|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| URL | POST | GET | PUT | PATCH | DELETE |
| ~/players | 1 | 2 | - | - | - |
| ~/players/{n\_name} | - | 3 | 4 | 4 | 5 |
| ~/messages | 6 | - | - | - | - |
| ~/messages/{n\_nad};{n\_odb} | - | 7 | - | - | - |
| ~/messages/{n\_nad};{n\_odb}/{m\_id} | - | - | 8 | 8 | 9 |
| ~/histories | 10 | 11 | - | - | - |
| ~/histories/{g\_name} | - | 12 | - | - | - |
| ~/player\_merges | 13 | 14 | - | - | - |
| ~/player\_merges/{pm\_id} | - | 15 | - | - | - |

Zasoby: ~/players  
 (1) Dodawanie gracza [POST];   
 (2) Pobieranie par Nick i pkt do rankingu (stronicowanie) [GET];  
 ~/players/{uname}  
 (3) Pobieranie profilu gracza [GET];   
 (4) Aktualizacja pkt, (nie - ilości wiadomości) [PATCH/PUT];  
 (5) Usunięcie gracza [DELETE];  
 ~/messages   
 (6) Wysłanie wiadomości (utworzenie nowego zasobu w messages,   
 aktualizacja ilości wiadomości odebranych i wysłanych) [POST];  
 ~/messages/{u\_nad};{u\_odb}   
 (7) Pobranie wiadomości odebranych/wysłanych [GET];   
 ~/messages/{u\_nad};{u\_odb}/{m\_id}  
 (8) Edycja wiadomości [PATCH/PUT];   
 (9) Usunięcie wiad. [DELETE];  
 ~/histories  
 (10) Dodanie gry do historii [POST];

(11) Pobranie wszystkich historii (stronicowanie) [GET]  
 ~/histories/{g\_name}  
 (12) Pobranie historii gry o danej nazwie [GET];  
 ~/player\_merges  
 (13) Łączenie dwóch kont graczy [POST]  
 (14) Historia łączeń kont [GET]  
 ~/player\_merges/{pm\_id}   
 (15) Kiedy i jakie konta zostały połączone [GET]  
Modele: Gracz:

* Nick – pseudonim gracza służący do identyfikacji i wpisania się do gry; [string]
* Rekord punktów – maksymalna uzyskana liczba pkt; [int]
* Wiadomości wysłane – ilość wysłanych wiadomości; [int]
* Wiadomości odebrane – ilość odebranych wiadomości; [int]

Gra z historii:

* Data – data na serwerze kiedy skończyła się rozgrywka; [data]
* Identyfikator gry – id/losowa nazwa; [string]
* Tabela par – (Nick gracza; pkt uzyskane przez gracza); [string; int]

Wiadomość:

* Nadawca – Nick gracza wysyłającego wiadomość; [string]
* Odbiorca – Nick gracza odbierającego wiadomość; [string]
* Treść – treść wiadomości; [string]

Historia:

* Tabela gier z historii; [Tab gier z hist.]

Połączenie kont:

* Data połączenia; [data]
* Nicki graczy połączonych i nick finalny; [strings]

W projekcie powinny wystąpić następujące elementy:

1. Elementem składowym zaliczenia jest projekt usługi, obejmujący:
   * opis hierarchii zasobów,  
     [URL ~\zasób\podzasób]
   * znaczenie poszczególnych operacji protokołu HTTP w odniesieniu do zasobów,  
     [Tabelka metod HTTP]
   * opis formatów danych używanych do reprezentacji danych wejściowych i wyjściowych dla poszczególnych zasobów.  
     [Przesyłane argumenty w request body]  
     [Jak reprezentujemy zasób: JSON i co w środku – reprezentacje]

[Warto dodać kody błędów do dokumentacji!]

1. Zakres funkcjonalny usługi jest dowolny. Usługa powinna mieć jednak nietrywialny poziom złożoności. Wśród udostępnianych zasobów powinny się znaleźć (co najmniej po 1 sztuce):
   * proste zasoby oferujące pełen zakres operacji CRUD,  
     [Utworzenie, pobranie, aktualizacja i usunięcie gracza/wiadomości]
   * zasoby-kolekcje będące reprezentacją zbiorów innych zasobów i stosujące stronicowanie w swoich reprezentacjach,  
     [Pobranie listy rankingowej graczy, historii, wiadomości – ma to być osobny zasób, czy tak jak jest]
   * zasoby-kontrolery umożliwiające atomowe wykonanie aktualizacji kilku innych zasobów,  
     [Merge – łączenie dwóch graczy]
   * zasoby przyjmujące zlecenia w trybie POST once exactly, eliminujące wielokrotne wysyłanie tych samych danych.  
     [Wystarczy jeden zasób robiący POST – by przećwiczyć mechanizm]
2. Wykonywanie aktualizacji zasobów (PUT) powinno być wykonywane w trybie weryfikacji, wykluczającej niesygnalizowane nadpisywanie współbieżnie nanoszonych zmian przez różnych klientów (lost update problem).  
   [Wszystkie użycia PUT, w tym PATCH]
3. Implementacja powinna wykorzystywać dowolnie wybrane środowisko programistyczne (framework) wspierające budowę usług sieciowych REST.  
   [python Tornado + tornado-rest-swagger dla dokumentacji]
4. Do prezentacji usługi można wykorzystać dowolnego klienta protokołu HTTP lub stworzyć dodatkową własną aplikację.  
   [Początkowo Swagger & Postman, po zrobieniu projektu WebSocket może uda się połączyć projekty, jak starczy czasu]