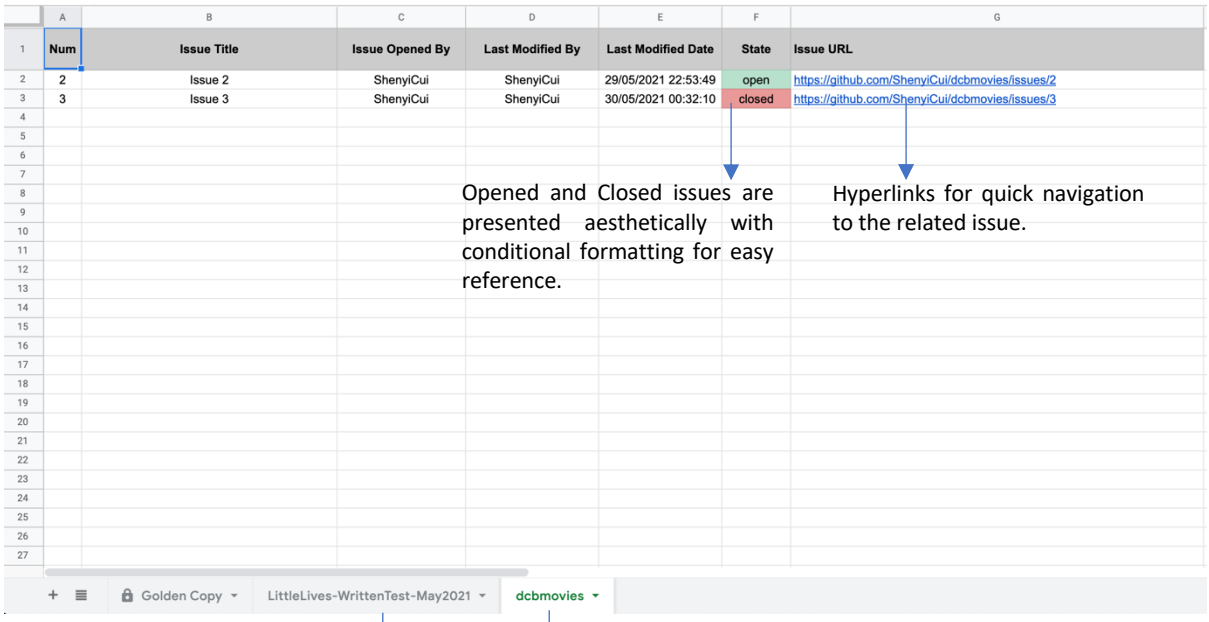
Execution of Design

Google Sheets



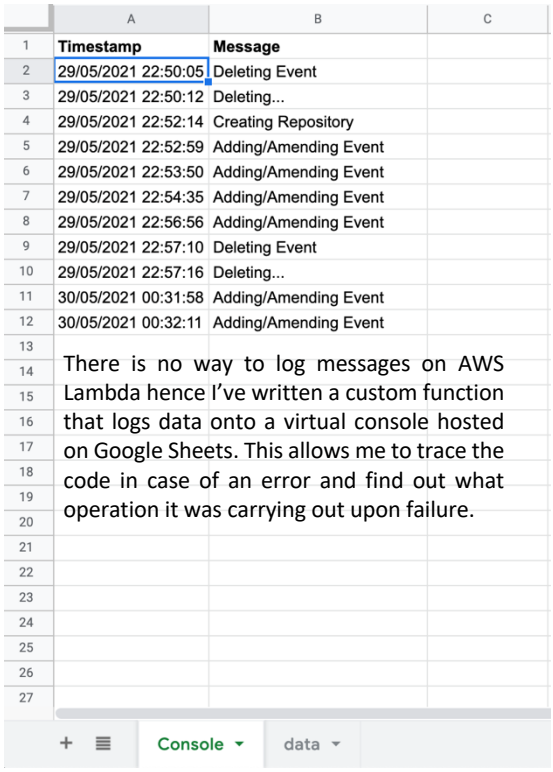
Num	Issue Title	Issue Opened By	Last Modified By	Last Modified Date	State	Issue URL
2	Issue 2	ShenyiCui	ShenyiCui	29/05/2021 22:53:49	open	https://github.com/ShenyiCui/dcbmovies/issues/2
3	Issue 3	ShenyiCui	ShenyiCui	30/05/2021 00:32:10	closed	https://github.com/ShenyiCui/dcbmovies/issues/3

Opened and Closed issues are presented aesthetically with conditional formatting for easy reference.

Hyperlinks for quick navigation to the related issue.

Golden Copy | LittleLives-WrittenTest-May2021 | dcbmovies

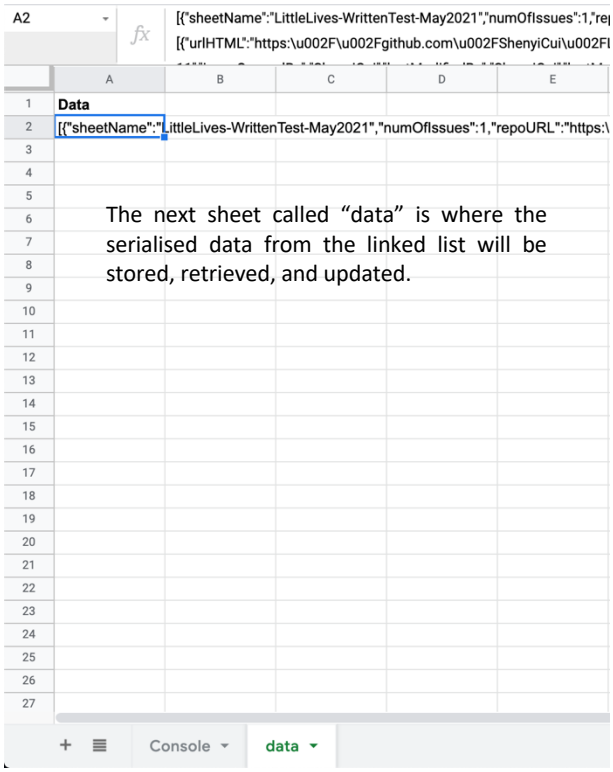
Different tabs used to navigate between different repository's Issue Logs



Timestamp	Message
29/05/2021 22:50:05	Deleting Event
29/05/2021 22:50:12	Deleting...
29/05/2021 22:52:14	Creating Repository
29/05/2021 22:52:59	Adding/Amending Event
29/05/2021 22:53:50	Adding/Amending Event
29/05/2021 22:54:35	Adding/Amending Event
29/05/2021 22:56:56	Adding/Amending Event
29/05/2021 22:57:10	Deleting Event
29/05/2021 22:57:16	Deleting...
30/05/2021 00:31:58	Adding/Amending Event
30/05/2021 00:32:11	Adding/Amending Event

There is no way to log messages on AWS Lambda hence I've written a custom function that logs data onto a virtual console hosted on Google Sheets. This allows me to trace the code in case of an error and find out what operation it was carrying out upon failure.

Console | data



A2 [{"sheetName":"LittleLives-WrittenTest-May2021","numOfIssues":1,"rep
[{"urlHTML":"https:\u002F\u002Fgithub.com\u002FShenyiCui\u002FI

Data
[{"sheetName":"LittleLives-WrittenTest-May2021","numOfIssues":1,"repoURL":"https:\

The next sheet called "data" is where the serialised data from the linked list will be stored, retrieved, and updated.

Console | data

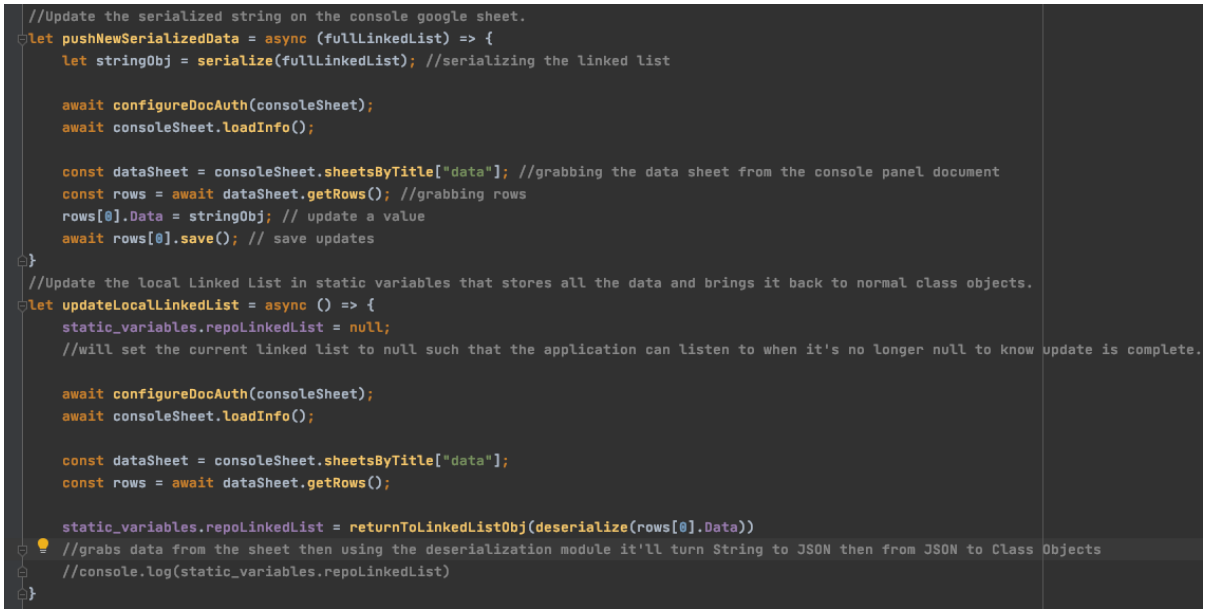
Object/Class Serialization

As I coded the function on AWS Lambda, I realised that Lambda could never save the variables and objects created in the program. Upon further research, I realised that this was due to Lambda's inherent stateless properties as a web service. Lambda, hence, presented a problem, as a large part of my program rested on Object Linked Lists that stored all the Repositories and Issues used to populate and navigate Google Sheets. Therefore, I decided to serialise the entire Object Linked List and save it on a separate Google Sheet to overcome the issue.

I wrote code that first turned an Object Linked List into String. After that, I would store this information on Google Sheets. When the program had to access this data, it would then pull it from Google Sheets and deserialise it, converting it from a String to a JSON Linked List. Lastly, I would iterate through the JSON linked list turning each JSON object into its respective class object; returning the String into an Object Linked List. By designing the code in this manner, I can benefit from the security, reliability, and ease of creating functions on Lambda with the stateful nature of creating code on a server.

```
//Update the serialized string on the console google sheet.
let pushNewSerializedData = async (fullLinkedList) => {
    let stringObj = serialize(fullLinkedList); //serializing the linked list

    await configureDocAuth(consoleSheet);
    await consoleSheet.loadInfo();

    const dataSheet = consoleSheet.sheetsByTitle["data"]; //grabbing the data sheet from the console panel document
    const rows = await dataSheet.getRows(); //grabbing rows
    rows[0].Data = stringObj; // update a value
    await rows[0].save(); // save updates
}
//Update the local Linked List in static variables that stores all the data and brings it back to normal class objects.
let updateLocalLinkedList = async () => {
    static_variables.repoLinkedList = null;
    //will set the current linked list to null such that the application can listen to when it's no longer null to know update is complete.

    await configureDocAuth(consoleSheet);
    await consoleSheet.loadInfo();

    const dataSheet = consoleSheet.sheetsByTitle["data"];
    const rows = await dataSheet.getRows();

    static_variables.repoLinkedList = returnToLinkedListObj(deserialize(rows[0].Data))
    //grabs data from the sheet then using the deserialization module it'll turn String to JSON then from JSON to Class Objects
    //console.log(static_variables.repoLinkedList)
}
```

Figure 1, Functions that will push and pull serialised and deserialised data from Google Sheets



```
//Update the serialized string on the console google sheet.
let pushNewSerializedData = async (fullLinkedList) => {
    let stringObj = serialize(fullLinkedList); //serializing the linked list

    await configureDocAuth(consoleSheet);
    await consoleSheet.loadInfo();

    const dataSheet = consoleSheet.sheetsByTitle["data"]; //grabbing the data sheet from the console panel document
    const rows = await dataSheet.getRows(); //grabbing rows
    rows[0].Data = stringObj; // update a value
    await rows[0].save(); // save updates
}
//Update the local Linked List in static variables that stores all the data and brings it back to normal class objects.
let updateLocalLinkedList = async () => {
    static_variables.repoLinkedList = null;
    //will set the current linked list to null such that the application can listen to when it's no longer null to know update is complete.

    await configureDocAuth(consoleSheet);
    await consoleSheet.loadInfo();

    const dataSheet = consoleSheet.sheetsByTitle["data"];
    const rows = await dataSheet.getRows();

    static_variables.repoLinkedList = returnToLinkedListObj(deserialize(rows[0].Data))
    //grabs data from the sheet then using the deserialization module it'll turn String to JSON then from JSON to Class Objects
    //console.log(static_variables.repoLinkedList)
}
```

Figure 2, Functions that will turn a JSON Linked List into a Class Object Linked List

Modular Code Design

For straightforward code maintenance and for the program to be extensible in the future, the program is designed and split into different modules that combine.

```
const { GoogleSpreadsheet } = require('google-spreadsheet'); //google-spreadsheet library
let static_variables = require('./static-variables')
let {searchForRepo, searchForRepoIndex} = require('./global-functions')
let {returnToLinkedListObj, returnIssueArrToClassObj} = require('./deserialize-classObj')
let serialize = require('serialize-javascript');//serialization library
let moment = require('moment'); //moment library

let { Repository } = require("../classes/Repository");
let { Issue } = require("../classes/Issue");
```

Figure 3, Showing the importing of written modules from the directory

Modular design allows for the different components of the program to be tested individually, only integrating the module after they've been proven to work.

Encapsulation

The created objects have their fields hidden from the programmer to prevent tampering. To access and manipulate data, the programmer must use the accessors and mutators functions. Furthermore, using functions to manage the data within Objects creates a high level of abstraction; this allows programmers to worry less about the implementation and will enable them to focus on the execution of the object. Abstraction allows for easy execution has programmers do not need to relearn the details of each object every time they revisit the code.

Durability

The code is designed to be highly durable to protect itself from the user. The serialised data is stored on a separate Google Document such that only the developer can access it; this will protect the integrity of the data. Furthermore, placing the database away from the main Google Document means that if users accidentally delete sheets, the data can be restored. The program is designed so that it'll repopulate even deleted sheets.

It also doesn't matter what order the Issues are in, as the code will search through the database to find the specific issue corresponding to the row data to update. This allows the user flexibility, letting them order and sort their issues in any way they please without breaking the program.

Disadvantages / Design Flaws

- The Google Document sheet tab names cannot be changed; this may be a nuisance to the user. Any accidental change of sheet tab names will cause the code to ignore the sheet and create a new one with the correct repository name.
- The code runs very slowly. This is likely due to the Object-Oriented design and because it constantly needs to access Google Sheets to push and retrieve data.

Libraries used:

- google-spreadsheet
    - https://www.npmjs.com/package/google-spreadsheet
- serialize-javascript
    - https://www.npmjs.com/package/serialize-javascript
- moment
    - https://www.npmjs.com/package/moment