

ECE 1779: Project 3

Group Members:

Corey Kirschbaum, #: 997384165

Mohammed Baquir, #:1007489816

Date: December 10, 2020

Contents

Table of Figures	4
Glossary.....	6
1. Introduction	7
2. Stock Portfolio Web Application	8
2.1. Financial Stock API Details	8
2.2. File Directory Layout	9
1.1.1. Flask Web Application.....	10
1.1.2. /app Web Application Files	11
1.1.3. Web Application Log-in.....	12
1.1.4. Login - / or /login.....	12
1.1.5. Password Recovery - / recovery.....	13
3. Stock Portfolio Background Lambda Function.....	22
4. Database Design.....	23
4.1. Application Use Cases for Database	23
4.2. Design.....	24
4.2.1. User Information.....	24
4.2.1.1. User Login Information	25
4.2.1.2. User Stock Info	25
4.2.1.3. User Stock Date Pulled.....	26
4.2.2. Gainers, Losers, Actives.....	26
4.2.2.1. Gainers Date Pulled.....	26
4.2.2.2. Gainers Stocks	27
4.2.2.3. Losers Date Pulled.....	27
4.2.2.4. Losers Stocks	27
4.2.2.5. Actives Date Pulled	27
4.2.2.6. Actives Stocks.....	28
5. How to run the app.....	28
5.1. Amazon S3.....	28
6. AWS Cost.....	28
6.1. Cost Model for 10 users	31
6.2. Cost Model for 1000 users.....	31

6.3. Cost Model for 1,000,000 users.....	32
Future Improvements	32
Group Member Efforts.....	32
Corey Kirschbaum	32
Mohammed Baquir	33
References:	33

Table of Figures

Figure 1: Project 3 Architecture	7
Figure 2: Financial API Account Types	8
Figure 3: Zappa Deploy	10
Figure 4: Zappa Settings	10
Figure 5: API Gateway	11
Figure 6: Bootstrap Style Sheets Used	11
Figure 7: Login Page	12
Figure 8: Password Recovery Page	13
Figure 9: Password Recovery - Bad Email Format	13
Figure 10: Password Recovery - Non-User Email	14
Figure 11: Password Recovery - User Email	14
Figure 12: Password Recovery Email	14
Figure 13: Logout Result	15
Figure 14: Change Password Page	15
Figure 15: Admin DB Scan to View All Users Info	16
Figure 16: Add/Delete User Page	16
Figure 17: Add User Page	17
Figure 18: Most Active Stocks Page	18
Figure 19: Most Gaining Stocks Page	18
Figure 20: Most Losing Stocks Page	19
Figure 21: My Portfolio Page	20
Figure 22: Find a Stock Page	21
Figure 23: Find a Stock Page - With Results	21
Figure 24: Stock Details Page	22
Figure 25: Snippet of Welcome Email	23
Figure 26: Snippet of Cloudwatch Log	23
Figure 27: DynamoDB Table - User Information	25
Figure 28: DynamoDB Table - User Information Schema	25
Figure 29: DynamoDB Table - User Stock Information Schema	25
Figure 30: DynamoDB Table - User Stock Date Pulled Schema	26
Figure 31: DynamoDB Table - Gainers, Losers, Actives	26
Figure 32: DynamoDB Table - Gainers Date Pulled Schema	26
Figure 33: DynamoDB Table - Gainers Stock Information Schema	27
Figure 34: DynamoDB Table - Losers Date Pulled Schema	27
Figure 35: DynamoDB Table - Losers Stock Information Schema	27
Figure 36: DynamoDB Table - Actives Date Pulled Schema	27
Figure 37: DynamoDB Table - Actives Stocks Information Schema	28
Figure 38: Finance API Paid Tier	29
Figure 39: Application Number of HTTP Requests	30
Figure 40: Cloudwatch Cost	30

Figure 41: DynamoDB Table Details.....	31
Figure 42: Cost Breakdown per 10 Users.....	31
Figure 43: Cost Estimator per 10 users	31
Figure 44: Cost Breakdown per 1000 users	32
Figure 45: Cost Estimator per 1000 users.....	32
Figure 46: Cost Breakdown per 1 million users	32
Figure 47: Cost Estimator per 1 million users	32

Glossary

S3	Amazon Simple Storage Service
Lambda	Amazon Lambda Function
ECE	Electrical and Computer Engineering
AWS	Amazon Web Services
App	Application
DynamoDB	AWS database
Stock Symbol = Stock Ticker = Stock ID	Is the stock alias name used by stock markets to identify a company.

1. Introduction

The following report is for ECE1779 Cloud Computing Project 3.

During the processing of brainstorming project ideas much difficulty was experienced in narrowing down potential project as permissions issues occurred when attempting to use various Amazon AWS services, due to using a student account.

Potential Ideas and issues found:

- Using AWS recognition to detect/classify images and then perform lyrical sentence generation from the key words returned from AWS recognition. For creating picture captions for users when they upload to social media
- Using AWS recognition for anything (like detect famous person)
- Using AWS sage maker, or any of the AI services
- Creating background lambda for monitoring data on a schedule
- ... many more

The final project idea is a stock portfolio manager. One AWS lambda will be running behind an API Gateway (deployed with zappa) to take user http requests, with the lambda function running the main stock portfolio web application. The stock portfolio application will allow users to login, view and add stocks to their portfolio. Users can additionally search for stocks and see more specific details about the requested stock to help users with their decision of buying or selling.

A background lambda will be running that will capture events from the DynamoDB database for when a new user is created and send a welcome email to the new user.

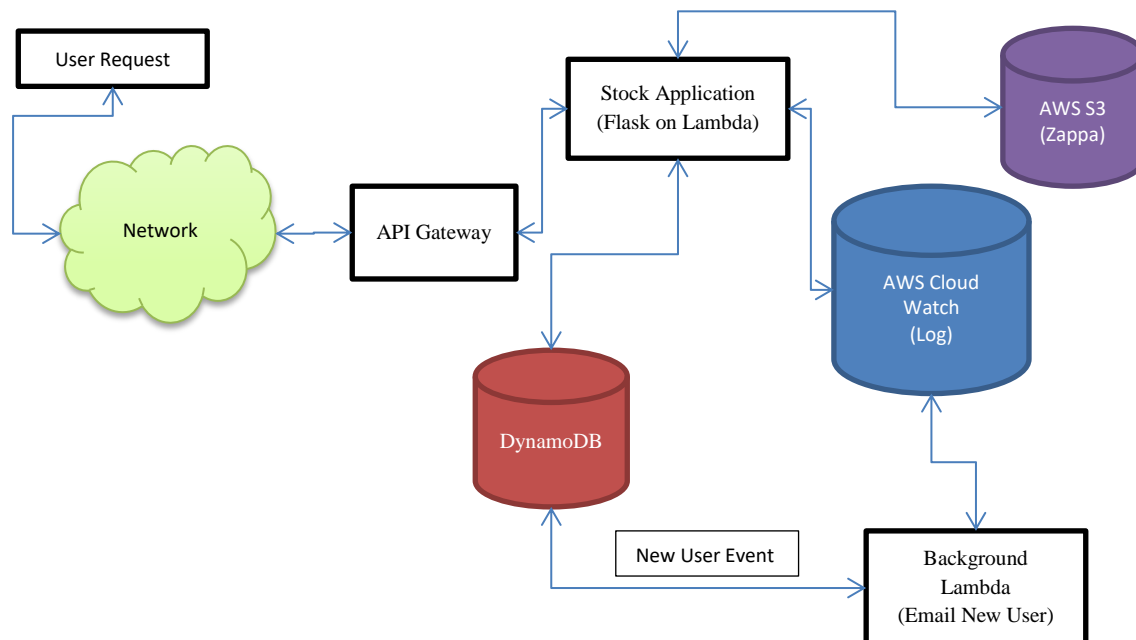


Figure 1: Project 3 Architecture

This Stock Portfolio application is called “Fill My Stocks”, as it rhymes with “Fill My Socks” which as it is close to the holidays the name was funny and catchy.

This application is creative and useful as we were able to work around the limitations of AWS student account permission issues to produce an application (on our student account) that provides a simple and easy means for users to play around with stocks in a risk free manner. Users can use the tool as a real time simulator or they can enter their actually stock trades for easier interface and means of tracking their portfolio for less money. Our group members personally felt this application was easy to use and did provide a quick means of looking at the performance for different stocks.

The application can be expanded a lot more with additional features if the project continued to production and full time deployment.

2. Stock Portfolio Web Application

2.1.Financial Stock API Details

The stock portfolio web application will be using/collecting stock information from <https://financialmodelingprep.com>. Currently the application is using the free tier version which limits to 250 requests per day, which is why the Stock Portfolio web application is using three account keys in a round robin fashion to minimize possibility of using up all 250 requests per a day. As well, some parts of the Stock Portfolio web application attempts to cache API results in the dynamoDB table on a duration period to minimize number of API calls. If the Stock Portfolio Web Application was, in the future, to be production use for users one of the higher end API accounts can be used which have unlimited API calls.

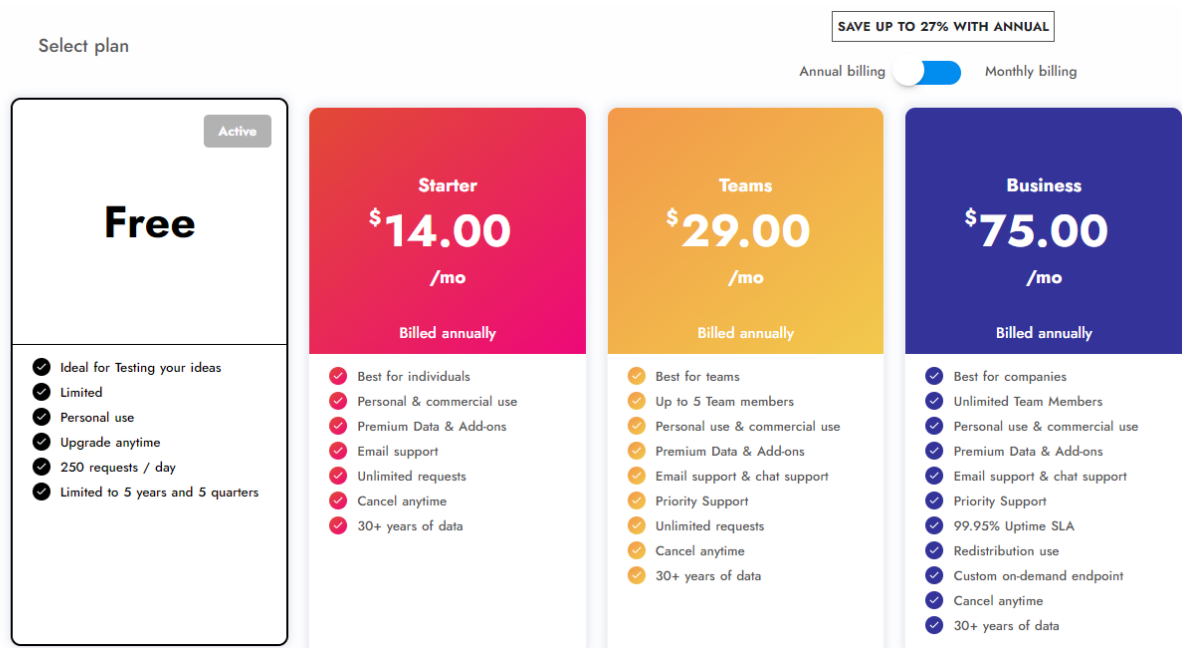


Figure 2: Financial API Account Types

See below or `app/stock_api_util.py` or <https://financialmodelingprep.com/developer/docs/> for details.

```
stock_api_keys = ["262818c9a1b67405a31443da2d8a0912", "68385f2403ec5217fdff3df55822cecd",  
                  "e4e15b1f03f4a991889497d721fabfae", "e9c3a15a5b6cdca5f924d27cc05c0b9b"]
```

2.2.File Directory Layout

```
.  
├── README.md  
├── app  
│   ├── __init__.py  
│   ├── add_user.py  
│   ├── change_password.py  
│   ├── config.py  
│   ├── forms.py  
│   ├── login.py  
│   ├── main.py  
│   ├── my_stocks.py  
│   ├── recovery.py  
│   ├── s3_utils.py  
│   ├── static  
│   │   └── flot  
│   ├── stock_api_util.py  
│   ├── templates  
│   │   ├── add_user.html  
│   │   ├── base.html  
│   │   ├── change_password.html  
│   │   ├── find_stock.html  
│   │   ├── login.html  
│   │   ├── my_stocks.html  
│   │   ├── recovery.html  
│   │   ├── stock_changes.html  
│   │   ├── stock_details.html  
│   │   ├── stock_update.html  
│   │   └── user_new.html  
│   ├── top_stocks.py  
│   └── user_model.py  
├── group.txt  
├── lambda_function.py  
├── requirements.txt  
├── run.py  
├── zappa_settings.json  
├── zappa_settings_corey.json  
└── zappa_settings_mbaquir.json
```

The above file structure output above was generated with the `>tree` command.

1.1.1. Flask Web Application

The application's flask web application is deployed using zappa. Zappa will deploy the flask application as a lambda function and an AWS API Gateway to allow external user http requests to be processed by the lambda function. AWS lambda will handle the application scaling.

Flask application name = webapp

```
(.venv) corey@Home_Terminal:~/ece1779_a3$ zappa deploy dev
Calling deploy for stage dev..
Downloading and installing dependencies..
- zopec.interface==5.2.0: Using locally cached manylinux wheel
- markupsafe==1.1.1: Downloading
100%|████████████████████████████████████████████████████████████████████████████████| 32.7k/32.7k [00:00<00:00, 9.13MB/s]
- lazy-object-proxy==1.4.3: Using locally cached manylinux wheel
- greenlet==0.4.17: Using locally cached manylinux wheel
- gevent==20.9.0: Using locally cached manylinux wheel
Packaging project as zip.
Uploading ece1779-a3-dev-1607537859.zip (19.3MiB)..
100%|████████████████████████████████████████████████████████████████████████████████| 20.2M/20.2M [00:02<00:00, 9.80MB/s]
Uploading ece1779-a3-dev-template-1607537880.json (1.5KiB)..
100%|████████████████████████████████████████████████████████████████████████████████| 1.57k/1.57k [00:00<00:00, 16.8kB/s]
Waiting for stack ece1779-a3-dev to create (this can take a bit)..
75%|████████████████████████████████████████████████████████████████████████████████| 3/4 [00:13<00:04, 4.35s/res]
Deploying API Gateway..
Deployment complete!: https://00ibdyifte.execute-api.us-east-1.amazonaws.com/dev
```

Figure 3: Zappa Deploy

The Zappa settings can see in the zappa_settings.json

```
{
  "dev": {
    "app_function": "app.webapp",
    "aws_region": "us-east-1",
    "profile_name": "default",
    "project_name": "ece1779-a3",
    "runtime": "python3.8",
    "s3_bucket": "ece1779.zappa.a3",
    "keep_warm": false,
    "debug": true,
    "log_level": "DEBUG",
    "http_methods": ["GET", "POST"],
    "use_precompiled_packages": true
  }
}
```

Figure 4: Zappa Settings

The Lambda functions API Gateway created by Zappa can be seen below.

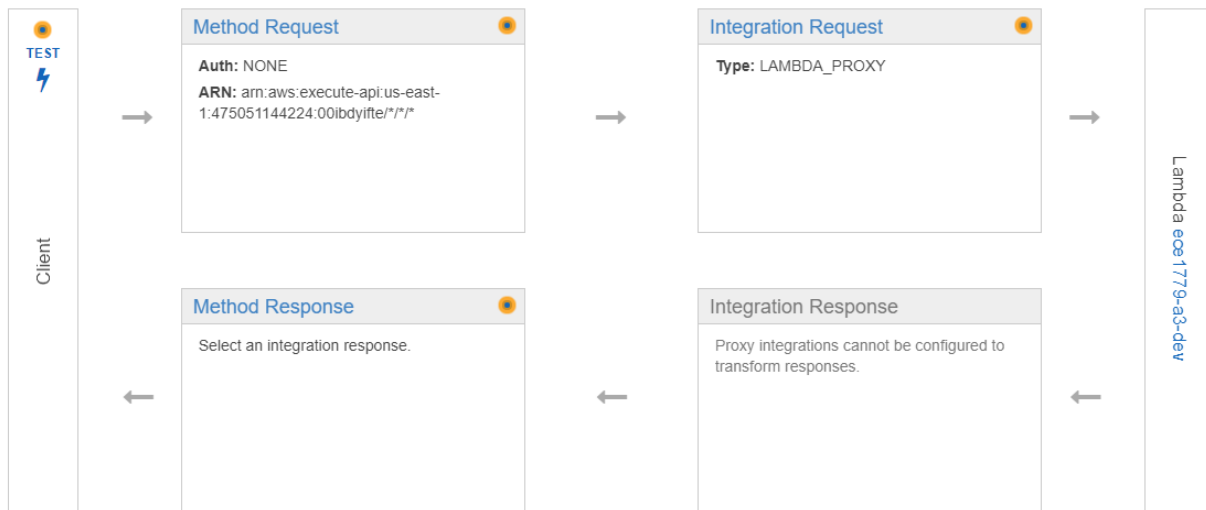


Figure 5: API Gateway

Bootstrap style sheets are used to make a more visually appealing web site. See `app/templates/base.html` for bootstrap styles used or see below.

```

<head>
  <title>Fill My Stocks</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
  <script type="text/javascript" src="{{ url_for('static', filename='flot/jquery.js') }}"></script>
  <script type="text/javascript" src="{{ url_for('static', filename='flot/jquery.flot.js') }}"></script>
  <style>

```

Figure 6: Bootstrap Style Sheets Used

1.1.2. /app Web Application Files

All code files for our web application is contained with the /app folder. At the high level this folder contains:

- /static/flot -> API for plotting data in web html pages
- /templates -> all our HTML files
- __init__.py -> contains our webapp configurations
 - o Email info
 - o Secret key used by Flask for security in sessions (Flask Login) and to prevent CSRF attacks (Flask WTF for forms)
 - o Import all of our application python files
- *.py -> These are all of implemented application files in python (flask application endpoints)
 - o Login
 - o View User Stocks Portfolio
 - o Get Top Stock Changers
 - o Find Stocks
 - o Add/Update Stocks
 - o Password Recovery

- Change Password
- Add/Delete User

1.1.3. Web Application Log-in

We have already created an admin account in the web application. To view pages that only admin can see for example “Creating and deleting users” use the following admin credentials:

username: mbaquir

Password: password

This will allow you to see the admin privileged pages.

1.1.4. Login - / or /login

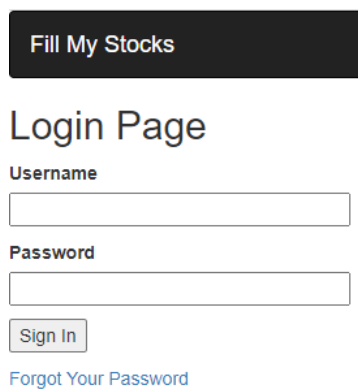
The main entry point to the web application is the login page. As we are using flask login package, we configured it to redirect all none-logged-in URL requests to the login page. This is further enforced with flask login decorator `@login_required`, which enforces a flask route endpoint will only generate if user is logged-in, else redirects to login page. Once logged in the user will then be redirected to either the `most_active_stocks` page or the URL they were originally attempting to access. Note: `main.py` only forwards between login and the `most_active_stocks` page to make it easier to update and follow where the code is for main page redirection.

The page contains a link to the password recovery page as well.

Flask flash messages are used to show additional information. The title and the flash message are part of the `base.html` page all HTML pages inherit from.

User login details are compared against our DynamoDb database users information (error checking, if fail flash message below). For database design please see the database design section further in this document.

The `werkzeug.security` package is used to `generate_password_hash()` and `check_password_hash()`, see `user_model.py` for details.



Fill My Stocks

Login Page

Username

Password

Sign In

[Forgot Your Password](#)

Figure 7: Login Page

Related files:

- login.py
- /template/login.html
- user_model.py
- __init__.py -> sets /login endpoint as login_view for flask login

1.1.5. Password Recovery - / recovery

If a user forgets their password, on the login page a link is provided that will take the user to a page that will allow them to enter their email address. If a user with the specified email address exists in the database, an email with a new password is sent to the user for them to login, where they can change their password. This is accomplished by generating a new random string as a password and updating the user's password in the database (users table) to be this new password. The email is sent using Gmail.

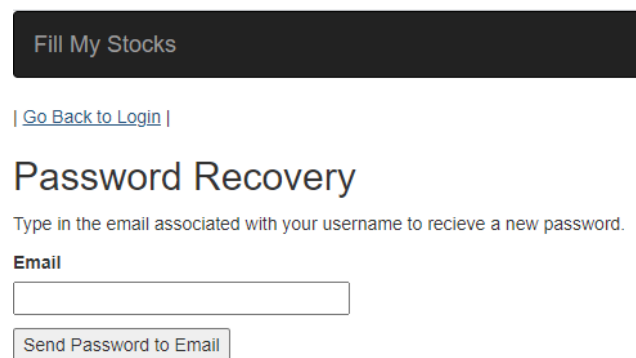


Figure 8: Password Recovery Page

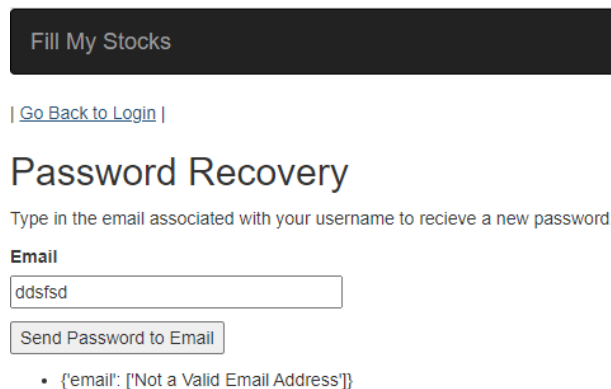


Figure 9: Password Recovery - Bad Email Format

Fill My Stocks

[| Go Back to Login |](#)

Password Recovery

Type in the email associated with your username to recieve a new password.

Email

corey.kbaum@gmail.com

Send Password to Email

- Could not find an account with that email

Figure 10: Password Recovery - Non-User Email

Fill My Stocks

Login Page

Username

Password

Sign In

[Forgot Your Password](#)

- Email has been sent with a new password. Please use the new password to log in.

Figure 11: Password Recovery - User Email

Search Current Mailbox

Current Mailbox

All Unread

By Date Newest

Today

ece1779.a1.webapp@gma...
Password Recovery for Fill My Sto...
EXTERNAL EMAIL: 7:50 PM

ECE 1779 on Piazza
[Instr Note] Dec.10 Demo Link
EXTERNAL EMAIL: Instructor shawn 7:43 PM

ece1779.a1.webapp@gma...
Welcome to Fill My Stocks!!
EXTERNAL EMAIL: Hi "mbaquir1"! 7:28 PM

Corey Kirschbaum

Reply Reply All Forward

ece1779.a1.webapp@gmail.com

Mohammed Baquir

Password Recovery for Fill My Stock App

EXTERNAL EMAIL:

Hello,
This is your new temporary password:'oqzdyL9wHA1UYYaAsJb6lvJko'.
Please login in with the new password.
Change your password after you log in.
Thanks,
Fill My Stock App password Recovery Team

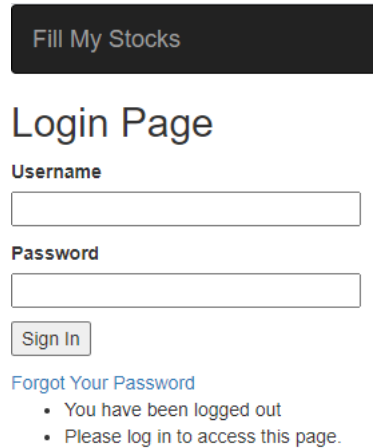
Figure 12: Password Recovery Email

Related files:

- recovery.py
- /template/recovery.html
- __init__.py -> sets up email in config

1.1.6. Logout - /logout

All HTML pages inherit the base.html template which contains a navigation bar. The navigation bar will only display the necessary URL links based on if the user is logged in or is an “admin”. The logout URL link will be available for all logged in user and will log the user out from flask login - logout_user(), and redirect the user back to the login page with a successful logout message.



Fill My Stocks

Login Page

Username

Password

Sign In

[Forgot Your Password](#)

- You have been logged out
- Please log in to access this page.

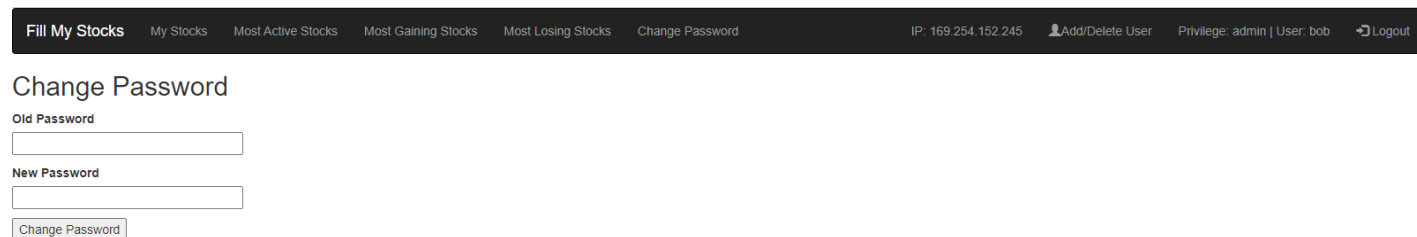
Figure 13: Logout Result

Related files:

- login.py
- /template/login.html
- /template/base.html

1.1.7. Change Password - /change_password

All users have the ability to change their password. Once a user is logged-in the base.html template will generate a navigation bar which contains a link to the change password page. The base.html does this dynamic loading with the use of “{% if current_user.is_authenticated %}”. Note: Just as the login page, flask forms are used to provide some validation and security on the client side. This will update the user’s password in the DynamoDB database.



Fill My Stocks My Stocks Most Active Stocks Most Gaining Stocks Most Losing Stocks Change Password IP: 169.254.152.245 Add/Delete User Privilege: admin | User: bob Logout

Change Password

Old Password

New Password

Change Password

Figure 14: Change Password Page

Related files:

- change_password.py
- /template/change_password.html

1.1.8. Add/Delete User -> /add_user

This page is accessible from the navigation bar once logged in. However, it will only appear if the user account type is “admin”. Current user account type is visible at the top left corner of the navigation bar.

If a regular user tries to access the page, it will be redirected back to the main page. The base.html handles the visibility of this link similar to that of change password.

The /add_user page will collect all users information from the DynamoDB table and display it in table format. All users’ information will be collected using a scan over the DynamoDB table. It is understood that a scan is a slow and expensive operation, and that No-SQL/DynamoDB tables should be designed to not use scans. However, for all other queries on the table a lookup using the partition key and sort key are used. As this feature will only be used by Admins, to see all users, we let our DB design require a scan to collect this information.

```
pe = 'user_name,email,user_type'
response = users.scan(
    FilterExpression=Key('pk').begins_with("user#") & Key('sk').begins_with("info#"),
    ProjectionExpression=pe
)
```

Figure 15: Admin DB Scan to View All Users Info

The ability to delete or create a user is provided, which on success will perform the correct operation in the database’s users information to reflect. Note: creating a user involves clicking the create button which redirects to “/add_user/create”. We only allow unique emails across all users, for recovery purposes, so new users cannot create an account with the same email as an existing user.

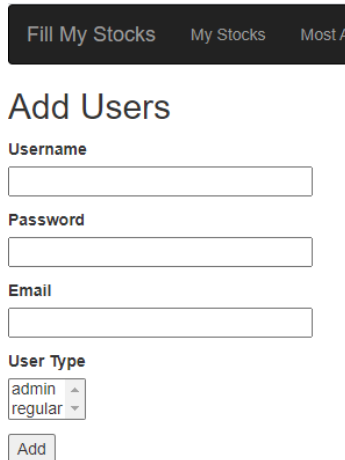
[Fill My Stocks](#) [My Stocks](#) [Most Active Stocks](#) [Most Gaining Stocks](#) [Most Losing Stocks](#) [Change Password](#) IP: 169.254.243.205 [Add/Delete User](#) Privilege: admin | User: bob [Logout](#)

Add, Delete or Edit Users

Create New User

Name	Email	User Type	
bob	corey.kirschbaum@mail.utoronto.ca	admin	Delete
corey	corey.kbaum@gmail.com	admin	Delete
mbaquir	someEmail@gmail.com	regular	Delete

Figure 16: Add/Delete User Page



Fill My Stocks My Stocks Most A

Add Users

Username

Password

Email

User Type
admin
regular

Add

Figure 17: Add User Page

Related files:

- `add_user.py`
- `/template/add_user.html`

1.1.9. Most Active Stocks -> `/most_active_stocks`

The most active stocks page acts at the main home page of the Stock Portfolio application, once user's login they will be redirected to the most active stocks page.

The most active stocks page will display the stocks with the most activity based on the returned result from the `https://financialmodelingprep.com/api` using the "actives" endpoint. However, to reduce the number of API calls when using the free tier, the API results are cached in the dynamoDB and are only updated if they are 30 minutes or older. Therefore, the first time most active stocks is executed users may find a delay for the page to load, but subsequent requests will be fast as the data will be pulled from the database.

The most active stocks page can be seen below. It will additionally provide buttons to allow users to see stock details or add the stock to their portfolio.

Fill My Stocks	My Stocks	Most Active Stocks	Most Gaining Stocks	Most Losing Stocks	Change Password	IP: 169.254.243.205	Add/Delete User	Privilege: admin User: bob	Logout
----------------	-----------	--------------------	---------------------	--------------------	-----------------	---------------------	-----------------	------------------------------	--------

Most Active Stocks

Stock ID	Stock Name	Latest unit price	Change Amount	Change Percent		
GLSI	Greenwich LifeSciences, Inc.	79.31	74.11	1425.19	Details	Add
AI	Arlington Asset Investment Corp	90.72	48.72	116.00	Details	Add
RCKT	Rocket Pharmaceuticals Inc	57.09	25.06	78.24	Details	Add
MTSC	MTS Systems Corp	58.53	20.01	51.95	Details	Add
BQ	Boqi Holding Limited	6.08	551777980.0	22.58	Details	Add
AZO	Autozone Inc	1134.79	38.8	3.54	Details	Add
NVR	NVR Inc	4007.5	77.72	1.98	Details	Add
BRK-A	Berkshire Hathaway Inc	340517	-1764.0	-0.52	Details	Add
CABO	Cable One Inc	2060.01	-29.0	-1.39	Details	Add
MTD	Mettler-Toledo International Inc	1144.65	-16.68	-1.44	Details	Add
GOOG	Alphabet Inc.	1786.21	-32.34	-1.78	Details	Add

Figure 18: Most Active Stocks Page

1.1.10. Most Gaining Stocks -> /stocks_gainers

The most gaining stocks page will display the stocks with the most gains based on the returned result from the <https://financialmodelingprep.com/api> using the “gainers” endpoint. However, to reduce the number of API calls when using the free tier, the API results are cached in the dynamoDB and are only updated if they are 30 minutes or older. Therefore, the first time this page is accessed users may find a delay for the page to load, but subsequent requests will be fast as the data will be pulled from the database.

The most gaining stocks page can be seen below. It will additionally provide buttons to allow users to see stock details or add the stock to their portfolio.

Fill My Stocks	My Stocks	Most Active Stocks	Most Gaining Stocks	Most Losing Stocks	Change Password	IP: 169.254.156.29	Add/Delete User	Privilege: admin User: bob	Logout
----------------	-----------	--------------------	---------------------	--------------------	-----------------	--------------------	-----------------	------------------------------	--------

Most Gaining Stocks

Stock ID	Stock Name	Latest unit price	Change Amount	Change Percent		
XBIO	Xenetic Biosciences Inc	3.935	2.855	264.35	Details	Add
AI	Arlington Asset Investment Corp	95.98	53.98	128.52	Details	Add
RCKT	Rocket Pharmaceuticals Inc	58.185	26.155	81.66	Details	Add
SLS	Sellas Life Sciences Group Inc	6.1999	2.5099	68.02	Details	Add
MTSC	MTS Systems Corp	58.56	20.04	52.02	Details	Add
YQ	17 Education & Technology Group Inc.	20.79	5.73	38.05	Details	Add
BNTC	Benitec Biopharma Inc	3.725	0.965	34.96	Details	Add
CATM	Cardtronics PLC	34.38	8.51	32.90	Details	Add
DIVC	Citigroup Inc. C-Tracks ETN Miller/Howard Strategic Dividend Reinvestors Due 9/16/2014	30.9196	6.9196	28.83	Details	Add
OILD	ProShares UltraPro 3x Short Crude Oil	45.32	9.32	25.89	Details	Add
QS	QuantumScape Corporation	72.88	14.98	25.87	Details	Add

Figure 19: Most Gaining Stocks Page

1.1.11. Most Losing Stocks -> /stocks_losers

The most losing stocks page will display the stocks with the most losses based on the returned result from the <https://financialmodelingprep.com/api> using the “losers” endpoint. However, to reduce the number of API calls when using the free tier, the API results are cached in the dynamoDB and are only updated if they are 30 minutes or older. Therefore, the first time this page is accessed users may find a delay for the page to load, but subsequent requests will be fast as the data will be pulled from the database.

The most losing stocks page can be seen below. It will additionally provide buttons to allow users to see stock details or add the stock to their portfolio.

Fill My Stocks My Stocks Most Active Stocks Most Gaining Stocks Most Losing Stocks Change Password IP: 169.254.156.29 Add/Delete User Privilege: admin User: bob Logout									
Most Losing Stocks									
Stock ID	Stock Name	Latest unit price	Change Amount	Change Percent					
FEYE	FireEye Inc	13.5704	-1.9496	-12.56	Details	Add			
AUTL	Autolus Therapeutics PLC	9.185	-1.335	-12.69	Details	Add			
RENN	Renren Inc	5.99	-0.91	-13.19	Details	Add			
LITB	LightInTheBox Holding Co Ltd	2.4601	-0.3799	-13.38	Details	Add			
LYL	Dragon Victory International Ltd	2.6126	-0.4074	-13.49	Details	Add			
ONCY	Oncolytics Biotech Inc	3.32	-0.52	-13.54	Details	Add			
SOL	ReneSola Ltd	5.235	-0.825	-13.61	Details	Add			
TUSK	Mammoth Energy Services Inc	2.75	-0.44	-13.79	Details	Add			
GMDA	Gamida Cell Ltd	9.48	-1.52	-13.82	Details	Add			
OBLN	Obalon Therapeutics Inc	1.693	-0.277	-14.06	Details	Add			
KLXE	KLX Energy Services Holdings Inc	7.2799	-1.2001	-14.15	Details	Add			

Figure 20: Most Losing Stocks Page

1.1.12. My Stocks -> /my_stocks

Users can view their portfolio on the My Stocks page, as seen below. User can view all the stocks in their portfolio, which are stored in the DynamoDB, their portfolio evaluation, and their gains. The page allows users to add, remove, update, or view details on the stocks in their portfolio.

My Stocks

Add New Stock

Stock ID	Stock Name	Quantity	Most Recent Price	Total Value	Pocket	Changes			
AMD	Advanced Micro Devices, Inc.	10	91.7758	915.116	-939.69	-24.57	Add More	Details	Remove
FB2A.DE	Facebook Inc	20	227.0	4540.0	-4706.00	-166.00	Add More	Details	Remove
GLSI	Greenwich LifeSciences, Inc.	13	69.9439	1102.27	-1170.00	-67.73	Add More	Details	Remove
INTC	Intel Corporation	13	50.475	654.355	-648.18	6.18	Add More	Details	Remove
TD	The Toronto-Dominion Bank	13	56.88	737.88	-734.09	3.79	Add More	Details	Remove

Current Portfolio Value:
\$7762.64 (\$-248.33)

seven thousand, seven hundred and sixty-two point six four

Figure 21: My Portfolio Page

The algorithm to find the gains/loss for a user's portfolio is the following:

- The database will track all the stocks per a user in the same partition
- Each stock stored will contain:
 - quantity of shares user owns
 - current price per share of stock (updated every 15 minutes)
 - Pocket = the amount of money in the user's pocket generated by the stock
 - Total value/portfolio = the amount of money in the user's portfolio as a result of the amount they own of the stock

Here’s an example of how the algorithm works:

1) User buys 5 shares of a stock at \$100 per a share
+5 @ \$100 => 5 shares | Portfolio=5*\$100=\$500 Pocket=-\$500 => Portfolio + Pocket = 0 gain

2) User buys 6 more shares of the same stock at \$150 per a share
+6 @ \$150 => 5+6=11 shares | Portfolio=11*\$150=\$1650 Pocket=Pocket - 6*\$150=-\$1400 => \$250 gain

3) User sells 3 shares of the stock at \$180 per a share
-3 @ \$180 => 11-3=8 shares | Portfolio=8*\$180=\$1440 Pocket=Pocket + 3*\$180=-\$860 => \$580 gain

Total = Portfolio + Pocket = \$580 (positive)

1.1.13. Find Stock -> /find

The find stock page, which is only accessible by clicking the “Add New Stock” button on my stocks page, allows the user to type in a stock symbol or company name and get a listing of stocks with the symbol provided or similar. Then the user can select the appropriate stock entry from the listing and view details on the stock.

Find a stock to Add

Stock Name or ID

Find Stock

Figure 22: Find a Stock Page

[Fill My Stocks](#)
[My Stocks](#)
[Most Active Stocks](#)
[Most Gaining Stocks](#)
[Most Losing Stocks](#)
[Change Password](#)

IP: 169.254.14.197
[Add/Delete User](#)
Privilege: admin | User: bob
[Logout](#)

Find a stock to Add

Stock Name or ID

Find Stock

Stock ID	Stock Name	Stock Exchange	
AMDVX	American Century Mid Cap Value Fund R6 Class	Nasdaq	Details
DOX	Amdocs Limited	NasdaqGS	Details
CPT	Camden Property Trust	NYSE	Details
AMD	Advanced Micro Devices, Inc.	NasdaqGS	Details
CAC	Camden National Corporation	NasdaqGS	Details
DAMDX	Dunham Monthly Distribution Fund Class A	Nasdaq	Details
AMD.DE	Advanced Micro Devices, Inc.	XETRA	Details
AMDIND.NS	AMD Industries Limited	NSE	Details
KAMDHENU.NS	Kamdhenu Limited	NSE	Details

Figure 23: Find a Stock Page - With Results

1.1.14. Details Stock -> /my_stocks/details/< string:stock_id>

The details page will allow users to view the stock price, the historical VWAP data in a plot using the historical-price-full API endpoint from <https://financialmodelingprep.com/api>, and some news on the stock using the stock_news API endpoint from <https://financialmodelingprep.com/api>.

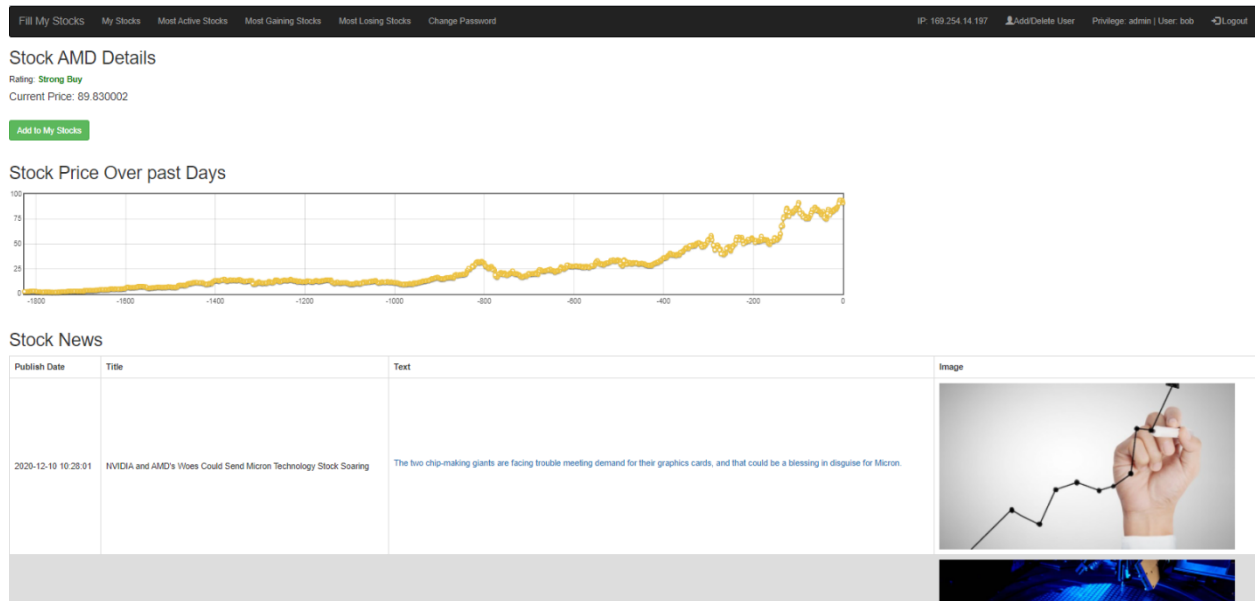


Figure 24: Stock Details Page

3. Stock Portfolio Background Lambda Function

From the Stock Portfolio Web Application file directory layout section above the “lambda_function.py” contains the code used by the background lambda function.

The background lambda function will get executed on a DynamoDB event. The lambda function will parse the event and only perform when a new user has been added. When a new user is added the lambda function will send a welcome email to the user.

DynamoDB is added as a trigger to the lambda function. This means that the lambda function will run on any actions done on the DynamoDB which includes (INSERTS,REMOVE,MODIFY).

Our lambda function filters the event to only run when the eventName is “INSERT”. We check to see if a new user has been added as part of the DynamoDB event. We check to make sure email is part of the event to the DynamoDB. We then sent a welcome email to the user and notify them of their registration.

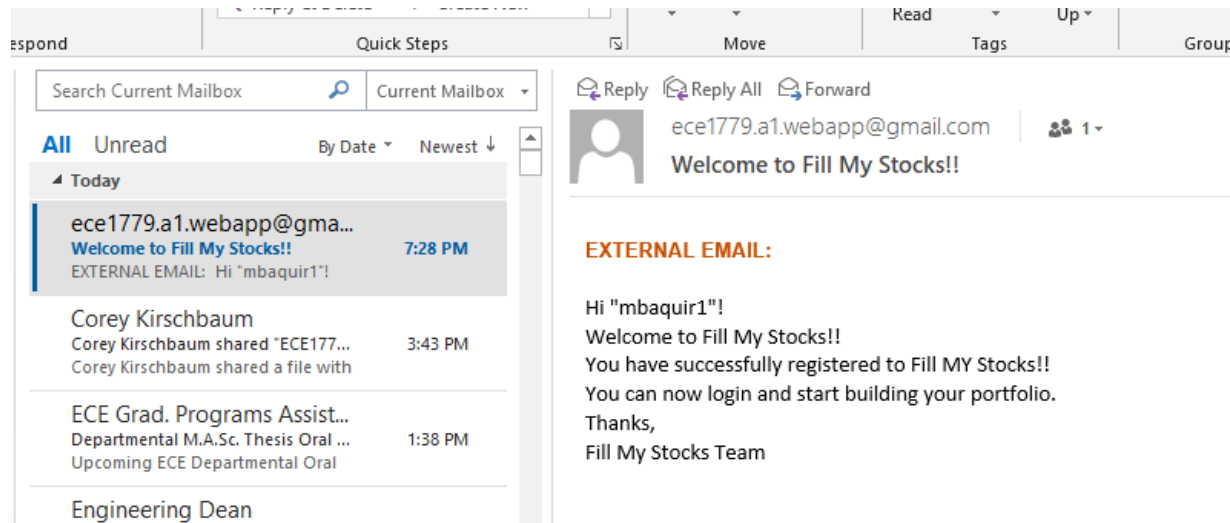


Figure 25: Snippet of Welcome Email

▶	2020-12-09T19:28:02.089-05:00	START RequestId: 1f006085-71d5-4f9d-a1be-241accff5c4d Version: \$LATEST
▶	2020-12-09T19:28:02.092-05:00	event_name: REMOVE
▶	2020-12-09T19:28:02.093-05:00	END RequestId: 1f006085-71d5-4f9d-a1be-241accff5c4d
▶	2020-12-09T19:28:02.093-05:00	REPORT RequestId: 1f006085-71d5-4f9d-a1be-241accff5c4d Duration: 1.26 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 50 MB
▶	2020-12-09T19:28:17.316-05:00	START RequestId: 4a0a3c54-9ed9-4a20-aa7a-03846e0fff98 Version: \$LATEST
▶	2020-12-09T19:28:17.319-05:00	event_name: INSERT
▶	2020-12-09T19:28:17.319-05:00	receiver_email: "mohammed.baquir@mail.utoronto.ca"
▶	2020-12-09T19:28:17.319-05:00	user_name: "mbaquir1"
▶	2020-12-09T19:28:18.085-05:00	Email sent!
▶	2020-12-09T19:28:18.086-05:00	END RequestId: 4a0a3c54-9ed9-4a20-aa7a-03846e0fff98
▶	2020-12-09T19:28:18.086-05:00	REPORT RequestId: 4a0a3c54-9ed9-4a20-aa7a-03846e0fff98 Duration: 767.52 ms Billed Duration: 768 ms Memory Size: 128 MB Max Memory Used: 50 MB

Figure 26: Snippet of Cloudwatch Log

4. Database Design

AWS DynamoDB database was used for this project, with a database design requiring only a single table.

Researching DynamoDB best practices and design, reviewing course lecture material, and going through Amazons documentation on DynamoDB; it was suggested that it was best practice to only use one table to store all of an applications data [1, 2]. This was confusing coming from a relational database background, however it did result is devising our database design to be optimal for our use cases.

The AWS DynamoDB table is:

```
DYNAMO_DB_TABLE = 'ece1779_a3'
```

See app/s3_utils.py for database table name.

4.1.Application Use Cases for Database

The database will be used for the following queries:

- Find user information on login (user name, password)
- Find the stocks in a user's portfolio (user name -> stock info)
- Check if user exists with email during recovery (email -> user info)

- Get all users and their information when Administrator request on Add/Remove page

The database will additionally support extra features for caching data from the API used to collect stock information (<https://financialmodelingprep.com/api>) to reduce number of API requests made. This requirement came to exist due to the limitations of the free tier account for the API. However, it was noticed after caching the most gainers, losers, and actives in the database the /stock_gainers, /stock_losers, and /stock_actives pages loaded visually faster. Hence, such a feature might be used in production deployment of the Stock Portfolio web application.

- Store stock gainers information, with the ability to get all entries
- Store stock losers information, with the ability to get all entries
- Store stock actives information, with the ability to get all entries
- Store date/time an entry was added to database
 - For gainers store one date/time value for all entries
 - For losers store one date/time value for all entries
 - For actives store one date/time value for all entries
 - Store one date/time value per user for deciding when to update latest price stored per stock share

In future improvements, as more features are added to the application, more use cases involving caching data from the stock API with date/time of last pull can be added; or even more items as the table design will be generic enough to add any data needed without the need of using more than one table.

4.2.Design

Based on the use cases specified above the design will consist of a single table containing sparse data, using a generic partition key and sort key.

As our design will use generic partition and sort key, so that any kind of data can be stored, their column names will be as generic as possible:

Partition key = pk (String)

Sort Key = sk (String)

And to have the ability, on recovery, to find a user based on their email a secondary index on 'email' is used.

Name: email-index (auto generated)

partition key: email(string)

Remaining attribute columns of used as index's attributes

4.2.1. User Information

All information per specific user is stored in the same partition. See below of example and details.

pk	sk	latest_price	pocket	portfolio	quantity	stock_changes	stock_id	stock_name	latestTime	email	password_hash	user_name	user_type
user#Corey	info#Corey									corey.kbaum@gmail.com	pbkdf2:sha256:150000\$m9...	Corey	admin
user#Corey	datePull#								2020-12-11T20:02:58.197115				
user#Corey	stockID#AMD	91.575	-915.75	915.75	10	0	AMD	Advanced Micro Devices, Inc.					
user#Corey	stockID#AMZN	3111.83	-28006.47	28006.47	9	0	AMZN	Amazon.com, Inc.					
user#Corey	stockID#TD	56.3137	-844.7054999999999	844.7054999999999	15	0	TD	The Toronto-Dominion Bank					

Figure 27: DynamoDB Table - User Information

4.2.1.1. User Login Information

pk	sk	user_name	password_hash	email	user_type
user#username	info#username	username	hash_password	some.email@mail.com	['admin' or 'regular']

```
{
  "email": "corey.kbaum@gmail.com",
  "password_hash": "pbkdf2:sha256:150000$m9uzUn00$584d14c1ef9999df48f302cdeda772cd953a4807bc1ea966ca13dba735a0273f",
  "pk": "user#Corey",
  "sk": "info#Corey",
  "user_name": "Corey",
  "user_type": "admin"
}
```

Figure 28: DynamoDB Table - User Information Schema

Note: In future improvements the values for pk and sk of the following might improving partitioning/lookup/sorting/readability behaviour.

pk = user#username

sk = user#username#info

Or

pk = user#username

sk = info

4.2.1.2. User Stock Info

pk	sk	stock_id	stock_name	quantity	latest_price	pocket	portfolio
user#username	stockID#stockid	stockid	Company Name	Number of Shares	Share Price	Pocket Value	Portfolio Value

```
{
  "latest_price": "91.66",
  "pk": "user#bob",
  "pocket": "-939.6899999999999",
  "portfolio": "916.5999999999999",
  "quantity": 10,
  "sk": "stockID#AMD",
  "stock_changes": "-23.090000000000032",
  "stock_id": "AMD",
  "stock_name": "Advanced Micro Devices, Inc."
}
```

Figure 29: DynamoDB Table - User Stock Information Schema

Note: In future improvements the values for pk and sk of the following might improving partitioning/lookup/sorting/readability behaviour.

pk = user#username

sk = user#username#stockID#stock_symbol

4.2.1.3. User Stock Date Pulled

pk	sk	latestTime
user#username	datePull#	Date time of when all users stocks latest price attribute was last updated using the finance API

```
{
  "latestTime": "2020-12-11T20:02:58.197115",
  "pk": "user#Corey",
  "sk": "datePull#"
}
```

Figure 30: DynamoDB Table - User Stock Date Pulled Schema

Note: In future improvements the values for pk and sk of the following might improving partitioning/lookup/sorting/readability behaviour.

pk = user#username

sk = user#username#datePull

4.2.2. Gainers, Losers, Actives

The gainers, losers, and actives cached from the fiance API are stored on their own partitions.

Below figure illustrates how all the gainers, losers, and actives are stored in the DynamoDB table.

	pk	sk	changes	changesPercent	companyName	price	ticker	latestTime
<input type="checkbox"/>	gainers#	datePull#						2020-12-11T20:02:33.471270
<input type="checkbox"/>	losers#	datePull#						2020-12-11T00:53:25.430720
<input type="checkbox"/>	actives#	datePull#						2020-12-11T20:01:42.819795
<input type="checkbox"/>	gainers#	stockID#AESE	0.16	13.22	Allied Esports Entertainment Inc	1.37	AESE	
<input type="checkbox"/>	gainers#	stockID#ALPN	1.9401	17.53	Alpine Immune Sciences Inc	13.0101	ALPN	
<input type="checkbox"/>	losers#	stockID#AMJL	-0.3557	-22.86	Credit Suisse X-Links Monthly Pay 2xLeveraged Alerian MLP Index Excha...	1.2	AMJL	
<input type="checkbox"/>	losers#	stockID#AOB	-1.67	-19.26	AquaBounty Technologies Inc	7	AOB	
<input type="checkbox"/>	gainers#	stockID#ARDX	0.72	12.20	Ardelyx Inc	6.62	ARDX	
<input type="checkbox"/>	actives#	stockID#ARGX	9.9468	3.50	argenx SE	293.867	ARGX	
<input type="checkbox"/>	gainers#	stockID#ATTO	1.8555	17.41	Atento SA	12.5105	ATTO	
<input type="checkbox"/>	actives#	stockID#AZO	21.22	1.86	Autozone Inc	1159.07	AZO	
<input type="checkbox"/>	actives#	stockID#BKNG	-24.91	-1.18	Booking Holdings Inc	2079.96	BKNG	
<input type="checkbox"/>	actives#	stockID#BRK-A	-585.9	-0.17	Berkshire Hathaway Inc	339914	BRK-A	
<input type="checkbox"/>	actives#	stockID#CABO	11.095	0.52	Cable One Inc	2127.97	CABO	

Figure 31: DynamoDB Table - Gainers, Losers, Actives

4.2.2.1. Gainers Date Pulled

pk	sk	latestTime
gainers#	datePull#	Date time of when gaining stocks was last updated using the finance API

```
{
  "latestTime": "2020-12-11T20:02:33.471270",
  "pk": "gainers#",
  "sk": "datePull#"
}
```

Figure 32: DynamoDB Table - Gainers Date Pulled Schema

4.2.2.2. Gainers Stocks

pk	sk	ticker	companyName	price	changesPercentage	changes
gainers#	stockID#stockid	stockid	Company Name	Share Price	Share Price % Change	Share Price change

```
{
  "changes": "0.16",
  "changesPercentage": "13.22",
  "companyName": "Allied Esports Entertainment Inc",
  "pk": "gainers#",
  "price": "1.37",
  "sk": "stockID#AESE",
  "ticker": "AESE"
}
```

Figure 33: DynamoDB Table - Gainers Stock Information Schema

4.2.2.3. Losers Date Pulled

pk	sk	latestTime
losers#	datePull#	Date time of when losing stocks was last updated using the finance API

```
{
  "latestTime": "2020-12-11T00:53:25.430720",
  "pk": "losers#",
  "sk": "datePull#"
}
```

Figure 34: DynamoDB Table - Losers Date Pulled Schema

4.2.2.4. Losers Stocks

pk	sk	ticker	companyName	price	changesPercentage	changes
losers#	stockID#stockid	stockid	Company Name	Share Price	Share Price % Change	Share Price change

```
{
  "changes": "-0.3557",
  "changesPercentage": "-22.86",
  "companyName": "Credit Suisse X-Links Monthly Pay 2xLeveraged Alerian MLP Index Exchange Traded Notes due May 16 2036",
  "pk": "losers#",
  "price": "1.2",
  "sk": "stockID#AMJL",
  "ticker": "AMJL"
}
```

Figure 35: DynamoDB Table - Losers Stock Information Schema

4.2.2.5. Actives Date Pulled

pk	sk	latestTime
actives#	datePull#	Date time of when active stocks was last updated using the finance API

```
{
  "latestTime": "2020-12-11T20:01:42.819795",
  "pk": "actives#",
  "sk": "datePull#"
}
```

Figure 36: DynamoDB Table - Actives Date Pulled Schema

4.2.2.6. Actives Stocks

pk	sk	ticker	companyName	price	changesPercentage	changes
actives#	stockID#stockid	stockid	Company Name	Share Price	Share Price % Change	Share Price change

```
{
  "changes": "9.9468",
  "changesPercentage": "3.50",
  "companyName": "argenx SE",
  "pk": "actives#",
  "price": "293.867",
  "sk": "stockID#ARGX",
  "ticker": "ARGX"
}
```

Figure 37: DynamoDB Table – Actives Stocks Information Schema

5. How to run the app

We are using the AWS educate account for running our Stock Portfolio Application (lambda's, DynamoDB, S3).

The educate account we are using is: 'mohammed.baquir@mail.utoronto.ca'

Stock Portfolio Lambda = ece1779-a3-dev

5.1. Amazon S3

A single amazon S3 will be used as the lambda deployment location used by Zappa. The S3 Bucket region is us-east-1.

S3 Bucket = ece1779-a2-s3

6. AWS Cost

We are using the financial modeling prep free tier API for the prototype. In order to move to production and handle many users we would move to their Teams tier, which would give us unlimited requests. Currently since we are on the free tier where we can only make 250 requests per day, we have added caching to our DB where we show data 30mins old. Once we move to the paid tier, we can show latest stock data.

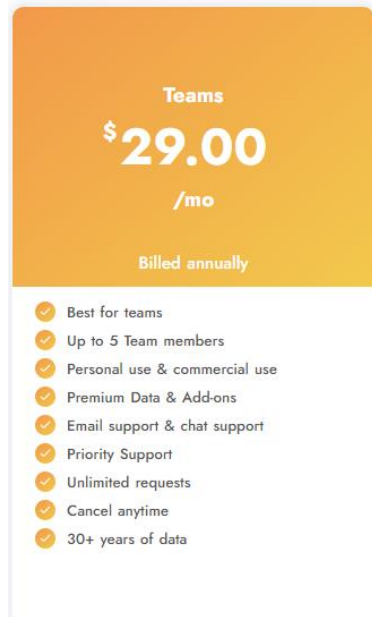


Figure 38: Finance API Paid Tier

These are the list of amazon services used by the application to take into account to do cost analysis.

- AWS Lambda
- Amazon Cloudwatch
- Amazon Simple Storage Service (S3)
- Amazon DynamoDB
- Amazon API Gateway

One of the busiest pages we have in the app is the Stock Details page. As we can see from the network tab below, to view this 1 page we make 13 http requests and transfer an average of 250 Kb.

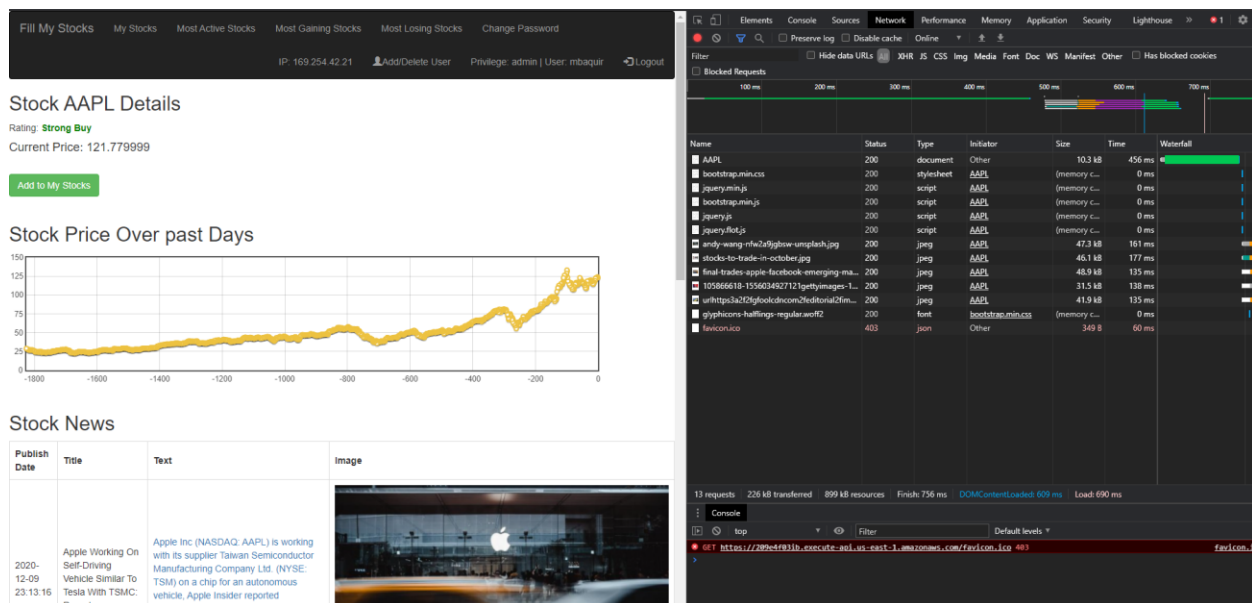


Figure 39: Application Number of HTTP Requests

That is 1 user just viewing 1 html page. Let's assume on average the user will look at 30 different stock details in an hour. That would mean that around 400 requests would be made. Let us assume an average user will make **1000** http requests per day using our app.

The cost for Amazon S3 and Amazon Cloudwatch should stay fairly consistent and not depend on the users. Currently we are only using S3 to deploy the app through zappa. We are not storing any user data currently in S3. Future features might require storing financial stock data to s3. But for now let's assume S3 cost will stay fairly consistent. We are currently not using any custom metrics. Let's assume we want detailed monitoring to be able to do analysis on our application.

This is what the cost for S3 and cloudwatch look like:

Services (5)		
<div> <div>Amazon CloudWatch</div> <div>Region: US East (Ohio)</div> <div>Edit</div> <div>Action ▾</div> </div>		
Monthly:		2.00 USD
<div> <div>Amazon Simple Storage Service (S3)</div> <div>Region: US East (Ohio)</div> <div>Edit</div> <div>Action ▾</div> </div>		
S3 Standard storage (10 GB per month)		Monthly: 0.23 USD

Figure 40: Cloudwatch Cost

S3 will cost us 0.23 USD per month and Cloudwatch will be 2 USD per month.

The only 3 services where the requests change with the number of users are Lambda, API Gateway and DynamoDB. The costliest service out of the 3 is the DynamoDB.

To measure the table size required accurately, we just had 1 user entry in the table. Looking at the storage size we can see that it is only 258 bytes. On average each entry in the DB takes approximately

150 bytes. The maximum number of stocks we can get from the API is 15000. Assuming a user adds 15000 stocks to its portfolio, we would still not exceed 1GB database size.

Table details

Table name	ece1779_a3
Primary partition key	pk (String)
Primary sort key	sk (String)
Point-in-time recovery	DISABLED Enable
Encryption Type	DEFAULT Manage Encryption
KMS Master Key ARN	Not Applicable
Encryption Status	
CloudWatch Contributor Insights	DISABLED Manage Contributor Insights NEW
Time to live attribute	DISABLED Manage TTL
Table status	Active
Creation date	December 6, 2020 at 8:38:47 PM UTC-5
Read/write capacity mode	Provisioned
Last change to on-demand mode	-
Provisioned read capacity units	5 (Auto Scaling Error)
Provisioned write capacity units	5 (Auto Scaling Error)
Last decrease time	-
Last increase time	-
Storage size (in bytes)	258.00 bytes
Item count	2 Manage live count
Region	US East (N. Virginia)
Amazon Resource Name (ARN)	arn:aws:dynamodb:us-east-1:633720668941:table/ece1779_a3

Figure 41: DynamoDB Table Details

6.1. Cost Model for 10 users

For only 10 users, we would need 1 GB DynamoDB size. (smallest size)

Region	Description	Service	Upfront	Monthly	First 12 months total	Currency	Configuration summary
US East (Ohio)	fill_my_stocks_gateway	Amazon API Gateway	0	0.0304	0.36 USD		HTTP API requests units (thousands), Average size of each request (256 KB), REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), Average message size (32 KB), Requests (1 per day)
US East (Ohio)	fill_my_stocks_DynamoDB	DynamoDB provisioned capacity	180	11.41	316.92 USD		Average item size (all attributes) (500 Byte), Write reserved capacity term (1 year), Read reserved capacity term (1 year), Data storage size (1 GB)
US East (Ohio)	fill_my_stocks_lambda	AWS Lambda	0	0	0 USD		Number of requests (30000)
US East (Ohio)	fill_my_stocks_s3	S3 Standard	0	0.23	2.76 USD		S3 Standard storage (10 GB per month)
US East (Ohio)	fill_my_stocks_s3	Data Transfer	0	0	0 USD		DT Inbound: Not selected (0 TB per month)
US East (Ohio)	fill_my_stocks_cloudwatch	Amazon CloudWatch	0	2.0046	24.06 USD		

Figure 42: Cost Breakdown per 10 Users

My Estimate Info	Add service	Add support	Add group	Clear estimate	Action ▼	Save and share
First 12 months total			Total upfront		Total monthly	
344.10 USD			180.00 USD		13.68 USD	

Figure 43: Cost Estimator per 10 users

6.2. Cost Model for 1000 users

For only 1000 users, we would still need 1 GB DynamoDB size. So the cost for DB,S3 and cloudwatch stays fairly consistent.

Region	Description	Service	Upfront	Monthly	First 12 months total	Currency	Configuration summary
US East (Ohio)	fill_my_stocks_gateway	Amazon API Gateway	0	0.3042	3.65 USD		HTTP API requests units (thousands), Average size of each request (256 KB), REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), Average message size (32 KB), Requests (10 per day)
US East (Ohio)	fill_my_stocks_dynamodb	DynamoDB provisioned capacity	180	11.41	316.92 USD		Average item size (all attributes) (500 Byte), Write reserved capacity term (1 year), Read reserved capacity term (1 year), Data storage size (1 GB)
US East (Ohio)	fill_my_stocks_lambda	AWS Lambda	0	0	0 USD		Number of requests (3000000)
US East (Ohio)	fill_my_stocks_s3	S3 Standard	0	0.23	2.76 USD		S3 Standard storage (10 GB per month)
US East (Ohio)	fill_my_stocks_s3	Data Transfer	0	0	0 USD		DT Inbound: Not selected (0 TB per month)
US East (Ohio)	fill_my_stocks_cloudwatch	Amazon CloudWatch	0	2.0046	24.06 USD		

Figure 44: Cost Breakdown per 1000 users

My Estimate Info	Add service	Add support	Add group	Clear estimate	Action ▼	Save and share
First 12 months total	Total upfront		Total monthly			
347.39 USD	180.00 USD		13.95 USD			

Figure 45: Cost Estimator per 1000 users

6.3. Cost Model for 1,000,000 users

For only 1000,000 users, we would need at max 2 GB DynamoDB size.

Region	Description	Service	Upfront	Monthly	First 12 months total	Currency	Configuration summary
US East (Ohio)	fill_my_stocks_gateway	Amazon API Gateway	0	3.0417	36.5 USD		HTTP API requests units (thousands), Average size of each request (256 KB), REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), Average message size (32 KB), Requests (100 per day)
US East (Ohio)	fill_my_stocks_dynamodb	DynamoDB provisioned capacity	180	11.66	319.92 USD		Average item size (all attributes) (500 Byte), Write reserved capacity term (1 year), Read reserved capacity term (1 year), Data storage size (2 GB)
US East (Ohio)	fill_my_stocks_lambda	AWS Lambda	0	0.4	4.8 USD		Number of requests (3000000)
US East (Ohio)	fill_my_stocks_s3	S3 Standard	0	0.23	2.76 USD		S3 Standard storage (10 GB per month)
US East (Ohio)	fill_my_stocks_s3	Data Transfer	0	0	0 USD		DT Inbound: Not selected (0 TB per month)
US East (Ohio)	fill_my_stocks_cloudwatch	Amazon CloudWatch	0	2.0046	24.06 USD		

Figure 46: Cost Breakdown per 1 million users

My Estimate Info	Add service	Add support	Add group	Clear estimate	Action ▼	Save and share
First 12 months total	Total upfront		Total monthly			
388.04 USD	180.00 USD		17.34 USD			

Figure 47: Cost Estimator per 1 million users

Future Improvements

- Sometimes had issues with Gmail security blocking authentication. Maybe for future improvements try using “Amazon Simple Email Service” (Though will need a non-student account)
- Add background lambda to monitor stock prices and notify users (with email) when their stocks drop below a minimum threshold (would require non-student account)
- Try to connect application to stock market to allow users to actually buy or sell stocks

Group Member Efforts

Group members worked together very evenly on this project with each member working on all aspects in some manner.

Corey Kirschbaum

- Documentation
- Generating Project Ideas
- Database table use case and design
- Investigate use of zappa for deploy or zip
- Algorithm of calculating users portfolio gains/loses

- Found Stock API to use
- Worked on Stock Portfolio App

Mohammed Baquir

- Documentation
- Generating Project Ideas
- Database table use case and design
- Investigate use of zappa for deploy or zip
- Algorithm of calculating users portfolio gains/loses
- Found Stock API to use
- Worked on Stock Portfolio App
- Worked on background lambda function

References:

- [1] <https://www.trek10.com/blog/dynamodb-single-table-relational-modeling>
- [2] <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-modeling-nosql-B.html>