SRS - Spring 2024

Ways of implementing a database

1. Connect SQL Server directly to React App through JavaScript

dbFiles > dbConfig.js > ...

```
const config = {
  user: 'CodingWithKevin',
  password: 'foo',
  server: 'DESKTOP-S8MLIO9',
  database: 'SQL Tutorial',
  options: {
    trustServerCertificate: true,
    trustedConnection: false,
    enableArithAbort: true,
    instancename: 'SQLEXPRESS'
  },
  port: 1433
}
```

```
const getEmployees = async() => {
  try {
    let pool = await sql.connect(config);
    let employees = pool.request().query("SELECT * from EmployeeDemographics")
    console.log(employees);
    return employees;
  }
  catch(error) {
    console.log(error);
  }
}

const createEmployee = async(Employee) => {
  try {
    let pool = await sql.connect(config);
    let employees = pool.request()
    .query(`INSERT INTO EmployeeDemographics VALUES
    (${Employee.EmployeeID}, '${Employee.Firstname}', '${Employee.Lastname}', ${Employee.Age}, '${Employee.Gender}')
    `)
    return employees;
  }
  catch(error) {
    console.log(error);
  }
}
```

Pt. 2

2. Python or Java code that reads in a json file, creates insert statements, and appends a .sql file full of insert statements - (from Quiz-App-REACT)

```python
import json

def generate_sql_inserts(json_filepath, output_sql_filepath):
    with open(json_filepath, 'r') as file:
        textbook_data = json.load(file)

    data_section = textbook_data['data']
    insert_statements = []

    topic_id = 1
    paragraph_id = 1
    answer_id = 1

    for section in data_section:
        # Insert statement for the topic
        topic_title = section.get('title', '').replace("'", "''")
        subject = 'Mathematics'  # Assuming the subject; change as needed
        insert_statements.append(f"INSERT INTO Topics (TopicID, Title, Subject) VALUES ({topic_id}, '{topic_title}', '{subject}');")

        for paragraph in section.get('paragraphs', []):
            # Insert statement for the paragraph
            paragraph_content = paragraph.get('context', '').replace("'", "''")
            insert_statements.append(f"INSERT INTO Paragraphs (ParagraphID, TopicID, Content) VALUES ({paragraph_id}, {topic_id}, '{paragraph_content}');")
```

React App

- Started front end for the react app with routes set up to allow for traversing the app
- Goal is to set up buttons that call java methods
- One java method to return card data from the database and run it through the algorithm
- Other method chooses a rating and updates the database accordingly

```
function Pages() {

  return (
    <>
      <BrowserRouter>
        <Routes>
          <Route path='/' element={<Home />}></Route>
          <Route path='/decks' element={<Decks />}></Route>
          <Route path='/flashcard' element={<FlashCard />}></Route>
        </Routes>
      </BrowserRouter>
    </>

  );
}

export default Pages
```

SQL database

- One main table that contains the card front and back along with a unique id to identify it
- One table per user that contains a reference to the card id along with all the scheduling info
- This allows us to not have to duplicate cards whenever a new user wants to study from the same bank of cards

```sql
DROP TABLE IF EXISTS card;
CREATE TABLE card (
  front varchar(30) NOT NULL,
  back varchar(200) NOT NULL,
  id int NOT NULL,
  PRIMARY KEY (id)
);

DROP TABLE IF EXISTS deck1;
CREATE TABLE deck1 (
    due date NOT NULL,
    stability float(5,2) NOT NULL,
    difficulty float(3,2) NOT NULL,
    elapsedDays int(6) NOT NULL,
    lapses int(3) NOT NULL,
    state char(10) NOT NULL,
    lastReview date NOT NULL,
    id int,
    FOREIGN KEY (cardMaterial) References card(id)
);
```

SQL database plans

- For the sake of Naman’s addition to algorithm which applies different weights, we need to have a method of keeping a history of all the reviews for one card
- We could possible do this by making each column a list that gets appended on every review. This would make the most recent entry the current scheduling info and getting the history would just mean iterating through the list in order
- **Separating the cards by deck**- since all the card for a user are stored in the same table, adding a deck column would allow us to filter by deck whenever the user wants to study a specific topic.

Backend

- Recently we were able to use a backend server to create an API for displaying the database information
- Through Node.js the front end is then able to take in the URL of the API and read in the data to the front end (just like slide 2)
- For now i have to figure out how to display joined tables for our database implementation and then iterate through the rows as you flip cards

Final database design

- Dian found a more straightforward way of connecting our backend server to SQL so we made the switch
- Getting the full card info now is a simple as making a query that selects all the cards on a natural join using the card id since that is what connects the front and back with the scheduling info

Flask API

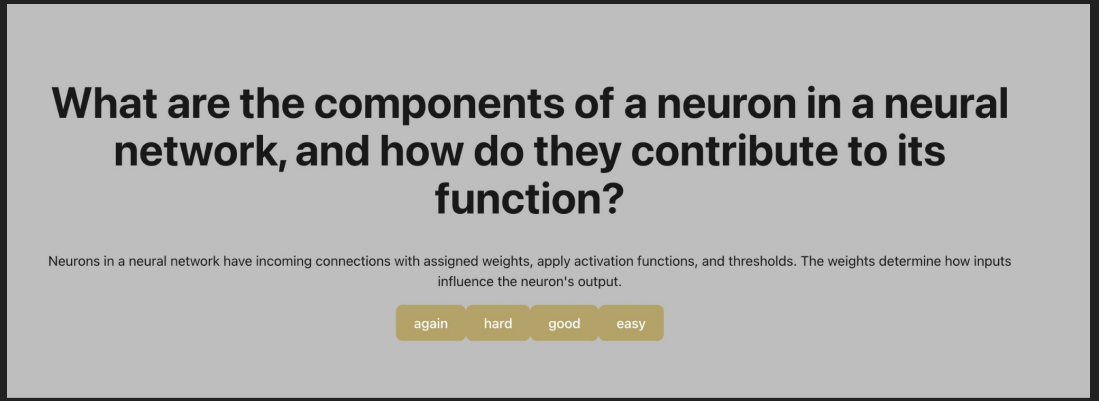
- In order to properly integrate the algorithm with the front end, we needed a way of taking the cards from the front end that you are currently interacting with, hitting a button to apply the algorithm to it and update the SQL database with the new scheduling info that came from running the algorithm
- The algorithm was previously in java but there was no easy way to combine javascript and java in a live environment so we made the switch to python so we could host a virtual environment with the backend.

Algorithm implementation

- We create a post request in the front end that returns a json with the card info of the current card id and the rating you decided to give it.
- The python script will run the algorithm on that information when you hit a button and directly interacts with the database to update the scheduling info for that card only

4 different buttons

These buttons all use the same template, the only difference being that the card rating the send to python in the post request corresponds with the displayed rating



References

- https://github.gatech.edu/VIP-ITS/SRS-Spring-2024/tree/Chaz
- Contains all the work on the frontend, backend, database file, and python code