

“How to Run” - Team 2

Garrett Davidson, Noah Maxey, Nate Ohlson, Riley Shaw

Requirements:

- Computer running at least macOS 10.12
- MySQL
 - MySQL server should be running on localhost
 - Must have a user named “test” with password “password”
 - User must have permissions to create and edit a database

Running the server:

Execute `./HiLingualServer`` from the terminal. The server will launch on localhost and begin listening on port 8180.

Interacting with the server:

1. We recommend using Postman to send requests to the server. All documentation is shown under the assumption you will be using Postman. <https://www.getpostman.com/>
2. Create a HiLingual account.
 - Most requests require authorization, so you will need an account. You must use either Facebook or Google to create an account.
 - For Facebook, we have created a list of test users with their id’s and session tokens:
https://docs.google.com/document/d/1rWMalipAXE4YDt57I5Bi_857Yie0RYLUP6HZBRv_TgM/edit?usp=sharing
 - For Google there is no way to create test accounts, so to test it you will need to use your personal Google account, or create one to test with. (You can revoke these permissions when you are done testing.)
 - i. Go here: <https://developers.google.com/oauthplayground/>
 - ii. On the left, under Step 1, select “Google OAuth2 API v2”
 - iii. Check “<https://www.googleapis.com/auth/plus.login>” then hit “Authorize APIs”
 - iv. Under Step 2, hit “Exchange authorization code for tokens”
 - v. In the response section, the “id_token” is your authorityToken for use later. You can choose any unique arbitrary integer for you GoogleID.

Note: Throughout this document, “authorityToken” refers to access tokens from Google or Facebook while “Auth-Token” refers to HiLingual tokens. Your “authorityToken” is only required for registering and logging in/out. Everywhere else you should be using “Auth-Token.”

Register:

http://localhost:8180/auth/register

Request type	authority	authorityAccountId	authorityToken	Body
POST	Either "FACEBOOK" or "GOOGLE"	The "User ID" from the Facebook test user table or "GoogleID" chosen earlier.	Token given by Facebook or Google.	JSON containing authority, authorityAccountId and authorityToken.

Example:

localhost:8180/auth/register

POST

localhost:8180/auth/register

Params

Send

Authorization

Headers (1)

Body

Pre-request script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

Text

```
1 {
2   "authority": "FACEBOOK",
3   "authorityAccountId": "140435239686268",
4   "authorityToken": "EAA0cGLfuCWEBAFjx42FyaUtT6kaXEdpcpxQa75cccbyi0gPZAQMqvNZAxcwDik0G0QEEYFTystZCoZAGhau0az
5 }
```

Login:

http://localhost:8180/auth/login

Request type	authority	authorityAccountId	authorityToken	Body
POST	Either "FACEBOOK" or "GOOGLE"	The "User ID" from the Facebook test user table or "GoogleID" chosen earlier.	Token given by Facebook or Google.	JSON containing authority, authorityAccountId and authorityToken.

Example:


localhost:8180/auth/login

POST ▾

localhost:8180/auth/login

Params

Send ▾

 ▾

Authorization

Headers (1)

Body

Pre-request script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

Text ▾

```
1 {
2   "authorityAccountId": "113901922344055",
3   "authorityToken": "EAA0cGLfuCWEBAPxsLTvNhhX8ZC6qu84pyo779518ECQkFfqpQpToonZAKBwSRudU0cEj0K7r8nt0iiIs1DU9B5",
4   "authority": "FACEBOOK"
5 }
```

Logout:

http://localhost:8180/auth/logout

Request type	authority	authorityAccountId	authorityToken	Body	UserId
POST	Either "FACEBOOK" or "GOOGLE"	The "User ID" from the Facebook test user table or "GoogleID" chosen earlier.	Token given by Facebook or Google.	JSON containing authority, authorityAccountId, user_id and authorityToken.	HiLingual User id

Example:


localhost:8180/auth/logout

POST ▾

localhost:8180/auth/logout

Params

Send ▾

 ▾

Authorization

Headers (1)

Body

Pre-request script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

Text ▾

```
1 {
2   "user_id": "1",
3   "authority": "FACEBOOK",
4   "authorityAccountId": "113901922344055",
5   "authorityToken": "EAA0cGlfuCWEBAPxslTvNhhX8ZC6qu84pyo77951BECQkFfqpQpToonZakBwSRudU0cEjOK7r8nt0iiIs1DU9B8
6 }
```

Messages:

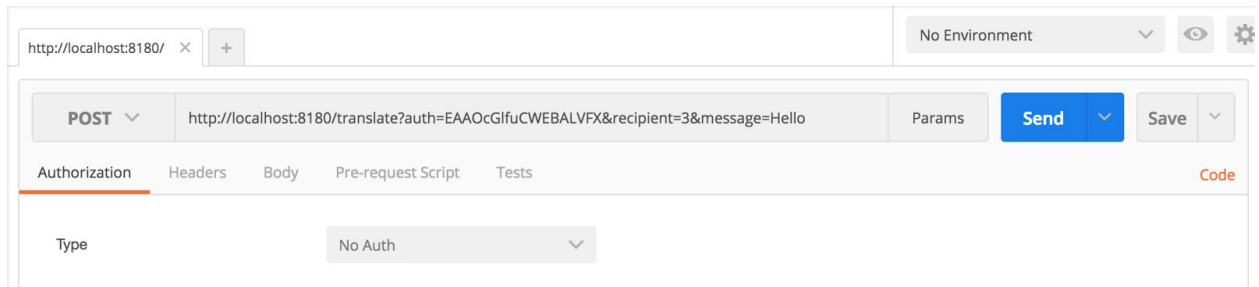
Response: Should not say invalid message.

Sending a Message:

`http://localhost:8180/chat?auth={Auth-Token}&recipient={User-id}&message={Message}`

Request type	Auth-Token	User-id	Message
POST	An already logged in user's auth token.	The user you want to send to. User ids are autoincremented when registering, starting with 1. Is a number.	The message that you want to send.

Example:

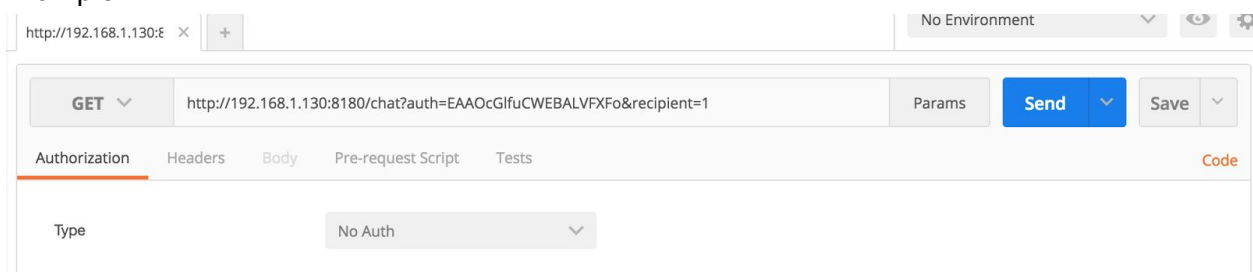


Receiving Messages:

`http://localhost:8180/chat?auth={Auth-Token}&recipient={User-id}`

Request type	Auth-Token	User-id
GET	An already logged in user's auth token.	The user you want to receive from. (they are incrementally increased based on when you added them so 1,2,3....) Is a number.

Example:



Sending a Picture Message:

http://localhost:8180/picture?auth={Auth-Token}&recipient={User-id}

Request type	Auth-Token	User-id	Picture
POST	An already logged in user's auth token.	The user you want to send to. User ids are autoincremented when registering, starting with 1. Is a number.	Should be a valid Image file.

Example:

The screenshot shows a REST client interface with a POST request to `http://localhost:8180/picture?auth=SADefdaseE&recipient=1`. The request body is configured as `form-data` with a key `key` and a value `test.jpg`. The interface includes tabs for Authorization, Headers, Body, Pre-request Script, and Tests. The Body tab is active, and the form-data type is selected. The key-value pair is displayed in a table with a 'Choose Files' button next to the value field.

Sending an audio message:

http://localhost:8180/audio?auth={Auth-Token}&recipient={User-id}

Request type	Auth-Token	User-id	Audio
POST	An already logged in user's auth token.	The user you want to send to. User ids are autoincremented when registering, starting with 1. Is a number.	Should be a valid m4a audio data.

Example:

The screenshot shows a REST client interface with a POST request to `http://localhost:8180/audio?auth=adsf&recipient=1`. The request body is configured as `form-data` with a key `audio` and a value `under10.mp3`. The interface includes tabs for Authorization, Headers, Body, Pre-request Script, and Tests. The Body tab is active, and the form-data type is selected. The key-value pair is displayed in a table with a 'Choose Files' button next to the value field.

Flashcards:

Sending flashcards:

<http://localhost:8180/flashcard?auth={Auth-Token}&setid{Set-id}>

Request type	Auth-Token	Set-id	Flashcards
POST	An already logged in user's auth token.	What you want the set of flashcards to be called. Has to be Unique.	A valid JSon array with front and back fields.

Example:

The screenshot shows a REST client interface with a POST request to `http://localhost:8180/flashcard?auth=2&setid=set1`. The request body is a JSON array of flashcards:

```
1 {
2   "FlashcardRingName": [
3     {
4       "front": "Hello",
5       "back": "Hallo"
6     },
7     {
8       "front": "I",
9       "back": "Ich"
10    }
11  ]
12 }
```

Get flashcards:

<http://localhost:8180/flashcard?auth={Auth-Token}&setid{Set-id}>

Request type	Auth-Token	Set-id
Get	An already logged in user's auth token.	What you want the set of flashcards to be called.

Example:

The screenshot shows a REST client interface with a GET request to `http://localhost:8180/flashcard?auth=asdfASDWewr&setid=set1`. The Authorization tab is selected, showing "Type" as "No Auth".

Editing flashcards:

<http://localhost:8180/flashcard?auth={Auth-Token}&setid{Set-id}>

Request type	Auth-Token	Set-id	Flashcards
POST	An already logged in user's auth token.	The set-id should match a previously submit set.	A valid JSon array with front and back fields.

Example:

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8180/flashcard?auth=adsfAFIiWJD&setid=set1`
- Method: `POST`
- Body Type: `JSON (application/json)`
- Body Content (JSON):

```
1 {
2   "NewFlashcardName": [
3     {
4       "front": "Hello",
5       "back": "Hallo"
6     },
7     {
8       "front": "I",
9       "back": "Ich"
10    },
11    {
12      "front": "NO",
13      "back": "NEIN"
14    }
15  ]
16 }
```

Translations:

Send a translation:

<http://localhost:8180/translate?auth={Auth-Token}&language={Language}&message={message}>

Request type	Auth-Token	Language	message
POST	An already logged in user's auth token.	A language code for a language. http://www.science.co.il/Language/Locale-codes.asp	The message you want to be translated

Example:

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8180/translate?auth=EAAOcGlfuCW&language=de&message=Hello`
- Method: `POST`

User:

Update:

<http://localhost:8180/user/update>

Request type	Auth-Token	User Id	name	displayName
POST	An already logged in user's auth token.	User's user id	Proper name of the user	The name that other users see

bio	gender	birthdate	Native languages	Learning languages
The bio of the person	Either all caps "MALE" or "FEMALE"	An integer of unix timestamp	Comma separated list of languages	Comma separated list of languages

Example:


localhost:8180/user/update

POST ▾

localhost:8180/user/update

Params

Send ▾

 ▾

Authorization

Headers (1)

Body

Pre-request script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

Text ▾

```
1 {
2   "user_id": "1",
3   "authorityToken": "EAAOcGl fuCWEBAPxsLTvNhhX8ZC6qu84pyo77951BECQkFfqPqToonZakBwSRudU0cEj0K7r8nt0iiIs1DU9f",
4   "name": "Bobby Smith",
5   "displayName": "bbob",
6   "bio": "I'm a cool cat",
7   "gender": "MALE",
8   "birthdate": "1477595593",
9   "nativeLanguages": "English",
10  "learningLanguages": "Spanish"
11 }
```

Matching

<http://localhost:8180/user/match>

Request type	authorityToken	Body
POST	Token given by Facebook or Google.	JSON containing authorityToken.

Example:

POST ▾

192.168.1.122:8180/user/match/

Params

Send ▾

Save ▾

AuthorizationHeadersBody ●Pre-request ScriptTestsManage CookiesGenerate Code

form-data

x-www-form-urlencoded

●raw

binary

Text ▾

1{

2 "authorityToken": "EAA0cGlFuCWEBAEao5uIsn8qvShrEGYDiv8ZAE8gm3gym5iyQJBFS1CZCndJSP1zmUBPBb8U5rLxkkZAZ

3 AjZABV0sLorAAZBvuyudv33uNgekspgHggEt9yu8DQzIuzDaPv5ZCZCQLC7wVZC3uRLESoaNYyUR1Js3qVDhKa6QhxdzH8If

4 Gcj00SAIO"

5 }