Тестовое задание "Backend developer coding task "Social tournament service"

Автор: Елена Лазарь, helen.lazar.77@gmail.com

Skype: morrah77

Текст задания

Please read the task, find possible issues you see and will face during implementation and how u plan technically to solve such issues. On proposed solutions please comment on benefits and drawbacks of picked solution. Please produce a 5-minute audio in which you declare from 3 to 5 possible problems, relevant number of solutions and their pros and cons.

Примечание. в переписке по Skype была оговорена возможность предоставления ответа в pdf-виде, вот почему составлен этот документ.

1. **предложенный API:**

    1. **неполон, отсутствуют следующие необходимые для предполагаемого tournament flow запросы:**

        - - запрос участника турнира к потенциальному backer'у с предложением вложить средства в определенном размере, например, POST /backRequest -d {"requestorId":"P1","backerId":"P2","requestedSum":100.0 }
        - - запрос на подтверждение участия в качестве backer'а со стороны пользователя, получившего предложение, описанное выше
        - - запрос на создание ID турнира (см. п. 2)

    2. **не соответствует соглашениям REST в части "запрос существующих сведений - GET, изменение данных - PUT/POST/DELETE" - хотя бы из-за кеширования (да, кешируется GET, как правило, в браузерах, но кто знает, вдруг именно эти вызовы будут использованы в браузерном UI-приложении, а не только для бмена между внутренними сервисами системы? сказано же, " REST service ... should be implemented...")**

        - - например, запросы /fund, /take, /joinTournament, /announceTournament изменяют существующие данные, их лучше реализовать как post-запросы с идентификаторами/суммами в теле запроса, в том же JSON.
        - - GET-запросы, кроме кеширования, обычно ограничены в части длины строки запроса.

2. **/joinTournament?tournamentId=1&playerId=P1&backerId=P2&backerId=P3**

    Нежелательно использовать несколько query-параметров с одним именем. Да, технически возможно найти готовый или слепить свой query-парсер, собирающий одноименные параметры в массив, но все же нежелательно по причине невозможности в дальнейшем перескочить на любой web-фреймворк - а вдруг придется, скажем, на Spring переезжать? Там с conventions строго... Ну, и длина GET-запроса весьма ограничена, зачем лишние символы?

1. **Если есть железное требование реализовать GET-запрос:**

    1. **лучше так:**

        /joinTournament?tournamentId=1&playerId=P1&backerIds=P2,P3,P4
        Достоинства:

        - - отсутствие повторяющихся имен параметров
        - - сокращение длины строки запроса по сравнению с предложенным вариантом, позволяющее увеличить количество backer'ов

        Условный недостстаток - convention о неиспользовании в идентификаторах игроков символа ',', которое наверняка уже присутствует вследствие хранения информации об игроках в БД

    2. **или так:**

        /joinTournament?
        tournamentId=1&playerId=P1&backerId[0]=P2&backerId[1]=P3&backerId[2]=P4
        Достоинства: см. предыдущий пункт
        недостатки:

        - - лишняя по сравнению с предыдущим вариантом длина query string
        - - необходимость парсить имена по префиксу со сбором значений в массив (недостаток условный, наверняка удастся найти и привернуть готовую библиотеку, в кр. случае, написать и покрыть тестами свою не так уж долго)

2. **Наилучшеее решение:**

    POST /joinTournament -d {"playerId":P1, "backerIds":[P2,P3,P4]}
    Достоинства:

    - - практически нет ограничений на количество backer'ов
    - - соблюдается соглашение 'request data by GET, change data by POST'
    - - существует масса готовых библиотек для разбора JSON, в Go и js они даже частью стандартной библиотеки являются
    - - если возникнет необходимость позволить пользователям - игрокам и их backer'ам - вкладывать в турнир разные суммы, изменения в API и серверной логике потребуются меньшие, чем при решении, предложенном в исходном задании, или двух решениях, предложенных выше.

    Пример:
    POST /joinTournament -d {"playerId":P1,"playerDeposit:500, "backers":
    [{"id":P2,"deposit":100},{"id":P3,"deposit":400},{"id":P4,"deposit":200}]}

    Недостатки:
    не обнаружено

3. **REST call /announceTournament?tournamentId=1&deposit=1000**

Передавать ID вновь объявленного турнира в параметрах GET-запроса - а где генерится этот ID? Где сведения об инициаторе турнира?

Подковерное соглашение о том, что все турниры объявляются от имени системы? Крррайне неочевидно, следовательно, unmaintainable.
Еще о неподдерживаемости: турнир может быть привязан ко времени, региону (часовой пояс, law restrictions etc.), обрасти в будущем множеством параметров, теми же ID инициатора, ID игры.
Другие недостатки, присущие GET, описаны в п. 1.2

1. **Решение 1:**

   request:
   POST /announceTournament -d {"deposit":1000}

   response:
   200 OK / {"tournamentId":1}
   или
   400 Bad request

   Достоинства:

   - - ID турнира создается на сервере, исключаются ошибки, связанные с объявлением турнира с одинаковым ID, упрощается валидация данных запроса.
   - - в респонсе в будущем можно возвращать параметры турнира, добавляемые сервером по умолчанию - дату создания, инициатора и т. п.

   Недостатки:
   - - кто-нибудь может сказать о нарушении SRP. Считаю, что все принципы/паттерны/прочие спопобы унификации взаимодействий должны использоваться разумно, одновременное создание запроса и задание ему основных параметорв недостатком не считаю.

2. **Решение 2:**

   request 1:
   POST /createTournament -D {"deposit":1000}

   response:
   200 OK / {"tournamentId":1}
   или
   400 Bad request

   request 2:
   POST /announceTournament -d {"tournamentId":1,"deposit":1000}

   response:
   200 OK / пустой респонс или {"tournamentId":1,"deposit":1000}
   или
   400 Bad request

   Достоинства:
   те же, что в предыдущем решении
   SRP во всей красе: отдельный call на создание, отдельный на настройку

   Недостатки:

- - два запроса вместо одного
- - избыточность ради соблюдения принципов архитектуры === unmaintainability

3. **Дополнение**

   Очевидно, что в букет вызовов нашего API просится еще и GET /tournamentInfo?tournamentId=1 для извлечения сведений о турнире.

4. **Запрос /take, а, скорее всего, и /fund должны требовать авторизации, соответственно необходимо ее реализовать.**

5. **/resultTournament with a POST document in format {"winners": [{"playerId": "P1", "prize": 500}]} - Очевидно, при записи победителей турнира потребуется не только авторизация, но и ID турнира в теле запроса**

   request:
   POST /resultTournament -d {"tournamentId":1,"winners": [{"playerId": "P1", "prize": 500}]}
   response:
   200 OK

6. **Серверная логика и структура хранимых данных в задании не описана. Можно реализовать так:**

   1. **Примерная структура данных в реляционной БД:**

      - relation Users (id, name etc...)
      - relation tournaments (id, deposit, announceDate, gameTypeId, status(pending|playing|complete), completionDate etc...)
      - relation tournamentsPlayers (tournamentId, userId, userDeposit)
      - relation tournamentsBackers (tournamentId, userId, backerId, backerDeposit)
      - relation tournamentsWinners (tournamentId, userId, prize)
      - неизбежная relation UserAuth(userId, lastUpdate, authToken), например
      - *// далее - только в части операций и баланса для bonus points*
      - relation userPointsOperations (userId, operationType(debt|credit => 0|1), sum, date) (одна из возможных реализаций операционной таблицы!)
      - relation userPointsBalance(userId, balance)

   2. **действия, производимые сервером при обработке http-запросов: (действия, необходимые для валидации данных, проверки наличия запрашиваемых турниров и достаточности баланса юзеров не привожу)**

      - /fund -d {"playerId":P1,"points":300} => INSERT INTO `userPointsOperations` (`userId`, `operationType`, `sum`) VALUES('P1','0',300)
      - /take -d {"playerId":P1,"points":300}
        =>
        INSERT INTO `userPointsOperations` (`userId`, `operationType`, `sum`) VALUES('P1','1',300)
      - POST /joinTournament -d {"playerId":P1,"playerDeposit:500, "backers": [{"id":P2,"deposit":100},{"id":P3,"deposit":400},{"id":P4,"deposit":200}]}
        =>
        INSERT INTO `tournamentsPlayers` (`tournamentId`, `userId`, `userDeposit`)

VALUES (`1`,`P1`, 500);
INSERT INTO `tournamentsBackers` (`tournamentId`, `userId`, `backerId`, `backerDeposit`) VALUES (`1`,`P1`, `P2`, 100),(`1`,`P1`, `P3`, 400),(`1`,`P1`, `P4`, 200);
INSERT INTO `userPointsOperations` (`userId`, `operationType`, `sum`) VALUES ('P1','1',500),('P2','1',100),('P3','1',400),('P4','1',200);

- POST /resultTournament -d {"tournamentId":1,"winners": [{"playerId": "P1", "prize": 2400}]}
  =>
  SELECT deposit, status FROM tournaments WHERE ID=1
  ...
  SELECT tournamentId, userId, backerId, backerDeposit FROM tournamentsBackers WHERE tournamentId=1

  Далее следует пересчет причитающихся игроку и вкладчикам долей выигрыша в соответствии с вложенными суммами

  INSERT INTO `tournamentsWinners` (`tournamentId`, `userId`, `prize`) VALUES (`1`,`P1`, 2400);
  INSERT INTO `userPointsOperations` (`userId`, `operationType`, `sum`) VALUES ('P1','0',1000),('P2','0',200),('P3','0',800),('P4','0',400);

дальше все sraight-forward, не привожу