MONTCLAIR STATE UNIVERSITY

DATABASE SYSTEMS – FINAL PROJECT

CSIT 555

SUBMITTED TO

INSTRUCTOR: XIAOFENG LI (KEVIN)

MONTCLAIR STATE UNIVERSITY

SUBMITTED BY :

GAURI PRAKASH DHANAWADE

TABLE OF CONTENT

INTRODUCTION

The Tax and Payment Tracking System is an online application that helps businesses and individuals handle their tax payments more efficiently. This system, built using Flask, SQLite, and JavaScript, provides an easy-to-use interface for effectively recording,

monitoring, and analysing taxes. This system allows users to conveniently enter data about their tax payments, such as the firm name, payment amount, payment date, status, due date, and tax rate. The application allows users to add, update, and delete payment records, giving them greater flexibility and control over their tax data.

SYSTEM ARCHITECTURE

The Tax and Payment Tracking System is a web-based application built using Flask as the web framework, SQLite as the relational database management system, and JavaScript for dynamic client-side interactions. The system has a three-tier architecture, which includes the presentation layer, the application layer, and the data layer, as stated below:

Presentation Layer

- The presentation layer handles the application's user interface (UI) components.
- It is made up of HTML templates, CSS stylesheets, and client-side JavaScript code.
- HTML templates are used to structure web page layouts and show dynamic content via template engines such as Jinja2.
- CSS stylesheets define the visual appearance and layout of UI elements, resulting in a consistent and visually pleasing user experience.
- Client-side JavaScript code improves interactivity and dynamic behaviour, including form validation, AJAX queries, and DOM manipulation.
- Input forms, tables, buttons, and modal dialogs are examples of UI components that offer users a responsive and intuitive interface for interacting with the system.

Application Layer

- The application layer contains the system's business logic as well as its server-side functions.
- It is built with the Flask web framework, which manages HTTP requests, URL routing, and request handling.
- Flask routes are used to bind URL endpoints to Python functions (view functions) that handle requests, interact with the database, and produce responses.
- View functions use the SQLite3 library to conduct CRUD (Create, Read, Update, and Delete) activities on payment records.
- The application layer also includes AJAX request handlers, which allow the client and server to communicate dynamically and asynchronously.
- In addition, the application layer includes error handling, session management, and authentication procedures to assure security and reliability.

Data Layer

- The data layer manages the system's data storage and retrieval.
- It uses SQLite as a relational database management system to store payment records in a structured format.
- The payments table schema specifies fields such as the firm name, payment amount, payment date, status, due date, and tax rate.

- Indexes are established on relevant fields to improve query performance and enable data retrieval.
- CRUD actions on the payments table are carried out with SQL queries conducted in Python using the SQLite3 module.
- Data integrity and consistency are ensured by restrictions such as primary keys, foreign keys, and data type validations.

DATABASE STRUCTURE AND CODE

The Tax and Payment Tracking System's database structure consists of a single table named "payments" that records tax payment information. The following SQL code defines the table structure, including primary keys, indexes, and constraints:

Code:

```
def create_table():
    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS payments
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  company TEXT,
                  amount REAL,
                  payment_date TEXT,
                  status TEXT,
                  due_date TEXT,
                  tax_rate REAL)''')
    # Create indexes
    c.execute('''CREATE INDEX IF NOT EXISTS idx_company ON payments (company)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_payment_date ON payments (payment_date)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_due_date ON payments (due_date)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_status ON payments (status)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_tax_rate ON payments (tax_rate)''')
    conn.commit()
    conn.close()
create_table()
```

Explanation:

- **ID**: The primary key column has auto-incremental integer values, ensuring that each entry has a unique identity.
- **Company(STRING)**: A column with the name of the firm making the payment. It is defined as NOT NULL to ensure data integrity.
- **Amount(FLOAT)**: A real number column that represents the payment amount. It is defined as NOT NULL with a positive value (CHECK constraint).
- **Payment_date(STRING (DATE))**: A column containing the date of payment. It supports NULL values for circumstances where the payment date is unclear.
- **Status(STRING)**: A column that indicates the payment status (for example, "paid" or "unpaid"). It's defined as NOT NULL.
- **due_date(STRING(DATE))**: This column represents the payment's due date. It's defined as NOT NULL.

- **tax_rate(FLOAT):** A real number column indicating the tax rate applied to the payment amount. It is defined as NOT NULL and must be in the range [0, 1] (CHECK constraint).

Constraints are added to ensure data integrity and consistency:

- company_not_null: Ensures that the company name is not empty.
- amount_positive: Ensures that the payment amount is non-negative.
- tax_rate_range: Ensures that the tax rate is within the range [0, 1].

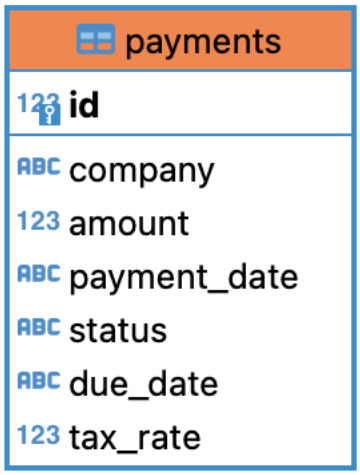
Indexes:

- Indexes are created on relevant columns to optimize query performance for common search and filter operations.
- Indexes are defined on the "company", "payment_date", "due_date", "status", and "tax_rate" columns to facilitate efficient data retrieval and filtering.

Primary Key:

- The primary key is “id INTEGER PRIMARY KEY AUTOINCREMENT”
- id: This column serves as the primary key for the "payments" table.
- INTEGER: Indicates that the data type of the primary key is an integer.
- PRIMARY KEY: Specifies that the "id" column is the primary key, uniquely identifying each record in the table.
- AUTOINCREMENT: This attribute ensures that each new row inserted into the table will automatically be assigned a unique integer value for the "id" column, incrementing from the highest existing value. It prevents the reuse of previously deleted "id" values, ensuring data integrity and uniqueness.

ER DIAGRAM:



Database Connection:

app.py

```python
from flask import Flask, render_template, request, redirect, url_for, jsonify
import sqlite3
import datetime
```

SQL SCHEMA:

```python
def create_table():  1 usage
    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS payments
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  company TEXT,
                  amount REAL,
                  payment_date TEXT,
                  status TEXT,
                  due_date TEXT,
                  tax_rate REAL)''')
    # Create indexes
    c.execute('''CREATE INDEX IF NOT EXISTS idx_company ON payments (company)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_payment_date ON payments (payment_date)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_due_date ON payments (due_date)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_status ON payments (status)''')
    c.execute('''CREATE INDEX IF NOT EXISTS idx_tax_rate ON payments (tax_rate)''')

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

Controllers and Endpoints:

```python
@app.route( rule: '/submit', methods=['POST'])
def submit():
    company = request.form['company']
    amount = float(request.form['amount'])
    payment_date = request.form['paymentDate']
    status = request.form['status']
    due_date = request.form['dueDate']
    tax_rate = float(request.form['taxRate'])

    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute( sql: '''INSERT INTO payments (company, amount, payment_date, status, due_date, tax_rate)
                VALUES (?, ?, ?, ?, ?, ?)''', parameters: (company, amount, payment_date, status, due_date, tax_rate))
    conn.commit()
    conn.close()
    return redirect(url_for('index'))
```

```python
@app.route('/')
def index():
    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()

    current_year = datetime.datetime.now().year
    next_year = current_year + 1
    due_dates = [
        f"04/15/{current_year}",
        f"06/15/{current_year}",
        f"09/15/{current_year}",
        f"01/15/{next_year}"
    ]
    c.execute('''SELECT * FROM payments''')
    records = c.fetchall()
    return render_template( template_name_or_list: 'index.html',due_dates=due_dates,records=records)
```

```python
@app.route( rule: '/insert', methods=['POST'])
def insert_record():
    data = request.get_json()
    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute( sql: '''INSERT INTO payments (company, amount, payment_date, status, due_date, tax_rate)
                VALUES (?, ?, ?, ?, ?, ?)''',
               parameters: (data['company'], data['amount'], data['payment_date'],
               data['status'], data['due_date'], data['tax_rate']))
    conn.commit()
    conn.close()
    return 'Record inserted successfully'
```

app.py

```python
@app.route('/summary')
def summary():
    due_date = request.args.get('dueDate')

    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute( sql: '''SELECT * FROM payments WHERE due_date=?''', parameters: (due_date,))
    data = c.fetchall()
    total_amount = sum(row[2] for row in data)

    # Retrieve tax rate and format it as percentage
    tax_rate = data[0][6] * 100 if data else 0
    tax_due = total_amount * (tax_rate / 100)  # Calculate tax due based on percentage
    conn.close()

    html = '<table border="1"><tr><th>ID</th><th>Company</th><th>Amount</th><th>Payment Dates</th><th>Status</th><th>Due
    for row in data:
        payment_date = row[3] if row[3] else 'NA'
        formatted_date = format_date_mmddyyyy(payment_date)
        html += f'<tr><td>{row[0]}</td><td>{row[1]}</td><td>{row[2]}</td><td>{formatted_date}</td><td>{row[4]}</td><td>{r
        html += '</td></tr>'
    html += f'<tr><td colspan="5"><strong>Total Amount:</strong></td><td>&dollar;{total_amount}</td></tr><tr><td colspan=
    html += '</table>'
```

```python
@app.route( rule: '/update', methods=['POST'])
def update():
    edit_id = request.form['editId']
    company = request.form['editCompany']
    amount = float(request.form['editAmount'])
    payment_date = request.form['editPaymentDate']
    status = request.form['editStatus']
    due_date = request.form['editDueDate']
    tax_rate = float(request.form['editTaxRate'])

    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute( sql: '''UPDATE payments SET company=?, amount=?, payment_date=?, status=?, due_date=?, tax_rate=? WHERE id=?
              parameters: (company, amount, payment_date, status, due_date, tax_rate, edit_id))
    conn.commit()
    conn.close()

    return jsonify({'success': True})
```

```python
@app.route( rule: '/delete', methods=['DELETE'])
def delete():
    delete_id = request.args.get('id')

    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute( sql: '''DELETE FROM payments WHERE id=?''', parameters: (delete_id,))
    conn.commit()
    conn.close()
```

API's:

1. **POST '/submit':**
   Description: Submits payment data via a form submission.
   Controller: submit() function.
   Method: POST**.**

2. **GET '/':**
   Description: Renders the main page of the application, displaying payment records and a form for submitting new payments.
   Controller: index() function.
   Method: GET.

3. **POST '/insert':**
   Description: Inserts a new payment record via an AJAX request.
   Controller: insert_record() function.
   Method: POST.

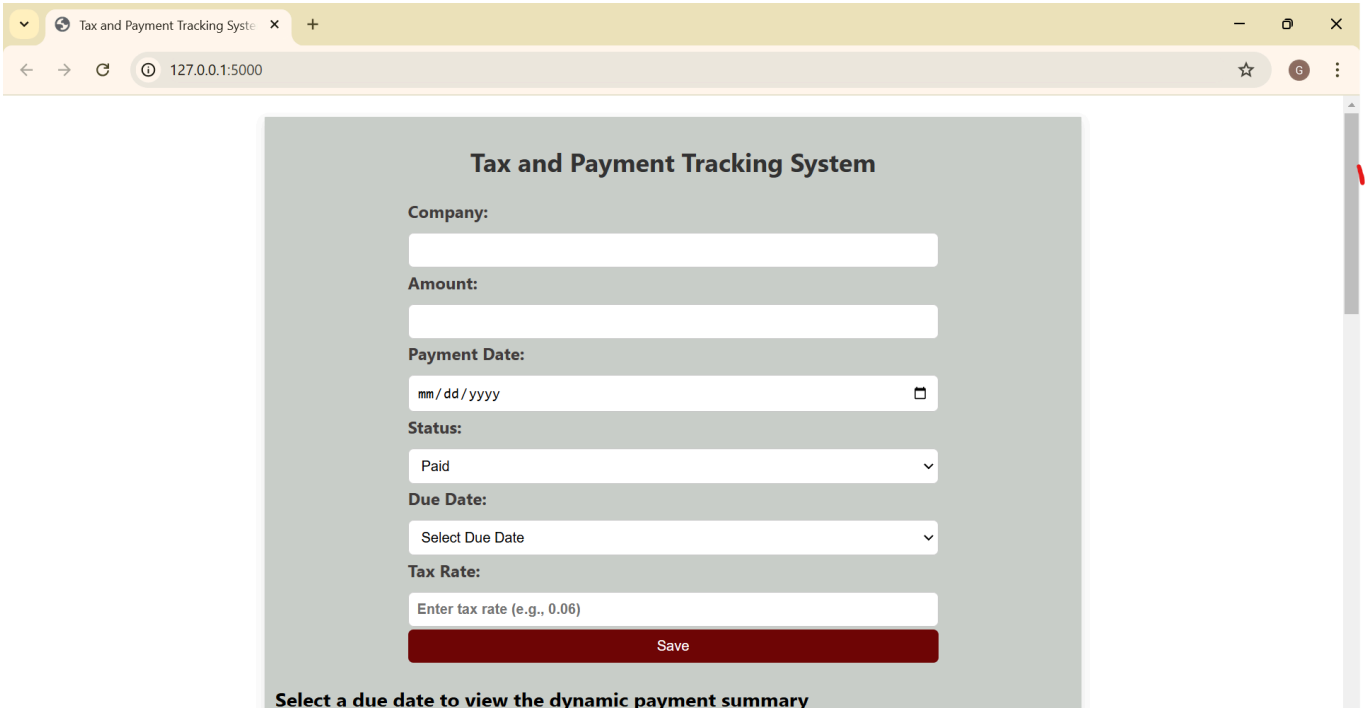
4. **GET '/summary':**
   Description: Fetches payment summary data for a specific due date.
   Controller: summary() function.
   Method: GET.

5. **POST '/update':**
   Description: Updates an existing payment record.
   Controller: update() function.
   Method: POST.
6. **DELETE '/delete':**

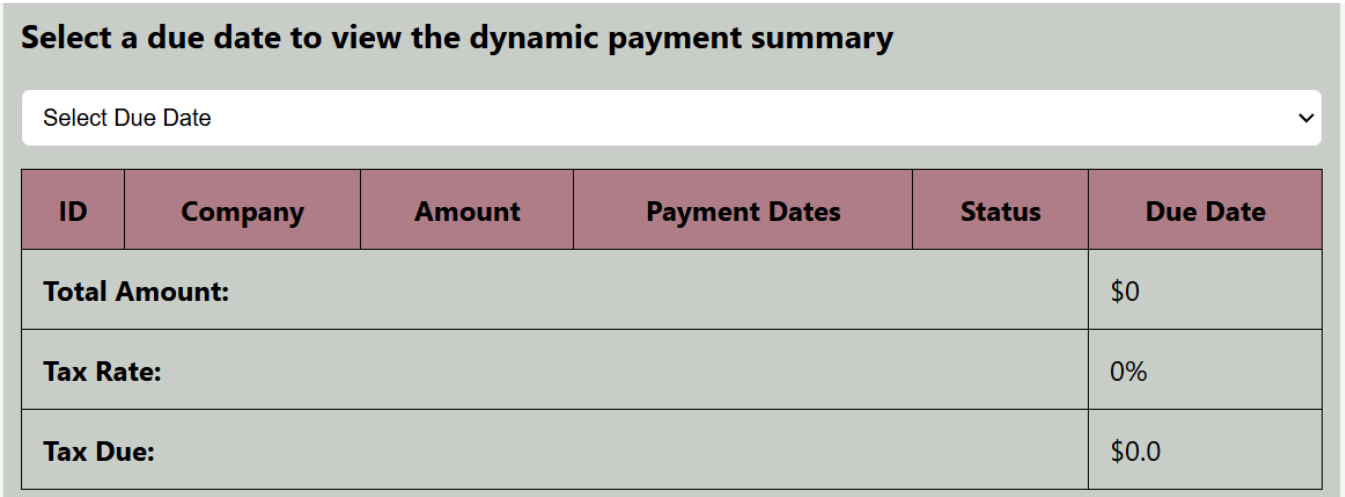
Description: Deletes an existing payment record.
Controller: delete() function.
Method: DELETE.

SCREENSHOTS

Form to enter the data and save:



Dynamic Payment Summary:



Due Date 04/15/2024:

Select a due date to view the dynamic payment summary

04/15/2024

ID	Company	Amount	Payment Dates	Status	Due Date
3	tek	15200.0	06/09/2023	paid	04/15/2024
Total Amount:					$15200.0
Tax Rate:					12.0%
Tax Due:					$1824.0

Due Date 06/15/2024:

Select a due date to view the dynamic payment summary

06/15/2024

ID	Company	Amount	Payment Dates	Status	Due Date
1	derm	4100.0	09/26/2023	paid	06/15/2024
4	tek	15200.0	07/12/2023	paid	06/15/2024
9	tek	16800.0	NA	unpaid	06/15/2024
Total Amount:					$36100.0
Tax Rate:					8.0%
Tax Due:					$2888.0

Due Date 09/15/2024:

Select a due date to view the dynamic payment summary

09/15/2024

ID	Company	Amount	Payment Dates	Status	Due Date
5	tek	11400.0	08/11/2023	paid	09/15/2024
10	tek	16800.0	NA	unpaid	09/15/2024
Total Amount:					$28200.0
Tax Rate:					9.0%
Tax Due:					$2538.0

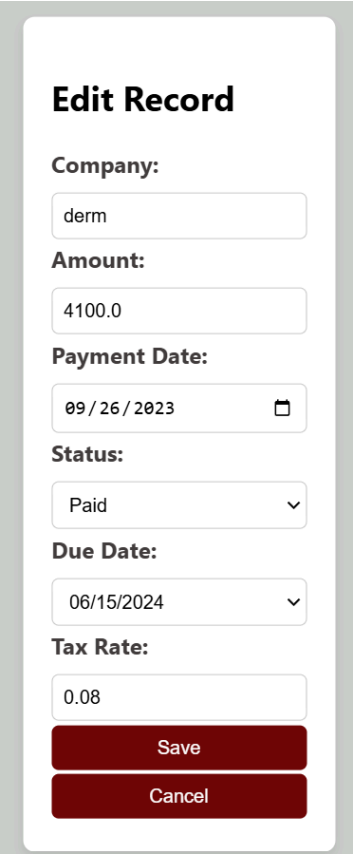
Due Date : 1/15/2025

Select a due date to view the dynamic payment summary

01/15/2025

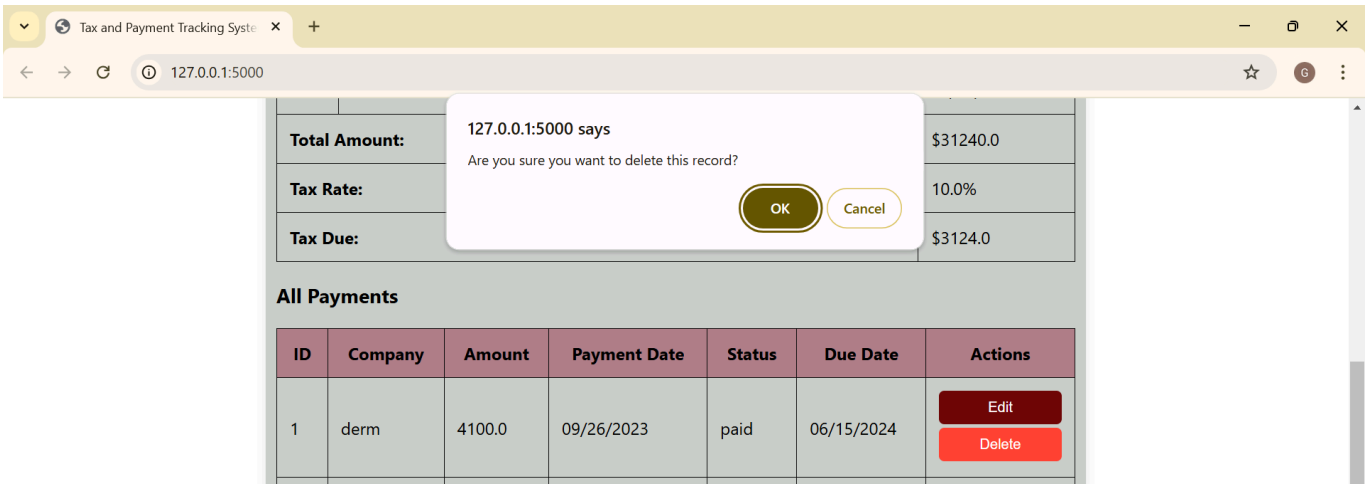
ID	Company	Amount	Payment Dates	Status	Due Date
6	tek	14440.0	09/21/2023	paid	01/15/2025
11	tek	16800.0	12/26/2024	unpaid	01/15/2025
Total Amount:					$31240.0
Tax Rate:					10.0%
Tax Due:					$3124.0

Edit the data:

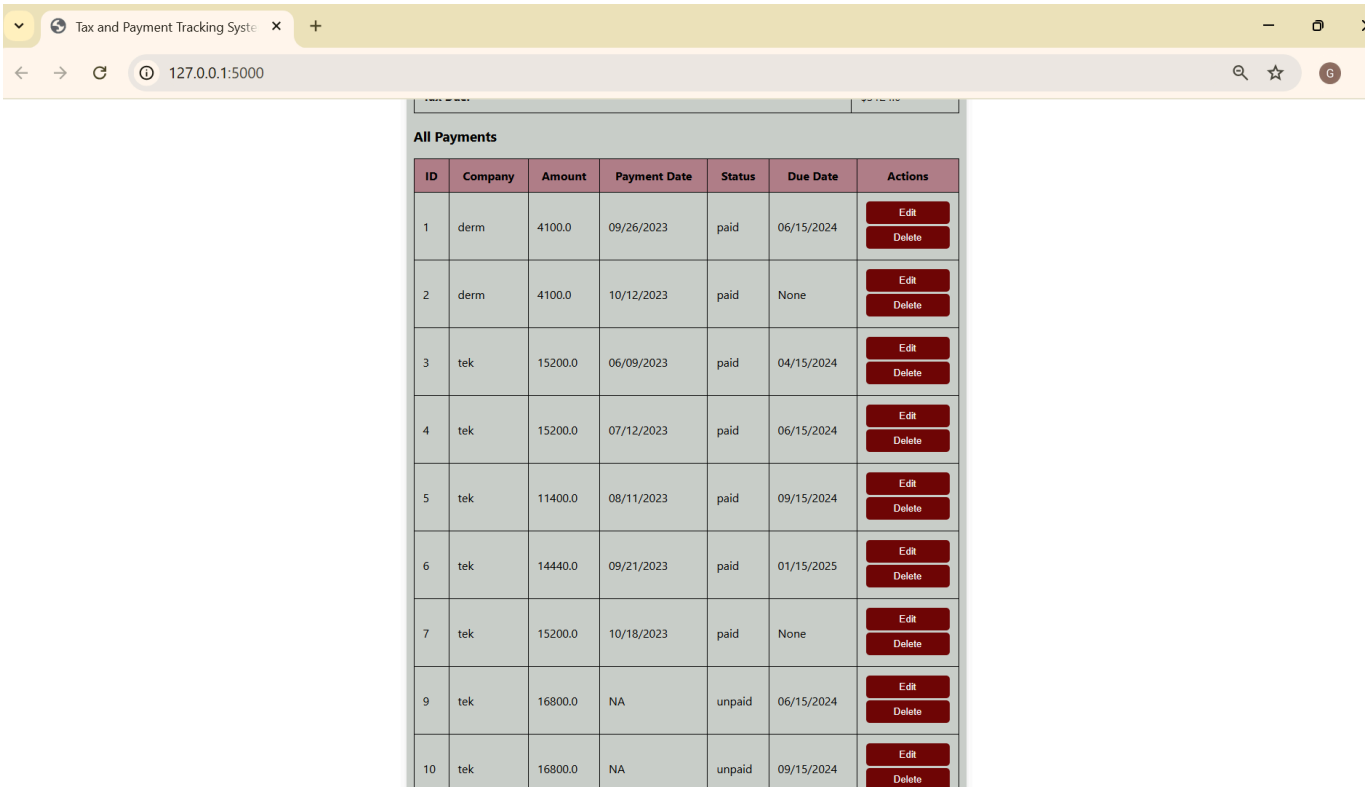


Delete the data:

After Clicking on the delete button a pop up will a appear to confirm the operation:



ALL Payments:



GIT Repository Link: https://github.com/GauriMSU/CSIT555DBMS.git

Youtube Video Link : https://youtu.be/eko1Ygd8D2g