Dashboard
[Web Programming II](#)
[Lab 2](#)

Immersive Reader

[Skip To Content](#)
[Dashboard](#)

- Account
- [Dashboard](#)
- Courses
- [Calendar](#)
- [Inbox](#)
- History
- [Search](#)
- [Help](#)

-

[Close](#)

- *[My Dashboard](#)*
- [Web Programming II](#)
- [Assignments](#)
- [Lab 2](#)

Immersive Reader

2022 Spring Semester

- [Home](#)
- [Announcements](#)
- [Assignments](#)
- [Grades](#)
- [Modules](#)
- [Zoom](#)
- [Quizzes](#)
- [Course Survey](#)
- [Panopto Video](#)
- [Course Survey](#)
- [Library Resources](#)
- [Record Your Name](#)

Lab 2

- Due Feb 7 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip

CS-554 Lab 2

Redis

For this assignment, you will practice your usage of redis and async / await together. You will promisify the redis-client for use, and you will use all of its methods

with the use of the *await* keyword.

Your Dummy Data

You will use the following JSON as [dummy data (Links to an external site.) (Links to an external site.)](#) for your app. **This data is to be accessed via your data module, which is not to be confused with your cache**.

You can populate your data module however you see fit (reading a file at runtime, flat file lookup, copy and pasting into the file, etc).

This data module will expose a single method, `getById(id)` that returns a promise to resolve to an entry with the requested id. It should return a failed promise if given an ID that does not exist in the data set, or resolve with the entry if it does.

Your `getById` promise should not resolve or reject for at 5 seconds to emulate a long running operation. You can achieve this with the following pseudocode (see lecture 6 server code for a concrete example!):

```
getById = ((id) => {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        // find project
        if (hasProject) {
          resolve(project);
        } else {
          reject(new Error("something went wrong"));
      }}, 5000);
    });
})
```

Your cache

You will cache your data in Redis for this assignment after it has been accessed; you should not cache all data by default.

Your Routes

You will only have 2 routes!

/api/people/history

This route will respond with an array of the last 20 users in the cache from the recently viewed list. **You can have duplicate users in your 20 user list**.

For each user being printed out, you will print out their entire user information.

For example, if user 19 visited the site, then user 190, then user 12, then user 190, then user 11, you would output:

```
[
{"id":11,"first_name":"Carolyn","last_name":"Austin","email":"caustina@cloudflare.com","gender":"Female","ip_address":"107.126.74.59"},
    {"id":190,"first_name":"Chris","last_name":"Fox","email":"cfox59@4shared.com","gender":"Male","ip_address":"23.186.9.97"},
    {"id":12,"first_name":"David","last_name":"Berry","email":"dberryb@apache.org","gender":"Male","ip_address":"169.242.213.17"},
    {"id":190,"first_name":"Chris","last_name":"Fox","email":"cfox59@4shared.com","gender":"Male","ip_address":"23.186.9.97"},
    {"id":19,"first_name":"Andrew","last_name":"Williams","email":"awilliamsi@mlb.com","gender":"Male","ip_address":"191.131.32.57"}
]
```

/api/people/:id

This route will do two things:

1) Check if the user has a cache entry in redis. If so, render the result from that cache entry

2) If not, query the data module for the person and fail the request if they are not found, or send JSON and cache the result if they are found.

If the person is found, then you will add their ID to a list of recently viewed people (this list allows duplicates!) ordered by most recent accesser first. This list can be infinitely large.

General Requirements

1. Remember to submit your `package.json` file but **not** your `node_modules` folder.

[Previous](#) [Next](#)

Submission

Submitted!
Feb 7 at 7:07pm
[Submission Details](#)
[Download Yufu_Liao_CS554_A_Lab2-1.zip](#)
Grade: 100 (100 pts possible)

Graded Anonymously: no

Comments:

Good job!

<div align="right">Simon Gao, Feb 22 at 1:26pm</div>