1 Structures

There are several standardized structures used by the API.

1.1 `book`

Books have the following fields:

- `name`: the name of the book
- `author`: the author of the book
- `uuid`: the id of the book (immutable)
- `chapters`: a list `chapter_instance`s.

1.1.1 *`chapter_instance`*

A chapter instance is a specific chapter at a specific commit. This is used by the `Book` object to reference the exact version. It is a list containing:

[`chapter_uuid`, `author`, `commit sha`, `chapter_name`]

The first two fields and the last one are required. The commit sha can be `null` if the chapter is not pinned to a specific version.

1.2 `chapter`

Chapters have the following fields:

- `name`: the name of the chapter
- `author`: the author of the chapter
- `uuid`: the id of the chapter (immutable)
- `scrap`: a list of `scrap_instance` structures.

1.2.1 *`scrap_instance`*

A scrap instance is a specific scrap at a specific commit. This is used by the `Chapter` object to reference the exact version. It is a list containing:

[`scrap_uuid`, `author`, `commit sha`]

The first two fields are required. The commit sha can be `null` if the scrap is not pinned to a specific version.

1.3 `scrap`

Scraps have the following fields:

- `author`: the author of the book
- `uuid`: the id of the book (immutable)
- `text`: the text of the scrap

2 Making POST requests

Any request that writes data to the server must be authenticated. Authentication is handled with an `Authorization: Token {token}` header.

After login, the `onSignIn` function will be given a Google oAuth ID token. `POST` this ID token to the server to receive a token for this service. If in doubt, see the example page code for details.

3 Backend API calls

3.1 `/books`

3.1.1 *`GET /books/{author}`*

Returns all books for a user.

```
[
  {
    "name": "Plumbus",
    "author": "hagrid",
    "uuid": "19b66178-856d-4dad-bbc2-a9575ecfd36b",
    "chapters": [
      [
        "hagrid",
        "aec55377-716d-4274-b006-44913f73ca7f",
        null,
        "Chapter Name"
      ],
      ...
    ]
  },
  ...
]
```

3.1.2 *`GET /books/{author}/{uuid}`*

Returns the information about a single book, given by author and uuid.

```
{
  "name": "Plumbus",
  "author": "hagrid",
  "uuid": "19b66178-856d-4dad-bbc2-a9575ecfd36b",
  "chapters": [
    [
      "hagrid",
      "aec55377-716d-4274-b006-44913f73ca7f",
      null,
      "Chapter Name"
    ],
    ...
  ]
}
```

3.1.3 *POST /books/{author}/{uuid}*

Updates a book with information from the request body. Fields that can be updated are:

- `chapters`
- `name`

Only fields being updated need to be included. If updating chapters, existing chapters WILL be overwritten; include existing chapters that you do not wish to remove.
The chapters must be a list of lists, with the inner lists containing:

1) author
2) chapter uuid
3) commit sha

3.1.3.1 Example: change name and change chapter uuid / pin chapter:

```
POST /books/hagrid/19b66178-856d-4dad-bbc2-a9575ecfd36b
{
  "name": "New Book Name",
  "chapters": [
    [
      "hagrid",
      "68c47c74-f6fb-4e5b-a68c-f2c6b4265bd1",
      "ad3df920f3ce04687732c5572a76e541e6a1b799"
    ]
  ]
}
```

`POST` update requests return the message body of the new commit:

```
{
  "message": "update:..."
}
```

3.1.4 *POST /books/{author}/{uuid}/fork*

TODO

3.1.5 *GET /books/{author}/{uuid}/pdf*

Returns a PDF file containing the book.

3.1.6 *GET /books/{author}/{uuid}/history*

Returns the history of the book.

```
[
  [
    "a632575d435f86e19434092fe3e2fca98a3bb7e5",
    "update: changed name from Hagrid’s Big Book of BBQ to New Book Name. "
  ],
  [
    "b76c2469b00ee63c0e462faa32a5875abbbd51c7",
    "update: updated chapters (TODO diff). "
  ],
  [
    "15ccb3513d209ca9770cb188ace7fe9bcd97b433",
    "Created book named Hagrid’s Big Book of BBQ"
  ]
]
```

3.1.7 *POST /books/new*

Create a new book object. The name and author are both required; chapters cannot be added. The author must be the same as the logged in user.

```
POST /books/new
{
  "name": "My Favorite Book",
  "author": "magellan"
}
```

`POST` creation requests return a book structure:

```
{
  "name": "My Favorite Book",
  "author": "magellan",
  "uuid": "802619b6-f102-4b63-9fbe-f578c7af671f",
  "chapters": []
}
```

3.2 `/chapters`

3.2.1 *`GET /chapters/{author}`*

Returns all chapters for a user.

```
[
  {
    "author": "hagrid",
    "chapter one",
    "uuid": "68c47c74-f6fb-4e5b-a68c-f2c6b4265bd1",
    "scraps": [
      [
        "hagrid",
        "f47c9536-7e72-4fd1-9532-282cedfddc72"
      ],
      ...
    ]
  },
  ...
]
```

3.2.2 *`GET /chapters/{author}/{uuid}`*

Returns information about a single chapter, given by author and uuid.

```
{
  "author": "hagrid",
  "name": "chapter one",
  "uuid": "68c47c74-f6fb-4e5b-a68c-f2c6b4265bd1",
  "scraps": [
    [
      "hagrid",
      "f47c9536-7e72-4fd1-9532-282cedfddc72"
    ],
    ...
  ]
}
```

3.2.3 *`POST /chapters/{author}/{uuid}`*

Updates a chapter with information from the request body. Fields that can be updated are:

- `scraps` (a list of `scrap_instances`)

- `name`

Only fields being updated need to be included. If updating scraps, existing scraps WILL be overwritten; include existing scraps that you do not wish to remove.

3.2.3.1 Example: change name and replace scrap / pin scrap:

```
POST /scraps/hagrid/19b66178-856d-4dad-bbc2-a9575ecfd36b
{
  "name": "New Chapter Name",
  "scraps": [
    [
      "hagrid",
      "68c47c74-f6fb-4e5b-a68c-f2c6b4265bd1",
      "ad3df920f3ce04687732c5572a76e541e6a1b799"
    ]
  ]
}
```

`POST` update requests return the message body of the new commit:

```
{
  "message": "update: ..."
}
```

3.2.4 `POST /chapters/{author}/{uuid}/fork`

TODO

3.2.5 `GET /chapters/{author}/{uuid}/pdf`

Returns a PDF file containing the chapter

3.2.6 `GET /chapters/{author}/{uuid}/history`

Returns the history of the chapter.

```
[
  [
    "3c27eaa113f876ba389f27835478f0b09c60aa39",
    "update: changed name from Second Part to Second Stanza. "
  ],
  [
    "018a57d8b72a131873ee0534c42c47bf8abc5c25",
    "update: updated scraps (TODO diff). "
  ],
  [
    "61de182a1b638611633f829e98f35e6423038be5",
    "Created chapter named Second Part"
  ]
]
```

3.2.7 `POST /chapters/new`

Create a new chapter object. The name and author are both required; scraps cannot be added. The author must be the same as the logged in user.

```
POST /chapters/new
{
  "name": "My Favorite Chapter",
  "author": "magellan"
}
```

`POST` creation requests return a chapter structure:

```
{
  "name": "My Favorite Chapter",
  "author": "magellan",
  "uuid": "802619b6-f102-4b63-9fbe-f578c7af671f",
  "scraps": []
}
```

3.3 `/scraps`

3.3.1 `GET /scraps/{author}`

Returns all scraps for a user.

```
[
  {
    "author": "hagrid",
    "text": "But then someone shouted...\"Hey, that's Argentina!\"\nMagellan...",
    "uuid": "1a530c40-d8ed-4362-b402-4481c50e50a2"
  },
  ...
]
```

3.3.2 `POST /scraps/{author}/{uuid}`

Updates a scrap with information from the request body. Fields that can be updated are:

- `text`
- `latex`
- `tags`

3.3.2.1 Example: change text, latex, tags:

```
POST /scraps/hagrid/1a530c40-d8ed-4362-b402-4481c50e50a2
{
  "text": "hello world!",
  "latex": false,
  "tags": [
    "tag1",
    "tag2",
    "tag3"
  ],
}
```

`POST` update requests return the message body of the new commit:

```
{
  "message": "update: changed text from This is a text to hello world!."
}
```

3.3.3 POST /scraps/{author}/{uuid}/fork

TODO

3.3.4 GET /scraps/{author}/{uuid}/pdf

Returns a PDF file containing the scrap

3.3.5 GET /scraps/{author}/{uuid}/history

Returns the history of the scrap.

```
[
  [
    "aa62f9f63c013795ee69c2dec3321593b8fbbecd",
    "update: changed text from Lovely sentence about flowers to Hello World."
  ],
  [
    "0e0b4ef415bc3cbd6e4ed004d719b611c8fc657c",
    "update: changed text from to Lovely sentence about flowers."
  ],
  [
    "2fc3a0061705f34c6d559b75085c895751b815a3",
    "Created new scrap"
  ]
]
```

3.3.6 `POST /scraps/new`

Create a new scrap object. The author is required; text can be added. The author must be the same as the logged in user.

```
POST /scraps/new
{
  "author": "magellan"
}
```

`POST` creation requests return a scrap structure:

```
{
  "author": "magellan",
  "uuid": "802619b6-f102-4b63-9fbe-f578c7af671f",
  "text": ""
}
```

3.3.7 `POST /images/new`

Create a new image. This is a `form/multipart` file upload; the image must be in the field "image". When successful, it will return a scrap object that can be included in a chapter to include the picture:

```
{
author: "hagrid",
text: "\includegraphics[width=\textwidth]{/images/hagrid/mhcf7dezlahdxu5jt6gvi}",
latex: true,
image: true,
isNew: true,
uuid: "266bfe4c-2338-4a03-afdd-a9765fdbe629"
}
```

3.4 `/search`

Search:

```
/search?q=url%20encoded%20query
```

Search for specific types:

```
/search?q=abc123&type=books
/search?q=abc123&type=chapters
/search?q=abc123&type=books&type=chapters
```

Search only hagrid's items (can specify any user):

```
/search?q=abc&user=hagrid
```

3.5 `Favorites`

There are three API calls in the favoriting family.

3.5.1 *POST /{type}/{author}/{id}/favorite*

Adds a favorite. If the object is already favorited, it stays in the favorites. The return body is empty; the return status code is 204.

Ex: `POST /books/hagrid/19b66178-856d-4dad-bbc2-a9575ecfd36b/favorite`

3.5.2 *DELETE /{type}/{author}/{id}/favorite*

Removes a favorite. If the object isn't favorited, it stays not favorited without an error. The return body is empty; the return status code is 204.

Ex: `DELETE /books/hagrid/19b66178-856d-4dad-bbc2-a9575ecfd36b/favorite`

3.5.3 *GET /favorites*

Gets all of the user's favorites. The token is used to determine whose favorites to return.

```
[
  {
    "type": "book",
    "author": "evantschuy",
    "uuid": "19b66178-856d-4dad-bbc2-a9575ecfd36b"
  },
  {
    "type": "book",
    "author": "hagrid",
    "uuid": "19b66178-856d-4dad-bbc2-a9575ecfd36b",
    "name": "Plumbus: How They're Made"
  },
  {
    "type": "scrap",
    "author": "hagrid",
    "uuid": "31f54157-1ab4-48fc-a606-849e28270faf",
    "text": "this is a new scrap created 14:03 13 April 2017"
  }
]
```

To filter to specific types, use the same filter as for `search`: `?type=...`:

`GET /favorites?type=book`

```
[
  {
    "type": "book",
    "author": "evantschuy",
    "uuid": "19b66178-856d-4dad-bbc2-a9575ecfd36b"
  },
  {
    "type": "book",
    "author": "hagrid",
    "uuid": "19b66178-856d-4dad-bbc2-a9575ecfd36b",
    "name": "Plumbus: How They’re Made"
  }
]
```

Multiple types can be specified:

```
GET /favorites?type=book&type=chapter
```

3.6 `/accounts/myaccount`

3.6.1 *`POST /accounts/myaccount`*

Provides an endpoint to update the name to be displayed on book renders:

```
{"name": "D. Campbell"}
```

3.6.2 *`GET /accounts/myaccount`*

Provides an endpoint to view the name to be displayed on book renders:

```
{
  "userid": "dcampbell",
  "name": "D. Campbell"
}
```