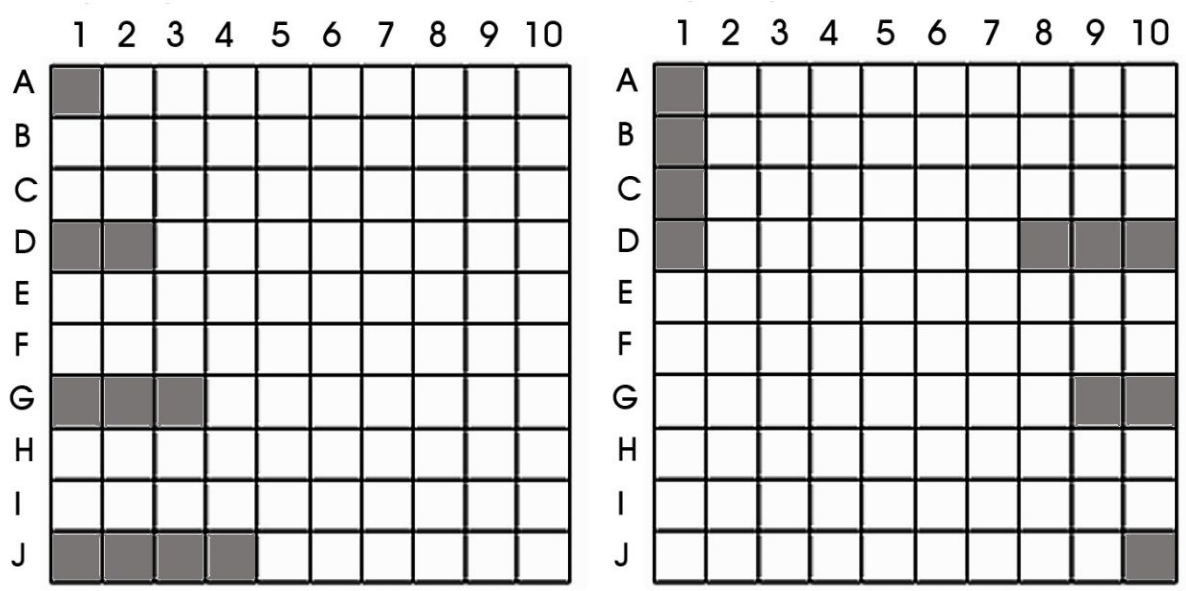
Zadanie programistyczne - Backend Developer

Napisz prostą aplikację REST-ową (sam backend) umożliwiającą graczom przeprowadzenie uproszczonej rozgrywki w Statki oraz spełniającą poniższe założenia:

- Rozmiar planszy 10x10.
- Równolegle grę może prowadzić wiele par graczy.
- Każdy z graczy ma po jednym z następujących statków: czteromasztowiec, trzymasztowiec, dwumasztowiec, jednomasztowiec.

Uproszczenia:

- Gracze nie rozmieszczają statków samodzielnie i ich układ na obu planszach jest stały (poniżej odpowiednio plansze Gracza A i Gracza B).



Standardowy scenariusz rozgrywki:

1. Gracz A wysyła request POST do adresu */game*, aplikacja generuje id i w odpowiedzi zwraca graczowi w nagłówku *Set-Auth-Token* z jego tokenem oraz w body adres z zaproszeniem w postaci *<app_url>/game/{id}/join* do przesłania Graczowi B.
2. Gracz B wysyła POST na podany adres do dołączenia, w odpowiedzi dostaje w nagłówku *Set-Auth-Token* swój token, a w body aktualny (początkowy stan gry).
3. Po dołączeniu do gry, rozgrywkę (strzelanie) zaczyna Gracz B (zaproszony do gry).
4. Gracze na zmianę wysyłają strzały w postaci PUT */game/{id}* ze swoim Auth-Token w nagłówku. Format body: *{ "position": "A5" }* . W odpowiedzi dostają JSON-a z informacją czy trafili, jaki statek, czy został zatopiony. W przypadku trafienia, dany gracz wykonuje kolejny strzał.
5. Gracze mogą sprawdzać stan gry poprzez wywołanie GET */game/{id}* wraz z odpowiednim tokenem.

Szczegóły API:

- Tworzenie nowej gry
  - Request: POST */game*

    - Response: { “invitationUrl”: “<app_url>/game/{id}/join”} oraz nagłówek *Set-Auth-Token* z tokenem.
- Dołączenie do gry:
    - Request: POST */game/{id|/join*
    - Response: początkowy stan gry, body takie same jak w GET */game/{id}*, czyli { “gameStatus:” “YOUR_TURN”, “yourScore”: 0, “opponentScore: 0 } oraz nagłówek *Set-Auth-Token* z tokenem.
- Strzał:
    - Request: PUT */game/{id}* z Auth-Token w nagłówku, body*: { “position”: “A5” }*
    - Response :
        - { “result”: “MISS” }, gracz sam musi pamiętać gdzie już strzelał. Ponowny strzał w to samo miejsce skutkuje utratą szansy poprzez zwrócenie MISS, a kolejka przechodzi do drugiego gracza
        - { “result”: “HIT”, “shipType”: “FOUR_DECKER”, “sunken”: false }, typy statków w odpowiedzi: FOUR_DECKER, THREE_DECKER, TWO_DECKER, ONE_DECKER
- Stan gry:
    - Request: GET */game/{id}* z Auth-Token w nagłówku
    - Response: { “gameStatus:” “YOUR_TURN”, “yourScore”: 4, “opponentScore: 2 }, score to ilość zestrzelonych masztów przez aktualnego gracza i przeciwnika. Możliwe wartości stanu gry: AWAITING_PLAYERS, YOUR_TURN, WAITING_FOR_OPPONENT_MOVE, YOU_WON, YOU_LOST.

Sytuacje wyjątkowe (błędy, próby dołączenia do nieistniejącej gry, niepoprawne wartości w requestach, itd.) powinny skutkować zwróceniem błędnego statusu HTTP wraz z opisem błędu (format JSON-a: ma mieć pole “message”, reszta dowolnie).

Wymagania techniczne:

1. Backend powinien być zaimplementowany w Javie 8+, Scali i/lub Kotlinie
2. Projekt powinien budować się przy użyciu Mavena, SBT lub Gradle’a.
3. Wybór pozostałych bibliotek i frameworków jest dowolny.
4. Nie używamy bazy danych (PostgreSQL, H2, etc.), restart aplikacji oznacza utratę wszystkich gier
5. Aplikacja powinna się uruchamiać na wybranym serwerze embedded jako fat-jar.

Co powinno znaleźć się w kompletnym rozwiązaniu:

1. Skrypt compile.sh, który buduje projekt.
2. Skrypt deploy.sh, który buduje i uruchamia aplikację, a następnie na wyjściu wyświetla adres, pod którym została ona uruchomiona.
3. Plik README.md z odpowiedziami na pytania znajdujące się końcu tego dokumentu.

Co podlega ocenie:

1. Czy aplikacja kompiluje się, uruchamia poprawnie i działa zgodnie z wymaganiami.
2. Prawidłowa obsługa typowych sytuacji spotykanych przy grze sieciowej.
3. Jakość kodu i sposób pracy zbliżony do pracy nad projektem komercyjnym.
4. Czystość kodu w repozytorium.
5. Sposób zamodelowania domeny oraz struktura projektu.
6. Dobór technologii zgodnie z potrzebami projektu oraz własnymi kompetencjami.

Rozwiązanie traktujemy jako zamkniętą całość i rzadko prosimy kandydatów o poprawki rzeczy, do których mamy zastrzeżenia, więc prosimy o dostarczanie kompletnego rozwiązania, nawet kosztem przekroczenia terminu o kilka dni. A wszelkie założenia, które zostały przyjęte przy rozwiązywaniu, a mogą mieć wpływ na ocenę, prosimy umieszczać w pliku README.md.

Pytania do zadania:

Please write your answers in English in README.md

1. What technologies have you chosen to implement the backend? Why?
2. How much time did you spend on this task? Could you please divide this number into a few most important areas?
3. How would you modify your application if the next feature to implement would be to allow players to place their ships on the gameboard? What new things need to be implemented to make it work?
4. Assume that we have to fetch game ID from the external application. What challenges and potential problems do you see and how would you prepare your application to handle them properly?

Informacje końcowe:

Ewentualne pytania (jesteśmy Twoim Klientem) oraz informację, że zadanie jest gotowe do oceny prosimy wysyłać na czlowieki@softwaremill.com z cc do osoby, która przeprowadziła z Tobą pierwszą rozmowę.

Powodzenia! :)