# Tax and Payment Tracking System

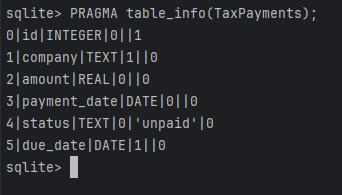
## Table of Contents

1. Project Overview  
2. SQL Schema and Table Design  
3. UI Design  
4. Backend Code  
5. Tax Breakdown Feature  
6. Documentation of CRUD Operations  
7. GitHub Repository  
8. YouTube Video

## Project Overview

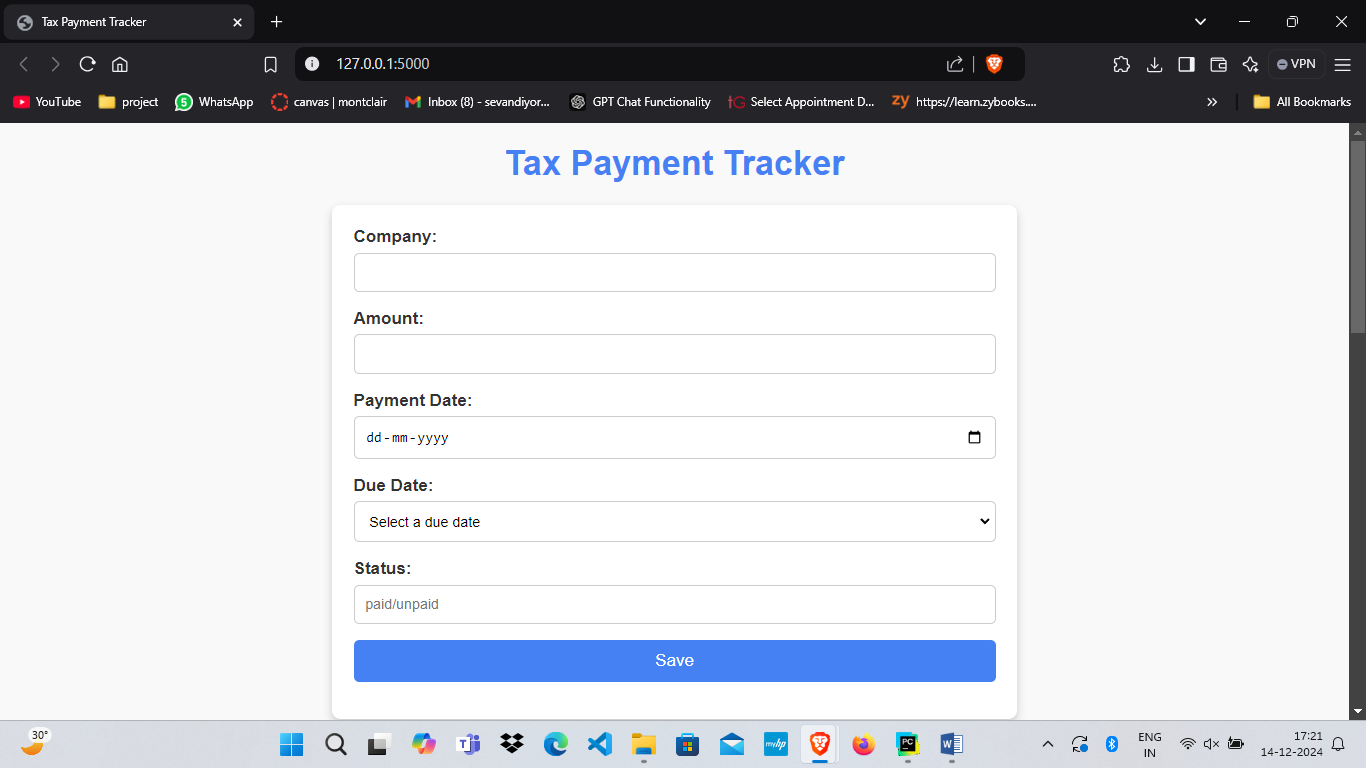
Objective: To build a tax and payment tracking system for small companies such as LLCs, C Corporations, or S Corporations to simplify tax tracking and calculation processes.  
  
Scope:  
- CRUD operations for tax records.  
- Dynamic filtering for tax breakdown by due dates.  
- Tax calculation based on a user-provided rate.  
- User-friendly UI with dropdowns and confirmation popups.  
  
Tech Stack:  
- Backend: Python Flask  
- Database: SQLite  
- Frontend: HTML, CSS, JavaScript

## SQL Schema and Table Design

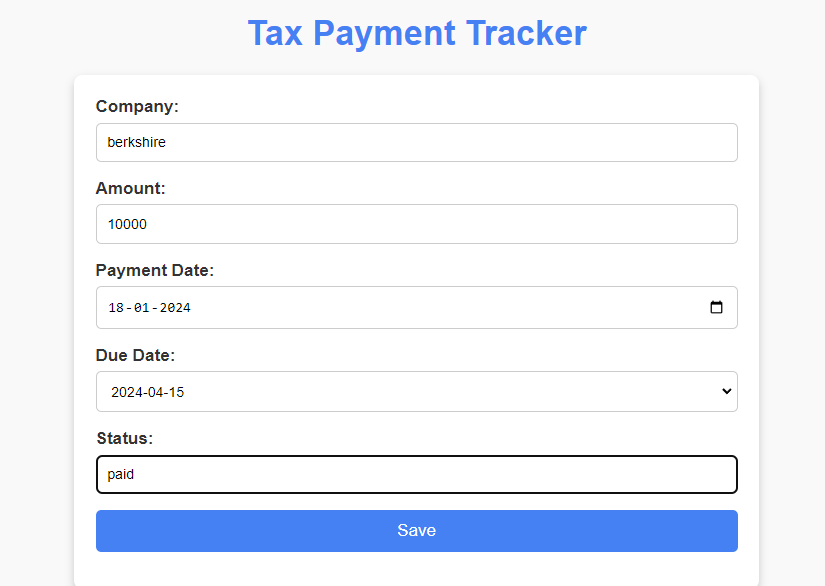
Database Table Structure:  
  
  
Primary Key: ID (unique for each record).  
Indexes: Created on `Company` and `Due Date` for faster querying.  
Constraints:  
- `Amount` must be greater than 0.  
- `Due Date` is mandatory.

## UI Design

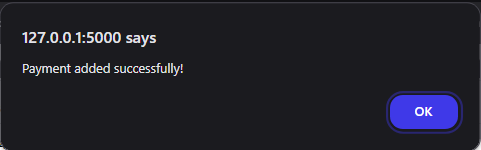
Screenshots and Features:  
1. HTML Form:  
 - Fields for `Company`, `Amount`, `Payment Date`, `Status`, and `Due Date`.  
 - A dropdown for `Due Date` dynamically generated based on the current year.



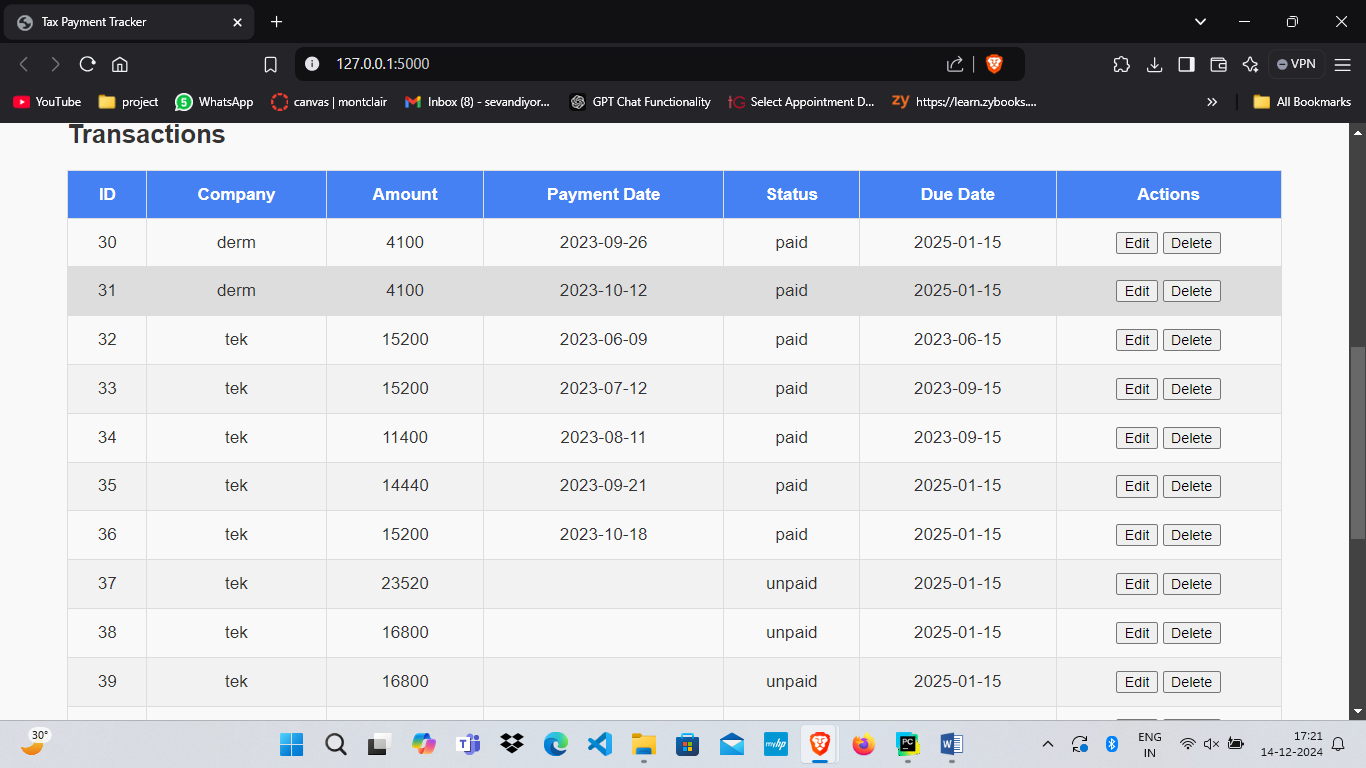
* Add new record for company name



* Record added successfully



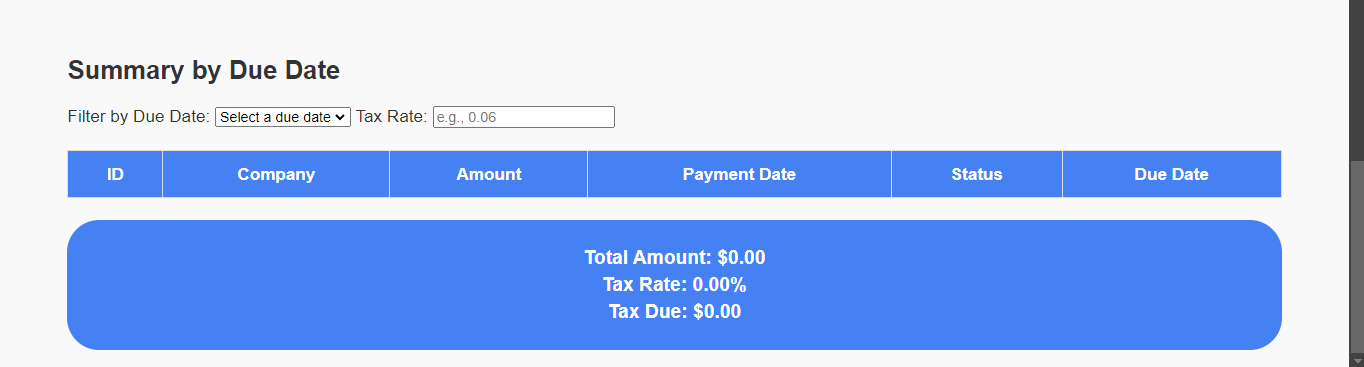
2. Save Button: Saves data and refreshes the page.



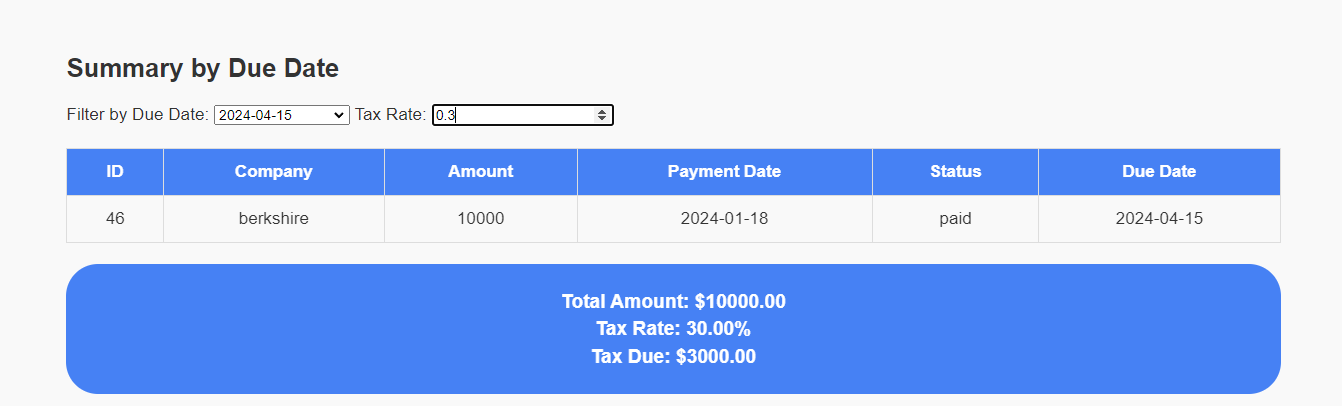
* Record is added to the database at the end with unique ID

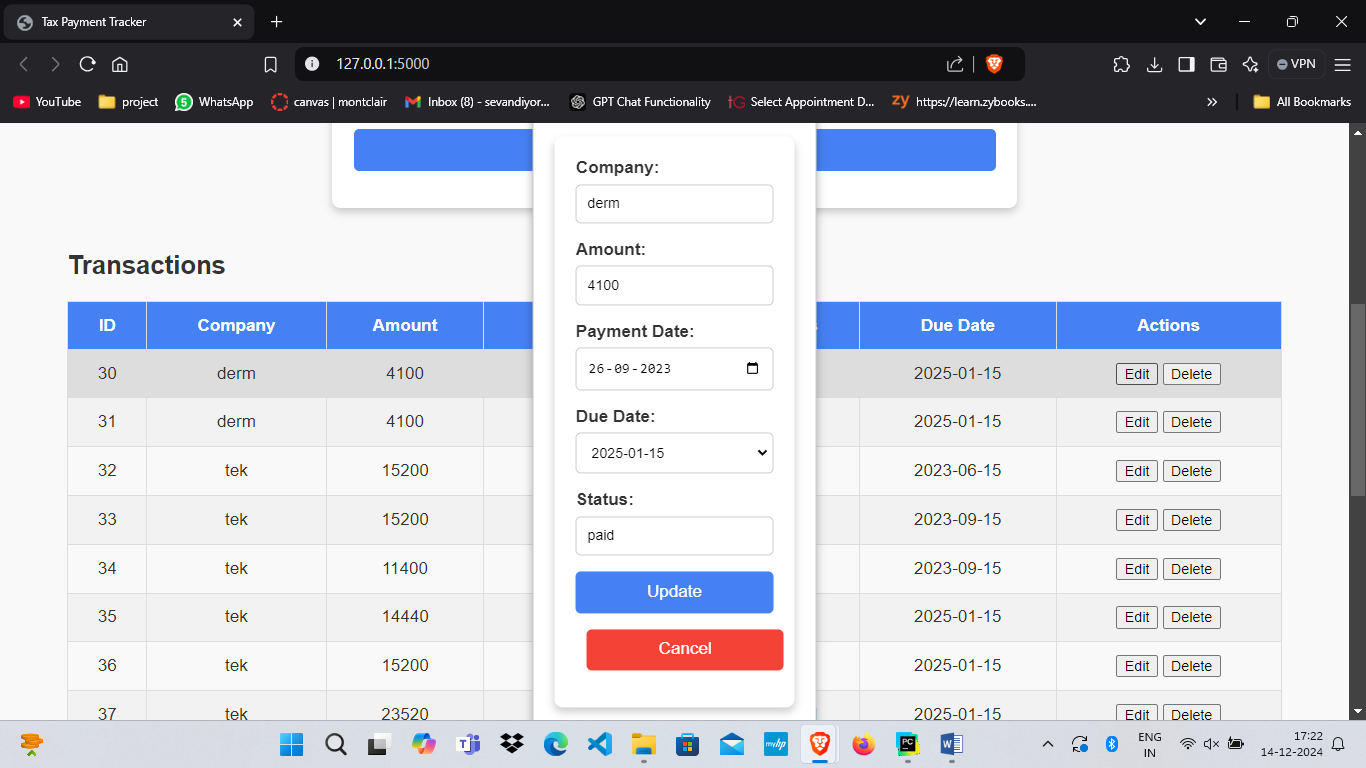


3. Dynamic Summary Table: Displays total amounts, tax rates, and tax dues based on selected filters.

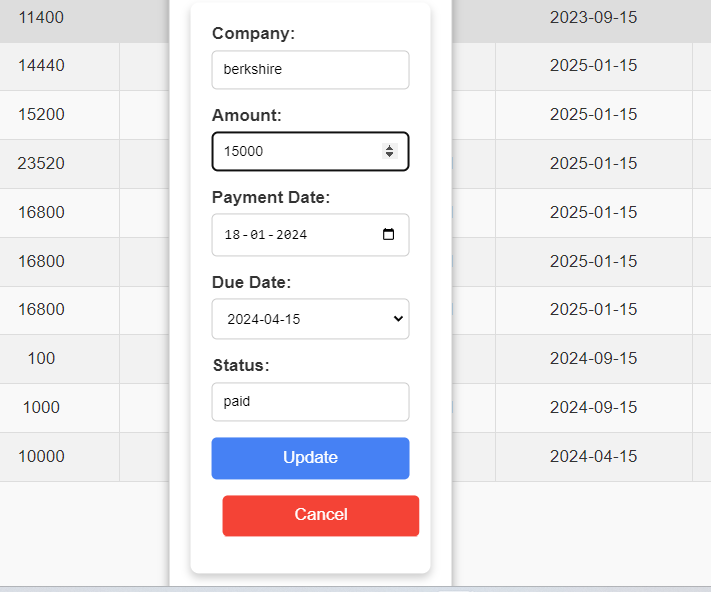


* Select due date to see the records and set the tax rate accordingly to see the tax due

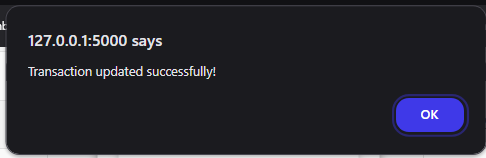
  
  
4. Popups: For confirmation of updates and deletions.



* Updating the amount of Berkshire company

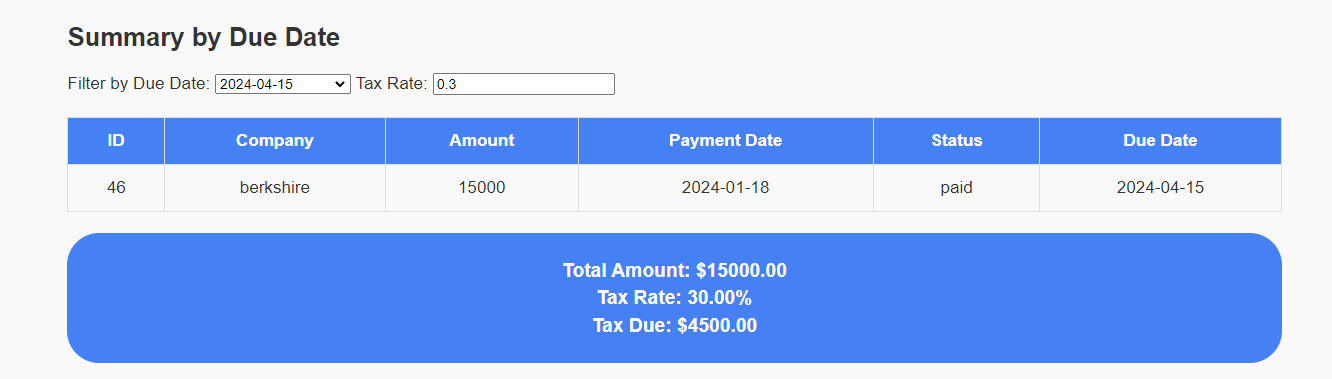


* Update successfully





* Amount is Dynamically changed in summary section and returned tax rate according to updated amount.



## Backend Code

Endpoints:  
- `POST /add`: Adds a record to the database.

*# Route to add a new transaction in database*@app.route('/payments', methods=['POST'])  
def add\_payment():  
 data = request.json  
 if not data.get('due\_date'): *# Validate due\_date* return jsonify({"error": "Due date is required!"}), 400  
  
 conn = connect\_db()  
 cursor = conn.cursor()  
 try:  
 cursor.execute("""  
 INSERT INTO TaxPayments (company, amount, payment\_date, status, due\_date)  
 VALUES (?, ?, ?, ?, ?)""",  
 (data['company'], data['amount'], data.get('payment\_date'), data['status'], data['due\_date'])  
 )  
 conn.commit()  
 return jsonify({"message": "Payment added successfully!"}), 201  
 except Exception as e:  
 return jsonify({"error": str(e)}), 400  
 finally:  
 conn.close()

- `GET /view`: Fetches all records.

*# Route to fetch all transactions from a database*@app.route('/payments', methods=['GET'])  
def get\_payments():  
 conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM TaxPayments")  
 rows = cursor.fetchall()  
 conn.close()  
 return jsonify(rows)

- `PUT /update`: Updates a record.

*# Update an existing payment in database*@app.route('/payments/<int:id>', methods=['PUT'])  
def update\_payment(id):  
 data = request.json  
 if not data.get('due\_date'): *# Validate due\_date* return jsonify({"error": "Due date is required!"}), 400  
  
 conn = connect\_db()  
 cursor = conn.cursor()  
 try:  
 cursor.execute("""  
 UPDATE TaxPayments  
 SET company = ?, amount = ?, payment\_date = ?, status = ?, due\_date = ?  
 WHERE id = ?""",  
 (data['company'], data['amount'], data.get('payment\_date'), data['status'], data['due\_date'], id)  
 )  
 conn.commit()  
 return jsonify({"message": "Payment updated successfully!"}), 200  
 except Exception as e:  
 return jsonify({"error": str(e)}), 400  
 finally:  
 conn.close()

- `DELETE /delete`: Deletes a record.

*# Route to delete a transaction from database*@app.route('/payments/<int:id>', methods=['DELETE'])  
def delete\_payment(id):  
 conn = connect\_db()  
 cursor = conn.cursor()  
 try:  
 cursor.execute("DELETE FROM TaxPayments WHERE id = ?", (id,))  
 conn.commit()  
 return jsonify({"message": "Payment deleted successfully!"}), 200  
 except Exception as e:  
 return jsonify({"error": str(e)}), 400  
 finally:  
 conn.close()

- `GET /summary`: Get the summary data from the database.

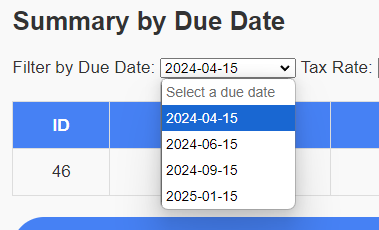
*# Route to get summary section data*@app.route('/payments/summary', methods=['GET'])  
def get\_summary():  
 due\_date = request.args.get('due\_date') *# Get the due date* tax\_rate = float(request.args.get('tax\_rate', 0.0)) *# Tax rate provided by the user* conn = connect\_db()  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM TaxPayments WHERE due\_date = ?", (due\_date,))  
 rows = cursor.fetchall()  
  
 total\_amount = sum(row[2] for row in rows)  
 tax\_due = total\_amount \* tax\_rate  
  
 conn.close()  
 return jsonify({  
 "records": rows,  
 "total\_amount": total\_amount,  
 "tax\_due": tax\_due  
 })

Database Connection: Python Flask is used to connect to the SQLite database, with SQLAlchemy for ORM functionality.

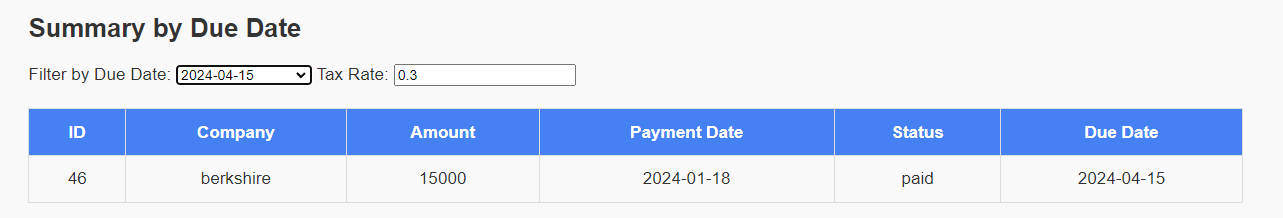
app = Flask(\_\_name\_\_)  
  
*# Database connection function*def connect\_db():  
 return sqlite3.connect('tax\_system.db')  
  
*# Home route*@app.route('/')  
def home():  
 return render\_template('index.html')

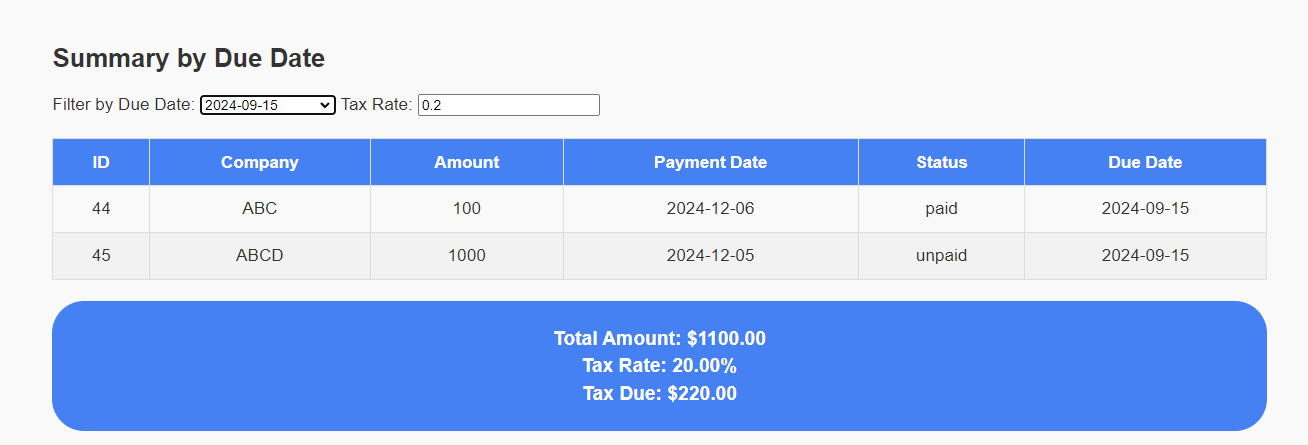
## Tax Breakdown Feature

Filter Functionality:  
- Users can select a due date from a dropdown.



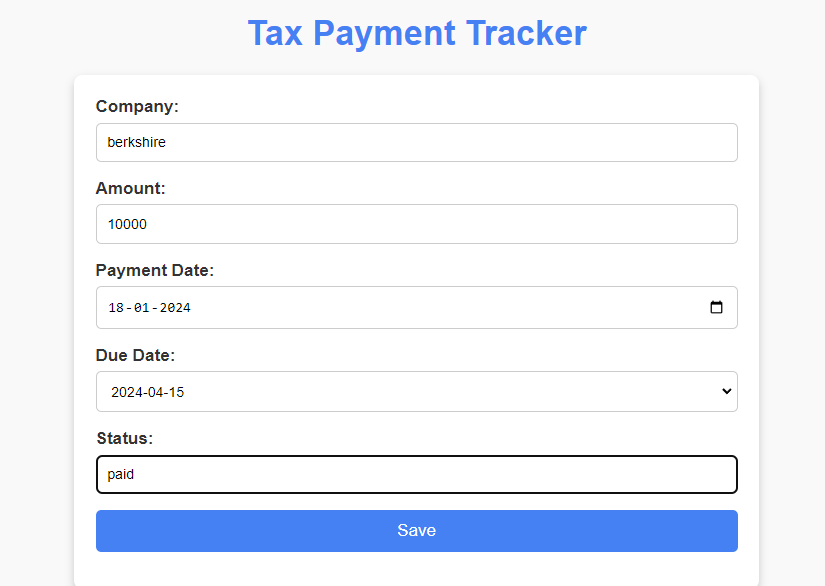
- Displays records matching the selected date.



- Calculates:  
 - Total Amount (Paid and Unpaid).  
 - Tax Due based on user-provided tax rate.  
  
Example:  


## CRUD Operations

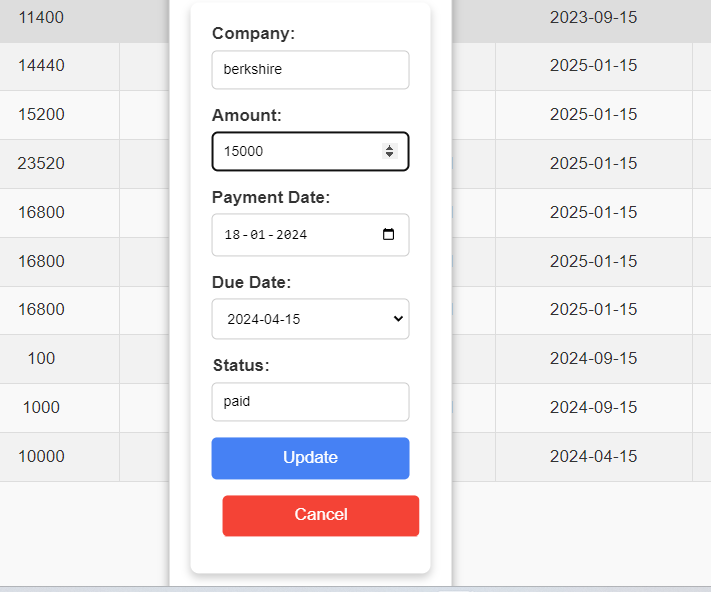
Create: Add records via the HTML form.



Read: Display all records in a tabular format.

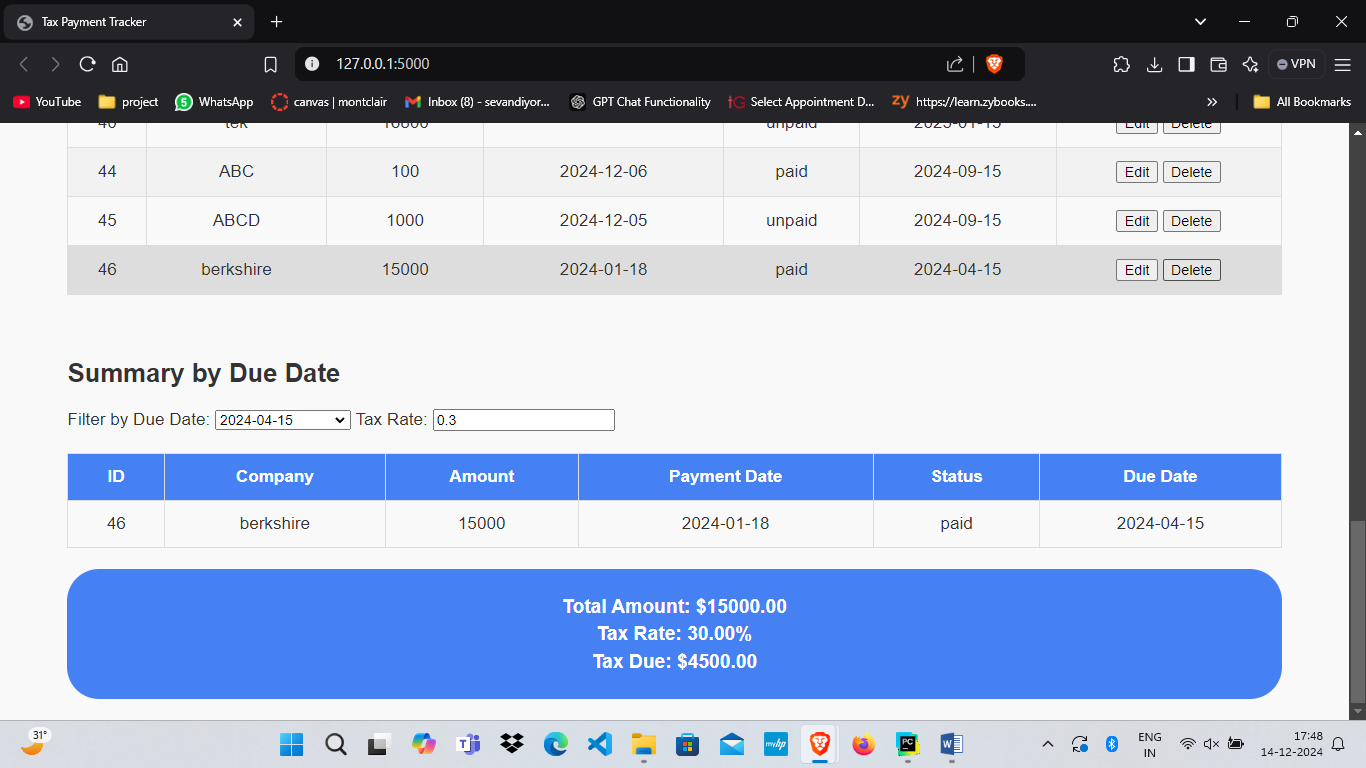


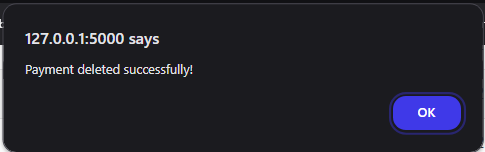
Update: Edit records using a popup form.



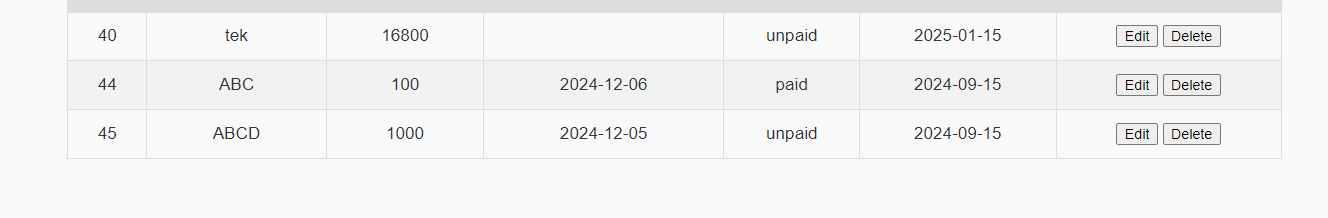


Delete: Confirm deletion with a popup before removing records.





* Deletion successful



## GitHub Repository

Link: Provide the public repository link here.  
  
Structure:  
- `/backend`: Python Flask code.  
- `/frontend`: HTML, CSS, and JavaScript files.  
- `/database`: SQLite database schema.

## YouTube Video

Link: Provide the YouTube video link here.  
  
Content: The video should cover:  
- SQL schema and table design.  
- CRUD operations.  
- UI walkthrough and dynamic features.  
- Tax breakdown and filter functionality.