Java back-end developer test

1 Purpose

The goal of this test is to provide us with a full understanding of your coding style and skills. We'll pay particular attention to:

- The code structure
- Consideration of concurrency issues
- The design
- Choice of data structures
- Quality and use of unit tests

The solution should not take more than half a day to write, though it will be difficult to write a "complete" and top-notch solution in such a short time span. The goal is not to get a solution covering all special cases in a 100% robust way; the functions should be error free when used correctly but our main goal is to understand your approach to the problem.

2 Description

Write a HTTP-based mini game back-end in Java which registers game scores for different users and levels, with the capability to return high score lists per level. There shall also be a simple login-system in place (without any authentication...).

Deliver a zip file containing:

- The code in the src-folder
- A compiled version in a executable .jar file in the root folder
- An optional readme.txt or .pdf with thoughts and considerations around the program
- Please do not include your name or any other identifying information in any of the files in the zip, for example in the readme.txt or autogenerated author comments - this is because we run a blind review process to ensure that bias does not play a role in marking the tests

Please note: This type of exhaustive specification is extremely rare at King and does by no means represent the way we do or want to work. In reality the "specification" is almost always verbal and implicit and it is up to the individual developers or groups to formulate code that best represents the solution to a need of a game. We do not use documentation as a means of communication, but in this specific case we see no other way!

3 Nonfunctional requirements

- This server will be handling a lot of simultaneous requests, so make good use of the available memory and CPU, while not compromising readability or integrity of the data.
- Do not use any external frameworks or any classes not included in the JDK, except for testing. For HTTP, use com.sun.net.httpserver.HttpServer.
- There is no need for persistence to disk, the application shall be able to run for any foreseeable

future without crashing anyway.

4 Functional requirements

The functions are described in detail below and the notation <value> means a call parameter value or a return value. All calls shall result in the HTTP status code 200, unless when something goes wrong, where anything but 200 must be returned. Numbers parameters and return values are sent in decimal ASCII representation as expected (ie no binary format).

Users and levels are created “ad-hoc”, the first time they are referenced.

4.1 Login

This function returns a session key in the form of a string (without spaces or “strange” characters) which shall be valid for use with the other functions for 10 minutes. The session keys should be “reasonably unique”.

```
Request: GET /<userid>/login
Response: <sessionkey>

<userid> : 31 bit unsigned integer number
<sessionkey> : A string representing session (valid for 10 minutes).

Example: http://localhost:8081/4711/login --> UICSNDK
```

4.2 Post a user's score to a level

This method can be called several times per user and level and does not return anything. Only requests with valid session keys shall be processed.

```
Request: POST /<levelid>/score?sessionkey=<sessionkey>
Request body: <score>

Response: (nothing)
<levelid> : 31 bit unsigned integer number
<sessionkey> : A session key string retrieved from the login function.
<score> : 31 bit unsigned integer number

Example: POST http://localhost:8081/2/score?sessionkey=UICSNDK (with the post
body: 1500)
```

4.3 Get a high score list for a level

Retrieves the high scores for a specific level. The result is a comma separated list in descending score order. Because of memory reasons no more than 15 scores are to be returned for each level. Only the highest score counts. ie: an user id can only appear at most once in the list. If a user hasn't submitted a score for the level, no score is present for that user. A request for a high score list of a level without any scores submitted shall be an empty string.

```
Request: GET /<levelid>/highscorelist
```

Response: CSV of <userid>=<score>

<levelid> : 31 bit unsigned integer number

<score> : 31 bit unsigned integer number

<userid> : 31 bit unsigned integer number

Example: http://localhost:8081/2/highscorelist -> 4711=1500,131=1220