Podium Documentation

Release 7.4.6

Top Free Games

Jun 18, 2019

Contents

Contents:

CHAPTER 1

Overview

What is Podium? Podium is a blazing-fast HTTP Leaderboard service and library. It could be used to manage any number of leaderboards of people or groups, but our aim is players in a game.

Podium allows easy creation of different types of leaderboards with no set-up involved. Create seasonal, localized leaderboards just by varying their names.

1.1 Features

- **Multi-tenant** - Just vary the name of the leaderboard and you can have any number of tenants using leaderboards;
- **Seasonal Leaderboards** - Including suffixes like `year2016week01` or `year2016month06` is all you need to create seasonal leaders. I'm serious! That's all there is to it;
- **No leaderboard configuration** - Just start notifying scores for members of a leaderboard. There's no need to create, configure or maintain leaderboards. Let Podium do that for you;
- **Top Members** - Get the top members of a leaderboard whether you need by absolute value (top 200 members) or percentage (top 3% members);
- **Members around me** - Podium easily returns members around a specific member in the leaderboard. It will even compensate if you ask for the top member or last member to make sure you get a consistent amount of members;
- **Batch score update** - In a single operation, send a member score to many different leaderboards or many members score to the same leaderboard. This allows easy tracking of member rankings in several leaderboards at once (global, regional, clan, etc.);
- **Easy to deploy** - Podium comes with containers already exported to docker hub for every single of our successful builds. Just pick your choice!
- **Use as library** - You can use podium as a library as well, adding leaderboard functionality directly to your application;

1.2 Architecture

Podium is based on the premise that you have a backend server for your game. That means we only employ basic authentication (if configured).

1.3 The Stack

For the devs out there, our code is in Go, but more specifically:

- Web Framework - Echo based on the insanely fast FastHTTP;
- Database - Redis.

1.4 Who's Using it

Well, right now, only us at TFG Co, are using it, but it would be great to get a community around the project. Hope to hear from you guys soon!

1.5 How To Contribute?

Just the usual: Fork, Hack, Pull Request. Rinse and Repeat. Also don't forget to include tests and docs (we are very fond of both).

CHAPTER 2

Hosting Podium

There are three ways to host Podium: docker, binaries or from source.

2.1 Docker

Running Podium with docker is rather simple. Our docker container image comes bundled with the API binary. All you need to do is load balance all the containers and you're good to go.

Podium uses Redis to store leaderboard information. The container takes parameters to specify this connection:

- `PODIUM_REDIS_HOST` - Redis host to connect to;
- `PODIUM_REDIS_PORT` - Redis port to connect to;
- `PODIUM_REDIS_PASSWORD` - Password of the Redis Server to connect to;
- `PODIUM_REDIS_DB` - DB Number of the Redis Server to connect to;

Other than that, there are a couple more configurations you can pass using environment variables:

- `PODIUM_NEWRELIC_KEY` - If you have a New Relic account, you can use this variable to specify your API Key to populate data with New Relic API;
- `PODIUM_SENTRY_URL` - If you have a sentry server you can use this variable to specify your project's URL to send errors to;
- `PODIUM_BASICAUTH_USERNAME` - If you specify this key, Podium will be configured to use basic auth with this user;
- `PODIUM_BASICAUTH_PASSWORD` - If you specify `BASICAUTH_USERNAME`, Podium will be configured to use basic auth with this password.
- `PODIUM_EXTENSIONS_DOGSTATSD_HOST` - If you have a statsd datadog daemon, Podium will publish metrics to the given host at a certain port. Ex. localhost:8125]* `PODIUM_EXTENSIONS_DOGSTATSD_RATE` - If you have a statsd daemon, Podium will export metrics to the deamon at the given rate

- `PODIUM_EXTENSIONS_DOGSTATSD_TAGS_PREFIX` - If you have a statsd daemon, you may set a prefix to every tag sent to the daemon

2.2 Binaries

Whenever we publish a new version of Podium, we'll always supply binaries for both Linux and Darwin, on i386 and x86_64 architectures. If you'd rather run your own servers instead of containers, just use the binaries that match your platform and architecture.

The API server is the `podium` binary. It takes a configuration yaml file that specifies the connection to Redis and some additional parameters. You can learn more about it at default.yaml.

2.3 Source

Left as an exercise to the reader.

CHAPTER 3

Meaning of Leaderboard Names

Leaderboard names carry a lot of semantic weight in Podium. Each leaderboard name is composed of two parts: leaderboard name and an optional season suffix.

3.1 Seasonal Leaderboards

If you want a leaderboard to be seasonal and have an expiration, Podium allows you to do it just by adding a suffix to it.

Let's say you want a weekly leaderboard for your Cario Sisters game. You would name that leaderboard `cario-sisters-year2016week01` when reporting scores for the first week, `cario-sisters-year2016week02` when reporting for the next week and so on.

Podium will expire the leaderboard in twice as many time as you provisioned your leaderboard to contain. That means a leaderboard with a week of data will be expired within 2 weeks after it's appointed start.

3.2 Available expirations

Podium supports many different expirations:

- Unix timestamps from and to;
- yyyymmdd timestamps from and to;
- Yearly expiration;
- Quarterly expiration;
- Monthly expiration;
- Weekly expiration.

3.2.1 Unix Timestamp Expiration

In order to use this type of expiration use leaderboard names like `cario-sisters-from1469487752to1469487753`. This means a leaderboard from the first timestamp to the second timestamp.

This kind of leaderboard has the ultimate flexibility, allow for configuration of a leaderboard duration up to the second. Just remember this is UTC timestamps.

3.2.2 yyyymmdd Timestamp Expiration

In order to use this type of expiration use leaderboard names like `cario-sisters-from20201010to20201011`. This means a leaderboard from the first timestamp to the second timestamp.

3.2.3 Yearly Expiration

In order to use this type of expiration use leaderboard names like `cario-sisters-year2016`. This means a leaderboard ranging from 1st of January of 2016 to the 1st of January of 2017(not included).

3.2.4 Quarterly Expiration

In order to use this type of expiration use leaderboard names like `cario-sisters-year2016quarter01`. This means a leaderboard ranging from 1st of January of 2016 to the 1st of April of 2016(not included).

3.2.5 Monthly Expiration

In order to use this type of expiration use leaderboard names like `cario-sisters-year2016month03`. This means a leaderboard ranging from 1st of March of 2016 to the 1st of April of 2016(not included).

3.2.6 Weekly Expiration

In order to use this type of expiration use leaderboard names like `cario-sisters-year2016week21`. This means a leaderboard ranging from the 23rd of May of 2016 to the 30th of May of 2016(not included).

This mode is a little odd as it uses week numbers and Week 1 does not start in the first of january. For more information about week numbers, refer to this page.

CHAPTER 4

Podium API

4.1 Healthcheck Routes

4.1.1 Healthcheck

`GET /healthcheck`

Validates that the app is still up, including the connection to Redis.

- Success Response
  - Code: `200`
  - Content:

```
"WORKING"
```

- Error Response

  It will return an error if it failed to connect to Redis.
  - Code: `500`
  - Content:

```
"<error-details>"
```

4.2 Status Routes

4.2.1 Status

`GET /status`

Returns statistics on the health of Podium.

- Success Response
  - Code: `200`
  - Content:

```
{
  "app": {
    "errorRate": [float]        // Exponentially Weighted Moving Average␣
↪Error Rate
  },
}
```

4.3 Leaderboard Routes

4.3.1 Create or Update a Member Score

`PUT /l/:leaderboardID/members/:memberPublicID/score`

optional query string

- prevRank=[true|false]
  - if set to true, it will also return the previous rank of the player in the leaderboard, -1 if the player didn't exist in the leaderboard
  - e.g. `PUT /l/:leaderboardID/members/:memberPublicID/score?prevRank=true`
  - defaults to "false"
- scoreTTL=[integer]
  - if set, the score of the player will be expired from the leaderboard past [integer] seconds if it does not update it within this interval
  - e.g. `PUT /l/:leaderboardID/members/:memberPublicID/score?scoreTTL=100`
  - defaults to none (the score will never expire)

Atomically creates a new member within a leaderboard or if member already exists in leaderboard, update their score.

Leaderboard ID should be a valid leaderboard name and memberPublicID should be a unique identifier for the member associated with the score.

- Payload

```
{
  "score":      [integer]  // Integer representing member score
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "member": {
    "publicID":     [string]  // member public id
    "score":        [int]     // member updated score
    "rank":         [int]     // member current rank in leaderboard
    "previousRank": [int]     // the previous rank of the player in the␣
↪leaderboard, if requests
    "expireAt":     [int]     // unix timestamp of when the score will be␣
↪expired, if scoreTTL is sent
  }
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.2 Create or Update many Members Score

`PUT /l/:leaderboardID/scores`

optional query string

- prevRank=[true|false]
  - if set to true, it will also return the previous rank of the player in the leaderboard, -1 if the player didn't exist in the leaderboard
  - e.g. `PUT /l/:leaderboardID/scores?prevRank=true`
  - defaults to "false"
- scoreTTL=[integer]
  - if set, the score of the player will be expired from the leaderboard past [integer] seconds if it does not update it within this interval
  - e.g. `PUT /l/:leaderboardID/scores?scoreTTL=100`
  - defaults to none (the score will never expire)

Atomically creates many new members within a leaderboard or if some members already exists in leaderboard, update their scores.

Leaderboard ID should be a valid leaderboard name and publicID should be a unique identifier for the member associated with the score.

- Payload

```
{
  "members": [{
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
    }, ...]
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "members": [{
    "publicID":     [string]  // member public id
    "score":        [int]     // member updated score
    "rank":         [int]     // member current rank in leaderboard
    "previousRank": [int]     // the previous rank of the player in the␣
↪leaderboard, if requests
    "expireAt":     [int]     // unix timestamp of when the score will be␣
↪expired, if scoreTTL is sent
  }, ...]
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.3 Increment a Member Score

`PATCH /l/:leaderboardID/members/:memberPublicID/score`

optional query string

- scoreTTL=[integer]
  - if set, the score of the player will be expired from the leaderboard past [integer] seconds if it does not update it within this interval
  - e.g. `PUT /l/:leaderboardID/members/:memberPublicID/score?scoreTTL=100`
  - defaults to none (the score will never expire)

Atomically creates a new member within a leaderboard with the given increment as score. If member already exists in leaderboard just increment their score.

Leaderboard ID should be a valid leaderboard name and memberPublicID should be a unique identifier for the member associated with the score.

WARNING: Incrementing a member score by 0 is not a valid operation and will return a 400 Bad Request result.

- Payload

```
{
  "increment":      [integer]  // Integer representing increment in member score
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "member": {
    "publicID": [string]  // member public id
    "score":    [int]     // member updated score
    "rank":     [int]     // member current rank in leaderboard
    "expireAt": [int]     // unix timestamp of when the score will be expired,
↪ if scoreTTL is sent
  }
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.4 Remove a leaderboard

`DELETE /l/:leaderboardID`

Remove the entire leaderboard from Podium.

WARNING: This operation cannot be undone and all the information in the leaderboard will be destroyed.

`leaderboardID` should be a valid leaderboard name.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.5 Get a member score and rank

`GET /l/:leaderboardID/members/:memberPublicID`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/members/:memberPublicID?order=asc`
  - defaults to "desc"
- scoreTTL=[true|false]
  - if set to true, will return the member's score expiration unix timestamp
  - e.g. `GET /l/:leaderboardID/members/:memberPublicID?scoreTTL=true`
  - defaults to "false"

Gets a member score and rank within a leaderboard.

Leaderboard ID should be a valid leaderboard name and memberPublicID should be a unique identifier for the desired member.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string]  // member public id
  "score":    [int]     // member updated score
  "rank":     [int]     // member current rank in leaderboard
  "expireAt": [int]     // unix timestamp of when the member's score will be␣
↪erased (only if scoreTTL is true)
}
```

- Error Response

  It will return an error if the member is not found.

  - Code: `404`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.6 Get multiple member scores and rank

`GET /l/:leaderboardID/members?ids=publicIDcsv`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/members?ids=publicIDcsv?order=asc`
  - defaults to "desc"
- scoreTTL=[true|false]
  - if set to true, will return the member's score expiration unix timestamp
  - e.g. `GET /l/:leaderboardID/members?ids=publicIDcsv?scoreTTL=true`
  - defaults to "false"

Gets multiple members' score and ranks within a leaderboard.

If any public IDs are not found, they will be returned in the `notFound` list in the response. This is so a list of all the desired members (i.e.: player's friends) can be retrieved and only the ones in the leaderboard get returned.

Leaderboard ID should be a valid leaderboard name and publicIDcsv should be a comma-separated list of the desired members Public IDs.

- Success Response
  - Code: `200`
  - Content:

```
{
  "members": [
    {
      "publicID": [string]    // member public id
      "rank":     [int]       // member rank in the specific leaderboard
      "position": [int]       // member rank for all members returned in this
↪request
      "score":    [int]       // member score in the leaderboard
      "expireAt": [int]       // unix timestamp of when the member's score
↪will be erased (only if scoreTTL is true)
    }
  ],
  "notFound": [
    "[string]"                // list of public ids that were not found in
↪the leaderboard
  ],
  "success": true
}
```

- Error Response

  It will return an error if a list of member ids is not supplied.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.7 Remove members from leaderboard

`DELETE /l/:leaderboardID/members?ids=memberPublicID1,memberPublicID2,...`

Removes specified members from leaderboard. If a member is not in leaderboard, do nothing.

Leaderboard ID should be a valid leaderboard name and ids should be a list of unique identifier for the members being removed, separated by commas.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.8 Get a member score and rank in many leaderboards

`GET /m/:memberPublicID/scores?leaderboardIds=leaderboard1,leaderboard2,...`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /m/:memberPublicID/scores?leaderboardIds=leaderboard1,leaderboard2,...?order=asc`
  - defaults to "desc"
- scoreTTL=[true|false]
  - if set to true, will return the member's score expiration unix timestamp
  - e.g. `GET /m/:memberPublicID/scores?leaderboardIds=leaderboard1,leaderboard2,...?scoreTTL=true`
  - defaults to "false"

Get a member score and rank within many leaderboards.

Leaderboard Ids should be valid leaderboard names separated by commas.

- Sucess Response
  - Code: 200
  - Content:

```
{
  "scores": [
    {
      "leaderboardID": "teste",
      "rank": 1,
      "score": 100,
      "expireAt": [int]     // unix timestamp of when the member's score will␣
↪be erased (only if scoreTTL is true)
    },
    {
```

(continues on next page)

(continued from previous page)

```
      "leaderboardID": "teste2",
      "rank": 1,
      "score": 100,
      "expireAt": [int]     // unix timestamp of when the member's score will
↪be erased (only if scoreTTL is true)
    }
  ],
  "success": true
}
```

- Error Response
  - Code: 500
  - Content:

```
{
  "reason": "Could not find data for member teste3 in leaderboard teste3.",
  "success": false
}
```

4.3.9 Get a member rank

`GET /l/:leaderboardID/members/:memberPublicID/rank`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/members/:memberPublicID/rank?order=asc`
  - defaults to "desc"

Gets a member rank within a leaderboard.

Leaderboard ID should be a valid leaderboard name and memberPublicID should be a unique identifier for the desired member.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string]  // member public id
  "rank":     [int],    // member current rank in leaderboard
}
```

- Error Response

  It will return an error if the member is not found.
  - Code: `404`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

- Code: `500`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.10 Get members around a member

`GET /l/:leaderboardID/members/:memberPublicID/around?pageSize=10`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/members/:memberPublicID/around?pageSize=10?order=asc`
  - defaults to "desc"
- getLastIfNotFound=[true|false]
  - if set to true, will treat members not in ranking as being in last position
  - if set to false, will return 404 when the member is not in the ranking
  - e.g. `GET /l/:leaderboardID/members/:memberPublicID/around?getLastIfNotFound=true`
  - defaults to "false"

Gets a list of members with ranking around that of the specified member within a leaderboard.

The `pageSize` querystring parameter specifies the number of members that will be returned from this operation. This means that `pageSize/2` members will be above the specified member and the other `pageSize/2` will be below.

Podium will compensate if no more members can be found above or below (first or last member in the leaderboard ranking) to ensure that the desired number of members is returned (up to the number of members in the leaderboard).

Leaderboard ID should be a valid leaderboard name and memberPublicID should be a unique identifier for the desired member.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "members": [
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    //...
  ],
}
```

- Error Response

  It will return an error if the member is not found.

  - Code: `404`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.11 Get members around a score

`GET /l/:leaderboardID/scores/:score/around?pageSize=10`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/scores/:score/around?pageSize=10?order=asc`
  - defaults to "desc"

Gets a list of members with score around that of the specified specified in the request. If the `score` parameter falls outside the leaderboard [minScore, maxScore], it will return the bottom/top rank members in the leaderboard, respectively.

The `pageSize` querystring parameter specifies the number of members that will be returned from this operation. That means there will be around `pageSize/2` (+-1) members with score above the specified score, and `pageSize/2`(+-1) with score below.

Podium will compensate if no more members can be found above or below (first or last member in the leaderboard ranking) to ensure that the desired number of members is returned (up to the number of members in the leaderboard).

Leaderboard ID should be a valid leaderboard name and `score` should be a valid number.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "members": [
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    //...
  ],
}
```

- Error Response

  It will return an error if the leaderboard is not found or the request has invalid parameters.
  - Code: `404`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.12 Get the number of members in a leaderboard

`GET /l/:leaderboardID/members-count/`

Gets the number of members in a leaderboard.

Leaderboard ID should be a valid leaderboard name.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "count": [int],      // number of members in leaderboard
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.13 Get the top N members in a leaderboard (by page)

`GET /l/:leaderboardID/top/:pageNumber?pageSize=:pageSize`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/top/:pageNumber?pageSize=:pageSize?order=asc`
  - defaults to "desc"

Gets the top N members in a leaderboard, by page.

`leaderboardID` should be a valid leaderboard name, `pageNumber` is the current page you are looking for and `pageSize` is the number of members per page that will be returned.

This means that if you want the top 20 members, you'll call `/l/my-leaderboard/top/1?pageSize=20` for the first 20, `/l/my-leaderboard/top/2?pageSize=20` for members 21-40 and so on.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "members": [
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    //...
  ]
}
```

- Error Response
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.3.14 Get the top x% members in a leaderboard

`GET /l/:leaderboardID/top-percent/:percentage`

optional query string

- order=[asc|desc]
  - if set to asc, will treat the ranking with ascending scores (less is best)
  - e.g. `GET /l/:leaderboardID/top-percent/:percentage?order=asc`
  - defaults to "desc"

Gets the top x% members in a leaderboard.

`leaderboardID` should be a valid leaderboard name, `percentage` is the % of members you want to return.

The number of members is bound by the configuration `api.maxReturnedMembers`, that defaults to 2000 members.

- Success Response

  - Code: `200`
  - Content:

```
{
  "success": true,
  "members": [
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    {
      "publicID": [string]  // member public id
      "score":    [int],    // member updated score
      "rank":     [int],    // member current rank in leaderboard
    },
    //...
  ]
}
```

- Error Response

  If the percentage is not a valid integer between 1 and 100, you'll get a 400.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

4.4 Member Routes

4.4.1 Create or update score for a member in several leaderboards

`PUT /m/:memberPublicID/scores`

optional query string

- prevRank=[true|false]
  - if set to true, it will also return the previous rank of the player in the leaderboard, -1 if the player didn't exist in the leaderboard
  - e.g. `PUT /l/:leaderboardID/members/:memberPublicID/score?prevRank=true`

  - defaults to "false"

- scoreTTL=[integer]
  - if set, the score of the player will be expired from the leaderboards past [integer] seconds if it does not update it within this interval
  - e.g. `PUT /l/:leaderboardID/members/:memberPublicID/score?scoreTTL=100`
  - defaults to none (the score will never expire

Atomically creates a new member within many leaderboard or if member already exists in each leaderboard, updates their score.

`memberPublicID` should be a unique identifier for the member associated with the score. Each `leaderboardID` should be a valid leaderboard name.

- Payload

```
{
  "score": [integer],                       // Integer representing member score
  "leaderboards": [array of leaderboardID]  // List of all leaderboards to update
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "scores": [
    {
      "leaderboardID": [string] // leaderboard where this score was set
      "publicID": [string]      // member public id
      "score":    [int],        // member updated score
      "rank":     [int],        // member current rank in leaderboard
      "previousRank": [int]     // the previous rank of the player in the␣
↪leaderboard, if requests
    },
    {
      "leaderboardID": [string] // leaderboard where this score was set
      "publicID": [string]      // member public id
      "score":    [int],        // member updated score
      "rank":     [int],        // member current rank in leaderboard
      "previousRank": [int]     // the previous rank of the player in the␣
↪leaderboard, if requests
    },
    //...
  ]
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

- Code: `500`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

CHAPTER 5

Library

For detailed information, check our reference. All examples below have imported the leaderboard module using:

```
import "github.com/topfreegames/podium/leaderboard"
```

5.1 Creating the client

```
const host = "localhost"
const port = 1234
const password = ""
const db = 0
const connectionTimeout = 200

leaderboards, err := leaderboard.NewClient(host, port, password, db,␣
↪connectionTimeout)
```

5.2 Creating, updating or retrieving member scores

```
const leaderboardID = "lbID"
const playerID = "playerID"
const score = 100
const wantToKnowPreviousRank = false //do I want to receive also the previous rank on␣
↪the user?
const scoreTTL = "100"               //how many seconds my score will be kept on the␣
↪leaderboard

member, err := leaderboards.SetMemberScore(context.Background(), leaderboardID,␣
↪playerID, score, wantToKnowPreviousRank, scoreTTL)
if err != nil {
    return err
```

(continues on next page)

(continued from previous page)

```
}

playerPrinter := func(publicID string, score int64, rank int) {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", publicID, score, rank)
}

playerPrinter(member.PublicID, member.Score, member.Rank)

const order = "desc"     //if set to asc, will treat the ranking with ascending␣
↪scores (less is best)
const includeTTL = false //if set to true, will return the member's score expiration␣
↪unix timestamp
member, err = leaderboards.GetMember(context.Background(), leaderboardID, playerID,␣
↪order, includeTTL)
if err != nil {
    return err
}

playerPrinter(member.PublicID, member.Score, member.Rank)
```

5.3 Setting and getting member scores in bulk

```
const leaderboardID = "lbID"

players := leaderboard.Members{
    &leaderboard.Member{Score: 1000, PublicID: "playerA"},
    &leaderboard.Member{Score: 2000, PublicID: "playerB"},
}

err := leaderboards.SetMembersScore(context.Background(), leaderboardID, players,␣
↪false, "")
if err != nil {
    return err
}

const order = "desc"     //if set to asc, will treat the ranking with ascending␣
↪scores (less is best)
const includeTTL = false //if set to true, will return the member's score expiration␣
↪unix timestamp
members, err := leaderboards.GetMembers(context.Background(), leaderboardID, []string{
↪"playerA", "playerB"}, order, includeTTL)

for _, member := range members {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", member.PublicID, member.Score,␣
↪member.Rank)
}
```

5.4 Retrieving leaderboard leaders

```
const leaderboardID = "myleaderboardID"
```

(continues on next page)

(continued from previous page)

```
players := leaderboard.Members{
    &leaderboard.Member{Score: 10, PublicID: "player1"},
    &leaderboard.Member{Score: 20, PublicID: "player2"},
}

err = leaderboards.SetMembersScore(context.Background(), leaderboardID, players,␣
↪false, "")
if err != nil {
    log.Fatalf("leaderboards.SetMembersScore failed: %v", err)
}
const pageSize = 10
const pageIdx = 1 //starts at 1
leaders, err := leaderboards.GetLeaders(context.Background(), leaderboardID, pageSize,
↪ pageIdx, "desc")
if err != nil {
    log.Fatalf("leaderboards.GetLeaders failed: %v", err)
}

for _, player := range leaders {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", player.PublicID, player.Score,␣
↪player.Rank)
}
```

5.5 Incrementing player scores

```
const leaderboardID = "lbID"
const playerID = "playerA"
const scoreIncrement = 500
const scoreTTL = ""
member, err := leaderboards.IncrementMemberScore(context.Background(), leaderboardID,␣
↪playerID, scoreIncrement,
    scoreTTL)
if err != nil {
    return err
}

fmt.Printf("Player(id: %s, score: %d rank: %d)\n", member.PublicID, member.Score,␣
↪member.Rank)
```

5.6 Number of players on a leaderboard

```
const leaderboardID = "lbID"
count, err := leaderboards.TotalMembers(context.Background(), leaderboardID)
if err != nil {
    return err
}
fmt.Printf("Total number of players on leaderboard %s: %d\n", leaderboardID, count)
```

5.7 Removing players from a leaderboard

```
const leaderboardID = "lbID"
const playerIdToRemove = "playerID"
err := leaderboards.RemoveMember(context.Background(), leaderboardID,␣
↪playerIdToRemove) //removing a single player
if err != nil {
    return err
}

playerIDsToRemove := make([]interface{}, 2)
playerIDsToRemove[0] = "playerA"
playerIDsToRemove[1] = "playerB"

err = leaderboards.RemoveMembers(context.Background(), leaderboardID,␣
↪playerIDsToRemove) //removing multiple players
if err != nil {
    return err
}
```

5.8 Total pages of a leaderboard

```
const leaderboardID = "lbID"
const pageSize = 10
pageCount, err := leaderboards.TotalPages(context.Background(), leaderboardID,␣
↪pageSize)
if err != nil {
    return err
}
fmt.Printf("total pages: %d\n", pageCount)
```

5.9 Getting players around a player

```
const leaderboardID = "lbID"
const pageSize = 10
const getLastIfNotFound = false //if set to true, will treat members not in ranking␣
↪as being in last position
//if set to false, will return 404 when the member is not in the ranking
const order = "asc"
members, err := leaderboards.GetAroundMe(context.Background(), leaderboardID,␣
↪pageSize, "playerID",
    order, getLastIfNotFound)
if err != nil {
    return err
}
for _, member := range members {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", member.PublicID, member.Score,␣
↪member.Rank)
}
```

5.10 Getting players around a score

```
const leaderboardID = "lbID"
const pageSize = 10
const score = 1500
const order = "desc"

members, err := leaderboards.GetAroundScore(context.Background(), leaderboardID,␣
↪pageSize, score, order)
if err != nil {
    return err
}
for _, member := range members {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", member.PublicID, member.Score,␣
↪member.Rank)
}
```

5.11 Top percentage of a leaderboard

```
const leaderboardID = "lbID"
const pageSize = 10
const percent = 10
const maxMembersToReturn = 100
const order = "asc"
top10, err := leaderboards.GetTopPercentage(context.Background(), leaderboardID,␣
↪pageSize, percent,
    maxMembersToReturn, order)
if err != nil {
    return err
}
for _, member := range top10 {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", member.PublicID, member.Score,␣
↪member.Rank)
}
```

5.12 Getting members in a range

```
const leaderboardID = "lbID"
const startOffset = 0
const endOffset = 10
const order = "asc"
members, err := leaderboards.GetMembersByRange(context.Background(), leaderboardID,␣
↪startOffset, endOffset, order)
if err != nil {
    return err
}
for _, member := range members {
    fmt.Printf("Player(id: %s, score: %d rank: %d)\n", member.PublicID, member.Score,␣
↪member.Rank)
}
```

5.13 Removing a leaderboard

```
const leaderboardID = "lbID"
err := leaderboards.RemoveLeaderboard(context.Background(), leaderboardID)
```

CHAPTER 6

Podium's Benchmarks

You can see podium's benchmarks in our CI server as they get run with every build.

6.1 Running Benchmarks

If you want to run your own benchmarks, just download the project, and run:

```
$ make bench-redis bench-podium-app bench-run
```

6.2 Generating test data

If you want to run your perf tests against a database with more volume of data, just run this command, instead:

```
$ make bench-redis bench-seed bench-podium-app bench-run
```

Warning: This will take a long time running.

6.3 Results

The results should be similar to these:

```
BenchmarkSetMemberScore-8                          30000        284307 ns/op       0.
↪32 MB/s        5635 B/op         81 allocs/op
BenchmarkSetMembersScore-8                          5000       1288746 ns/op       3.
↪01 MB/s       51452 B/op        583 allocs/op
BenchmarkIncrementMemberScore-8                    30000        288306 ns/op       0.
↪32 MB/s        5651 B/op         81 allocs/op
BenchmarkRemoveMember-8                            50000        202398 ns/op       0.
↪08 MB/s        4648 B/op         68 allocs/op
```

(continues on next page)

(continued from previous page)

```
BenchmarkGetMember-8                                        30000        215802 ns/op        0.
↪33 MB/s        4728 B/op         68 allocs/op
BenchmarkGetMemberRank-8                                    50000        201367 ns/op        0.
↪28 MB/s        4712 B/op         68 allocs/op
BenchmarkGetAroundMember-8                                  20000        397849 ns/op        3.
↪14 MB/s        8703 B/op         69 allocs/op
BenchmarkGetTotalMembers-8                                  50000        192860 ns/op        0.
↪16 MB/s        4536 B/op         64 allocs/op
BenchmarkGetTopMembers-8                                    20000        306186 ns/op        3.
↪85 MB/s        8585 B/op         66 allocs/op
BenchmarkGetTopPercentage-8                                  1000      10011287 ns/op       11.
↪88 MB/s      510300 B/op         77 allocs/op
BenchmarkSetMemberScoreForSeveralLeaderboards-8              1000     106129629 ns/op        1.
↪03 MB/s      516103 B/op         98 allocs/op
BenchmarkGetMembers-8                                        2000       3931289 ns/op        9.
↪13 MB/s      243755 B/op         76 allocs/op
```