianę danych (synchronizację). Komunikacja z tym narzędziem jest możliwa dzięki odpowiednio skonfigurowanemu API, które zostało udostępnione przez twórców całego systemu.

Działanie aplikacji *TeamSync* polega na uruchomieniu w tle systemu synchronizującego dane i udostępnienie użytkownikowi wygodnego interfejsu do umieszczania i odczytywania komentarzy, co odbywa się poprzez odpowiednią manipulację plikami wewnątrz synchronizowanych folderów. Jedyną drogą wymiany danych jest wspomniany system BitTorrent Sync.



Rys. 4.1: Uproszczona architektura systemu *TeamSync*.

Poniżej zostaną omówione w oddzielnych podrozdziałach poszczególne moduły architektury systemu: BitTorrent Sync, system *TeamSync* wraz z komunikacją pomiędzy jego częścią serwerową a kliencką oraz serwer NTP.

4.1 BitTorrent Sync

W poniższej sekcji opisanych zostanie kilka aspektów zarówno systemu BitTorrent Sync, jak i zaimplementowanemu przez jego twórców API, dzięki któremu możliwe jest wykonywanie większości funcji tego systemu z poziomu kodu. Rozdział rozpocznie sekcja *secret* opisująca łańcuch znaków wymieniany między użytkownikami, po której zostanie przedstawiony bardziej szczegółowy opis protokołu komunikacji z systemem BitTorrent Sync.

4.1.1 Secret

W aplikacji BitTorrent Sync *secret* jest ciągiem 32 znaków, służącym do identyfikacji synchronizowanych folderów. Dodając katalog do tego systemu, użytkownik może posłużyć się istniejącym już *secretem*, jeśli otrzymał go od użytkownika, który wcześniej zainicjował folder. Może też sam go zainicjować, generując *secret* tworząc losowy ciąg znaków odpowiadający wyrażeniu regularnemu `[A-Z0-9]{32}`, co odpowiada trzydziestu dwóm losowym znakom wybranym ze zbioru cyfr od 0 do 9 oraz liter alfabetu od `A` do `Z`.

Duża ilość znaków (32 znaki) oraz duży ich zbiór (24 litery + 10 cyfr daje łącznie moc zbioru znaków równą 34) powodują, że istnieje znikoma szansa na powtórzenie się dwóch łańcuchów znakowych.

Podczas generowania wartości *secret*, tworzone są dwa łańcuchy znaków o dwóch typach synchronizacji:

read_write — użytkownicy, którzy otrzymają *secret* pochodzący z tej wartości będą mogli zarówno odczytywać dane, jak i je modyfikować oraz dodawać swoje; modyfikacje danych — w przeciwieństwie do trybu `read_only` — będą się propagować do pozostałych użytkowników,

read_only — użytkownicy, którzy otrzymają ten *secret*, będą mogli wyłącznie pobierać dane, nie mogąc ich modyfikować (niezależnie od modyfikacji, zmiany nie będą propagowane do innych użytkowników).

W zależności od tego, który z *secretów* użytkownik otrzyma, tak system będzie uspójniał dane.

4.1.2 Protokół BitTorrent Sync API

Twórcy systemu BitTorrent Sync umożliwili korzystanie z dodatkowej funkcjonalności aplikacji nie tylko użytkownikom, ale również programistom chcącym testować lub udoskonalać narzędzia związane z wymianą plików. Sterowanie systemem z poziomu kodu jest możliwe dzięki zaimplementowanemu przez twórców aplikacji API, które umożliwia komunikację pomiędzy narzędziem a stworzoną przez programistę aplikacją.

API narzędzia odbiera żądania — z odpowiednio dobranymi parametrami — nasłuchując na porcie wskazanym przez użytkownika w pliku konfiguracyjnym (dokładny opis pliku konfiguracyjnego znajduje się w podrozdziale 5.2.1). Pełen adres, na który programista musi skierować żądanie HTTP ma postać wzorca:

```
http://<adres>:<port>/api
```

W przypadku aplikacji *TeamSync* zaimplementowanej w ramach niniejszej pracy magisterskiej, ze względu na lokalizację systemu BitTorrent Sync (lokalna maszyna) użyty adres to `localhost` oraz port `8787` (zmieniony z domyślnego `8888` ze względu na jego dużą popularność w aplikacjach). Pełen adres, na który są wysyłane żądania pomiędzy aplikacją *TeamSync*, a API systemu BitTorrent Sync to:

`http://localhost:8787/api`

Komunikaty BitTorrent Sync API

Żądania wysyłane z systemu *TeamSync* do API zawierają w nagłówku parametr uwierzytelniający *Basic HTTP Authentication* [16]. Dane dostępu zawierają wartości `login` oraz `password`, które — aby zostały poprawnie odebrane — muszą odpowiadać danym umieszczonym wewnątrz pliku konfiguracyjnego systemu BitTorrent Sync. Uruchamiając po raz pierwszy aplikację *TeamSync* dane uwierzytelniające są tworzone wraz z plikami konfiguracyjnymi i ustawione domyślnie na `team` oraz `sync` odpowiednio dla wartości `login` oraz `password`.

Zapytania przyjmowane przez BitTorrent Sync API zawierają dodatkowo tzw. „metodę" wraz z jej argumentami. Metoda oznacza funkcję, która zostanie wykonana na podanych argumentach np. pobieranie listy folderów wprowadzonych do systemu i współdzielonych z innymi użytkownikami, pobieranie listy użytkowników, z którymi użytkownik synchronizuje dany folder, zwrócenie ustawień konkretnego folderu lub ich zmiana i wiele innych. W aplikacji *TeamSync* użyto niewielkiej ilości możliwych metod ze względu na ich dużą liczebność i użyteczność odbiegającą od głównego celu projektu.

Użyte metody wraz z typem przyjmowanych argumentów zostaną opisane w oddzielnych sekcjach. Wszystkie zwroty użyte w poniższych podrozdziałach dotyczące aplikacji takie jak: „aplikacja", „system", „narzędzie", będą dotyczyły systemu BitTorrent Sync. Sposób działania narzędzia zaimplementowanego w ramach niniejszej pracy nie będzie omawiany podczas opisu poniższych metod.

Przykłady dotyczące sposobu wysyłania zapytań przez aplikację *TeamSync* są napisane według wzoru podyktowanego przez wymagania funkcji pakietu `requests` języka *Python*, które zostały użyte do wysyłania żądań do BitTorrent Sync API.

```
requests.get('http://adres:port/api',
             auth=('login', 'password'),
             params={
                 'method': 'nazwa_metody',
                 'arg1': 'wartość_argumentu',
                 [ . . . ]
             }
)
```

Pierwszy argument wewnątrz zapytania reprezentuje adres wraz z numerem portu, na który zostanie wysłane żądanie. Dane uwierzytelniające przesyłane są wewnątrz parametru `auth`, natomiast nazwa metody oraz pozostałe argumenty przesyłane są wewnątrz parametru `params`, który ma strukturę słownika z nazwą oraz wartością poszczególnych argumentów.

get_secrets

Metoda *get_secrets* nie wymaga żadnych dodatkowych argumentów i służy do generowania przez aplikację ciągów znakowych identyfikujących synchronizowany folder. Zapytanie wysyłane do BitTorrent Sync API w celu ich uzyskania:

```
requests.get('http://localhost:8787/api',
             auth=('team', 'sync'),
             params={
                 'method': 'get_secrets',
             }
)
```

Odpowiedź aplikacji będąca słownikiem w formacie `JSON` [24] zawiera dwie wartości:

- *read_write* — *secret* służący do synchronizacji folderu w trybie zapisu/odczytu,
- *read_only* — *secret*, za pomocą którego użytkownik może udostępnić swoje dane, nie pozwalając na modyfikację lub dodawanie danych innym użytkownikom.

add_folder

Metoda dodająca wskazany w argumencie folder do systemu. Po otrzymaniu żądania zawierającego tę metodę, do wybranego katalogu — który w momencie wykonywania metody musi być pusty (wymaganie systemu BitTorrent Sync) — aplikacja dodaje ukryty folder `.sync` (dokładny opis zawartości katalogu `.sync` znajduje się w podrozdziale 5.1) i od tego momentu katalog wskazany w żądaniu będzie włączony do synchronizacji.

Oprócz metody komunikat zawiera dodatkowe parametry:

- *dir* — pełna ścieżka do folderu,
- *secret* — losowy ciąg znaków będący tzw. „secretem", za pomocą którego możliwa jest synchronizacja katalogów między użytkownikami. Jeśli parametr ten pozostanie pusty, system otrzyma informację, że folder jest nowy i nie jest jeszcze synchronizowany przez żadnego z użytkowników. Aplikacja wygeneruje secret i od tego momentu użytkownik może przekazywać go innym węzłom, aby współdzielić wskazany katalog.

Zakładając, że synchronizowany ma zostać katalog o ścieżce `/home/user/testowy_katalog`, zapytanie zostanie wysłane z serwera aplikacji *TeamSync* w następujący sposób:

```
requests.get('http://localhost:8787/api',
             auth=('team', 'sync'),
             params={
                 'method': 'add_folder',
                 'dir': '/home/user/testowy_katalog',
                 'secret': 'A3LL43HJ257YCKMOLAD7QSEAS7U373BVO'
             }
)
```

W przypadku otrzymania takiego komunikatu z pustym argumentem `secret`, aplikacja wygenerowałaby nowy secret i zwróciłaby go w odpowiedzi. W innym przypadku, w odpowiedzi przesyłany jest wyłącznie kod błędu wewnątrz zmiennej `error`. W przypadku otrzymanej wartości równej 0 użytkownik jest informowany, że operacja dodawania nowego folderu przebiegła pomyślnie.

get_folders

Za pomocą tej metody otrzymywana jest lista folderów, które zostały wprowadzone do systemu. Podobnie jak przy metodzie *get_secrets* żadne dodatkowe argumenty nie są wymagane.

```
requests.get('http://localhost:8787/api',
             auth=('team', 'sync'),
             params={
                 'method': 'get_folders',
             }
)
```

W odpowiedzi na zapytanie *get_folders* BitTorrent Sync API zwraca listę, wewnątrz której znajdują się informacje o wszystkich synchronizowanych w systemie folderach w formacie `JSON`. Wśród kluczy słowników są:

- *error* — w przypadku niepowodzenia przesyłana jest tylko ta wartość ustawiona na 1; w przypadku powodzenia wartość 0,
- *dir* — bezwzględna ścieżka folderu w systemie plików lokalnego użytkownika,
- *files* — liczba plików, które znajdują się wewnątrz folderu,
- *size* — łączny rozmiar plików wewnątrz folderu w bajtach,
- *secret* — łańcuch znaków służący do identyfikacji współdzielonych folderów między zdalnymi użytkownikami (więcej w sekcji 4.1.1),
- *type* — rodzaj synchronizacji; możliwe dwie wartości:
  - *read_write* — użytkownicy mogą zarówno odczytywać, jak i modyfikować pliki,
  - *read_only* — tylko właściciel może modyfikować, pozostali użytkownicy wyłącznie odczytywać,
- *down_speed* — chwilowa prędkość pobierania; jeśli wszystkie dane w folderze są zsynchronizowane, wartość 0,
- *up_speed* — chwilowa prędkość wysyłania; jeśli wszystkie dane w folderze są zsynchronizowane, wartość 0,
- *id* — unikalny identyfikator folderu, który może różnić się pomiędzy użytkownikami.

Przykładowa odpowiedź BitTorrent Sync API na powyższe żądanie może wyglądać następująco:

```
[
    {
        "error": 0,
        "dir": "/home/user/testowy katalog",
        "files": 9,
        "size": 3079,
        "secret": "A3LL43HJ257YCKMOLAD7QSEAS7U373BVO",
        "type": "read_write",
        "down_speed": 0,
        "up_speed": 0,
        "id": 902958156496830188
    }
]
```

Otrzymanie takiej odpowiedzi wskazuje na fakt, iż jednym (i jedynym, ze względu na obecność tylko jednego obiektu typu `JSON` w liście katalogów) z synchronizowanych folderów w systemie jest folder o ścieżce `/home/user/testowy katalog`. Został on poprawnie odczytany przez system BitTorrent Sync (`'error': 0`) i zawiera dziewięć plików o łącznym rozmiarze 3079 bajtów. Aby uwspólnić go z innymi użytkownikami w trybie umożliwiającym im zarówno odczyt, jak i modyfikację zawartości (tryb `read_write`), należy przekazać *secret* o wartości `A3LL43HJ257YCKMOLAD7QSEAS7U373BVO`.

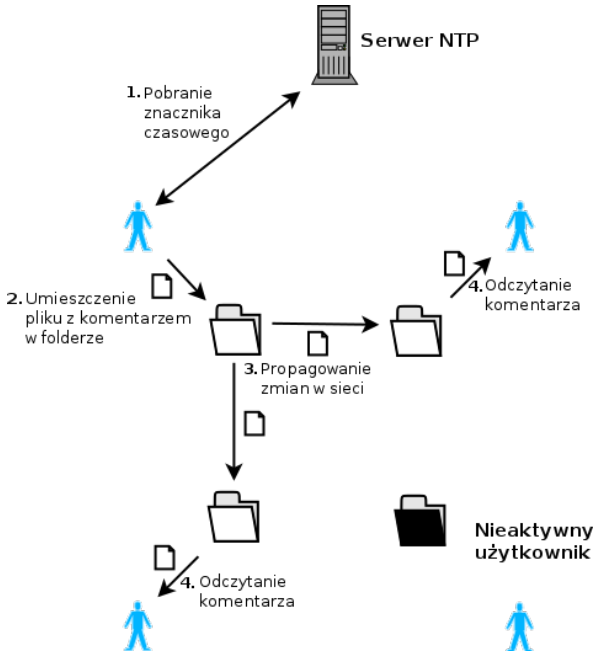
Wartości `down_speed` oraz `up_speed` sugerują brak wymiany danych między użytkownikami w chwili wysyłania żądania *get_folders*. Wewnętrznym identyfikatorem folderu w systemie lokalnym użytkownika jest liczba `902958156496830188`.

remove_folder

Ostatnią z używanych przez system *TeamSync* metod jest metoda *remove_folder*, która usuwa z systemu wskazany synchronizowany folder — nie usuwa fizycznie katalogu z systemu plików, lecz blokuje jego dalszą synchronizację. Aby ponownie użyć tego samego katalogu, należy usunąć z niego wszystkie pliki (wymaganie aplikacji dotyczące pustego folderu). W parametrze żądania — oprócz metody — aplikacja potrzebuje wartość *secret* folderu, który użytkownik zamierza usunąć.

```
requests.get('http://localhost:8787/api',
             auth=('team', 'sync'),
             params={
                 'method': 'remove_folder',
                 'secret': 'A3LL43HJ257YCKMOLAD7QSEAS7U373BVO'
             }
)
```

W przeciwieństwie do metody *add_folder*, w której parametr *secret* był opcjonalny, tutaj konieczne jest przesłanie tego parametru. Jeśli nie zostanie on przesłany lub okaże się niepoprawny, aplikacja zwróci kod błędu w zmiennej `error`. W przypadku poprawnego przebiegu operacji *remove_folder*, kod błędu będzie równy 0.



Rys. 4.2: Uproszczony schemat dodawania nowego komentarza do systemu i odczytanie go przez aktywnych użytkowników.

4.2 *TeamSync*

Po opisaniu modułu BitTorrent Sync API z Rys. 4.1 umieszczonego no początku rozdziału, przedstawiony zostanie dokładniej sposób działania systemu *TeamSync* — jak zapisywane są komentarze oraz wątki, jak są przekazywane w sieci oraz w jaki sposób system unika konfliktów.

4.2.1 Komentarze

Rozproszona architektura nie daje pewności, że modyfikacja pliku odbędzie się bezkonfliktowo, ponieważ zawsze może wystąpić sytuacja, w której inny węzeł „równocześnie" (pod względem zegara logicznego, nie czasu rzeczywistego) nie dokona edycji danych. Podczas testów aplikacji BitTorrent Sync, która jest odpowiedzialna za wymianę danych — a więc również za rozwiązywanie konfliktów — nie wyizolowano priorytetów określających, którą wersję i z którego węzła należy usunąć, a z którego rozpropagować.

Z powyżej opisanych powodów umieszczanie komentarzy w aplikacji *TeamSync* polega na *dodawaniu* nowych plików przez użytkowników. Algorytm dodawania nowego komentarza z lokalnej perspektywy wygląda następująco:

1. Pobierz znacznik czasowy z serwera NTP.
2. Umieść pobrane od użytkownika dane (treść komentarza, identyfikator) oraz pobrany wcześniej znacznik czasowy do zmiennej o strukturze słownika (JSON).
3. Zapisz słownik do pliku o nazwie złożonej ze znacznika czasowego oraz identyfikatora użytkownika wewnątrz katalogu z wątkiem.

Umieszczenie komentarza w systemie i rozpropagowanie go pomiędzy wszystkich aktywnych użytkowników zaprezentowane jest na rysunku 4.2.

Struktura komentarzy

Wszystkie zapisywane komentarze mają swoją strukturę, która — podobnie jak w przypadku wszystkich danych przesyłanych i zapisywanych do plików w aplikacji *TeamSync* — jest strukturą słownikową zapisaną w formacie JSON. Każdy plik komentarza zawiera pięć kluczy wraz z wartościami:

- *comment* — treść komentarza,
- *timestamp* — znacznik czasowy pobierany podczas powstawania komentarza,
- *history* — lista zawierająca obiekty o strukturze słownikowej, które przechowują wszelkie zmiany treści komentarza w wartościach kluczy `comment` oraz `timestamp`. W momencie zmiany treści przez autora wypowiedzi, dodawany jest nowy element w liście `history` z obiektem o tych kluczach z wartościami wypełnionymi nową treścią (`comment`) oraz nowym znacznikiem czasowym (`timestamp`),
- *uid* — autor komentarza,
- *readby* — zmienna o strukturze słownika przechowująca informację o użytkownikach, którzy odczytali komentarz i o czasie, w którym to zrobili. W kluczu znajduje się identyfikator autora, natomiast jako wartość wpisywany jest znacznik czasowy odczytania posta.

Po zatwierdzeniu zmiany wyświetlana jest najnowsza treść komentarza, natomiast znacznik czasowy pozostaje bez zmian. Taka implementacja jest wymuszona przez konieczność zachowania ciągłości logicznej konwersacji — edytowany post nie może przemieszczać się w chronologicznym porządku wymiany opinii, ponieważ część wypowiedzi może zostać źle odczytana w przypadku modyfikacji komentarzy przed nią. Intencje edycji zazwyczaj polegają na korygowaniu napisanych wcześniej zdań (interpunkcja, „literówki", ortografia) lub dodawaniu mniej znaczących faktów — rzadko zdarza się całkowita modyfikacja treści, a w przypadku gdy jednak wystąpi, użytkownicy mogą przejrzeć historię komentarzy, gdy dostrzegą logiczne niezgodności w przepływie opinii.

Konflikty

Struktura pliku z komentarzem może być bardzo dynamiczna i często zmieniana, ponieważ istnieje lista (`history`), w której dodawane sa elementy, oraz słownik (`readby`) uzupełniany o kolejne pary klucz-wartość. Czy wobec tego nie generuje to konfliktów, które mogłyby naruszyć spójność danych? Czy użytkownik `A` może znaleźć się w sytuacji, że „ominie" go część informacji, gdy użytkownik `B` nadpisze swoją część zamiast części użytkownika `C`? Lub czy jest możliwość, że modyfikacja użytkownika `A` zostanie pominięta, ponieważ w tym samym czasie użytkownik `B` dokonał zmian w tym samym komentarzu i „wygrał" synchronizację?

Odpowiedź brzmi: TAK, jest taka możliwość.

Modyfikacja komentarzy zachodzi w dwóch przypadkach: edycji treści oraz odczytaniu go przez użytkownika. Poniżej zostaną rozważone obydwa przypadki i w każdym z nich opisane zostaną możliwości konfliktów oraz ich skutki.

Modyfikacja treści — wystąpienie konfliktu w przypadku edytowania treści jest niemożliwe, ponieważ interfejs użytkownika pozwala na poprawienie treści tylko w przypadku, gdy jesteś autorem komentarza. Jeden użytkownik nie jest w stanie dokonać edycji dwóch postów jednocześnie.

Odznaczenie jako przeczytane — w tym przypadku konflikt może wystąpić, lecz nie będzie miał żadnych poważnych konsekwencji. Żeby to wyjaśnić, załóżmy, że użytkownicy `A` oraz `B` nie przeczytali jeszcze pewnego komentarza `x` i zrobią to w tym samym momencie (albo każdy

```
{
    "comment": "Ostateczna treść komentarza",
    "timestamp": "1439395604000",
    "history": [
        {
            "comment": "Pierwsza treść",
            "timestamp": "1439395604000"
        },
        {
            "comment": "Pierwsza treść zmodyfikowana",
            "timestamp": "1439395945000"
        },
        {
            "comment": "Ostateczna treść komentarza",
            "timestamp": "1439396291000"
        }
    ],
    "uid": "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH",
    "readby": {
        "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH": "1439395604000",
        "2SAFDSBCXGFD765GDFS4GFDS35FDSBVB": "1439395888000",
        "3REW54GFDS6578FDSGDF5BSH652F2B34": "1439396782000"
    }
}
```

Rys. 4.3: Struktura przykładowego komentarza.

z nich zrobi to na swojej lokalnej wersji, gdy żaden z pozostałych użytkowników należących do folderu nie będzie dostępny). W obecnej chwili (przed wzajemną synchronizacją) obydwaj użytkownicy mają oznaczony post `x` jako przeczytany. W momencie synchronizacji „wygra" wersja pliku któregoś z użytkowników. Załóżmy, że wygra użytkownik `A`: wówczas użytkownik `B` zobaczy w swojej aplikacji *TeamSync*, że ponownie ma nieprzeczytany komentarz, ale wówczas znowu aplikacja „przeczyta" komentarz ponownie odpowiednio go modyfikując (wstawiając nową parę klucz-wartość do słownika `readby`).
Nawet w przypadku częstych konfliktów, wpływa to nieznacznie tylko i wyłącznie na wygodę użytkownika (ponowne „kliknięcie" na nieprzeczytany wątek).

Przykładowy komentarz

Na rysunku 4.2.1 przedstawiona została struktura komentarza, który dla lepszego zobrazowania własnej struktury został odczytany przez trzech użytkowników i był modyfikowany dwukrotnie.

W poniższym przykładzie dobrze widać zmianę treści komentarza poprzez pobranie jej z najświeższej modyfikacji oraz niezmienność znacznika czasowego (`timestamp`) niezależnie od tego, czy komentarz był modyfikowany. Zauważalna jest też łatwość, z jaką można odtworzyć całą historię zmian wypowiedzi. Oglądanie zmian wszystkich komentarzy jest dostępne dla każdego z użytkowników niezależnie od przynależności, o ile taka historia dla konkretnego przypadku istnieje.

Dzięki słownikowi `readby` można odczytać, który z użytkowników i w jakim czasie odczytał ten komentarz: na przykład użytkownik o identyfikatorze `2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH` podczas odczytywania wypowiedzi umieścił swój wpis, pobierając znacznik czasowy `1439395604000`, co odpowiada dacie `12.08.2015r.` i godzinie `18:06` czasu polskiego.

4.2.2 Wątki

Wewnątrz każdego folderu z wątkiem oprócz plików z komentarzami znajduje się plik o nazwie `meta`, który zawiera dane związane z wątkiem prezentowane w interfejsie użytkownika. Podobnie jak w przypadku komentarzy, są one ustrukturyzowane jako para klucz/wartość w formacie JSON. Klucze znajdujące się w pliku `meta` obejmują następujące informacje:

- *uid* — autor wątku,
- *timestamp* — znacznik czasowy pobrany podczas tworzenia wątku (identyczny ze znacznikiem czasowym pierwszego komentarza),
- *fileabout* — ścieżka wewnątrz folderu wskazująca na plik, którego dotyczy watek,
- *topic* — tytuł watku.

Plik `meta` został wprowadzony w celu szybszego odczytywania listy wątków — podczas przygotowywania listy, odczytywanie przez aplikację komentarzy jest niepotrzebne, dzięki czemu aplikacja działa szybciej. Pobieranie komentarzy z folderu zawierającego wątek (treści, znaczników czasowych itd.) następuje *po* wybraniu konkretnej dyskusji.

Informacje dodatkowe wątku (np. ilość napisanych komentarzy, znacznik czasowy najświeższego komentarza) są na bieżąco obliczane przez aplikację podczas pobierania listy wątków. Znaczniej łatwiej (oraz szybciej) byłoby zapisać te dane w pliku `meta` i odczytywać je z otwartego już pliku. Natomiast wówczas pojawiłaby się możliwość wystąpienia konfliktów podczas jego edycji. Węzeł, który dodawałby nowy komentarz, musiałby edytować plik `meta` i inkrementować licznik komentarzy oraz zmodyfikować czas najświeższego komentarza. W przypadku dwóch użytkowników dodających komentarz w tym samym czasie dochodziłoby do konfliktów i umieszczania fałszywych danych wewnątrz pliku.

Dwie z informacji o wątku — autor oraz znacznik czasowy — są powielone z pierwszego komentarza w dyskusji (ponieważ podczas zakładania wątku umieszczany jest również inicjujący go komentarz). Zaimplementowano to w taki sposób, aby mieć dostęp do danych o wątku bez konieczności odczytywania któregokolwiek z komentarzy (np. odczytanie pierwszego komentarza wymaga sortowania, ponieważ funkcja modułu `os` języka *Python* o nazwie `listdir` listuje nieuporządkowaną zawartość katalogu przyjmowanego w argumencie.

Przykładowy plik `meta`

Poniżej przedstawiony został przykładowy plik z metadanymi wątku o nazwie „Przykładowy wątek”.

```
{
    "topic": "Przykładowy wątek",
    "timestamp": "1439395604000",
    "fileabout": "/katalog1/test.txt",
    "uid": "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH",
}
```

Rys. 4.4: Struktura przykładowego pliku z metadanymi wątku.

Według tego przykładowego pliku wątek został utworzony — zgodnie ze znacznikiem czasowym — 30. sierpnia 2015 r. o godzinie `9:23`. Autorem jest użytkownik o identyfikatorze `2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH`. Dyskusja o tytule „Przykładowy wątek” rozpoczęta przez

tego użytkownika dotyczy pliku `test.txt`, który znajduje się wewnątrz folderu `katalog1` wewnątrz folderu synchronizowanego.

Strukturę przykładowego synchronizowanego folderu — zawierający zarówno wątek z powyższego przykładu jak i plik opisywany w dyskusji — przedstawiono poniżej:

```
[root]/
    .sync/
        [ pliki aplikacji BitTorrent Sync ]
    .Users/
        [ pliki z danymi o użytkownikach ]
    .Comments/
        katalog1/
            1439395604000@#&$Przykładowy wątek/
                meta
                1439395604000@#&$2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH
        1439423576000@#&$Inny wątek/
            meta
            1439423576000@#&$2SAFDSBCXGFD765GDFS4GFDS35FDSBVB
            1439476598000@#&$2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH
    katalog1/
        test.txt
        ccc.txt
    aaa.txt
    bbb.txt
```

Widać wewnątrz poniższej struktury, że „Przykładowy wątek" został umieszczony w folderze `.Comments` w lokalizacji takiej samej, jak oryginalny plik, o którym powstała dyskusja (`katalog1`). Wewnątrz niego znajduje się plik z metadanymi o wcześniej przedstawionej strukturze oraz jeden komentarz. Warto zauważyć, że w nazwie komentarza jest ten sam znacznik czasowy, który jest zapisany w pliku `meta` — według zasad opisanych wcześniej.

W przykładzie umieszczono również drugi wątek, zlokalizowany najpłycej w katalogu `.Comments`. Oznacza to, że w interfejsie graficznym będzie on wylistowany zawsze, gdy użytkownik nawigując po folderze, będzie przebywał na najwyższym jego poziomie — „korzeniu". „Inny wątek" — bo tak należy odczytać tytuł tej dyskusji, zawiera dwa komentarze — jeden napisany przez użytkownika `2SAFDSBCXGFD765GDFS4GFDS35FDSBVB` w chwili pobrania znacznika czasowego `1439423576000` oraz przez użytkownika `2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH` w chwili `1439476598000`.

4.2.3 Nazwy folderów i plików

Foldery zawierające wątki oraz pliki z komentarzami są tworzone przez dowolny węzeł w dowolnym czasie, co zwiększa niebezpieczeństwo wystąpienia konfliktów w nazwach tworzonych plików za każdym razem, gdy użytkownik wypowiada się w dyskusji lub rozpoczyna nową. Dodatkowo drzewiasta struktura katalogu `.Comments`, wewnątrz którego przechowywane są komentarze, powoduje, że „zwykłe" foldery przechowywane są obok katalogów zawierających pliki komentarzy. Przeszukiwanie ich wszystkich przez serwer poprzez przeglądanie wszystkich lokalizacji jest wolniejsze i przy folderze mocno rozbudowanym systemem katalogów może odczuwalnie spowolnić aplikację.

Mając na uwadze powyższe problemy, projektując aplikację *TeamSync* przyjęto następujące dwie zasady — dotyczące nazw zarówno plików komentarzy, jak i folderów z wątkami — jako konieczne do spełnienia. Konsekwencje nieprzestrzegania każdej z zasad będą wyjaśnione w dalszej części sekcji.

1. System powinien uniemożliwiać (zapewniać poziom prawdopodobieństwa *w praktyce* uniemożliwiający) powtarzanie się *nazwy* pliku komentarza lub folderu wątku.
2. Nazwy powinny zawierać w sobie pewien element (ciąg znaków), dzięki któremu serwer odczyta informację, że wybrany folder lub plik to odpowiednio folder zawierający wątek oraz plik przechowujący wypowiedź. Informacja o wyjątkowości katalogu musi być zawarta w jego nazwie.

Opisy sposobów realizacji obydwóch zasad zostały opisane w poniższych podrozdziałach.

Unikalność nazwy

Jeśli w przypadku plików z komentarzami unikalność wprowadzanych nazw nie byłaby zachowana, efektem byłby konflikt skutkujący nadpisaniem pliku. Użytkownik `A` zatwierdzający treść odpowiedzi umieściłby swój plik, który chwilę później byłby nadpisany przez użytkownika `B`. Wersja użytkownika `A` zostałaby uznana jako nieaktualna i w konsekwencji przeniesiona do katalogu `Archive` w ukrytym folderze `.sync`, do którego użytkownicy nie mają dostępu z poziomu aplikacji *TeamSync*.

Podobnie — lecz z innymi konsekwencjami — wygląda sytuacja w przypadku tworzenia nowego wątku. Gdyby ten sam użytkownik `A` tworzył nowy wątek w tym samym momencie, co użytkownik `B`, również wystąpiłby konflikt, lecz z odmiennymi skutkami. W kodzie serwera zastosowano instrukcję warunkową przed tworzeniem nowego folderu wątku sprawdzającą, czy taki folder już istnieje. Aplikacja tworzy nowy katalog, jeśli jeszcze nie istniał wcześniej w systemie plików — w odwrotnym przypadku serwer kontynuuje działanie bez żadnych zmian. Efektem konfliktu byłoby nienadpisanie folderu z wątkiem, a nadpisanie pliku `meta` oraz (w zależności czy unikalność nazwy pliku z komentarzem została zachowana) możliwość nadpisania lub dodania nadmiarowego komentarza. W obecnym przypadku (użytkownicy `A` oraz `B` próbujący dodać nowy wątek w tym samym momencie) dodanie komentarza mogłoby skutkować następującą zawartością folderu z wątkiem:

– plik `meta` z treścią umieszczoną przez zwycięzcę konfliktu `A` vs. `B`,
– komentarz użytkownika `A`,
– komentarz użytkownika `B`.

Jak widać zapewnienie unikalności nazwy plików z komentarzami oraz folderów z wątkami w kontekście działania najważniejszej funkcjonalności jest niezbędne. Dlatego uzyskanie jednoznacznych nazw w przypadku plików z komentarzami odbywa się dzięki następującym składnikom:

– znacznik czasowy pobrany z serwera NTP podczas pisania nowego komentarza z dokładnością do jednej sekundy,
– identyfikator autora wypowiedzi (*uid*).

Dzięki kombinacji tych dwóch składników nie ma możliwości uzyskać w systemie identycznej pary nazw pliku z komentarzem. Żeby złamać zasadę i uzyskać taką parę, ten sam użytkownik musiałby wysłać dwa komentarze w ciągu jednej sekundy, co — ze względu na długość przetwarzania umieszczania odpowiedzi w systemie plików, oczekiwanie na odpowiedź ze znacznikiem

czasowym z serwera NTP i otrzymania odpowiedzi przez klienta w przeglądarce — jest bardzo mało prawdopodobne.

Nawet jeśliby tak się stało, druga wysłana przez użytkownika wypowiedź byłaby pusta, ponieważ nie miałby czasu na wypełnienie jej treścią — interfejs graficzny automatycznie po zatwierdzeniu komentarza usuwa całą treść z pola tekstowego i jest gotowy do umieszczania następnej wypowiedzi. W momencie odnotowania przez serwer faktu, że komentarz ma pustą treść, przekaże błąd użytkownikowi za pomocą komunikatu: „*Treść komentarza nie może być pusta*". Użytkownik musiałby bardzo szybko umieścić dowolną treść wewnątrz komentarza i ponownie zatwierdzić umieszczanie wypowiedzi przyciskiem „*Odpowiedz*". Jednak — jak zostało napisane powyżej — jest to bardzo mało prawdopodobne.

Przykładowa nazwa pliku z komentarzem mogłaby być następująca:

```
1439939641000<<separator>>2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH
```

Komentarz został wprowadzony do systemu w chwili o znaczniku czasowym równym `14399-39641` (`000` na końcu znacznika są dodane w wyniku jego konwersji z jednostek sekund na milisekundy) przez użytkownika o identyfikatorze `2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH`. Łańcuch znaków `<<separator>>` zostanie omówiony w późniejszej części podrozdziału.

Z tych samych powodów, dla których nazwy plików z komentarzami łączone są z dwóch powyżej opisanych składników (identyfikator użytkownika oraz znacznik czasowy), zdecydowano się otrzymywać w taki sposób również nazwy katalogów zawierających treść dyskusji. Dodatkowo system *TeamSync* uniemożliwia zatwierdzenie nowego wątku z pustym polem tekstowym zawierającym tytuł dyskusji — jest to dodatkowe „utrudnienie" umieszczania dwóch wątków w tej samej sekundzie.

Separator nazwy

Ciąg znaków zapisywany we wcześniejszych przykładach jako `separator` wchodzi w skład nazwy zarówno plików z komentarzami, jak i folderów z wątkami. Dzięki jego obecności w nazwie serwer przeszukując folder `.Comments` rozróżnia foldery pełniące funkcję przechowywania wątków od tych, które służą do grupowania wątków w strukturę będącą odwzorowaniem struktury folderu synchronizowanego.

Ponieważ opisywany łańcuch znaków (opisywany dalej jako *separator*) jest częścią nazwy pliku, musi składać sie ze znaków akceptowanych przez system operacyjny. Na przykład w systemach operacyjnych z rodziny *Unix*, w separatorze nie mógłby się znaleźć znak `/`, ze względu na pełnioną funkcję w systemie plików podczas określania lokalizacji.

Podstawowym problemem w implementacji związanym z nazwami plików i katalogów „specjalnych" (tych z wątkami oraz komentarzami) był fakt, że skoro użytkownicy mogą dodawać pliki o dowolnych nazwach, to mogą dodać też plik zawierający w nazwie separator. Wówczas serwer czytając listę wątków lub próbując przeczytać jeden z komentarzy, zwróciłby błąd. Ze względu na brak możliwości wprowadzenia takiego separatora, aby użytkownik nie mógł go — na drodze przypadku lub nie — umieścić w nazwie „zwykłego" pliku, zdecydowano się na zminimalizowanie prawdopodobieństwa wystąpienia podobnej sytuacji.

Separator — w obecnej implementacji — ma postać ciągu czterech znaków `@#&$`, a jego postać została wybrana ze względu na bardzo mało realną szansę zawarcia takiego ciągu znaków w nazwie pliku przez użytkowników. W ostatecznej postaci plik z komentarzem z poprzedniej sekcji zapisany w systemie plików użytkownika mógłby wyglądać następująco:

```
1439939641000@#&$2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH
```

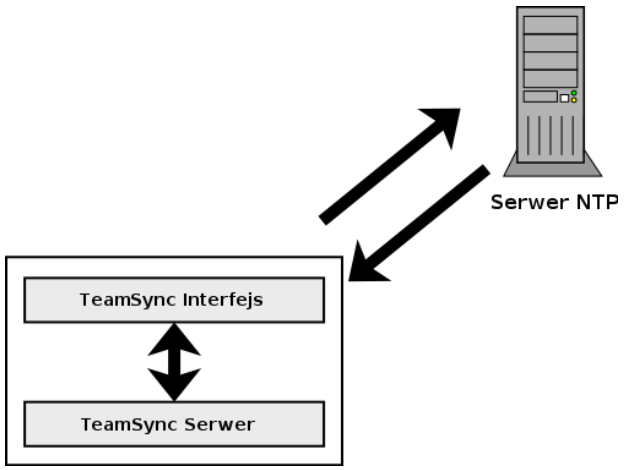
Jako dodatkowe zabezpieczenie można potraktować fakt, że serwer podczas przeszukiwania folderu `.Comments` w poszukiwaniu folderów z wątkami przyrównuje nazwę plików do wzorca wyrażenia regularnego:

```
[0-9]{13} @#&\$ [A-Z0-9]{32}
```

Aby serwer odczytał fałszywy folder z wątkiem (czyli ten wprowadzony przez użytkownika — niezawierający ani pliku `meta`, ani plików z komentarzami), użytkownik musiałby stworzyć folder (albo plik) o nazwie pasującej do powyższego wzorca. Podczas rzetelnego korzystania z aplikacji *TeamSync w praktyce* niemożliwe jest spowodowanie błędu.

Użytkownicy chcący zaburzyć pracę działania programu, są w stanie to osiągnąć, nazywając folder w podobny sposób (według wzorca zaprezentowanego wcześniej wyrażenia regularnego), umieszczając go w dowolnej lokalizacji wewnątrz katalogu synchronizowanego i tworząc w nim wątek. Wówczas — podczas tworzenia wątku — serwer umieści wewnątrz katalogu `.Comments` folder o nazwie spełniającej warunki folderu „specjalnego", a będącego nim — brak będzie plików z komentarzami oraz pliku `meta`. Zabezpieczenie aplikacji w takim wypadku polega na ignorowaniu folderów, które spełniają kryterium nazwy, lecz nie zawierają w sobie pliku z metadanymi wątku.

4.3 Serwer NTP



Rys. 4.5: Komunikacja systemu *TeamSync* z serwerem NTP.

Istnienie możliwości zamieszczania komentarzy przez użytkowników w dowolnej chwili działania systemu *TeamSync* oraz konieczność zachowania informacji o czasie wprowadzania wypowiedzi są źródłami kilku problemów. Chcąc umożliwić rzetelne — nie tylko co do treści, ale również co do czasu zamieszczenia komentarza — dzielenie się swoimi opiniami przez użytkowników, należało uniknąć w systemie dwóch nastepujących przeszkód:

– nie ma pewności, że na każdym z węzłów jest ten sam czas systemowy, z którego użytkownik pobierałby znacznik czasowy podczas umieszczania komentarza. Jeśli różnica między czasami zamieszczanych komentarzy byłaby większa niż kilka minut, wyświetlane wypowiedzi często byłyby przedstawiane w nieprawidłowej kolejności. Komentarze napisane później (według czasu rzeczywistego) — w wyniku różnic czasów systemowych na węzłach — mogłyby zostać umieszczone w sieci jako napisane wcześniej niż w rzeczywistości,

– nie ma gwarancji, że wszystkie węzły w jednolity sposób będą wprowadzać znaczniki czasowe do systemu — problem użytej przez nich jednostki oraz formatu.

Serwer NTP [14], od którego użytkownicy pobierają znaczniki czasowe podczas zamieszczania komentarzy, pełni rolę neutralnego arbitra i zarazem stanowi rozwiązanie dwóch powyższych zastrzeżeń. Otrzymywanie znaczników czasowych z tego samego dla wszystkich użytkowników, neutralnego źródła jest kluczowym punktem w algorytmie dodawania nowych wypowiedzi — w przypadku jego braku, użytkownicy mogliby wprowadzać do sieci fałszywe dane.

Potrzeba zachowania jednolitego formatu oraz jednostki znacznika czasowego wynika z konieczności zapewnienia unikalności nazw plików zawierających dane z komentarzami. Bez dostępu do serwera NTP użytkownik nie ma możliwości napisać wypowiedzi w żadnym z wątków, ponieważ narażałoby to system na generowanie konfliktów (np. poprzez dwukrotne uzyskanie tego samego znacznika czasowego przy dodaniu komentarza).

Zapytanie wysyłane jest do serwera NTP *zawsze*, gdy użytkownik umieści treść komentarza (lub nowego wątku) w interfejsie graficznym i zatwierdzi jego dodanie. W odpowiedzi serwer NTP odsyła wiele wartości takich jak: znacznik czasowy, przesunięcie, precyzja, wersja protokołu itd. Najważniejszym z nich (i jedynym użytym w systemie) jest sam znacznik czasowy.

Znacznikiem czasowym, który użytkownicy otrzymują w odpowiedzi jest skoordynowanym czasem uniwersalnym (*UTC — Universal Coordinated Time*) w postaci liczby całkowitej zawierającą sekundy.

Rozdział 5

Ogólne działanie systemu i konfiguracja

Jak zostało napisane w sekcji 4.2, użytkownicy nie wymieniają ze sobą informacji poprzez komunikaty, lecz poprzez umieszczanie odpowiednio zainicjowanych i umieszczonych plików. Aby lepiej zrozumieć sposoby zamieszczania komentarzy, opisano w poniższych sekcjach jaką strukturę ma synchronizowany katalog, jak system jest skonfigurowany oraz jak rekompensowane są wady spowodowane przez architekturę systemu.

5.1 Struktura katalogów

Podczas tworzenia synchronizowanego folderu użytkownik może dołączyć do istniejącego folderu (założonego przez innego użytkownika) lub zainicjować własny. W obydwóch przypadkach uwspólniany folder musi być pusty — wymaganie to jest powielone z aplikacji BitTorrent Sync.

Aby ułatwić i ujednoznacznić opis implementacji zastosowano następujące terminy i nazwy zmiennych:

właściciel — użytkownik, który stworzył folder i rozdysponował klucze (*secrets*) do pozostałych użytkowników,

[root] — ścieżka będąca bezwzględną lokalizacją synchronizowanego katalogu w systemie operacyjnym użytkownika.

Podczas inicjowania nowego (a więc również pustego) folderu przez właściciela tworzone są wewnątrz niego dodatkowe katalogi:

```
[root]/
    .sync/
    .Users/
    .Comments/
```

Katalog *.sync* tworzony jest przez aplikację BitTorrent Sync i domyślnie jest ukryty. Znajduje się w nim archiwum oraz pliki niezbędne do poprawnego działania synchronizacji folderu. Usunięcie katalogu *.sync* skutkuje pełnym zanikiem funkcjonalności synchronizacji i wystąpieniem błędu. Pełna struktura katalogu *.sync* wygląda następująco:

```
.sync/
    Archive/
        [ lista plików w archiwum ]
    ID
```

```
IgnoreList
StreamsList
```

- folder *Archive* zawiera wersje plików, które zostały usunięte/zmodyfikowane podczas synchronizacji,
- plik *ID* reprezentuje identyfikator folderu, dzięki któremu możliwe jest znalezienie innych użytkowników lub urządzeń uwspólniających ten sam folder przez aplikację BitTorrent Sync,
- plik *IgnoreList* jest plikiem tekstowym, wewnątrz którego zawarta jest lista plików pomijanych podczas synchronizacji (wspierane są znaki: „*" — oznaczający dowolny ciąg znaków oraz „?" — oznaczający jeden dowolny znak),
- plik *StreamsList* w odróżnieniu od *IgnoreList* zawiera listę plików (obsługującą rozszerzone atrybuty plików), które będą synchronizowane (tzw. „white list").

Pozostałe dwa foldery (*.Comments* oraz *.Users*) tworzone są przez aplikację zaimplementowaną w ramach niniejszej pracy i podobnie jak katalog *.sync* są katalogami ukrytymi. Ewentualne usunięcie któregoś z tych katalogów będzie skutkować niepoprawnym działaniem aplikacji: w przypadku zniknięcia folderu *.Comments* znikną wszystkie komentarze, a w przypadku *.Users* aplikacja nie będzie mogła rozpoznać użytkowników piszących komentarze i współdzielących folder, co będzie skutkować błędami.

Wszystkie usunięte pliki z synchronizowanego folderu są utrzymywane w archiwum wewnątrz tego folderu, więc możliwe jest „ręczne" przekopiowanie utraconych danych z powrotem w przypadku przypadkowego (lub celowego) ich usunięcia. Czas, jaki mają użytkownicy na przywrócenie usuniętych danych to 30 dni — po tym czasie dane sa usuwane bezpowrotnie Do ewentualnej rozbudowy aplikacji można dołączyć automatyczne odtwarzanie nie tylko całych usuniętych struktur *.Users* oraz *.Comments*, ale również usuniętych plików, do których odnosiły się niektóre wątki w systemie.

5.1.1 Katalog *.Users*

Podczas dołączania użytkownika do synchronizowanego folderu umieszcza on w katalogu *.Users* plik tekstowy o rozszerzeniu JSON, wewnątrz którego zapisuje słownik z danymi o sobie. Tworzony jest słownik o kluczach *uid* oraz *identity*, gdzie wartość *uid* to identyfikator użytkownika, a *identity* to nazwa użytkownika widziana przez innych użytkowników w aplikacji. Zainicjowany słownik zostaje zapisany do katalogu *.Users* pod nazwą `<UID użytkownika>.json`.

Taki sposób rozwiązania problemu zapisywania i wymiany danych o użytkownikach został wprowadzony ze względu na perspektywy rozwoju aplikacji. Utworzono fundamenty, które mogą być wygodnie i efektywnie rozbudowywane w kolejnych wersjach aplikacji.

Rozwiązaniem łatwiejszym byłby jeden plik z listą wszystkich użytkowników, w którym podczas dołączania do folderu kolejni użytkownicy wprowadzaliby dane na swój temat. Niestety wszelkie modyfikacje plików grożą możliwością wystąpienia konfliktu podczas równoczesnego wprowadzania zmian. Zrezygnowano więc z takiego rozwiązania na rzecz tworzenia nowych plików zamiast edytowania już istniejących.

Przykładowa struktura wewnątrz folderu *.Users*:

```
.Users/
    2HI7KMNQWPONLKFORRWXGOTIZSHBGFIH.json
    2SAFDSBCXGFD765GDFS4GFDS35FDSBVB.json
    3REW54GFDS6578FDSGDF5BSH652F2B34.json
```

Zawartość pliku `2HI7KMNQWPONLKFORRWXGOTIZSHBGFIH.json` pochodzącego z katalogu *.Users*:

```
{
    "uid": "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH",
    "identity": "Filip Rachwalak"
}
```

Rys. 5.1: Przykładowy plik użytkownika.

W przypadku takiej zawartości folderu *.Users* system TeamSync wyświetli w graficznym interfejsie trzech użytkowników. Biorąc pod uwagę, że zawartość pliku `2HI7KMNQWPONLKFORRWXGOTIZSHBGFIH.json` jest identyczna z przedstawioną powyżej, wyświetlaną nazwą jednego z użytkowników będzie „Filip Rachwalak".

5.1.2 Katalog *.Comments*

W przeciwieństwie do katalogu *.Users* — którego struktura jest płaska, a elementy tworzą listę — folder *.Comments* jest znacznie bardziej złożony. W miarę przybywania wątków katalog zwiększa swoją objętość zarówno wszerz jak i wgłąb, ponieważ wewnątrz niego odzwierciedlona jest drzewiasta struktura synchronizowanego katalogu. Dla łatwiejszego zrozumienia przyjmijmy następujący folder, który dwóch użytkowników współdzieli ze sobą:

```
[root]/
    .sync/
        [ pliki aplikacji BitTorrent Sync ]
    .Users/
        2HI7KMNQWPONLKFORRWXGOTIZSHBGFIH.json
        2SAFDSBCXGFD765GDFS4GFDS35FDSBVB.json
    .Comments/
    test1/
        test2/
            eee.txt
        ccc.txt
        ddd.txt
    aaa.txt
    bbb.txt
```

Jeśli użytkownik utworzyłby wątek dotyczący pliku *[root]/aaa.txt*, aplikacja umieściłaby folder z wątkiem na pierwszym poziomie katalogu *.Comments* zgodnie z lokalizacją pliku w głównym folderze. Natomiast jeśli użytkownik zacząłby wątek o plik *[root]/test1/ccc.txt*, program stworzyłby dodatkowy folder wewnątrz *.Comments* o nazwie *test1* i w nim umieściłby folder z nowym wątkiem.

Aby łatwiej było zrozumieć tworzenie nowych folderów z wątkami, załóżmy, że użytkownik utworzył trzy wątki o trzech plikach: *aaa.txt* w głównej lokalizacji, *ccc.txt* wewnątrz katalogu *test1*, oraz *eee.txt* wewnątrz katalogu *test2*. Wówczas struktura folderu *.Comments* wyglądałaby następująco:

```
.Comments/
    test1/
        test2/
            wątek_o_eee/
                [ pliki z komentarzami ]
        wątek_o_ccc/
            [ pliki z komentarzami ]
    wątek_o_aaa/
        [ pliki z komentarzami ]
```

Łatwo zauważyć, że w miarę przybywania wątków, struktura folderu rozrasta się i zbliża do struktury głównego folderu (oczywiście z pominięciem trzech ukrytych, dodatkowych katalogów oraz folderów przechowujących wątki). Przechowywanie komentarzy w ten sposób ma niestety swoją wadę — wrażliwość na przenoszenie danych w obrębie uwspólnionego katalogu. Wyeliminowanie tego ograniczenia mogłoby należeć do najbliższych ulepszeń aplikacji w przyszłości.

5.2 Pliki konfiguracyjne

Ustawienia aplikacji, identyfikator użytkownika, adres serwera BitTorrent Sync API — wszystkie niezbędne informacje do poprawnego działania systemu TeamSync, tworzonego w ramach niniejszej pracy magisterskiej, zawarte są wewnątrz plików konfiguracyjnych w formacie JSON (pliki `config.json` oraz `btsyncconfig.json`). Zlokalizowane są one wewnątrz głównego katalogu aplikacji — istnieje więc tylko jedna instancja każdego z plików i jest globalna w skali aplikacji (nie jest powielana w każdym synchronizowanym folderze).

Podczas uruchamiania aplikacji sprawdzana jest obecność tych plików. W przypadku ich obecności pobierane są z nich ustawienia i wykorzystywane w programie. Wszelka samodzielna modyfikacja tych plików np. poprzez edycję za pomocą dowolnego narzędzia do modyfikacji tekstu, może zakłócić działanie aplikacji ze względu na np. kodowanie znaków lub składnię formatu JSON. Jeśli aplikacja nie znajdzie któregoś z dwóch plików (np. podczas pierwszego uruchomienia programu), zainicjuje je i wypełni domyślnymi ustawieniami. Szczegóły przydzielanych ustawień oraz ich lista zostaną dokładnie omówione w następnych podrozdziałach.

5.2.1 Konfiguracja BitTorrent Sync

Pierwszym z dwóch plików z ustawieniami jest `btsyncconf.json` zawierający informacje potrzebne do komunikacji programu Team Sync z aplikacją BitTorrent Sync. Podczas uruchamiania zaimplementowanej aplikacji inicjowany jest również BitTorrent Sync z przełącznikiem `--config` wskazującym na plik z ustawieniami.

Aby możliwa była komunikacja z systemem BitTorrent Sync, konieczna jest znajomość adresu wraz z numerem portu, na którym API będzie nasłuchiwać i na który system TeamSync będzie wysyłał żądania. W pliku konfiguracyjnym znajdują się również pola niezbędne podczas autoryzacji żądania: `login` oraz `password`. Istnieje możliwość zablokowania domyślnego interfejsu wystawianego przez aplikację BitTorrent Sync, który nie jest potrzebny do poprawnego działania całości i został domyślnie zablokowany. Ostatnie dwa elementy obecne w pliku z ustawieniami to: `storage_path` oraz `api_key`.

Wartość pola `api_key` jest ciągiem znaków, który jest generowany przez twórców BitTorrent Sync i wydawany na prośbę deweloperów chcących testować API lub wytwarzać własne oprogramowanie z jego wykorzystaniem (z zastrzeżeniem o niekomercyjnym przeznaczeniu oprogramowania). Jeśli plik konfiguracyjny aplikacji BitTorrent Sync odczyta plik konfiguracyjny i nie będzie miał wypełnionego tego pola (lub będzie wypełnione niepoprawnym kluczem), API nie będzie odpowiadało na żądania.

`storage_path` jest ścieżką do katalogu ustawianą przez użytkownika, w którym będą przechowywane informacje potrzebne aplikacji BitTorrent Sync:

- pliki potrzebne do uruchomienia graficznego interfejsu aplikacji BitTorrent Sync,
- pliki tekstowe logów rejestrujące powstawanie nowych torrentów i nadawanie im znaczników czasowych,
- plik ustawień,
- plik historii,
- plik logów rejestrujący wszelkie zdarzenia dotyczące synchronizacji (w zależności od użytych jej sposobów),
- plik przechowujący identyfikator procesu aplikacji BitTorrent Sync, użyteczny podczas wyłączania programu TeamSync.

Poniżej umieszczono przykładowy plik konfiguracyjny `btsyncconfig.json`. Z wyjątkiem `api_key` oraz `storage_path` identyczny plik jest tworzony podczas pierwszego uruchomienia aplikacji TeamSync.

```
{
    "use_gui": false,
    "webui": {
        "api_key": "<<ciąg znaków wg. wyrażenia regularnego [A-Z0-9]{136}>>",
        "login": "team",
        "password": "sync",
        "listen": "127.0.0.1:8787"
    },
    "storage_path": "/home/user/teamsync/.btsync-files"
}
```

Rys. 5.2: Przykładowy plik konfiguracyjny `btsyncconfig.json`.

Interpretacja przykładowego pliku konfiguracyjnego jest następująca: BitTorrent Sync nie będzie miał dostępnego oryginalnego interfejsu graficznego, natomiast wystawione API będzie odbierało żądania na adresie `127.0.0.1` oraz porcie `8787`. Aby polecenia wewnątrz zapytań zostały wykonane, wewnątrz nich muszą zawierać się dane uwierzytelniające `team` jako *login* oraz `sync` jako *password*. Ścieżka zapisu plików potrzebnych dla aplikacji BitTorrent Sync to `/home/user/teamsync/.btsync-files`.

5.2.2 Konfiguracja TeamSync

Plik `config.json`, wewnątrz którego zapisana jest konfiguracja systemu *TeamSync*, zawiera zarówno informacje dotyczące lokalizacji aplikacji, adresu serwera NTP, zapisanych tożsamości użytkownika w różnych folderach, jak i danych ułatwiających korzystanie z programu np. zapamiętane ostatnie ustawienia sortowania wątków. Podobnie jak plik `btsyncconfig.json` jest on

wczytywany na początku działania programu lub tworzony jest nowy, z domyślnymi ustawieniami w przypadku jego nieobecności wewnątrz folderu przechowującego aplikację.

Poniżej opisane zostaną pola konfiguracji — wraz z wartościami, jakie mogą przyjmować — podzielone na grupy odpowiadające powierzonym im funkcjom.

Ustawienia ogólne

- *uid* — identyfikator użytkownika generowany przez BitTorrent API. W celu uzyskania identyfikatora podczas tworzenia konfiguracji wysyłane jest żądanie `get_secrets` (podrozdział 4.1.2, metoda *get_secrets*) zwracające trzy ciągi losowych znaków — jeden z wyników użyty jest jako wartość pola `uid`,
- *operating_system* — system operacyjny sprawdzany przez program przy tworzeniu konfiguracji podczas pierwszego uruchomienia,
- *btsync_server_address* — adres oraz port, na którym BitTorrent Sync API nasłuchuje żądań od aplikacji TeamSync. Podobnie jak w pliku konfiguracyjnym systemu BitTorrent Sync, ustawione domyślnie na `127.0.0.1:8787`,
- *identities* — wartość tego pola ma strukturę słownika, który w kluczach przechowuje `secret` folderu synchronizowanego przez użytkownika, a w wartości tzw. „tożsamość", czyli nazwa, którą użytkownik wpisuje służąca do identyfikacji użytkowników między sobą wewnątrz folderu.

NTP

Zawarte w tej grupie ustawienia są potrzebne do komunikacji z serwerem NTP, z którego korzysta aplikacja podczas umieszczania komentarzy.

- *ntp_server_name* — nazwa domenowa serwera NTP, z którego użytkownik chce korzystać. Domyślnie przyjmowana jest wartość `pl.pool.ntp.org` ze względu na lokalizację,
- *ntp_server_version* — wersja serwera NTP, domyślnie wersja 3 (wersja 4 w momencie pisania pracy jest w trakcie implementacji).

W przypadku dalszego rozwoju aplikacji możliwa będzie zmiana tych wartości przez użytkownika, co umożliwi wygodne korzystanie z aplikacji w innej strefie czasowej lub w nowszej wersji serwera NTP.

Ułatwienia dostępu

Grupa ustawień, której jedynym zadaniem jest przyspieszanie pracy serwera aplikacji, aby nie było konieczne każdorazowe sprawdzanie ścieżki katalogu, wewnątrz którego znajduje się aplikacja.

- *application_path* — bezwzględna ścieżka, w której znajduje się główna część aplikacji,
- *btsync_exe_file* — bezwzględna lokalizacja wykonywalnego pliku aplikacji BitTorrent Sync,
- *btsync_conf_file* — bezwzględna lokalizacja pliku z konfiguracją aplikacji BitTorrent Sync.

Zapisywanie powyższych ścieżek do pliku konfiguracyjnego wpływa również na przejrzystość kodu, ponieważ częste ich używanie w formie wymagającej każdorazowej konkatenacji odpowiednich łańcuchów znakowych z dodatkowymi lokalizacjami byłoby uciążliwe dla programisty. Znacznie łatwiej i przejrzyściej czytać powyższe dane z pliku i przechowywać je w pamięci programu.

Ustawienia interfejsu graficznego

Ustawienia z tej grupy stworzone zostały z myślą o przyszłym rozwoju aplikacji w sferze graficznego interfejsu użytkownika. Obecnie nie ma możliwości ich zmiany, użytkownik musi pozostać przy ustawieniach domyślnych.

- *comments_date_format* — format daty wpisu, który użytkownik będzie mógł zmieniać w oparciu o kilka objaśnionych reguł. Wartość jest ustawiona domyślnie na `HH:mm - d.MM.yyyy`, a przykładowy wynik takiego formatowania to `12:34 - 7.04.2015`,
- *threadsorting1* — pole, którego wartość zapamiętuje „ulubiony" typ sortowania wątków. Poniżej znajduje się lista możliwych wartości z ich objaśnieniem:
  - *name* — sortowanie według nazwy wątku,
  - *timestamp* — sortowanie według czasu rozpoczęcia wątku,
  - *numberofcomments* — sortowanie według ilości komentarzy w wątku,
  - *lastcomment* — sortowanie według najświeższej odpowiedzi.
- *threadsorting2* — kierunek sortowania, możliwe wartości to: *ascending* oraz *descending*, domyslnie *ascending*.

Wewnątrz tej grupy ustawień w ewentualnym przyszłym rozwoju aplikacji będą umieszczane dane związane np. z kolorem czcionki albo ramek, w których wyświetlane są komentarze.

Przykładowy plik konfiguracyjny

Poniżej przestawiony zostanie przykładowy plik konfiguracyjny wypełniony domyslnymi wartościami podczas pierwszego tworzenia. Aby urzeczywistnić przykład do słownika *identities* zostaną dodane przykładowe wpisy.

```
{
    "uid": "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH",
    "operating_system": "Linux",
    "btsync_server_address": "127.0.0.1:8787",
    "identities": {
        "AKO6XM5HJZJ2DFAG2NJ7T4PVAIA4U25XQ": "Filip Rachwalak",
        "AQXKJKAC6XNJKNLE2YW6EOMCCBHG2UBUL": "Jan Iksiński"
    },

    "ntp_server_name": "pl.pool.ntp.org",
    "ntp_server_version": 3,

    "application_path": "/home/user/teamsync",
    "btsync_exe_file": "/home/user/teamsync/btsync",
    "btsync_conf_file": "/home/user/teamsync/btsyncconfig.json",

    "commentsDateFormat": "HH:mm - d.MM.yyyy",
    "threadsorting1": "timestamp",
    "threadsorting2": "ascending",
}
```

Rys. 5.3: Przykładowy plik konfiguracyjny `config.json`.

Według powyższej konfiguracji użytkownik posiada identyfikator `2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH` i pracuje na systemie operacyjnym `Linux`. Adres BitTorrent Sync API, na który wysyłane

będą żądania to domyślny adres `127.0.0.1:8787`. Użytkownik ma dwa współdzielone foldery: pierwszy o łańcuchu *secret* `AKO6XM5HJZJ2DFAG2NJ7T4PVAIA4U25XQ`, wewnątrz którego inni identyfikują go pod nazwą `Filip Rachwalak` i drugi o łańcuchu *secret* `AQXKJKAC6XNJKNLE2YW6EOMCC-BHG2UBUL` gdzie tą nazwą jest `Jan Iksiński`.

Ustawienia serwera `NTP` są domyślne dla Polski i dla aktualnej wersji usługi. Aplikacja TeamSync została zainstalowana wewnątrz lokalizacji `/home/user/teamsync`, o czym świadczy właściwość `application_path`. Bezwzględne ścieżki pliku wykonywalnego oraz konfiguracyjnego aplikacji BitTorrent Sync to odpowiednio `/home/user/teamsync/btsync` oraz `/home/user/teamsync/-btsyncconfig.json`.

Ustawiony format czasu zamieszczenia komentarza wyświetlany w intefejse graficznym razem z nim to `HH:mm - d.MM.yyyy`. Oznacza to, że przykładowa godzina będzie wyświetlana w następujący sposób: `22:12`, a data: `7.11.2014`. Oddzielać je będzie myślnik, co — dla wymienionej wyżej przykładowej daty oraz godziny — da wyjściowy efekt: `22:12 - 7.11.2014`. Ustawienia dotyczące sortowania skutkować będą zmianą kolejności wątków podczas wyświetlania ich listy według czasu powstania wątku (`timestamp`) w porządku rosnącym (`ascending`).

5.3 Spójność komentarzy w wątku

Z uwagi na rozproszoną architekturę aplikacji TeamSync, należy zwrócić uwagę na możliwość logicznego zaburzenia przepływu dyskusji. Nie ma pewności, że wszyscy użytkownicy w każdej chwili będą mieli „aktualną" wersję rozmowy. Może zajść sytuacja, w której użytkownik `A` wypowie się na pewien temat widząc tylko jeden komentarz w wątku (początkowy), podczas gdy w rzeczywistości już inni użytkownicy się wypowiedzieli, lecz użytkownik `A` nie zdążył uspójnić z nimi nowych danych. Wówczas może zdarzyć sie sytuacja, w której użytkownicy czytający wypowiedź użytkownika `A` — po ostatecznym, całkowitym uspójnieniu danych — będą się zastanawiać nad jej logicznym sensem.

Dla lepszego opisania problemu w dalszej części sekcji rozważana będzie następująca sytuacja: w systemie istnieje wątek dotyczący pewnego zdjęcia z jednym komentarzem o treści: „*Zdjęcie zrobione w Afryce podczas zachodu słońca u stóp Kilimandżaro*" (wszyscy użytkownicy w systemie posiadają swoją lokalną kopię tego komentarza). Dodatkowo istnieje trzech użytkowników: `A`, `B` oraz `C`, którzy będą wypowiadać się na temat tego pliku. Jednak działają wewnątrz systemu rozproszonego, więc można założyć, że częsta będzie sytuacja, w której nieobecny będzie jeden lub kilku z węzłów współdzielących folder. Niech użytkownik `C` będzie nieobecny w systemie (do tej pory wszyscy użytkownicy — łącznie z użytkownikiem `C` — mają poprawnie zsynchronizowany wątek o zdjęciu — mogą odczytać pierwszy komentarz). Wówczas — podczas nieobecności węzła `C` — tak mogłaby wyglądać krótka konwersacja użytkowników `A` oraz `B`:

Autor	**Treść komentarza**	**Czas dodania**	**Odczytane przez `C`**
`A`	To zdjęcie bardzo mi się podoba!	20:35:00	—
`B`	Faktycznie, ładne zdjęcie	20:38:00	—

Komentarze przechowywane są w plikach, w których zawarta jest treść (odpowiadająca kolumnie **Treść komentarza**), autor (kolumna **Autor**) oraz znacznik czasowy (kolumnie **Czas dodania**). Kolumna **Odczytane przez `C`** jest odzwierciedleniem zmiennej *readby* dla użytkownika `C` wewnątrz komentarza (dokładny opis mechanizmu *readby* znajduje się w sekcji 4.2.1). Widać, że użytkownicy napisali po jednym komentarzu o godzinie 20:35 oraz 20:38. Wartości w ostatniej kolumnie — przechowującej czas przeczytania komentarzy przez użytkownika `C` — nie

są wprowadzone, ponieważ użytkownik `C` nie zsynchronizował jeszcze folderu z komentarzami z użytkownikami `A` oraz `B`, co oznacza, że nie przeczytał ich wypowiedzi.

Następnie obydwaj użytkownicy rozłączają się z sieci. Po chwili niepozwalającej na ewentualną komunikację z obydwoma użytkownikami, (np. o godzinie 21:20) użytkownik `C` uruchamia aplikację TeamSync i umieszcza swój komentarz: „*Nie wiem jak Wam, ale mi się podoba!*" (według jego najlepszej wiedzy jest pierwszym, który odpowiada na wypowiedź inicjującą wątek). Po jakimś czasie (około godziny 22:00) do systemu podłączają się dwaj pozostali użytkownicy i wszyscy trzej uspójniają wszystkie dane. Okazuje się, że kolejność umieszczenia wypowiedzi wewnątrz wątku jest następująca:

Autor	**Treść komentarza**	**Czas dodania**	**Odczytane przez `C`**
`A`	To zdjęcie bardzo mi się podoba!	20:35:00	22:00:14
`B`	Faktycznie, ładne zdjęcie	20:38:00	22:00:15
`C`	Nie wiem jak Wam, ale mi się podoba!	21:20:00	21:20:13

Z punktu widzenia użytkowników `A` i `B`, logika wypowiedzi użytkownika `C` jest mocno zaburzona. Jest to konsekwencja rozproszonej architektury systemu (systemu BitTorrent Sync odpowiedzialnego za transport danych), której *nie ma możliwości* zniwelować bez implementacji dodatkowych algorytmów rozproszonych, lub włączania w architekturę nieustannie dostępnego węzła, synchronizującego wszystkie dane na bieżąco w czasie rzeczywistym. Wówczas można już mówić o pewnej centralizacji systemu, czego w ramach niniejszej pracy chciano uniknąć.

W systemie TeamSync zaimplementowano mechanizm, dzięki któremu można „usprawiedliwiać" użytkowników, których logika i sens komentarzy bardzo lub tylko nieznacznie odbiegają od norm, lub którzy powielają wypisane wcześniej kwestie. Problem rozwiązano poprzez możliwość ustawienia takiego sposobu szeregowania dyskusji, aby każdy z użytkowników mógł przeczytać ją w takiej kolejności, w jakiej przeczytał ją autor wątpliwego komentarza. Kontynuując przykład ze zdjęciem z Afryki, jeśli użytkownik `A` chciałby odczytać dyskusję tak, jak odczytał ją użytkownik `C` musiałby zmienić w interfejsie graficznym spójność na „spójność według użytkownika `C`" (dokładny opis funkcjonalności znajduje się w dodatku, w sekcji A.1). Wówczas zobaczyłby następującą kolejność komentarzy:

Autor	**Treść komentarza**	**Czas dodania**	**Odczytane przez `C`**
`C`	Nie wiem jak Wam, ale mi się podoba!	21:20:00	21:20:13
`A`	To zdjęcie bardzo mi się podoba!	20:35:00	22:00:14
`B`	Faktycznie, ładne zdjęcie	20:38:00	22:00:15

Po obejrzeniu dyskusji w powyższej kolejności użytkownicy mogą w większym stopniu zrozumieć się nawzajem i uniknąć nieporozumień wynikających nie z ich błędu, lecz z założeń architektonicznych systemu. Co prawda funkcjonalność zmiany punktu odniesienia w czytaniu wątków w pewnym stopniu rekompensuje wady architektury, natomiast nie niweluje ich całkowicie, z czym jednak użytkownicy systemu rozproszonego powinni się liczyć.

Rozdział 6

Implementacja

6.1 Zastosowane technologie

Wybór technologii użytych w implementacji został podyktowany jej architekturą — interfejs użytkownika dostępny jest tylko i wyłącznie w przeglądarce internetowej, z tego powodu wybrane przez autora i opisane poniżej narzędzia są kojarzone jedynie w obszarze technologii internetowych. Interfejs graficzny wywarł niemały wpływ na wybór narzędzi, z uwagi na fakt, że osoba korzystająca z aplikacji powinna w łatwy i intuicyjny sposób:

- poruszać się po katalogach, plikach oraz wątkach,
- wyszukiwać wątki, komentarze lub pojedyncze słowa wewnątrz nich,
- wydzielić te wątki, które najbardziej interesują użytkownika,
- przeglądać statystyki dotyczące wpisywanych komentarzy.

Narzędzie, które zostało wybrane do realizacji tych zadań to AngularJS [17] — technologia ułatwiająca kontrolę nad elementami interfejsu, stworzona w języku JavaScript. Jednakże, aby cały system prawidłowo i sprawnie działał, potrzebna była również technologia, która umożliwia przekazanie do interfejsu danych w odpowiedniej formie, pobranie z niego zadań wyznaczonych przez użytkownika i odpowiednie ich zrealizowanie. Do tych celów użyto języka Python [18] wraz z narzędziem o nazwie Django [19] do implementacji aplikacji internetowych. Połączenie Python/Django oraz AngularJS okazało się wystarczające do stworzenia aplikacji — poza nimi użyte zostały wyłącznie skrypty odpowiadające za wygląd interfejsu.

AngularJS

AngularJS jest otwartą biblioteką języka JavaScript, która została stworzona w 2009 roku i obecnie jest wspierana przez firmę Google. Najważniejszą funkcją narzędzia jest dwukierunkowe wiązanie danych (tzw. „emphtwo-way data binding"), które wyjątkowo łatwo obsługuje się wewnątrz szablonów HTML — za ich pomocą zmiany wewnątrz modelu są natychmiast odzwierciedlone w interfejsie i na odwrót: elementy wprowadzone bądź zmienione przez użytkownika natychmiastowo znajdują odzwierciedlenie w modelu.

Biblioteka korzysta z wzorca MVC (Model-View Controller), aby ułatwić nie tylko implementację, ale również testowanie tworzonego systemu. AngularJS umożliwia bardzo łatwe zarządzanie dynamiczną treścią poprzez wspomniane wcześniej szablony i dodatkowe komendy wewnątrz szablonów. Dynamika aplikacji zaimplementowanej w niniejszej pracy była głównym wyzwaniem podczas jej tworzenia, a biblioteka AngularJS znacząco wpłynęła zarówno na poprawność, jak i objętość kodu.

Django

O ile AngularJS został wykorzystany w znacznej mierze, o tyle stopień wykorzystania funkcjonalności Django został zredukowany do minimum. Biblioteka Django jest biblioteką do kompleksowego budowania stron internetowych w dowolnych architekturach (np. REST [20]), stworzoną, by jak najbardziej automatyzować kod i jak najszybciej — a zarazem jak najprościej — tworzyć skomplikowane serwisy. Narzędzie użyte zostało do implementacji z uwagi na prostotę i możliwości testowania, debugowania oraz możliwości rozbudowania zaimplementowanej aplikacji o dodatkowe rozwiązania.

Biblioteka Django powstała w 2003 roku, stworzona została przez programistów związanych ze środowiskiem dziennikarskim, dzięki czemu przystosowana jest do szybkiej i nieskomplikowanej pracy. Podobnie jak AngularJS pozwala (poprzez mechanizm szablonów) na łatwe i intuicyjne umieszczanie dynamicznych elementów na stronie. Wybrana została głównie ze względu na swoją prostotę oraz dodatkowe funkcjonalności ułatwiające pracę nad aplikacją np. możliwość automatycznego ponownego uruchamiania serwera podczas wprowadzania w nim zmian (co jest robione automatycznie).

6.2 Komunikacja z serwerem

Komunikacja między klientem a serwerem odbywa się poprzez zapytania `HTTP`, wewnątrz których przesyłane są dane potrzebne zarówno stronie klienta do ich wyświetlenia, jak i stronie serwera do wprowadzenia ich do systemu. Poniżej dokładnie opisane zostaną operacje wykonywane przez serwer, podzielone na grupy ze zbliżonym obszarem funkcjonalności.

6.2.1 Foldery

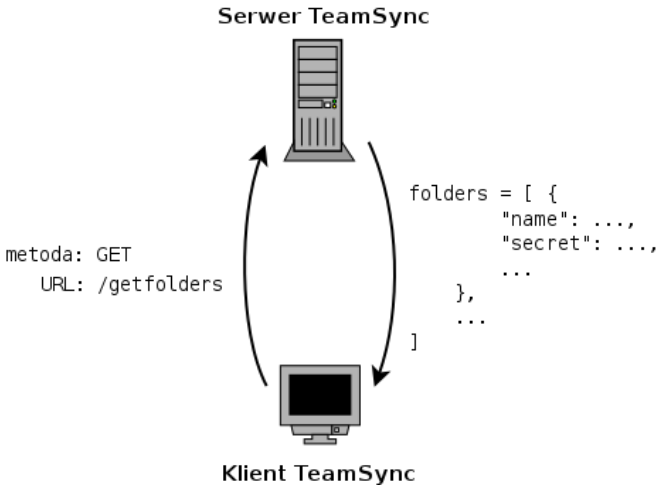
Operacje dostępne dla folderów w aplikacji TeamSync zostały ograniczone do dodawania ich oraz usuwania. Dodatkowe okna z ustawieniami, edycja folderów zostały przewidziane jako ewentualną dalszą rozbudowę systemu. W pracy skupiono się na pełnowartościowym korzystaniu z podstawowej funkcjonalności (synchronizacja danych oraz wprowadzanie komentarzy), co zostało spełnione bez konieczności implementacji edycji folderów.

Pobieranie listy folderów

Aby umożliwić użytkownikowi wybór folderu z listy synchronizowanych katalogów, aplikacja przeglądarkowa wysyła do serwera zapytanie `HTTP` za pomocą metody `GET` na adres `http://<adres oraz port serwera TeamSync>/getfolders`. Odpowiedź z serwera umieszczana jest wewnątrz listy `folders`, która przechowuje informacje dotyczące katalogów zapisane w formacie `JSON`.

W odpowiedzi serwera znajdują się wszystkie informacje dotyczące folderów otrzymane z BitTorrent Sync API (w sekcji 4.1.2 znajduje się dokładne wyjaśnienie zwracanych danych przez metodę `get_folders`) z dodatkowymi danymi:

- *name* — ułatwia interfejsowi wyświetlanie nazwy katalogu w liście,
- *identity* oraz *uid* — tożsamość i identyfikator użytkownika,
- *users* — lista użytkowników zawierająca ich identyfikatory oraz tożsamości, potrzebna do wyświetlania wszystkich użytkowników w folderze oraz autorów komentarzy. Wartości wewnątrz tej listy są pobrany z katalogu `.Users` wewnątrz folderu współdzielonego.



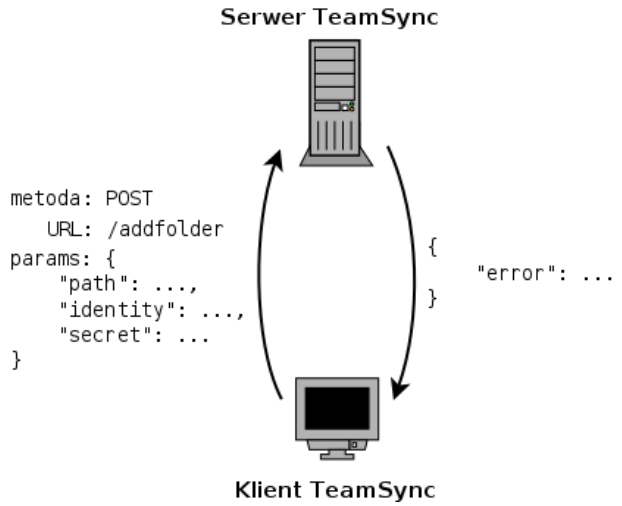
Rys. 6.1: Komunikacja w systemie *TeamSync* podczas wywoływania metody `getfolders` — pobieranie listy folderów.

```
[
    {
        "name": "testowy katalog",
        "dir": "/home/user/testowy katalog",

      /* wartości mniej istotne, pobrane podczas
         wykonywania metody get_folders takie jak: size,
         down_speed, up_speed, error, indexing, id, type */

        "secret": "A3LL43HJ257YCKMOLAD7QSEAS7U373BVO",
        "identity": "Filip Rachwalak",
        "uid": "FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM",
        "users": [
            {
                "uid": "PWGGJTDPC35O43SLEKFPBPIG3NYV7PH7",
                "identity": "Jan Iksiński"
            },
            {
                "uid": "FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM",
                "identity": "Filip Rachwalak (Ty)"
            }
        ]
    }
]
```

Rys. 6.2: Przykładowa odpowiedź serwera na żądanie pobrania listy synchronizowanych folderów.

W powyższym przykładzie użytkownik o identyfikatorze `FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM` i tożsamości `Filip Rachwalak` posiada tylko jeden współdzielony folder, który dzieli z użytkownikiem `Jan Iksiński` o identyfikatorze `PWGGJTDPC35O43SLEKFPBPIG3NYV7PH7`. Bezpośrednia ścieżka katalogu to `/home/user/testowy katalog`, natomiast nazwa, jaka będzie wyświetlana na liście w przeglądarce to `testowy katalog`. *Secret* folderu to `A3LL43HJ257YCKMOLAD7QSEAS7U373BVO`.

Tworzenie folderu



Rys. 6.3: Komunikacja w systemie *TeamSync* podczas wywoływania metody `addfolder` — tworzenie folderu.

Użytkownik ma możliwość wyboru pomiędzy rodzajem dodawanego katalogu — inicjacja własnego lub dołączenie do już istniejącego, stworzonego przez innego użytkownika folderu. Podczas obydwóch tych operacji przeglądarkowy interfejs graficzny wysyła do serwera zapytanie metodą `POST` na adres `http://<adres oraz port serwera TeamSync>/addfolder`. W parametrach żądania znajdują się:

- *path* — bezwzględna ścieżka do folderu, którego zawartość od tej pory będzie synchronizowana z pozostałymi użytkownikami,
- *identity* — zadeklarowana przez użytkownika tożsamość, czyli nazwa użytkownika, jaka będzie wyświetlana innym węzłom,
- *secret* — za jego pomocą BitTorrent Sync odnajdzie w sieci pozostałych użytkowników; jeśli ten parametr pozostanie pusty, serwer stworzy nowy folder (nie w fizycznym sensie, lecz w logicznym — doda podany w parametrze *path* folder do folderów współdzielonych).

Serwer otrzymując żądanie z powyższymi parametrami, przekazuje je poprzez zapytanie `HTTP` do BitTorrent Sync API, za pomocą funkcji `add_folder` (dokładny opis metody znajduje się w sekcji 4.1.2). Jeśli podstawowa walidacja wprowadzanych danych się nie powiedzie, lub jeśli BitTorrent Sync API zwróci błąd, serwer oprócz zwrócenia odpowiedzi z treścią błędu nie wykona żadnych dodatkowych instrukcji.

W przypadku powodzenia serwer musi wykonać następujące czynności:

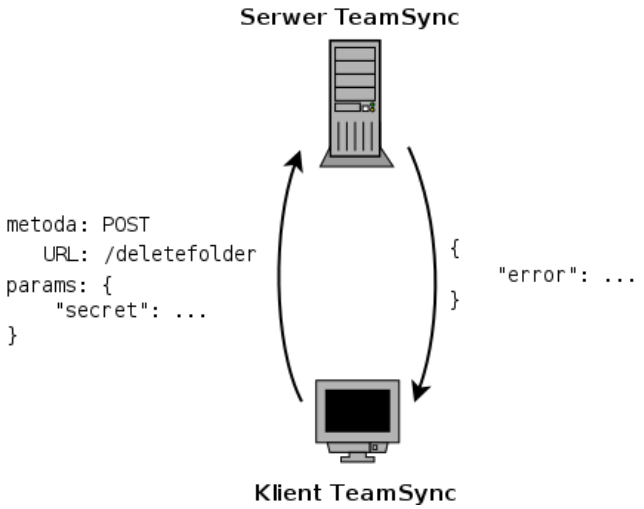
Utworzenie folderów `.Comments` oraz `.Users` — jeśli użytkownik nie dołącza do istniejącego folderu, tylko inicjuje swój katalog, musi stworzyć lokalizacje, do których pozostali użytkownicy będą mogli dopisywać pliki ze swoimi danymi (`.Users`) oraz komentarze (`.Comments`).

Uaktualnienie folderu `.Users` — aby inni użytkownicy mogli zobaczyć nowy węzeł w swoim folderze, użytkownik dołączający do katalogu musi umieścić w folderze `.Users` swoje dane — identyfikator oraz tożsamość.

Uaktualnienie pliku konfiguracyjnego — do słownika `identities` — przechowującego wszystkie tożsamości użytkownika — w pliku konfiguracyjnym (`config.json`), dodawana jest nowa tożsamość.

Po wykonaniu powyższych czynności serwer zwraca komunikat do aplikacji klienckiej o powodzeniu operacji, a w przeglądarce odświeżana jest lista folderów.

Usuwanie folderu



Rys. 6.4: Komunikacja w systemie *TeamSync* podczas wywoływania metody `addfolder` — tworzenie folderu.

Usuwanie katalogu odbywa się poprzez wysłanie do serwera żądania `HTTP` za pomocą metody `POST` na adres `http://<adres oraz port serwera TeamSync>/deletefolder`. Parametrem przesyłanym wewnątrz zapytania jest *secret* folderu. Serwer nie robi nic poza przesłaniem *secreta* za pomocą funkcji `remove_folder` protokołu BitTorrent Sync API (dokładny opis funkcji `remove_folder` w sekcji 4.1.2) i przekazuje odpowiedź — o pomyślnym lub niepomyślnym usunięciu — do klienta.

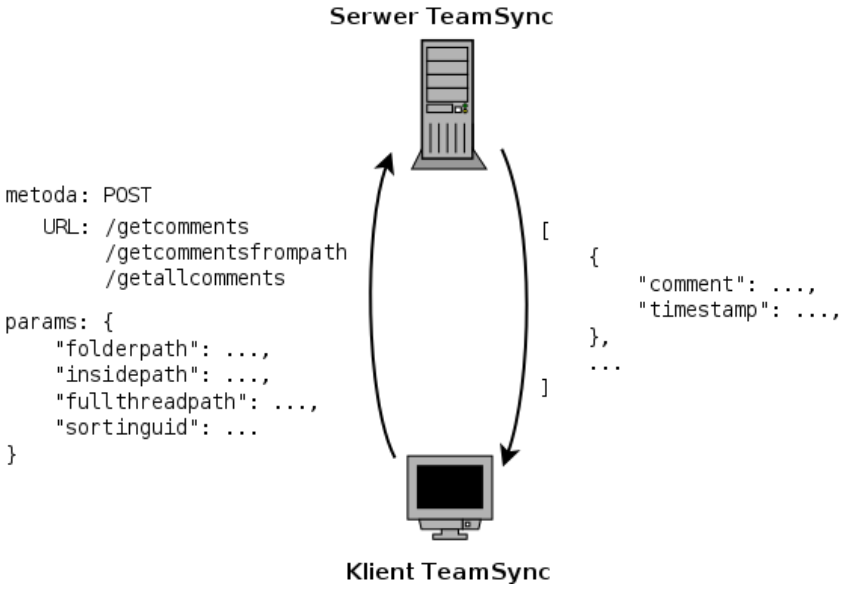
W przypadku niepowodzenia rola serwera kończy się na przekazaniu wyniku. Natomiast jeśli usunięcie folderu z listy synchronizowanych katalogów przebiegnie pomyślnie, serwer musi zmodyfikować plik konfiguracyjny `config.json` w celu usunięcia ze zmiennej `identities` tożsamości, której użytkownik już nie będzie potrzebował, ponieważ folder stanowiący jej środowisko przestał istnieć.

6.2.2 Komentarze

Aplikacja TeamSync nie umożliwia usuwania komentarzy przez użytkowników nawet w przypadku gdy osobą, która chciałaby usunąć wypowiedź z systemu, jest jej autor. Usunięcie komentarza można zastąpić poprzez zmodyfikowanie lub całkowite usunięcie jego treści, natomiast obecność wypowiedzi w systemie będzie zawsze widoczna.

Czytanie komentarzy

Główną czynnością aplikacji TeamSync podczas wyświetlania komentarzy jest pobranie ich z serwera i przeniesienie odpowiedzi serwera do listy `comments`, która jest główną strukturą wyświetlaną w sekcji komentarzy. Zawiera ona komentarze w takiej strukturze, w jakiej są one przeczytane przez serwer z plików wewnątrz folderu z wątkiem.



Rys. 6.5: Komunikacja w systemie *TeamSync* podczas wywoływania metod `getcomments`, `getallcomments` oraz `getcommentsfrompath` — czytanie komentarzy.

Pobieranie listy komentarzy inicjowane jest z kodu `javascript` przeglądarki, a użyte do tego celu zapytania różnią się w zależności od widoku (aby zapoznać się dokładniej z widokami, należy przeczytać dodatek A.2), w którym użytkownik aktualnie się znajduje:

W widoku nowego wątku nie są pobierane żadne komentarze. W momencie wprowadzenia danych i zatwierdzenia nowego wątku system automatycznie przechodzi do widoku aktualnego wątku.

W widoku aktualnego wątku odświeżanie listy komentarzy odbywa się za pomocą funkcji `refreshComments` wywoływanej np. podczas przechodzenia do nowego wątku albo wprowadzenia nowego komentarza. Jej zadaniem jest wysłanie na URL serwera `http://<adres oraz port serwera TeamSync>/getcomments` zapytania metodą `POST`, wewnątrz którego znajdują się następujące dane:

- *fullthreadpath* — pełna ścieżka lokalizacji wątku, niezbędna do pobrania listy komentarzy,
- *sortinguid* — identyfikator użytkownika, według którego mają zostać posortowane komentarze. Jeśli wartość ta pozostanie pusta (domyślnie), serwer posortuje komentarze według ich znaczników czasowych pobranych w momencie umieszczenia ich w systemie TeamSync. W przeciwnym wypadku (*sortinguid* wskazuje któregoś z użytkowników), wypowiedzi zostaną posortowane w takiej kolejności, w jakiej zostały odczytane przez wskazanego w argumencie *sortinguid* użytkownika. Serwer wówczas — zanim zwróci w odpowiedzi listę komentarzy — posortuje dane według struktury `readby`, wewnątrz której umieszczone są znaczniki czasowe momentu odczytania wiadomości przez wskazanego użytkownika (sortowanie komentarzy według różnych spójności zostało szczegółowo opisane w sekcji 5.3).

Jak zostało opisane na początku sekcji — serwer odpowiada listą komentarzy odczytanych z plików i w zależności od zawartości argumentu *sortinguid* sortuje wyjściowe dane przed ich wysłaniem do klienta.

W widoku bieżącej lokalizacji do zmiennej `comments` zostaną przypisane komentarze z odpowiedzi serwera na zapytanie wysłane przez funkcję `getCommentsFromPath` uruchamianą w kodzie `javascript` przeglądarki klienta. Funkcja ta wysyła żądanie metodą `POST` na adres URL serwera `http://<adres oraz port serwera TeamSync>/getcommentsfrompath` z danymi w formacie JSON:

- *folderpath* — bezwzględna ścieżka synchronizowanego folderu.
- *insidepath* — lokalizacja wewnątrz folderu, z której poziomu będą pobierane wszystkie komentarze. Nie będą brane pod uwagę komentarze w wątkach umieszczonych wewnątrz poziomów, wgłąb lokalizacji.

Konieczność wysłania dwóch ścieżek zamiast jednej podyktowana jest faktem, iż w przypadku pobierania komentarzy z jednego wątku zmienna *fullthreadpath* zawarta jest — obok innych informacji, np. nazwie, znaczniku czasowym powstania itd. — w obiekcie wyświetlanym przez graficzny interfejs. Podczas pobierania komentarzy z jednego wątku wystarczy przekazać tą zmienną serwerowi. Natomiast w obecnym wypadku pobierania komentarzy z całej lokalizacji, nie ma jednego konkretnego wątku, z którego można by pozyskać tą informację, więc zostało przyjęte rozwiązanie, w którym klient wysyła dwie ścieżki, a serwer łączy je w całość (łącznie z katalogiem `.Comments` ścieżki łączone są na wzór `<folderpath>/.Comments/<insidepath>`) i dopiero z tak uzyskanej ścieżki pobierane są komentarze.

W widoku wszystkich komentarzy w kodzie skryptu interfejsu uruchamiana jest nie przyjmująca żadnego argumentu funkcja `getAllComments`. Jej zadaniem jest wysłanie metodą `POST` na adres URL serwera `http://<adres oraz port serwera TeamSync>/getallcomments` bezwzględnej ścieżki synchronizowanego folderu. Serwer po jej odebraniu pobierze wszystkie komentarze napisane wewnątrz katalogu niezależnie od poziomu zagłębienia wewnątrz niego.

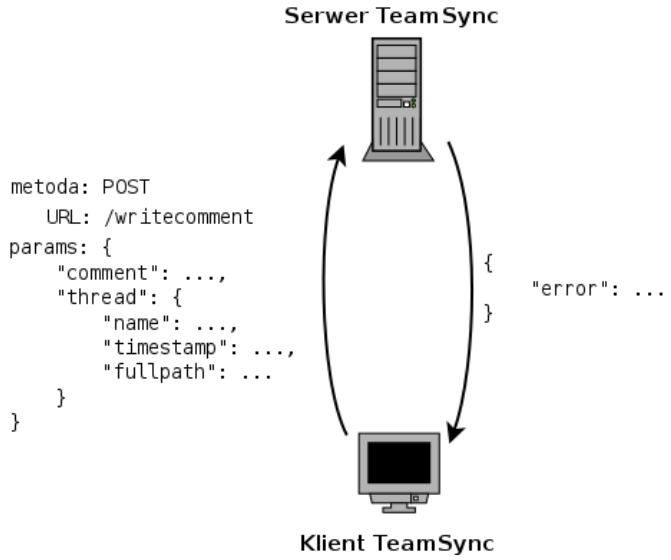
W widoku wszystkich komentarzy użytkownika procedura uzyskiwania od serwera wyświetlanych postów wygląda identycznie jak w przypadku widoku wszystkich komentarzy. Jedyną różnicą w sposobie ich wyświetlania jest filtrowanie ich po stronie klienta według *uid* autora wypowiedzi. Użytkownik wybierając autora, którego komentarze chciałby wyświetlić, w rzeczywistości pobiera wszystkie wypowiedzi z folderu i dopiero po ich uzyskaniu nakładany jest filtr. Dzięki takiemu podejściu możliwe jest szybsze oglądanie komentarzy różnych użytkowników, ponieważ pobierane one są raz. Zmieniając autora, zmieniane jest tylko filtrowanie wypowiedzi.

Pisanie komentarzy

Użytkownik przeglądając komentarze, może odpowiadać na nie tylko w przypadku, gdy przegląda je w widoku wątku (aby zapoznać się dokładniej z widokami, należy przeczytać dodatek A.2). Nie ma możliwości odpisywania odpowiedzi na komentarz, nie znając kontekstu całego wątku, gdyż może to doprowadzić do nieporozumień między użytkownikami. Dlatego też możliwość komentowania we wszystkich widokach oprócz widoku wątku została zablokowana.

W widoku wątku — na końcu konwersacji po obecnie ostatnim poście — znajduje się pole tekstowe, gdzie użytkownik może wpisać treść swojej odpowiedzi, a następnie kliknąć przycisk, który metodą `POST` wysyła na adres `http://<adres oraz port serwera TeamSync>/writecomment` żądanie zawierające obiekt `JSON` z wartościami o następujących kluczach:

- *comment* — treść komentarza wpisana przez użytkownika,



Rys. 6.6: Komunikacja w systemie *TeamSync* podczas wywoływania metody `writecomment` — pisanie komentarzy.

– *thread* — obiekt JSON zawierający dane dotyczące aktywnego wątku (tego, w którym odpowiada użytkownik), które zawiera nastepujące informacje:

  – *aaa* — dsadsadasda
  – *dsadasdsa* — dasczxczxcxz

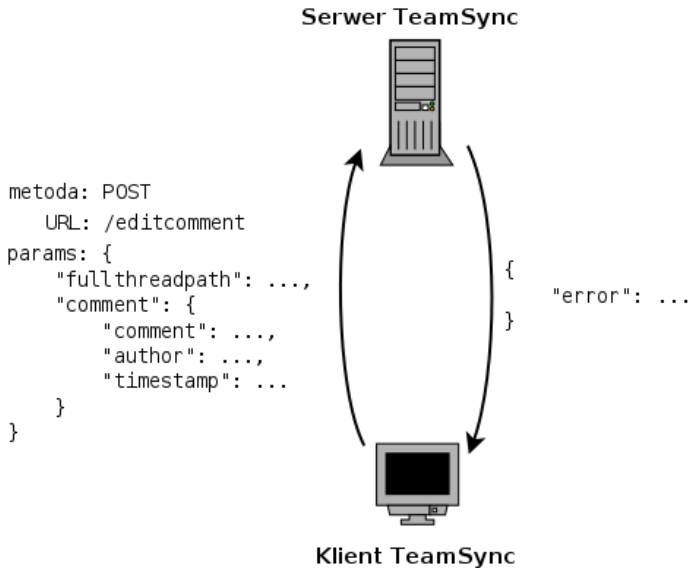
```
{
    "comment": "Treść nowego komentarza",
    "thread": {
        "numberofcomments": 7,
        "name": "Testowy tytul",
        "timestamp": "1438513896000",
        "lastcomment": "1439395572000",
        "path": "/",
        "fullpath": "/home/user/aaa/.Comments/1438513896000@#&$Testowy tytul",
        "type": "thread",
        "unreadcomment": false
    }
}
```

Rys. 6.7: Przykładowe żądanie wysyłane do serwera podczas odpowiedzi na wątek.

Po otrzymaniu od klienta żądania z tymi danymi serwer za pomocą ścieżki `fullpath` wewnątrz słownika `thread` pobierze z serwera NTP znacznik czasowy (aby uwiarygodnić przykład, założono, że pobrany znacznik czasowy to `1439396297000`) i umieści w podanej lokalizacji (`/home/user/aaa/.Comments/1438513896000#&$Testowy tytul`) nowy plik z komentarzem o nazwie `1439396297000#&$2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH`:

```
{
    "comment": "Treść nowego komentarza",
    "timestamp": "1439396297000",
    "history": [
        {
            "comment": "Treść nowego komentarza",
            "timestamp": "1439396297000"
        }
    ],
    "uid": "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH",
    "readby": {
        "2HI7KRUNSSONIUJKMRWXGOTIZSHBGFIH": "1439395604000"
    }
}
```

Rys. 6.8: Plik komentarza zapisany w wyniku otrzymania przez serwer wyżej zaprezentowanego żądania.



Rys. 6.9: Komunikacja w systemie *TeamSync* podczas wywoływania metody `editcomment` — edycja komentarzy.

Edycja komentarzy

Modyfikacja komentarza może być dokonana tylko przez jego autora. Dzięki zastosowaniu takiego podejścia obniżone zostało ryzyko wystąpienia konfliktu podczas synchronizacji, ponieważ liczba osób mogących edytować plik komentarza została zredukowana do jednej.

Po zatwierdzeniu zmian do serwera wysyłane są dane niezbędne do odnalezienia komentarza w systemie plików i dopisania następnego wpisu do słownika `history` znajdującego się wewnątrz pliku z wypowiedzią wypowiedzi. Informacje wysyłane są za pomocą metody `POST` na adres `http://<adres oraz port serwera TeamSync>/editcomment`. Tymi danymi są:

- lokalizacja wątku — pełna ścieżka folderu przechowującego pliki komentarzy oraz plik z metadanymi w dyskusji (plik `meta`),
- cały obiekt JSON zawierający dane komentarza w formie, w jakiej jest on odczytywany z pliku, z tą różnicą, że w zmiennej `comment` (zawierającej treść komentarza) znajduje się nowa,

zmodyfikowana przez użytkownika wypowiedź.

Po otrzymaniu żądania zawierającego powyższe informacje serwer odtwarza tytuł pliku komentarza, posługując się zmiennymi `timestamp` oraz `uid`, otrzymując pełną ścieżkę komentarza. Wczytuje plik komentarza zapisany na dysku, modyfikuje zmienną `comment` treścią podaną przez użytkownika i dodaje do słownika `history` nowy element zawierający pobrany znacznik czasowy z serwera *NTP* oraz nową treść wypowiedzi. Następnie zapisuje komentarz w tym samym pliku.

Aby zaoszczędzić czas, który tracony jest na odczytywanie pliku z systemu plików, serwer mógłby pominąć ten etap i — po dodaniu wpisu do zmiennej `history` — od razu komentarz zapisać. Jednak dane przesyłane z przeglądarki do serwera zawierają informacje nadmiarowe dotyczące interfejsu graficznego, które są zbędne dla serwera. Do poprawnego działania funkcji serwerowi wystarczyłyby dane:

- *comment* — nowa treść komentarza,
- *timestamp* — znacznik czasowy utworzenia komentarza (potrzebne do identyfikacji pliku w folderze wątku),
- *uid* — autor komentarza (potrzebne do identyfikacji pliku w folderze wątku),
- *fullthreadpath* — pełna ścieżka wątku, w którym komentarz jest fizycznie zapisany.

Jednakże zdecydowano się na przesłanie całej struktury komentarza ze względu na niewielką nadmiarowość danych oraz większe możliwości manipulacji zapisywaniem komentarza podczas ewentualnych ulepszeń aplikacji TeamSync w przyszłości.

6.2.3 Wątki

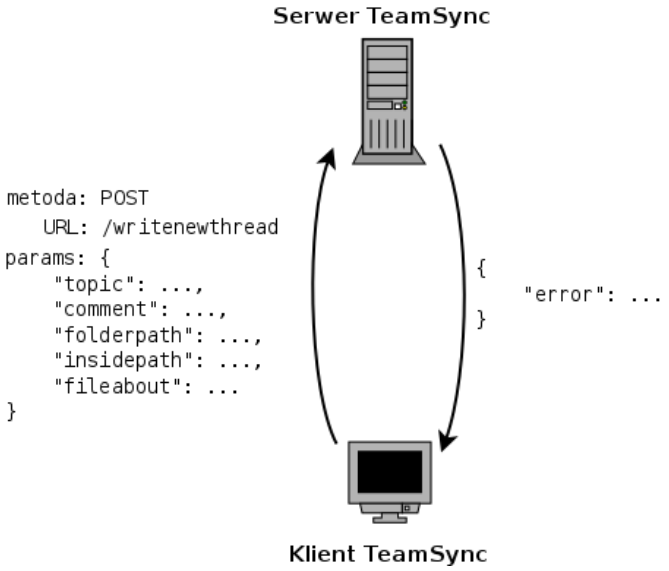
Użytkownicy mają możliwość tworzyć komentarze oraz je edytować, natomiast *TeamSync* nie umożliwia usuwania ich ze względu na zachowanie spójności logicznej dyskusji. W przypadku wątków możliwości użytkowników zostały dodatkowo ograniczone — edytowanie wątku zostało zminimalizowane wyłącznie do edycji treści pierwszego komentarza. Ze względu na fakt, iż była ona omówiona wcześniej (sekcja 6.2.2), omówiona zostanie wyłącznie procedura dodawania do systemu nowej dyskusji.

Tworzenie nowego wątku

Użytkownik tworząc nowy wątek, wysyła — poprzez aplikację kliencką w przeglądarce — do serwera żądanie `HTTP` za pomocą metody `POST` na adres `http://<adres oraz port serwera TeamSync>/writenewthread`. W zapytaniu przesyłane sa następujące parametry:

- *topic* — tytuł wątku wprowadzony przez użytkownika,
- *comment* — treść pierwszego komentarza wprowadzona przez użytkownika,
- *folderpath* — bezwzględna ścieżka synchronizowanego folderu w lokalnym systemie operacyjnym użytkownika,
- *insidepath* — lokalizacja wątku wewnątrz folderu nie uwzględniając samego katalogu przechowującego wątek,
- *fileabout* — nazwa pliku wybranego przez użytkownika, którego dotyczyć będzie wątek.

Po otrzymaniu żądania serwer w pierwszej kolejności pobiera znacznik czasowy z serwera NTP, który jest potrzebny do wygenerowania nazwy folderu z nowym wątkiem łączącej ze sobą pobrany znacznik, identyfikator użytkownika oraz separator nazwy (dokładny opis separatora znajduje się w



Rys. 6.10: Komunikacja w systemie *TeamSync* podczas wywoływania metody `writenewthread` — tworzenie nowego watku.

sekcji 4.2.3). Przyjmując jako separator użyty w implementacji ciąg znaków: `@#&$`, nazwa katalogu wygląda następująco:

```
< znacznik czasowy >@#&$< identyfikator użytkownika >
```

Następnie serwer ustala pełną ścieżkę wątku z przekazanych w żądaniu parametrów `folderpath` oraz `insidepath`:

```
folderpath/.Comments/insidepath/< nazwa katalogu z wątkiem >
```

Po ustaleniu pełnej lokalizacji wątku serwer tworzy folder o podanej wyżej ścieżce i umieszcza w nim plik `meta` uzupełniając go wartościami podanymi przez użytkownika w żądaniu. Poniżej przedstawiono przykładowe parametry przesyłane w celu stworzenia nowego wątku w postaci obiektu `JSON`.

```
{
    "topic": "Testowy tytuł",
    "comment": "Przykładowa treść komentarza",
    "insidepath": "/abc",
    "fileabout": "aaa",
    "folderpath": "/home/user/testowy katalog"
}
```

Rys. 6.11: Parametry przesyłane w przykładowym zapytaniu tworzącym nowy wątek.

Według parametrów zawartych w przykładzie nowy wątek będzie miał tytuł `Testowy tytuł`, a treść pierwszego komentarza to `Przykładowa treść komentarza`. Lokalizacja uwspólnionego folderu, wewnątrz którego zamieszczany jest wątek, to `/home/user/testowy katalog`, natomiast wątek umieszczony jest jeden poziom głębiej, w katalogu `abc`. Dodatkowo utworzona dyskusja dotyczy pliku o pełnej ścieżce `/home/user/testowy katalog/abc/aaa`.

W wyniku otrzymania powyższego żądania — zakładając, że pobrany znacznik czasowy to `1441566810000`, a identyfikator użytkownika zakładającego wątek to `FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM` — serwer utworzy wątek w lokalizacji o pełnej ścieżce:

```
/home/user/testowy katalog/.Comments/abc/
                1441566810000@#&$FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM
```

Wewnątrz tego folderu zostanie umieszczony plik `meta` oraz plik pierwszego komentarza o podanych poniżej zawartościach.

```
{
    "topic": "Testowy tytuł",
    "timestamp": "1441566810000",
    "fileabout": "/abc/aaa",
    "uid": "FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM"
}
```

Rys. 6.12: Zawartość pliku `meta` utworzonego w wyniku otrzymania przykładowego żądania.

```
{
    "comment": "Przykładowa treść komentarza",
    "timestamp": "1441566810000",
    "history": [
        {
            "comment": "Przykładowa treść komentarza",
            "timestamp": "1441566810000"
        }
    ],
    "readby": {
        "FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM": "1441566810000"
    },
    "uid": "FWZPQQKUE3JPQKWY557FERJ4SD3BSFMM"
}
```

Rys. 6.13: Zawartość pliku z pierwszym komentarzem utworzonego w wyniku otrzymania przykładowego żądania.

W pliku `meta` zapisanym w folderze w wyniku otrzymania przez serwer przykładowego żądania umieszczony został tytuł `Testowy tytuł` wpisany przez użytkownika, znacznik czasowy zwrócony przez serwer NTP, identyfikator autora wątku oraz wewnętrzna ścieżka do pliku, na którego temat powstała dyskusja. Jeśli wątek nie dotyczyłby żadnego z plików, wartość `fileabout` byłaby równa `<brak>`.

Zapisywanie plików komentarzy zostało dokładniej opisane w sekcji 6.2.2.

6.3 Aplikacja przeglądarkowa

Wśród podstawowych założeń implementacyjnych aplikacji *TeamSync* — poza założeniami wynikającymi z zastosowania systemu *BitTorrent Sync* jako narzędzia odpowiedzialnego za wymianę danych — znajdują się wymagania dotyczące interakcji człowieka z systemem. Podstawowymi założeniami — oprócz prostoty interfejsu oraz łatwości jego obsługi — w tym zakresie są: częstość odświeżania informacji oraz szybkość działania.

Aby sprostać powyższym wymaganiom i zachować interfejs w statycznym środowisku — jakim jest serwer WWW — zastosowano w pracy asynchroniczne zapytania (*AJAX* [21]) między częścią serwerową i kliencką. Podczas niemal każdej z czynności wykonywanych przez użytkownika w interfejsie odświeżana jest większa część danych pokazywanych użytkownikowi:

- komentarze,
- listy wątków,
- listy plików/katalogów przechowywanych w synchronizowanym folderze.

Two-way data binding

Ważnym mechanizmem umożliwiającym dynamiczną zmianę treści strony unikając jej całkowitego przeładowania, jest tzw. „*two-way data binding*" zaimplementowany przez twórców narzędzia *AngularJS*. Interfejs działający w tym systemie wyświetla elementy *modelu*, które użytkownik zaimplementuje — w przypadku aplikacji *TeamSync* wewnątrz modelu znajdują się np. wszystkie listy przechowujące dane (listy plików, wątków, komentarzy, identyfikator użytkownika, itp.) albo ustawienia aplikacji.

Ogólna koncepcja metody "*two-way data binding*" polega na zrealizowaniu jednocześnie dwóch następujących założeń dotyczących sposobu prezentacji w interfejsie danych pochodzących z modelu:

1. Jeśli dane zmienią się wewnątrz modelu, zmiana nastąpi również w interfejsie graficznym.
2. Jeśli dane prezentowane w interfejsie graficznym zostaną zmienione, zmiana nastąpi również w *modelu*.

W ten sposób — korzystając z pierwszego założenia — jeśli wewnątrz kodu przeglądarki nastąpi odświeżenie dowolnej zmiennej (np. listy plików) poprzez zapytanie asynchroniczne, zostanie ona z natychmiastowym skutkiem zmieniona w interfejsie graficznym. Przykładem wykorzystania drugiego założenia może być sytuacja, w której użytkownik chce posortować wątki według żądanego kryterium. Wprowadzając odmienne kryterium niż obecne, wywołuje szereg operacji nie na zmiennych wyświetlanych w interfejsie, lecz na zmiennych wewnątrz modelu.

Dokładniejszy opis i przykłady obydwóch powyższych założeń zostaną przedstawione w następnych podrozdziałach.

6.3.1 Wyświetlanie plików

Podczas odświeżania listy plików, w kodzie *javascript* interfejsu wysyłane jest asynchroniczne zapytanie metodą `POST`, a cała struktura, którą przeglądarka otrzyma w odpowiedzi od serwera, umieszczana jest wewnątrz zmiennej `files` reprezentującej listę plików. Aby przedstawić całą otrzymaną listę w interfejsie użytkownika, należy umieścić ją w strukturze, która umożliwi czytelne odwzorowanie listy. W tym celu posłużono się znacznikami `ul` oraz `li`, które są stworzone z myślą o prezentacji list (ul = unordered list). Przykładowe użycie znaczników `ul` oraz `li` wygląda następująco:

```
<ul>
    <li>Element 1</li>
    <li>Element 2</li>
    <li>Element 3</li>
</ul>
```

AngularJS umożliwia dynamiczną manipulację elementami listy poprzez dyrektywę `ng-repeat` dodawaną do tego znacznika, który ma zostać powtórzony. W przypadku powyższego przykładowego kodu dyrektywa `ng-repeat` zostanie dodana do pierwszego znacznika `li`. Jeśli w modelu istniałaby zmienna `list`, będąca tablicą zawierającą trzy elementy o typie znakowym: „`Element 1`", „`Element 2`" oraz „`Element 3`", poniższy kod w pliku *HTML* byłby jednakowy z wcześniejszym:

```
<ul>
    <li ng-repeat="item in list">{{ item }}</li>
</ul>
```

Dyrektywa `ng-repeat` tworzy pętlę przechodzącą po wszystkich elementach listy, która wewnątrz każdego znacznika `li` umieszcza element, którego aktualnie dotyczy iteracja. Podwójny nawias „{" służy do umieszczania zawartości przechowywanych wewnątrz zmiennych, które służą jako iterator (w powyższym przypadku iteratorem jest zmienna `item`).Jeśli modyfikacji (na przykład podczas asynchronicznego zapytania do serwera i zaktualizowania listy) ulegnie lista `list`, wewnątrz której odbywa się iteracja w dyrektywie `ng-repeat`, zmiana zostanie przeniesiona na interfejs. Spełnione zostanie pierwsze założenie metody *two-way data binding* — zmiana modelu wpłynie na interfejs graficzny.

Dyrektywa `ng-repeat` została wykorzystana w aplikacji *TeamSync* np. do prezentowania listy plików. Zmienna `files`, w której umieszczane są elementy pochodzące z odpowiedzi serwera, stanowi listę, wewnątrz której dyrektywa `ng-repeat` iteruje, wyświetlając odpowiednie informacje użytkownikowi.

```
<ul>
    <li ng-repeat="file in files">{{ file.name }}</li>
</ul>
```

Ze względu na założenia metody *two-way data binding*, w momencie odebrania odpowiedzi od serwera, przeglądarka natychmiast wyświetli aktualną listę plików, czego efektem jest niezbędna dynamika aplikacji. Ponieważ zawarte wewnątrz listy `files` elementy są obiektami `JSON`, wewnątrz podwójnych nawiasów klamrowych musiała zostać umieszczona któraś z wartości obiektu, a nie cały obiekt. Skoro użytkownik musi widzieć nazwę pliku, aby móc nawigować po synchronizowanym folderze, została użyta wartość obiektu `file`, której kluczem jest `name` (nazwa).

W podobny sposób działają wszystkie dynamiczne elementy strony interfejsu graficznego, które są modyfikowane zarówno przez serwer, jak i użytkownika. Przykładem zmiennych, które prezentowane sa w podobny sposób do plików, są:

- lista folderów,
- lista wszystkich użytkowników w folderze,
- lista komentarzy,
- lista wątków.

6.3.2 Filtrowanie komentarzy

Wyszukiwanie wewnątrz komentarzy wpisywanych przez użytkownika fraz w aplikacji *TeamSync* odbywa się w całości po stronie klienta, dzięki czemu jest bardzo szybkie i dynamiczne. Filtr jest uaktualniany i zaczyna działać podczas wprowadzenia każdego znaku. Odbywa się to dzięki

użyciu słowa kluczowego `filter` dostępnego w systemie *AngularJS*, którego umieszczenie w odpowiedni sposób wewnątrz kodu HTML aplikacji spowoduje wykluczenie ze zbioru danych elementów niepasujących do wzorca.

Użycie funkcji `filter` zostanie opisane na kodzie HTML z wcześniejszego przykładu, do którego dodano funkcjonalność filtrowania:

```
<ul>
    <li ng-repeat="item in list | filter:testFilter">{{ item }}</li>
</ul>
```

Znak „|" jest znakiem, który powoduje poddanie zbioru danych sprzed tego znaku (w tym przykładzie jest to lista `list`) działaniu operacji po tym znaku — filtrowaniu, lub sortowaniu (w tym przykładzie jest to filtrowanie). Słowo `filter` jest słowem kluczowym oznaczającym wykonywanie filtrowania na zbiorze danych `list`. `testFilter` jest zmienną pochodzącą z modelu, wewnątrz której przechowywana jest wartość filtru. Jeśli byłby spełnione poniższe warunki:

```
testFilter = "1"
list = ["Element 1", "Element 2", "Element 3"]
```

Wewnątrz znacznika `ul` zostałyby wyświetlone wyłącznie te elementy `li`, których zawartość zawierałaby znak „1". Posługując się obecnym przykładem, użytkownik wyświetlając powyższy kod z powyższymi danymi wewnątrz modelu zobaczyłby w swojej przeglądarce wyłącznie pierwszy element o treści `Element 1`.

Wewnątrz aplikacji *TeamSync* funkcjonalność słowa kluczowego `filter` została wykorzystana do filtrowania tresci komentarzy w poszukiwaniu żądanego przez użytkownika słowa. Wszystkie komentarze aktualnie wyświetlane na ekranie zostają poddawane filtracji w poszukiwaniu wprowadzonej frazy. W uproszczonej wersji kod HTML tej funkcjonalności w aplikacji *TeamSync* wygląda w następujący sposób:

```
<ul>
    <li ng-repeat="comment in comments | filter: searchphrase">
</ul>
```

Zmienna `comments` jest listą, do której asynchronicznie pobierane są z serwera komentarze w formie obiektów `JSON`. Podobnie jak w wyświetlaniu plików, również tutaj podczas aktualizacji tej zmiennej w modelu (dzięki technice *two-way data binding*) natychmiast aktualizowane są elementy w interfejsie graficznym. Po słowie kluczowym `filter` — sygnalizującym działanie filtra zbioru danych `comments` — znajduje się zmienna `searchphrase`, której wartość jest ciągiem znaków wyszukiwanym wśród komentarzy. Wyświetlany zbiór wypowiedzi (spośród listy `comments`) zostanie ograniczony tylko do tych, które zawierają frazę wpisaną do zmiennej `searchphrase`.

Podczas tej operacji system *AngularJS* skorzysta z drugiego założenia metody *two-way data binding* — użytkownik wprowadzając w interfejsie graficznym szukaną frazę, automatycznie uaktualni zmienną `searchphrase` w modelu. Z kolei zmienna `searchphrase` pobrana z modelu zostanie użyta do wyświetlenia zubożonego zbioru komentarzy o te, które nie pasują do wpisanego wzorca.

W systemie *AngularJS* możliwe jest stosowanie wielu filtrów na tym samym zbiorze danych. W aplikacji *TeamSync* zostało to wykorzystane, aby dodatkowo móc wyfiltrować te wiadomości, które zostały umieszczone przez konkretnych użytkowników.

Rozdział 7

Zakończenie

W ramach niniejszej pracy zaprojektowano oraz zaimplementowano system służący do wymiany zarówno plików, jak i komentarzy. System *TeamSync* w części transportu danych opiera się na aplikacji *BitTorrent Sync* działającej na protokole BitTorrent. Komunikację pomiędzy systemami umożliwia API stworzone przez producentów aplikacji *BitTorrent Sync*, które umożliwia kontrolowanie synchronizowanych danych. Wewnątrz stworzonego systemu *TeamSync* zostało zaimplementowanych wiele funkcjonalności zarówno podstawowych (wymiana komentarzy), jak i dodatkowych (wyszukiwanie, filtrowanie). Wszystkie najistotniejsze funkcje systemu *TeamSync* zostały wymienione poniżej.

Użytkownicy mogą zarządzać (dodawać, modyfikować oraz usuwać) folderami, które współdzielą z innymi użytkownikami w sieci. System *TeamSync* zapewnia pełną nawigację po synchronizowanych katalogach z poziomu interfejsu użytkownika. Interfejs dodatkowo umożliwia tworzenie przez użytkowników nowych wątków, wypowiadanie się wewnątrz nich oraz edycję komentarzy wraz z możliwością przeglądania historii zmian. Tworzone wątki mogą (lecz nie muszą) dotyczyć dowolnych plików zawartych w folderze.

Poza operacjami dokonywanymi na komentarzach system umożliwia swobodne wyszukiwanie informacji, poprzez narzędzia sortujące i filtrujące wypowiedzi oraz wątki. Dyskusje mogą zostać posortowane alfabetycznie, chronologicznie, według największej ilości postów oraz według najpóźniej dodanej odpowiedzi. Użytkownik ma możliwość filtrować wątki według procentowego udziału wybranych użytkowników w dyskusji. System również umożliwia wyszukiwanie wpisywanych przez użytkownika fraz w komentarzach z możliwością zmiany zbioru komentarzy — wątek, wszystkie komentarze, wszystkie komentarze konkretnego użytkownika. Dostępny jest też tryb statystyk, w którym wyświetlana jest ilość wypowiedzi każdego z użytkowników oraz ich procentowy udział w zbiorze komentarzy.

Szczegółowy model systemu, jego architektura, ogólny sposób działania, konfiguracja oraz impementacja zostały opisane w poprzednich rozdziałach niniejszej pracy.

System *TeamSync* został zaimplementowany w sposób umożliwiający jego dalszy rozwój. Wśród dodatkowych funkcjonalności, które mogłyby zostać zaimplementowane bez konieczności dokonywania dużych zmian w strukturze kodu (zarówno części klienckiej, jak i serwerowej) są:

1. Wykrywanie typów plików obrazów lub muzycznych (np. `jpg`, `mp3`) i możliwość ich podglądu (lub odtworzenia w przypadku plików muzycznych).
2. Dostosowanie aplikacji TeamSync do pracy z systemami operacyjnymi: Windows oraz MacOS.
3. Możliwość dokonywania zmian w synchronizowanym folderze (np. wyłączenie tzw. „trackera", wyszukiwanie innych użytkowników za pomocą DHT [13].

4. Dodanie kilku dodatkowych ustawień dotyczących graficznego interfejsu:
   - możliwość zmiany formatu daty wyświetlanej obok komentarzy,
   - możliwość definiowania własnego koloru czcionki lub ramki, wewnątrz której wyświetlane są komentarze użytkownika,
   - możliwość wybrania przez użytkownika pliku graficznego (tzw. „avatara") umieszczanego obok jego nazwy na liście użytkowników oraz obok każdego komentarza.

Dodatek A

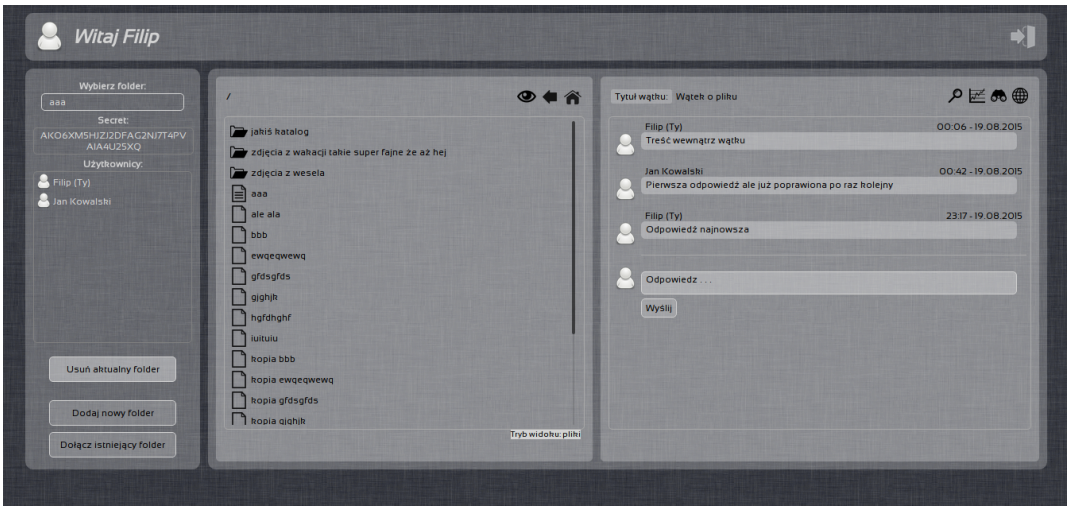
Instrukcja użytkownika

A.1 Graficzny interfejs użytkownika

Posługiwanie się przez użytkownika wszystkimi wymienionymi funkcjonalnościami powinno być łatwe oraz intuicyjne. Dostępne opcje zostały zaimplementowane w taki sposób, aby przeciętny odbiorca oprogramowania mógł dokonać żądanych działań w jak najkrótszym czasie i z jak najmniejszym wysiłkiem. Dostęp do większości funkcji uzyskiwany jest poprzez kliknięcia myszką komputera, która jest łatwiejszym sposobem nawigowania przez użytkowników wewnątrz aplikacji niż klawiatura.

Interakcja odbiorcy z aplikacją TeamSync poprzez klawiaturę została zredukowana do niezbędnego minimum, czyli wprowadzania danych (komentarzy, tytułów wątków, tożsamości oraz łańcucha *secret* podczas dodawania nowego folderu itp.) oraz wyszukiwania fraz tekstowych. Nie ma możliwości, aby użytkownik nawigując wewnątrz systemu musiał ręcznie wpisywać np. nazwę innego użytkownika będącego w systemie, którego komentarze chciałby wyświetlić — zamiast wprowadzania tych danych ręcznie, system wylistuje użytkowników pozostawiając tylko konieczność wyboru poprzez manipulację myszką.

Struktura graficzna aplikacji



Rys. A.1: Wygląd przykładowego folderu w graficznym interfejsie użytkownika.

Aby użytkownik łatwo odnalazł potrzebne mu funkcje ekran podzielony został na trzy części (nie licząc paska nagłówkowego) w układzie kolumnowym. Pierwsza — część folderów — służy do manipulacji synchronizowanymi katalogami. Druga — środkowa — służy do nawigacji po folderach oraz do przeglądania wątków, lecz nie wyświetlając wśród nich komentarzy. Trzecia natomiast przeznaczona jest do wyświetlania treści komentarzy i manipulowania zbiorem wyświetlanych wypowiedzi. Jeśli w danym momencie nie jest wybrany żaden z folderów z listy, dwie ostatnie części zajmujące główną część ekranu są nieaktywne.

W dalszej części podrozdziału omówione zostaną dokładniej przedstawione sekcje.

Sekcja folderów

Jak opisano wcześniej, wewnątrz najmniejszej z trzech sekcji użytkownik może wykonywać operacje związane folderami, które współdzieli z innymi użytkownikami. Znajduje się w niej między innymi lista wszystkich zsynchronizowanych folderów (według ograniczeń systemu BitTorrent Sync może ich być maksymalnie 10) wewnątrz kontrolki typu *drop-down list* [22], dzięki której użytkownik może w łatwy sposób dokonać wyboru.

Poniżej listy folderów znajdują się obszary, które po wybraniu katalogu wypełniają się odpowiednio danymi: ciągiem znaków tzw. *secretem* oraz listą użytkowników, którzy przynależą do wybranego folderu. Aby ułatwić użytkownikowi identyfikację samego siebie spośród innych, do wyświetlanej nazwy jest dodawany ciąg znaków „(Ty)".

Poza powyżej wymienionymi danymi w sekcji folderów znajdują się również przyciski umożliwiające usunięcie aktualnie wybranego folderu (przycisk jest nieaktywny, jeśli żaden folder nie został jeszcze wybrany) oraz dodanie nowego katalogu, które może odbyć się na dwa sposoby. Pierwszy zakłada, że folder jest inicjowany od zera i nie ma potrzeby wpisywania łańcucha *secret*, ponieważ zostanie on wygenerowany przez system. Drugi wymaga jego podania i łączy wskazany z systemu plików pusty katalog z folderem innego użytkownika, od którego otrzymano łańcuch *secret*. W obydwóch przypadkach konieczne jest podanie ścieżki do folderu, który ma być uwspólniony oraz tożsamości, czyli nazwy wyświetlanej dla innych użytkowników, mającej swój zasięg tylko wewnątrz katalogu.

Sekcja wątków

Sekcja wątków składa się z nagłówka oraz części głównej zajmującej przeważający obszar całej sekcji. W nagłówku wyświetlana jest ścieżka aktualnej lokalizacji (nie bezpośrednia, lecz ta wewnątrz folderu, który będzie „korzeniem" ścieżki), w której znajduje się użytkownik, zmieniająca się dynamicznie w interfejsie w miarę postępu nawigacji po katalogu.

Wewnątrz nagłówka, po przeciwnej stronie ścieżki znajdują się ikony funkcji, które pomagają w nawigacji oraz zmianie trybu widoku w głównej części sekcji. Naciśnięcie ikony prezentującej strzałkę skierowaną w lewo („do tyłu") powoduje cofnięcie się ścieżki lokalizacji o jeden poziom. Uruchomienie funkcji ukrytej pod ikoną domu sprawia, że — niezależnie od obecnej lokalizacji, w której się znajduje — użytkownik zostanie sprowadzony do podstawowego poziomu w folderze — jego „korzenia".

Główna część sekcji służy do prezentacji plików oraz wątków w dwóch możliwych trybach, pomiędzy którymi przełączanie odbywa się za pomocą wciśnięcia kursorem myszki ikony zawierającej grafikę przedstawiającą oko, w nagłówku sekcji wątków.

Tryb plików — uruchamiając ten tryb, użytkownik może przeglądać zawartość synchronizowanego folderu w takiej postaci, w jakiej jest on zapisany na dysku. Uruchamiając kliknięciem

myszy element z ikoną przedstawiającą folder użytkownik przechodzi do lokalizacji wewnątrz niego. Oprócz plików i katalogów wylistowanych w trybie plików, użytkownik widzi również wątki — nie tylko te umieszczone w odpowiedniej lokalizacji ale też te przyporządkowane do konkretnych plików (jeśli do pliku jest przypisany wątek, ikona reprezentująca ten plik jest dostrzegalnie zmieniona, a po jej kliknięciu rozwijana jest lista wszystkich dyskusji dotyczących tego pliku).

Tryb wątków — przełączenie do tego trybu skutkuje wylistowaniem tylko i wyłącznie wątków napisanych wewnątrz folderu (niezależnie od lokalizacji), a uruchamianie funkcji nawigacji w tym trybie — poprzez kliknięcia wewnątrz obszarów ikon (*back* oraz *home*) — nie wywołuje żadnego efektu. Dodatkowo, wyświetlane są dwie ikony, dzięki którym użytkownik będzie miał możliwość posortować wątki według kilku kategorii oraz przefiltrować je według procentowego udziału grupy użytkowników wewnątrz nich (sekcja A.3). Obydwie te funkcje są dostępne z dodatkowych menu kontekstowych, pojawiających się w momencie kliknięcia kursorem myszki odpowiedniej ikony i umożliwiających precyzowanie dostępnych parametrów.

Sekcja komentarzy

Sekcja komentarzy swoją strukturą oraz rozmiarami podobna jest do sekcji wątków — zawiera nagłówek oraz część główną. Wewnątrz nagłówka znajduje się pole z opisem aktualnego widoku np. „Nowy wątek", „Wątek:" z tytułem aktualnej dyskusji lub „Wszystkie komentarze". Obok wyświetlanego tekstu, wyrównane do prawej strony uszeregowane są ikony, za pomocą których możliwa jest manipulacja zbiorem, kolejnością lub filtracją pokazywanych komentarzy.

Poniżej znajduje się lista dostępnych ikon wraz z opisem uruchamianych przez nie funkcji.

Ikona zmiany widoków — prezentowana jako globus. Klikając na nią kursorem, użytkownik uaktywnia menu kontekstowe, które zawiera w sobie jedną kontrolkę typu *drop-down list* z pięcioma wartościami: *nowy wątek*, *aktualny wątek*, *wszystkie komentarze z lokalizacji*, *wszystkie kometarze użytkownika*, *wszystkie komentarze*. Jeśli użytkownik wybierze opcję *wszystkie komentarze użytkownika*, wyświetlana jest dodatkowa kontrolka typu *drop-down list* wewnątrz której wylistowani są użytkownicy. Dokładniejszy opis efektów użycia widoków znajduje sie w sekcji A.2.

Ikona zmiany spójności — ikona przedstawiająca lornetkę, służy do uruchamiania menu kontekstowego zawierającego jedną kontrolkę typu *drop-down list* zawierającą `n + 1` pozycji, gdzie `n` to liczba użytkowników współdzielących folder. Pierwsza wartość wyświetlana na liście to „NTP (globalna spójność)", natomiast pozostałe elementy to nazwy użytkowników. Użyteczność tej funkcjonalności została dokładnie opisana w sekcji 5.3.

Ikona statystyk — graficznie przedstawiona jako wykres. Za jej pomocą użytkownik może wyświetlić statystyki napisanych komentarzy w aktualnym ich zbiorze. Generowane zestawienie zawiera listę wszystkich autorów postów wraz z:

- sumaryczną liczbą komentarzy użytkownika,
- łączną liczbą wszystkich komentarzy,
- procentowym udziałem postów użytkownika w aktualnym zbiorze wypowiedzi.

Użytkownik może niezależnie od widoku w jakim obecnie pracuje przejść poprzez kliknięcie kursorem myszy w ikonę statystyk do trybu, w którym wyświetlane jest wyłącznie zestawienie z wymienionymi wyżej danymi. Aby z powrotem móc czytać komentarze, należy ponownie kliknąć ikonę. Statystyki zawsze będą obejmowały zakres aktualnego widoku —

jeśli użytkownik przegląda konkretny wątek, zestawienie zostanie wygenerowane tylko dla tego wątku.

Ikona wyszukiwania — reprezentowana przez grafikę przedstawiającą lupę. Jej naciśnięcie powoduje pojawienie się wewnątrz nagłówka, tuż obok ikony pola tekstowego, do którego użytkownik musi wpisać wyszukiwaną frazę. W trakcie wpisywania szukanego tekstu komentarze są przeszukiwane i odfiltrowane są te z nich, które nie pasują do wpisanego przez użytkownika wzorca. Po ponownym wciśnięciu ikony, interfejs graficzny usuwa pole tekstowe z ekranu, a odrzucone wcześniej komentarze zostają przywrócone. Dzieje się tak z uwagi na fakt, że filtr wyszukiwania działa tylko w momencie, gdy pole tekstowe, do którego należy wpisać szukaną frazę, jest widoczne.

Ikony będące wewnątrz nagłówka są dynamicznymi elementami interfejsu graficznego — jeśli nie są potrzebne, nie są wyświetlane. Ma to miejsce np. podczas zamieszczania nowego wątku — podczas wpisywania jego tytułu oraz treści pierwszego komentarza nie ma potrzeby wyszukiwania czegokolwiek w komentarzach, ani zmiany spójności kolejności wypowiedzi w wątku. Podobnie, jeśli żaden z wątków nie został jeszcze wybrany — a więc komentarze nie są jeszcze wyświetlane — to ikony spójności, statystyk oraz wyszukiwania nie są potrzebne użytkownikowi. Nie są więc wyświetlane w interfejsie, co wpływa na jego przejrzystość oraz awaryjność (uniemożliwianie popełniania błędów przez użytkownika).

W części głównej sekcji komentarzy znajdują się wypowiedzi użytkowników wewnątrz dodających przejrzystości ramek, otoczone dodatkowymi danymi takimi jak:

- grafika przedstawiająca tzw. „avatar" użytkownika,
- nazwa autora komentarza,
- czas i data umieszczenia komentarza w formacie pobranym z pliku konfiguracyjnego (sekcja 5.2.2),
- jeśli zbiór komentarzy obejmuje więcej niż jeden wątek, dodatkowo blisko pod wypowiedzią zamieszczany jest tytuł dyskusji, z której pochodzi komentarz. Jest on hiperłączem prowadzącym do całej zawartości wątku, z którego pochodzi wypowiedź, dzięki czemu użytkownik poprzez jedno kliknięcie myszy może przejść do całej rozmowy i łatwo zapoznać się z kontekstem, w którym został umieszczony interesujący go komentarz.

Graficzny interfejs aplikacji TeamSync w łatwy sposób umożliwia edycję komentarzy oraz przeglądanie historii zmian każdej z wypowiedzi (o ile taka zmiana miała miejsce). Jeśli użytkownik jest autorem komentarza, po umieszczeniu nad nim kursora w sekcji komentarzy wyświetlona zostanie dodatkowa ikona przedstawiająca długopis obok treści wypowiedzi. Po jej uruchomieniu cała aktualna treść komentarza zostanie umieszczona wewnątrz pola tekstowego i dodane zostaną dodatkowe dwa przyciski pod polem tekstowym: przycisk zatwierdzający zmianę treści oraz anulujący ją.

Umieszczenie kursora myszki nad przynajmniej raz zmodyfikowanym przez autora komentarzem skutkuje pojawieniem się — w miejscu tuż obok ikony edycji wypowiedzi — ikony historii posta. Po jej uruchomieniu zostaje wyświetlona lista z wcześniejszymi wersjami komentarza wraz z czasem i datą zmiany jego treści.

Użytkownik przeglądający komentarze w obrębie jednego wątku ma możliwość szybkiego zabrania głosu w dyskusji — za ostatnim, najnowszym komentarzem znajduje się pole tekstowe, do którego wpisywana jest treść odpowiedzi oraz przycisk zatwierdzający komentarz. Po zatwierdzeniu, wprowadzona wypowiedź będzie wypowiedzią najświeższą i system gotowy jest do dalszej pracy.

A.2 Widoki

Aby stworzyć użytkownikowi jak najlepsze warunki do łatwego i szybkiego przeglądania komentarzy, wyszukiwania zawartych w nich informacji lub przeglądania statystyk ich dotyczących zaimplementowano tzw. „widoki", pomiędzy którymi użytkownik może wybierać w zależności od aktualnej potrzeby. Jako widok można rozumieć zakres komentarzy wyświetlanych w aplikacji. Użytkownik ma do wyboru pięć widoków, między którymi może w dowolnej chwili wybrać, jednak nie pomiędzy wszystkimi widokami jest możliwe przejście. Dodatkowo, w niektórych przypadkach przejść międyz widokami jest możliwość utraty części danych np. niezatwierdzoną odpowiedź w którymś z wątków.

Widok nowego wątku

Widok, który uruchamiany jest podczas inicjowania wątku. Nie są wyświetlane w nim żadne komentarze, a użytkownik proszony jest o podanie tytułu wątku, treści pierwszego komentarza oraz ewentualnego pliku, którego wątek będzie dotyczył. Do trybu nowego wątku można przejść wyłącznie poprzez wciśnięcie przycisku „Nowy wątek" w jednej z lokalizacji, bądź „Nowy wątek w pliku", jeśli dyskusja będzie dotyczyć pliku (użytkownik musi wskazać konkretny plik z sekcji plików). Taka forma rozpoczęcia wątku jest konieczna ze względu na konieczność wskazania aplikacji lokalizacji wątku. Opcja „Nowy wątek" wewnątrz rozwijanego menu w interfejsie graficznym — dotyczącego zmiany widoku — jest zablokowana.

Widok aktualnego wątku

Widok uruchamiany, gdy użytkownik wybierze z listy plików element z ikoną wątku lub dowolny element z listy wątków. Po wybraniu dyskusji następuje odczytywanie komentarzy z folderu z wątkiem przez serwer i wyświetlenie ich przez przeglądarkę w graficznym interfejsie. Przejście do tego widoku jest możliwe wyłącznie poprzez uruchomienie kursorem odpowiedniego elementu z listy plików lub wątków. Aplikacja musi otrzymać precyzyjną informację, którą konkretnie dyskusję użytkownik chce przeczytać, aby ją wyświetlić. Podobnie jak w przypadku widoku nowego wątku, opcja w menu „Aktualny wątek" jest zablokowana podczas przeglądania komentarzy w innym widoku.

Widok z bieżącej lokalizacji

Zakres komentarzy w tym widoku obejmuje wypowiedzi z wszystkich dyskusji pobranych z bieżącej lokalizacji — tej, którą użytkownik obecnie przegląda w przeglądarce plików. Przejście do tego widoku możliwe jest z każdego innego widoku, użytkownik musi jednak liczyć się z możliwą utratą treści np. podczas tworzenia nowego wątku lub odpowiedzi w już istniejącym. Podczas przełączania widoków wpisywane i niezatwierdzone dane w tych polach tekstowych są tracone.

Widok wszystkich komentarzy użytkownika

Aplikacja wyświetli wyłącznie te wypowiedzi, których autorem jest wybrany użytkownik. Podobnie jak z widokiem dotyczącym lokalizacji, przejście na niego jest możliwe w każdym momencie i należy liczyć się z możliwością utraty danych.

Widok wszystkich komentarzy

Zakres wyświetlanych komentarzy obejmujący wszystkie komentarze wprowadzone do aplikacji niezależnie od ich lokalizacji lub autora. Do tego widoku użytkownik może przejść w dowolnym momencie licząc się z utratą niezatwierdzonych komentarzy lub wątków. Przydatny użytkownikowi, gdy chce wyszukać ciąg tekstowy w bazie wszystkich wypowiedzi.

A.3 Filtrowanie listy wątków

Celem aplikacji TeamSync jest nie tylko zaawansowana manipulacja umieszczanymi komentarzami, ale również możliwość łatwego wyszukiwania dyskusji na poziomie nie pojedynczych wypowiedzi, a całych wątków. Jeśli użytkownik chciałby wydzielić z wszytkich dyskusji takie, w których wybrani przez niego użytkownicy brali większościowy bądź mniejszościowy udział, system umożliwi mu to za pomocą intuicyjnego menu kontekstowego uruchamianego w sekcji plików ikoną przedstawiającą grupę ludzi.

Uruchamiając tą część interfejsu użytkownik ma możliwość zaznaczyć użytkownika (jednego lub kilku) z listy wszystkich użytkowników współdzielących folder i wybrać próg procentowy dla zaangażowania wybranych użytkowników w wątek. Dostępne progi procentowe to `10%`, `30%`, `50%`, `70%`, `90%`. Dodatkowo istnieje możliwość określenia, czy aplikacja ma wybrać te wątki, w których suma komentarzy wybranej grupy użytkowników jest większa od łącznej liczby wypowiedzi w wątku, czy te, w których jest ona mniejsza.

Przykładem użycia funkcjonalności mogłaby być sytuacja, w której użytkownik chciałby odnaleźć wątek nie pamiętając konkretnych treści komentarzy zawartych wewnątrz niego, natomiast pamiętając, że ilość wypowiedzi dwóch innych użytkowników (`A` oraz `B`) zdecydowanie zdominowała dyskusję, co rzadko ma miejsce w pozostałych wątkach. Wówczas — ustawiając filtr według określonych preferencji (zaznaczony próg `>70%` lub `>90%` oraz wybrani użytkownicy `A` oraz `B` z listy wszystkich użytkowników) — przy niewielkim wysiłku znacznie zawęziłby obszar poszukiwań wątku.

A.4 Użycie wybranych funkcji interfejsu graficznego

W niniejszej części dodatku zostaną zaprezentowane sposoby użycia wybranych funkcji systemu *TeamSync*.

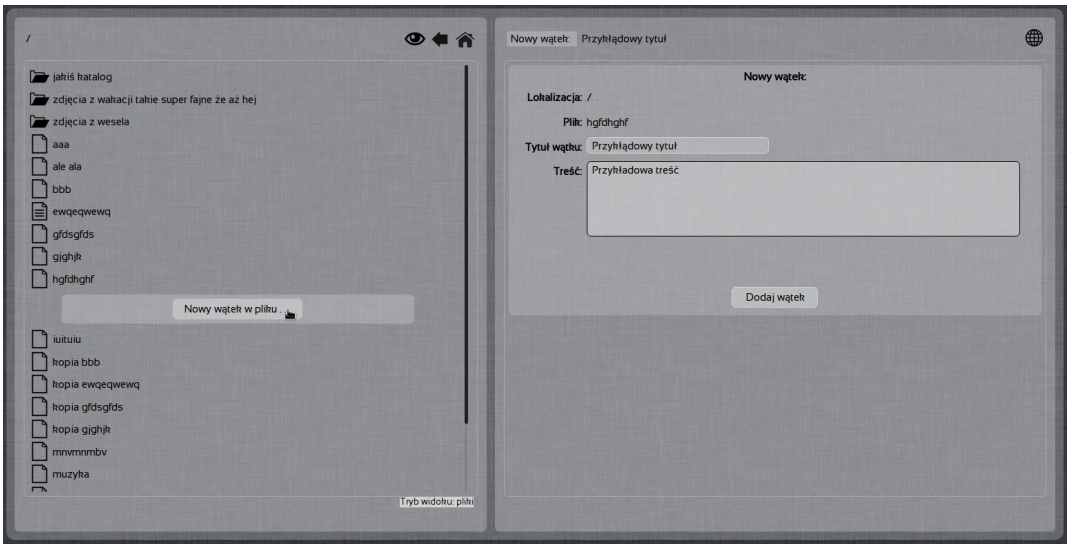
Tworzenie nowego wątku

Aby utworzyć nowy wątek w systemie, należy wykonać następujące kroki w sekcji plików:

1. Nawigując wewnątrz sekcji plików, znaleźć żądaną lokalizację wątku (lub żądany plik, któłrego ma dotyczyć dyskusja).
2. Kliknąć kursorem myszy przycisk „*Nowy wątek*” (lub „*Nowy wątek w pliku*”).

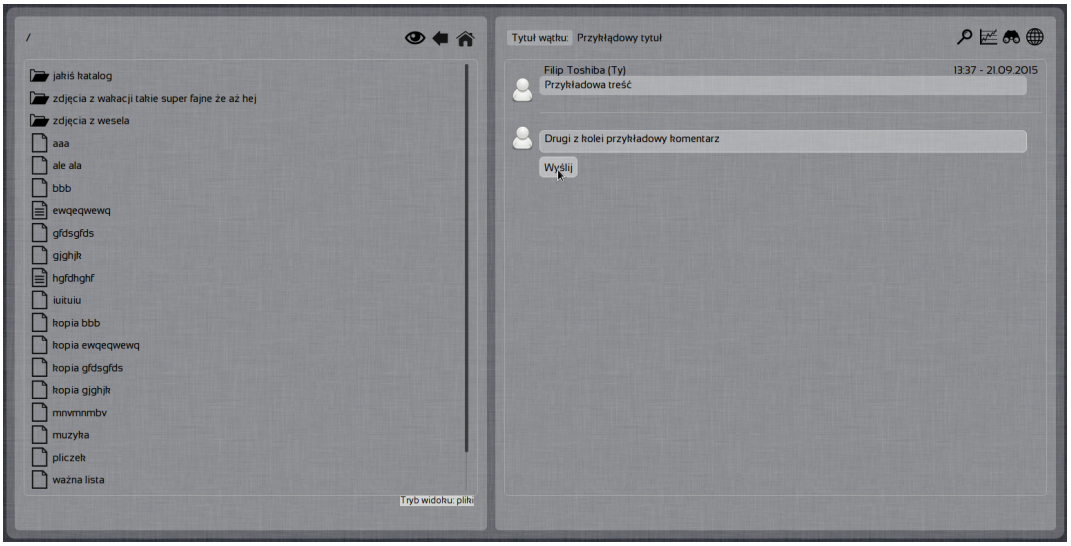
Po wykonaniu powyższych czynności, następne kroki należy wykonać wewnątrz sekcji komentarzy, gdzie pojawią się puste pola tekstowe: „*Tytuł wątku*” oraz „*Treść komentarza*”:

1. Uzupełnić wyżej wymienione pola tekstowe (w przypadku nie wypełnienia, zostanie wyświetlony odpowiedni komunikat).
2. Zatwierdzić utworzenie nowego wątku przyciskiem „*Dodaj wątek*”



Rys. A.2: Zrzut ekranu z interfejsu graficznego podczas tworzenia nowego wątku.

Wprowadzanie nowego komentarza



Rys. A.3: Zrzut ekranu z interfejsu graficznego podczas dodawania nowego komentarza.

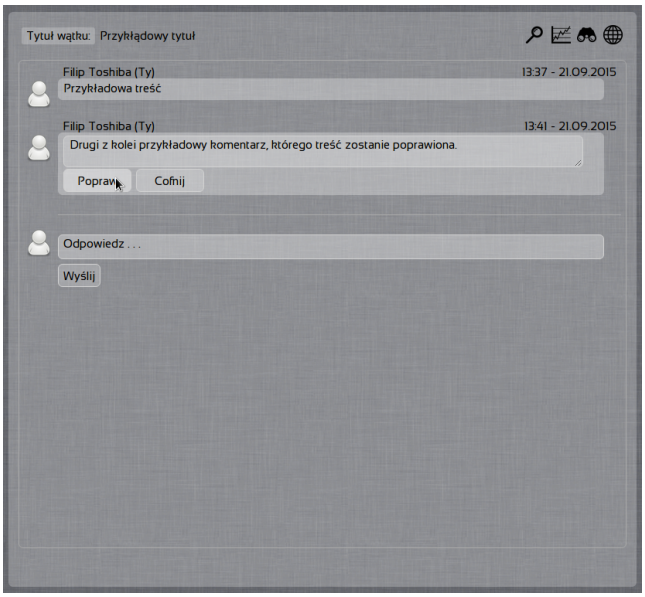
Podczas dodawania nowego komentarza użytkownik musi brać pod uwagę ograniczenie systemu polegające na tym, że aby umieścić nową wypowiedź, użytkownik powinien się znajdować w widoku aktualnego wątku (dokładny opis widoków znajduje się w sekcji A.2). Jest to konieczne ze względu na fakt, iż nie będąc w widoku wątku, system nie będzie mógł umieścić komentarza w systemie plików, ponieważ nie będzie wiedział, w którym wątku stworzyć plik z nowym komentarzem.

Aby dodać nowy komentarz w dyskusji, użytkownik musi wykonać nastepujące kroki:

1. Po wyświetleniu komnetarzy z jednej dyskusji, użytkownik powinien przejść na sam dół sekcji komentarzy (tam, gdzie znajduje się najświeższa wypowiedź).
2. Kliknięciem kursora myszy aktywować pole tekstowe i wpisać do niego treść wypowiedzi.

3. Zatwierdzić komentarz przyciskiem „*Dodaj*”.

Modyfikacja komentarza

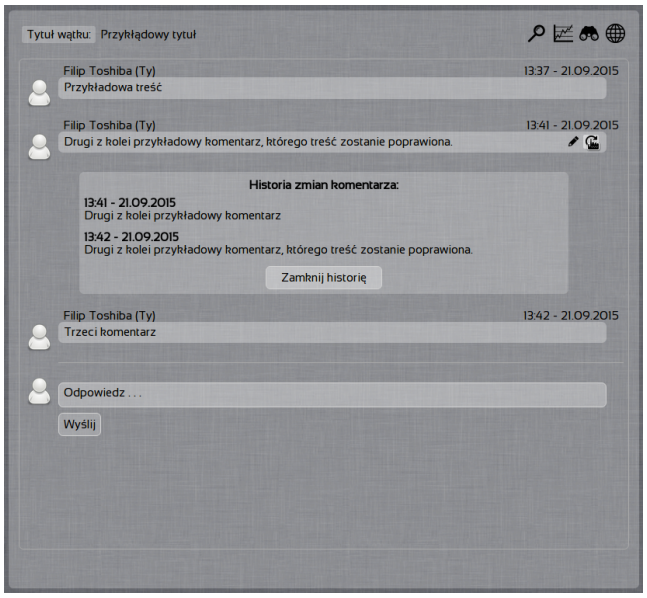


Rys. A.4: Zrzut ekranu z interfejsu graficznego podczas edycji komentarza.

Jeśli użytkownik chce zmodyfikować treść komentarza, może tego dokonać, o ile jest jego autorem. Edycja treści odbywa się w sekcji komentarzy i przebiega w następujący sposób:

1. Po ustawieniu przez użytkownika kursora nad obszar komentarza wyświetlane są na nim przyciski inicjujące edycję oraz pokazujące historię zmian w jego treści (o ile zmiany kiedykolwiek nastąpiły).
2. Po kliknięciu przycisku uruchamiającego edycję aktualny tekst komentarza umieszczany jest wewnątrz pola tekstowego, dodatkowo pojawia się przycisk akceptujący zmiany. Tekst jest edytowalny.
3. Po wprowadzeniu zmian i zatwierdzeniu ich przyciskiem plik zk omentarzem jest modyfikowany.

Wyświetlanie historii zmian komentarza



Rys. A.5: Zrzut ekranu z interfejsu graficznego podczas wyświetlania historii komentarza.

Przeglądanie historii zmian komentarzy jest możliwe tylko dla tych wypowiedzi, które zostały w przeszłości zmodyfikowane przez ich autorów. Historia może być wyświetlana przez każdego użytkownika w systemie. Przeglądanie modyfikacji odbywa się w sekcji komentarzy w następujący sposób:

1. Po ustawieniu przez użytkownika kursora nad obszar komentarza wyświetlane są na nim przyciski inicjujące edycję oraz pokazujące historię zmian w jego treści (o ile zmiany kiedykolwiek nastąpiły).
2. Po uruchomieniu przycisku historii wyświetlona zostanie ramka pod komentarzem, zawierająca pełną treść komentarza po edycji, wraz z jej datą.
3. Aby wyłączyć przeglądanie historii komentarza, należy kliknąć kursorem myszy na przycisk „*Zamknij historię*” wewnątrz ramki.

Literatura

[1] http://www.dropbox.com/.

[2] http://www.sugarsync.com.

[3] http://onedrive.live.com.

[4] http://www.mediafire.com/.

[5] http://www.box.com/.

[6] http://www.drive.google.com/.

[7] http://mega.nz.

[8] http://www.sher.ly/.

[9] http://technet.microsoft.com/en-us/library/cc939973.aspx.

[10] http://www.samba.org.

[11] http://www.symform.com/.

[12] http://www.getsync.com/.

[13] http://en.wikipedia.org/wiki/Distributed_hash_table.

[14] http://www.ntp.org/.

[15] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[16] http://en.wikipedia.org/wiki/Basic_access_authentication.

[17] http://angularjs.org.

[18] http://python.org.

[19] http://djangoproject.com.

[20] http://en.wikipedia.org/wiki/Representational_state_transfer.

[21] http://en.wikipedia.org/wiki/Ajax_(programming).

[22] http://en.wikipedia.org/wiki/Drop-down_list.

[23] Specification for the advanced encryption standard (aes). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[24] Douglas Crockford. The application/json media type for javascript object notation (json).

[25] Jose R. C. Cruz. Error connection with reed-solomon. [on-line] http://www.drdobbs.com/testing/error-correction-with-reed-solomon/240157266, 2013.

[26] Harry Fairhead. Ntp the latest open source security problem.

[27] Johan Pouwelse, Paweł Garbacki, Dick Epema, Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis, 2005.

[28] J. Reschke R. Fielding. Hypertext transfer protocol (http/1.1): Authentication.

[29] Scott Wolchok, J. Alex Halderman. Crawling bittorrent dhts for fun and profit.

[30] Xining Li Xiao Shu. A scalable and robust dht protocol for structured p2p network.

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Filip Rachwalak",
    title = "{TeamSync --- System wymiany danych i komentarzy w systemach P2P}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2015",
}
```