# HttpServer Game Scores

The code implements 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.

## Compiling the application

Prerequisites:

- JDK 8
- Maven 3.0+

## Running the application

### Start the server

```
java -jar target/httpserver-scores.jar
```

### Stop the server

```
`Ctrl+c`.
```

### Configuration

Configuration file location:

```
src\main\resources\configuration.properties
```

To customize the behaviour of the application the following parameters can be passed as arguments:

| Parameter | Type | Default | Description |
|---|---|---|---|
| BASE_URI | String | localhost | Server host name |
| SERVER_PORT | Integer | 8081 | Server port |
| EXPIRATION_TIME | Integer | 600000 | Expired timeout in milli sec |
| EXPIRATION_TIME_PERIOD_CHECK | Integer | 10000 | Expired timeout cleanup task check period in ms |
| EXPIRATION_TIME_PERIOD_DELAY | Integer | 0 | Expired timeout cleanup task check delay in ms |
| HIGH_SCORES_LIMITATION | Integer | 15 | Limitation of high scores list |
| HIGH_SCORES_THRESHOLD_IMITATION | Integer | 1000 | Space limitation for each level for server space performance improvement |

## Technical approach

The architecture of the application was made as simple as possible.

- Handler is in charge of receiving the request and forwarding it to the appropriate controller.
- Controller gathers the information that it requires for processing from the request and forwards it to the service.
- Manager applies the logic to the received request.

### Data Structures

### Session

As the requirement specification doesn't declare that one user should only have one single session in the memory. Design as creating new session key every time login request even same user.

- Use `ConcurrentHashMap<String, UserSession>` to store the data, the key is session key.
- Creating session, generate key by UUID, then put into ConcurrentHashMap.
- Get session will get O(1) average runtime to retrieve user session.
- When server started, jvm will start a clean up task for space performance and logout, that task will execute per 10 seconds.
- Clean up expired and duplicated session for single user, eventually consistent.

### Score

Initially, I tried to work with post score as the following code to hold everything in memory:

```
levelScores.computeIfAbsent(levelId, n -> new ConcurrentSkipListSet<>()).add(userScore);
```

The code is easy, but scalability is important here, it could not reasonably be thought to hold all the information required.
Therefore, I changed my design as these:

- Use `ConcurrentHashMap<levelId, NavigableSet<UserScore>>` to maintain all the score we need.
- Use `ConcurrentSkipListSet` to maintain top limit (15) user score by different users.
- Use `ConcurrentHashMap<Integer, ConcurrentHashMap<Integer, UserScore>>` for de-duplicate same user in the score map.
- Use compareAndSet method of `AtomicInteger` to update high score if needed.
- As we only need top 15 high score, use another `THRESHOLD_NUM` for scalability, not need hold all scores.
- Get high score list by traverse `ConcurrentSkipListSet`, also de-duplicate same user just in case, guarantee eventually consistent.

## Concurrency

- Managers for both session and score data storage is using double check singleton pattern.
- Use lock free code logic rather than Synchronized method for low latency performance

# Endpoints

## Login

### Input and output

| | Value | Description |
|---|---|---|
| **Path** | `/<userid>/login` | Requests the creation of a new session key every time the endpoint is called. |
| **Method** | `GET` | |
| **Response** | `<sessionkey>` | Unique string that represent the session. |

Example:

```
GET http://localhost:8081/2/login -> UICSNDK
```

### Response Error Codes

Http code always be 200.

| RESPONSE ERROR CODES | Description |
|---|---|
| 408 | INVALID PARAMETER |
| 410 | INVALID URL |

## Score

### Input and output

| | Value | Description |
|---|---|---|
| **Path** | `/<levelid>/score?sessionkey=<sessionkey>` | Method can be called several times per user and level. Requests with invalid session keys are ignored. |
| **Method** | `POST` | |
| **Request Body** | `<score>` | Integer number that represents the users score for the level. |
| **Response** | | Empty response. |

### Example:

```
POST http://localhost:8081/10/score?sessionkey=UICSNDK" 100
```

### Response Error Codes

Http code always be 200.

| RESPONSE ERROR CODES | Description |
|---|---|
| 408 | INVALID PARAMETER |
| 409 | INVALID SESSION KEY |
| 410 | INVALID URL |

## Get high score list

### Input and output

| | Value | Description |
|---|---|---|
| **Path** | `/<levelid>/highscorelist` | Retrieves the high score list for a level. The list size is determined by the Application configuration. |
| **Method** | `GET` | |
| **Response** | CSV of `<userid>=<score>` | Comma separated list with user id and scores. |

### Example:

```
GET http://localhost:8081/2/highscorelist -> 3=100
```

### Response Error Codes

| RESPONSE ERROR CODES | Description |
|---|---|
| 408 | INVALID PARAMETER |
| 410 | INVALID URL |

## Improvements

- Interceptor can be added for logging, performance monitoring
- Dependency Injection instead of creating objects and in terms of loose coupling
- More Unit tests for better coverage and thread safety.