GenPare Backend API

This document is a comprehensive list of all GenPare backend features.

Table of contents

1. Document format

Every possible call to the backend has an entry in this document. Said entry is divided into the path, HTTP method, request data (if any), response data (if any) and possible errors that might be thrown.

1.1. Response codes

If a call specifies the response data to be "none", assume that the status code returned is `204 No Content` on success. Otherwise, assume it to be `200 OK` on success, or any of the specified error codes in case of an unsuccessful call.

1.2. Request and response data

Request and response data are always to be transmitted in the JSON format. A format of the required or supplied JSON object is given by specifying the type of the field in question.

1.3. Types

If a field specifies `null` as its type, the field must be present, but it isn't allowed to have a value other than `null`. Failure to do this will result in an error. This is an idiosyncrasy of GSON and hence is subject to change (with prior notice).

If a type has a question mark `?` appended to it, it must be present, but can be `null`.

1.3.1 Date

The type `Date` is to be formatted as `yyyy-mm-dd`.

1.3.2. IntRange

Specifies a range of numbers.

```JSON
{
  "min": Int,
  "max": Int
}
```

1.3.3. AnonymizedResult

A JSON object.

```
{
  "age": IntRange,
  "salary": IntRange,
  "gender": Gender,
  "jobTitle": String,
  "state": State,
  "levelOfEducation": LevelOfEducation
}
```

JSON

1.4. Enumerations

There are some enumerations specified as the data type. These are always transmitted as a string. These must always be transmitted in all UPPERCASE and ***verbatim***.

1.4.1. Gender

- FEMALE
- MALE
- DIVERSE

1.4.2. State

- BADEN_WURTTEMBERG
- BAVARIA
- BERLIN
- BRANDENBURG
- BREMEN
- HAMBURG
- HESSE
- LOWER_SAXONY
- MECKLENBURG_WESTERN_POMERANIA
- NORTH_RHINE_WESTPHALIA
- RHINELAND_PALATINATE
- SAARLAND
- SAXONY
- SAXONY_ANHALT
- SCHLESWIG_HOLSTEIN
- THURINGIA

1.4.3. LevelOfEducation

- NONE
- HAUPTSCHULE
- MITTLERE_REIFE
- ABITUR
- FACHHOCHSCHULREIFE
- DIPLOM
- MAGISTER
- BACHELOR
- MASTER
- DOKTOR
- GESELLE
- MEISTER
- EINFACHER_DIENST
- MITTLERER_DIENST
- GEHOBENER_DIENST
- HOEHERER_DIENST

1.5. Filters

1.5.1. Age filter

Filters the data set by age. (`min` ≤ age ≤ `max`)

```
{
  "name": "age",
  "min": Int,
  "max": Int
}
```

JSON

1.5.2. Job title filter

Filters the data set by job title.

```
{
  "name": "jobTitle",
  "desiredJobTitle": String
}
```

JSON

1.5.3. Level of education filter

Filters the data set by level of education.

```
{
  "name": "levelOfEducation",
  "desiredLevelOfEducation": LevelOfEducation
}
```

JSON

1.5.4. Salary filter

Filters the data set by salary. (`min` ≤ salary < `max`)

```
{
  "name": "salary",
  "min": Int,
  "max": Int
}
```

JSON

1.5.5. State filter

Filters the data set by state.

```
{
  "name": "state",
  "desiredState": State
}
```

JSON

1.6. Result transformers

1.6.1. Average result transformer

Calculates the average of the salaries for every gender, female, male and diverse. If there is no data for a specific group, the resulting field will be `null`.

Format:

```
{
  "name": "average"
}
```
JSON

Response object:

```
{
  "resultOf": "average",
  "averageTotal": Int?,
  "averageFemale": Int?,
  "averageMale": Int?,
  "averageDiverse": Int?
}
```
JSON

1.6.2. List result transformer

Returns an anonymized list of all filtered results. Age and salary information is given as a range.

Format:

```
{
  "name": "list"
}
```
JSON

Response object:

```
{
  "results": List<AnonymizedResult>
}
```
JSON

2. Member management

Member login is handled by Auth0 in the frontend. The backend still needs information about an user to function properly. Member management handles these interactions.

2.1. Member creation

Must be called when a member is being registered.

Path: `/members`
HTTP method: `POST`
Request data:

```
{
  "id": null,
  "email": String,
  "name": String,
  "birthdate": Date,
  "gender": Gender
}
```
JSON

Response data:

```
{
  "id": Long,
  "email": String,
  "name": String,
  "birthdate": Date,
  "gender": Gender
}
```
JSON

Possible errors:

- The ID field is set to anything other than null: `400 Bad Request`
- A member registers themselves with a known email address: `409 Conflict`

2.2. Member deletion

Must be called when a member is deleted.

Path: `/members`
HTTP method: `DELETE`
Request data:

```
{
  "email": String,
  "sessionId": Long
}
```

JSON

Response data: none
Possible errors:

- Unknown email address: `204 No Content` (subject to change in future releases)
- Invalid session id: `403 Unauthorized`

2.3. Member info fetch

Can be called to get member data.

Path: `/members`
HTTP method: `GET`
Request data: `?sessionId=<Long>`
Response data:

```
{
  "id": Long,
  "email": String,
  "name": String,
  "birthdate": Date,
  "gender": Gender
}
```

JSON

Possible errors:

- Missing session id: `400 Bad Request`
- Invalid session id: `404 Not Found`

2.4. Member name change

Must be called when a member requests a name change.

Path: `/members`
HTTP method: `PATCH`
Request data:

```
{
  "name": String,
  "sessionId": Long
}
```
JSON

Response data: none
Possible errors:

- Invalid/unknown session id: `403 Unauthorized`

2.5. Session ID fetching

Must be called to receive a session id for a member.

Path: `/members/session`
HTTP method: `GET`
Request data: `?email=<String>`
Response data:

```
{
  "sessionId": Long
}
```
JSON

Possible errors:

- Unknown email address: `404 Not Found`

2.6. Session ID invalidation

Must be called when a member logs out. Invalidates their session id.

Path: `/members/session`
HTTP method: `DELETE`
Request data: `?email=<String>`
Response data: none
Possible errors: none

3. Data management

Data management is concerned with every aspect of salary information, be it entering, changing or comparing of such.

3.1. Salary comparison

Must be called when a member requests salary comparison data. Due to the complexity of this call, it will be described in more detail after the information block.

Path: `/salary`
HTTP method: `POST`
Request data:

```
{
  "filters": List<Filter>,
  "resultTransformers": List<ResultTransformer>
}
```
JSON

Response data:

```
{
  "results": List<Result>
}
```
JSON

Possible errors:

- Empty filter list: `400 Bad Request`
- Empty result transformer list: `400 Bad Request`

3.1.1. Filter explanation

The chief purpose of GenPare is to provide a framework of salary comparisons. Since comparison can only be sensibly performed on a subset of the total known salaries, filters are employed to reduce the size of the dataset taken into consideration. It is possible to specify more than one filter at a time, but there can never be a request without any filters in place.

For example, if a user wants to only compare to salaries of people ages 20 to 40, the frontend would send this filter object:

```
{
  "name": "age",
  "min": 20,
  "max": 40
}
```

JSON

A list of available filters can be seen in 1.5. Filters.

3.1.2. Result transformer explanation

Filters only limit the view on the data set. To display the resulting data set to the user sensibly, result transformers must be employed. They transform the data set into another data set of different characteristics, for example, one result transformer calculates the averages of the salaries. This approach was chosen in order to more efficiently use the available resources like CPU power. Each result transformer has its own response object.

A list of available result transformers and their response objects can be seen in 1.6. Result transformers.

3.2. List known job titles

Can be called to find out all known job titles.

Path: `/salary`
HTTP method: `GET`
Request data: none
Response data:

```
List<String>
```

JSON

3.3. Fetch salary entry

Can be called to find out salary data for a member (self).

Path: `/salary`
HTTP method: `GET`
Request data: `?sessionId=<Long>`
Response data:

```
{
  "salary": Int,
  "jobTitle": String,
  "state": State,
  "levelOfEducation": LevelOfEducation
}
```

JSON

3.4. New salary entry

Must be called when a member inputs their salary data. The job title can't be longer than 63 characters.

Path: `/salary`
HTTP method: `PUT`
Request data:

```
{
  "sessionId": Long,
  "salary": Int,
  "jobTitle": String,
  "state": State,
  "levelOfEducation": LevelOfEducation
}
```
JSON

Response data:

```
{
  "sessionId": Long,
  "salary": Int,
  "jobTitle": String,
  "state": State,
  "levelOfEducation": LevelOfEducation
}
```
JSON

Possible errors:

- Invalid/unknown session id: `404 Not Found`
- Salary information exists for user: `409 Conflict`
- Job title exceeds 63 characters: `400 Bad Request`

3.5. Modify salary entry

Must be called when a member changes their salary information. The job title can't be longer than 63 characters. If a field is `null`, its value won't be changed.

Path: `/salary`
HTTP method: `PATCH`
Request data:

```
{
  "sessionId": Long,
  "salary": Int?,
  "jobTitle": String,
  "state": State?,
  "levelOfEducation": LevelOfEducation?
}
```

JSON

Response data:

```
{
  "sessionId": Long,
  "salary": Int,
  "jobTitle": String,
  "state": State,
  "levelOfEducation": LevelOfEducation
}
```

JSON

Possible errors:

- Invalid/unknown session id: `404 Not Found`
- Salary information doesn't exist yet for this member: `404 Not Found`
- Job title exceeds 63 characters: `400 Bad Request`