

## LAB 2

npm install in each folder.

install and run redis server

use 'node index.js' command line in folder 'recipe-worker'

use 'npm start' command line in folder 'recipe-server'

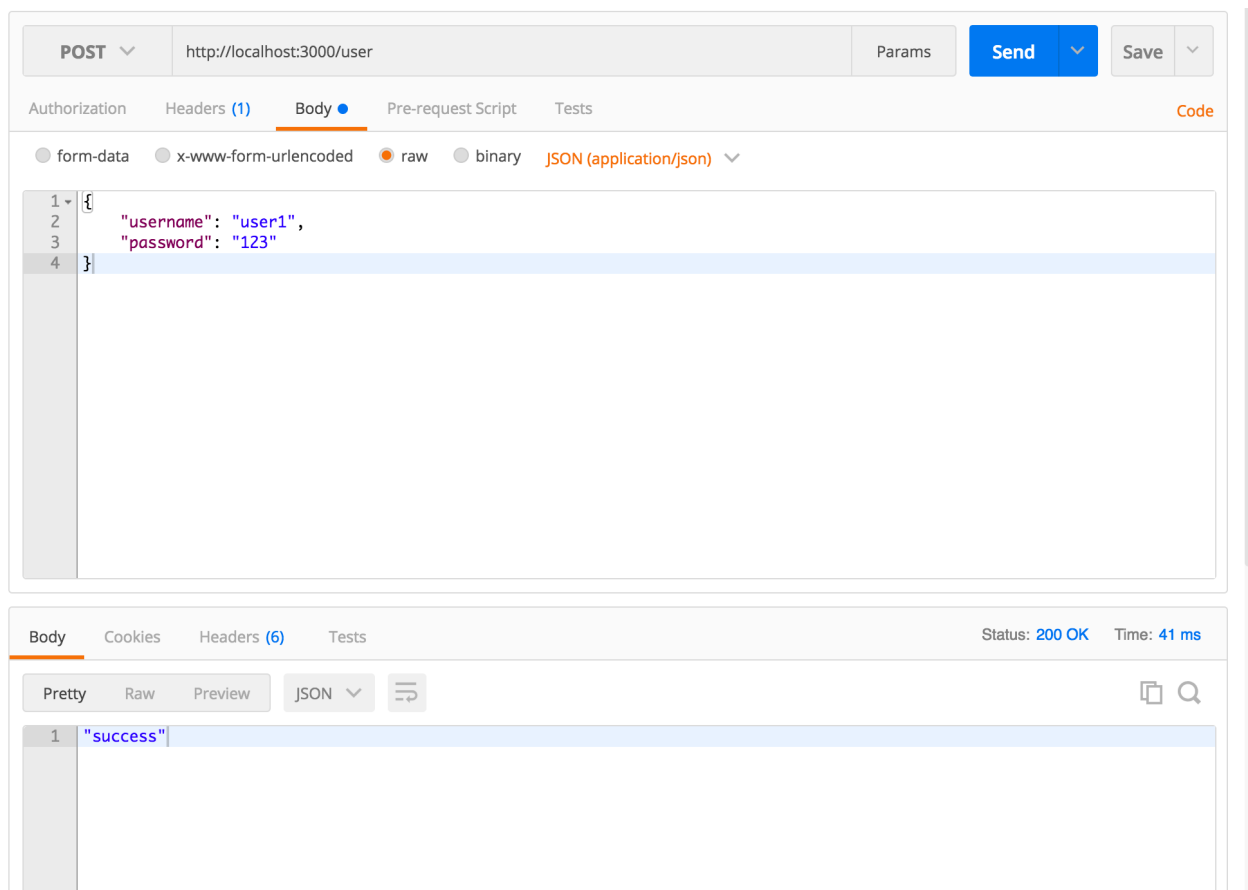
### Use Postman to test the api

#### Screenshot in each route:

#### USER

1.

POST	/users	Posting to this route must allow for the creation of a user. Making a new user updates the location in redis that you are using to cache your list of users; whether this is a set, a hash, or a JSON string is up to you. You should also cache the entire user in redis so that they are cached by their ID.
------	--------	--



Result:

```
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> keys *
1) "12e8972a-569e-4ebe-87dd-a46acccec7bb"
127.0.0.1:6379> |
```

Body format:

```
{
  "username": "user1",
  "password": "123"
}
```

2.

POST	/users/session	Posting your username and password to this route must create a new session for the user with the credentials provided and respond with the authentication token that will associate the user to the token. The token returned will be the value provided to the server in the Auth-Token header
------	----------------	---

POST

http://localhost:3000/user/session

Params

Send

Save

AuthorizationHeaders (2)BodyPre-request ScriptTestsCode

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

1 {

2 "userId": "12e8972a-569e-4ebe-87dd-a46acccec7bb",

3 "username": "user1",

4 "password": "123"

5 }

BodyCookiesHeaders (6)TestsStatus: 200 OKTime: 52 ms

PrettyRawPreviewJSON

1 {

2 "id": "12e8972a-569e-4ebe-87dd-a46acccec7bb",

3 "username": "user1",

4 "token": "123",

5 "recipes": []

6 }

Result:

```
[127.0.0.1:6379> keys *
1) "sess:nr08eIIK7evXNw81IN8M0jA12mNHZUqf"
```

Headers:

Authorization	Headers (2)	Body ●	Pre-request Script	Tests
✓	Content-Type	application/json		
✓	Auth-Token	123		
	key	value		

Body format:

```
{
  "userId": "12e8972a-569e-4ebe-87dd-a46acccec7bb",
  "username": "user1",
  "password": "123"
}
```

3.

GET	/users/:id	Must provide public facing info about the user; no sensitive data must be displayed; results will be cached for 5 minutes. You can cache using the express-redis-cache package or you may cache the data yourself.
-----	------------	--

http://localhost:3000/user/12e8972a-569e-4ebe-87dd-a46acccec7bb

Params

Send

Save

Authorization

Headers (1)

Body

Pre-request Script

Tests

Code

Type

No Auth

Body

Cookies

Headers (6)

Tests

Status: 200 OK

Time: 62 ms

Pretty

Raw

Preview

JSON

⌵

⌵

1 {

2 "username": "user1",

3 "userId": "12e8972a-569e-4ebe-87dd-a46acccec7bb"

4 }

Result: (cache for 5 mins)

```
127.0.0.1:6379> keys *
1) "sess:nr08eIIK7evXNw81IN8M0jAl2mNHZUqf"
2) "12e8972a-569e-4ebe-87dd-a46acccec7bb"
3) "erc:user1"
```

'erc: user1' is the cache

4.

GET	/users	Provides public facing info about all users; results will be cached for 10 minutes. You can cache using the express-redis-cache package or you may cache the data yourself.
-----	--------	---

The screenshot shows a REST client interface. The top bar indicates a GET request to `http://localhost:3000/user/`. Below this, the 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response is a JSON object with 'username' and 'userId' fields. The status bar at the top right shows 'Status: 200 OK' and 'Time: 66 ms'.

```
{
  "username": "user1",
  "userId": "12e8972a-569e-4ebe-87dd-a46acccec7bb"
}
```

Result: (cache all user for 10 mins)

**THE “erc:user” is the whole user cache**

```
127.0.0.1:6379> keys *
1) "sess:nr08eIIK7evXNw81IN8M0jAl2mNHZUqf"
2) "erc:user"
3) "12e8972a-569e-4ebe-87dd-a46acccec7bb"
4) "erc:user1"
127.0.0.1:6379> |
```

5.

PUT	/users	PUTting to this route must update the current user; note, you'll have to be authenticated to do this! If a cache entry exists for this user, it must be updated and exist for 5 minutes before expiring (basically, reset the cache time)
-----	--------	---

http://localhost

http://localhost

http://localhost

http://localhost

http://localhost

http://localhost

+

No Environment

PUT

http://localhost:3000/user/

Params

Send

Save

Authorization

Headers (2)

Body

Pre-request Script

Tests

Code

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```
1 {  
2   "userId" : "6625dcff-373d-4e92-b7b1-d8872c17ef8d",  
3   "username": "user1",  
4   "password": "123",  
5   "newPassword": "123321"  
6 }
```

Body

Cookies

Headers (6)

Tests

Status: 200 OK

Time: 31 ms

Pretty

Raw

Preview

JSON

1 "success"

Result:

```
127.0.0.1:6379> keys *
1) "6625dcff-373d-4e92-b7b1-d8872c17ef8d"
127.0.0.1:6379> SMEMBERS 6625dcff-373d-4e92-b7b1-d8872c17ef8d
1) "{\"userId\":\"6625dcff-373d-4e92-b7b1-d8872c17ef8d\", \"username\":\"user1\", \"password\":\"123321\"}"
```

headers:

Authorization

Headers (2)

Body

Pre-request Script

Tests

✓	Content-Type	application/json	<div><div></div><div></div></div>
✓	Auth-Token	123	<div><div></div><div></div></div>
	key	value	

body format:

PUT http://localhost:3000/user/ Params Send Save

Authorization Headers (2) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {  
2   "userId": "6625dcff-373d-4e92-b7b1-d8872c17ef8d",  
3   "username": "user1",  
4   "password": "123",  
5   "newPassword": "123321"  
6 }
```

6.

DELETE	/users	Will delete your user based on the Auth-token provided and invalidate session; If a cache entry exists for this user, it will be removed.
--------	--------	---

Before:

```
127.0.0.1:6379> keys  
1) "6625dcff-373d-4e92-b7b1-d8872c17ef8d"  
2) "sess:AhiLXEVOYNyUD0TvGbNAXSslcub4A0mQ"  
127.0.0.1:6379> |
```

session exist

DELETE http://localhost:3000/user/ Params Send Save

Authorization Headers (2) Body Pre-request Script Tests Code

Content-Type	application/json	⋮ ×
Auth-Token	123	⋮ ×
key	value	

Bulk Edit Presets

Body Cookies Headers (6) Tests Status: 200 OK Time: 31 ms

Pretty Raw Preview JSON

```
1 "success"
```

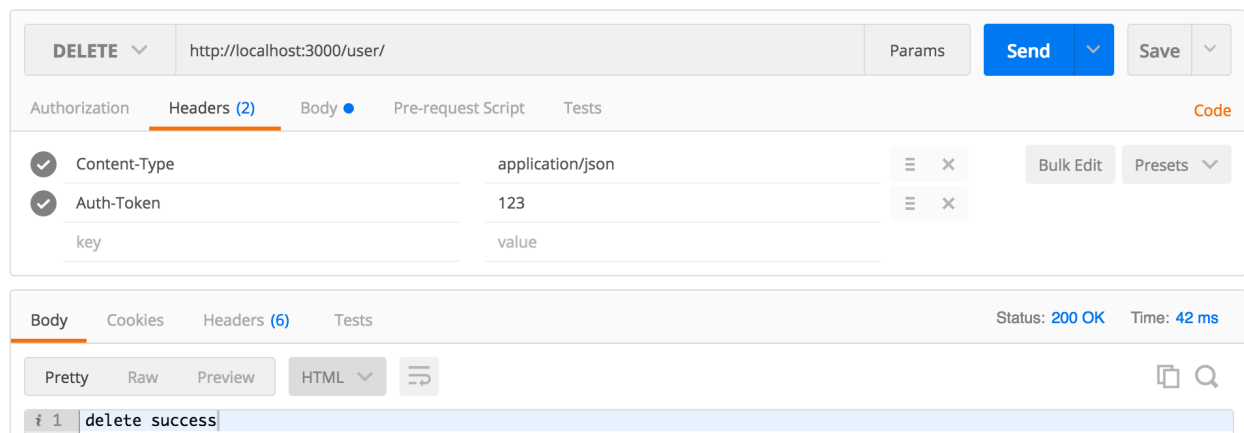
Headers:

Auth-Token match with session

Body: (user name should be provided)

```
1 {  
2   "username" : "user1"  
3 }
```

After:



```
127.0.0.1:6379> keys *  
1) "6625dcff-373d-4e92-b7b1-d8872c17ef8d"  
2) "sess:AhiLXEVOYNYUD0TvGbNAXSslcub4A0mQ"  
127.0.0.1:6379> keys *  
(empty list or set)  
127.0.0.1:6379> |
```

## Recipes

1.

POST	/recipes	<b>Modify this route</b> to add some association between the user and the recipe; you can only make recipes while logged in; the new recipe will be cached for 1 hour; the recipe list cache entry will be updated if it exists
------	----------	---

```
127.0.0.1:6379> keys *
1) "67d5bf27-5458-4811-ae8-52b846e39606"
2) "sess:PHGUIuI5jhN-ZNUPp0ZFZnrd0jj1NfVD"
127.0.0.1:6379> 
```

User session exist, user logged in.  
body:

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/recipes`. The request body is a JSON object representing a recipe. The response is also a JSON object, indicating a successful creation of the recipe with a `200 OK` status and a response time of `40 ms`.

**Request:**

```
{
  "title": "ONE Chemex coffee",
  "description": "make a cup of coffee",
  "ingredients": "coffee",
  "steps": {
    "first": "one",
    "second": "two"
  }
}
```

**Response:**

```
{
  "requestId": "e8408fcf-13dd-406d-8a17-37396855e5e8",
  "recipe": {
    "title": "ONE Chemex coffee",
    "description": "make a cup of coffee",
    "ingredients": "coffee",
    "steps": {
      "first": "one",
      "second": "two"
    }
  },
  "creator": "user1",
  "creator_id": "67d5bf27-5458-4811-ae8-52b846e39606"
}
```

```
{
  "title": "ONE Chemex coffee",
  "description": "make a cup of coffee",
  "ingredients": "coffee",
  "steps": {
    "first": "one",
    "second": "two"
  }
}
```



## Headers:

Authorization	Headers (2)	Body	Pre-request Script	Tests
✓	Content-Type	application/json		
✓	Auth-Token	123		
	key	value		

```
127.0.0.1:6379> keys *
1) "67d5bf27-5458-4811-ae8-52b846e39606"
2) "sess:PHGUiI5jhN-ZNUPp0ZFZnrd0jj1NfVD"
3) "erc:e8408fcf-13dd-406d-8a17-37396855e5e8"
4) "recipes"
127.0.0.1:6379> hmget recipes e8408fcf-13dd-406d-8a17-37396855e5e8
1) "{\"title\":\"ONE Chemex coffee\",\"description\":\"make a cup of coffee\",\"ingredients\":\"coffee\",\"steps\":{\"first\":\"one\",\"second\":\"two\"},\"creator\":\"user1\",\"creator_id\":\"67d5bf27-5458-4811-ae8-52b846e39606\"}"
127.0.0.1:6379>
```

All recipes store in “recipes” list.  
cache the new recipe when it be created.

```
127.0.0.1:6379> hvals erc:e8408fcf-13dd-406d-8a17-37396855e5e8
1) "1478474688935"
2) "json"
3) "3600"
4) "{\"requestId\":\"e8408fcf-13dd-406d-8a17-37396855e5e8\",\"recipe\":{\"title\":\"ONE Chemex coffee\",\"description\":\"make a cup of coffee\",\"ingredients\":\"coffee\",\"steps\":{\"first\":\"one\",\"second\":\"two\"},\"creator\":\"user1\",\"creator_id\":\"67d5bf27-5458-4811-ae8-52b846e39606\"}}"
```

2.

		Provide data about the recipe and the creator; if the user is logged in, a		
GET	http://localhost:3000/recipes/e8408fcf-13dd-406d-8a17-37396855e5e8	Params	Send	Save
Authorization	Headers (1)	Body	Pre-request Script	Tests
Type	No Auth			
Body	Cookies	Headers (6)	Tests	Status: 200 OK Time
Pretty	Raw	Preview	JSON	
1	{			
2	"title": "ONE Chemex coffee",			
3	"description": "make a cup of coffee",			
4	"ingredients": "coffee",			
5	"steps": {			
6	"first": "one",			
7	"second": "two"			
8	},			
9	"creator": "user1",			
10	"creator_id": "67d5bf27-5458-4811-ae8-52b846e39606"			
11	}			

use this part to test how many recipes

```
let temp = JSON.stringify(result);
console.log(req.session.cUser.recipes.length);
res.send(JSON.parse(temp));
```

```
Your routes will be running on http://localhost:3000
au success
2
au success
3
au success
4
au success
5
au success
6
au success
7
au success
8
au success
9
au success
10
au success
10
```

3.

GET	/recipes	Provide the id, title, creator, and creator ID of each recipe; the result should be cached for 1 hour. You can cache using the express-redis-cache package or you may cache the data yourself.
-----	----------	--

cache in "erc:all\_recipe" for 1 hours

```
127.0.0.1:6379> keys *
1) "67d5bf27-5458-4811-ae8-52b846e39606"
2) "erc:all_recipe"
3) "sess:PHGUIuI5jhN-ZNUPp0ZFZnrd0jj1NfVD"
4) "recipes"
5) "erc:e8408fcf-13dd-406d-8a17-37396855e5e8"
127.0.0.1:6379> |
```

Pretty Raw Preview JSON

```
1  [
2  {
3    "id": "e8408fcf-13dd-406d-8a17-37396855e5e8",
4    "title": "ONE Chemex coffee",
5    "creator": "user1",
6    "creator_id": "67d5bf27-5458-4811-ae8-52b846e39606"
7  }
8  ]
```

```
127.0.0.1:6379> hvals erc:all_recipe
1) "1478475476448"
2) "json"
3) "3600"
4) "[{"id":"e8408fcf-13dd-406d-8a17-37396855e5e8","title":"ONE Chemex coffee","creator":"user1","creator_
":"67d5bf27-5458-4811-ae8-52b846e39606"}]"
127.0.0.1:6379>
```

4.

PUT	/recipes/:id	Will allow you to update a recipe; only the creator of a recipe may update it; you must be authenticated to use this route; cache entries for that recipe must be updated
-----	--------------	---

The screenshot shows a REST client interface with a browser-like tab bar at the top. The active tab is titled 'http://localhost:3000/recipes/e8408fcf-13dd-406d-8a17-37396855e5e8'. Below the tab bar, the HTTP method is set to 'PUT'. The URL is 'http://localhost:3000/recipes/e8408fcf-13dd-406d-8a17-37396855e5e8'. There are buttons for 'Params', 'Send', and 'Save'. The request body is a JSON object:

```
{
  "title": "****UPDATE*** ONE chemex coffee",
  "description": "make a cup of coffee",
  "ingredients": "coffee",
  "steps": {
    "first": "one",
    "second": "two"
  },
  "creator": "user1",
  "creator_id": "67d5bf27-5458-4811-ae8-52b846e39606"
}
```

Below the request body, the response is shown. The status is '200 OK' and the time is '70 ms'. The response body is a JSON object:

```
{
  "requestId": "e8408fcf-13dd-406d-8a17-37396855e5e8",
  "recipe": {
    "title": "****UPDATE*** ONE chemex coffee",
    "description": "make a cup of coffee",
    "ingredients": "coffee",
    "steps": {
      "first": "one",
      "second": "two"
    },
    "creator": "user1",
    "creator_id": "67d5bf27-5458-4811-ae8-52b846e39606"
  }
}
```

“recipes” be updated

```
127.0.0.1:6379> hmget recipes e8408fcf-13dd-406d-8a17-37396855e5e8
1) "{\"title\":\"****UPDATE*** ONE chemex coffee\\\", \"description\\\": \"make a cup of coffee\\\", \"ingredients\\\": \"coffee\\\", \"steps\\\": {\\\"first\\\": \"one\\\", \"second\\\": \"two\\\"}, \"creator\\\": \"user1\\\", \"creator_id\\\": \"67d5bf27-5458-4811-ae8-52b846e39606\\\"}\""
```

cache for this recipe also be update.

```
127.0.0.1:6379> hvals erc:e8408fcf-13dd-406d-8a17-37396855e5e8
1) "{\"requestId\\\": \"e8408fcf-13dd-406d-8a17-37396855e5e8\\\", \"recipe\\\": {\\\"title\\\": \"****UPDATE*** ONE chemex coffee\\\", \"description\\\": \"make a cup of coffee\\\", \"ingredients\\\": \"coffee\\\", \"steps\\\": {\\\"first\\\": \"one\\\", \"second\\\": \"two\\\"}, \"creator\\\": \"user1\\\", \"creator_id\\\": \"67d5bf27-5458-4811-ae8-52b846e39606\\\"}}\""
```

headers: (Auth-Token must be provide for auth)

PUT ▼		http://localhost:3000/recipes/e8408fcf-13dd-406d-8a17-37396855e5e8	
Authorization		Headers (2)	Body ●
✓	Auth-Token	123	
✓	Content-Type	application/json	
	key	value	

body formate:

```
{
  "title": "***UPDATE*** ONE chemex coffee",
  "description": "make a cup of coffee",
  "ingredients": "coffee",
  "steps": {
    "first": "one",
    "second": "two"
  },
  "creator": "user1",
  "creator_id": "67d5bf27-5458-4811-aee8-52b846e39606"
}
```

5.

DELETE	/recipes/:id	Will allow you to delete a recipe; only the creator of a recipe may update it; you must be authenticated to use this route. If a cache entry exists for this recipe, it will be removed.
--------	--------------	--

```
127.0.0.1:6379> keys *
1) "67d5bf27-5458-4811-ae8-52b846e39606"
2) "erc:all_recipe"
3) "erc:d070db03-3d64-4ff2-82d5-25bacf3ea785"
4) "sess:PHGUIuI5jhN-ZNUPp0ZFZnrd0jj1NfVD"
5) "recipes"
127.0.0.1:6379> hkeys recipes
1) "d070db03-3d64-4ff2-82d5-25bacf3ea785"
127.0.0.1:6379>
```

cache and 'recipes' exist.

The screenshot shows a REST client interface. At the top, a DELETE request is configured for the URL `http://localhost:3000/recipes/d070db03-3d64-4ff2-82d5-25bacf3ea785`. The 'Headers' tab is active, showing an 'Auth-Token' header with a value of '123'. Below the headers, the 'Body' tab is active, displaying the response in JSON format: `"Recipe deleted"`. The status bar at the bottom indicates a successful response with a status of 200 OK.

Need provide 'Auth-Token' in headers.

```
127.0.0.1:6379> hkeys recipes  
(empty list or set)  
127.0.0.1:6379> keys *  
1) "67d5bf27-5458-4811-ae8-52b846e39606"  
2) "erc:all_recipe"  
3) "sess:PHGUIuI5jhN-ZNUPp0ZFZnrd0jj1NfVD"  
127.0.0.1:6379> |
```

cache and the recipe store in 'recipes' list be deleted.