Friday, Jul 26th 10:26 AM
Last night as sleeping I had the idea of organizing the user's codes and study sessions by using classes, so a topic would be a class and that'll hold other sub classes which are the study sessions. And they'll both have their properties. Then for the data frame I'll probably have a list of topic object and it'll be pretty simple.

Then if I need to do I'll have a code that converts it into a table or whatever I need to

Friday, Jul 26th 10:39 AM
Gonna focus on my progress and journal in this one note just so I can keep track. Wanna know what to work on next time and stuff

Friday, Jul 26th 11:16 AM
Okay so learnt about 💡 IPC, and before I start getting into that, I need to develop the main object oriented code in the backend using Python which is nice. Then later will figure out how to use 💡 IPC to get data from the user using JS at the front end

> **ChatGPT4o**
>
> - **Develop the Backend in Python**:
>   - **Objective**: Focus on building the core functionality of your application using Python, leveraging object-oriented principles to manage your data effectively.
>   - **Steps**: Develop classes and methods for your data structures (e.g., `Topic`, `StudySession`), and ensure they handle the necessary data operations.
> - **Learn and Implement IPC in Electron**:
>   - **Objective**: Understand how IPC works in Electron to facilitate communication between the frontend (JavaScript) and the backend (Python).
>   - **Steps**: Once your backend logic is solid, integrate IPC to send and receive data between the Electron main process and the renderer process.
> - **Frontend Development with JavaScript**:
>   - **Objective**: Develop the user interface using HTML, CSS, and JavaScript.
>   - **Steps**: Collect data from the user through forms or other UI elements, and send this data to the backend using IPC. Display results from the backend in the UI.

Saturday, Jul 27th 10:27 AM
Decided it's best to do an object-oriented data structure cause rn what I got is a list of lists and that'll get super confusing really fast

```
difficulty = [colour_to_difficulty(i) for i in colour]

# Add UTC timezone to dates as required by FSRS code
dates = [date.astimezone(dt.timezone.utc) for date in dates] # ensure datetimes are set to UTC as required by FSRS module

# add topic with data to table
table[topic] = [dates,difficulty]
```

Saturday, Jul 27th 10:30 AM
Oh yeah I was working on 💡 Python Backend Plan.canvas

Saturday, Jul 27th 10:54 AM
ChatGPT4o says I should use `json` when transferring data between Python and Java in the IPC

```
import sys
import json
from study_session import StudySession, Topic

def fetch_data(arg):
    # Create some example data
    algebra_session = StudySession('Algebra', '2024-07-25', 'Reviewed algebraic expressions')
    calculus_session = StudySession('Calculus', '2024-07-26', 'Introduction to derivatives')

    math_topic = Topic('Mathematics')
    math_topic.add_session(algebra_session)
    math_topic.add_session(calculus_session)

    mechanics_session = StudySession('Mechanics', '2024-07-27', 'Newton\'s laws of motion')
    optics_session = StudySession('Optics', '2024-07-28', 'Reflection and refraction')
    physics_topic = Topic('Physics')
    physics_topic.add_session(mechanics_session)
    physics_topic.add_session(optics_session)

    data = {
        'topics': [math_topic.to_dict(), physics_topic.to_dict()]
    }
    return data

if __name__ == '__main__':
    arg = sys.argv[1] if len(sys.argv) > 1 else None
    result = fetch_data(arg)
    print(json.dumps(result))
```

Saturday, Jul 27th 11:09 AM
Gonna just have AI summarize the problems and solutions when I speak with it cause it's more clear and easier and I just wanna put the ideas down on paper quickly to make as much progress as possible

> **ChatGPT4o**
>
> You initially faced issues with managing and organizing study sessions and topics within your application, unsure whether to include topic information directly in each `StudySession` or rely on a hierarchical structure. The solution you devised involves using a top-down approach, where each `Topic` contains a list of `StudySession` objects, requiring users to first add and manage topics. Once topics are established, users can then add study sessions and select the appropriate topic from a pre-populated list, streamlining data management and ensuring sessions are logically linked to their respective topics.

okay so now I'm going to have it so `StudySession` s don't need a topic property since they'll always be part of the `Topic` class

Saturday, Jul 27th 11:11 AM
I think I'll do the same with subjects so `Topic` will be part of `Subject` . So maybe can rate the student's overall confidence with the subject when maybe using AI or some objective measurement. Need to find a way to measure student's performance 💡 Measure Performance

Tuesday, Jul 30th 10:46 AM
Alright been awhile but now gonna stack the Python backend coding

Tuesday, Jul 30th 10:53 AM
Learnt about doing this to learn how probably handle statuses in Python

```
class CompletionStatus(Enum):
    COMPLETE = "complete"
    INCOMPLETE = "incomplete"
    CANCELLED = "cancelled"
```

Tuesday, Jul 30th 10:55 AM
Techniques should be a drop down but instead of you needing to pre-define all the techniques you use, it'll be like Obsidian's metadata where it'll just show all your previous inputs to allow for easier autofill

Tuesday, Jul 30th 11:40 AM
Okay finished the basics of `StudySession` and `Topic` class

```
import datetime as dt

class CompletionStatus(Enum):
    Complete = "complete"
    Incomplete = "incomplete"
    Cancelled = "cancelled"

class Difficulty(Enum):
    Hard = "Hard"
    Good = "Good"
    Easy = "Easy"

class StudySession:
    def __init__(self, date : dt.datetime, difficulty : Difficulty, creation : dt.datetime = dt.datetime.now(), status :
CompletionStatus = CompletionStatus.Complete):
        self.date = date
        self.difficulty = difficulty
        self.creation = creation
        self.start_time = None
        self.end_time = None
        self.technique = ""
        self.notes = ""
        self.status = status

    # returns basic information of study session to allow for identification and reference
    def name(self):
        return f"{self.date.strftime("%Y-%m-%d")}:{self.difficulty}"

    def to_dict(self):
        return {
            'date': self.date,
            'difficulty': self.difficulty,
            'creation': self.creation,
            'start_time': self.start_time,
            'end_time': self.end_time,
            'technique': self.technique,
            'notes': self.notes,
            'status': self.status
        }
```

```
from StudySession import  *
from typing import List

class Order(Enum):
    Higher = "Higher"
    Mid = "Mid"
    Lower = "Lower"

class Topic:
    def __init__(self, name : str, sessions : List[StudySession], order : Order = None, hidden : bool = False):
        self.name = name
        self.sessions = sessions
        self.order = order
        self.hidden = hidden

    def add_session(self, session : StudySession):
        self.sessions.append(session)

    def remove_session(self, session: StudySession):
        try:
            self.sessions.remove(session)
        except ValueError:
            print(f"Error: Session {session.name()} could not be removed since it's not a study session studied for the topic {self.name}")

    def to_dict(self):
        return {
            "name": self.name,
            "sessions": [session.to_dict() for session in self.sessions],
            "order": self.order,
            "hidden": self.hidden,
        }
```

Wednesday, Jul 31st 11:04 AM

Okay done the making the final object `Subject`

```
from Topic import *

class KnowledgeType(Enum):
    Procedural = "Procedural"
    Mixed = "Mixed"
    Declarative = "Declarative"

class Subject:
    def __init__(self, name : str, topics : List[Topic], color : str = None, knowledge_type : KnowledgeType = None):
        self.name = name
        self.topics = topics
        self.color = color
        self.knowledge_type = knowledge_type

    def add_topic(self, topic :  Topic):
        self.topics.append(topic)
        return None

    def remove_topic(self, topic : Topic):
        self.topics.remove(topic)
        return None

    def to_dict(self):
        return {
            'name': self.name,
            'topics': [topic.to_dict() for topic in self.topics],
            'color': self.color,
            'knowledge_type': self.knowledge_type.value if self.knowledge_type else None
        }
```

Wednesday, Jul 31st 11:05 AM
Next time is probably to find a way to store these objects, probably in `.json` file. So I need to probably make a function that adds a subject to a JSON file and also able to take data from a JSON file. So I need to probably make an entire new Python script or class just to read and update a JSON file

Wednesday, Jul 31st 11:12 AM
Okay but the BEST solution for my situation would be to use 💡 SQLite. So I'll learn probably like my 7th coding language this is fine.

> **ChatGPT4o**
>
> For managing and storing data in your application, there are several options, each with its own advantages and drawbacks. Here's a ranking from best to worst for your situation, considering your need for simplicity, ease of integration, and future scalability:
>
> ### 1. SQLite
>
> - **Pros**:
>   - **Lightweight and Embedded**: SQLite is a serverless database that is lightweight and embedded within the application, so you don't need a separate server process.
>   - **Structured Query Language (SQL)**: Provides powerful querying capabilities and transactions.
>   - **File-Based**: Data is stored in a single file, which simplifies backup and transfer.
>   - **Widely Supported**: Easily integrates with many programming languages, including Python and JavaScript.
> - **Cons**:
>   - **Limited Concurrency**: Not ideal for highly concurrent applications with lots of write operations.
>   - **Requires SQL Knowledge**: You'll need to write SQL queries, which might add some complexity if you're not familiar with SQL.
>
> ### 2. JSON Files
>
> - **Pros**:
>   - **Simple Format**: Easy to read and write with native support in Python and JavaScript.
>   - **Human-Readable**: Data is stored in a human-readable format, making it easy to inspect and debug.
>   - **Good for Small Projects**: Ideal for smaller-scale projects or applications with relatively simple data needs.
> - **Cons**:
>   - **Not Scalable for Large Data**: Performance may degrade with very large files or complex data structures.
>   - **Limited Querying**: Lacks the advanced querying capabilities of databases.
>
> ### 3. CSV Files
>
> - **Pros**:
>   - **Simple and Efficient**: Simple to use for tabular data; easily handled by many tools.
>   - **Easy Integration**: Widely supported and easy to integrate with various languages and libraries.
> - **Cons**:
>   - **Limited Data Types**: Best for simple, tabular data; doesn't handle nested or complex data structures well.
>   - **No Schema**: Lacks data validation and structure.

4. XML Files

- **Pros**:
  - **Structured Data**: Supports complex nested structures and metadata.
  - **Widely Supported**: Many tools and libraries support XML.
- **Cons**:
  - **Verbosity**: XML is often more verbose and harder to read than JSON.
  - **Complexity**: Parsing and manipulating XML can be more complex compared to JSON.

5. NoSQL Databases (e.g., MongoDB)

- **Pros**:
  - **Flexible Schema**: Supports unstructured or semi-structured data, and schema-less design.
  - **Scalable**: Can handle large amounts of data and high throughput.
- **Cons**:
  - **Overhead**: Requires running a database server, which adds overhead.
  - **Complexity**: More complex to set up and manage compared to file-based solutions.

Recommendation

1. SQLite is highly recommended if you need a more robust and scalable solution with the capability to handle structured data and complex queries. It provides a good balance of simplicity and functionality.

2. JSON Files are ideal for smaller projects or simpler data needs where you want human-readable files and ease of integration without setting up a database server.

For your current project, which involves organizing study sessions and topics, **SQLite** could offer more flexibility and scalability in the long term, especially if you anticipate more complex data needs or if you plan to expand the functionality of your application.

3. CSV Files and **XML Files** are less suitable for complex data structures or larger-scale applications but might still be useful for specific cases or simpler data needs.

4. NoSQL Databases are less suitable for a smaller project where setting up a database server might be overkill, but they are a good option if you plan to handle large volumes of unstructured data or need high scalability.

Wednesday, Jul 31st 11:23 AM
Okay I see the relation between `to_dict()` and 💡 SQLite, it's pretty smart. It's like the `to_dict()` is the wire and 💡 SQLite is the SSD

ChatGPT4o

Tables in databases are used to structure and organize data in a way that allows for efficient storage, retrieval, and management. Each table typically represents a type of entity or object, such as `StudySession`, `Topic`, or `Subject`, and contains columns that correspond to the attributes of these entities.

Why Use Tables?

1. **Structured Data Storage**: Tables provide a structured format to store data. Each row in a table represents an instance of the entity (e.g., a `StudySession`), and each column represents an attribute of that entity (e.g., date, difficulty).
2. **Efficient Querying**: Tables allow for efficient querying and manipulation of data. SQL (Structured Query Language) is used to perform operations such as selecting, inserting, updating, and deleting records.
3. **Data Integrity**: Tables can enforce constraints and relationships between different types of data, ensuring that the data adheres to certain rules and relationships.
4. **Scalability**: Using tables allows for scaling the database to handle large amounts of data while maintaining performance.

Relation to `to_dict()`

The `to_dict()` function in your classes converts an object into a dictionary, which can be helpful for various purposes, including:

- **Serialization**: Converting objects into a format that can be saved to a file or sent over a network.
- **Data Transfer**: When communicating between different parts of an application or between different applications.

However, the `to_dict()` function does not directly create or manage tables in a database. Instead, it helps to convert the object's data into a format (like a dictionary) that can be easily inserted into a database table or used in other contexts.

How Tables and `to_dict()` Interact

1. **Creating Tables**: You define the structure of tables in SQL, specifying the columns and their data types. This is done using SQL commands like `CREATE TABLE`.
2. **Inserting Data**: To insert data into a table, you would typically convert your object's attributes into a format that matches the table's schema. For example, you would use the data from `to_dict()` to populate the columns of a table.

```
def insert_study_session(session: StudySession):
    conn = connect_db()
    cursor = conn.cursor()

    # Convert the StudySession object to a dictionary
    session_dict = session.to_dict()

    # Insert data into the table
    cursor.execute('''
        INSERT INTO study_sessions (date, difficulty, creation, status, technique, notes)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (session_dict['date'], session_dict['difficulty'], session_dict['creation'],
          session_dict['status'], session_dict['technique'], session_dict['notes']))

    conn.commit()
    conn.close()
```

3. **Retrieving Data**: When you query data from a table, you might convert the retrieved rows into objects by mapping database columns back to object attributes.

```
def query_study_sessions():
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute('SELECT * FROM study_sessions')
    rows = cursor.fetchall()

    conn.close()

    # Convert rows to StudySession objects
    sessions = []
    for row in rows:
        sessions.append(StudySession(
            date=row[1],
            difficulty=row[2],
            creation=row[3],
            status=row[4],
            technique=row[5],
            notes=row[6]
        ))

    return sessions
```

Summary

- **Tables**: Used to organize and store data efficiently in a relational database.
- `to_dict()`: Converts an object's attributes into a dictionary format that can be used for various purposes, including inserting data into a database table.
- **Interaction**: Tables are created and managed using SQL, while `to_dict()` helps with converting objects into a format suitable for database operations.

Using tables is crucial for managing data in a relational database, while `to_dict()` is a tool to facilitate the transfer of data between your application and the database.

Wednesday, Jul 31st 11:41 AM

Tomorrow learn how to use 💡 SQLite and create a python class that handles reading and updating the `.db` file

Thursday, Aug 1st 10:48 AM

Okay problem I was having now the new plan

> **ChatGPT4o**
>
> You initially faced challenges in organizing and linking StudySessions, Topics, and Subjects within your application, using object-oriented structures where each Topic contained a list of StudySessions and each Subject held a list of Topics. The solution involved using SQLite tables to manage these relationships, with each table having an `INTEGER PRIMARY KEY AUTOINCREMENT` column that automatically generates unique IDs for each new row, so you don't need to manually create unique IDs for each object. This ensures that each StudySession, Topic, and Subject is uniquely identified and correctly linked within the database, aligning with your object-oriented approach.

Thursday, Aug 1st 10:53 AM

So I got to create an `id` attribute for my classes

ChatGPT4o

Using IDs with Python Classes

- **Object Creation**: When you create a new `Topic` object in Python, you typically don't provide an ID because it will be generated by the database when you insert it.
- **Storing IDs**: After inserting the record into the database, the database provides the unique ID of that record. This ID can then be stored in your Python object if you need to reference that specific record later.

Example Workflow

1. **Define a `Topic` Object**:

   ```python
   class Topic:
       def __init__(self, name: str, subject_id: int, order: Optional[int] = None, hidden: Optional[bool] = False):
           self.name = name
           self.subject_id = subject_id
           self.order = order
           self.hidden = hidden
           self.id = None  # Initially, the ID is not set
   ```

2. **Insert into Database**:

   ```python
   def insert_topic(self, topic: Topic):
       conn = self.connect_db()
       cursor = conn.cursor()

       cursor.execute('''
           INSERT INTO topics (name, subject_id, `order`, hidden)
           VALUES (?, ?, ?, ?)
       ''', (topic.name, topic.subject_id, topic.order, topic.hidden))

       topic.id = cursor.lastrowid  # Retrieve the ID assigned by the database
       conn.commit()
       conn.close()
   ```

   - Here, `cursor.lastrowid` retrieves the ID generated for the newly inserted `Topic`.
3. **Use the ID**:
   - Now that `topic.id` is set, you can use it to refer to this `Topic` in other database operations or when linking it to other records.

Thursday, Aug 1st 11:08 AM

A little confused on what t his does in the `study_sessions` table

```
FOREIGN KEY (topic id) REFERENCES topics(id)
```

Thursday, Aug 1st 11:10 AM

Okay next time finish `create_tables(self)`

Thursday, Aug 1st 2:40 PM

Just realized idk if the tables will keep track of Study Sessions in order. Maybe might need to add a `lapse` property. Kinda just want it to be in order naturally by being placed in the table though

Friday, Aug 2nd 8:49 AM

I don't think that topic will need a `lapse`, just study sessions

Friday, Aug 2nd 8:54 AM

Okay and this is how `lapse` will be updated when adding a new session

```
def insert_study_session(session: StudySession):
    """Insert a new study session into the database with incremented lapse."""
    conn = sqlite3.connect('study_sessions.db')
    cursor = conn.cursor()

    # Get the maximum lapse value for the topic
    cursor.execute('SELECT MAX(order) FROM study_sessions')
    max_lapse = cursor.fetchone()[0]
    new_lapse = max_lapse + 1 if max_lapse is not None else 1

    cursor.execute('''
        INSERT INTO study_sessions (date, difficulty, creation, status, technique, notes, order)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', (session.date, session.difficulty, session.creation, session.status, session.technique, session.notes, new_lapse))

    conn.commit()
    conn.close()
```

alright sick can start coding the tables now

Friday, Aug 2nd 9:37 AM

Alright done with the `create_tables`. Also made it so the `conn.commit()` only is after all the functions in the class is executed. So maybe can have a `run()`. Next time you need to figure out `insert_subjects` and ensure that the order is proper

> self.date = date
>
> self.difficulty = difficulty
>
> self.creation = creation
>
> self.start_time = None
>
> self.end_time = None
>
> self.technique = ""
>
> self.notes = ""
>
> self.completion_status = completion_status
>
> self.id = None
>
> self.order = order
>
> self.topic_id = None

Saturday, Aug 3rd 10:31 AM

Okay I need to make a flowchart on how this whole `order` and `id` stuff will work in 💡 SQLite. 💡 Python Backend Plan

Saturday, Aug 3rd 10:33 AM

We don't have to have a variable `order` since it's organized automatically by `date`. 🤯

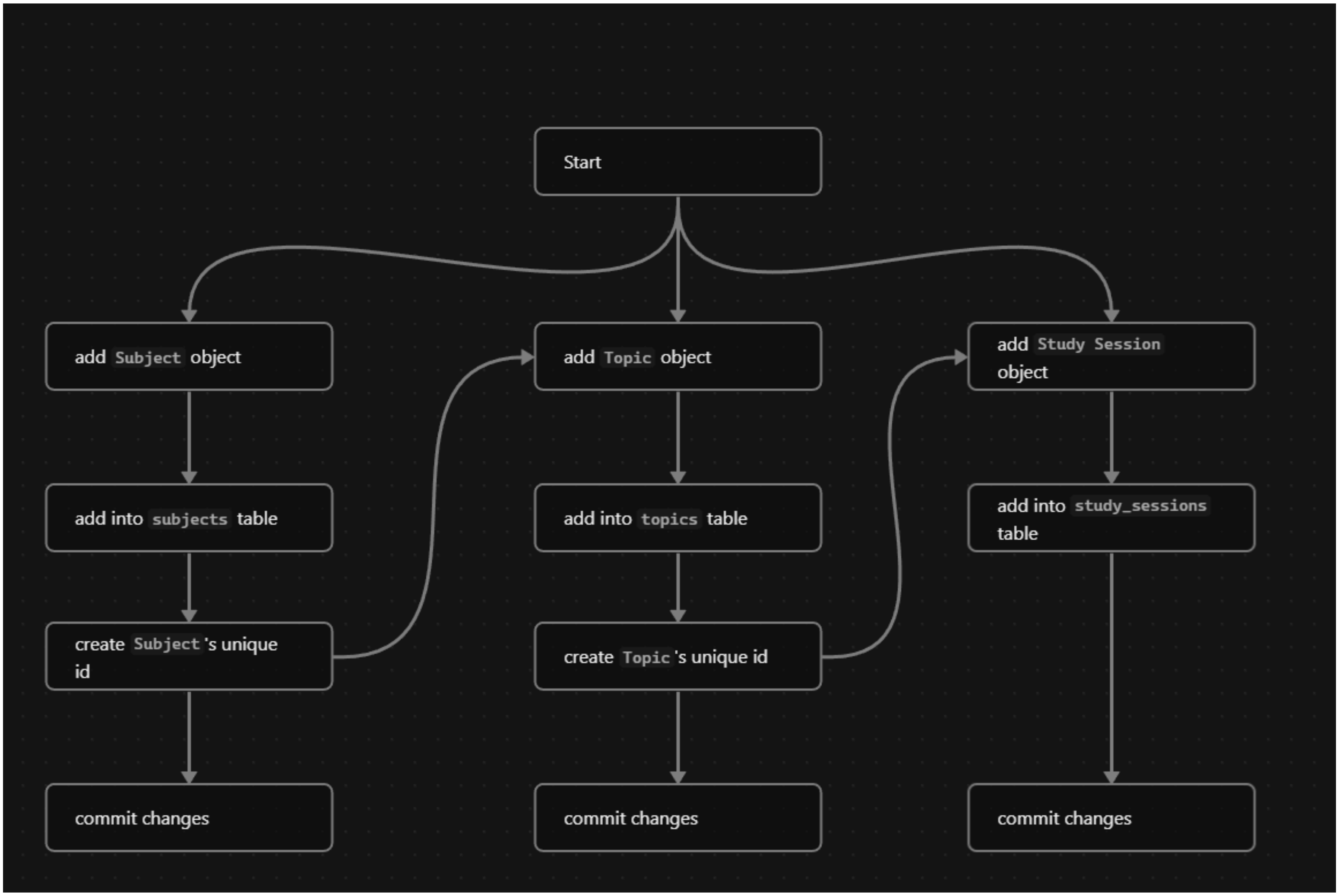
And for topic, just order it by `creation` or `name`

Now only problem is linked a table of study sessions with a topic, and a table of topics with a subject

Saturday, Aug 3rd 11:08 AM

Alright created a plan that actually makes sense in 💡 Python Backend Plan. Turns out we don't need separate tables, as long as we have the `id` links.

And for order we can organize based on `date` or `name` , easy



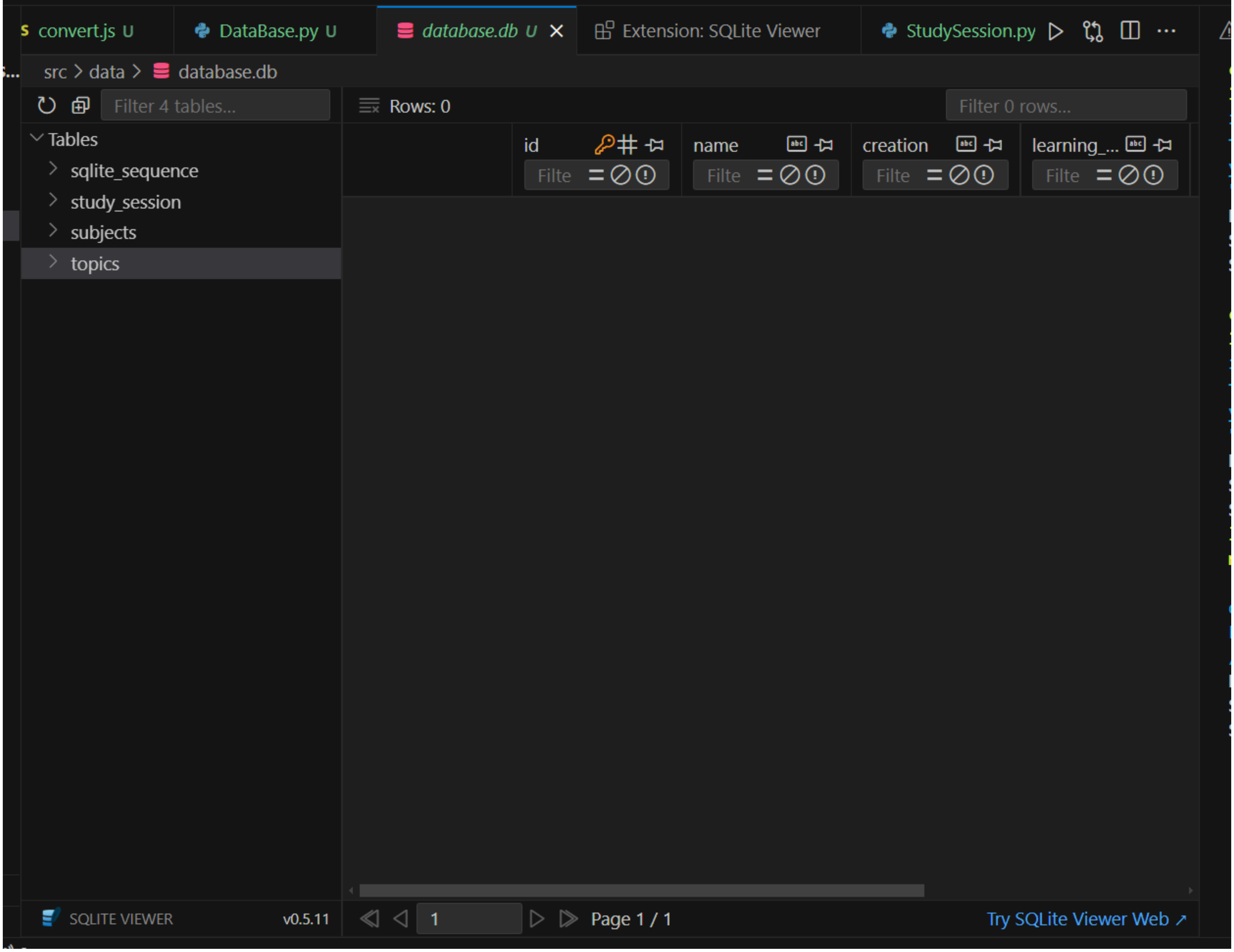
also we HAVE to commit changes so it's possible to link a study session to a topic's id. I think. Cause the user will need to choose from a drop box after adding the `Topic` , so when they add the `Study Session` , the topic id has to be ready in between drop boxes. Remember, as long as we have an 💡 undo button.

Monday, Aug 5th 10:43 AM

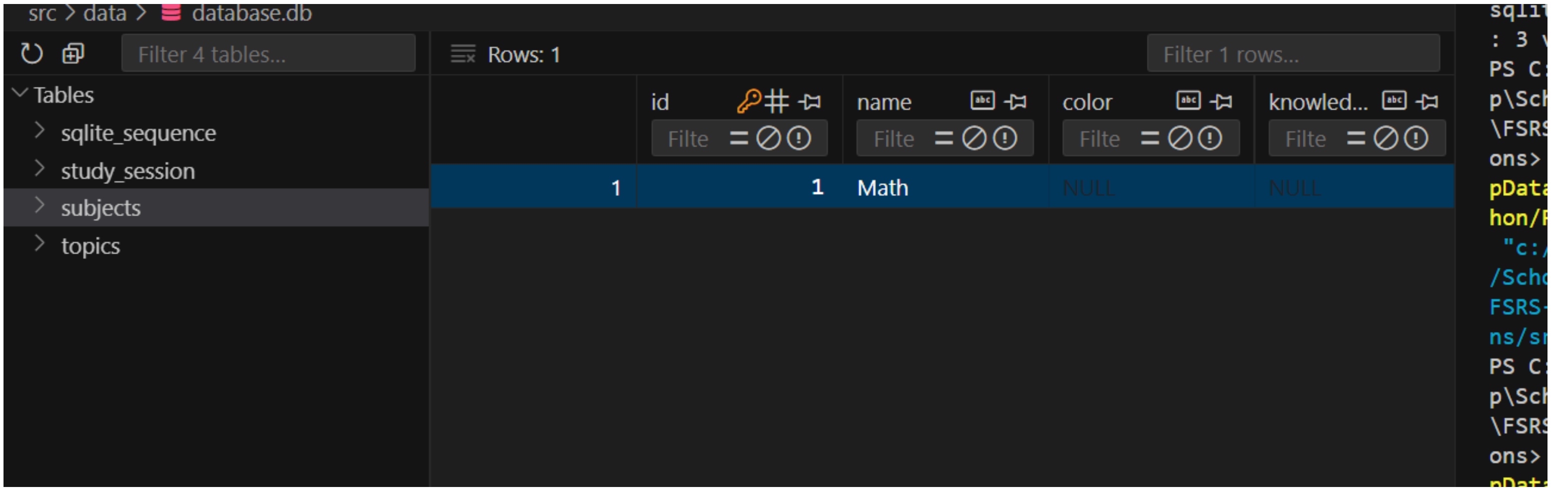
Okay now today's goal is just to finish the some functions for the database now that I know how it is organized

Monday, Aug 5th 11:10 AM

Installed SQLite Viewer to see my databases



Monday, Aug 5th 11:39 AM
Omg finally added a row



	id	name	color	knowled...
1	1	Math	NULL	NULL

okay tomorrow's goal is to try to add a `study_session` and `topic`. Bonus if you can have them connect through id.

Wednesday, Aug 7th 11:08 AM
okay let's try that in 30 mins

Wednesday, Aug 7th 11:30 AM
Realized that my custom objects like `Difficulty` complicates how they are inserted cause they need to be converted using `.value` to string, but if they are `None` by default, then that leads to an error. So need to figure that out tomorrow

Thursday, Aug 8th 11:13 AM
Okay fixed that, gonna make a 💡 Handling Exceptions notes to handle possible errors the user can encounter

Thursday, Aug 8th 11:20 AM
Next time need to link their ids, so play around with that in `DataBase` to ensure that works. Then I guess next time work on the interface? Or actually probably work on the JavaScript function's or Python function that'll be able to add those.

Actually next time make a remove function so you can remove objects from table. Then search if you can link a javascript button function to a python function. Then try to make the other python code like the FSRS calculator. Actually probably need to that next

- ☐ Remove functions + other DataBase function AI can give ideas about
    - ☑ ~~`remove`~~
    - ☑ ~~`fetch`~~
- ☑ ~~Ensure linking of `id` s work~~
- ☐ FSRS python code
- ☐ Graph python code

Friday, Aug 9th 10:09 AM
When we remove a subject, we have to remove all the topics and study sessions with it. I can't think of a way of not doing that. Cause like then we will just have such an unorganized system with unlinked topics and study sessions. My only worry is the user deletes a subject to re-add it with a different name, so I should be able to ensure users can 💡 Rename subjects.

But at the same time I should add a 💡 Warnings that allows the user to choose if they want to delete all the previous objects. Cause I think users might have a study session without a topic. But that would make the things complicated for the rest of the code. So I think it will be required to make a subject and topic. Like yeah. But then the app is not as customizable.

No but then I cannot imagine adapting the code for that then. Literally every student will have a subject. But maybe not will all have a topic hm.

Man I guess they'll just need a topic then. Maybe I can make it so you can move topics or study sessions to other subjects. Yeah, actually 100%. Cause there's gonna be a drop box that allows the user to choose, so for sure they can change if they want in the future. It's not permanently set 💡 Object Options

Friday, Aug 9th 10:21 AM
okay going to start on `remove study_session` , then link it to `remove topic` then link to `remove subject`

Friday, Aug 9th 10:28 AM
Oh boy I might need to do query first cause it seems that once you add a study session in the table, the `id` is set, and only way to get that `id` is by removing it from the table again.

Maybe the way of fixing that is that the `insert_study_session` returns the created `id` for that object so the object can have it's id set right away.

Friday, Aug 9th 10:37 AM
okay just tried that and it worked pretty well

Friday, Aug 9th 10:39 AM
Just realized `.commit()` is named that because you are commit to the changes you made so funny

Friday, Aug 9th 10:40 AM
Tomorrow work on the `fetch` and then `remove`

Saturday, Aug 10th 10:40 AM
OKAY, MAYBE I WILL, GOSH

Saturday, Aug 10th 10:47 AM
Realized I will all need a `fetchall` function where I get all subjects nested with their topics nested with their study sessions

Saturday, Aug 10th 11:09 AM
I just realized we I seriously don't need the parameters cause it's just messy and I can just directly access the object values directly, so next time might remove those. Just have it so forces just the basics, we don't need the default values.

So next time

- ☑ ~~Remove optional parameters~~
- ☑ ~~Finish making `fetch` by id~~
    - ☑ ~~`fetch_study_session`~~
    - ☑ ~~`fetch_topic`~~
    - ☑ ~~`fetch_subject`~~
- ☑ ~~Then make `fetchall` where it gets all the subjects and their topics and study sessions as objects~~
    - ☑ ~~`topic_fetchall`~~

    - ☑ ~~`subject_fetchall`~~

probably can get in one sitting if NOT DISTRACTED

Monday, Aug 12th 3:27 PM
Realized the table making the id is a bit weird because now I need to find a way where when I take out an object and place it back in it doesn't make a duplicate. Cause that's what it is doing now. So maybe when I fetch an object I remove it? Or I need to actually check where if there already exists an id then just update the values

Monday, Aug 12th 3:38 PM
Uhhh yeah requires updating and now the Enum types individual thing will be spammed so now need to make a function just for that

Monday, Aug 12th 3:46 PM
Okay done that function, just need to update code with it

Monday, Aug 12th 3:58 PM
Okay for some reason it's still duplicating despite function is done so check that next time for `insert_study_session`

Tuesday, Aug 13th 10:00 AM
Oh nvm it's working. Okay now will continue to repeat the insert update for `topic` and `study_session`

- ☑ ~~`insert_subject`~~
- ☑ ~~`insert_topic`~~

Tuesday, Aug 13th 10:19 AM
Okay back on track, now making `fetch_topic`

Tuesday, Aug 13th 12:42 PM
Okay done those fetches, okay now the `fetchall` somehow

Tuesday, Aug 13th 1:13 PM
I regret removing the parameters, so gonna add them back. pain

Tuesday, Aug 13th 1:21 PM
K done that

Tuesday, Aug 13th 1:28 PM
Okay confirmed that `id` s do link well, just now need to make `fetchall` so we can get all objects as a list

Tuesday, Aug 13th 3:03 PM
Okay finally done a milestone which is having a subject object have a list of topic and study session objects

```
print(Math.topics[0].sessions[0].difficulty)
```

going to try to have AI test my code more deeply but idk if I can cause data slow

Tuesday, Aug 13th 3:09 PM
Okay AI made a pretty good test and it works flawlessly, finally done `DataBase.py`

```
    # Create subjects
    Math = Subject("Math", color="Red")
    Science = Subject("Science", color="Blue")

    # Insert subjects
    Math.id = db.insert_subject(Math)
    Science.id = db.insert_subject(Science)

    # Create topics
    Algebra = Topic("Algebra", subject_id=Math.id)
    Calculus = Topic("Calculus", subject_id=Math.id)
    Physics = Topic("Physics", subject_id=Science.id)
    Chemistry = Topic("Chemistry", subject_id=Science.id)

    # Insert topics
    Algebra.id = db.insert_topic(Algebra)
    Calculus.id = db.insert_topic(Calculus)
    Physics.id = db.insert_topic(Physics)
    Chemistry.id = db.insert_topic(Chemistry)

    # Create study sessions
    Session1 = StudySession(dt.datetime.now(), Difficulty.Hard, topic_id=Algebra.id)
    Session2 = StudySession(dt.datetime.now(), Difficulty.Good, topic_id=Algebra.id)
    Session3 = StudySession(dt.datetime.now(), Difficulty.Easy, topic_id=Calculus.id)
    Session4 = StudySession(dt.datetime.now(), Difficulty.Hard, topic_id=Physics.id)
    Session5 = StudySession(dt.datetime.now(), Difficulty.Good, topic_id=Chemistry.id)

    # Insert study sessions
    Session1.id = db.insert_study_session(Session1)
    Session2.id = db.insert_study_session(Session2)
    Session3.id = db.insert_study_session(Session3)
    Session4.id = db.insert_study_session(Session4)
    Session5.id = db.insert_study_session(Session5)

    # Fetch and print subject details
    Math = db.fetch_subject(Math.id)
    for topic in Math.topics:
        print(f"Topic: {topic.name}")
        for session in topic.sessions:
            print(f"  Session ID: {session.id}, Difficulty: {session.difficulty}")
```

Tuesday, Aug 13th 3:11 PM

Nvm still have to make `remove` , and omg and make sure when I remove a `subject` it removes all the `topics` and then removing a `topic` removes all the `study_sessions` , I thought I was done but there was still 90% more

Tuesday, Aug 13th 3:29 PM

Okay added feature, now `DataBase.py` is officially done

Tuesday, Aug 13th 3:31 PM

Now time recreate the `FSRS` code for the third time

Tuesday, Aug 13th 3:42 PM

I made `add_session` and `remove_session` for `Topic` but I'll comment that out cause I should only add and remove sessions directly from the `.db` file to ensure id's are consistent and handled by SQL

Tuesday, Aug 13th 4:02 PM

Man this error, idk why FSRS thinks this is so important it's so annoying

```
raise ValueError("datetime must be timezone-aware and set to UTC")
```

Tuesday, Aug 13th 4:38 PM

Oh boy might need to replace `Difficulty` with `Rating` so entire code is more consistent but that's like a root thing I did so that'll take awhile to undo

Tuesday, Aug 13th 4:52 PM

Okay tried and it broke the code too much for no reason so will stick to my own class. Just will use integers in `FSRS` then

Tuesday, Aug 13th 5:08 PM

Probably should plan what I want in the graph in the 💡 Python Backend Plan

Tuesday, Aug 13th 5:23 PM

I'm not sure if latest difficulty or latest notes are needed since that's assuming there's only a 💡 Gnatt Chart. But if I make a 💡 Spacing Table it can include all that data in a way more accessible way. Like user can see their latest notes or technique from there. Maybe since make a hyperlink in the Gnatt chart that jumps to that row in the table

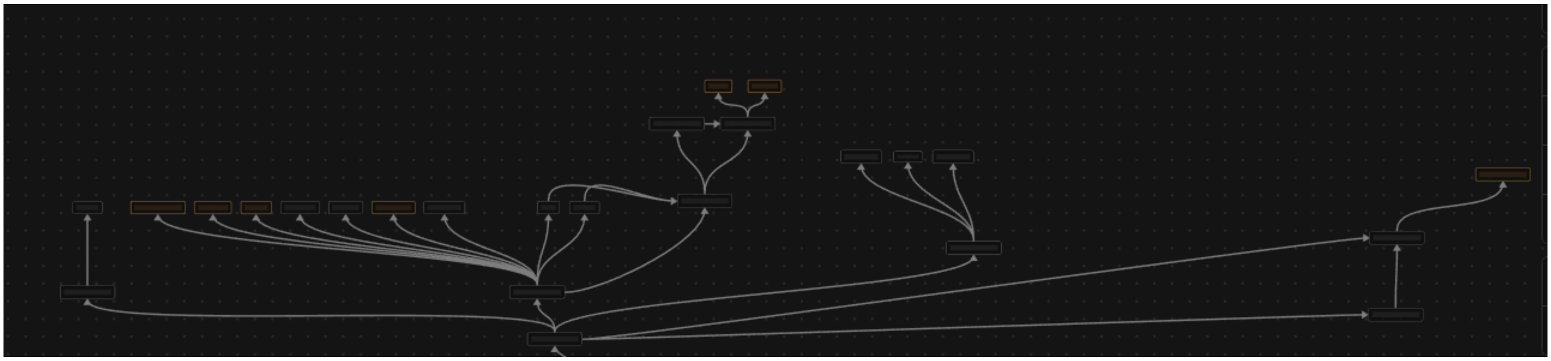
Tuesday, Aug 13th 5:31 PM

All data still should be calculated in the `TopicFSRSCalculations` . Or maybe I should make a `Statistics` class

oh god I need to plan this out

Tuesday, Aug 13th 5:55 PM

Okay planned a bit in 💡 Python Backend Plan on what data I want to have from Statistics. And if I have my own class it'll be super easy to add new stats based on what users want



Tuesday, Aug 13th 6:10 PM
Okay now will make `Statistics`

Tuesday, Aug 13th 6:30 PM
I don't even need date, or at least start_time. Date includese start_time. But I guess not everyone will know their exact start_time.

No yeah the user can place in date and it'll include the current time. And the user can change the start time if needed. And for the total time, it'll only if end time is there, which will be in the full datetime format

Tuesday, Aug 13th 6:34 PM

☑ ~~Gonna remove `start_time` and `creation` for study sessions~~

Tuesday, Aug 13th 6:50 PM

☑ ~~Now will work on `StudySessionStatistics`~~

Tuesday, Aug 13th 7:40 PM
Huh I think I'm actually done the `Statistics` class it really wasn't that bad. I say after working on it for an hour and half.

I think now I can start the `Graph` class

Tuesday, Aug 13th 7:59 PM
If I can convert my SQL table into Pandas and then add extra rows of data that'll be graph that'll be perfect. As long as I don't affect the original SQL table

Tuesday, Aug 13th 8:09 PM
That doesn't even make sense cause the whole point was that I work with Objects. So I don't need a pandas table, I just work with objects. I guess I'll still need a Pandas table to hold those calculated values

Tuesday, Aug 13th 8:13 PM

☑ ~~I need to make a `fetchall_subjects()`~~

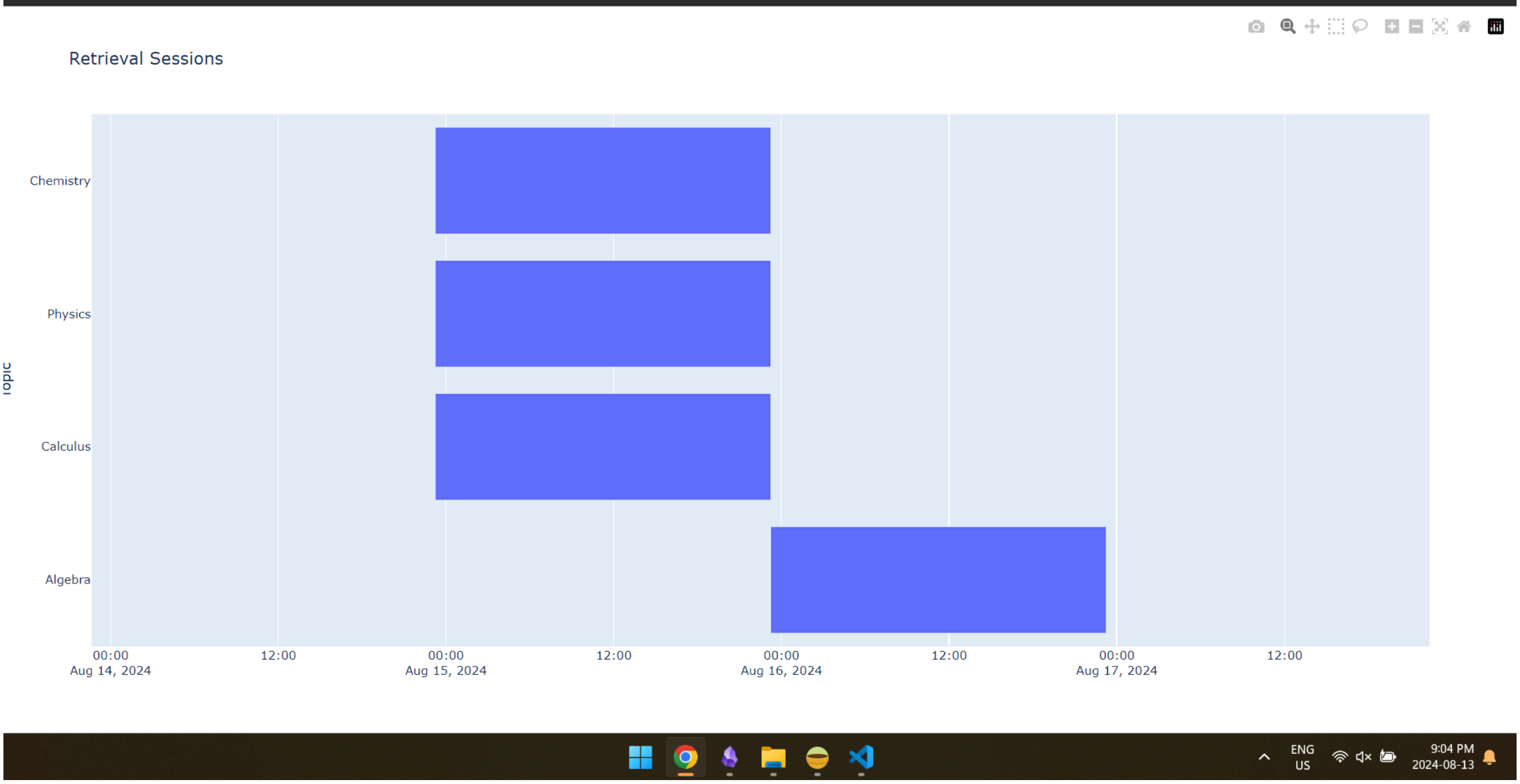
Tuesday, Aug 13th 8:17 PM
This is so hot, literally what I've been wanting since the start of coding project

```
def __init__(self, subjects : List[Subject]):
    self.subjects = subjects
    print(subjects[0].topics[0].sessions)
```

Tuesday, Aug 13th 8:27 PM
Okay updated 💡 Python Backend Plan to show how we can have different pandas tables with different stats for different graphs

Tuesday, Aug 13th 9:01 PM
Don't forget about adding subject `color` to `topics_dataframe`

Tuesday, Aug 13th 9:04 PM
Progress wowww



Tuesday, Aug 13th 9:09 PM

✅ ~~Make `timedelta_to_string`~~

Wednesday, Aug 14th 11:34 AM
Now let's ensure the color and grouping by subject works

Wednesday, Aug 14th 11:52 AM
It's actually crazy how good the code is when you create a good foundation

```
def color_map(self):
    color_map = {}

    for subject in self.subjects:
        color_map[subject.name] = subject.color

    return color_map
```

Wednesday, Aug 14th 11:53 AM
Okay well `GnattChart` is basically done. I guess next thing I want to do is the `SpacingTable` to visualize the `DataBase`, but it has to be an html or has to be interactive so idk how I'll do that

Wednesday, Aug 14th 12:11 PM
Okay this is what AI says I should do for the table

> **ChatGPT4o**
>
> Here are the files specifically related to creating the interactive table for managing study sessions, along with the purpose of each file and an outline of the functions within each file:
>
> ## File Structure Outline
>
> 1. `index.html`
>   - **Purpose:** The main HTML file that includes the structure for displaying the interactive table and modals for editing study sessions.
> 2. `render.js`
>   - **Purpose:** Handles the UI logic and interactions within the renderer process, such as populating the table, handling user inputs, and triggering modals.
> 3. `index.js`

- **Purpose:** The main process script for your Electron app. It manages IPC communications and database interactions specific to fetching and updating study session data.

4. `preload.js`
    - **Purpose:** A script that runs in the context of the renderer process, providing a secure bridge between the renderer and main processes through IPC.

Function Outline for Each File

`index.html`

- **Main Structure:**
    - `<table id="studyTable">`
        - **Purpose:** Displays topics and study sessions in a tabular format.
    - `<button id="tableBtn">`
        - **Purpose:** Triggers the population of the table with data from the database.
    - `<modal id="editSessionModal">`
        - **Purpose:** A modal popup for editing the metadata of a selected study session.

`render.js`

- **UI Interaction Functions:**
    - `fetchStudyData()`
        - **Purpose:** Fetches study data from the database via IPC and populates the table.
    - `populateTable(data)`
        - **Purpose:** Populates the HTML table with the data fetched from the database, including topics and associated study sessions.
    - `openEditSessionModal(session)`
        - **Purpose:** Opens a modal for editing a specific study session's metadata.
    - `saveSessionChanges(sessionId)`
        - **Purpose:** Sends updated session data back to the main process to save changes to the database.

`index.js`

- **IPC Handlers:**
    - `ipcMain.handle('fetch-data', async () => {...})`
        - **Purpose:** Fetches all subjects, topics, and study sessions from the database and returns the data to the renderer process.
    - `ipcMain.handle('update-session', async (event, id, data) => {...})`
        - **Purpose:** Updates a study session's data in the database based on the changes made in the renderer process.
    - `ipcMain.handle('add-session', async (event, data) => {...})`
        - **Purpose:** Adds a new study session to the database, based on user input from the renderer process.

`preload.js`

- **Exposing API to Renderer:**
    - `contextBridge.exposeInMainWorld('api', {...})`
        - **Purpose:** Provides safe APIs for the renderer process to interact with the main process via IPC.
- **API Methods:**
    - `fetchData()`
        - **Purpose:** Invokes IPC to fetch study data from the main process.
    - `updateSession(id, data)`
        - **Purpose:** Invokes IPC to update session data in the main process.
    - `addSession(data)`
        - **Purpose:** Invokes IPC to add a new study session in the main process.

These files and functions are all directly involved in creating, displaying, and managing the interactive table where users can view and edit their study sessions.

Wednesday, Aug 14th 12:17 PM

From the video this guy used

```
t> npm install better-sqlite3-with-prebuilds
```

Monday, Aug 19th 10:48 AM

Oh god right need to figure this out

Monday, Aug 19th 11:28 AM

It seems Dash could be good cause it can directly make a .html out of a Panadas table which is something I can easily make

Monday, Aug 19th 11:33 AM

Like this app shows what I could do. Maybe better than creating a JavaScript table myself from scratch but will continue to research possible ways to visualize the data from `database.db`

Monday, Aug 19th 6:00 PM

Okay let's follow this plan step by step basically. I think that's the best way\

ChatGPT4o

To create an interactive table within your Electron app that allows users to update and interact with the `.db` file by adding new `StudySession`, `Topic`, and `Subject` objects, you can follow these steps. This setup will also allow users to customize the metadata and move data within the table:

1. Designing the HTML Table Interface

- In your `index.html` (or another dedicated HTML file), create a table structure using `<table>`, `<thead>`, and `<tbody>` elements. This table will display the Topics as rows and StudySessions as columns.
- Each row will represent a Topic, and the columns will represent the details of each StudySession.

2. Populating the Table with Data

- Use `render.js` to fetch data from your SQLite database and populate the table. You can do this when the table button is clicked, or when the page loads.

```
document.getElementById('tableBtn').addEventListener('click', () => {
  fetchStudyData();
});

function fetchStudyData() {
  // Use IPC to communicate with the main process and fetch data from the database
  window.api.fetchData().then(data => {
    populateTable(data);
  });
}

function populateTable(data) {
  const tableBody = document.querySelector('#studyTable tbody');
  tableBody.innerHTML = ''; // Clear existing rows

  data.forEach(topic => {
    let row = document.createElement('tr');

    // Topic name column
    let topicCell = document.createElement('td');
    topicCell.textContent = topic.name;
    row.appendChild(topicCell);

    // StudySessions columns
    topic.sessions.forEach(session => {
      let sessionCell = document.createElement('td');
      sessionCell.textContent = session.difficulty + ' - ' + session.date;
      sessionCell.dataset.sessionId = session.id;
      sessionCell.addEventListener('click', () => {
        openEditSessionModal(session);
      });
      row.appendChild(sessionCell);
    });
    tableBody.appendChild(row);
  });
}
```

3. Editing StudySession Data

- When a user clicks on a StudySession cell, open a modal that allows them to edit the metadata of that session. This could include changing the `difficulty`, `technique`, `notes`, etc.

```
function openEditSessionModal(session) {
  // Populate modal with session data
  document.getElementById('sessionDifficulty').value = session.difficulty;
  document.getElementById('sessionNotes').value = session.notes;
  document.getElementById('sessionTechnique').value = session.technique;

  // Open modal
  const modal = document.querySelector('#editSessionModal');
  modal.classList.add('is-active');

  // Save changes
  document.getElementById('saveSessionBtn').addEventListener('click', () => {
    saveSessionChanges(session.id);
    modal.classList.remove('is-active');
  });
}
```

4. Saving Changes to the Database

- Use IPC in Electron to save the changes made in the modal back to the database.

```
function saveSessionChanges(sessionId) {
  const updatedData = {
    difficulty: document.getElementById('sessionDifficulty').value,
    notes: document.getElementById('sessionNotes').value,
    technique: document.getElementById('sessionTechnique').value,
  };

  window.api.updateSession(sessionId, updatedData).then(() => {
    // Update the table or give feedback to the user
    fetchStudyData();
  });
}
```

5. IPC Setup for Communication

- In `preload.js`, set up IPC handlers to facilitate communication between the renderer process (your table UI) and the main process (database operations).

```
const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('api', {
  fetchData: () => ipcRenderer.invoke('fetch-data'),
  updateSession: (id, data) => ipcRenderer.invoke('update-session', id, data),
});
```

6. Main Process: Handling Database Operations

- In `index.js`, handle the actual fetching and updating of data.

```
const { ipcMain } = require('electron');
const db = new DataBase('path/to/your/database.db');
```

```
ipcMain.handle('fetch-data', async () => {
  return db.fetchall_subjects(); // Adjust based on your needs
});

ipcMain.handle('update-session', async (event, id, data) => {
  const session = db.fetch_study_session(id);
  session.difficulty = data.difficulty;
  session.notes = data.notes;
  session.technique = data.technique;
  db.insert_study_session(session); // Assuming this updates the session
});
```

7. Adding New Sessions, Topics, or Subjects

- Provide buttons or inputs to add new Topics, Sessions, or Subjects. These should trigger IPC events that insert the new data into the database.

```
document.getElementById('addSessionBtn').addEventListener('click', () => {
  const newSession = {
    // Collect data from the form
  };

  window.api.addSession(newSession).then(() => {
    fetchStudyData();
  });
});
```

8. Enhancing User Experience

- You can add animations, drag-and-drop features, or any other custom visuals to make the interaction more satisfying.
- Consider using libraries like `Sortable.js` for drag-and-drop functionality within the table.

By following these steps, you can create a highly interactive and user-friendly table interface in your Electron app that allows users to manage and update their study sessions efficiently. The combination of IPC for data management, a modal for editing, and a dynamically populated table will ensure a seamless user experience.

Monday, Aug 19th 6:20 PM
Managed to make my table look good using Bulma

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@1.0.1/css/bulma.min.css">
```

Monday, Aug 19th 6:22 PM
Okay next time make sure table is in view and then start on populating table with data

Tuesday, Aug 20th 10:23 AM
Okay done having the table is view, now will start populating table despite it's basically done cause the current html table is from the `DataBase`. Maybe I can just have it so the table isn't interactable and there's just side buttons. Cause the table can easily show data and be customized as a `.html`.

Tuesday, Aug 20th 10:32 AM
I'm unsure if I want the user to interact by being able to directly interact with the table or by having separate buttons to add data to the table.

Tuesday, Aug 20th 10:43 AM
Maybe a hybrid is good, but I think directly interacting with the table would be really cool. Plus my code is adaptable enough to do that. Maybe I'll try 💡 Figma next time

Tuesday, Aug 20th 10:48 AM
Maybe instead of editing each cell individually, the user can click on a row and it opens a module where the user can edit from there

Tuesday, Aug 20th 6:12 PM
Okay nice finally on the fun part of the code. So tomorrow figure out the layout from using this app. If you want to make tables interactive got to figure out how to show that in Figma. So far the buttons are interactive

Wednesday, Aug 21st 7:03 PM
I need to figure out how to make an interactable checkbox but it's probably not needed for me rn cause I'll actually add the options when coding, so probably can spend time keep designing the modals

Wednesday, Aug 21st 7:15 PM

Got to still add an intro page to have the user add their subjects and topics

Wednesday, Aug 21st 7:26 PM

Next time figure out how the modals for

- ☑ ~~Topic~~
- ☑ ~~Study Session\~~

Thursday, Aug 22nd 6:33 PM

Gonna do the topic modal

Thursday, Aug 22nd 7:26 PM

- ☐ Add notes for topics and subjects

Thursday, Aug 22nd 7:27 PM

I might need to rethink on how to reorganize buttons so it's more modular when I want to add or remove buttons. Probably don't be fancy and just have all buttons in a linear order. And then made on the side can have the stats such. That'll probably be better. Or maybe the first top half off the modal are the buttons and second bottom half is the stats? Or maybe the table. No table should be it's own screen hmmm cause there can place other visuals like pie chart

Friday, Aug 23rd 9:21 PM

- ☒ Code a start time

Friday, Aug 23rd 9:24 PM

Hmm cause we cant have a start time cause it makes sense how I made it where date is start_time + date and end time is end date + end time. Maybe I can have it extracted from the front end and placed properly in the backend

Yeah that's probably what I'll do

Friday, Aug 23rd 10:24 PM

How will I know what is the object id when a user hovers over the html file if the id column will be removed for the user? Can I make the id column invisible?
💡 Table Customization

Friday, Aug 23rd 10:45 PM

- ☑ ~~Okay now making the subjects html and need to export it~~

Friday, Aug 23rd 10:52 PM

- ☐ Should ensure this does not give an error if there's no sessions for topic

```
def latest_time(self):
    return StudySessionStats(self.topic.sessions[-1]).total_time()
```

Friday, Aug 23rd 11:02 PM

Maybe I should make it so whatever is in the table is editable, and the stats could be visualized

Friday, Aug 23rd 11:18 PM

Yeah tomorrow you should technically add notes to stuff but really should figure out the design since you can can code while on the bus

Saturday, Aug 24th 6:15 PM

Man the tables I have in `Graph` should be in `DataBase` cause we want all data from `DataBase` to have all the data and then we can display extra later or something. Actually I need to focus on one thing at a time. Let's just do design only. So just plan what buttons I want and where I want and then I'll code it later. I think that's the best approach. Just place all my ideas and see how it'll look and figure out code later. Cause I can't be going back and forth

Okay with that said, let's work on only what the Study Session page will have

Tuesday, Aug 27th 6:11 PM

- ☐ Might need to recode so that total time is actually total time from the timer not including breaks
    - ☐ But also have it so that we need know long total time was without breaks. Also total time can be paused

Wednesday, Aug 28th 3:15 PM

- ☐ Need to have a current mastery level for topic that can be updated in 💡 Start Study Session Timer 💡 Topic Learning Efficiency Tracker

Wednesday, Aug 28th 3:24 PM

- ☐ Should have 💡 iCS mode where it shows or hide advance learning theory buttons like
    - Mastery Level

Wednesday, Aug 28th 4:07 PM
Realized that all my styles should be variables, and styles are made to combine multiple properties.

Wednesday, Aug 28th 4:20 PM
Okay I think the app is basically done, now I just need to finish the prototype of how one clicks around the app, then test myself, then have 😜 Mama to test, and then to start coding

Friday, Aug 30th 9:29 AM
Adding some advice from 💡 Mama Feedback

Friday, Aug 30th 9:56 AM
If I can make my menus like this that would be perfect

Friday, Aug 30th 10:19 AM
Just realized I need a tree graph that shows all the possible settings cause designing each one is too much and also whatever I design will most likely not be in the end product
💡 Settings

Friday, Aug 30th 11:20 AM
Okay applied the advice and I think I am basically done the design

Friday, Aug 30th 8:09 PM
Okay going to research on how to get this app into code

Friday, Aug 30th 8:16 PM
Okay this is the process I should go for, seems simple enough

> **ChatGPT4o**
>
> The process of going from a Figma design to actual working code typically follows a structured workflow. Here's the usual order of operations:
>
> ## 1. Design Phase (Figma)
>
> - **Create the Visual Design**: Start by creating the full visual design in Figma, including all UI elements (buttons, forms, layouts, etc.). This phase focuses on what the app will look like.
> - **Define Interactions**: In Figma, you can also define some basic interactions (like hover states or button clicks) to visualize how users will interact with your app. This helps in planning the flow of the application.
>
> ## 2. Export Assets (from Figma)
>
> - **Export Design Assets**: Export necessary design assets (like icons, images, etc.) from Figma. These will be used directly in your code.
> - **Document Dimensions & Styles**: Take note of the dimensions, margins, padding, colors, fonts, etc., as defined in your Figma design. These will guide your CSS styling.
>
> ## 3. Set Up the Project Structure
>
> - **Initialize the Project**: Set up the project environment. For Electron, this involves creating the necessary files (`index.html` , `index.js` , `render.js` , etc.).
> - **Install Dependencies**: Install any required dependencies like `Electron` , `Node.js` modules, and Python libraries.
>
> ## 4. Translate the Design to Code (Frontend Development)
>
> - **Create HTML/CSS Structure**: Start by translating the Figma design into HTML and CSS. Focus on creating a static version of your app with all UI components in place.
> - **Layout and Styling**: Apply the styles (CSS) according to your Figma design. Ensure that the layout matches what you designed in Figma.
>
> ## 5. Add Interactivity (Frontend Development)

- **Basic Interactions**: Implement basic interactions using JavaScript (like button clicks, hover effects, etc.). At this point, the buttons might not be connected to any backend logic but should visually respond to user actions.
- **Event Listeners**: Add event listeners to buttons and other interactive elements to prepare them for connecting to your backend logic.

6. Integrate Backend Logic (Backend Development)

- **Connect Frontend with Backend**: This is where you link your UI elements (buttons, forms) to your backend logic (Python scripts, database operations). This involves making your buttons and other UI components functional.
- **Data Flow**: Set up the data flow between the front-end and back-end, ensuring that when a user interacts with the UI, the appropriate Python scripts are executed, and the results are displayed back on the UI.

7. Testing and Debugging

- **Test the Interface**: Run your application and test all interactions. Ensure that all buttons perform their intended actions and that the data is processed and displayed correctly.
- **Debug Issues**: Fix any bugs or issues that arise during testing. This might involve adjusting your front-end code, back-end code, or both.

8. Optimize and Finalize

- **Performance Optimization**: Optimize the performance of your application. This might include optimizing your Python code, reducing the load time of assets, or improving the efficiency of data queries.
- **Refine the UI**: Make any final adjustments to the UI for consistency with the Figma design. Ensure that the application is user-friendly and visually appealing.

9. Deployment

- **Package the App**: For Electron apps, package the application for distribution. This can involve creating installers for different operating systems.
- **Deploy**: Deploy your application to the desired platforms (Windows, macOS, Linux).

Summary

- **Design First**: Start by finalizing your design in Figma. It’s essential to have a clear visual and functional blueprint before diving into code.
- **Frontend Development**: Translate the design into a static frontend (HTML/CSS), then add interactivity with JavaScript.
- **Backend Integration**: Connect the frontend to your backend logic (Python/SQL).
- **Test and Refine**: Ensure everything works as expected, make adjustments, and optimize performance.

This approach ensures that you have a clear and consistent plan from design to implementation, resulting in a more cohesive and functional application.

Friday, Aug 30th 8:23 PM
Holy nvm it turns out I can customize font so will do that

Friday, Aug 30th 8:24 PM
okay nvm still trash

Friday, Aug 30th 8:43 PM
Okay after watching a quick video it seems it's pretty important to make sure everything is perfect before implementing in. Like the spacing and all. Unless I can get it right once and reuse it later. I also need to understand css

Friday, Aug 30th 8:48 PM
Okay it seems that making a theme is pretty easy, just need a bunch of variables

```css
/* theme.css */
:root {
    --primary-color: #4CAF50;
    --secondary-color: #FFC107;
    --primary-hover-color: #45A049;
    --secondary-hover-color: #FFB300;
    --background-color: #f5f5f5;
    --text-color: #333;
    --font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
```

Friday, Aug 30th 9:29 PM
Okay spoke to AI for awhile and these should be the steps

> **ChatGPT4o**

1. Design and Structure Your HTML First

A. Review the Figma Design

- **Identify Key Elements**: Understand the layout, components, and structure of your Figma design.
- **Map Out the Pages**: Determine the different pages or views in your app and the elements on each page (headers, buttons, forms, etc.).

B. Set Up Your Electron Project

- **Create Basic HTML Structure**: Begin with your `index.html` file, which serves as the main entry point for your Electron app.
- **Include Bulma for Initial Styling**: Link Bulma's CSS for basic styles and layout.
- **Build the Layout**: Use semantic HTML elements to structure your content logically.

Example Basic HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Electron App</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css"> <!-- Bulma CSS -->
    <link rel="stylesheet" href="styles.css"> <!-- Custom CSS -->
</head>
<body>
    <header class="hero is-primary">
        <div class="hero-body">
            <h1 class="title">App Title</h1>
            <p class="subtitle">Subtitle here</p>
        </div>
    </header>
    <main class="section">
        <div class="container">
            <button class="button is-primary">Click Me</button>
        </div>
    </main>
</body>
</html>
```

2. Style Using Bulma and Custom CSS

A. Apply Bulma Classes for Quick Styling

- **Utilize Bulma's Grid and Layout System**: Start with Bulma's default classes to arrange and style your content.
- **Add Components**: Use Bulma's components for buttons, forms, modals, etc., to build out the interface.

B. Identify Customization Needs

- **Compare with Figma Design**: Identify areas where Bulma's default styling doesn't match your Figma design.
- **Write Custom CSS**: Override Bulma's styles with your own custom CSS to match the design specifics.

Example Custom CSS:

```css
/* styles.css */

/* Customizing the hero section */
.hero.is-primary {
    background-color: #FF6347; /* Custom color from Figma */
    color: white;
}

/* Customizing the button */
.button.is-primary {
    background-color: #FF4500; /* Custom button color */
    padding: 12px 24px; /* Custom padding */
    font-family: 'Your Figma Font', sans-serif;
    border-radius: 10px; /* Custom border-radius */
}
```

3. Consider Using Sass for Advanced Customization

As your project grows, you might find the need for more advanced CSS management. Sass (Syntactically Awesome Style Sheets) is a CSS preprocessor that allows you to use variables, nested rules, mixins, and more. This can be particularly useful if you need to create a more complex or scalable style system that matches your Figma design closely.

Steps to Integrate Sass:

1. **Install Sass**:
    - You can install Sass via npm if you're using a build tool:

    ```
    npm install sass --save-dev
    ```

2. **Create a Sass File**:
    - Use a `.scss` file (e.g., `styles.scss`) where you can define variables, mixins, and your custom styles.
    - Import Bulma's Sass files if you need to override its default variables.
3. **Compile Sass to CSS**:
    - Compile your `.scss` files into regular CSS that your Electron app can use:

    ```
    sass styles.scss styles.css
    ```

4. **Include the Compiled CSS**:
    - Link the compiled `styles.css` in your HTML file.

4. Implement Multiple Themes

Implementing multiple themes allows users to switch between different visual styles (e.g., light and dark modes) in your Electron app.

A. Define CSS Variables for Theming

- Use CSS variables to define your theme properties like colors, fonts, etc., in your `styles.css` or `styles.scss` file.
- Create different themes by defining variables under different class scopes (e.g., `.light-theme`, `.dark-theme`).

Example Theme Definition in CSS:

```
/* Define default (light) theme */
:root {
    --primary-color: #FF6347;
    --background-color: #FFFFFF;
    --text-color: #333;
}

/* Define dark theme */
.dark-theme {
    --primary-color: #333333;
    --background-color: #121212;
    --text-color: #FFFFFF;
}

/* Apply variables */
body {
    background-color: var(--background-color);
    color: var(--text-color);
}

.button {
    background-color: var(--primary-color);
}
```

B. Implement Theme Switching in JavaScript

- Use JavaScript to toggle between themes by adding/removing the relevant theme class (e.g., `.dark-theme`) on the `<body>` element.
- Save the user's theme preference in `localStorage` to persist the theme across sessions.

Example Theme Switcher in JavaScript:

```
// Function to toggle themes
function toggleTheme() {
    const body = document.body;
    const currentTheme = body.classList.contains('dark-theme') ? 'light' : 'dark';
    body.classList.toggle('dark-theme');
    localStorage.setItem('theme', currentTheme);
}

// Apply the stored theme on page load
document.addEventListener('DOMContentLoaded', () => {
    const storedTheme = localStorage.getItem('theme') || 'light';
    if (storedTheme === 'dark') {
        document.body.classList.add('dark-theme');
    }
});

// Example usage: attach to a button click
document.getElementById('themeToggleBtn').addEventListener('click', toggleTheme);
```

Example HTML for Theme Toggle Button:

```
<button id="themeToggleBtn" class="button">Toggle Theme</button>
```

5. Add Interactivity Using JavaScript

A. Set Up Event Listeners

- **Use `render.js`**: In your `render.js` file, set up event listeners for your buttons and other interactive elements.
- **Basic Example**:

  ```
  const { exec } = require('child_process');

  document.getElementById('button').addEventListener('click', () => {
      exec('python3 path_to_your_script/YourScript.py', (error, stdout, stderr) => {
          if (error) {
              console.error(`Error: ${error}`);
              return;
          }
          console.log(`Output: ${stdout}`);
      });
  });
  ```

B. Connect to Python Backend

- **Call Python Scripts**: Use Node.js's `child_process` module to call your Python scripts from the Electron app.
- **Handle Outputs**: Process the output from Python and update the UI accordingly.

6. Integrate with Your Python Backend

A. Review Your Python Scripts

- **Understand the Logic**: Make sure you know what each script does (e.g., `Graph.py` generates a graph, `DataBase.py` handles database queries).

B. Trigger Python Scripts from the UI

- **Use `child_process` in `render.js`**: Trigger the execution of Python scripts based on user actions in the UI.
- **Example Integration**:

  ```
  document.getElementById('graphBtn').addEventListener('click', () => {
      exec('python3 path_to_your_script/Graph.py', (error, stdout, stderr) => {
          if (error) {
  ```

```
            console.error(`Error: ${error}`);
            return;
        }
        console.log(`Output: ${stdout}`);
        // Update the UI with the result, e.g., display a graph
    });
});
```

7. Test and Refine the Application

A. Test the UI and Backend Integration

- **Run in Electron**: Test how the HTML, CSS, and backend interact.
- **Check Console for Errors**: Debug any issues by checking the console output and refining your JavaScript and Python code.

B. Iterate Based on Feedback

- **Adjust CSS**: Refine the styles as you test on different devices and screen sizes.
- **Refine Interactivity**: Ensure that all buttons and forms interact correctly with the Python backend.

8. Final Steps and Enhancements

A. Add Additional Pages

- **Repeat the Process**: For each new page or feature, repeat the steps above—building the HTML structure, applying Bulma, customizing CSS, and integrating with the backend.

B. Optimize and Clean Up

- **Refactor Code**: As your project grows, consider refactoring your CSS, JavaScript, and Python code for better maintainability.
- **Add Sass (Optional)**: If needed, transition to using Sass for more advanced CSS management.
- **Enhance Theme Functionality**: Add more themes or refine the existing ones based on user feedback and requirements.

Summary

- **HTML First**: Start by building the HTML structure based on your Figma design.
- **Bulma for Styling**: Use Bulma to quickly style your app, then customize

the CSS to match your Figma design.

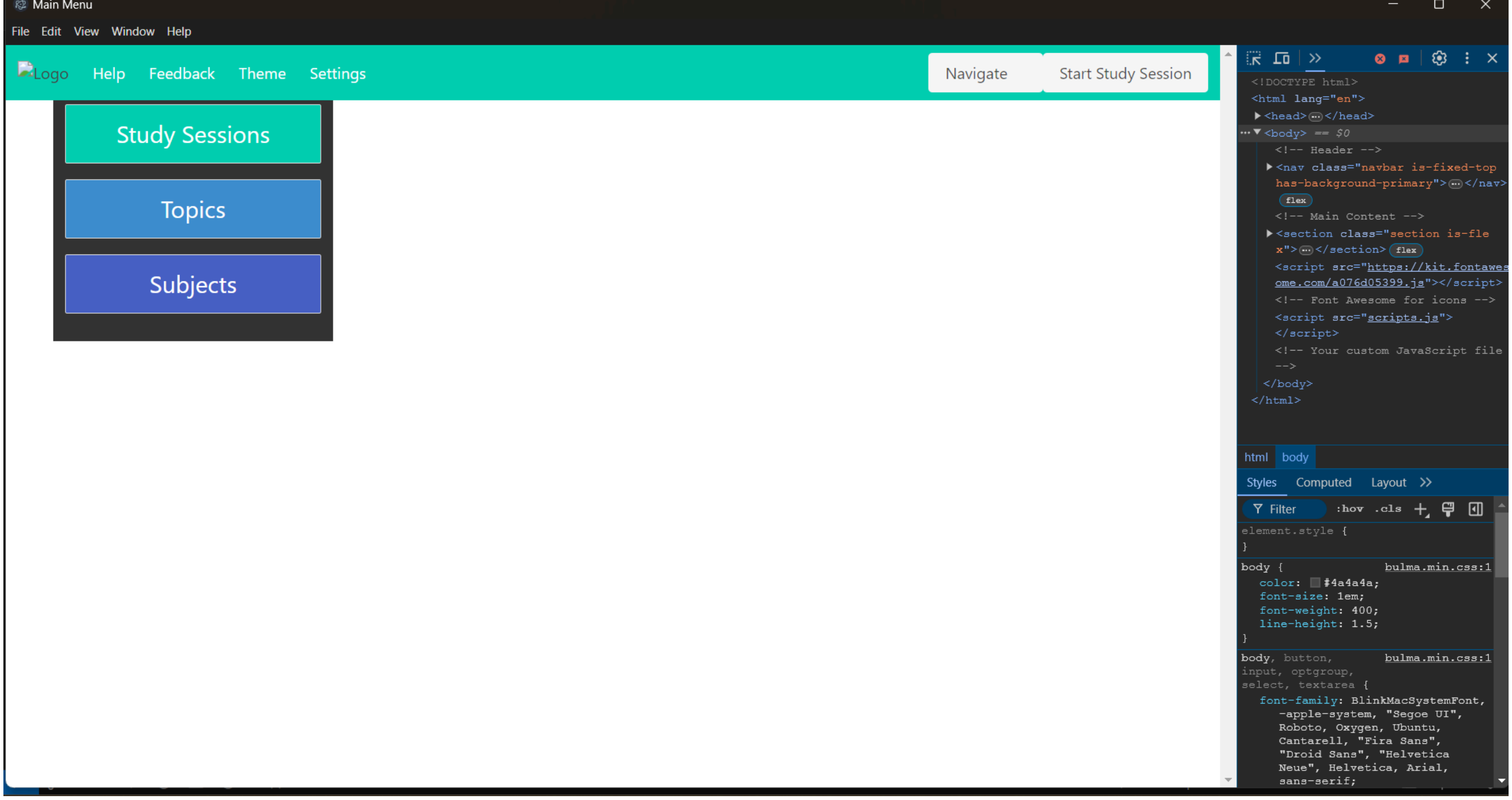
- **Sass for Advanced Customization**: Consider using Sass if your project requires more complex or scalable CSS management.
- **Implement Multiple Themes**: Use CSS variables and JavaScript to create and switch between different themes, such as light and dark modes.
- **JavaScript for Interactivity**: Add interactivity using JavaScript and connect to your Python backend.
- **Test and Iterate**: Continuously test, refine, and improve your app until it fully aligns with your design and functionality requirements.

By following this approach, you can efficiently develop a flexible, well-designed Electron app that not only stays true to your Figma design but also offers a dynamic user experience with multiple themes and rich interactivity connected to your Python backend.

> **ChatGPT4o**

Friday, Aug 30th 9:41 PM
Also I might just be able to use AI to help me start my html code by screenshotting the website which is pretty op if it works

Friday, Aug 30th 9:44 PM
Pretty good start for someone who knows nothing about html



```
<!DOCTYPE html>
<html lang="en">
  <head>…</head>
… <body> == $0
    <!-- Header -->
    <nav class="navbar is-fixed-top
    has-background-primary">…</nav>
     flex
    <!-- Main Content -->
    <section class="section is-fle
    x">…</section> flex
    <script src="https://kit.fontawes
    ome.com/a076d05399.js"></script>
    <!-- Font Awesome for icons -->
    <script src="scripts.js">
    </script>
    <!-- Your custom JavaScript file
    -->
  </body>
</html>
```

```
element.style {
}
body {                     bulma.min.css:1
   color: #4a4a4a;
   font-size: 1em;
   font-weight: 400;
   line-height: 1.5;
}
body, button,              bulma.min.css:1
input, optgroup,
select, textarea {
   font-family: BlinkMacSystemFont,
      -apple-system, "Segoe UI",
      Roboto, Oxygen, Ubuntu,
      Cantarell, "Fira Sans",
      "Droid Sans", "Helvetica
      Neue", Helvetica, Arial,
      sans-serif;
```

Friday, Aug 30th 10:02 PM

Okay tomorrow you can figure out how to code in html. Try to ensure you are doing most of the code so you are customizing it. possible you also need to customize html as going so padding is proper. Although remember that css is something you can do after the html set up

Saturday, Aug 31st 9:55 AM

Biggest concerns now is how Bulma does not has a normal text variable and also how to create a theme that does not have both light and dark mode

Saturday, Aug 31st 10:04 AM

Okay this is what I got to try next time

ChatGPT4o

If you want to **add** new text color classes (like `has-text-default` and `has-text-secondary`) to Bulma without overwriting any existing styles, you can do this by simply extending Bulma with your custom classes. This approach allows you to keep all of Bulma's predefined classes intact while adding your own.

Steps to Add Custom Text Color Classes to Bulma

1. Create a Custom Sass File

You'll define your new color variables and classes in a custom Sass file. This file will import Bulma, add your new classes, and then compile everything together.

Example: `custom-bulma.scss`

```
// Step 1: Import Bulma's base styles
@import "node_modules/bulma/sass/utilities/_all";

// Step 2: Define your custom color variables
$default-text-color: hsl(0, 0%, 20%); // Example: dark gray
$secondary-text-color: hsl(0, 0%, 50%); // Example: medium gray

// Step 3: Add custom text color classes
.has-text-default {
    color: $default-text-color;
}

.has-text-secondary {
    color: $secondary-text-color;
}
```

```
// Step 4: Import the rest of Bulma's styles
@import "node_modules/bulma/bulma";
```

2. Compile the Sass to CSS

Next, compile your custom Sass file into a CSS file that you can use in your project.

```
sass custom-bulma.scss custom-bulma.css
```

This will generate a `custom-bulma.css` file that includes all of Bulma's existing styles plus your new custom classes.

3. Link the Compiled CSS in Your HTML

Include the newly compiled CSS file in your HTML file to apply both Bulma's and your custom styles.

```
<link rel="stylesheet" href="custom-bulma.css">
```

4. Use Your New Text Color Classes

You can now use your custom text color classes in your HTML, just like any of Bulma's predefined classes.

Example:

```
<p class="has-text-default">This is default text color.</p>
<p class="has-text-secondary">This is secondary text color.</p>
```

5. Summary

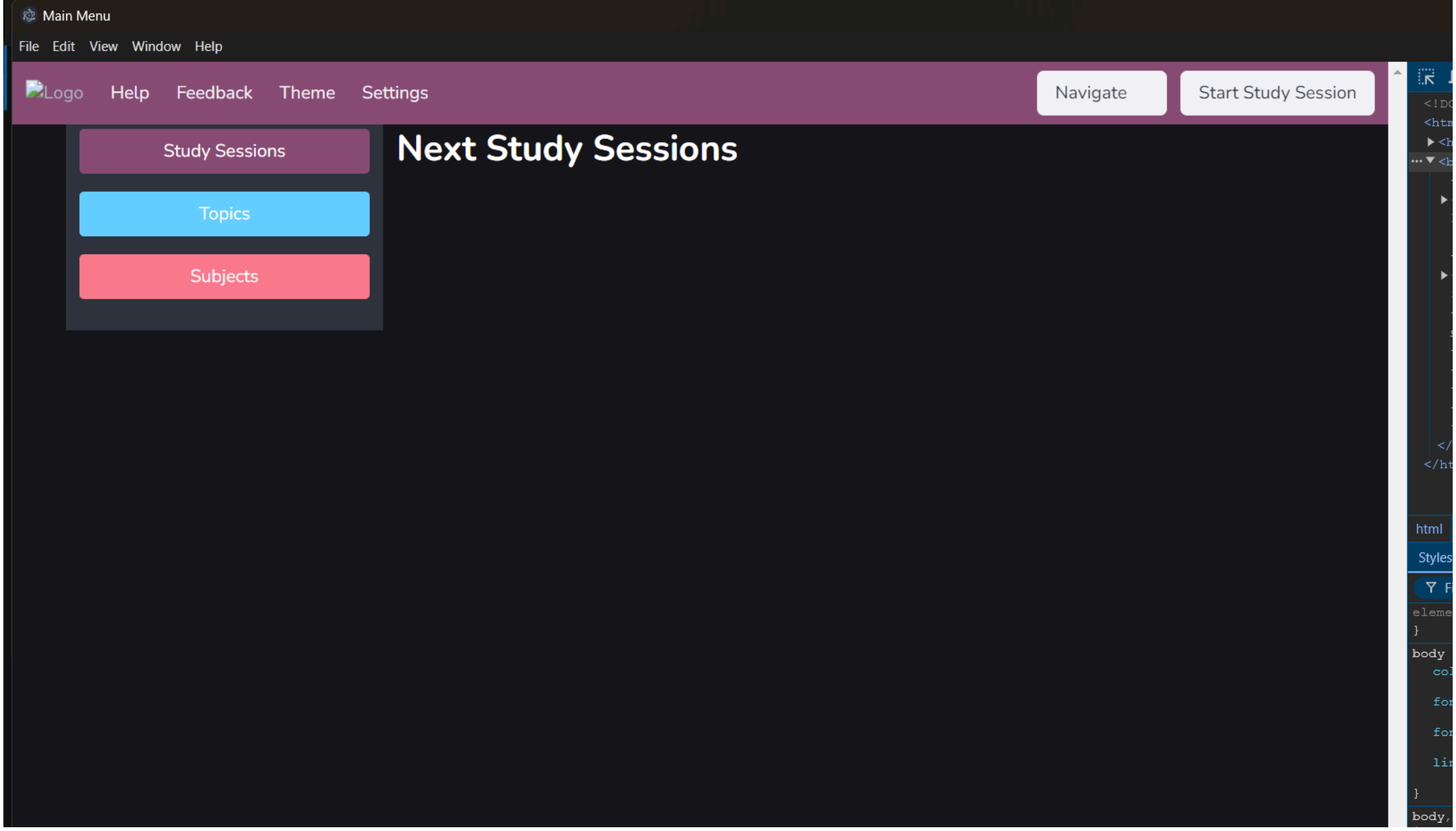
- **Keep Bulma Intact**: By following this approach, you don't overwrite any of Bulma's existing classes. Instead, you extend Bulma with new, custom classes.
- **Modular and Maintainable**: Your custom classes are compiled alongside Bulma's, ensuring that your project remains modular and maintainable.
- **Easy Integration**: Adding custom classes like `has-text-default` and `has-text-secondary` into Bulma allows you to seamlessly integrate your own design language while leveraging the full power of Bulma's utility classes.

This way, you're extending Bulma with new functionality without altering its existing behavior, making your application both flexible and future-proof.

> ” ChatGPT4o

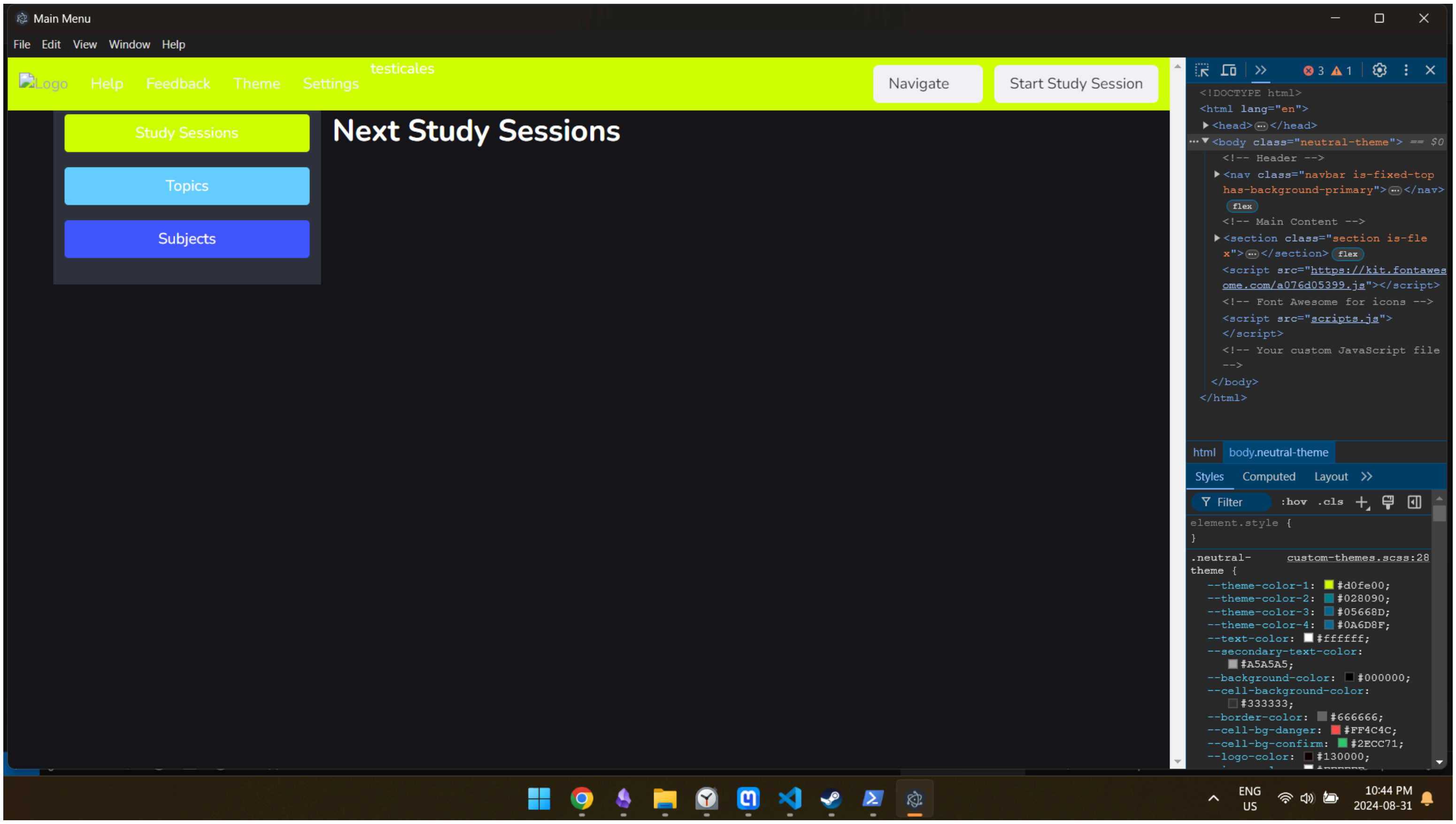
Saturday, Aug 31st 6:04 PM
Going to work on trying to make a custom theme so I know it'll work as I work on the HTML

Saturday, Aug 31st 10:01 PM
Finally got something to work



Saturday, Aug 31st 10:44 PM

Okay figured out how to change text colour and primary button colour finally



```
<!DOCTYPE html>
<html lang="en">
  <head>⋯</head>
… <body class="neutral-theme"> == $0
    <!-- Header -->
    <nav class="navbar is-fixed-top
    has-background-primary">⋯</nav>
    flex
    <!-- Main Content -->
    <section class="section is-fle
    x">⋯</section> flex
    <script src="https://kit.fontawes
    ome.com/a076d05399.js"></script>
    <!-- Font Awesome for icons -->
    <script src="scripts.js">
    </script>
    <!-- Your custom JavaScript file
    -->
  </body>
</html>
```

```
element.style {
}
.neutral-      custom-themes.scss:28
theme {
   --theme-color-1: #d0fe00;
   --theme-color-2: #028090;
   --theme-color-3: #05668D;
   --theme-color-4: #0A6D8F;
   --text-color: #ffffff;
   --secondary-text-color:
      #A5A5A5;
   --background-color: #000000;
   --cell-background-color:
      #333333;
   --border-color: #666666;
   --cell-bg-danger: #FF4C4C;
   --cell-bg-confirm: #2ECC71;
   --logo-color: #130000;
```

now tomorrow need to figure out how to ensure switching theme is easy and make sure you have more custom buttons to apply theme colours or figure out the easy way to set theme colours for specific buttons in your html from your css

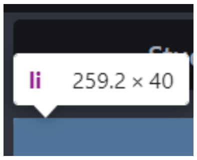
This is where I'll be doing all my themes
`node_modules\custom-themes.scss`

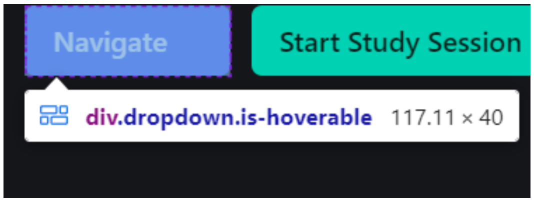
also maybe figure out how to see css changes live in your code

Sunday, Sep 1st 6:05 PM

Okay no more working on css I just need to make the app actually work and I can focus on css when it's Christmas or something

Sunday, Sep 1st 6:27 PM

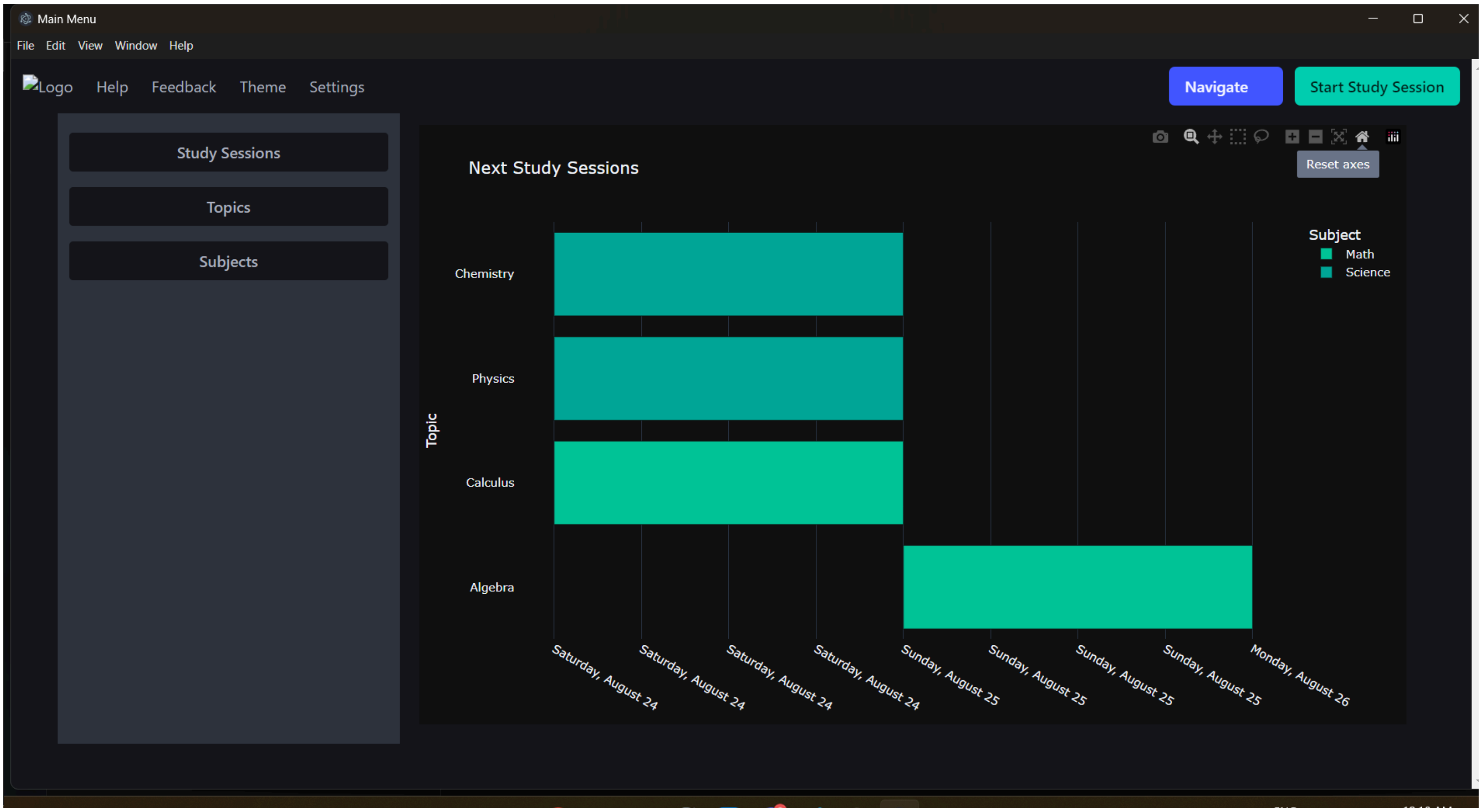




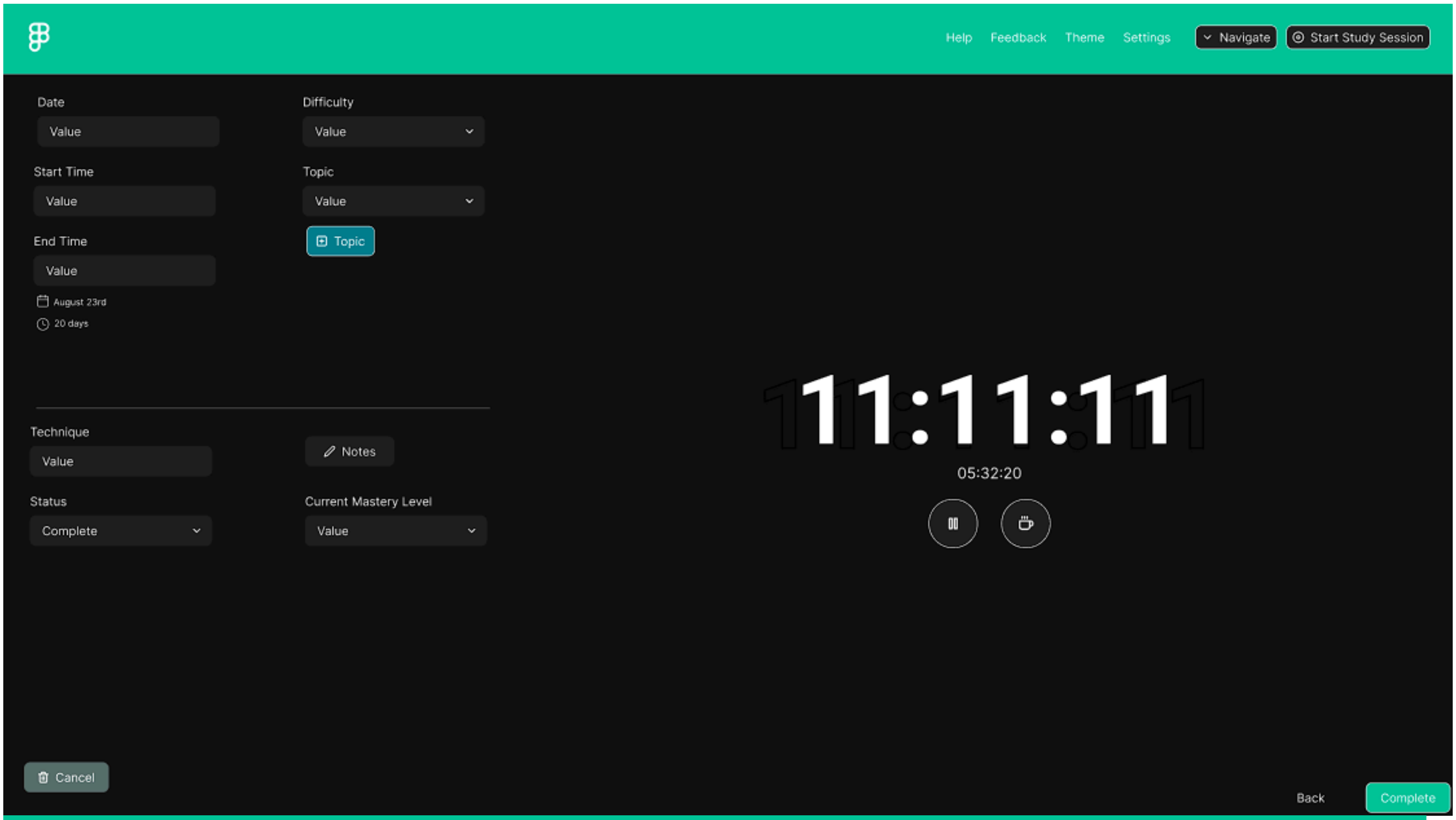
okay I guess the button is larger which actually does not matter

Monday, Sep 2nd 12:11 AM

pogress



Monday, Sep 2nd 9:53 PM



Date: Value

Difficulty: Value

Start Time: Value

Topic: Value

End Time: Value

August 23rd

20 days

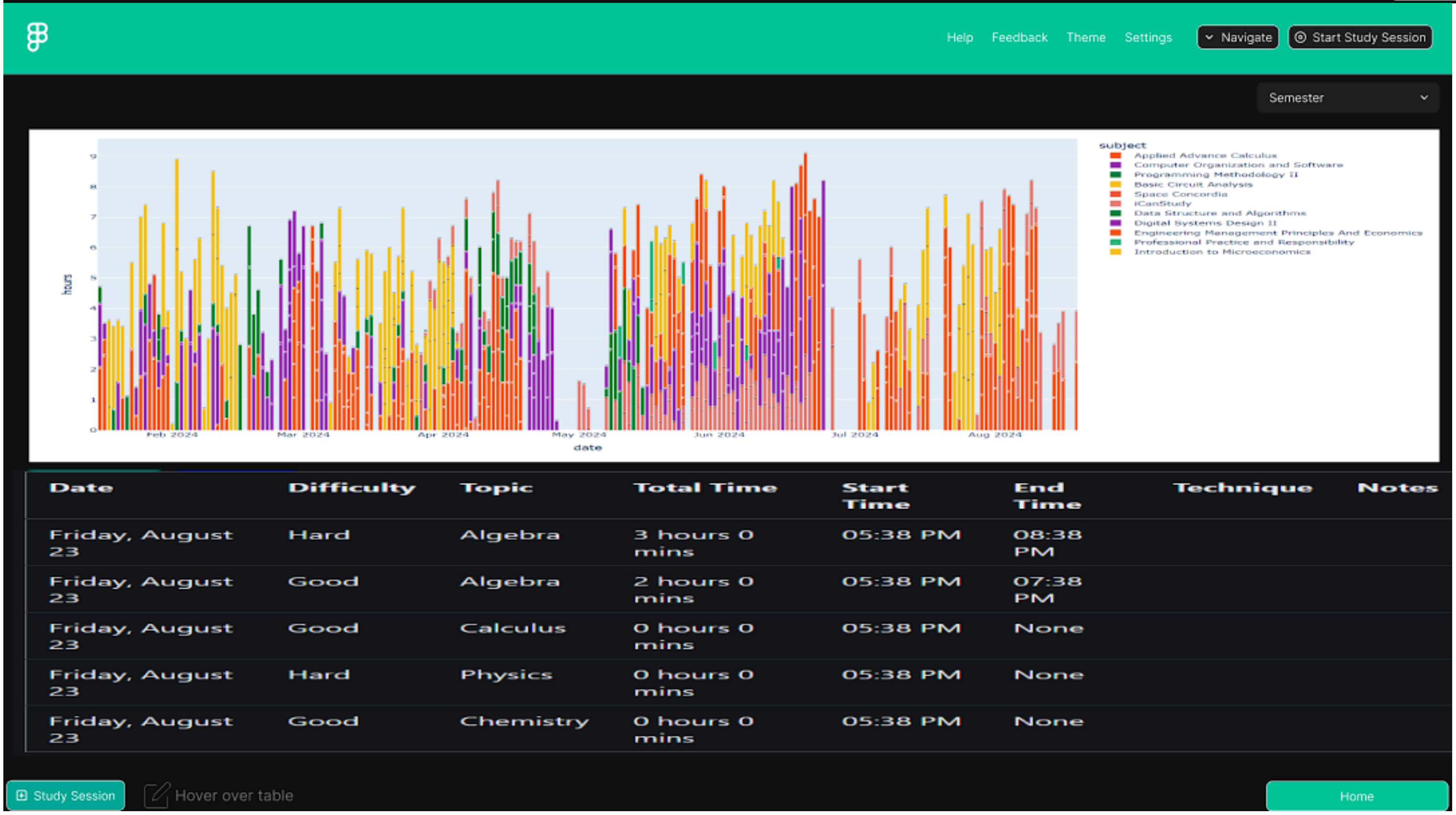
Technique: Value

Status: Complete

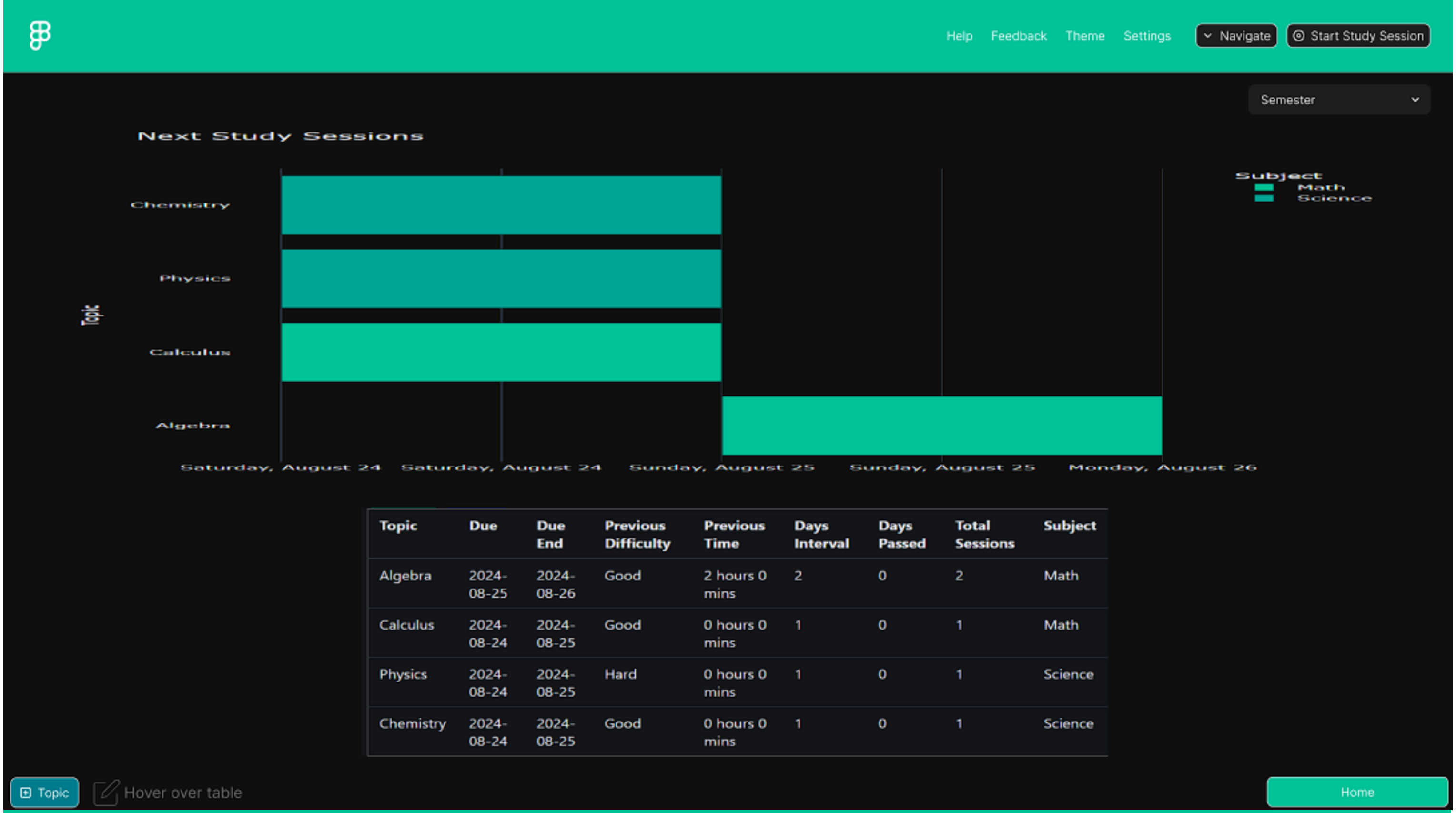
Current Mastery Level: Value

11:11:11

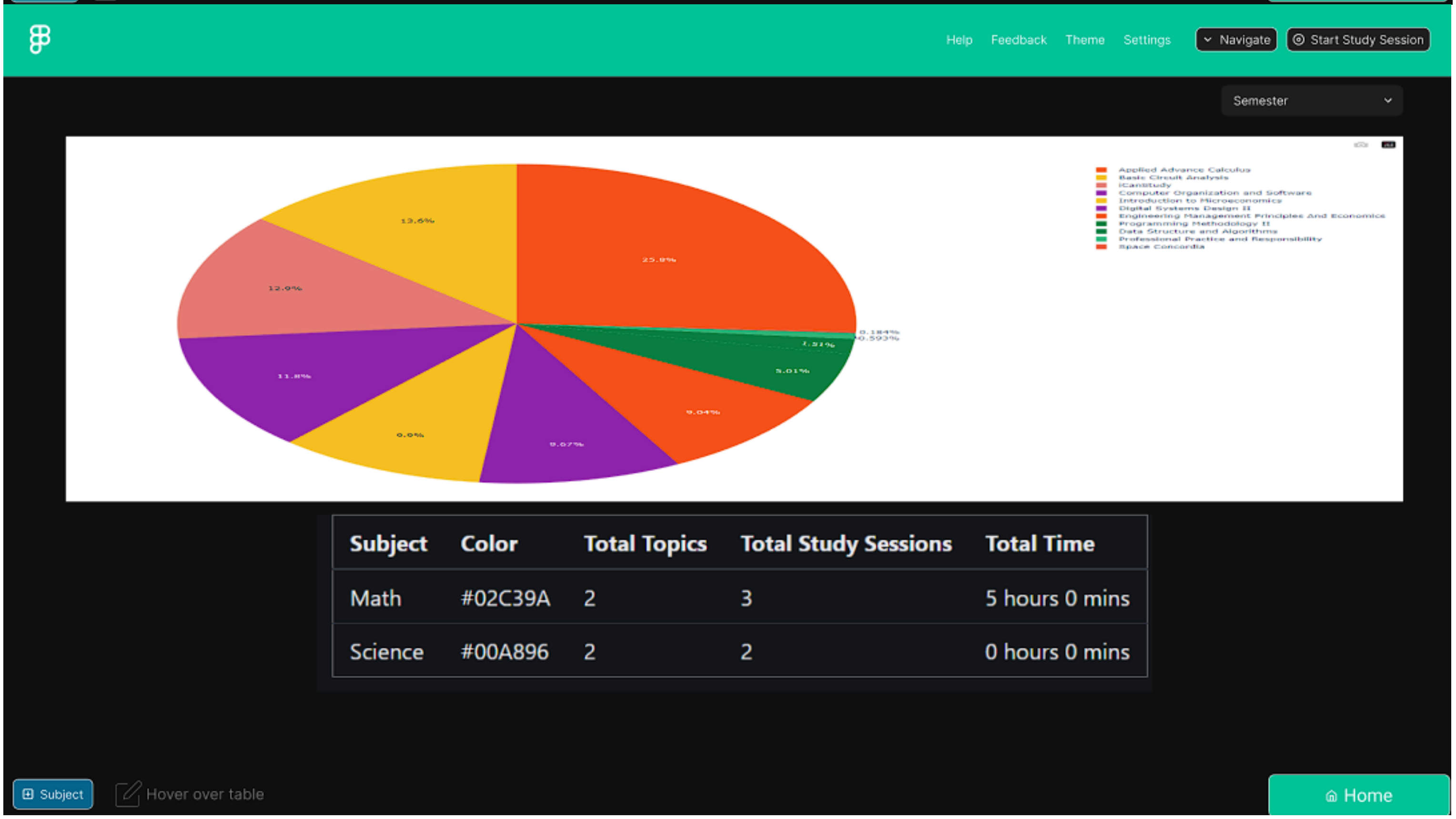
05:32:20



Date	Difficulty	Topic	Total Time	Start Time	End Time	Technique	Notes
Friday, August 23	Hard	Algebra	3 hours 0 mins	05:38 PM	08:38 PM		
Friday, August 23	Good	Algebra	2 hours 0 mins	05:38 PM	07:38 PM		
Friday, August 23	Good	Calculus	0 hours 0 mins	05:38 PM	None		
Friday, August 23	Hard	Physics	0 hours 0 mins	05:38 PM	None		
Friday, August 23	Good	Chemistry	0 hours 0 mins	05:38 PM	None		



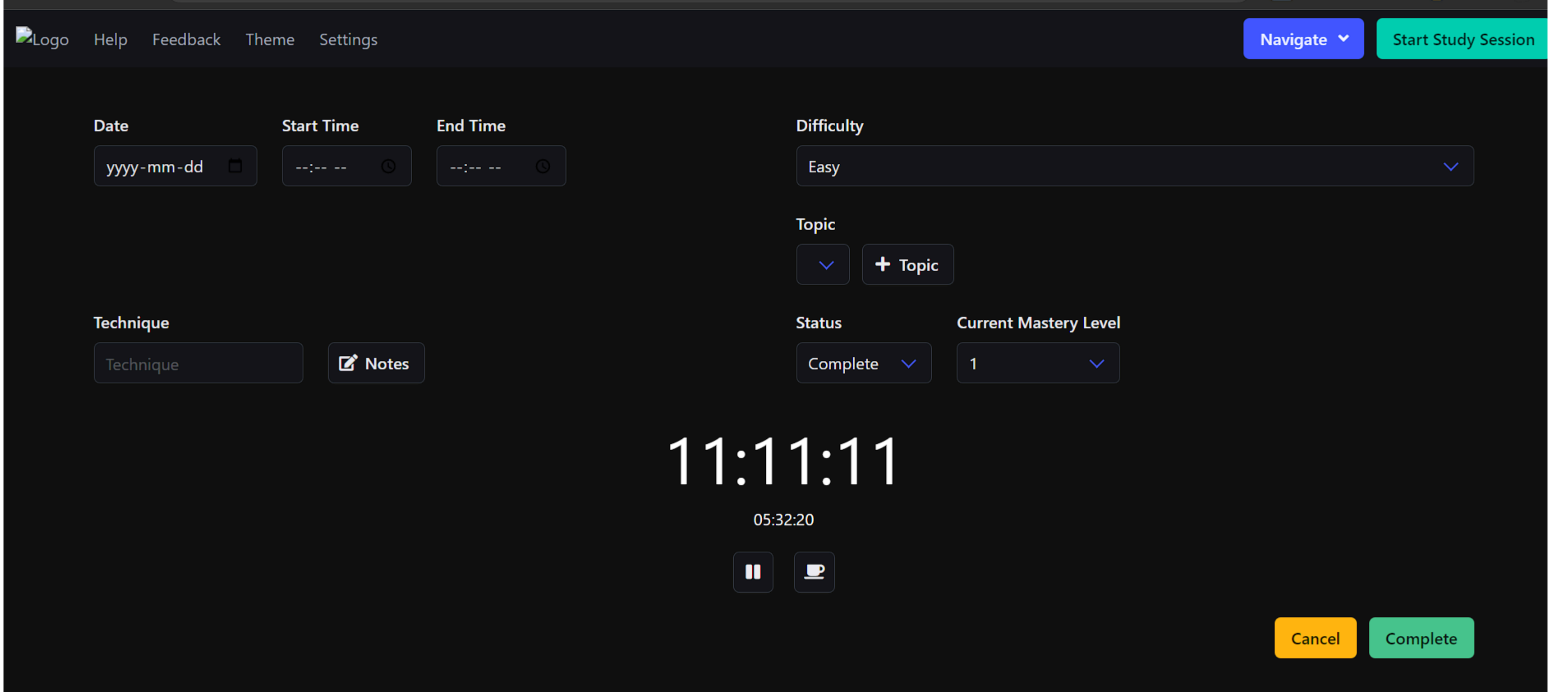
Topic	Due	Due End	Previous Difficulty	Previous Time	Days Interval	Days Passed	Total Sessions	Subject
Algebra	2024-08-25	2024-08-26	Good	2 hours 0 mins	2	0	2	Math
Calculus	2024-08-24	2024-08-25	Good	0 hours 0 mins	1	0	1	Math
Physics	2024-08-24	2024-08-25	Hard	0 hours 0 mins	1	0	1	Science
Chemistry	2024-08-24	2024-08-25	Good	0 hours 0 mins	1	0	1	Science


Help
Feedback
Theme
Settings
Navigate
Start Study Session
Semester
25.8%
13.6%
12.9%
11.8%
8.8%
9.67%
9.04%
5.01%
1.31%
0.184%
0.593%
Applied Advance Calculus
Basic Circuit Analysis
iCanStudy
Computer Organization and Software
Introduction to Microeconomics
Digital Systems Design II
Engineering Management Principles And Economics
Programming Methodology II
Data Structure and Algorithms
Professional Practice and Responsibility
Space Concordia
Subject
Hover over table
Home

Subject	Color	Total Topics	Total Study Sessions	Total Time
Math	#02C39A	2	3	5 hours 0 mins
Science	#00A896	2	2	0 hours 0 mins

Tuesday, Sep 3rd 5:27 PM

Okay done the start session although kinda scuffed idk what the html code is doing, going to recode it myself probably in Christmas



Date: yyyy-mm-dd
Start Time: --:-- --
End Time: --:-- --
Difficulty: Easy
Topic:
Technique: Technique
Status: Complete
Current Mastery Level: 1

11:11:11

05:32:20

Wednesday, Oct 23rd 9:21 AM
Alright first thing Im going to to do is recover those html files I deleted since I thought a team would work on it but nope

Tuesday, Nov 5th 2:52 PM
Alright realized how important this app is so gonna continue on it but first need to figure out what to continue on

Oh right it was makin the app work. So I need some super basic HTML with the basic buttons, and then code the Java so it interacts with my Python database. Alright so let's get the HTML pages done for

- ☐ Homepage
  - ☐ Panels
- ☐ Start Study Session
- ☐ Study Sessions
- ☐ Topics
- ☐ Subjects

Tuesday, Nov 5th 3:02 PM
The gnatt chart is gonna simply visualize the study sessions with learning efficiency. Also need to figure out how to add learning efficiency now that I figured it out. Probably like a simple star

- ☐ Need to remove current mastery level from design
- ☐ Difficulty can be 5 stars

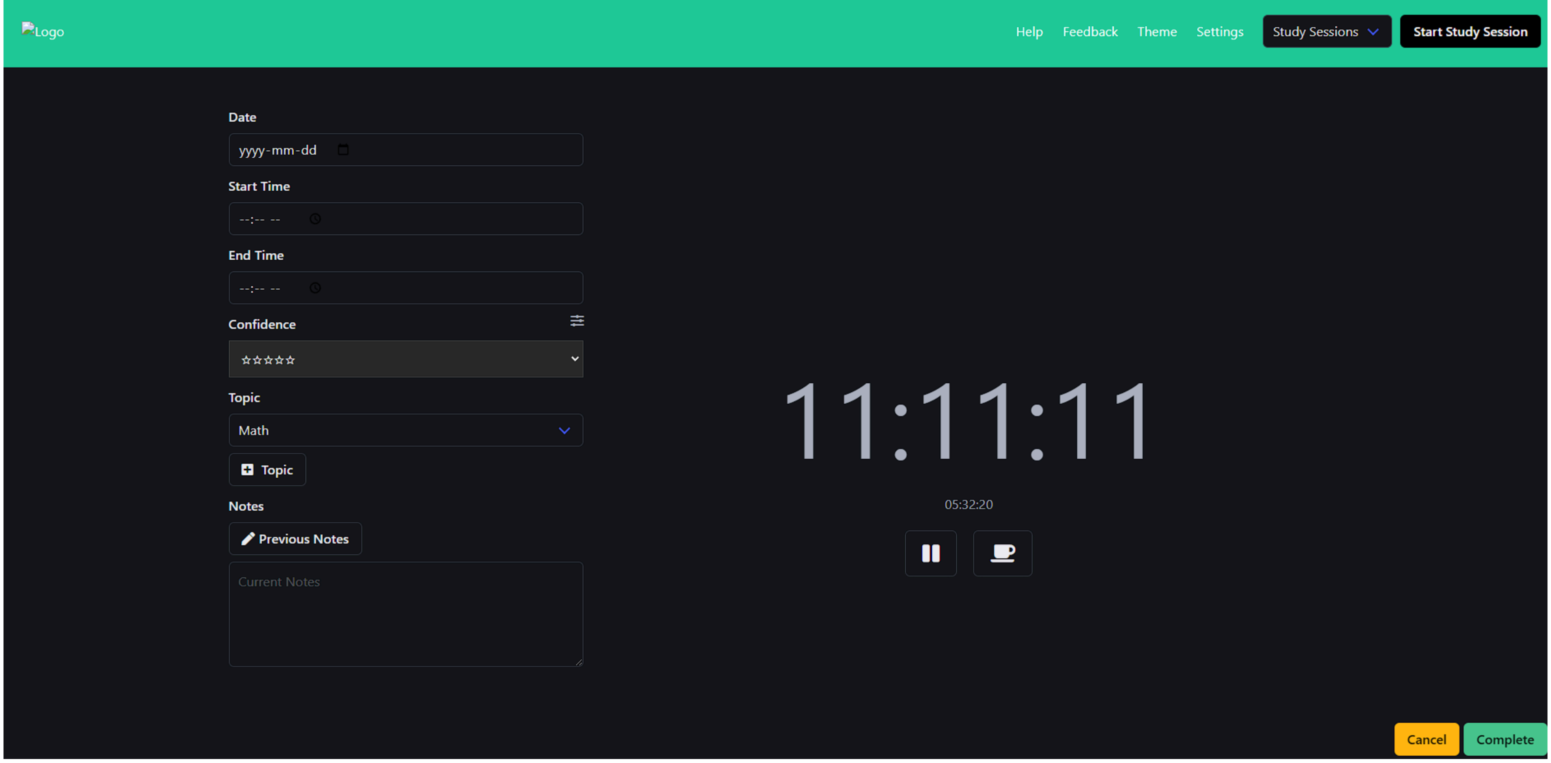
Tuesday, Nov 5th 3:48 PM
Next time continue on start study session html in this chat

Tuesday, Nov 5th 9:37 PM
Alright we are not in the future yet where we can continue to use the same chat

Tuesday, Nov 5th 10:10 PM
The thing is I shouldn't add features to the app that I personally wouldn't use. Like it should be as convenient for my current system as possible and extra features come later if people want or I want

Wednesday, Nov 6th 4:23 PM
Back at it

Wednesday, Nov 6th 4:32 PM
Okay basically done this html



Date: yyyy-mm-dd

Start Time: --:-- --

End Time: --:-- --

Confidence: ☆☆☆☆☆

Topic: Math

Notes

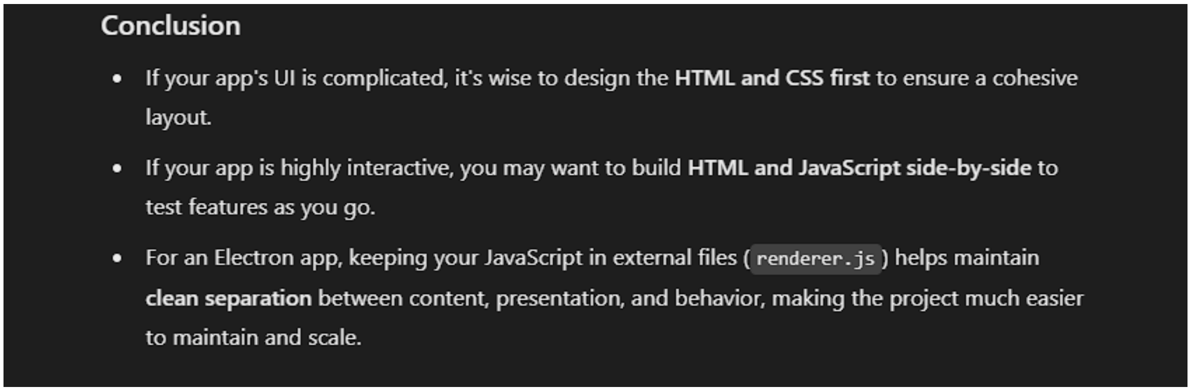
Current Notes

11:11:11

05:32:20

Wednesday, Nov 6th 4:35 PM

Alright maybe should start working on Java now to make sure the code works as expected + have a better idea on how java works



Conclusion

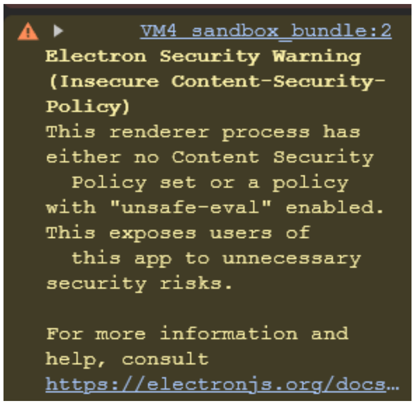
- If your app's UI is complicated, it's wise to design the **HTML and CSS first** to ensure a cohesive layout.
- If your app is highly interactive, you may want to build **HTML and JavaScript side-by-side** to test features as you go.
- For an Electron app, keeping your JavaScript in external files (`renderer.js`) helps maintain **clean separation** between content, presentation, and behavior, making the project much easier to maintain and scale.

Wednesday, Nov 6th 4:36 PM

Will first make sure that homepage can access this page

Wednesday, Nov 6th 5:32 PM

oh boy will need to fix this eventually



```
VM4 sandbox_bundle:2
Electron Security Warning
(Insecure Content-Security-
Policy)
This renderer process has
either no Content Security
  Policy set or a policy
with "unsafe-eval" enabled.
This exposes users of
  this app to unnecessary
security risks.

For more information and
help, consult
https://electronjs.org/docs…
```

Wednesday, Nov 6th 6:02 PM

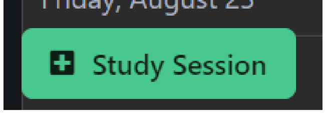
Okay got the java script stuff working. Yeah have a better idea on how it works nice. Asking the AI to explain as I code is pretty educational

Wednesday, Nov 6th 6:27 PM

Alright gonna work on the `study_sessions.html`, `topics.html`, and `subjects.html`

Wednesday, Nov 6th 6:42 PM

I am going to be using this button a lot and same for topics and subjects so should figure out how to make it a component or something reusable.



Thursday, Nov 7th 3:06 PM
Okay says I should make a javascript code

Thursday, Nov 7th 3:34 PM
Alright did that. Now need to code `add_study_session.html` which will be a pop up or something I can insert anywhere. Not sure if it should be a javascript or html. I guess html just because it's a bit more complex. Or maybe java since it'll need to be very dynamic. Hmmm, I'll figure that out next time

Thursday, Nov 7th 8:40 PM
Yeah java script seems to be the way

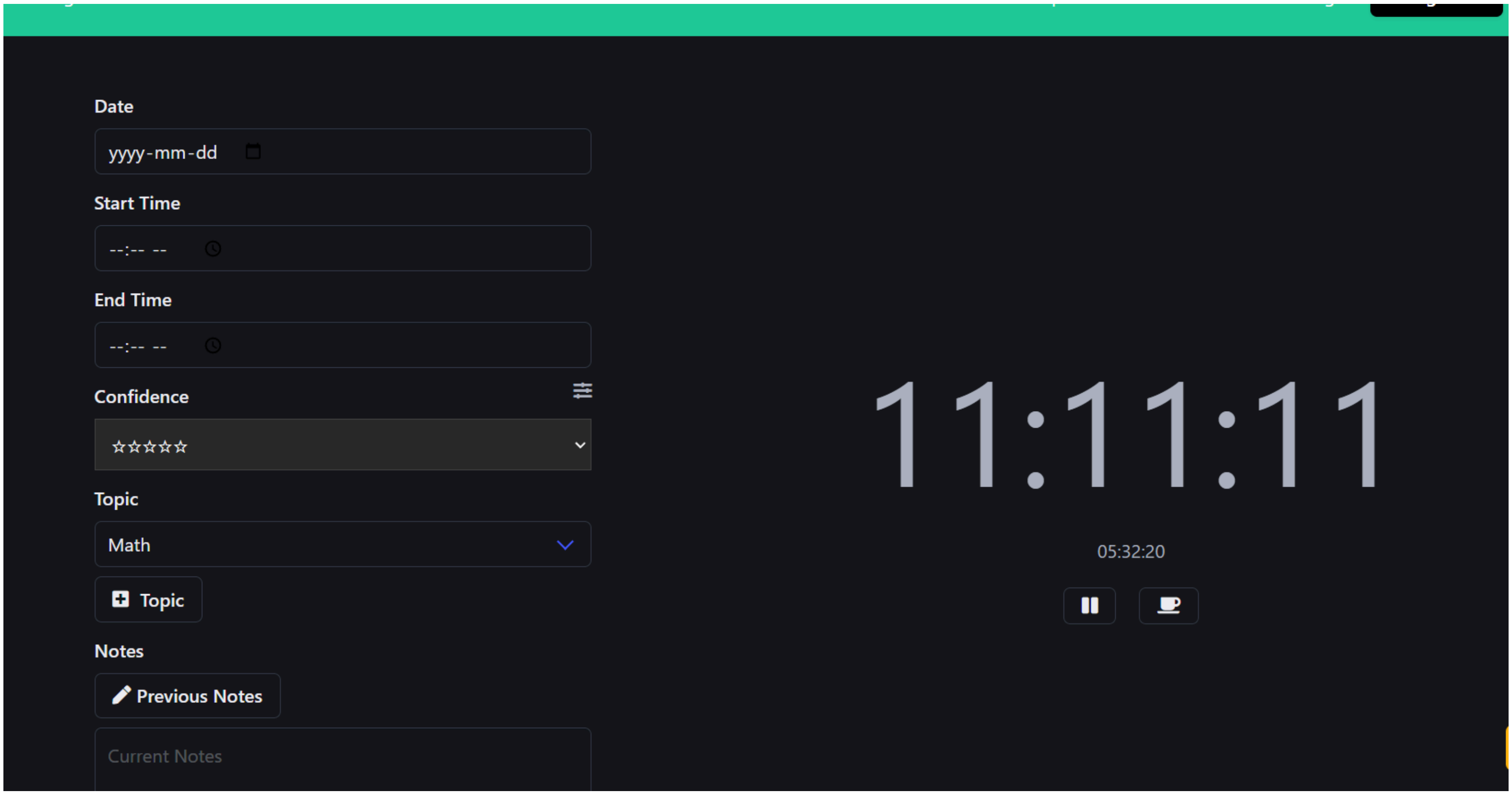
Thursday, Nov 7th 9:07 PM
For some reason my entire code is literally linked to nav bar. Everything loads after nav bar which is really confusing and not modular at all

Thursday, Nov 7th 9:10 PM
Alright I literally dont know enough about javascript to understand why importing java functions into html breaks everything. Probably should watch a YT vid of how it's supposed to be done cause AI is for sure overcomplicating or I dont see something simple

Next time search JavaScript reusable components

Friday, Nov 8th 2:58 PM
Hmm it seems templates are good maybe https://youtu.be/qC5xK6H0GlQ?t=86

Friday, Nov 8th 4:11 PM
Trying to replace this with the template here



Date
yyyy-mm-dd

Start Time
--:-- --

End Time
--:-- --

Confidence
☆☆☆☆☆

Topic
Math

and also probably should upload this code to GitHub for version control cause realized my version control is literally ChatGPT4o and Ctrl Z

Friday, Nov 8th 4:13 PM
Also realized that I cant rely on ChatGPT4o forever since eventually my code will be too big for it too handle so I need to understand why my code doesnt work so I can editr i

Saturday, Nov 9th 1:50 PM
Joe HAS just SEEN the CODE. He HAS critized SOME POINTS
FOr eXAMPLE the very very long html harded coded code spaghetti. That must fixed....

Sunday, Nov 10th 10:39 PM
These are the concepts you should learn I believe

```
  - HTTP Request Handling (`HttpServletRequest`, `HttpServletResponse`)
  - HTTP GET and POST Requests (`doGet`, `doPost`)
  - Page Redirection (`sendRedirect`)
  - Session Management (`HttpSession`)
  - Storing and Retrieving Session Data (`setAttribute`, `getAttribute`)
  - Form Inputs (`<input>`, `<select>`, `<textarea>`)
  - Form Submission Configuration (`action`, `method`)
  - Dynamic Content Loading (`<div>`, `fetch`)
  - Event Handling (`addEventListener`)
  - DOM Content Loading Event (`DOMContentLoaded`)
  - DOM Manipulation (`document.querySelector`, `document.getElementById`, `innerHTML`)
  - Fetch API for Asynchronous Resource Loading (`fetch`)
  - Dynamic UI Component Creation (`createButton`, `createForm`)
  - Modal Window Management (`openModal`, `closeModal`)
  - Responsive Design Frameworks (`Bulma CSS`)
  - Icon Integration (`Font Awesome`)
  - Client-Side Navigation (`window.location.href`)
  - Functions (`public`, `void`, `return`)
  - Objects and Classes (`class`, `new`, `this`)
  - Constructor for Object Initialization (`className()`)
  - Listener Interfaces (`ActionListener`, `MouseListener`)
  - Event Handling Methods (`actionPerformed`, `mouseClicked`)
  - Method Overloading and Reusability (`createStudySessionButton`, `createAddStudySessionForm`)
  - Encapsulation for Reusable Components (`private`, `public`, `getters`, `setters`)
```

Friday, Dec 20th 1:47 PM
Okay so now gonna prestudy, and it seems the concepts above are related to not just one language, so it would be confusing to learn multiple languages at once. According to AI and thirty, I should mostly focus on JavaScript. It will also work well for the back end. So gonna learn these concepts and collect my gaps in 💡 JavaScript

Saturday, Dec 21st 5:03 PM
Okay will do a more focused course that's more for my specific need, which will be this one

But for that I need to know 💡 HTML and 💡 CSS so will start learning those two quickly here

Sunday, Dec 22nd 3:29 PM
Okay done 💡 HTML, now on 💡 CSS

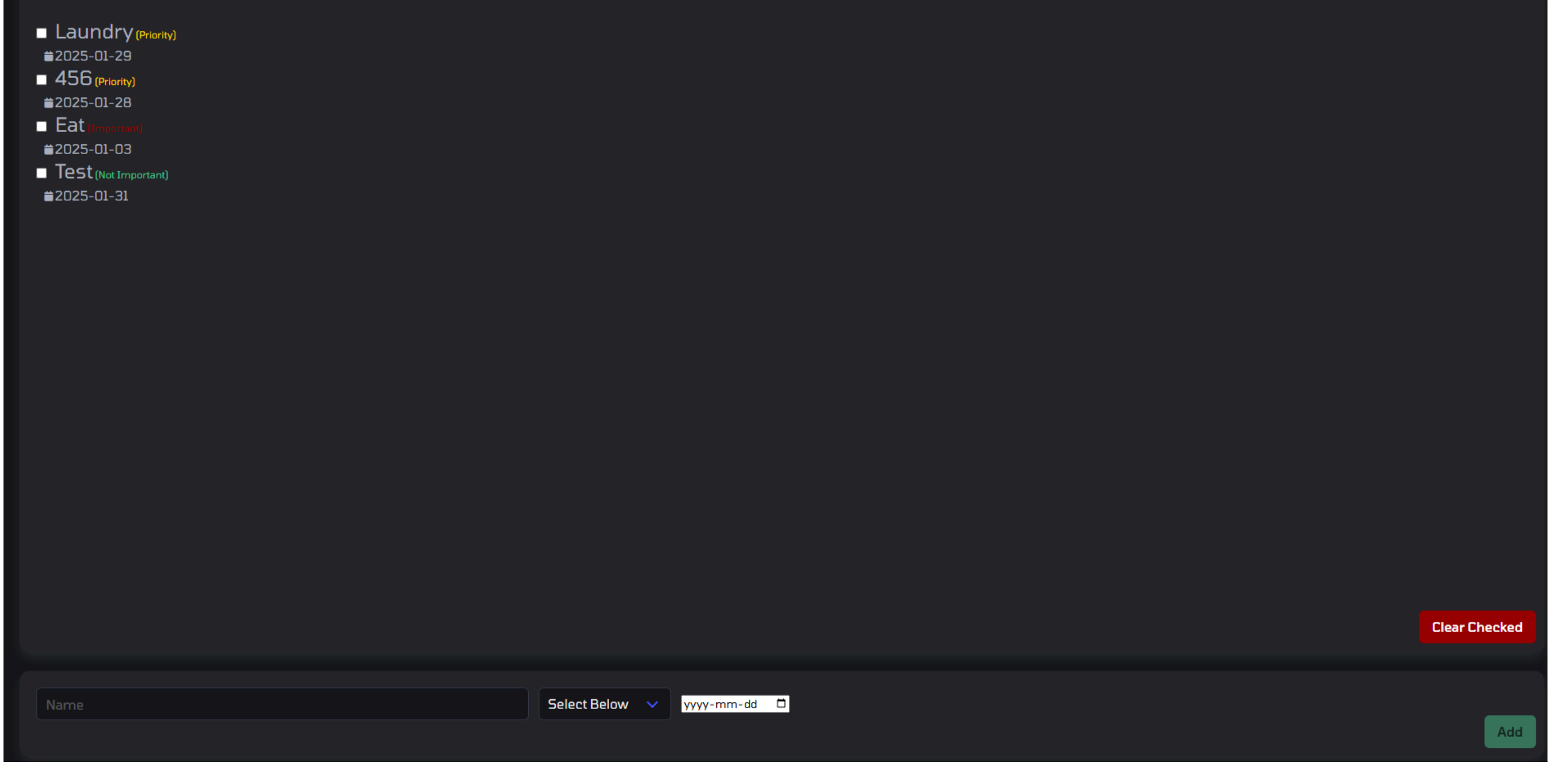
Monday, Dec 23rd 1:48 PM
Okay done the course, here's the certificate

Saturday, Dec 28th 11:09 AM
Continuing on learning DOM, will watch one more vid then will practice DOM before continuing in 💡 JavaScript

Sunday, Dec 29th 12:29 PM
Yesterday made cat website after watching DOM vids, now will finish rest quickly which are on JavaScript basics

Monday, Dec 30th 4:40 PM
Completed another small project which is the random number game

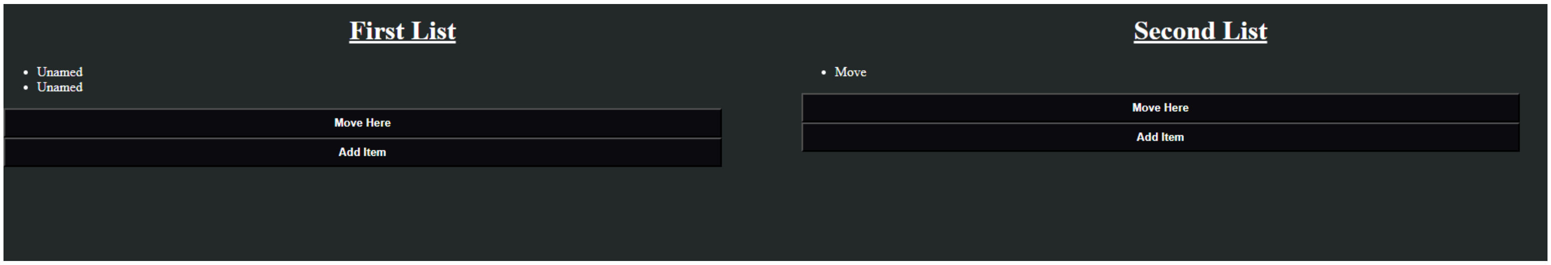
Friday, Jan 3rd 3:51 PM
Made checklist app



☐ Laundry (Priority)
2025-01-29
☐ 456 (Priority)
2025-01-28
☐ Eat (Important)
2025-01-03
☐ Test (Not Important)
2025-01-31

Saturday, Jan 4th 1:47 PM

Another mini-project



First List

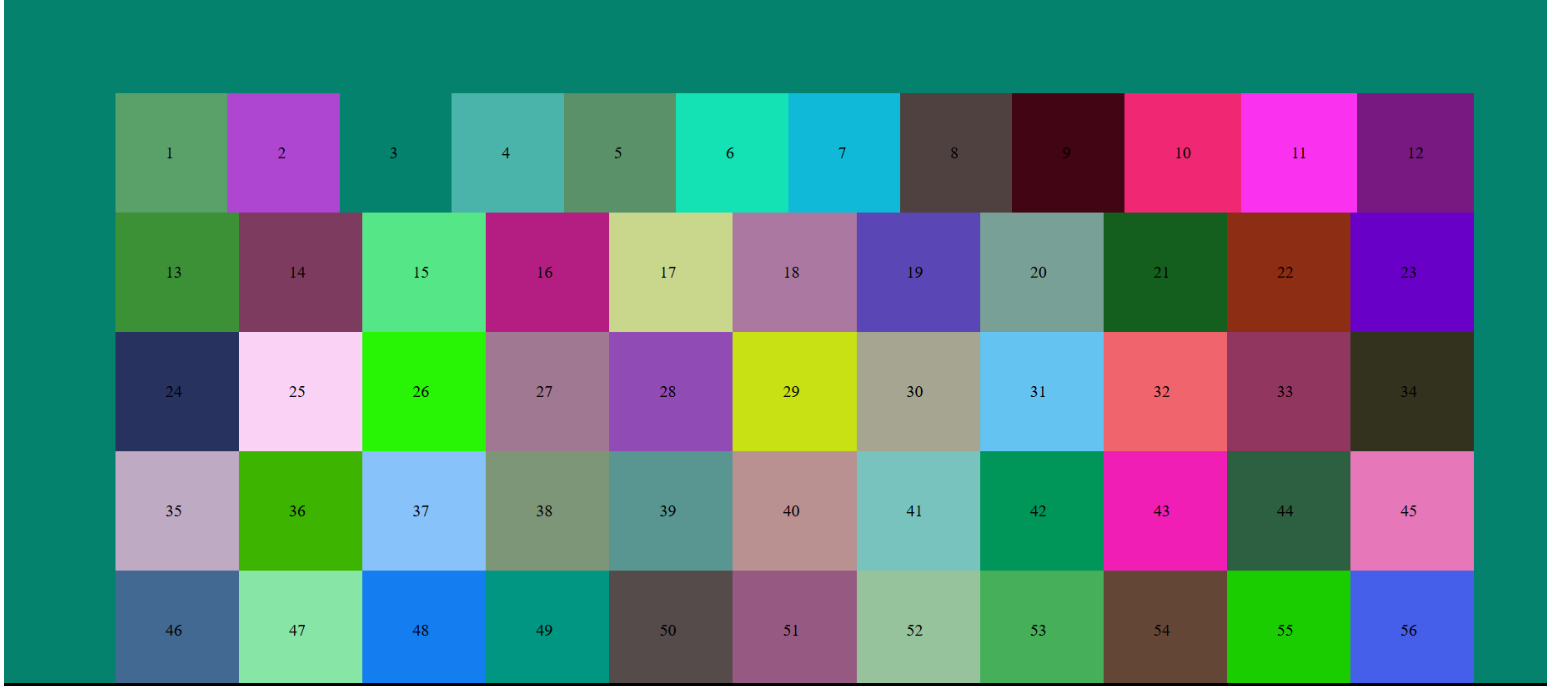
- Unamed
- Unamed

Second List

- Move

Sunday, Jan 5th 3:43 PM

Another mini-project for keypress and mouse hovers



Monday, Jan 6th 3:32 PM

After speaking to 😛 Joe, I need to do these tasks according to this Miro board we have created

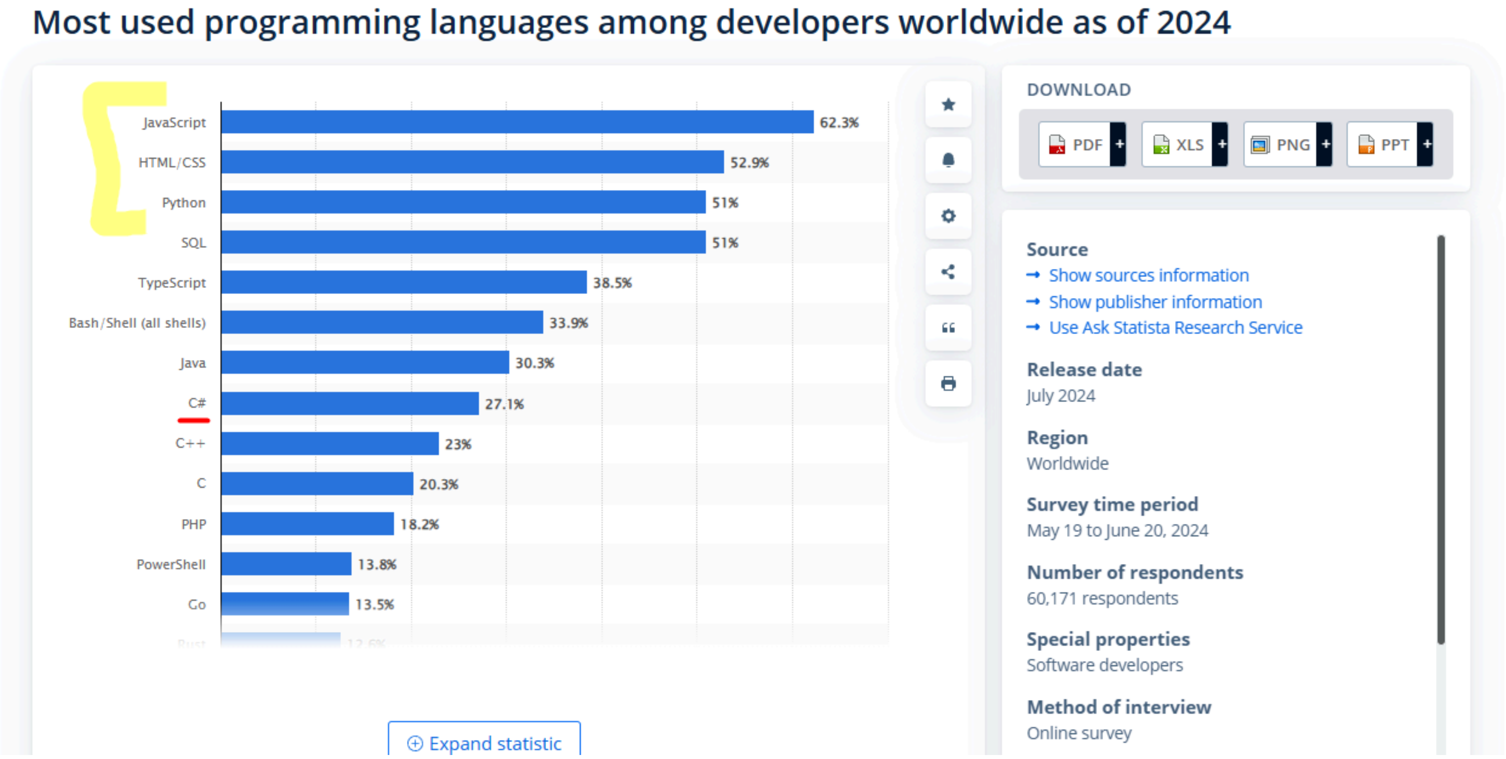
- ☑ ~~Complete certificate~~
- ☑ ~~Remove FSRS from Python backend~~
  - ☑ ~~Create new branch~~
- ☑ ~~Remove useless features~~
- ☒ Learn C# basics in Udemy (objects)
- ☒ Learn C# LINQ
- ☒ Learn SQL
- ☒ Learn ASP.NET to make an API
- ☒ Learn Entity Framework Core to interact with SQL
- ☒ Learn Postman to test API
- ☐ Complete React certificate
- ☐ Complete basic UI to prepare for feedback / tweaks
- ☐ Add themes feature
- ☐ Complete certificate to learn MEN of MERN, knowing how to make a RESTful API
  - ☐ Learn Node.js (learn JavaScript more indepth)
  - ☐ Learn Express.js as a framework to help make an API

    - ☐ Learn MongoDB for a database
    - ☐ Learn Postman for testing
- ☐ Rework backend
    - ☐ Create an API for database
    - ☐ Delete Python backend
    - ☐ Add backup feature

Tuesday, Jan 7th 12:36 PM
Okay the rest of the certificate are projects but since I already did mini-projects as going through the basics, I'll assume the certificate done and move on to working on my project. And as I work on my project I'll be applying JavaScript so yeah

Tuesday, Jan 7th 1:29 PM
Okay now I gotta remove the FSRS from the Python backend, then from there will figure out how to use C# / optimize the backend properly.

Tuesday, Jan 7th 1:48 PM
Yeah gonna just start removing the FSRS algorithm although it seems that C# is important too. But will make sure I get good at JavaScript, HTML, CSS, and Python. And even SQL. And those seem like the most popular languages these days


Most used programming languages among developers worldwide as of 2024

Source
→ Show sources information
→ Show publisher information
→ Use Ask Statista Research Service

Release date
July 2024

Region
Worldwide

Survey time period
May 19 to June 20, 2024

Number of respondents
60,171 respondents

Special properties
Software developers

Method of interview
Online survey

Tuesday, Jan 7th 4:14 PM
Nvm been told by 😜 Joe to learn C#. It is worth the time

Tuesday, Jan 7th 6:10 PM
Cleaned up the backend. Maybe another review soon, but now need to learn C# in order to convert backend into C#. So tomorrow will learn C#

Wednesday, Jan 8th 12:18 PM
Starting 💡 Csharpcertificate

Wednesday, Jan 8th 2:06 PM
Seems I need to learn how to make APIs

Wednesday, Jan 8th 2:54 PM
Okay this is the learning path so idk why I learnt JavaScript cause I won't be using it for probably like over 3 months. Should still test myself by doing small projects once a while but probably will just forget it all

> **ChatGPT4o**
>
> The order of learning in your plan is **mostly logical**, but it can be slightly tweaked to optimize your learning curve and reduce confusion. Here's the adjusted order with explanations for why each step fits in that sequence:
>
> ---
>
> ## Adjusted Order of Learning

1. Learn C# Basics (Objects, Classes, OOP)

- **Why First**:
  - C# is the foundation for everything else in your plan. Understanding its syntax, classes, and object-oriented programming (OOP) principles is critical before diving into frameworks like ASP.NET or Entity Framework Core.
  - Focus on:
    - Classes, methods, and properties.
    - Inheritance and interfaces.
    - Exception handling.
    - LINQ (for querying collections, which prepares you for EF Core).
- **Example Topics to Learn**:

```
public class StudySession
{
    public DateTime Date { get; set; }
    public int Difficulty { get; set; }

    public void DisplayDetails()
    {
        Console.WriteLine($"Date: {Date}, Difficulty: {Difficulty}");
    }
}
```

---

2. Learn SQL

- **Why Second**:
  - SQL is the backbone of relational databases, and understanding it early helps you make sense of how the database works before you integrate it with EF Core.
  - Learn the basics:
    - **CREATE**, **SELECT**, **INSERT**, **UPDATE**, **DELETE**.
    - Relationships (foreign keys, joins).
    - Simple queries and filtering.
- **How It Helps Later**:
  - When you start working with EF Core, understanding SQL makes it easier to debug queries and migrations.

---

3. Learn ASP.NET Core to Make an API

- **Why Third**:
  - After mastering C# basics, learning ASP.NET Core lets you start building APIs.
  - By this point, you'll know enough about C# to understand how ASP.NET Core organizes code (controllers, routing, middleware).
  - Focus on:
    - Setting up controllers and routing.
    - Handling HTTP methods (GET, POST, PUT, DELETE).
    - Structuring endpoints.
- **How It Helps Later**:
  - You'll have a basic API ready to connect with a database once EF Core is introduced.

---

4. Learn Entity Framework Core to Interact with SQL

- **Why Fourth**:
  - EF Core simplifies database operations, but to use it effectively, you need prior knowledge of SQL and ASP.NET Core.
  - Learn:
    - How to create models and map them to database tables.
    - Basic CRUD operations (Create, Read, Update, Delete).
    - Migrations to manage database schema changes.
- **How It Fits**:

    - Once you have a functioning API, you'll enhance it with database interactions using EF Core.

---

5. Learn Postman to Test the API

- **Why Fifth**:
  - By now, your API will have some endpoints that interact with the database. Postman helps you test these endpoints without needing a frontend.
  - Learn:
    - Sending GET, POST, PUT, and DELETE requests.
    - Adding headers and query parameters.
    - Testing API responses (e.g., JSON data).

---

6. Recode Backend Using C# and Maybe a Backend Framework

- **Why Last**:
  - After completing all the steps, you'll have the knowledge to refine and optimize your backend.
  - At this point, you can:
    - Rebuild the backend to fix early mistakes or improve the design.
    - Add advanced features like authentication, logging, and security.

Why This Order Works

1. **C# Basics First**:
   - You can't work with ASP.NET Core or EF Core without understanding C# syntax and OOP concepts.
2. **SQL Early**:
   - Database knowledge is foundational. Even though EF Core simplifies database operations, you'll debug and write custom queries more effectively if you know SQL.
3. **API Before EF Core**:
   - Learn how to build APIs with static or mock data before introducing EF Core. This way, you won't struggle with multiple new concepts (API + EF Core) simultaneously.
4. **Postman Late**:
   - Postman is a tool to test APIs. It's practical once you have API endpoints to test, making it better suited after you've built basic APIs with database integration.
5. **Recode Backend Last**:
   - Refactoring your backend after learning everything allows you to implement best practices you didn't know at the start.

Optional Tweaks

If your project is **urgent** or you don't have time for all these steps:

- Skip **EF Core** initially and use raw SQL in the API for database interaction. You can introduce EF Core later for cleaner code.
- Learn **Postman** earlier if you want to test APIs without database integration.

Conclusion

Your plan (with slight adjustments) is well-structured for a learner who wants to build a robust backend for an offline app and gain skills in modern backend development. The adjusted order ensures you build a solid foundation first (C# and SQL) before tackling advanced tools like EF Core and ASP.NET Core. This progression minimizes confusion and maximizes efficiency in learning.

Wednesday, Jan 8th 4:06 PM
Should learn about SwaggerHub

Wednesday, Jan 8th 4:36 PM

Just discovered about MERN and my entire life is so much simpler

Thursday, Jan 9th 12:44 PM

Okay probably gonna start with React cause I know my data base isn't that complex. If I learn React, then I can layout the program look and feel, all while getting feedback from students. Can adapt app UI to make sense to my current understanding of learning science / student preferences. It'll also be better for jobs since I can show my work visually, and also is popular in the market. Also working on visuals is more fun, especially after I just learnt about JavaScript

Thursday, Jan 9th 12:55 PM

Okay now starting on 💡 React

Monday, Jan 13th 12:31 PM

Gonna try to find time every once in awhile to work on Udemy, maybe treating it like a SIR