API Design Document v0.1

Table of Content

Conventions

1. Use nouns but no verbs

Resource	GET (read)	POST (create)	PUT (update)	DELETE
/cars	Return a list of cars	Create a new car	Bulk update of cars	Delete all cars
/cars/1	Return car id = 1	Method not allowed (405)	Update car id = 1	Delete car id = 1

Do not use verbs:

Bad Examples!

```
/getAllCars
/createNewCar
/deleteAllCar
```

2. Use plural nouns

```
/cars    YES
/car     NO!!!!
```

3. Use sub-resources for relations

- `GET /cars/711/drivers/` Returns a list of drivers for car 711.
- `GET /cars/711/drivers/4` Returns driver #4 for car 711.

4. Provide filtering, sorting and paging for collections

Filtering:

- `GET /cars?color=red` Returns a list of red cars.
- `GET /cars?seats<=2` Returns a list of cars with a maximum of 2 seats

Sorting:

Allow ascending and descending sorting over multiple fields.

```
GET /cars?sort=-manufactorer,+model
```

Returns a list of cars sorted by descending manufacturers and ascending models.

Paging:

Do not allow user to retrieve all the data in one request by default. Instead, GET operations, which return a list of requested items, return only the first 25(or another value) items.

URL	Description
/cars	Return the first 25 cars
/cars?limit=10	Return the first 10 cars
/cars?offset=5	Return cars from No.6 to No. 31
/cars?offset=10&limit=5	Return cars from No.6 to No. 10

To page through all available items, use the metadata section of the JSON response to get total number of items. For example, `"total": 77` means there are 77 cars in the database.

```
{
    "meta": {
        "result_set": {
            "count": 2,
            "offset": 0,
            "limit": 2,
            "total": 77
        }
    },
    "data": [
        {
            "id": "uuid1",
            "type": "car",
            "attributes": {
                "model": "Benz"
            }
        },
        {
            "id": "uuid2",
            "type": "car",
            "attributes": {
                "model": "Lamborghini"
            }
        }
    ]
}
```

Note: Send GET request to `/cars?limit=0` to get a json that just contains metadata.

5. Versioning the API

Make the API Version mandatory and do not release an unversioned API. Use a simple ordinal number.

```
/api/v1
```

6. Handle Errors with HTTP status codes

Useful status codes

- 200 – OK – Eyerything is working
- 201 – OK – New resource has been **created**
- 204 – OK – The resource was successfully **deleted**
- 304 – **Not Modified** – The client can use cached data
- 401 – **Unauthorized** – The request requires an user authentication
- 403 – **Forbidden** – The server understood the request, but is refusing it or the access is not allowed.
- 404 – **Not found** – There is no resource behind the URI.

- 500 – **Internal Server Error** – You should avoid this error. If an error occurs in the global catch blog, the stracktrace should be logged and not returned as response.
- 400 – **Bad Request** – The request was invalid or cannot be served. **The exact error should be explained in the error payload.**

Error payloads

All exceptions should be mapped in an error payload. Here is an example how a JSON payload should look like.

```
{
    "errors": [{
        "title": "This is a short human readable msg, MUST not be empty",
        "detail": "This is human readable detail msg, can be empty",
        "status": 404
    }]
}
```

JSON Specifications

Specifications

API

Users

0. Template

Description:

Request:

- URI

```

```

- Body

```
{

}
```

Response:

- Status Code:
- Body

```
{

}
```

Errors:

1. Registration

Description:

Request:

- URI

  ```
  POST /api/v1/users/
  ```

- Body

  ```
  {
      "data": {
          "type": "user",
          "id": "uuid",
          "attributes": {
              "email": "example@xxx.com",
              "password": "xxx"
          }
      }
  }
  ```

Response:

- Status Code: 201
- Body

  ```
  {
      "data": {
          "type": "user",
          "id": "uuid",
          "attributes": {
              "email": "example@xxx.com",
              "password": "xxx"
          }
      }
  }
  ```

Errors:

- Email exists

2. Login

Description:

Request:

- URI

```
POST /api/v1/sessions
```

- Body

```
{
    "data": {
        "attributes": {
            "email": "example@xxx.com",
            "password": "xxx"
        }
    }
}
```

Response:

- Status Code: 200
- Body

```
{
    "data": {
        "attributes": {
           "user_id": "xxx",
           "token": "xxx"
        }
    }
}
```

Errors:

- Bad authentication

3. Get User Profiles

Description:

Request:

- URI

```
GET /api/v1/users/{id}
```

- Body

  EMPTY

Response:

- Status Code: 200
- Body

```
{
    "data": {
        "id": "xxx",
        "type": "user",
        "attributes": {
            "username": "xxx",
            "email": "example@xxx.com",
            "password": "xxx",
            "phone": "1234567890",
            "credit": 100,
            "balance": 100
        }
    }
}
```

Errors:

- No such user

4. Update User Profiles

Description:

- Some infomation such as email, user balance **MUST** not be updated by this way!
- **Only** the fields that appears in the JSON body should be updated!

Request:

- URI

  ```
  PUT /api/v1/users/{id}
  ```

- Body

  ```
  {
      "data": {
          "id": "xxx",
          "type": "user",
          "attributes": {
              "username": "xxx",
              "password": "xxx",
              "phone": "1234567890",
              "credit": 100
          }
      }
  }
  ```

Response:

- Status Code: 200 OK
- Body

  EMPTY

Errors:

Pictures

1. Get a Picture

Description:

Request:

- URI

  ```
  GET /api/pictures/{picture_id}
  ```

- Body

  EMPTY

Response:

- Status Code: 200 OK
- Body

  Binary image file

Errors:

- 404 NOT FOUND

2. Batch Get Pictures

Step 1

Request:

- URI

  ```
  POST /api/v1/batchasks
  ```

- Body

  ```
  {
      "data": [
          {
              "type": "request",
              "attributes": {
                  "method": "GET",
                  "url": "/api/v1/users/{user_id1}"
              }
          },
          {
              "type": "request",
              "attributes": {
                  "method": "GET",
                  "url": "/api/v1/users/{user_id1}"
              }
          },
  ```

```
    ]
}
```

Response:

- Status Code: 201 Created
- Body

  ```
  {
      "data": {
          "attributes": {
              "url": "/api/v1/batchasks/{ask_id}"
          }
      }
  }
  ```

Step 2

Request:

- URI

  ```
  GET http://example.com/api/batchtask/{ask_id}
  ```

- Body

  EMPTY

Response:

- Status Code: 200 OK
- Body

  A **zip file** that contains the resources asked by user.

3. Delete a picture

Description:

Request:

- URI

  ```
  DELETE /api/v1/pictures/{picture_id}
  ```

- Body

  EMPTY

Response:

- Status Code: 204 DELETED

- Body

  EMPTY

Errors:

- 404 ERROR

Tasks

1. Create a Task

Description:

Request:

- URI

```
POST /api/v1/tasks
```

- Body

```
{
    "data": {
        "id": "uuid",
        "type": "task",
        "attributes": {
            "xml": "xxxx",
            "title": "xxx",
            "start": "xxxx",
            "end": "xxx",
            "descriptions": "xxx"
        }
    }
}
```

Response:

- Status Code: 201 Created
- Body

  EMPTY

Errors:

2. Get Pictures Related to a Tasks

The links of the pictures should be contained in the xml file of the task.

Commits

1. Upload a commit

Description:

Request:

- URI

```
POST /api/commits
```

- Body

```
{
    "data": {
        "id": "uuid",
        "type": "task",
        "attributes": {
            "task_id": "uuid",
            "author_id": "uuid",
            "picture_id": "uuid",
            "xml": "xxx"
        }
    }
}
```

Response:

- Status Code: 201 CREATED
- Body

  EMPTY

Errors:

2. Get User Commits

Description:

Request:

- URI

```
GET /api/commits?user={user_id}&task={task_id}
```

- Body

  EMPTY

Response:

- Status Code: 200 OK
- Body

```
{
    "data": [{
        "id": "uuid",
        "type": "task",
        "attributes": {
            "task_id": "uuid",
            "author_id": "uuid",
            "picture_id": "uuid",
            "xml": "xxx"
        }
    }]
}
```

Errors:

- Commits not found. 404

3. Update a commit

Description:

Request:

- URI

  ```
  PUT /api/v1/commits/{commit_id}
  ```

- Body

  ```
  {
      {
      "data": {
          "id": "uuid",
          "type": "task",
          "attributes": {
              "task_id": "uuid",
              "author_id": "uuid",
              "picture_id": "uuid",
              "xml": "xxx"
          }
      }
  }
  ```

Response:

- Status Code: 200 OK
- Body

  EMPTY

Errors:

- 404 NOT FOUND