**Final Project Cloud Developer**

**Functional Purpose of this project**

This project attempts to simulate the workflow of a high-end investment advisory firms that serves other companies that hold investments in their balance sheets. We are not attempting any real business/data enrichment logic. We are simply demonstrating the ability to coordinate activity between AWS and local tiers. A good analogy would be with a project in woodworking class. This wood working project would not result in a truly functional piece of furniture but instead, it just might be a small square object assembled with one piece attached via mortise/tenon with other piece being dovetail and so on just to show that the student has the rudiments of joinery down. Here, I am just demonstrating that I can write serverless solution.

This advisory firm will take investment from clients. These files will contain investment records with enough attributes to model the effects of interest scenarios on these investments. Below specifies the desired work flow in terms of how the files get through the simulations.

Rubric Checklist

**The application allows users to create, update, delete items. Done.** I went further added added s3 event function sendclientUploadNotification

 UploadNotification:

    handler: src/lambda/notifications/sendclientUploadNotification.handler

  GetOutstandingFiles:

    handler: src/lambda/http/getOutstandingFiles.handler

  UpdateQueuItem:

    handler: src/lambda/http/updateQueue.handler

  CreateFileStatus:

    handler: src/lambda/http/createFileStatus.handler

  CreateInvestmentItem:

    handler: src/lambda/http/createInvestment.handler

  UpdateFileStatus:

    handler: src/lambda/http/updateFileStatus.handler

  # TODO: Configure this function

  DeleteFileStatus:

    Handler: src/lambda/http/deleteFileStatus.handler

  GenerateUploadUrl:

    handler: src/lambda/http/generateUploadUrl.handler

**The application allows users to upload a file. Done**

**The application only displays items for a logged in user. Done.**

**Authentication is implemented and does not allow unauthenticated access. Done**

**The code is split into multiple layers separating business logic from I/O related code. Done**. Separate data access classes that lambda calls.

**All resources in the application are defined in the "serverless.yml" file: Done**

*Each function has its own set of permissions.: Done*

**Application has sufficient monitoring. Done – used console.log and logger**

**HTTP requests are validated: This is done for some of the main functions such as update file status**

**Data is stored in a table with a composite key. I went further and added global secondary index to 2 tables.**

**The application allows users to upload a file. Done**

**The application only displays items for a logged in user. Done**

**Scan operation is not used to read data from a database. Done**

**Desired workflow/architecture**

1. Client logs into the investment processor system. (React)
2. Client Adds new file attributes (Name, ect.) to be processed. (Inserts into dynamo via lamba)
3. Client then uploads investment file(json format) to S3
4. S3 event notification fires off to lambda function that adds file to que (Dynamo Db )
5. Local server at advisors premises continuously polls the que by querying the dynamo db queue table for investment to be processed. As a result, a record insert in step 4 leads to this server to download this file (from 3), read the data from the file and insert this data to another Dynamo db investment table. From this table, the advisor would do the simulations (out of scope)

**For this project:**

Actual behavior in project submission is very similar except that to minimize deployment overhead for project evaluation, process 5 takes place in the react front end whenever logging in. This is pseudo architecture for the above.

**Technical Approach:**

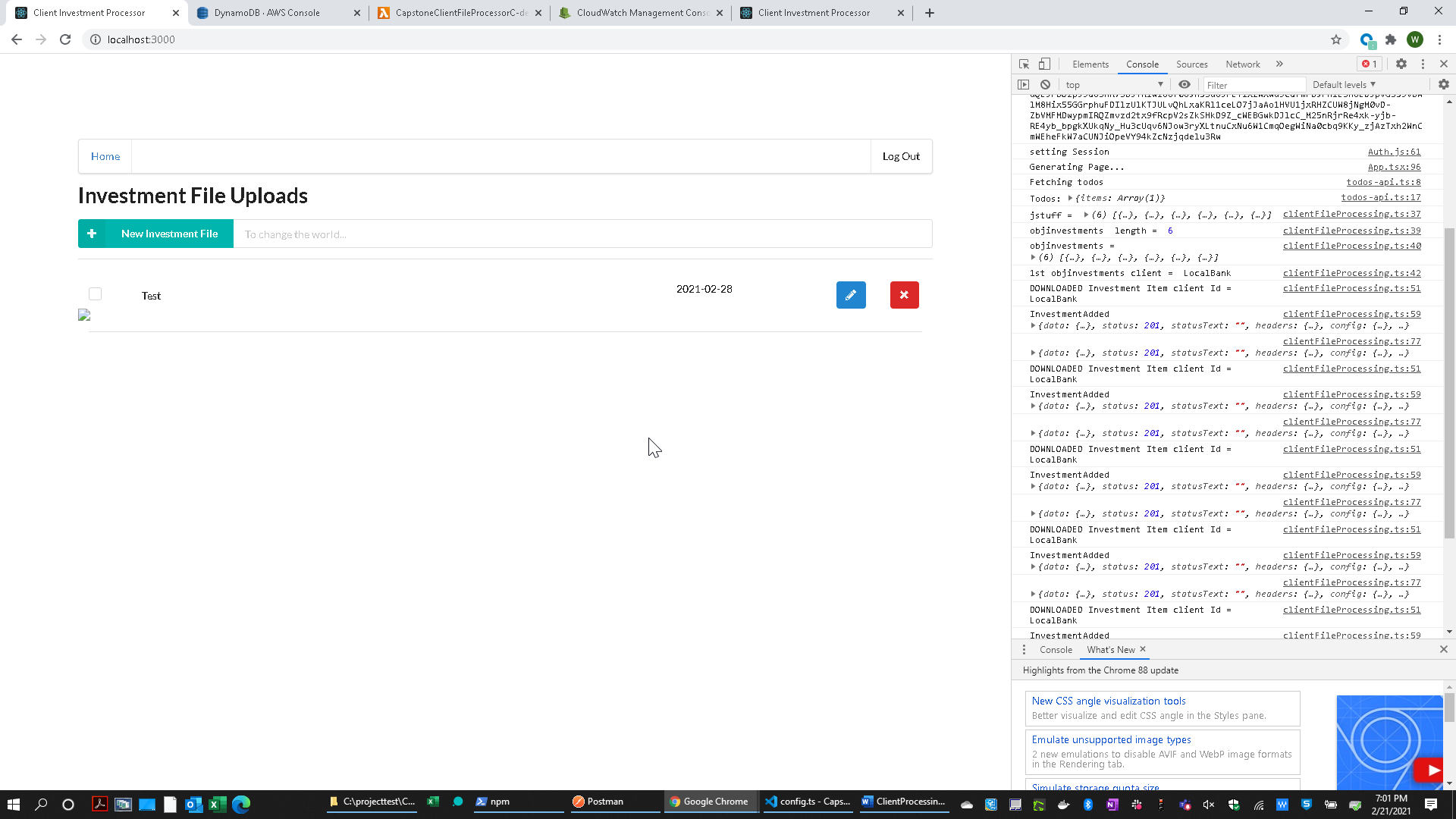
Repo: https://github.com/wjclemens69/CapstoneClientProcessor

This is based off the capstone project. The following has been done to the base project:

1. 3 Dynamo DB tables and with each having a global secondary index.
2. Added interfaces for stronger typed data.
3. Attached object created event to attachment bucket for the queue management piece.
4. Individual Dynamo DB permissions for all functions (Table and global indexes)
5. Total of 9 Lambda functions.
6. Data Access folder for the lambda function calls.
7. The s3 file download and read logic is called from the set session method (after authed) method in React Auth.js. The function is checkClientFileQueu.
8. React app was refactored for label changes.

Instructions:

1. When Checking out the code, set the secret access key to d4qg7eVu/HdMvb3nrGn3UBpu2PHwuUbbFH4HpZxT in config.ts. This enables us to download the s3 file and read from it.
2. Runnpm install and the react (npm run start) via the capstone front end folder.
3. Log into the solution and add a file Item similar to the image add in the Capstone project. (not the actual upload). This will add record to ClientFileStatusC table (dynamo).
4. Upload the file ClientFileInput.json located in the CapstoneBackend/TestFileInput folder.
5. When uploading the file, s3 objectcreated event fires an insert into the Capstone Processing Queue table.
6. Log out and then log back in. When logging back in, the react code (checkClientFileQueu) is called from within setsession. This sees the outstanding queue item, downloads the s3 file, and reads the s3 file records from the file into the dynamo db table called ClientInvestments-dev.



1. Note: This continues to use AUTH0. You can see the file processing taking place in the browser dev console.