```
┌─────────────────────────────┐
│     API DOCUMENTATION       │
│   MAJOR VOTING SYSTEM       │
└─────────────────────────────┘
```

SIGN IN TO THE VOTING SYSTEM

═══

[GET]	/v1/token

───

BODY
DATA
```
JSON
{
    "voter_id":  3,
    "password": "1234"
}
```

───

STATUS
RESPONSE
DATA
```
200 OK
JSON
{
    "access_token":  "jAHJf76h.jkAfWSjk98.KLJAdfdh",
    "expires_in":      500,
    "refresh_token": "jAHJf76h.jkAfWSjk98.KLJAdfdh"
}
```

STATUS   400 BAD REQUEST

STATUS   401 UNAUTHORIZED
NOTE   Wrong credentials

STATUS   406 NOT ACCEPTABLE
NOTE   Existing session for this voter

───

The voter enters their ID and password to get into the system. The
server answers with an *access token* that has to be included in all the
following calls to the API, as a *bearer* authentication header.

The access token is a *JWT token* containing the voter ID and the expiry
date of the token. It is *signed* and *encrypted* by the server, so it is
useless on the client side. Nevertheless, the client can call /v1/whoami
with the token in the headers to obtain the voter information.

The response also includes the expiry date of the token, as well as a
refresh token, that is meant to be used in case the client wants to
remain connected for longer than allowed by the expiry date. In this
case, the client must call the PATCH version of the current endpoint
instead.

───

Relevant documentation :

 - Bearer authentication
   https://datatracker.ietf.org/doc/html/rfc6750

 - JWT tokens
   https://jwt.io

REFRESH THE ACCESS TOKEN

═══

[PATCH]   /v1/token/refresh

───

HEADERS   Authorization: Bearer *<access token>*

BODY   JSON
DATA   { "refresh_token": "jAHJf76h.jkAfWSjk98.KLJAdfdh" }

───

STATUS   200 OK
RESPONSE   JSON
DATA   {
                     "access_token": "jAHJf76h.jkAfWSjk98.KLJAdfdh",
                     "expires_in":   500,
                     "refresh_token": "jAHJf76h.jkAfWSjk98.KLJAdfdh"
                 }

STATUS   400 BAD REQUEST

STATUS   401 UNAUTHORIZED
NOTE   Missing or wrong token(s)

───

A given voter can only have *one* session opened at a time. Trying to open
a new one will result in an error. Instead, the client can use the
current endpoint to ask to replace the active token of the current
session with a fresh one, to extend it and remain logged in.

They must include both the *access token* and the *refresh token*, and the
server will check that they belong to the same session. Indeed, both
tokens include the voter ID, and if they differ, the server will reject
the request.

GET INFORMATION ABOUT THE CURRENT USER

[GET]	/v1/whoami

HEADERS	Authorization: Bearer *<access token>*

STATUS	200 OK
RESPONSE	JSON
DATA	{ "voter_name": "Jean Dupont" }
STATUS	401 UNAUTHORIZED
NOTE	No valid access token

This endpoint can be used by the client to know more about the current voter logged in.

LOGOUT

[DELETE]	/v1/token

HEADERS	Authorization: Bearer *<access token>*

STATUS	200 OK

STATUS	400 BAD REQUEST

This endpoint must be used when the client wants to log out from the voting system. This makes sure the tokens used in the session are now invalid.

```
┌─────────────────────────────┐
│  API DOCUMENTATION          │
│  MAJOR VOTING SYSTEM        │
└─────────────────────────────┘
```

GET INFORMATION ON THE ELECTIONS OF THE CURRENT VOTER

[GET]	/v1/elections

HEADERS	Authorization: Bearer *<access token>*

STATUS	200 OK
RESPONSE	JSON ARRAY
DATA	[

```
              {  "id": 1, "name": "Election 1", "is_running": true  },
              {  "id": 2, "name": "Election 2", "is_running": false }
            ]
```

STATUS	401 UNAUTHORIZED
NOTE	No valid access token

Through this endpoint, the voter can read which elections they can (or could) vote for, that are still in the system.

GET DETAILED INFORMATION ABOUT AN ELECTION

[GET]	/v1/elections/:id

HEADERS	Authorization: Bearer *<access token>*

STATUS	400 BAD REQUEST
NOTE	Wrong ID format or election does not exist

STATUS	401 UNAUTHORIZED
NOTE	No valid access token

STATUS	403 FORBIDDEN
NOTE	The current voter does not take part in this election

STATUS	200 OK
RESPONSE	JSON

[...]

DATA
```json
{
    "name": "Election 9",
    "is_running": true,
    "parties": [
      { "id": 1, "name": "BFM TV",   "colour": "#0000FF" },
      { "id": 2, "name": "Backseat", "colour": "#FF0000" },
      { "id": 3, "name": "CNEWS",    "colour": "#000000" }
    ],
    "candidates": [
      { "id": 1, "name": "Nicolas Doze",    "party": 1 },
      { "id": 2, "name": "Jean Massiet",    "party": 2 },
      { "id": 3, "name": "Christine Kelly", "party": 3 }
    ]
}
```

This endpoint is used to get more information about the various candidates running for an election and which party they belong to. By making this call, the client knows the identifiers of the candidates and is able to create a ballot.

VOTE FOR AN ELECTION

═══

[POST]	/v1/elections/:id

HEADERS   Authorization: Bearer *<access token>*

BODY   JSON ARRAY
DATA
```json
[
  { "id": 1, "rating": 1 },
  { "id": 2, "rating": 2 },
  { "id": 3, "rating": 3 }
]
```

STATUS   200 OK

STATUS   400 BAD REQUEST
NOTE   Wrong ID format, election does not exist, or wrong format
               for the ballot

STATUS   401 UNAUTHORIZED
NOTE   No valid access token

STATUS   403 FORBIDDEN
NOTE   The current voter does not take part in this election or the
               election is not running anymore

In a *majority judgment* voting system, instead of choosing *one* candidate, the voter gives a *rating* to *every* candidate present in the election.

In our system, the ratings go from 1 to 7 (higher is better) and have the following names :

      1. Terrible   4. Passable   7. Excellent
      2. Bad   5. Sufficient
      3. Inadequate 6. Good

This allows the voter to give a more complex opinion that will be taken into account in the voting system.

If the ballot is incomplete, the *lowest* rating is assumed for all the remaining candidates. For instance, sending an empty ballot amounts to considering all candidates as Terrible.

GET RESULTS OF AN ELECTION
═══

[POST]   /v1/elections/:id/results
───

HEADERS   Authorization: Bearer *<access token>*
───

STATUS   200 OK
RESPONSE JSON
DATA

```
{
  "explanation": "ABC is elected",
  "results": [
    {
      "id": 8,
      "majority_rating": 4,
      "scores": [12.8, 16.3, 9, 37.7, 10.6, 3.5, 10.1]
    },
    {
      "id": 7,
      "majority_rating": 2,
      "scores": [34.8, 26.3, 9, 0.9, 10, 18, 1]
    }
  ]
}
```

STATUS   400 BAD REQUEST
NOTE   Wrong ID format or election does not exist

STATUS   401 UNAUTHORIZED
NOTE   No valid access token

───

At the end of the voting period, each candidate gets their *profile*
computed, *i.e.*, the list of proportions of voters that gave them each
rating. The winner is determined by computing the *majority ratings*.

The *majority rating* of a candidate is the *highest* rating for which
strictly more than half of the voters gave them *at least* this rating. In
other words, if we were to sort ratings in *increasing* order, the
majority rating would be the rating in the *middle*, *i.e.*, rating number K
if there are 2K ratings, or rating number K+1 if there are 2K+1 ratings.
The winner is the candidate with the highest majority rating.

The response of this endpoint includes both the majority ratings and the
scores for each rating in *increasing* order, for each candidate.

  NOTE
  In case of a tie, a more complex tie-breaking computation is carried
  out by the server.

  For each candidate, we compute the proportion of voters who gave them
  a *higher* rating than their majority judgment (their *proponents*) and
  the proportion of voters who gave them a *lower* rating (their
  opponents).

  If the highest value among all these special values is *unique* and it
  is a proportion of opponents, the corresponding candidate is excluded
  from the tie. If it is a proportion of proponents, the candidate is
  the winner of the election.

  If the value is not unique but it is both equal to a proportion of
  proponents and a proportion of opponents, the candidate with this
  proportion of opponents is excluded.

  If the value is not unique and equal to several proportions of
  proponents, all the other candidates are excluded.

  When no further exclusion is possible, the whole process is done once
  again, this time considering the proponents as the voters who gave the
  candidate a higher rating than the *next rating after* the majority
  rating, and the opponents as the voters who gave them a lower rating
  than the *previous rating before* the majority rating.

  This goes on until all the ratings have been exploited. In the case
  where several candidates are still left, it means they had the *exact
  same amount* of people giving them *each* rating, which is very unlikely.

──

```
┌─────────────────────────────┐
│   API DOCUMENTATION         │
│   MAJOR VOTING SYSTEM       │
└─────────────────────────────┘
```

ADD VOTERS TO THE SYSTEM
═══

[POST]   /v1/admin/voters
───

HEADERS   Authorization: Basic *<base64(user:password)>*

BODY   JSON ARRAY
DATA   [
                  { "id": 1, "password": "1234" },
                  { "id": 2, "password": "5678" }
                ]
───

STATUS   400 BAD REQUEST

STATUS   401 UNAUTHORIZED
NOTE   Invalid admin credentials

STATUS   200 OK
───

In this version of the API, the only way to add data to the system other
than votes is to give this data in JSON format to an endpoint reserved
for the administrator.

A basic authentication mode is used, checking the given credentials in a
base64-encoded format against a hardcoded string put into the system on
server startup.

In a later version of the API, we will consider allowing clients to add
voter accounts on their own with a dedicated endpoint.

Relevant documentation :

  - Basic authentication
    https://datatracker.ietf.org/doc/html/rfc7617

CREATE ELECTIONS IN THE SYSTEM

[POST]	/v1/admin/elections

HEADERS	Authorization: Basic *<base64(user:password)>*

BODY
DATA

```
JSON ARRAY
[
  {
    "id": 9,
    "name": "Election 9",
    "parties": [
      { "id": 1, "name": "BFM TV",   "colour": "#0000FF" },
      { "id": 2, "name": "Backseat", "colour": "#FF0000" },
      { "id": 3, "name": "CNEWS",    "colour": "#000000" }
    ],
    "candidates": [
      { "id": 1, "name": "Nicolas Doze",    "party": 1 },
      { "id": 2, "name": "Jean Massiet",    "party": 2 },
      { "id": 3, "name": "Christine Kelly", "party": 3 }
    ]
  },
  ...
]
```

STATUS	200 OK

STATUS	400 BAD REQUEST

STATUS	401 UNAUTHORIZED
NOTE	Invalid admin credentials

Like for voters, adding elections to the system is done by pushing a JSON object into it via the current endpoint.

TERMINATE AN ELECTION

[PATCH]	/v1/elections/:id

HEADERS	Authorization: Basic *<base64-encoded admin user:password>*

STATUS	200 OK

STATUS	401 UNAUTHORIZED
NOTE	Invalid admin credentials

STATUS   406 NOT ACCEPTABLE
NOTE   Election is not running
_____

This endpoint puts an end to a running election.