Projekt

Systemy odporne na błędy

Wydział Elektrotechniki Automatyki i Informatyki

Politechnika Świętokrzyska

Studia: **Stacjonarne II stopnia**	Kierunek: **Informatyka**
Grupa: **2ID22B**	Skład zespołu: 1. **Zachariasz Łukasz** 2. **Zakrzewski Mateusz**

Temat projektu:

BitTorrent: Trakery

1. WSTĘP

Celem projektu była implementacja oprogramowania w postaci symulatora protokołu BitTorrent z wykorzystaniem trackerów. W założeniach projektu było nadmienione, że system ma składać się z **3 trackerów** oraz **11 uczestników**.

W drugim rozdziale przedstawiono teorię omawianego zagadnienia: protokołu *BitTorrent* oraz podstawowych pojęć go dotyczących, czyli: modelu sieci *P2P*, *trakera* oraz pliku *torrent*. Na koniec przedstawiono też szczegółową zasadę działania opisanych elementów.

Trzeci rozdział poświecono architekturze oraz przybliżeniu sposobu implementacji rozproszonego symulatora, przedstawiającego działanie elementów protokołu z głównym naciskiem na wymianę danych. W rozdziale przedstawiono i opisano najważniejsze fragmentu zaimplementowanego symulatora oraz jego obsługę z poziomu interfejsu użytkownika.

W rozdziale czwartym opisano schemat testowania programu, poparty wynikowymi zrzutami ekranu. Ostatnie dwa rozdziały to wnioski z wykonanej pracy projektowej oraz bibliografia.

2. PROTOKÓŁ BITTORRENT

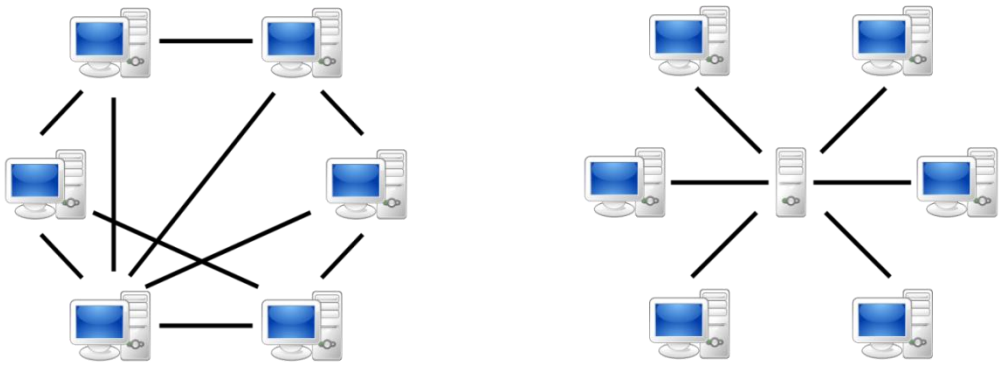
BitTorrent jest protokołem komunikacyjnym do udostępniania plików ***P2P*** (*ang. peer-to-peer*), który służy do dystrybucji danych i plików elektronicznych przez Internet w sposób zdecentralizowany. Aby wysyłać lub odbierać pliki, użytkownik (na swoim komputerze podłączonym do Internetu) korzysta ze specjalnego programu klienta, implementującego protokół *BitTorrent* (popularne programy klienckie to np.: *BitTorrent, µTorrent, Xunlei Thunder, Transmission, qBittorrent, Vuze, Deluge, BitComet* i *Tixati*). Za pomocą klienta tak zwane ***trakery*** (*ang. tracker*) dostarczają listę plików dostępnych do przesłania i pozwalają klientowi znaleźć użytkowników równorzędnych zwanych ***seedami*** (*ang. seed*), którzy mogą dostarczać pliki lub ich fragmenty.

BitTorrent jest jednym z najpopularniejszych protokołów do przesyłania dużych plików, takich jak np.: cyfrowe pliki wideo lub audio. Szacuje się, że sieci *P2P* stanowią łącznie około 43% do 70% całego ruchu internetowego, w zależności od lokalizacji (według stanu na luty 2009 r.). W lutym 2013 r. *BitTorrent* był odpowiedzialny za 3,35% całkowitej przepustowości na całym świecie, to ponad połowa z sześcioprocentowej całkowitej przepustowości przeznaczonej na udostępnianie plików w tym roku.

Protokół został zaprojektowany przez: ***Brama Cohena*** (byłego wówczas studenta z *USA*) w kwietniu 2001 r. i wydany (jako pierwsza dostępna wersja) 2 lipca 2001 r.

2.1. Model P2P

Model komunikacji *P2P* w sieci komputerowej, w odróżnieniu od architektury *klient-serwer,* zapewnia wszystkim *hostom* te same uprawnienia. Każdy węzeł sieci zwany *hostem*, czyli komputer użytkownika, może jednocześnie pełnić rolę klienta i serwera. W najpopularniejszej implementacji modelu *P2P*, jaką są programy do wymiany plików w Internecie, każdy *host* spełnia rolę serwera, przyjmując połączenia od innych użytkowników sieci, oraz klienta, łącząc się i wysyłając i/lub pobierając pliki z innych *hostów* działających w tej samej sieci *P2P*. Wymiana plików jest prowadzona bezpośrednio pomiędzy *hostami*. Sieć *P2P* charakteryzuje się zmiennością struktury węzłów sieci, spowodowaną zmiennością liczby i lokalizacji sieciowej aktualnie aktywnych *hostów*. Porównanie sieci *P2P* z siecią *klient-serwer* przedstawiono na rysunku 2.1.



Rys. 2.1. Schemat sieci *P2P* (po lewej) i sieci *klient-serwer* (po prawej).

Najpopularniejsze sieci *P2P* służą do współdzielenia plików w Internecie. Można wyróżnić dwie odmiany. Pierwszą, sieci bez centralnego serwera, które nie mają centralnej bazy o zasobach, oraz sieci z centralnym serwerem lub centralnymi serwerami, które przechowują informacje o użytkownikach podłączonych w danej chwili do sieci, oraz (w niektórych wypadkach) o udostępnianych zasobach. Centralne serwery oferują czasami także dodatkowe usługi, na przykład czat. Sieci wyposażone w centralny serwer są znacznie bardziej efektywne, gdyż nowy użytkownik podłączający się do sieci otrzymuje na wstępie listę wszystkich użytkowników podłączonych do danego serwera, ma także dostęp (najczęściej) do indeksu dostępnych plików, który może błyskawicznie przeszukać.

Zmienna struktura sieci *P2P*, uzależniona od liczby podłączonych użytkowników, niesie ze sobą ryzyko odcięcia od sieci w momencie, gdy wszyscy „sąsiedzi" rozłączą się w tym samym czasie. Ze względu na zachowanie się węzłów po utracie „sąsiada" można wydzielić dwa modele sieci: model pasywny – sieć nie podejmuje żadnych działań; model aktywny – sieć próbuje zastąpić utracone połączenie nowym, wybieranym w oparciu o specjalne protokoły, np. *Lspan, UDP, HDP*.

Protokół *BitTorrenta* stanowi szczególny przypadek sieci *P2P*. Zasadniczo jest to protokół udostępniania plików, polegający na centralnych serwerach zwanych *trakerami* do koordynowania użytkowników, nie tworzy on jednak sieci w tradycyjnym sensie. Zamiast tego tworzone są osobne sieci koordynujących użytkowników dla każdego zestawu plików (zwanego ***torrentem***). Nowsze rozszerzenia protokołu eliminują potrzebę scentralizowanych *trakerów*, pozwalając na korzystanie ze zdecentralizowanej sieci niezależnej od serwera do celów identyfikacji źródła, zwanej *Mainline DHT*. Użytkownicy tworzą plik indeksu zawierający metadane plików, które chcą udostępnić i przesyłają pliki indeksu na strony internetowe, gdzie są udostępniane innym osobom.

2.2. Traker

Traker to specjalny typ serwera, który pomaga w komunikacji między użytkownikami za pomocą protokołu *BitTorrent*. Podczas udostępniania plików *P2P* klient oprogramowania (na komputerze użytkownika końcowego) żąda pliku, a części żądanego pliku znajdujące się na komputerach równorzędnych są wysyłane do klienta, następnie ponownie składane w pełną kopię żądanego pliku. Serwer *trakera* śledzi, gdzie znajdują się kopie plików na komputerach równorzędnych, które są dostępne w momencie żądania klienta, i pomaga koordynować efektywną transmisję i ponowny montaż skopiowanego pliku. Klienci, którzy

już rozpoczęli pobieranie pliku, okresowo komunikują się z modułem śledzącym, aby negocjować szybszy transfer plików z nowymi urządzeniami równorzędnymi i udostępniać statystyki wydajności sieci. Po rozpoczęciu wstępnego pobierania pliku *P2P* komunikacja *P2P* może być kontynuowana bez połączenia z *trakerem*.

Od czasu stworzenia metody rozproszonej tablicy skrótów (*DHT*) dla *torrentów* bez *trackerów*, *trackery BitTorrenta* w dużej mierze stały się zbędne. Jednak wciąż są one często dołączane do *torrentów*, aby poprawić szybkość wykrywania „sąsiadów".

2.3. Plik torrent

Plik *torrent* lub inaczej *plik metainfo* to plik komputerowy zawierający metadane dotyczące plików i folderów, które mają być dystrybuowane, a także (zwykle) listę lokalizacji *trakerów*, będących komputerami pomagającymi uczestnikom systemu odnaleźć i stworzyć wydajną grupę dystrybucyjną zwaną *ławicą* lub *rojem* (*ang.* ***swarm***). Plik *torrent* nie zawiera treści plików, a jedynie informacje o nich, takie jak: nazwy, rozmiary, struktura folderów i wartości skrótu kryptograficznego do weryfikacji integralności pliku (*ang. cryptographic hash function - CHF*). Termin *torrent* może odnosić się do pliku metadanych lub do pobranych plików, w zależności od kontekstu. Plik *torrent* jest jak spis treści, ponieważ ułatwia efektywne wyszukiwanie informacji (ale nie zawiera samych informacji). Podaje on adresy komputerów na całym świecie, które mogą wysyłać części żądanego pliku. Same pliki *torrent* i metoda korzystania z plików *torrent* zostały stworzone w celu zmniejszenia obciążenia na serwerach centralnych. Za pomocą *torrentów* można pobrać małe części oryginalnego pliku z komputerów, które już go mają, zamiast głównego serwera.

Pliki *torrent* mają zwykle nazwy z rozszerzeniem *.torrent*.

Plik *torrent* jest *bencodowanym* (specjalne kodowanie protokołu *BitTorrent*) słownikiem z następującymi kluczami (klucze są uporządkowane leksykograficznie):

- *announce* – adres URL modułu śledzącego;
- *info* – odwzorowuje słownik, którego klucze zależą od tego, czy współdzielony jest jeden czy więcej plików:
    - *files* – lista słowników odpowiadających plikowi (tylko w przypadku udostępniania wielu plików). Każdy słownik ma następujące klucze:
        - *length* – rozmiar pliku w bajtach;

        - *path* – lista ciągów znaków odpowiadających nazwom podkatalogów, z których ostatni to rzeczywista nazwa pliku;
    - *length* – rozmiar pliku w bajtach (tylko przy współużytkowaniu jednego pliku);
    - *name* – sugerowana nazwa pliku, w którym plik ma zostać zapisany (jeśli jeden plik) / sugerowana nazwa katalogu, w którym pliki mają zostać zapisane (jeśli wiele plików);
    - *piece length* – liczba bajtów na sztukę. Zazwyczaj jest to $2^8\ KiB = 256\ KiB = 262{,}144\ B$;
    - *pieces* – lista haszowana: gdy zwraca 160-bitowy skrót, części będą ciągiem, którego długość jest wielokrotnością 20 bajtów. Jeśli *torrent* zawiera wiele plików, części są tworzone przez łączenie plików w kolejności, w jakiej pojawiają się w słowniku plików (wszystkie części w *torrencie* mają pełną długość, z wyjątkiem ostatniego, który może być krótszy).

Wszystkie ciągi muszą być zakodowane w *UTF-8*, z wyjątkiem kawałków zawierających dane binarne. Przykład jak mógłby wyglądać pozbawiony *bencodowania* plik *.torrent* dla jednego pliku danych zawiera listing 2.3.1.

```
{
      'announce': 'http://bttracker.debian.org:6969/announce',
      'info':
      {
          'length': 678301696,
          'name': 'debian-503-amd64-CD-1.iso',
          'piece length': 262144,
          'pieces': <binary SHA1 hashes>
      }
 }
```

Listing. 2.3.1. Przykładowa zawartość pozbawionego *bencodowania* pliku *torrent* dla obrazu *ISO Debiana*.

2.4. Szczegółowe działanie protokołu *BitTorrenta*

Protokół *BitTorrenta* umożliwia użytkownikom dołączenie do *ławicy hostów* w celu jednoczesnego przesyłania do siebie i pobierania plików, zamiast pobierania z jednego serwera źródłowego. Protokół jest alternatywą dla starszych źródeł lustrzanych i może działać skutecznie w sieciach o niższej przepustowości. Korzystając z protokołu *BitTorrent*, kilka podstawowych komputerów, może zastąpić duże serwery, jednocześnie skutecznie

dystrybuując pliki do wielu odbiorców. To niższe wykorzystanie przepustowości pomaga również zapobiegać dużym skokom ruchu w Internecie w danym obszarze, utrzymując wyższe prędkości Internetu dla wszystkich użytkowników, niezależnie od tego, czy używają protokołu *BitTorrent*.

Ci, którzy chcą pobrać plik, pobierają *torrent*, którego ich klient użyje do połączenia z *trackerem*, który ma listę adresów IP innych *seedów* (użytkownik, który posiada kompletny plik i udostępnia go innym osobom) i *peerów* (użytkownik, który w danym momencie pobiera i udostępnia dany plik) w *ławicy*. Gdy *peer* zakończy pobieranie całego pliku, może on z kolei działać jako *seed*. Rozpowszechniany plik jest podzielony na segmenty zwane kawałkami. Gdy każdy uczestnik otrzymuje nowy fragment pliku, staje się on źródłem (tego fragmentu) dla innych partnerów, uwalniając pierwotne ziarno od konieczności wysyłania tego fragmentu do każdego komputera lub użytkownika, który chce kopii.

Rozpowszechnianie pliku jest dla tych użytkowników, którzy tego chcą, przykładowo: jeden *seed* może wysyłać tylko jedną kopię pliku i ostatecznie dystrybuować go do nieograniczonej liczby *peerów*. Każdy kawałek jest chroniony hashem kryptograficznym zawartym w deskryptorze *torrenta*. Zapewnia to niezawodne wykrycie każdej modyfikacji elementu, a tym samym zapobiega zarówno przypadkowym, jak i złośliwym modyfikacjom elementów otrzymanych w innych węzłach. Jeśli węzeł zaczyna się od autentycznej kopii deskryptora *torrenta*, można zweryfikować autentyczność całego otrzymanego pliku.

Kawałki są zazwyczaj pobierane niesekwencyjnie i są porządkowane w prawidłowej kolejności przez klienta *BitTorrenta*, który monitoruje, które elementy potrzebuje, a które ma i które może przesłać do innych „sąsiadów”. Kawałki mają ten sam rozmiar podczas jednego pobierania (na przykład plik 10 MB może być przesłany jako dziesięć 1 MB lub czterdzieści 256 KB). Ze względu na naturę tego podejścia pobieranie dowolnego pliku można zatrzymać w dowolnym momencie i wznowić w późniejszym terminie, bez utraty wcześniej pobranych informacji, co z kolei sprawia, że *BitTorrent* jest szczególnie przydatny w przypadku przesyłania większych plików. Umożliwia to również klientowi wyszukiwanie dostępnych kawałków i natychmiastowe pobieranie ich, zamiast zatrzymywania pobierania oraz czekania na następny (i prawdopodobnie niedostępny) wiersz w linii, co zwykle skraca całkowity czas pobierania. To ewentualne przejście z *peera* do *seeda* określa ogólne „zdrowie” pliku (określone przez liczbę ile razy plik jest w całości dostępy).

W miarę jak więcej *seedów* dołącza do *ławicy*, prawdopodobieństwo udanego pobrania przez dowolny węzeł wzrasta. W porównaniu z tradycyjnymi schematami dystrybucji internetowej pozwala to na znaczną redukcję kosztów sprzętu i przepustowości oryginalnego dystrybutora.

BitTorrent sam w sobie nie oferuje użytkownikom anonimowości. Zazwyczaj adresy IP wszystkich *peerów* w *ławicy* można zobaczyć we własnym kliencie lub programie zapory ogniowej. Może to narazić użytkowników posiadających niezabezpieczone systemy na ataki.

3. ZAŁOŻENIA PROJEKTOWE

W niniejszym rozdziale zostaną przedstawione założenie projektowe zaimplementowanego systemu. Wymagania zaliczenia zakładały, że projekt powinien posiadać wygodny interfejs pozwalający na graficzną wizualizację aktualnego stanu systemu (z widocznym stanem poszczególnych serwerów, stanem sieci oraz przesyłanych komunikatach). Program powinien pozwalać w dowolnym momencie na wprowadzanie uszkodzeń zarówno do samych serwerów, jak również do poszczególnych połączeń (sieci) pomiędzy serwerami. W każdym momencie powinno także być możliwe wycofanie uszkodzenia, czyli stworzenie symulacji naprawy serwerów i sieci. Symulator powinien także mieć możliwość podglądu, który z serwerów posiada aktualne, a który nieaktualne dane.

W poniższych podrozdziałach przedstawiono architekturę i wykorzystane narzędzia oraz najważniejsze założenia projektowe i zaimplementowane funkcjonalności, pozwalające na spełnienie postawionych wcześniej wymagań.

3.1. Architektura i wykorzystane narzędzia.

Projekt powstał przy wykorzystaniu technologii takich jak ***Spring Framework*** oraz ***Angular 9***. Wymiana danych następuje więc z wykorzystaniem protokołu *HTTP*. Część *back-endowa* symulatora została napisana w języku ***Java*** w wersji 11, natomiast interfejs użytkownika został zaimplementowany jako aplikacja webowa z wykorzystaniem ***TypeScript***.

Projekt został w całości stworzony z wykorzystaniem środowiska *InteliJ IDEA*.

Warto zaznaczyć, że przygotowano również konfigurację dla uruchomienia projektu w obrębie jednego kontenera ***Docker***.

Część *back-end* została zaimplementowana jako projekt ***Maven***, który jest jednym budującym się modułem, ten zawiera odpowiednio pod sobą implementacje **klienta** oraz **trackera**, które również są odrębnymi dla siebie modułami. Części wspólne projektów z których korzysta zarówno aplikacja trackera jak i klienta (np.: Data Transfer Objects) zostały wydzielone do modułu **util.**

Aplikacja Angulara, która odpowiada za obsługę symulatora z poziomu wygodnego interfejsu może być uruchomiona niezależnie od części *back-endowej*.

Architektura projektu pozwala uruchomić dowolną liczbę instancji trackerów oraz klientów.

3.2. Najważniejsze założenia projektowe i zaimplementowane funkcjonalności.

- Najważniejszym założeniem, według którego implementowany był symulator to sposób wymiany danych jak najbardziej przybliżony do realnego działania protokołu BitTorrent.
- Kolejnym założeniem było dostarczenie możliwości użytkownikowi wprowadzania przerw w transmisji, włączania i wyłączania dostępności plików dla innych uczestników a także symulowania opóźnień.
- Zgodnie z założeniem tematu projektu, symulator obsługuje 11 instancji klientów, oraz 3 trakerów.
- Każdy z klientów może zrejestrować własny plik z danymi, który może być rozprzestrzeniany po sieci.
- Każdy klient po zarejestrowaniu swojego pliku otrzymuje wygenerowany od trackerów plik .torrent, który może zostać wykorzystany przez innych klientów.
- Każdy klient po otwarciu pliku .torrent otrzymuje informacje od trackerów jaki inny klient lub klienci posiadają dany plik (w całości lub jego części).
- W trakcie pobierania, inny klient może zablokować (zasymulować usunięcie) dostęp do pliku (inni klienci nie mogą go pobrać).
- Interfejs pozawala m.in. na podgląd:
    - Listy obecnych klientów (uruchomionych instancji),
    - Parametrów pobierania wybranego klienta,
    - Postępu pobierania pliku,
    - Obecnego statusu, rozmiaru oraz pliku .torrent (ID) z jakiego pochodzą dane,
    - Statusu pobrania każdej z części pliku oraz z jakiego IP pochodzi (klienta),

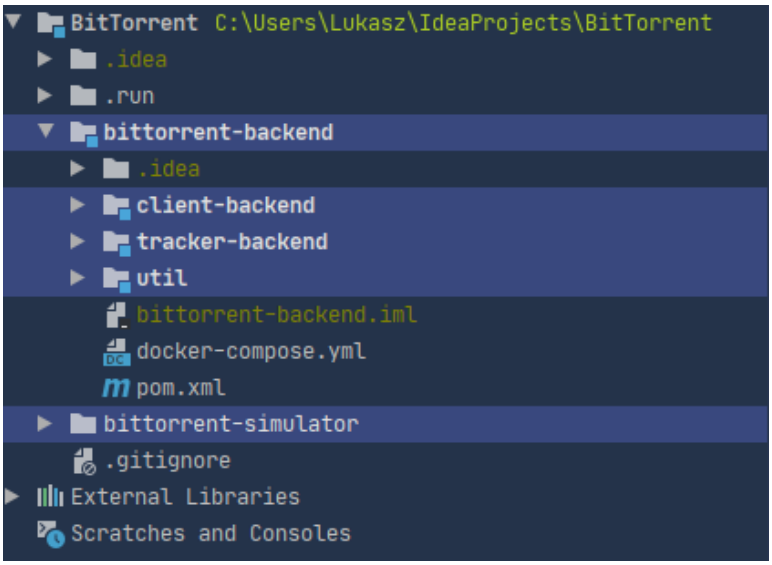
- Kompletny plik może składać się z części, z których każda została pobrana od innych klientów, którzy dane fragmenty tego pliku posiadali.
- Każdemu klientowi można ustawić symulowane opóźnienie pobierania pojedynczej części pliku.
- Dla każdego klienta wyświetlana jest lista plików (jeśli zarejestrował lub pobrał). Na liście tej klient może zablokować lub odblokować dostęp dla innych klientów.
- Jeżeli klient poprzez plik .torrent będzie próbował odwołać się za pośrednictwem trackera do pobrania danych, których żaden klient nie posiada, dostanie stosowną informację.

4. STRUKTURA PROJEKTU

Poniższe rozdziały opisują strukturę projektu dla poszczególnych jego części z wyszczególnieniem: ogólnej struktury projektu, struktury modułu klienta, struktury modułu *trakera*, struktury modułu *back-endowego* i struktury modułu *front-endowego*.

4.1. Ogólna struktura projektu.

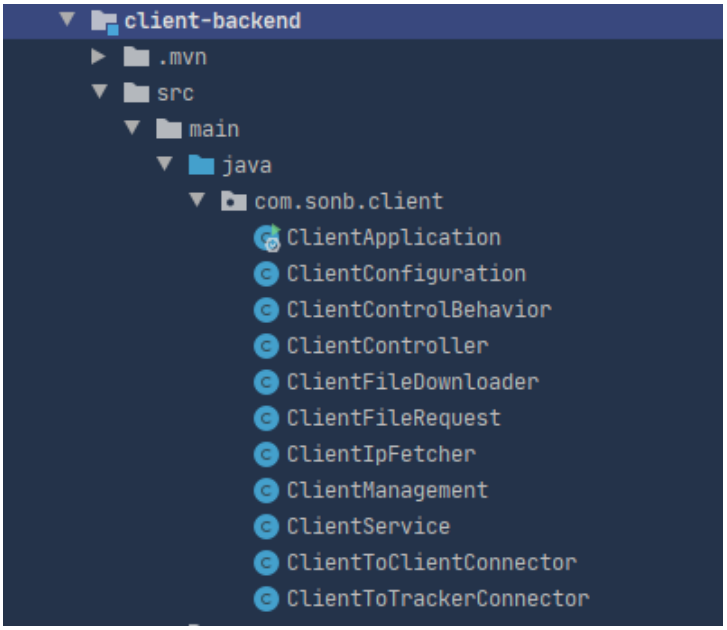
Jak wspomniano wyżej, projekt został podzielony na moduły podprojektowe zawierające osobno część *back-end* oraz *front-end*. Poniżej zaprezentowano ogólny podział w strukturze plików projektu z wyszczególnieniem instancji *klienta* oraz *trakera*.



Rys. 4.1.1. Ogólna struktura projektu.

4.2. Struktura modułu Klienta.

W strukturze klienta znajduje się 11 klas, które prezentują się następująco. Przeznaczenie każdej z klasy zostało krótko opisane w kolejnych podpunktach.



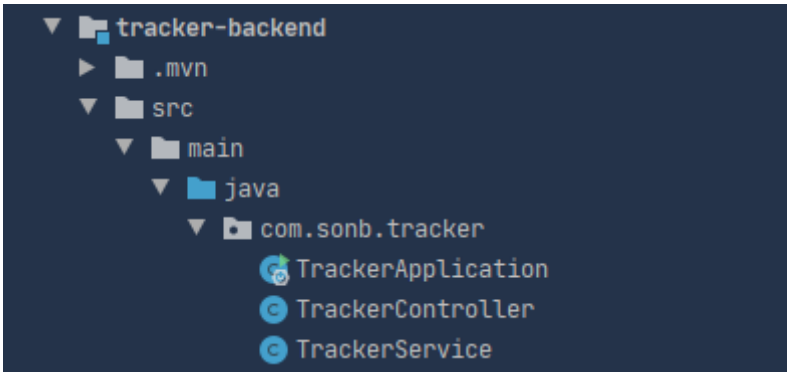
Rys. 4.2.1. Struktura modelu klienta.

- **ClientApplication –** Główna klasa startowa projektu Spring Framework.
- **ClientConfiguration –** Klasa definiująca konfigurację każdej z instancji (np.: port na jakim została uruchomiona). Klasa pozwala m.in. odczytać IP instancji klienta, czy wspomóc połączenie klienta zarówno z trackerami oraz innymi klientami.
- **ClientControlBehaviour –** Klasa stworzona w celu oddelegowania sterowaniem przepływu danych takimi jak maksymalna ilość prób pobrania danej części pliku od tego samego klienta czy symulowanego czasu opóźnienia pobierania.
- **ClientController –** Klasa obsługująca zapytania HTTP i oddelegowująca realizację żądań do Service'ów.
- **ClientFileDownloade –** Klasa realizująca pobieranie plików, z wyszczególnieniem na pobieranie w wątkach odrębnych części pliku. Klasa obsługuje sprawdzanie oraz ponawianie pobierania pliku w razie potrzeby.
- **ClientFileRequest –** Modelowa klasa requestu dla obiektu pliku rejestrowanego przez klienta.
- **ClientIpFetcher –** Klasa oferująca możliwość łatwego pobrania i sprawdzenia IP instancji klienta.

- **ClientManagement** – Klasa wybierająca klienta źródłowego.
- **ClientService** – Implementacja najważniejszych elementów działania aplikacji klienta. Obsługa zapisu, tworzenia plików, operacje na statusach oraz blokowaniu i odblokowania dostępu do pliku dla innych klientów.
- **ClientToClientConnector –** obsługa nawiązania połączeń z innymi klientami w celu pobrania pliku.
- **ClientToTrackerConnector –** obsługa nawiązywania połączeń z trakerami: podczas rejestracji pliku, usuwania oraz przywracania dostępu pliku u danego klienta.

4.3. Struktura modułu Trakera.

W strukturze klienta znajdują się 3 klasy, które prezentują się następująco. Przeznaczenie każdej z klasy zostało krótko opisane w kolejnych podpunktach. Struktura aplikacji trackera została zredukowana z racji wydzielenia niektórych elementów do kolejnego modułu, który zostanie opisany w kolejnym punkcie.

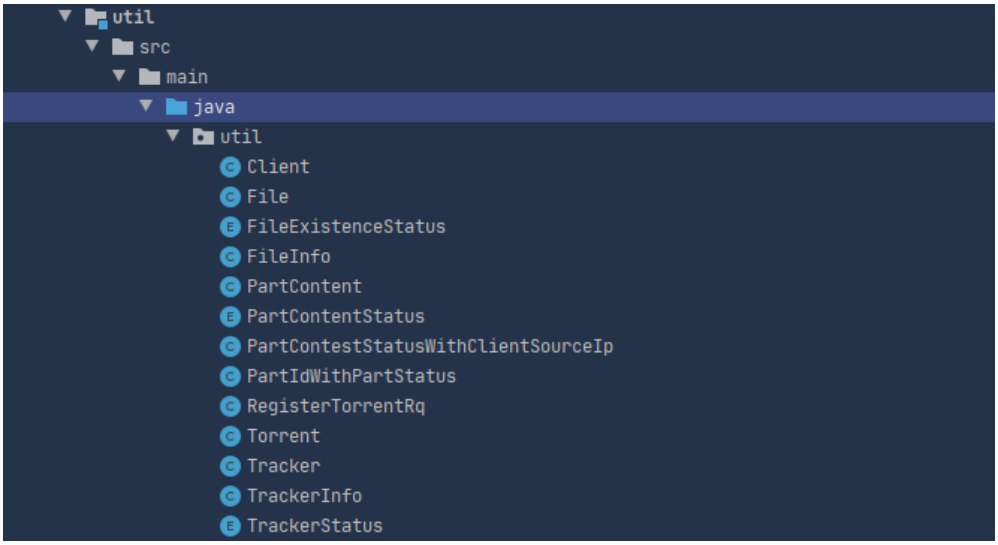


Rys. 4.3.1. Struktura modelu trakera.

- **TrackerApplication** – Główna klasa startowa projektu Spring Framework.
- **TrackerController** – Klasa obsługująca zapytania HTTP i oddelegowująca realizację żądań do Service'ów.
- **ClientService** – Implementacja najważniejszych elementów działania aplikacji trackera. Obsługa rejestracji, tworzenia plików, operacje na statusach oraz blokowaniu i odblokowania dostępu do pliku dla innych klientów. Tracker odpowiedzialny jest tutaj za wygenerowanie pliku .torrent oraz odnalezieniu klientów w momencie zgłoszenia tego pliku przez klienta, który chce dane pobrać.

4.4. Struktura modułu Util.

Jak zostało wcześniej wspomniane moduł dostarcza współdzielonych elementów dla każdego z modułów części *back-endowej*. Głównie są to modele klas obiektów wymiany danych (Data Transfer Objects, DTO).



Rys. 4.4.1. Struktura modelu util.

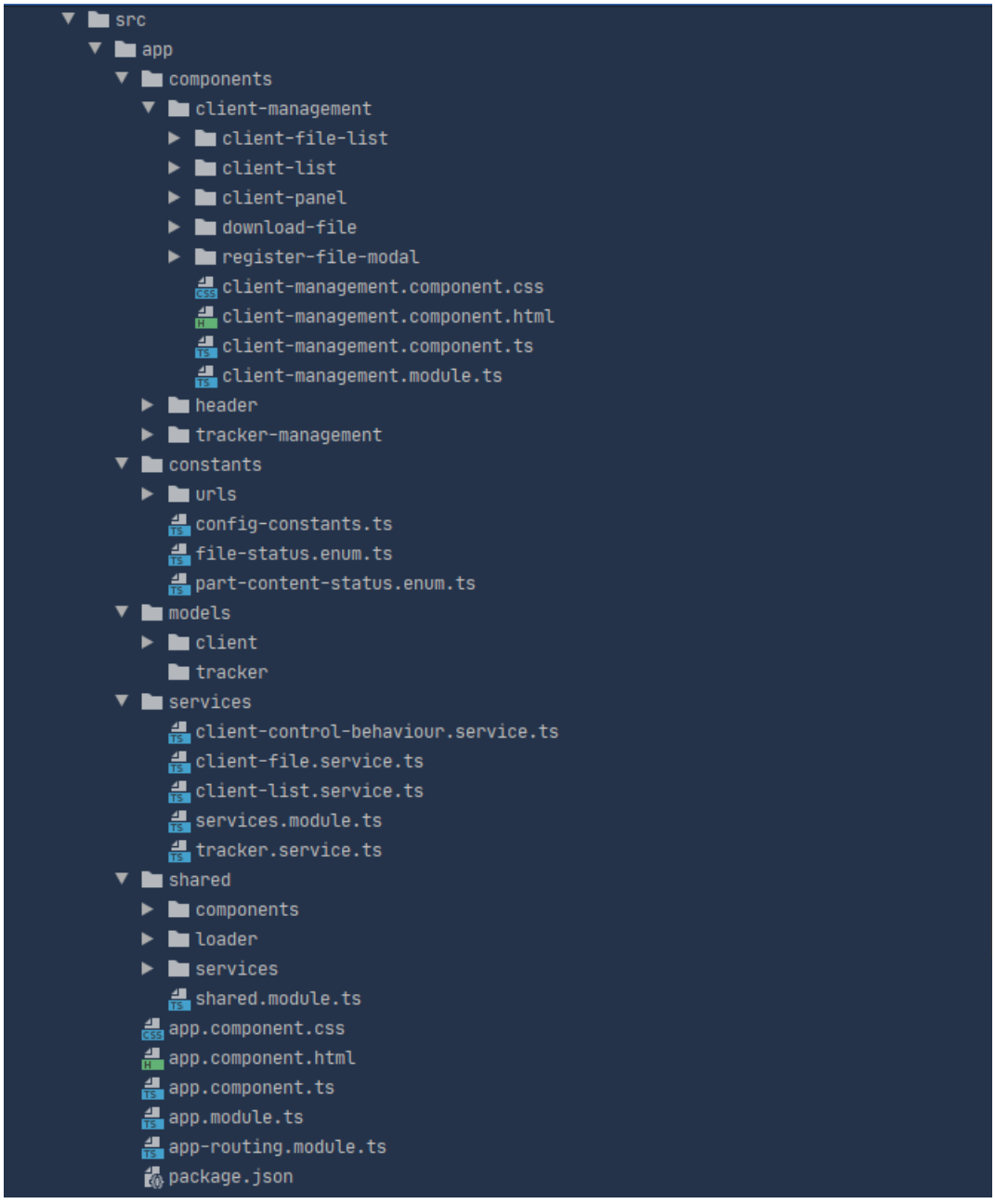
Przeznaczenie każdej z klas opisuje się następująco:

- **Client** – Struktura informacji o Kliencie,
- **File** – Struktura informacji o pliku danych, zawiera nazwę, rozmiar (ilość części), status każdej z części oraz ogólny status pliku.
- **FileExistenceStatus** – Enum reprezenujący możliwe statusy pliku.
- **FileInfo** – Struktura z informacjami dla pobranego pliku, zawiera m.in. informacje od jakiego klienta pochodzi dana część pliku oraz na żądanie jakiego torrenta (o jakim ID) został pobrany.
- **PartContent** – Informacja zawarta w FileInfo, reprezentuje obiekt ze statusem, danymi, oraz IP źródłowego klienta.
- **PartContentStatus** – Enum reprezentujący możliwe statusy każdej z części pliku.
- **PartContentStatusWithClientSourceIp** – Łączy pojedynczy part z IP źródłowego klienta.
- **PartIdWithPartStatus** – Łączy ID części pliku ze statusem.
- **RegisterTorrentRq** – Reprezentuje model żądania o informacje o pliku, jakiego żąda do pobrania klient.

- **Torrent** – Reprezenuje model zawartości pliku .torrent.
- **TrackerInfo** – Reprezentuje informacje o trackerze takie jak IP oraz jego status.
- **TrackerStatus** – Enum reprezentujący możliwe stany trackera.

4.5. Struktura projektu Angularowego w roli interfejsu dla symulatora.

Struktura aplikacji obsługującej interfejs została zaprezentowana na zrzucie ekranu umieszczonym poniżej.



Rys. 4.5.1. Struktura plików projektu interfejsu symulatora jako aplikacji webowej.

Aplikacja została podzielona na komponenty wizualne, których działanie i prezentacja wizualna zaimplementowane są w katalogu **components**. Wyróżnić można w nim następujące komponenty:

- Lista plików aktualnie wybranego klienta,
- Lista dostępnych (uruchomionych instancji) klientów,
- Widok panelu klienta,
- Modal pobierania pliku (wysłania pliku torrent)
- Modal rejestrowania pliku (tworzenia nowego i otrzymania pliku torrent)
- Główny komponent **ClientManagement** odpowiedzialny za łączenie komunikacji powyższych komponentów w obrębie obsługi klienta.

W katalogu **constants** znajdują się zestawy możliwych adresów IP klientów oraz trackertów, po za nimi w katalogu umieszczono typy wyliczeniowe statusów pliku, jego poszczególnych części oraz stałe wartości konfiguracyjne.

W katalogu **models** zawarto typy interfejsy obiektów wymiany danych (DTO).

Katalog **services** to implementacja wymiany danych dla poszczególnych aspektów za pomocą protokołu HTTP.

W katalogu **shared** umieszczono elementy wykorzystywane wielokrotnie w różnych miejscach aplikacji (np.: komponent reprezentujący ładowanie podczas pobierania danych czy service do budowania okien dialogowych).

5. KONFIGURACJA PROEJKTU I URUCHOMIENIE.

Aktualna wersja kodu źródłowego znajduje się na repozytorium GitHub pod adresem:

https://github.com/LukaszZachariasz/BitTorrent (branch **develop**).

Projekt wystarczy sklonować komendą: ***git clone <adres repozytorium>***

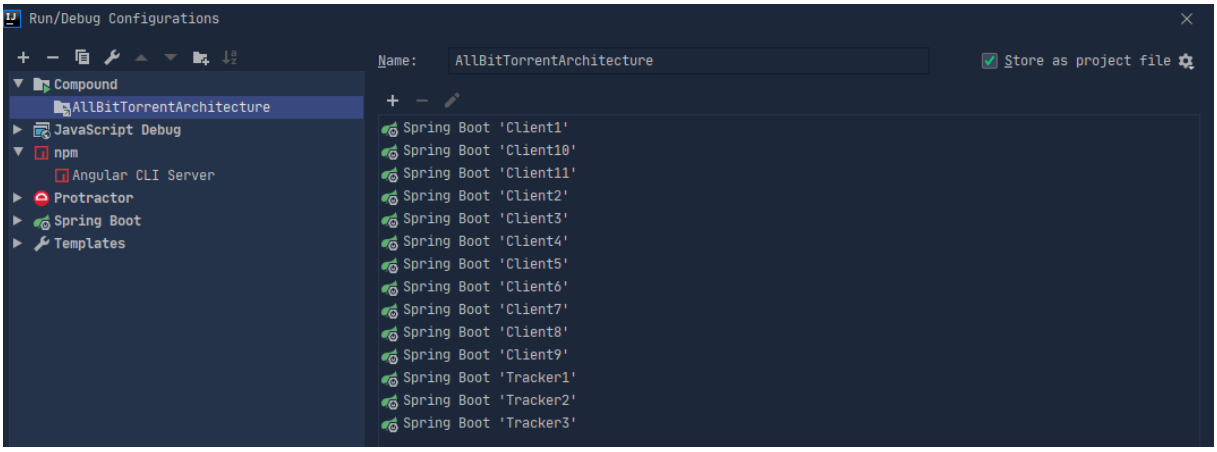
Następnie utworzony folder należy otworzyć w środowisku InteliJ IDEA i zaimportować jako projekt ***Maven***.

Do uruchomienia aplikacji webowej wymagane jest posiadanie zainstalowanego środowiska Node.js oraz Angular CLI. Aplikację uruchamiamy z poziomu katalogu bittorent-simulator poprzez komendy:

npm i – komenda zainstaluje wymagane zależności (*node_modules*),

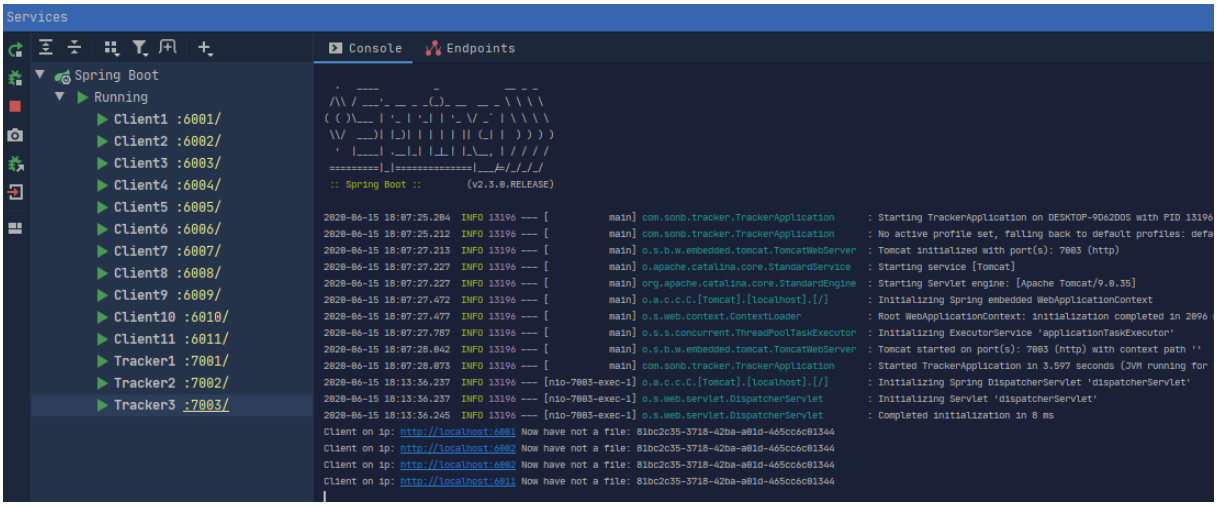
ng serve – komenda uruchomi budowanie aplikacji.

Dla uruchomienia kompletu instancji został przygotowany skonfigurowany plik w katalogu **.run,** który pozwala uruchomić wymaganą listę instancji, dla których skonfigurowano parametry startowe w postaci numeru portu: Dla **trackerów**: **70XX**, dla **klientów: 60XX**, gdzie: *XX – numer instancji 1-11*.



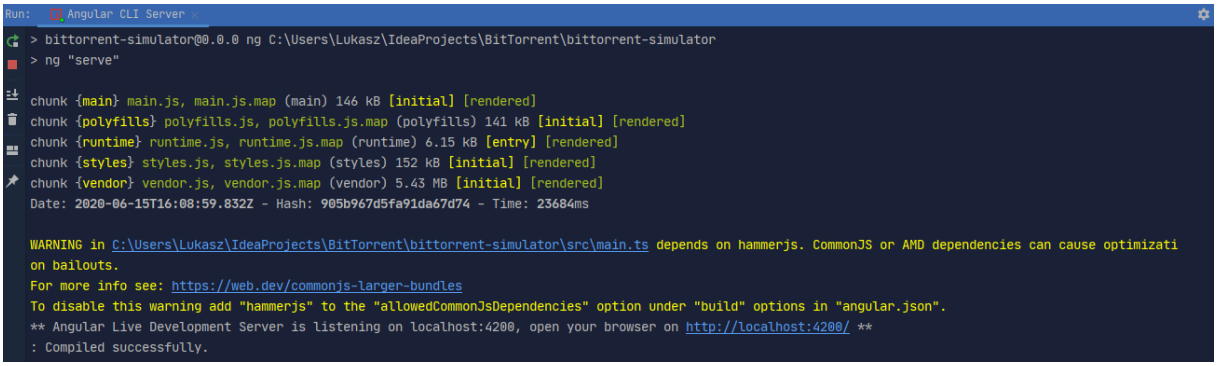
Rys. 5.1. Konfiguracja projektu.

Poniższy zrzut ekranu prezentuje stan uruchomionych instancji oraz numery portów.



Rys. 5.2. Stan uruchomionych instancji oraz numery portów.

Poniższy zrzut ekranu prezentuje stan uruchomionej aplikacji Angulara, dla której nie jest wymagana konfiguracja uruchomienia z wyjątkiem wymienionych wcześniej komend oraz posiadania zainstalowanego środowiska Node.js.

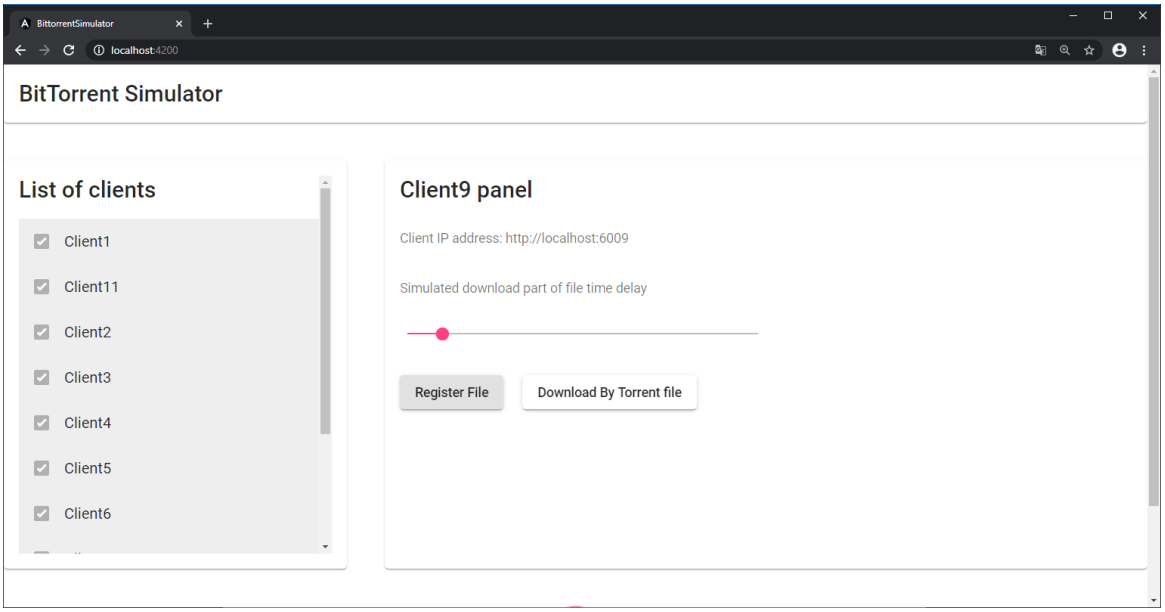


Rys. 5.3. Stan uruchomionych aplikacji Angulara.

6. PREZENTACJA I TESTOWANE SYMULATORA

Dostęp do lokalnie uruchomionego interfejsu otrzymujemy pod adresem http://localhost:4200.

Po wejściu i wcześniejszym uruchomieniu instancji części *back-end* otrzymujemy okno obsługi symulatora. Na zaprezentowanym przykładowym zrzucie ekranu uruchomiono podgląd Klienta numer 9.



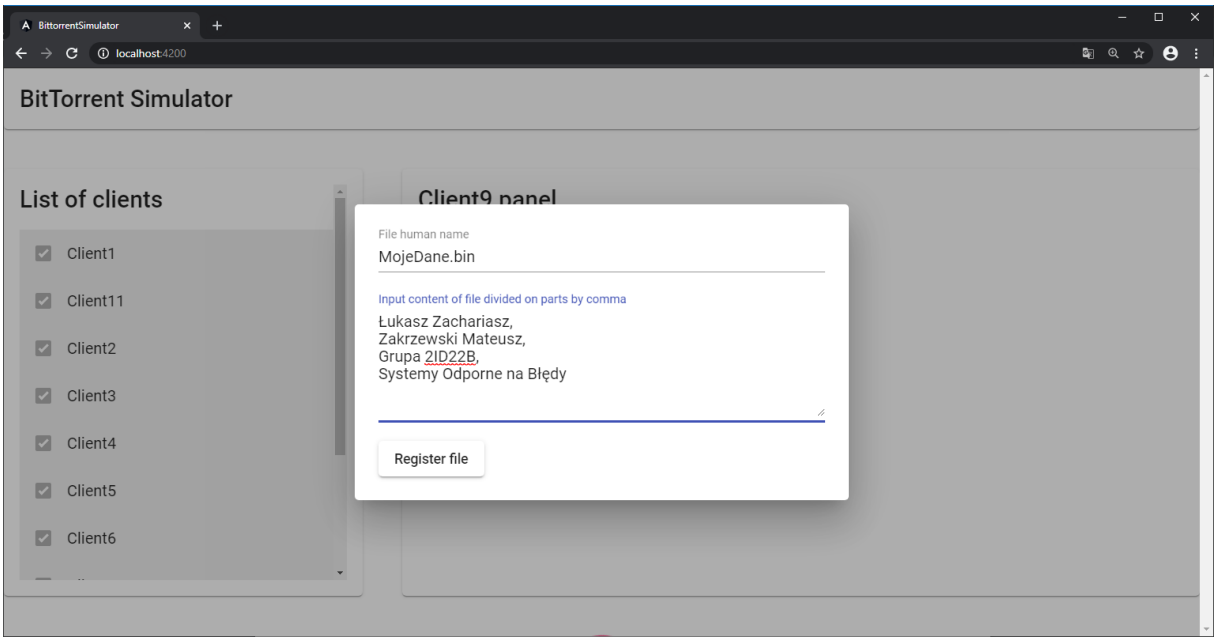
Rys. 6.1. Podgląd interfejsu klienta nr. 9.

Na widoku panelu klienta zgodnie z założeniami obsługujący może zarejestrować nowy plik lub pobrać nowy plik poprzez wygenerowany **.torrent** drogą dodawania zawartości przez innego klienta. Poniżej panelu znajduje się komponent listy plików, który podczas braku zawartości wyświetla ładowanie i komunikat o oczekiwaniu na pojawienie się pobranych / zarejestrowanych plików.

6.1. Rejestracja pliku z danymi i generowanie pliku .torrent.

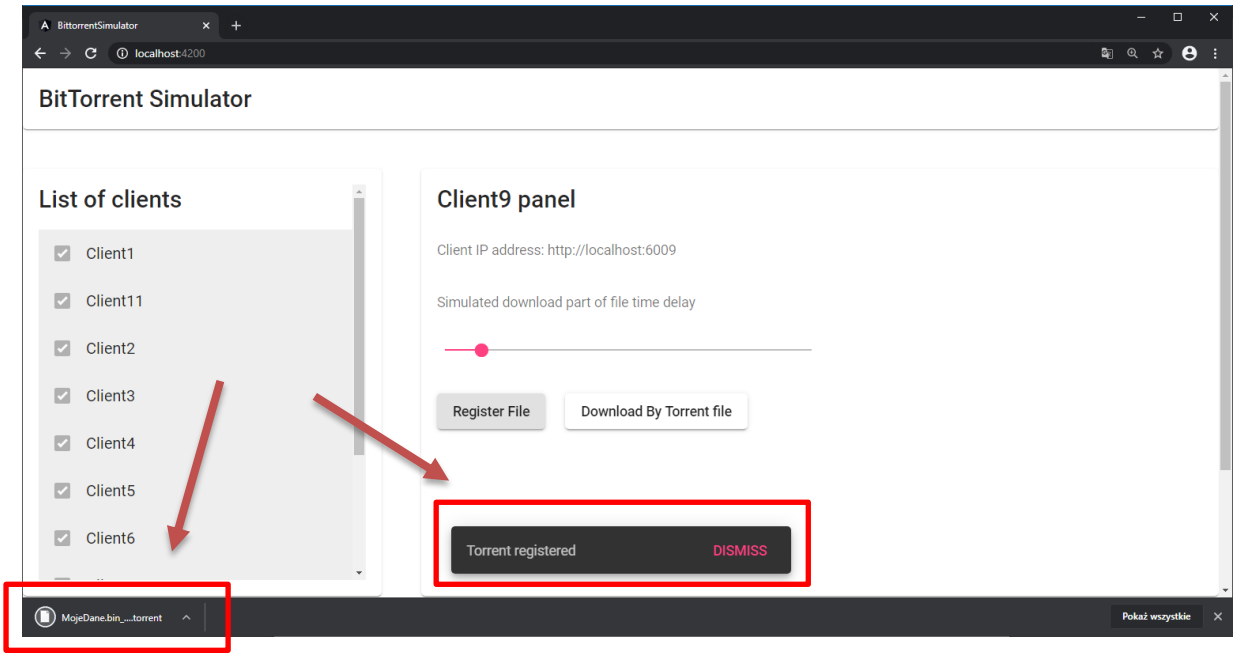
Po kliknięciu w przycisk Register File wyświetlony zostaje modal przedstawiony na kolejnym zrzucie ekranu. Posiada dwa pola tekstowe. W pierwszym z nich podajemy nazwę pliku, który będzie rozprzestrzeniany po sieci gdy inni klienci zażądają jego pobrania. W drugim polu użytkownik może zasymulować zawartość pliku – na potrzeby symulacji jest to ciąg tekstowy oddzielony przecinkami. Każde oddzielone przecinkiem wyrażenie symuluje część pliku.

Dla testu uzupełniono pola następującą zawartością:



Rys. 6.1.1. Podgląd interfejsu rejestracji pliku.

Po podaniu danych i kliknięciu Register file na poziomie modalu zostanie po pomyślnym zarejestrowaniu pliku u trackerów wygenerowany plik **.torrent.** O fakcie tym rejestrujący zostanie poinformowany.



Rys. 6.1.2. Wygenerowanie pliku .torrent.

Zawartość pliku .torrent można otworzyć dla przykładu w dowolnym edytorze tekstowym, nazwa wygenerowanego pliku to:

MojeDane.bin_aa470437-1b7a-489c-a55e-0bba63d829b0.torrent

Jego zawartość prezentuje się następująco:

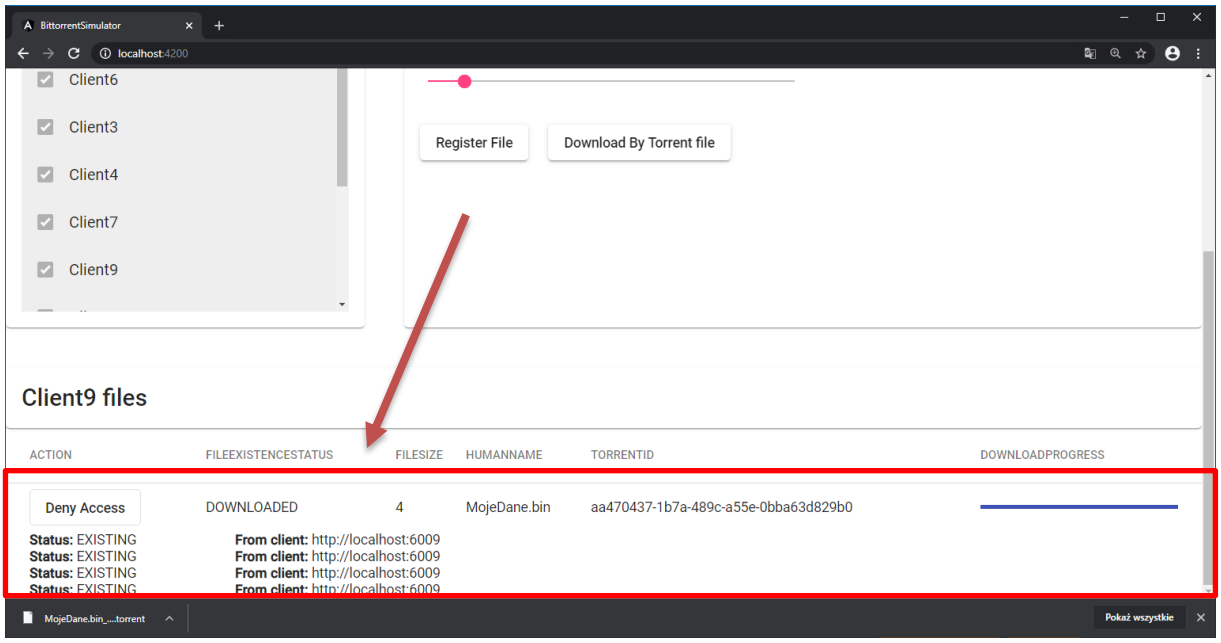
```
{
  "fileId": "aa470437-1b7a-489c-a55e-0bba63d829b0",
  "humanName": "MojeDane.bin",
  "pieceNumbers": 4,
  "trackerIps": [
    "http://localhost:7001",
    "http://localhost:7002",
    "http://localhost:7003"
  ]
}
```

Listing. 6.1.1. Zawartość utworzonego pliku .torrent.

Jak prezentuj zawartość, plik zawiera informację takie jak:

- ID pliku torrent,
- Nazwę pliku,
- Ilość części (oddzielona zawartość znakiem przecinka),
- Tablice IP trackerów, u których zarejestrowano plik.

Po utworzeniu pliku na liście plików klienta natychmiast pojawia się lista z obecnie posiadanym plikiem, jego statusem, oraz statusem jego części, również źródło wskazuje na samego siebie.

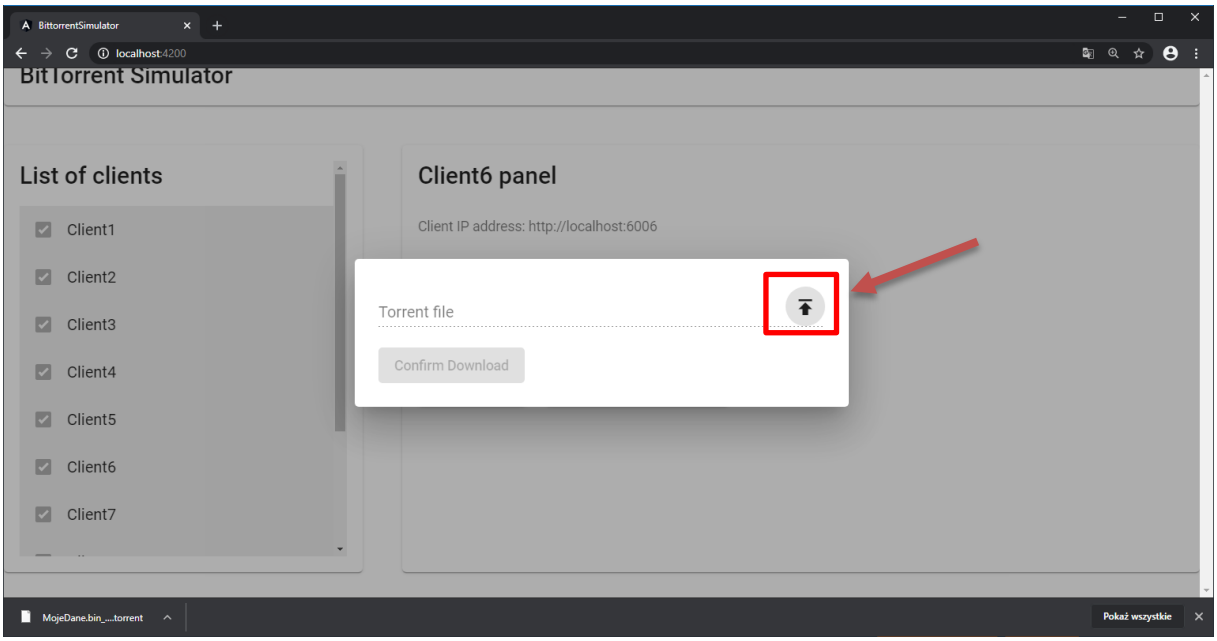


Rys. 6.1.3. Pobieranie za pomocą pliku torrent.

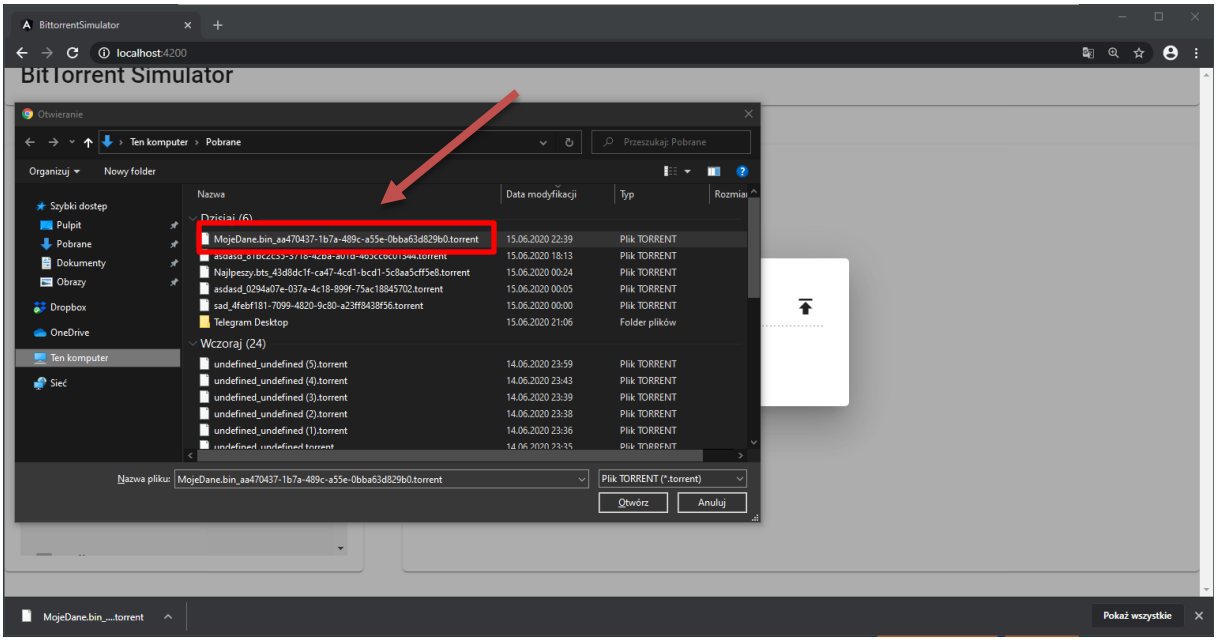
6.2. Pobieranie pliku przez innego klienta podając plik .torrent.

W tym momencie można przejść do panelu innego dostępnego klienta, dla przykładu: Klient numer 6. Wybieramy następnie przycisk Download by Torrent file. Możemy również zasymulować zmienną prędkość pobierania każdej z części, wartości można regulować w zakresie od 1 do 10 sekund na część pliku.

Po kliknięciu wspomnianego wyżej przycisku otwiera się modal, w którym należy wrzucić wygenerowany wcześniej plik **torrent**.

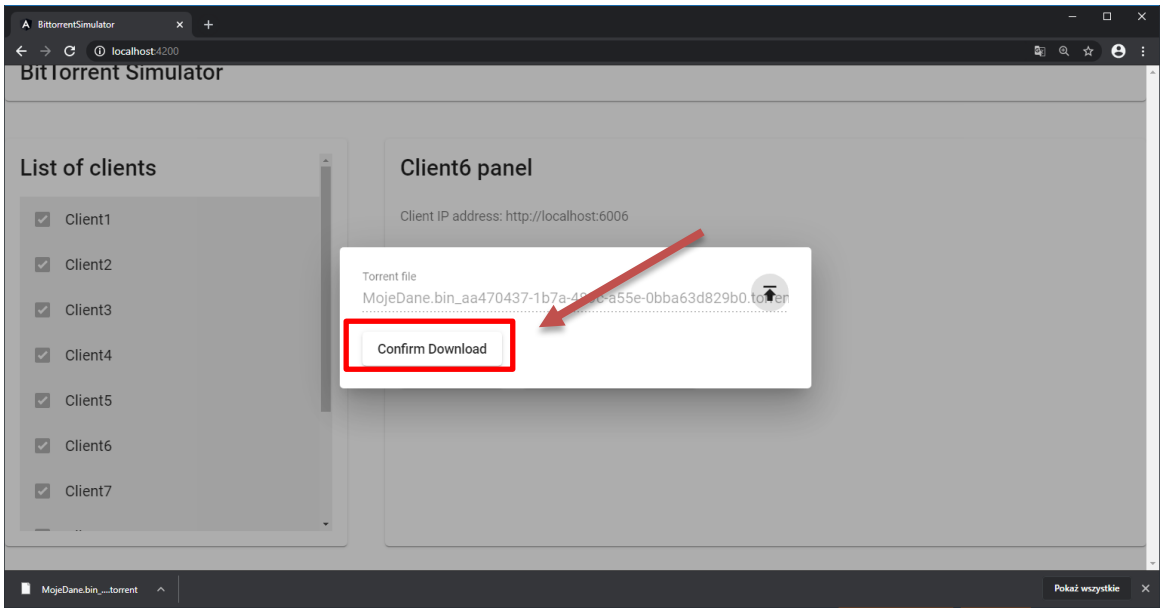


Rys. 6.2.1. Upload pliku torrent w celu pobrania pliku z danymi.



Rys. 6.2.2. Wybranie pliku torrent.

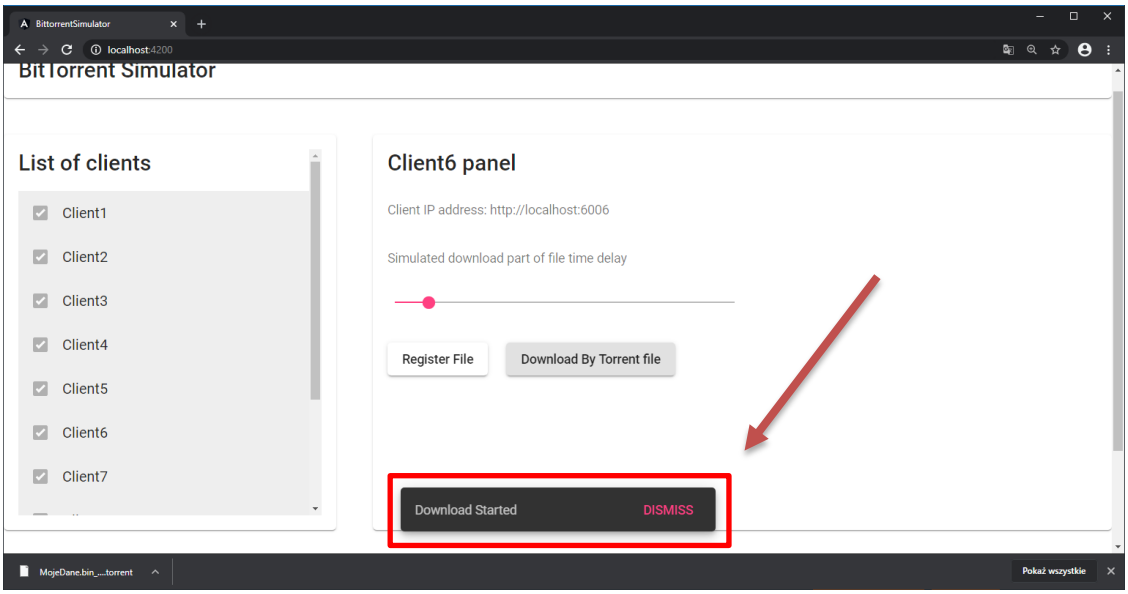
Na kolejnym etapie należy potwierdzić pobieranie pliku.



Rys. 6.2.3. Potwierdzenie pobrania pliku danych na podstawie pliku torrent.

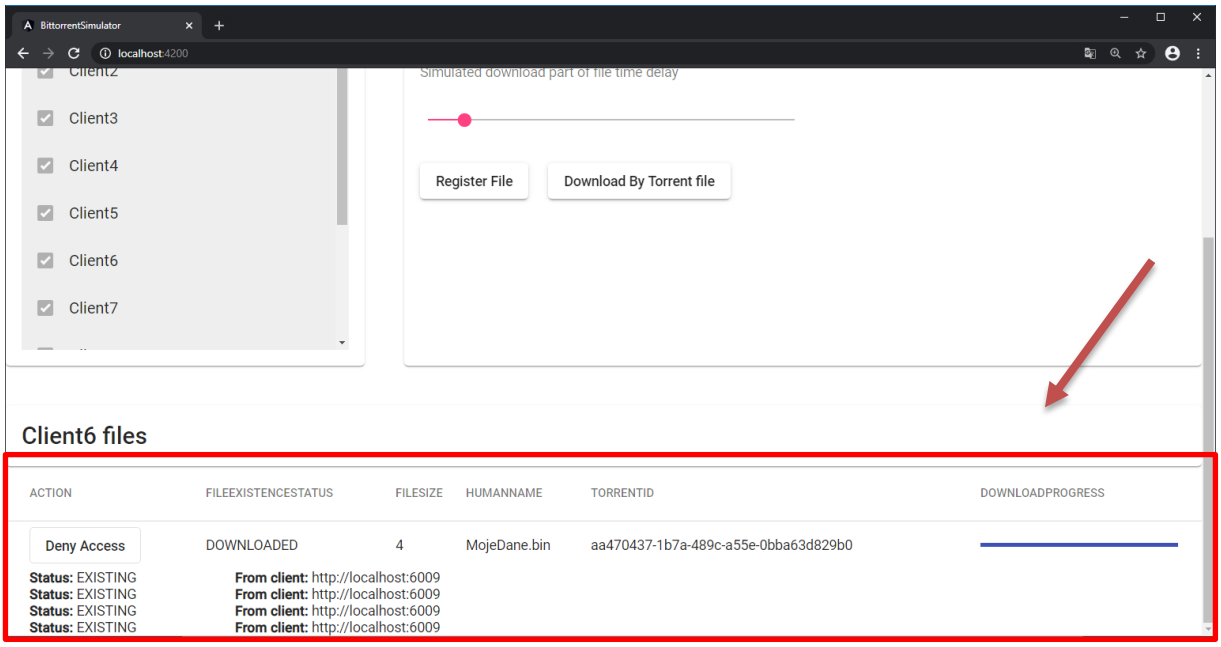
W tym momencie Klient numer 6 prosi trackery o listę IP Klientów, którzy posiadają plik, o którym informację przesłał do nich w postaci pliku torrent.

Po znalezieniu przynajmniej jednego klienta, który posiada plik lub jego część, zostaje odesłana informacje i Klient numer 6 podejmuje próbę nawiązania połączenia z innym Klientem, w tym przypadku Klientem numer 9, ponieważ tylko on posiada w tym momencie ten plik.



Rys. 6.2.4. Komunikat rozpoczęcia pobierania danych wskazywanych przez plik torrent.

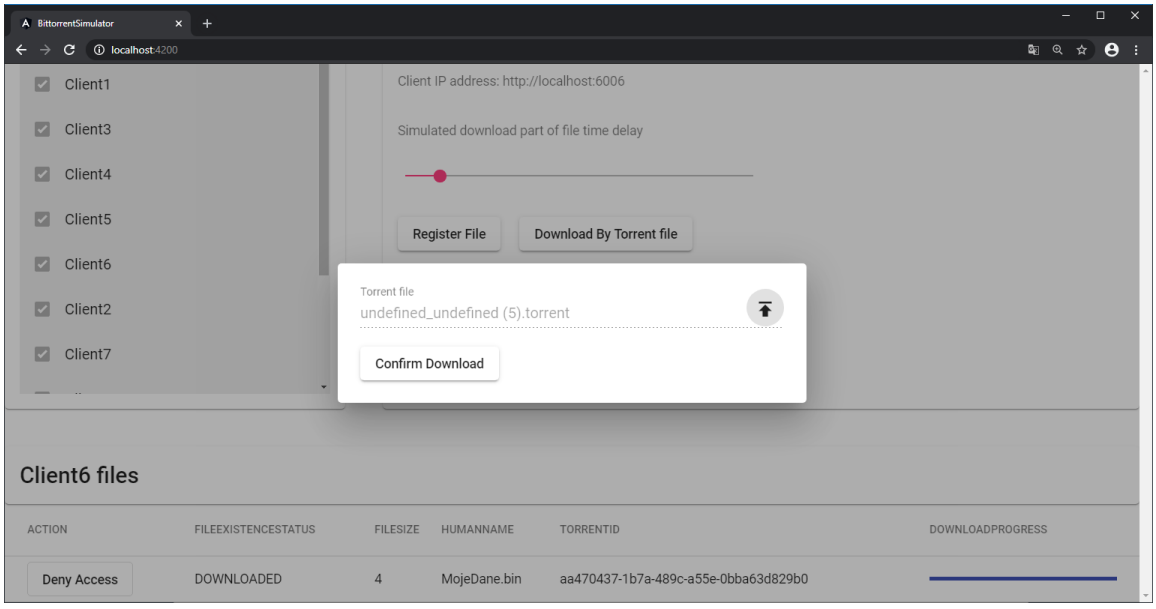
Pobieranie rozpoczęło się, co zostało zarejestrowane na liście plików Klienta numer 6.



Rys. 6.2.5. Stan pobierania pliku.

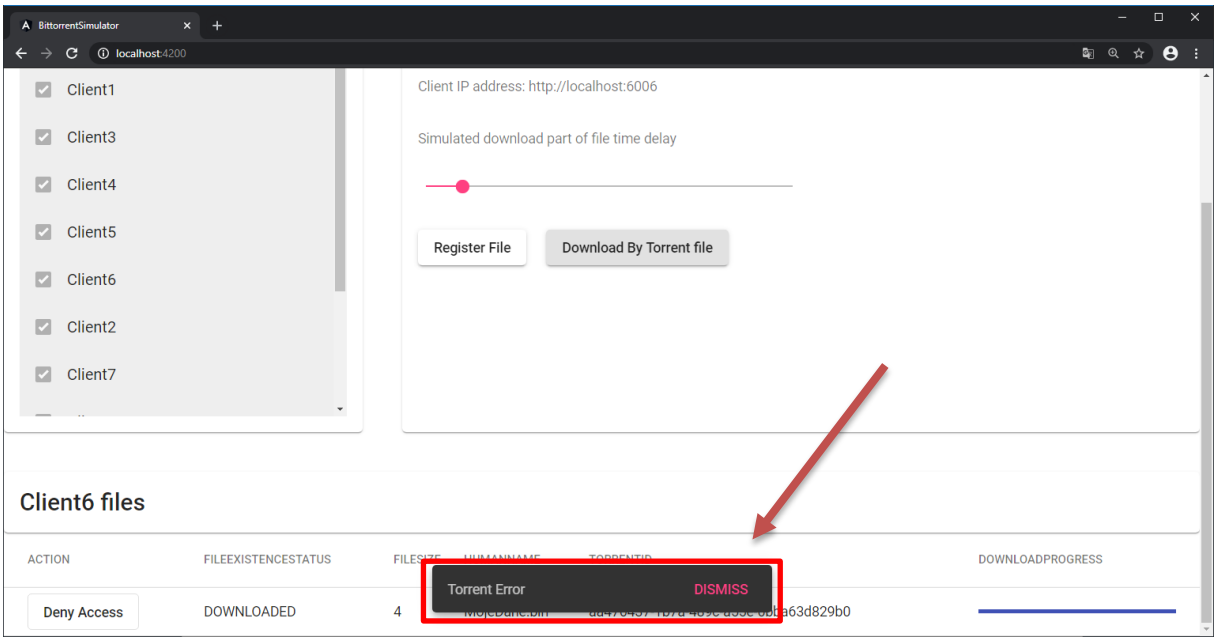
Na liście widać z jakiego źródła pochodzi każda z części pobranego pliku.

W momencie gdy wrzucony zostanie plik torrent, dla którego informacji trackerzy nie znajdą właściciela wówczas wyświetlony zostanie obsłużony błąd operacji. Dla przykładu wrzucono plik o nazwie *undefined*.



Rys. 6.2.6. Próba pobrania pliku z danymi, na którego torrent wskazuje lecz żaden tracker nie znajduje właściciela.

Na następnym zrzucie ekranu zaprezentowano komunikat o błędzie.



Rys. 6.2.7. Komunikat o błędzie.

6.3. Pobieranie pliku przez innych klientów podając plik .torrent i otrzymywanie części pliku z różnych źródeł.

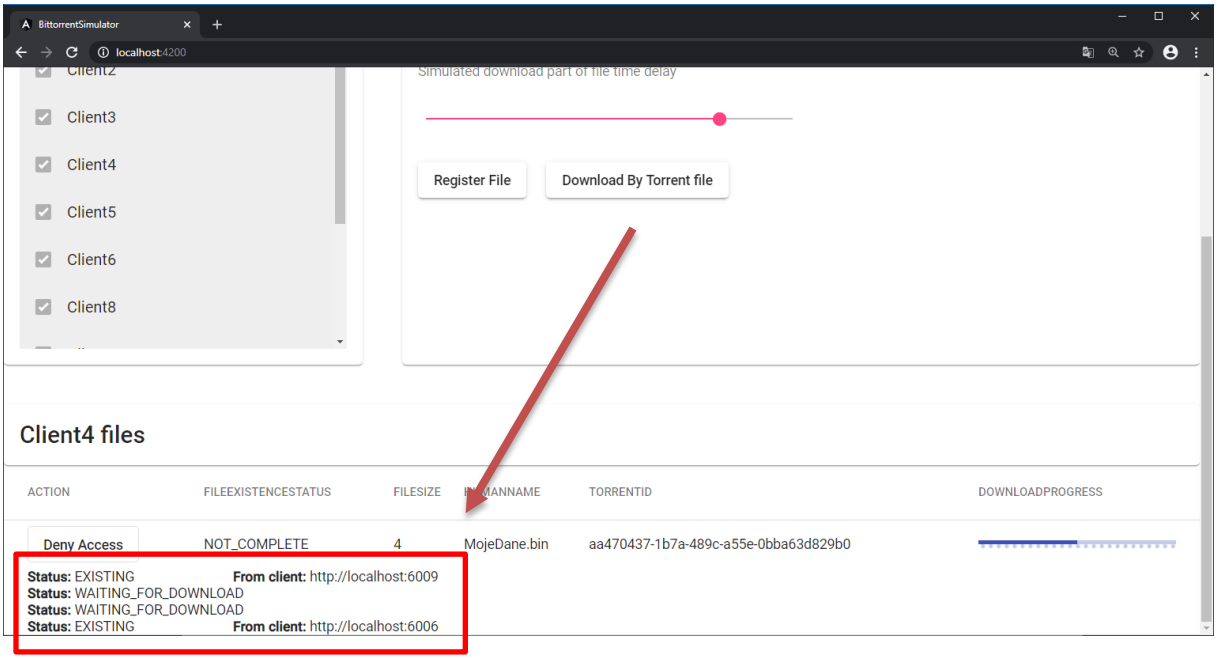
Ten scenariusz przedstawia najistotniejsze cechy protokołu, czyli równoczesne pobieranie różnych części plików z różnych źródeł. W tym momencie już dwóch Klientów posiada widoczny na zrzutach ekranu plik, jednak każda z części posiada tego samego właściciela jako źródło.

Teraz wywołajmy pobieranie tego samego pliku z przedstawionym wcześniej plikiem torrent u Kolejnych dwóch klientów. Dla przykładu: Klient numer 4 oraz Klient numer 5.

Najpierw pobieranie zostało wywołane u Klienta numer 4, dodatkowo wybrano dłuższy czas pobierania każdej z części (zaprezentowano na zrzucie ekranu). Następnie uruchomiono pobieranie u Klienta numer 5.

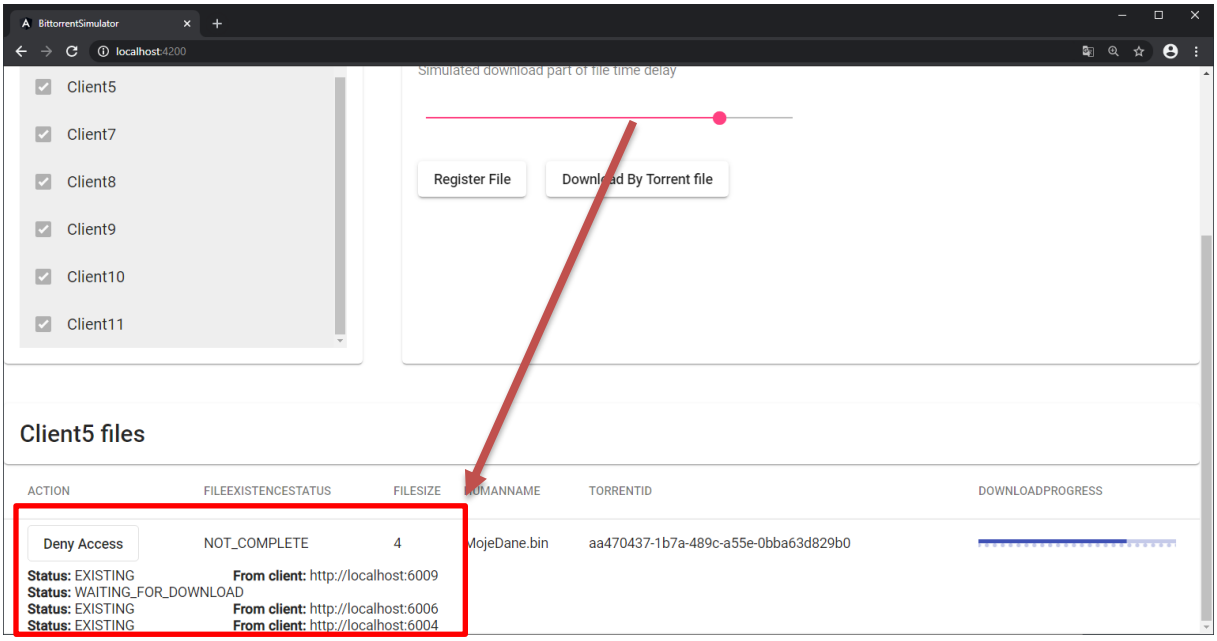
W momencie gdy Klient numer 4 jako pierwszy pobrał pewne części pliku, okazało się, że po opóźnionym uruchomieniu pobierania Klienta numer 5 pobrał on również pobraną w międzyczasie jedną z części od Klienta numer 4, co jest sytuacją jak najbardziej poprawną i oczekiwaną w wykorzystaniu protokołu BitTorrent.

Na załączonych zrzutach ekranu można zauważyć statusy, źródłowe IP dla każdej z części i postęp trwającego pobierania plików. **Klient numer 4 w trakcie pobierania pliku** został zaprezentowany na poniższym rysunku.



Rys. 6.3.1. Klient numer 4 w trakcie pobierania pliku.

Przy **kliencie numer 5 w trakcie pobierania pliku** można zaobserwować część pobraną od klienta wyżej, który jeszcze nadal nie posiada kompletnego pliku, współdzieli natomiast te pobrane.



Rys. 6.3.2. Klient numer 5 w trakcie pobierania pliku.

6.4. Blokowanie dostępu pobierania pliku lub jego części.

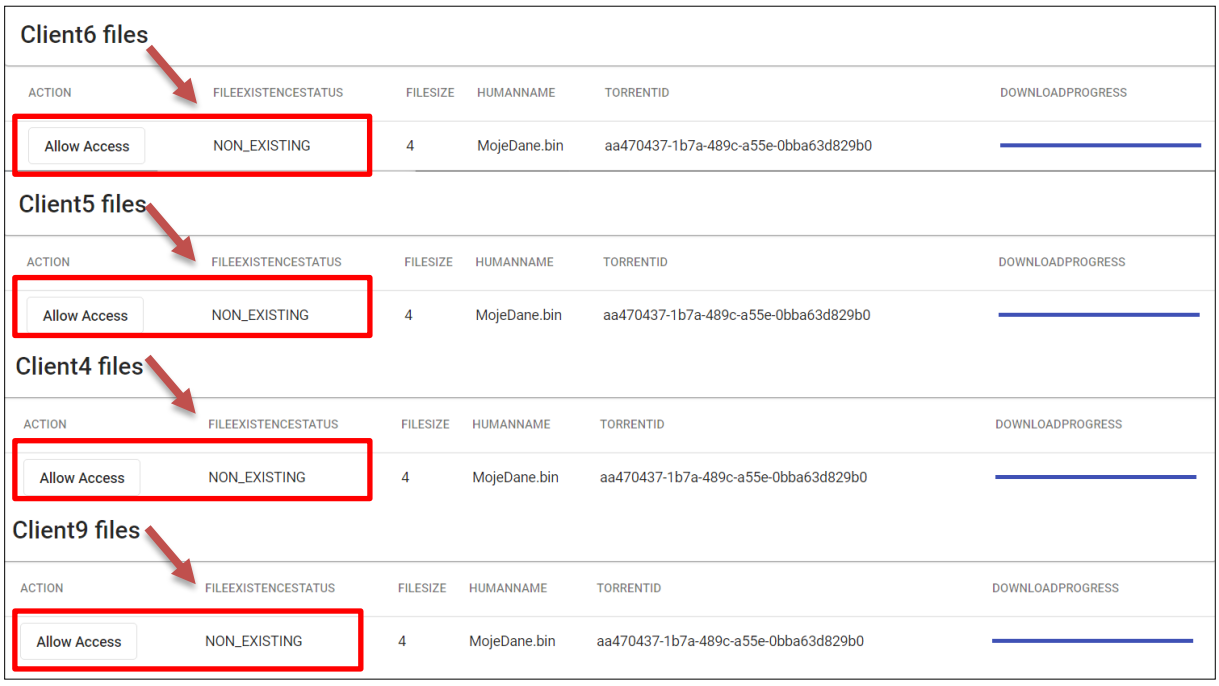
Kolejnym założeniem, które zostało zrealizowane jest możliwość wprowadzania usterki w postaci wywłaszczenia posiadania danego pliku przez konkretnego klienta w momencie gdy inny klient będzie próbował znaleźć go w sieci.

Jak zaprezentowano na poprzednich zrzutach ekranu, na liście plików pobranych / zarejestrowanych dla każdego klienta istnieje opcja wprowadzenia symulowanej usterki pod przyciskiem **Deny Access**. Powoduje ona „ukrycie" widoczności posiadania pliku dla innych klientów.

Oznacza to, że jeśli tylko jeden klient posiada dany plik w momencie gdy tylko jeden klient pobrał 50% jego zawartości, to ten w momencie zablokowania pliku u źródła przejdzie w stan oczekiwania aż dostęp do pliku zostanie odblokowany (lub inny klient nie wejdzie w jego posiadanie).

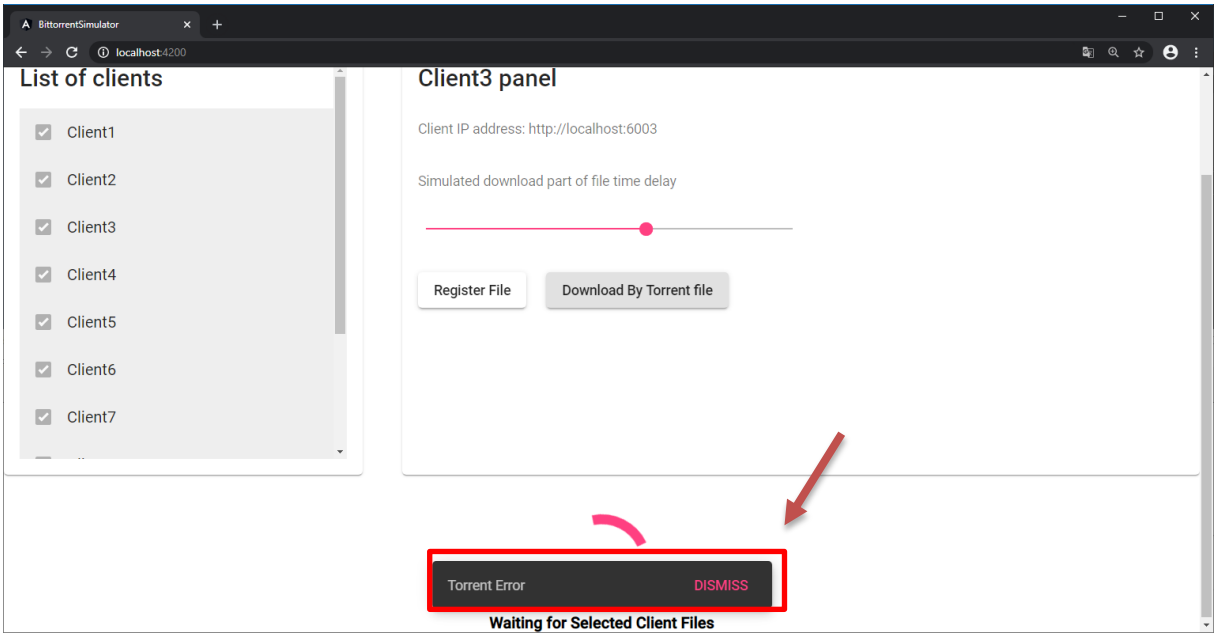
Powyższy scenariusz został przedstawiony na poniższych zrzutach ekranu.

Na początku u wszystkich dotychczasowych właścicieli pliku został zablokowany dostęp do utworzonego na początku tego rozdziału pliku, zablokowany dostęp oznaczony zostaje statusem pliku **NON_EXISTING**.



Rys. 6.4.1. Wprowadzanie symulowanej usterki.

W tym momencie u nowego klienta, dla przykładu numer 3 rozpoczęto próbę pobrania pliku na podstawie wciąż tego samego pliku torrent. Symulowany brak istnienia pliku został przedwcześnie zasygnalizowany.

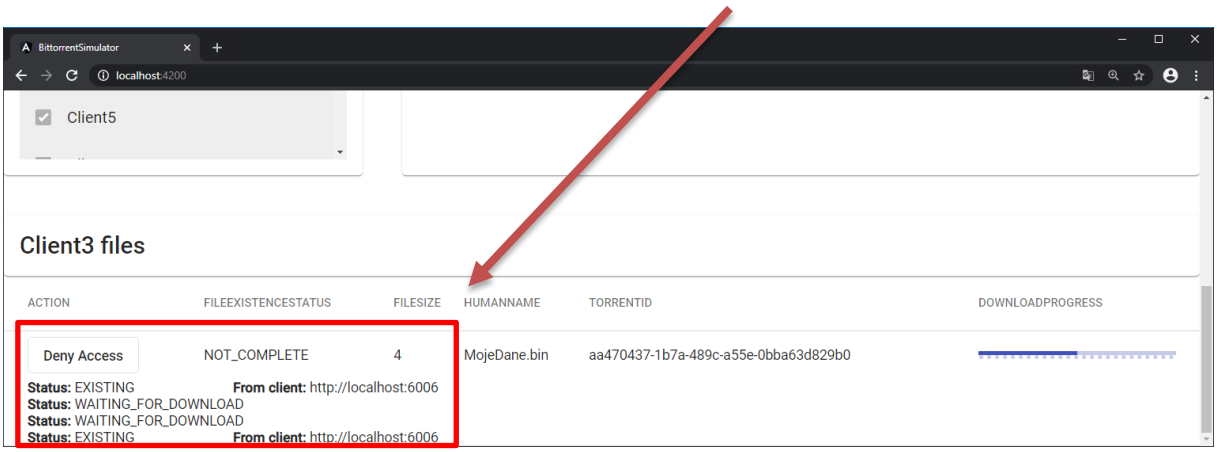


Rys. 6.4.2. Błąd wywołany symulowanym brakiem istnienia pliku.

Jednak teraz u jednego z właścicieli dostęp zostanie przywrócony odwróconą flagą tego samego przycisku **Allow Access (Odblokowano dla przykładu u Klienta numer 6).** Po tym zdarzeniu ponownie zostanie podjęta próba pobrania tego pliku przez klienta numer 3.

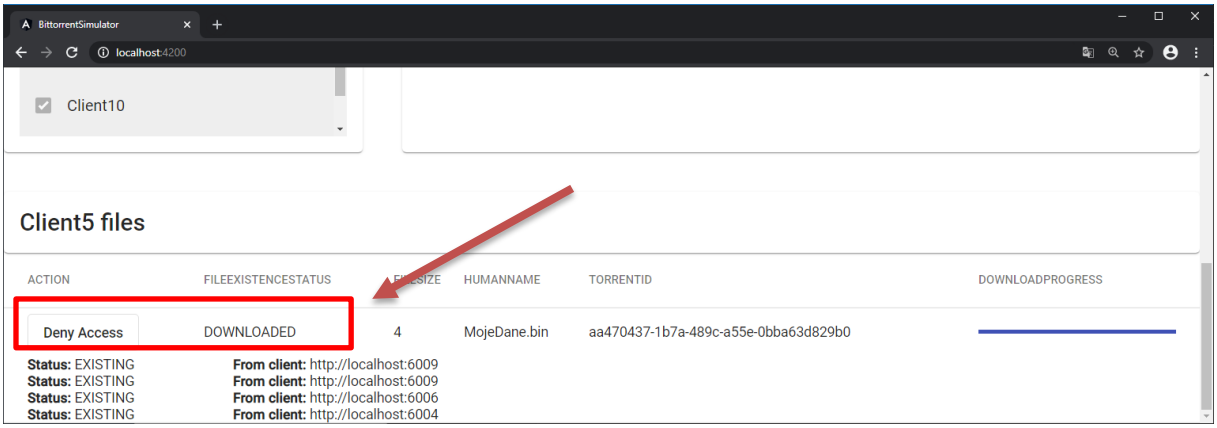
Po pobraniu 50% zawartości pliku u Klienta numer 6 ponownie zablokowano dostęp (Zasymulowano ponownie brak tego pliku w sieci tym razem w trakcie pobierania).

Na poniższym zrzucie ekranu zaprezentowano dwie części od wyłącznie klienta numer 6.



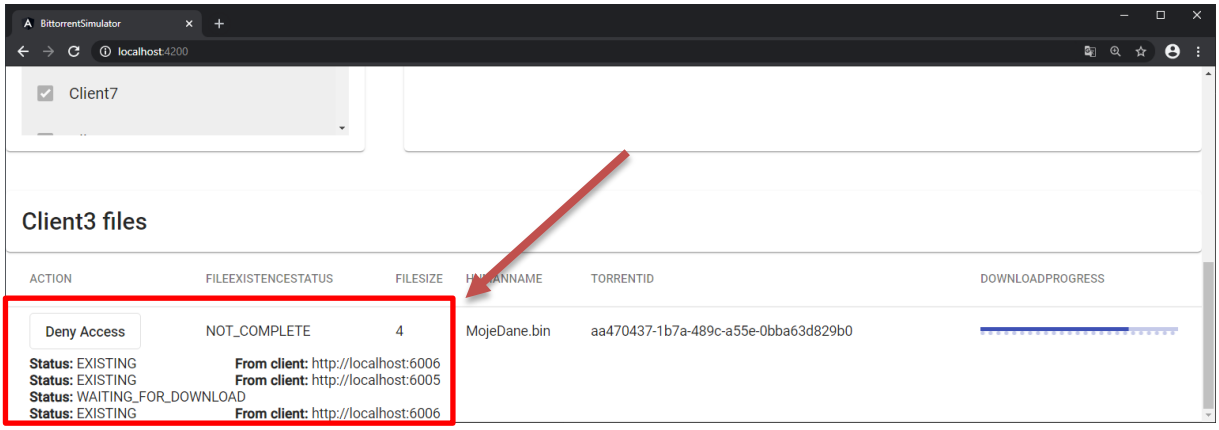
Rys. 6.4.3. Status pobierania pliku podczas symulowania błędu.

Klient numer 3 wszedł w stan oczekiwania na ponowne pojawienie się pliku, w tym wypadku można zasymulować wejście w jego posiadanie przez innego uczestnika sieci np.: Klienta numer 5, u którego zmieniono stan zablokowania pliku zaprezentowanego na poniższym zrzucie ekranu.



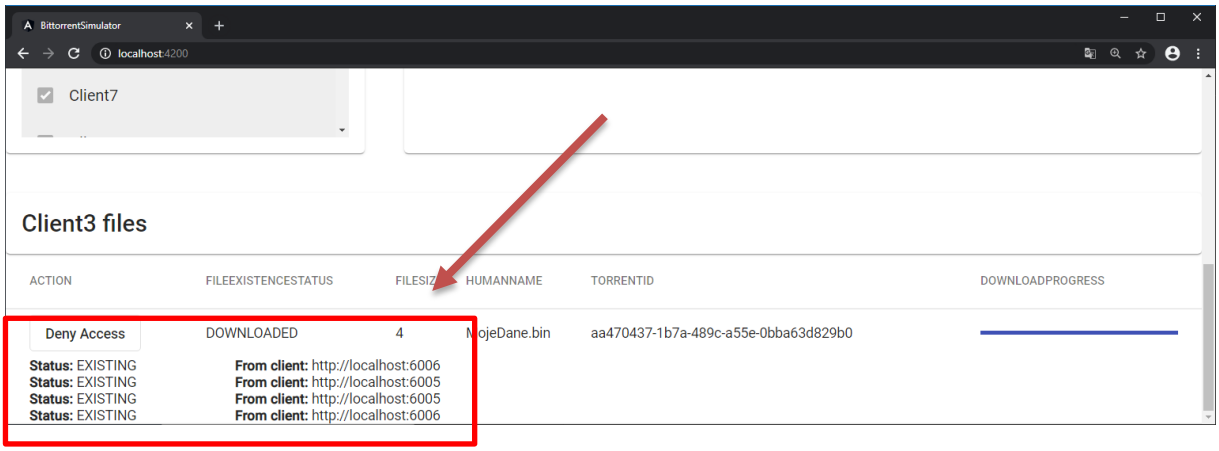
Rys. 6.4.4. Zmiana stanu pobierania pliku podczas symulowania błędu.

W tym momencie Klient numer 3 może dokończyć pobieranie pozostałych części pliku, prezentuje to poniższy zrzut ekranu. Pozostałe części pochodzą tym razem od Klienta numer 5. **Klienta numer 3 w trakcie wznowionego pobierania** zaprezentowano na poniższym zrzucie ekranu (widoczna część od Klienta numer 5).



Rys. 6.4.5. Klienta numer 3 w trakcie wznowionego pobierania podczas symulowania błędu.

Klient numer 3 po ukończonym pobieraniu pozostałych części, plik jest już kompletny.



Rys. 6.4.6. Klient numer 3 po ukończonym pobieraniu pliku podczas symulowania błędu.

7. WNIOSKI

Główne założenia dotyczące symulacji działania protokołu BitTorrent zostały zaimplementowane oraz przetestowane. Zgodnie z założeniem przetestowano działanie na wielu instancjach uczestników sieci. Zaprezentowano działanie pobierania poszczególnych części pliku z różnych źródeł.

Słabą stroną protokołu jest na pewno sytuacja, w której istnieje informacja o dostępności pliku, pewna część zostanie pobrana przez danego klienta a następnie udostępniający (seed) usunie plik (lub zniknie z sieci). Wówczas dotychczas rozpoczęte u innego uczestnika pobieranie pliku może nigdy nie dojść do skutku.

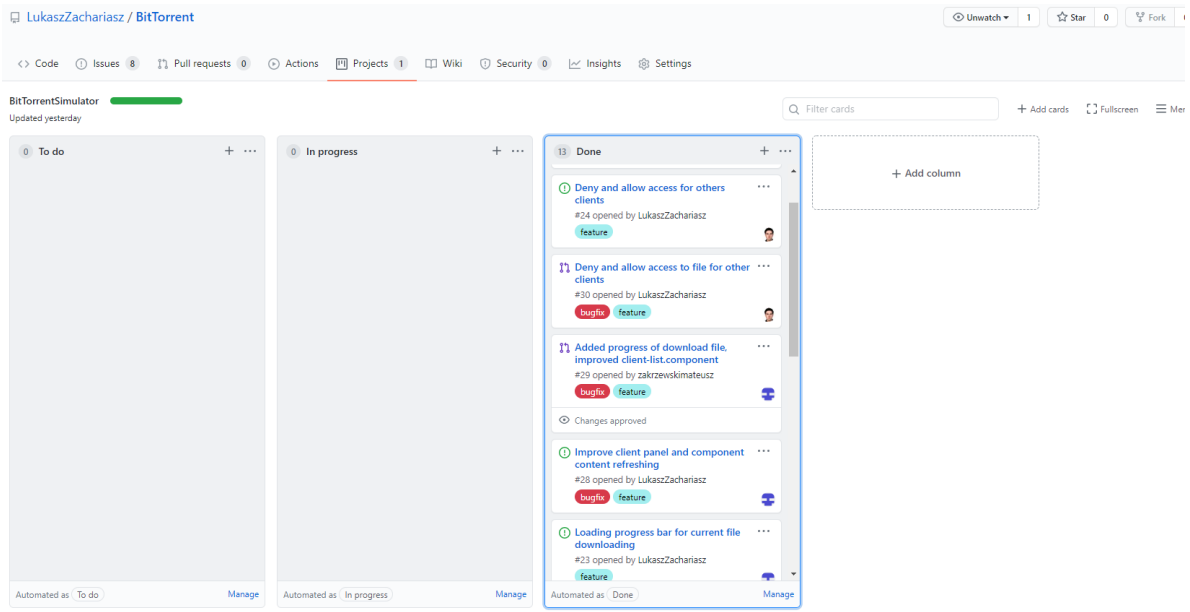
Niepodważalną zaletą protokołu jest możliwość rozproszonego pobierania pliku (poszczególnych części), skutkuje to ogólnie krótszym czasem pobierania pliku. Części pobierane są niezależnie.

Dobrą stroną protokołu jest również sytuacja, w której wielu uczestników posiada oraz udostępnia dany plik, wówczas istnieje wiele źródeł dostępu do tych samych danych dla innych klientów chcących uzyskać dany plik. Gwarantuje to dużą niezawodność.

Innym scenariuszem, który przemawia jako zaleta protokołu jest sytuacja, w której dwóch klientów posiada po 50% pliku jednak części te nie pokrywają się. W tej sytuacji dojdzie do

wymiany posiadanych części i tym sposobem obaj klienci będą w posiadaniu pełnych wersji pliku.

Podczas tworzenia projektu korzystano regularnie z narzędzia Systemu Kontroli Wersji Git. Na repozytorium GitHub powstała historia ostatnich zadań dla projektu, które sami sobie wyznaczaliśmy następnie je realizując.



Rys. 7.1. Zarządzanie projektem za pomocą: Systemu Kontroli Wersji Git.

8. BIBLIOGRAFIA

[1] *Angular dokumentacja* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://angular.io/docs/

[2] *BitTorrent* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://en.wikipedia.org/wiki/BitTorrent

[3] *Doker dokumentacja* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://docs.docker.com/

[4] *Java dokumentacja* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://docs.oracle.com/javase/7/docs/api/

[5] *Maven dokumentacja* [online], [dostęp: 12.06.2020], dostępny w Internecie: http://maven.apache.org/guides/

[6] *P2P* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://en.wikipedia.org/wiki/Peer-to-peer_file_sharing

[7] *Spring dokumentacja* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

[8] *Torrent* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://en.wikipedia.org/wiki/Torrent_file

[9] *Tracker* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://en.wikipedia.org/wiki/BitTorrent_tracker

[10] *TypeScript dokumentacja* [online], [dostęp: 12.06.2020], dostępny w Internecie: https://www.typescriptlang.org/docs/home.html