MATES Computer Science

Senior Capstone Project Bi-Weekly Progress Report

Project Title	MorningBread
Team Members	Max Bradshaw, Jake Dorick
Dates Covered by Report	April 8th, 2024 - April 19th, 2024
Link to Github	https://github.com/maxbshaw17/MorningBread

1. **Summary of Project** (Provide a one paragraph summary of your project. You can largely copy/paste this from one progress report to the next, unless there are significant changes.)

To provide a personalized morning financial report to our users by utilizing a web scraping tool for financial website articles to find the general topics of the articles. In doing so, we are creating and designing a website to give the public a general grasp of financial news for the day, over the course of the day with hourly, highly summarized news headlines/stock tickers. An account system will allow users to follow certain companies and manage tickers.

2. **Summary of Progress this Period** (Provide a high-level, one-paragraph overview of what was accomplished this progress period collectively by the team.)

Jake

Dynamically added articles were the main start and finish points of this period. Luckily, after working on it and struggling for the entire period, I finally got it working with the example table Max provided. With this, once the correct table is fully made and regularly updated, all I have to do is change a few words in the API, which will display and update on the website. The entire process took a combination of MySQL Workbench, Python Flask API, and a lot of JavaScript to finally be able to implement. Much of the code I wrote during this period will be copied and pasted for the articles' summaries and links, and the tickers.

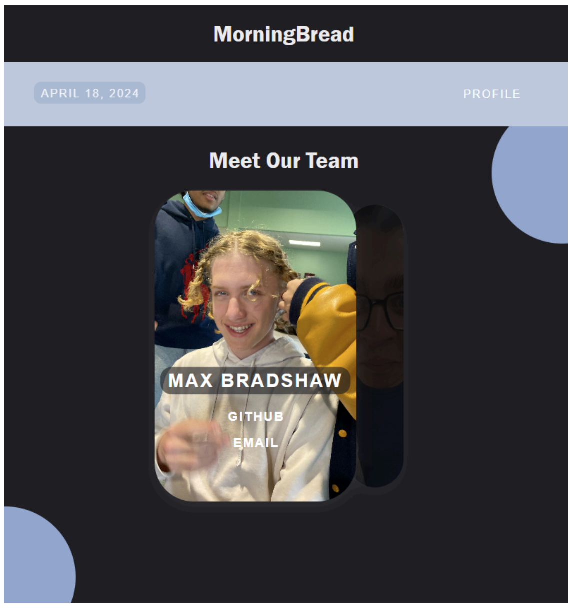
Max

I finished the core functionality with chatGPT summarization, but the code was quick and dirty. I have since been working on refactoring what I previously wrote to condense and generalize the functions. I also am switching everything to use pandas dataframes as input and output. I also created a class for the database connection object.

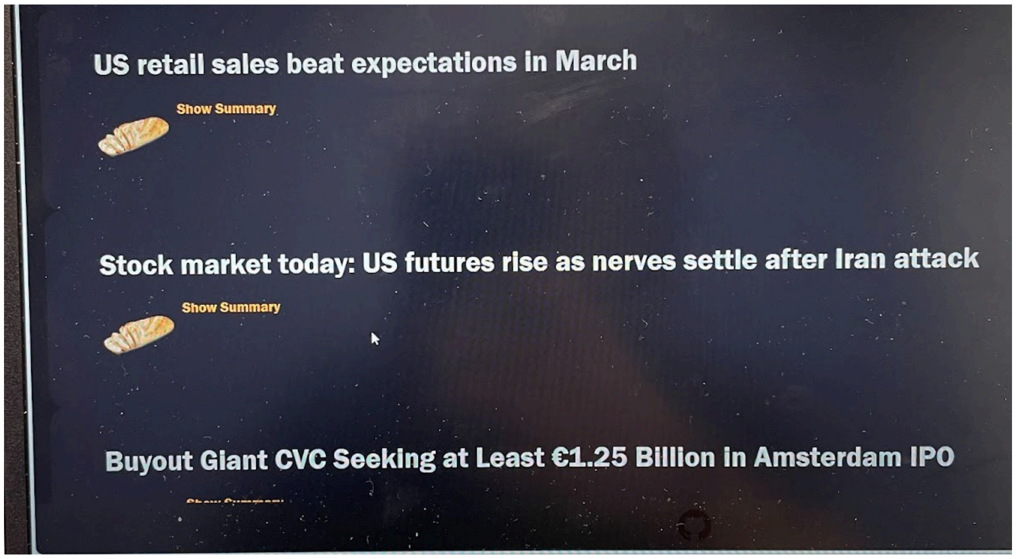
3. **Detailed Progress this Period, separated by Team Member** (Provide detailed information on the progress that you made in the reporting weeks. Include screenshots of code, your game or website, etc. Each team member should have a separate subsection covering their accomplishments. Not including screenshots, this section should be 1-2 pages.)

Jake

- Before I show the main and most important progress I made during this period, I also want to show what I did for fun over Spring Break
- I created an accordion-style About Us page for the website located on the Contact Us page
- I stole most of the code from this Instagram post I saw but modified it for our website's aesthetics and flair

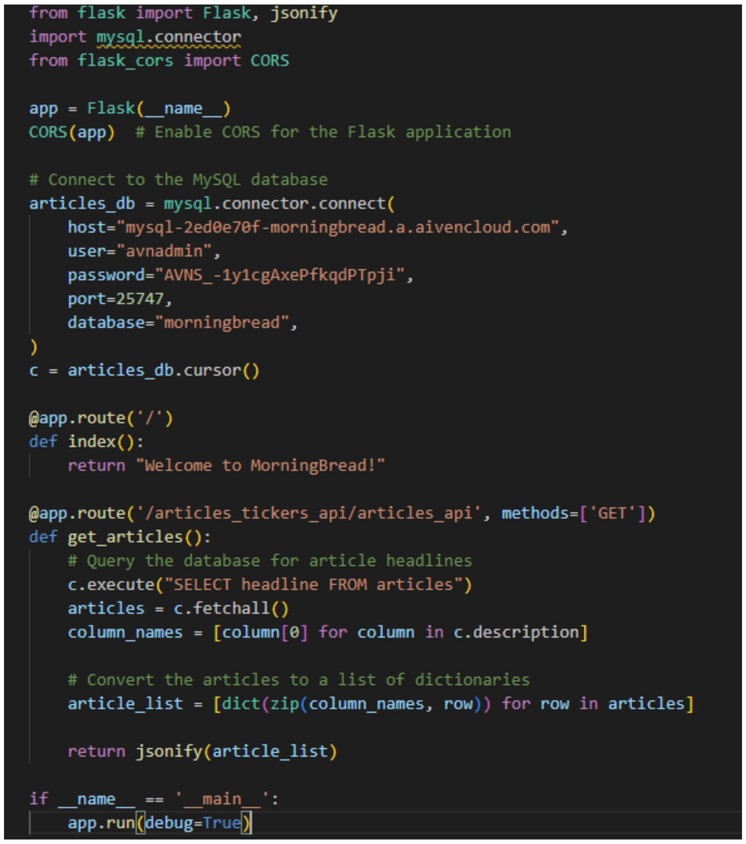


- More importantly, I spent the majority of my class time focusing on implementing the dynamically added tickers from the MySQL Workbench tables


US retail sales beat expectations in March
Show Summary
Stock market today: US futures rise as nerves settle after Iran attack
Show Summary
Buyout Giant CVC Seeking at Least €1.25 Billion in Amsterdam IPO

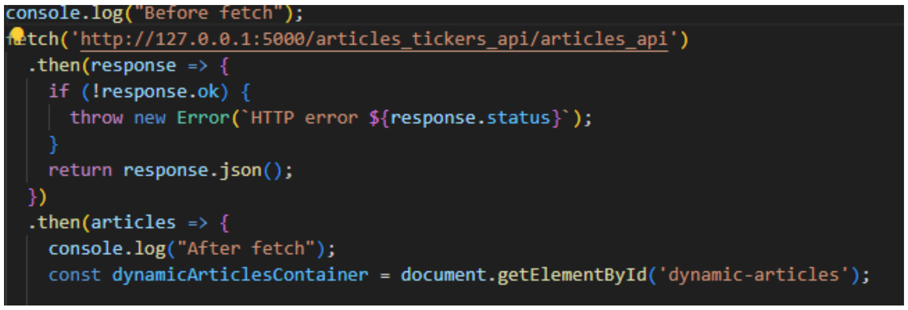
(almost) Final Product ^

- To accomplish this, I built off of my previous Flask API script to grab the correct column of the correct table in the MySQL Workbench as well as establish a port to house this information
- The JavaScript to tie the API into the website was a lot beefier than the Python API
- The JavaScript included:
    - Calling the API's port and connecting it to the website's port (and accounting for response errors)
    - Creating two inline HTML sections to correlate to the CSS made of the example article containers
    - Appending the articles to the section of the website where they below
    - Adding event listeners to the "Show/Hide Summary" buttons (making sure to allow the user to only click once to show/hide the summary)



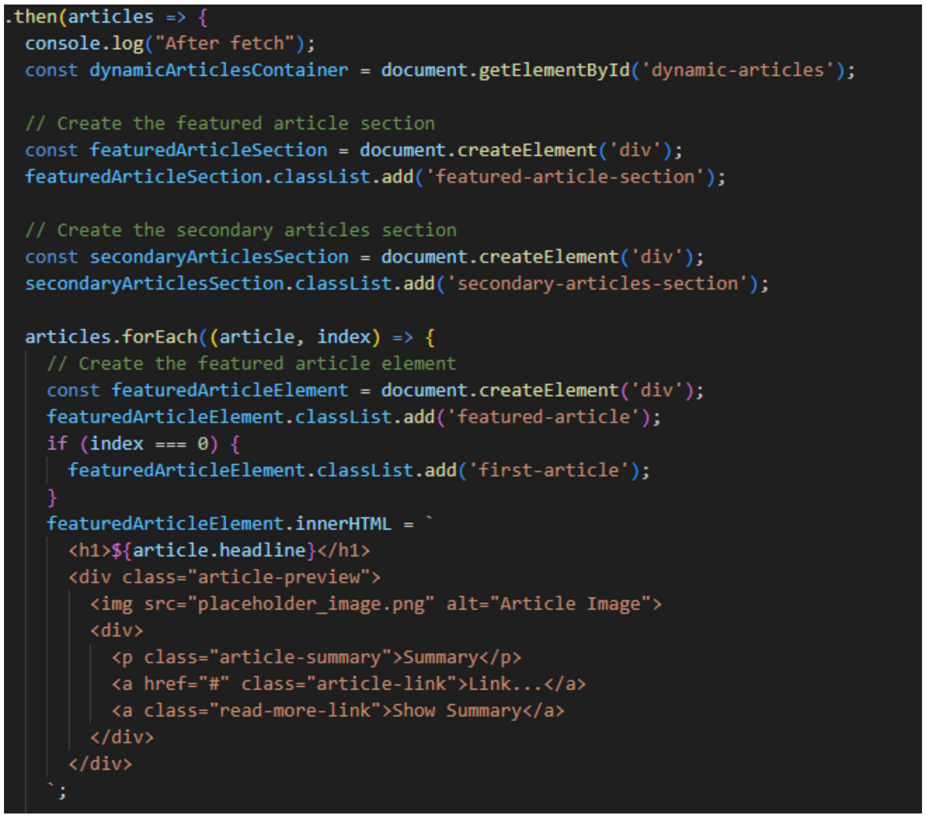
```python
from flask import Flask, jsonify
import mysql.connector
from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # Enable CORS for the Flask application

# Connect to the MySQL database
articles_db = mysql.connector.connect(
    host="mysql-2ed0e70f-morningbread.a.aivencloud.com",
    user="avnadmin",
    password="AVNS_-1y1cgAxePfkqdPTpji",
    port=25747,
    database="morningbread",
)
c = articles_db.cursor()

@app.route('/')
def index():
    return "Welcome to MorningBread!"

@app.route('/articles_tickers_api/articles_api', methods=['GET'])
def get_articles():
    # Query the database for article headlines
    c.execute("SELECT headline FROM articles")
    articles = c.fetchall()
    column_names = [column[0] for column in c.description]

    # Convert the articles to a list of dictionaries
    article_list = [dict(zip(column_names, row)) for row in articles]

    return jsonify(article_list)

if __name__ == '__main__':
    app.run(debug=True)
```

- With all of that included, I achieved the final product shown above
- I will include screenshots of the JavaScript code to show what I am referring to in the four sub-bullets I put



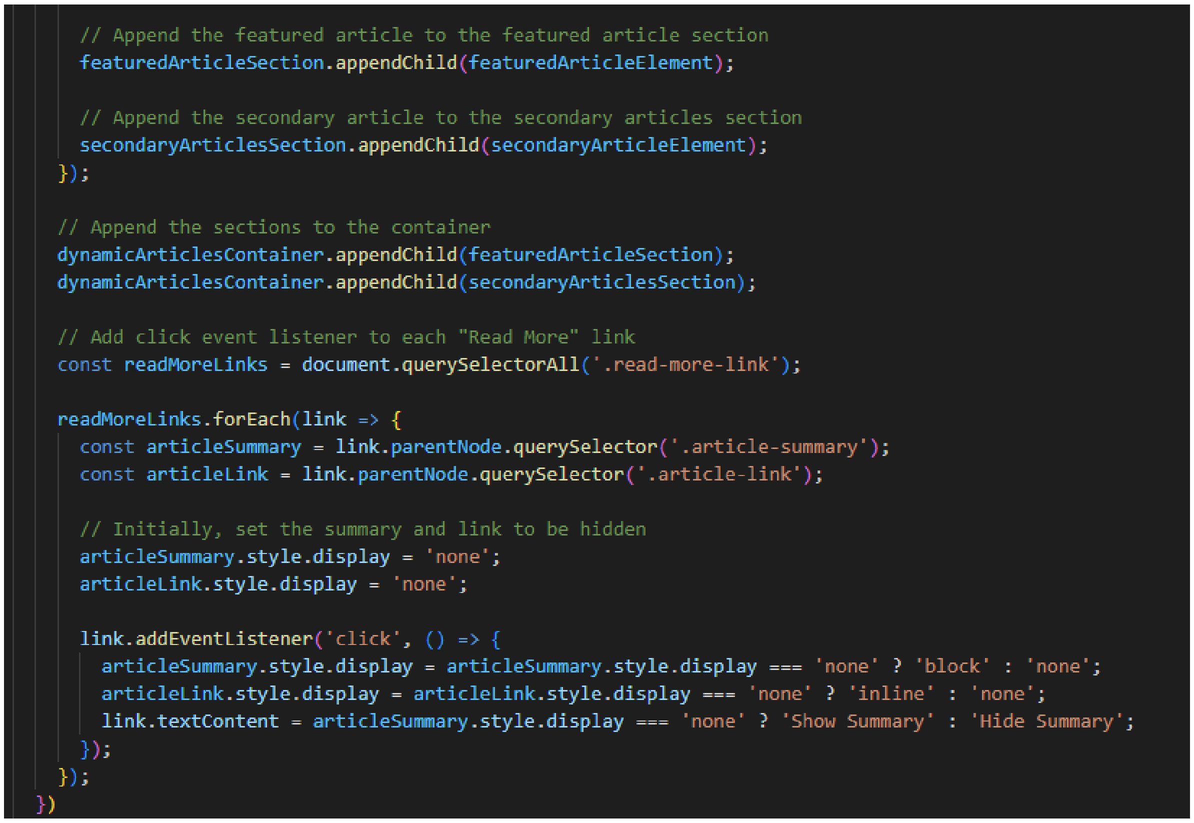
```
console.log("Before fetch");
fetch('http://127.0.0.1:5000/articles_tickers_api/articles_api')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error ${response.status}`);
    }
    return response.json();
  })
  .then(articles => {
    console.log("After fetch");
    const dynamicArticlesContainer = document.getElementById('dynamic-articles');
```

Calling the API from its port (local port 5000)



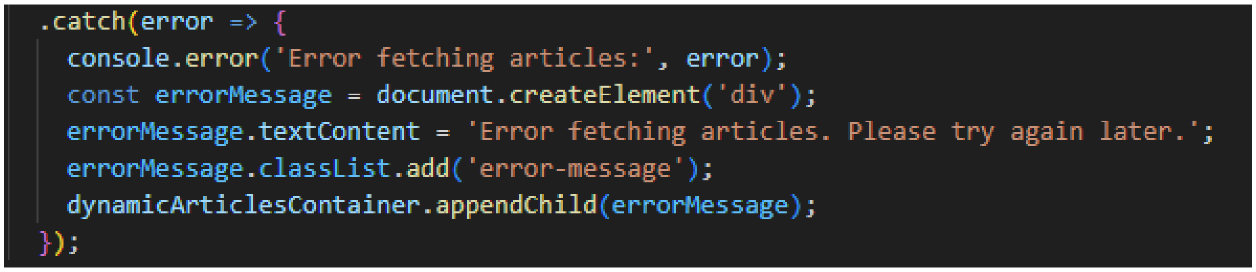
```
.then(articles => {
  console.log("After fetch");
  const dynamicArticlesContainer = document.getElementById('dynamic-articles');

  // Create the featured article section
  const featuredArticleSection = document.createElement('div');
  featuredArticleSection.classList.add('featured-article-section');

  // Create the secondary articles section
  const secondaryArticlesSection = document.createElement('div');
  secondaryArticlesSection.classList.add('secondary-articles-section');

  articles.forEach((article, index) => {
    // Create the featured article element
    const featuredArticleElement = document.createElement('div');
    featuredArticleElement.classList.add('featured-article');
    if (index === 0) {
      featuredArticleElement.classList.add('first-article');
    }
    featuredArticleElement.innerHTML = `
      <h1>${article.headline}</h1>
      <div class="article-preview">
        <img src="placeholder_image.png" alt="Article Image">
        <div>
          <p class="article-summary">Summary</p>
          <a href="#" class="article-link">Link...</a>
          <a class="read-more-link">Show Summary</a>
        </div>
      </div>
    `;
```

Matching the HTML of the articles to the CSS previously made



```
      // Append the featured article to the featured article section
      featuredArticleSection.appendChild(featuredArticleElement);

      // Append the secondary article to the secondary articles section
      secondaryArticlesSection.appendChild(secondaryArticleElement);
    });

    // Append the sections to the container
    dynamicArticlesContainer.appendChild(featuredArticleSection);
    dynamicArticlesContainer.appendChild(secondaryArticlesSection);

    // Add click event listener to each "Read More" link
    const readMoreLinks = document.querySelectorAll('.read-more-link');

    readMoreLinks.forEach(link => {
      const articleSummary = link.parentNode.querySelector('.article-summary');
      const articleLink = link.parentNode.querySelector('.article-link');

      // Initially, set the summary and link to be hidden
      articleSummary.style.display = 'none';
      articleLink.style.display = 'none';

      link.addEventListener('click', () => {
        articleSummary.style.display = articleSummary.style.display === 'none' ? 'block' : 'none';
        articleLink.style.display = articleLink.style.display === 'none' ? 'inline' : 'none';
        link.textContent = articleSummary.style.display === 'none' ? 'Show Summary' : 'Hide Summary';
      });
    });
  })
```

Appending the articles and the Event Listener

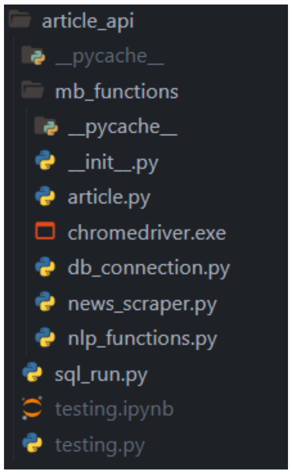
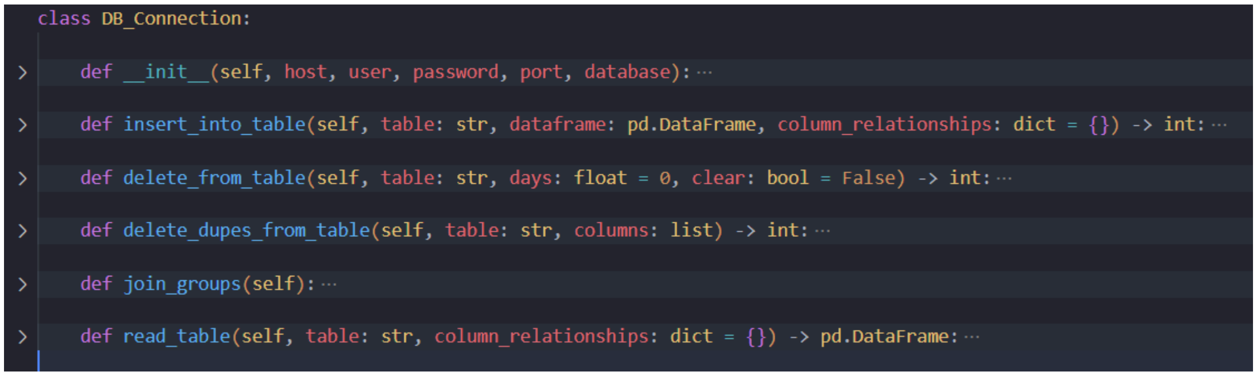


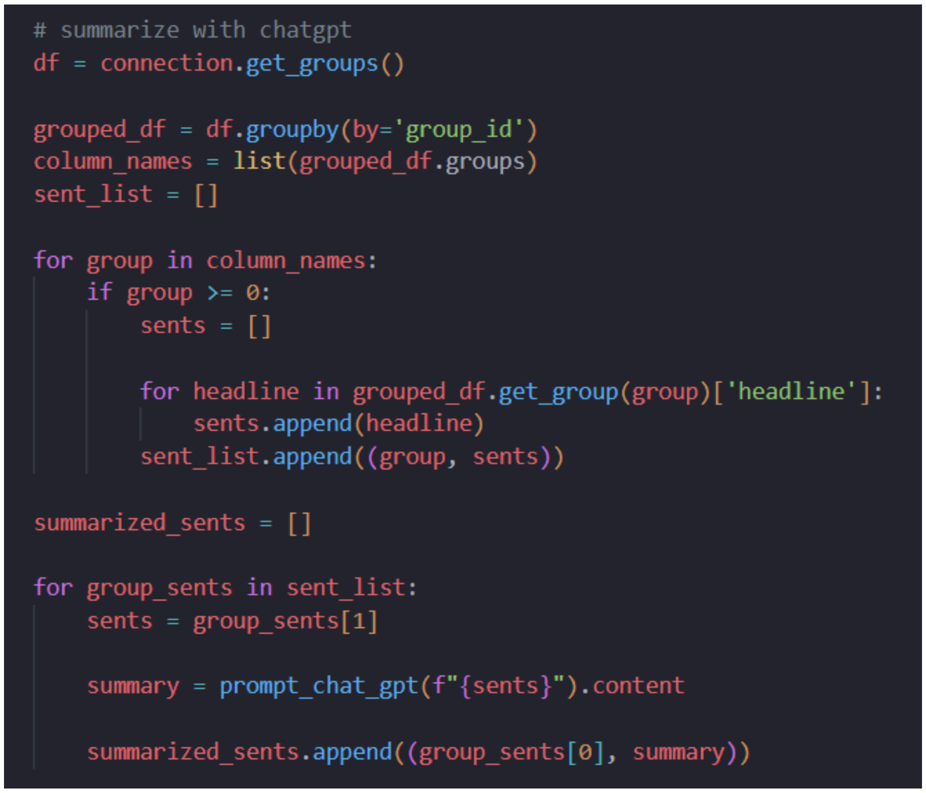
```
  .catch(error => {
    console.error('Error fetching articles:', error);
    const errorMessage = document.createElement('div');
    errorMessage.textContent = 'Error fetching articles. Please try again later.';
    errorMessage.classList.add('error-message');
    dynamicArticlesContainer.appendChild(errorMessage);
  });
```

Catch error

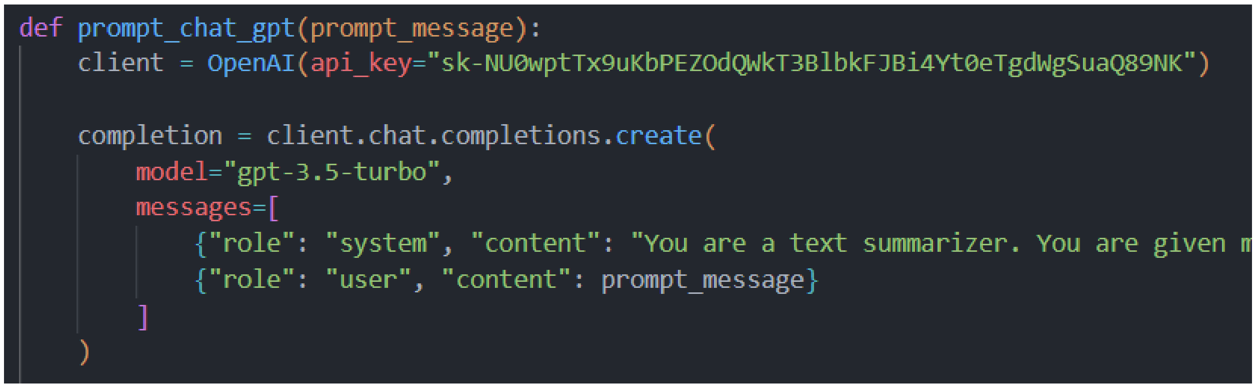
Max

- I changed the file format again, this should be the final iteration

- 
```
article_api
  __pycache__
  mb_functions
    __pycache__
    __init__.py
    article.py
    chromedriver.exe
    db_connection.py
    news_scraper.py
    nlp_functions.py
  sql_run.py
  testing.ipynb
  testing.py
```
- Created a class for the database connection object to simplify the SQL functions
- 
```python
class DB_Connection:

    def __init__(self, host, user, password, port, database): ...

    def insert_into_table(self, table: str, dataframe: pd.DataFrame, column_relationships: dict = {}) -> int: ...

    def delete_from_table(self, table: str, days: float = 0, clear: bool = False) -> int: ...

    def delete_dupes_from_table(self, table: str, columns: list) -> int: ...

    def join_groups(self): ...

    def read_table(self, table: str, column_relationships: dict = {}) -> pd.DataFrame: ...
```
- Added chatGPT summarization. This is very badly written but is just a proof of concept (don't look at my API key)

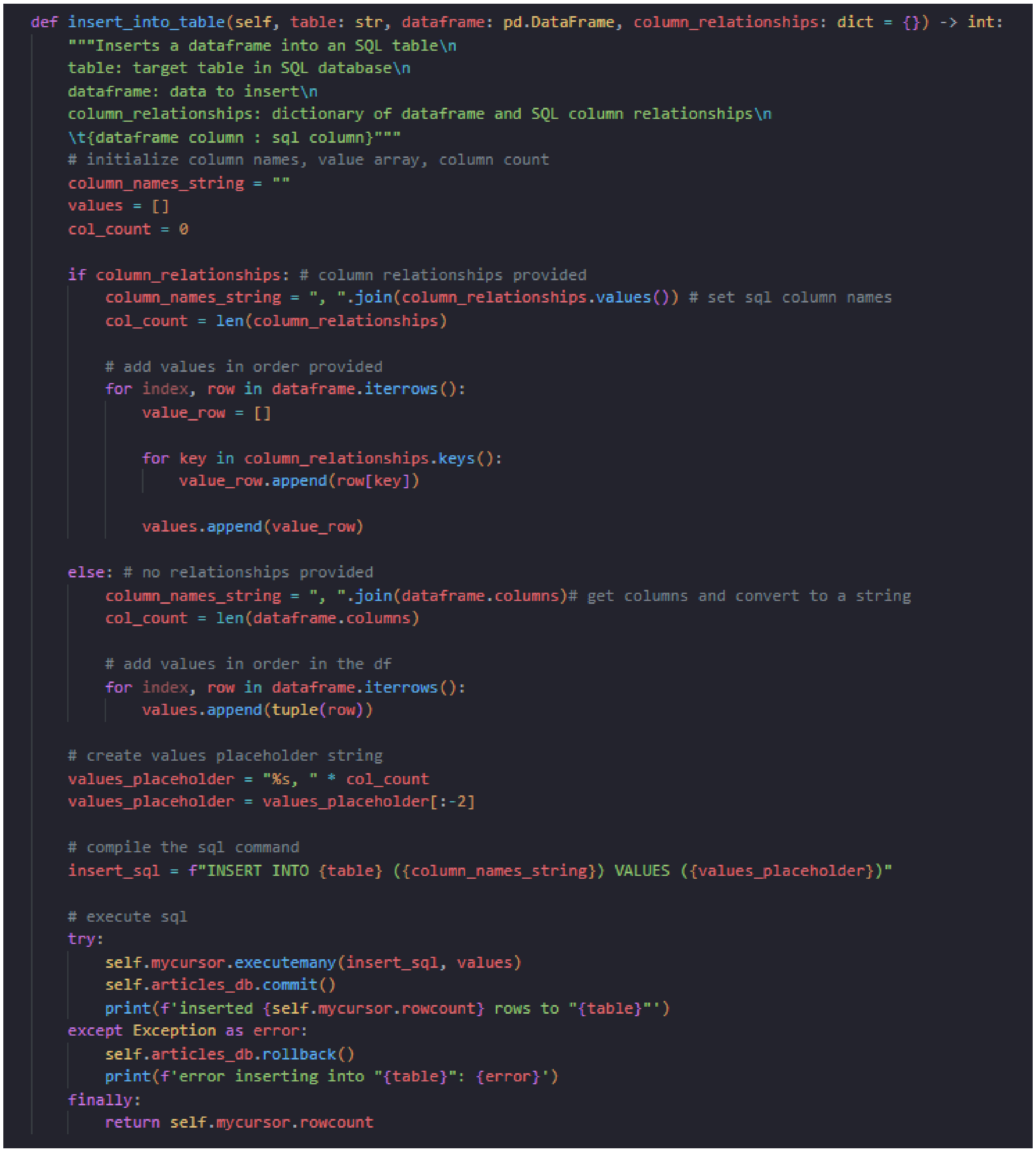
- 

```python
# summarize with chatgpt
df = connection.get_groups()

grouped_df = df.groupby(by='group_id')
column_names = list(grouped_df.groups)
sent_list = []

for group in column_names:
    if group >= 0:
        sents = []

        for headline in grouped_df.get_group(group)['headline']:
            sents.append(headline)
        sent_list.append((group, sents))

summarized_sents = []

for group_sents in sent_list:
    sents = group_sents[1]

    summary = prompt_chat_gpt(f"{sents}").content

    summarized_sents.append((group_sents[0], summary))
```

- 

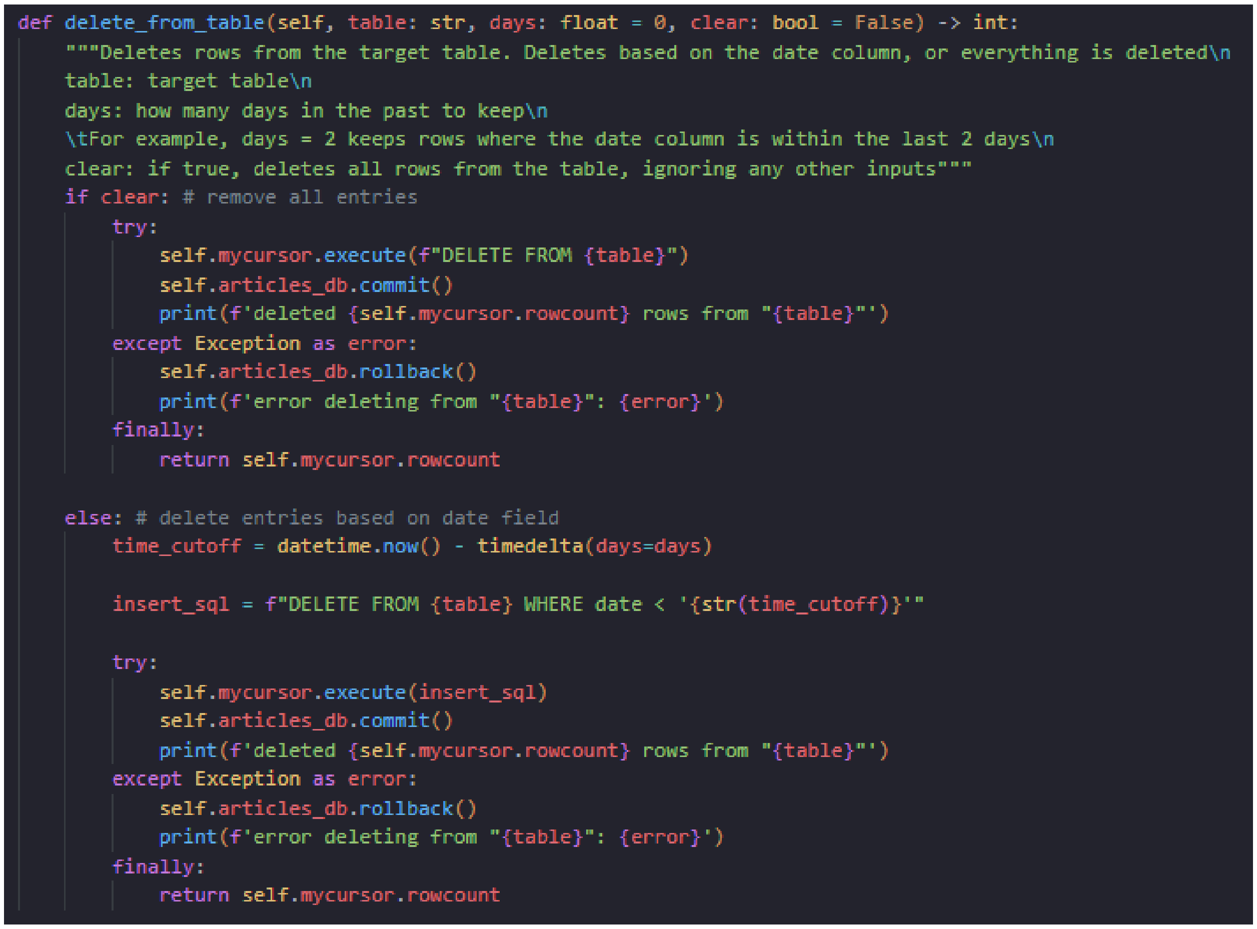
```python
def prompt_chat_gpt(prompt_message):
    client = OpenAI(api_key="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")

    completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a text summarizer. You are given m
            {"role": "user", "content": prompt_message}
        ]
    )
```

- Wrote a function to insert any dataframe into and SQL table, allowing for the user to specify the column relationships



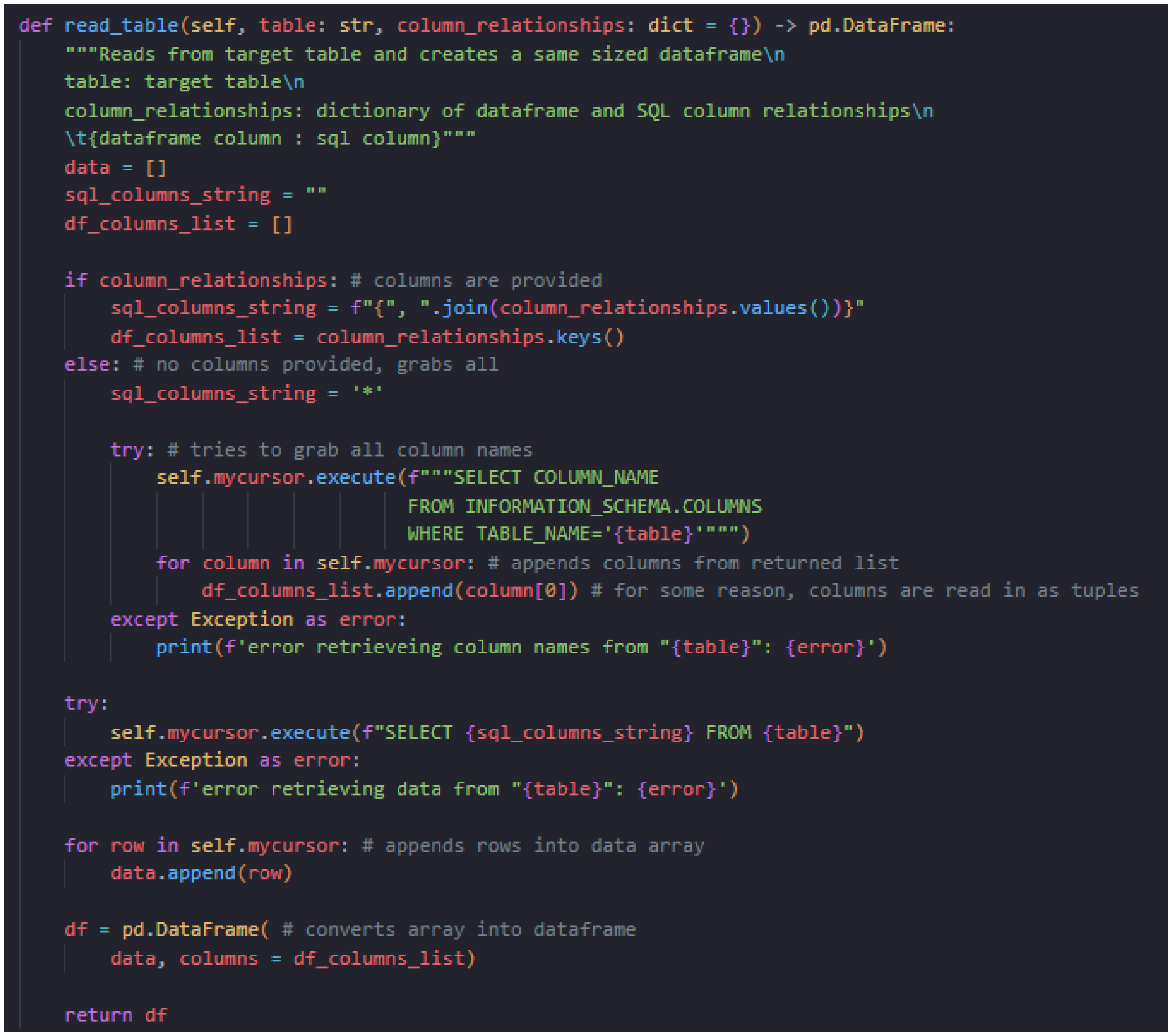
```python
def insert_into_table(self, table: str, dataframe: pd.DataFrame, column_relationships: dict = {}) -> int:
    """Inserts a dataframe into an SQL table\n
    table: target table in SQL database\n
    dataframe: data to insert\n
    column_relationships: dictionary of dataframe and SQL column relationships\n
    \t{dataframe column : sql column}"""
    # initialize column names, value array, column count
    column_names_string = ""
    values = []
    col_count = 0

    if column_relationships: # column relationships provided
        column_names_string = ", ".join(column_relationships.values()) # set sql column names
        col_count = len(column_relationships)

        # add values in order provided
        for index, row in dataframe.iterrows():
            value_row = []

            for key in column_relationships.keys():
                value_row.append(row[key])

            values.append(value_row)

    else: # no relationships provided
        column_names_string = ", ".join(dataframe.columns)# get columns and convert to a string
        col_count = len(dataframe.columns)

        # add values in order in the df
        for index, row in dataframe.iterrows():
            values.append(tuple(row))

    # create values placeholder string
    values_placeholder = "%s, " * col_count
    values_placeholder = values_placeholder[:-2]

    # compile the sql command
    insert_sql = f"INSERT INTO {table} ({column_names_string}) VALUES ({values_placeholder})"

    # execute sql
    try:
        self.mycursor.executemany(insert_sql, values)
        self.articles_db.commit()
        print(f'inserted {self.mycursor.rowcount} rows to "{table}"')
    except Exception as error:
        self.articles_db.rollback()
        print(f'error inserting into "{table}": {error}')
    finally:
        return self.mycursor.rowcount
```

-
- Wrote a function to delete rows from a table, either by specifying a date range or erasing everything



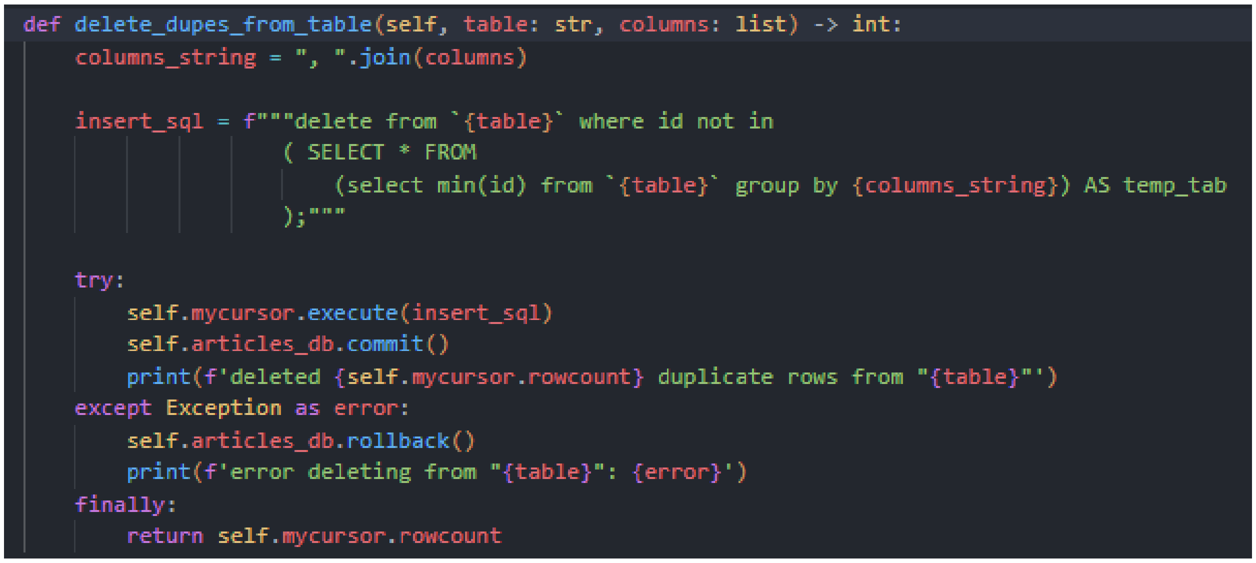
```python
def delete_from_table(self, table: str, days: float = 0, clear: bool = False) -> int:
    """Deletes rows from the target table. Deletes based on the date column, or everything is deleted\n
    table: target table\n
    days: how many days in the past to keep\n
    \tFor example, days = 2 keeps rows where the date column is within the last 2 days\n
    clear: if true, deletes all rows from the table, ignoring any other inputs"""
    if clear: # remove all entries
        try:
            self.mycursor.execute(f"DELETE FROM {table}")
            self.articles_db.commit()
            print(f'deleted {self.mycursor.rowcount} rows from "{table}"')
        except Exception as error:
            self.articles_db.rollback()
            print(f'error deleting from "{table}": {error}')
        finally:
            return self.mycursor.rowcount

    else: # delete entries based on date field
        time_cutoff = datetime.now() - timedelta(days=days)

        insert_sql = f"DELETE FROM {table} WHERE date < '{str(time_cutoff)}'"

        try:
            self.mycursor.execute(insert_sql)
            self.articles_db.commit()
            print(f'deleted {self.mycursor.rowcount} rows from "{table}"')
        except Exception as error:
            self.articles_db.rollback()
            print(f'error deleting from "{table}": {error}')
        finally:
            return self.mycursor.rowcount
```

-
- Wrote a function to read any SQL table and create a dataframe from the data, allowing the user to specify column relationships



```python
def read_table(self, table: str, column_relationships: dict = {}) -> pd.DataFrame:
    """Reads from target table and creates a same sized dataframe\n
    table: target table\n
    column_relationships: dictionary of dataframe and SQL column relationships\n
    \t{dataframe column : sql column}"""
    data = []
    sql_columns_string = ""
    df_columns_list = []

    if column_relationships: # columns are provided
        sql_columns_string = f"{", ".join(column_relationships.values())}"
        df_columns_list = column_relationships.keys()
    else: # no columns provided, grabs all
        sql_columns_string = '*'

        try: # tries to grab all column names
            self.mycursor.execute(f"""SELECT COLUMN_NAME
                                  FROM INFORMATION_SCHEMA.COLUMNS
                                  WHERE TABLE_NAME='{table}'""")
            for column in self.mycursor: # appends columns from returned list
                df_columns_list.append(column[0]) # for some reason, columns are read in as tuples
        except Exception as error:
            print(f'error retrieveing column names from "{table}": {error}')

    try:
        self.mycursor.execute(f"SELECT {sql_columns_string} FROM {table}")
    except Exception as error:
        print(f'error retrieving data from "{table}": {error}')

    for row in self.mycursor: # appends rows into data array
        data.append(row)

    df = pd.DataFrame( # converts array into dataframe
        data, columns = df_columns_list)

    return df
```

-
- I'm currently working on a duplicate deletion function, where the user can specify which rows to check for uniqueness

- 

```python
def delete_dupes_from_table(self, table: str, columns: list) -> int:
    columns_string = ", ".join(columns)

    insert_sql = f"""delete from `{table}` where id not in
                    ( SELECT * FROM
                        (select min(id) from `{table}` group by {columns_string}) AS temp_tab
                    );"""

    try:
        self.mycursor.execute(insert_sql)
        self.articles_db.commit()
        print(f'deleted {self.mycursor.rowcount} duplicate rows from "{table}"')
    except Exception as error:
        self.articles_db.rollback()
        print(f'error deleting from "{table}": {error}')
    finally:
        return self.mycursor.rowcount
```

4. **Difficulties Encountered this Progress Period** (Provide detailed information on the difficulties and issues that you encountered in the reporting weeks. Discuss mitigation strategies for how you got around or plan to get around these issues.)

Jake

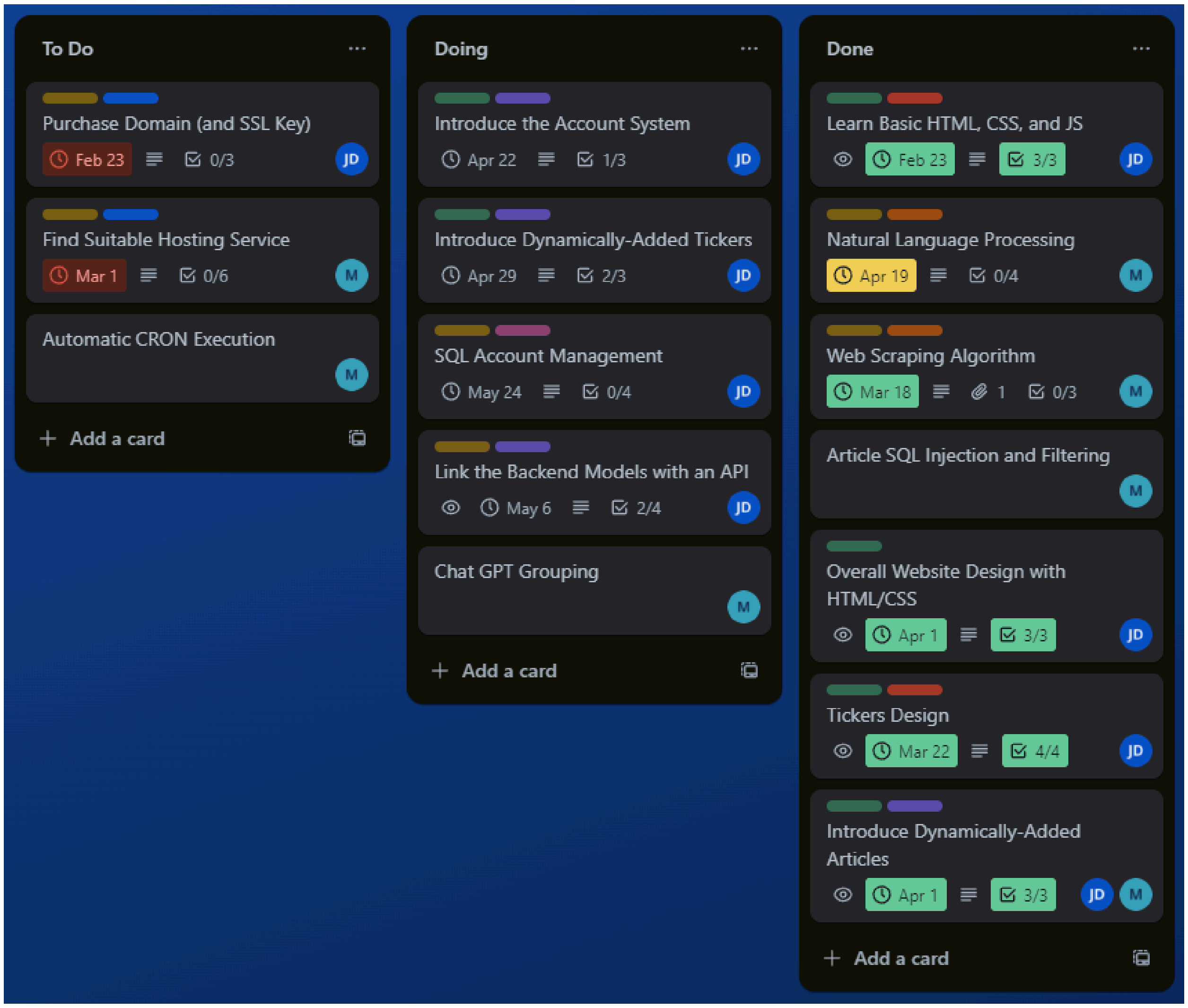
- Any small changes done to the MySQL database I cannot see because I.T. John has not come in yet to download it onto my computer
- While that is not a huge difficulty, it just requires more communication with Max on what any changes to table names are (changed twice during this period and I had to look at Max's code to see what it changed to)
- The inline HTML included in the JavaScript was difficult to figure out, but a mixture of the Internet and Claude helped me fix it in the end

Max

- SQL code is very finicky
- Writing these generalized functions was a pain in the butt, but hopefully they will be less work in the long run
- Like Jake said, I'm still deciding how I want the final database to formatted, so things change quite a bit

5. **Updated Trello Board and Discussion** (Provide screenshot of and link to updated Trello board. Discuss any changes made to board since last progress report and why.)

https://trello.com/b/sSuMPdYn/morningbread



To Do

- Purchase Domain (and SSL Key) — Feb 23 — 0/3 — JD
- Find Suitable Hosting Service — Mar 1 — 0/6 — M
- Automatic CRON Execution — M

Doing

- Introduce the Account System — Apr 22 — 1/3 — JD
- Introduce Dynamically-Added Tickers — Apr 29 — 2/3 — JD
- SQL Account Management — May 24 — 0/4 — JD
- Link the Backend Models with an API — May 6 — 2/4 — JD
- Chat GPT Grouping — M

Done

- Learn Basic HTML, CSS, and JS — Feb 23 — 3/3 — JD
- Natural Language Processing — Apr 19 — 0/4 — M
- Web Scraping Algorithm — Mar 18 — 1 — 0/3 — M
- Article SQL Injection and Filtering — M
- Overall Website Design with HTML/CSS — Apr 1 — 3/3 — JD
- Tickers Design — Mar 22 — 4/4 — JD
- Introduce Dynamically-Added Articles — Apr 1 — 3/3 — JD M

Domain and SSL Key are not top priorities despite them having such an early deadline, as well as the hosting service. Max and I will start on that soon despite how much they seem to be ignored.

6. **Tasks to Be Worked on in Next Progress Period** (Discuss the tasks to be worked on in the following two weeks. Discuss who is working on each.)

Jake

- ***If*** the article updater is finally finished relatively soon, I will update all the code I made in this period to reflect the new MySQL table

- That should hopefully not take too much time since the bulk of the code has already been made
- I hope to start on the account system this week (will need to have MySQL Workbench installed before I can do anything major)
- Maybe tickers? I'm not sure what Max's main goals are at the moment so I am not sure if that will be possible any time soon

Max

- Continue to work on duplicate deletion function
- Rewrite the chatGPT call in the ML functions file
- Refactor some of the ML functions to use dataframes instead of 2-dimensional lists
- Finish the file to be run and upload to remote server
- Automate script execution using CRON

7. **Additional Information** (Provide any additional information that you want to provide in this section; for example, one of your teammates is going away next week, your Github account is gone, etc. It could be good news as well.)

Jake

There's nothing major to report on my end. I am officially committed to WPI for data science so that's pretty chill 🤙.

Max

Nothing major to report.