PERSONAL INCOME TAX SYSTEM (PITS)

Assignment 3: Distributed Systems

Unit Code: CSI3344D

Due Date: Fri, 25 Oct 2024 21:29

Name	Student ID
Minaga Piyadigama	10659027
Nikesh Joshua	10646591

Executive Summary

The purpose of this project is to develop a simplified Personal Income Tax Return Estimate (PITRE) system, which mimics real-world tax estimation systems. The system is implemented as a distributed application using Pyro5, a Python-based Remote Procedure Call (RPC) framework, to enable client-server communication.

The project was divided into two phases -

- **Phase 1** - Implementation of a two-tier client-server architecture, where the TRE client communicates with a central GTO server to estimate an individual's tax return based on user input.
- **Phase 2** - Expansion of the system to a three-tier architecture by introducing a database server (PITD), which stores users' tax-related data. This allows users with Tax File Numbers (TFNs) to have their data automatically fetched from the database for tax return estimation.

Key features of the system include user authentication, secure communication between clients and servers, and accurate tax calculations based on the Australian tax rate system. The project was implemented using Python and Pyro5 for RPC, providing efficient distributed interactions and modularity in client-server communication.

Through this project, we have demonstrated the core concepts of distributed systems, such as client-server architecture, remote method invocation, and data management across multiple tiers.

Table of Contents

1. Introduction

This report presents the development of a distributed system aimed at providing a simplified Personal Income Tax Return Estimate (PITRE), which emulates a government tax estimation system. The system is designed to allow individual users to estimate their tax refunds or liabilities by interacting with the system over a distributed architecture.

The system was implemented using **Pyro5**, a Python-based Remote Procedure Call (RPC) library. RPC is widely recognized as a foundational communication mechanism in distributed systems, enabling remote method invocation between components in separate machines. Pyro5 provides a Pythonic approach to RPC by abstracting the complexity of network communication while maintaining modularity and flexibility in client-server communication. (*Pyro - Python Remote Objects - 5.14-Dev — Pyro 5.14-Dev Documentation*, n.d.)

The distributed system consists of two main phases -

- **Phase 1** - A two-tier client-server architecture, where the client communicates with a server to perform tax calculations based on user input.
- **Phase 2** - A three-tier system that introduces a database server for retrieving user-specific data for tax return estimation.

The scope of the project includes ensuring secure communication, implementing a simple but accurate tax calculation based on the Australian tax system, and managing user data efficiently. The use of Pyro5 establishes reliable and asynchronous communication between distributed components, which is essential in modern systems handling real-time or near-real-time requests.

Key assumptions made during the project include using a simplified tax formula and limiting the scope to handling basic tax-related data for individuals. The primary focus was on demonstrating distributed system design using Pyro5 for seamless client-server interactions.

2. Background

In modern distributed systems, efficient communication between different machines and applications is critical. As businesses and governments handle large amounts of data and tasks across multiple locations, distributed systems have become essential. One such challenge is the calculation of personal tax returns, where data from multiple sources, such as employers, banks, and government offices, must be collected, processed, and stored efficiently.

The Personal Income Tax Return Estimate (PITRE) system simulates how a distributed tax system can help individuals calculate their tax liabilities or refunds based on their taxable income. In this project, we use **Pyro5**, an RPC framework, to enable remote communication between clients and servers, mimicking how data is processed in a real-world distributed tax system.

3. Scope

3.1 Functional Scope

This refers to the specific functionality that the system is designed to deliver -

- User Authentication: Only authenticated users can access the system to calculate tax returns. Users with a valid Tax File Number (TFN) can retrieve their stored data, while users without a TFN can manually input their tax-related data.
- Tax Calculation: The system calculates tax based on Australian tax rates and includes components like the Medicare Levy and Medicare Levy Surcharge.
- Data Retrieval: For users with a TFN, the system retrieves payroll data from the PITD database, managed using MySQL.

3.2 Technical Scope

The technical scope outlines the tools and technologies used to build the system and the boundaries of the technical environment -

- Pyro5 for RPC: The client-server communication is managed using Pyro5, which allows methods on remote servers to be called as if they were local.
- MySQL Database: The tax-related data is stored and managed using MySQL, ensuring reliable data storage and retrieval.
- Multi-Tier Architecture: The system is designed as a three-tiered distributed system, with separate layers for the client, application logic, and database.
- Platform Compatibility: The system is developed in Python, making it compatible across different operating systems that support Python and Pyro5.

3.3 Project Scope

The project scope defines what the project is expected to deliver and what is out of scope for this specific implementation.

Included -

- Developing the PITRE system that allows users to estimate their tax return based on the Australian tax system.
- Implementing RPC-based communication using Pyro5 between the client and server.
- Managing and retrieving tax-related data from a MySQL database.
- Creating a user-friendly client interface for data input and output display.

Excluded -

- Complex tax deduction rules (e.g., work-related deductions) are not part of this simplified system.
- Real-time database updates or large-scale data synchronization across distributed servers.
- Integration with external tax systems or APIs beyond the current local setup.

4. Main Body

4.1 Understanding of Concepts/Techniques

The design and implementation of the PITRE system relied heavily on foundational concepts in distributed systems. Remote Procedure Call (RPC) is a core technique that enables communication between different components of a distributed system. In this project, **Pyro5** was used as the RPC framework. Pyro5 abstracts the complexity of network communication and allows objects in Python to invoke methods remotely over a network.

The system follows a three-tier architecture model commonly used in distributed systems -

- **Client Tier**: Handles user input and output.
- **Application Tier**: Contains the business logic (GTO server in this case).
- **Data Tier**: Stores user data in the PITD server.

This architecture ensures that the system is modular and can be extended easily.

5. Remote Procedure Call

5.1 What is RPC?

Remote Procedure Call (RPC) is a protocol that allows a program to execute a procedure (function) on a different address space (usually on another computer) without the programmer needing to explicitly code the details for remote communication. It abstracts the complexity of networking, enabling function calls to occur as if the two components were on the same system. Pyro5 is an implementation of RPC in Python, enabling remote method invocation between distributed Python objects over a network. (*RPC Implementation Using Python | Vrishabhdhwaj*, n.d.)

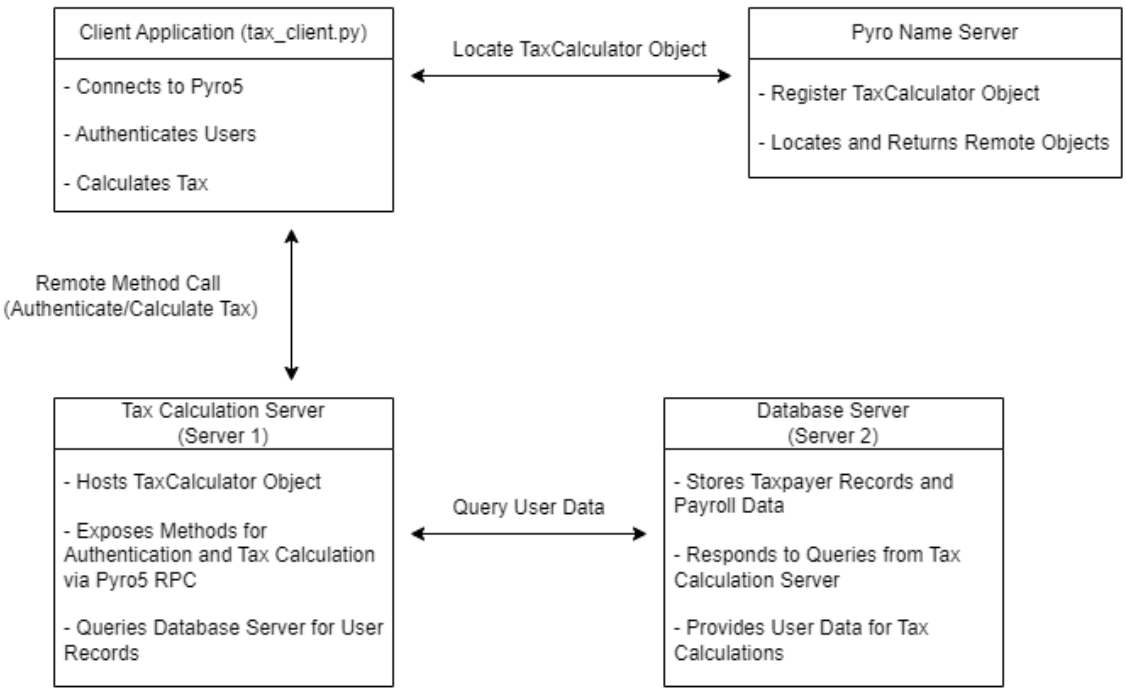
5.2 RPC Process

In an RPC system using Pyro5, the process generally follows these steps:

1. **Client Request**: The client program makes a call to a remote procedure as if it were local.
2. **Request Marshalling**: The client-side stub or proxy converts the call into a message format suitable for transmission over the network.
3. **Communication**: The marshalled message is sent over the network to the remote server.
4. **Server Unmarshalling**: The server receives the message, unpacks the request, and invokes the specified procedure.
5. **Response Marshalling**: The server processes the request, executes the procedure, and packs the response.
6. **Client Receives Response**: The client stub receives the response, unmarshalls it, and returns it as if the result were local.

6. RMI Diagram

RPC Architecture for Tax Calculation System Using Pyro5



6.1 Explanation

1. **Client Application (tax_client.py) -**
    - The client connects to the Pyro5 name server to locate the remote TaxCalculator object.
    - It authenticates users and requests tax calculations by communicating with the Tax Calculation Server.
2. **Pyro Name Server -**
    - It registers and locates the remote TaxCalculator object, enabling the client to access the server.
    - It facilitates the two-way communication between the client and the Tax Calculation Server.
3. **Tax Calculation Server (Server 1) -**
    - This server hosts the TaxCalculator object as a remote object accessible via Pyro5.
    - It handles user authentication and tax calculations using data stored on the Database Server.
    - It communicates with the Database Server to fetch or update user records.
4. **Database Server (Server 2) -**
    - The server stores all taxpayer records and payroll data.
    - It responds to data queries from the Tax Calculation Server to provide the necessary information for tax calculations.

6.2 RPC Flow

1. **Client Application Initialization** -
    - The **Client Application** (tax_client.py) connects to the Pyro Name Server to locate the TaxCalculator remote object. This step allows the client to access the methods exposed by the Tax Calculation Server (Server 1).
2. **Remote Method Invocation** -
    - The **Client Application** sends requests (like user authentication or tax calculation) to **Server 1** using Pyro5 RPC. The request is serialized, transmitted over the network, and executed remotely on **Server 1** as if it were a local method call.
3. **Querying the Database** -
    - When **Server 1** needs user data (e.g., payroll information), it queries **Server 2** (the Database Server). This interaction retrieves the necessary data to complete the requested calculation or process.
4. **Response Transmission** -
    - The response from **Server 1** is sent back to the **Client Application**. If it involved querying **Server 2**, the results from the database are included in this response.
5. **Output to User** -
    - The **Client Application** processes the response and displays the output to the user, whether it's a successful authentication, a tax calculation result, or any other response.

7. Application Requirements

The purpose of the Tax Calculator system is to provide users with an interactive interface for tax calculations, payroll management, and tax report retrieval through a distributed architecture using Pyro5 for Remote Procedure Call (RPC)-based communication. The key application requirements are outlined below, focusing on user authentication, data entry, tax calculation, report retrieval, technical implementation, security, error management, and modularity to ensure data integrity and smooth communication between components.

1. **User Authentication** -
    - Users must authenticate themselves using their credentials (Person ID and Password) before accessing any tax-related features.
    - The client application transmits the user credentials via an RPC call to the Tax Calculation Server (Server 1).
    - The server queries the user database to verify credentials. If credentials are valid, users can access services like tax calculations, tax report retrieval, and TFN registration.
    - If the authentication fails, the system prompts users with an error message, ensuring only authorized users access the system.
2. **Data Entry** -
    - Users can input various information like bi-weekly wages, tax withheld, personal details, and TFNs.
    - For users without a TFN, the system allows them to register by providing details like gross income, tax withheld, net income, and pay periods.
    - Input validation is applied rigorously to ensure that only correct data, such as valid TFNs and date formats (YYYY-MM-DD), are accepted.
    - If the validation fails, the system raises an error, ensuring data accuracy and system integrity.

3. **Tax Calculation** -
    - The system calculates tax based on user inputs or stored historical records from the database.
    - Users with a valid TFN can have their data automatically retrieved from the database.
    - Tax liabilities are calculated using Australian tax brackets, including the Medicare Levy and Medicare Levy Surcharge.
    - This process ensures that users receive an accurate tax estimation based on their input or historical records.
4. **Result Display** -
    - Once the tax calculation is complete, the system displays results, indicating whether users owe additional tax or are eligible for a refund.
    - Users are provided with clear and concise notifications like "You owe additional taxes - $XXX.XX" or "You will receive a refund - $XXX.XX."
    - The interface ensures that users can easily interpret their tax status.
5. **Technical Implementation** -
    - The Tax Calculator system uses Pyro5 to manage RPC-based communication between the client and server.
    - The system follows a multi-tier architecture, with the Tax Calculator object hosted on Server 1 and registered with the Pyro Name Server.
    - The system includes a database server (Server 2) that stores user data and payroll records. Server 1 retrieves necessary data for calculations from Server 2, ensuring secure and efficient communication.

6. **Security and Error Management** -
    - The system ensures data security by validating all inputs and allowing only authenticated users to access sensitive processes.
    - Database access is controlled, with only Server 1 able to query or retrieve user data.
    - Passwords are securely stored, and authentication uses encrypted password comparisons.
    - Comprehensive error handling provides users with appropriate feedback when invalid inputs or system errors occur.
7. **Extensibility and Modularity** -
    - The system is designed to be modular and extendable, allowing for easy addition of new features or updates without affecting existing components.
    - The architecture supports the independent functioning of each component, ensuring maintainability and scalability.

8. Application Design and Implementation Procedure (SDLC)

The Tax Calculator system adheres to the Software Development Life Cycle (SDLC) to ensure structured development, modularity, and maintainability. The system follows a multi-tier architecture that includes clients, servers, and a database to provide comprehensive tax-related services. The design and implementation are carried out in phases, adhering to SDLC best practices.

1. Application Design

The system is divided into three core components:

- **Client Application (tax_client.py)**: A command-line interface (CLI) for user interaction.
- **Tax Calculation Server (Server 1)**: The core processing unit handling tax calculations and user authentication.
- **Database Server (Server 2)**: Manages user and payroll data storage and retrieval.

Each component has a defined role to promote modularity, scalability, and maintainability.

a) Client Application (tax_client.py)

- The client application serves as the primary interface for users to interact with the system.
- It allows users to authenticate, register TFNs, enter payroll data, and retrieve tax records.
- The application connects to the Pyro Name Server to locate the remote TaxCalculator object and interacts with it through RPC.
- The client validates basic user inputs (e.g., ensuring that TFNs are numeric, and dates are correctly formatted) before sending data to the server for processing.

b) Tax Calculation Server (Server 1)

- The Tax Calculation Server hosts the TaxCalculator object, which is registered with the Pyro Name Server, making it accessible to clients remotely.
- It handles requests such as user authentication, TFN registration, and tax calculations.
- The server queries the database server for user records when needed, ensuring that data retrieval and updates are securely managed.
- The server includes validation methods for ensuring data integrity, such as checking wage values and validating date formats.

c) Database Server (Server 2)

- The database server (Server 2) uses MySQL to store and manage user and payroll data.
- It is accessible only through the Tax Calculation Server, ensuring that clients cannot directly access the database.
- The database server stores tables like users, tfn_free_users, payroll_records, and tax_history.
- The server responds to data queries from Server 1 and provides necessary user data for tax calculations.

2. Implementation Procedure

The development of the Tax Calculator system followed a structured SDLC approach:

- **Requirement Analysis** -
    - Defined key functionalities, including user authentication, tax calculation, and data retrieval.
    - Identified necessary components (Client, Server, Database) and their interactions.
- **System Design** -
    - Designed the multi-tier architecture, ensuring modularity and security.
    - Developed detailed class diagrams, process flows, and RPC communication models.
- **Implementation** -
    - Developed the client application using Python and Pyro5 for RPC-based communication.
    - Implemented the server with validation methods, tax calculation logic, and database interaction modules.
    - Set up the database server using MySQL and designed the database schema to store user and payroll data.
- **Testing and Validation** -
    - Conducted unit and integration tests to ensure the client-server communication and database interactions were functioning correctly.
    - Validated input handling, authentication processes, and error management to ensure system robustness.
- **Deployment and Maintenance** -
    - Deployed the components on separate machines to simulate a distributed environment.

- Provided documentation and user manuals to facilitate ease of use and future maintenance.

9. Implementation Procedure

9.1 Phase 1 Implementation

System Overview

The Phase 1 implementation of the Personal Income Tax Return Estimate (PITRE) system is designed as a two-tier client-server architecture using Pyro5 for remote method invocation (RMI). The system allows clients to calculate their estimated tax refund or additional tax owed based on biweekly wage data or a Tax File Number (TFN). The server performs calculations including base tax, Medicare Levy, and Medicare Levy Surcharge, and returns the result to the client.

- Client - Collects user data (TFN, income, etc.) and sends requests to the server.
- Server - Processes the client requests, performs tax calculations based on user data, and responds with the estimated refund or additional tax owed.

Architecture

The architecture of the system is designed to support interaction between the client and server using Pyro5. Below is a simplified flow:

- Client - Collects user inputs (e.g., TFN, biweekly wages, tax withheld, annual income, insurance status) and sends these inputs to the server via Pyro5.
- Server - The server calculates the estimated tax based on the client's inputs and returns either a refund or an additional tax amount.

User Manual

After installing pyro 5, open a new terminal and run the following command – (Pyro5 Name Server) and the output will look like the following image.

```
PS C:\Users\minag> python -m Pyro5.nameserver
>>
Not starting broadcast server for IPv6.
NS running on localhost:9090 (::1)
URI = PYRO:Pyro.NameServer@localhost:9090
```

And open another terminal and the run server with the following command.

```
PS C:\Users\minag> python server.py
Server 1 is ready and waiting for clients.
```

Open another terminal and the run the client with the following command.

With User choosing option 1: -

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 1
Enter your TFN: 123456
You will receive a refund: $1000.00
PS C:\Users\minag>
```

With User choosing option 2: -

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 2
Enter your biweekly net wages: 1500
Enter your tax withheld: 5000
Enter your estimated annual income: 60000
Do you have private health insurance (yes/no)? yes
You owe additional tax: $8875.00
PS C:\Users\minag>
```

Code Documentation

server.py

server.py client.py

C: > Users > minag > server.py > ...

```python
import Pyro5.api

@Pyro5.api.expose
class TaxCalculator:
    def calculate_tax(self, tfn):
        # Simulated calculation logic for TFN-based tax calculation
        if tfn == 123456:
            return 1000  # Example refund amount
        else:
            return "Error: TFN not found."

    def calculate_tax_with_biweekly_data(self, net_wages, tax_withheld):
        annual_income = net_wages * 26
        tax_due = annual_income * 0.325  # Tax rate applied based on the taxable income
        total_tax_due = tax_due - tax_withheld
        return total_tax_due

    def calculate_medicare_levy(self, annual_income):
        return annual_income * 0.02

    def calculate_medicare_levy_surcharge(self, annual_income, has_insurance):
        if not has_insurance and annual_income > 90000:
            if annual_income <= 105000:
                return annual_income * 0.01
            elif annual_income <= 140000:
                return annual_income * 0.0125
            else:
                return annual_income * 0.015
        return 0

def main():
    daemon = Pyro5.api.Daemon()
    ns = Pyro5.api.locate_ns()
    uri = daemon.register(TaxCalculator)
    ns.register("Aus.taxcalculator", uri)
    print("Server 1 is ready and waiting for clients.")
    daemon.requestLoop()

if __name__ == "__main__":
    main()
```

client.py

server.py client.py

C: > Users > minag > client.py > main

```python
import Pyro5.api

def main():
    tax_server = Pyro5.api.Proxy("PYRONAME:Aus.taxcalculator")

    choice = input("Enter 1 for tax calculation with TFN or 2 without TFN: ").strip()
    if choice == "1":
        tfn = input("Enter your TFN: ").strip()
        if tfn.isdigit():
            try:
                refund_or_payment = tax_server.calculate_tax(int(tfn))
                if isinstance(refund_or_payment, str):
                    print(refund_or_payment)
                elif refund_or_payment < 0:
                    print(f"You owe additional tax: ${-refund_or_payment:.2f}")
                else:
                    print(f"You will receive a refund: ${refund_or_payment:.2f}")
            except Exception as e:
                print(f"Error occurred during tax calculation: {e}")
        else:
            print("Error: TFN must be a valid number.")
    elif choice == "2":
        net_wages = float(input("Enter your biweekly net wages: "))
        tax_withheld = float(input("Enter your tax withheld: "))
        annual_income = input("Enter your estimated annual income: ")

        if not annual_income.isdigit():
            print("Error: Annual income must be a valid number.")
            return

        has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
        try:
            # Call the methods for tax calculation
            tax_due = tax_server.calculate_tax_with_biweekly_data(net_wages, tax_withheld)
            medicare_levy = tax_server.calculate_medicare_levy(float(annual_income))
            mls = tax_server.calculate_medicare_levy_surcharge(float(annual_income), has_insurance)
            total_tax = tax_due + medicare_levy + mls

            # Adjusting the logic for refund or tax owed
            if total_tax < 0:
                print(f"You will receive a refund: ${-total_tax:.2f}")
            else:
                print(f"You owe additional tax: ${total_tax:.2f}")
        except Exception as e:
            print(f"Error occurred during tax calculation: {e}")
```

1. Phase 1: Test Cases

Test Case ID	Input Description	Expected Output
TC1	TFN: 123456	"You will receive a refund: $1000.00"
TC2	TFN: 654321 (non-existent)	"Error: TFN not found."
TC3	Biweekly Wages: 1000, Tax Withheld: 10000, Annual Income: 52000, Insurance: yes	"You will receive a refund: $510.00"
TC4	Biweekly Wages: 1500, Tax Withheld: 5000, Annual Income: 100000, Insurance: no	"You owe additional tax: $10675.00"

Test Case 1

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 1
Enter your TFN: 123456
You will receive a refund: $1000.00
PS C:\Users\minag>
```

Test Case 2

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 1
Enter your TFN: 654321
Error: TFN not found.
PS C:\Users\minag>
```

Test Case 3

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 2
Enter your biweekly net wages: 1000
Enter your tax withheld: 10000
Enter your estimated annual income: 52000       found.
Do you have private health insurance (yes/no)? yes
You will receive a refund: $510.00
```

Test Case 4

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 2
Enter your biweekly net wages: 1500
Enter your tax withheld: 5000
Enter your estimated annual income: 100000
Do you have private health insurance (yes/no)? no
You owe additional tax: $10675.00
PS C:\Users\minag>
```

2. Phase 2 Implementation

Admin login credentials

Admin ID - admin001

Password – adminpass

User Login credentials

User ID - user101

Password - userpass101

For login purposes as a admin an user

Tax System as an Ordinary user

User chooses option 1 – User Login and login with user credentials

```
PS C:\Users\minag\__init__.py> python client1.py
Welcome to the Tax System

Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 1
Enter your Person ID: user101
Enter your password: UserPass101
Welcome, Alice Johnson!
```

User chooses Option 1 to register a new TFN

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Delete account
8. Logout
Enter your choice: 1
Enter your new TFN: 47394759
Enter your gross income: 5000
Enter your tax withheld: 100
Enter your net income: 50000
Enter pay period start date (YYYY-MM-DD): 2024-01-01
Enter pay period end date (YYYY-MM-DD): 2024-12-12
Do you have private health insurance (yes/no)? yes
TFN 47394759 and tax record registered successfully.
```

Database Snapshot of the Table users and payroll_records

```
mysql> SELECT * FROM users;
+----+-----------+----------+--------------+-----------------------+----------+-------+---------------+--------------+----------------------+
| id | person_id | tfn      | name         | email                 | password | role  | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+--------------+-----------------------+----------+-------+---------------+--------------+----------------------+
|  8 | admin001  | NULL     | Admin User   | admin@example.com     | adminpass | admin |          NULL |
NULL |                NULL |
|  9 | user101   | 47394759 | Alice Johnson | alice.johnson@example.com | UserPass101 | user  |      75000.00 |     10000.00 |                    1 |
```

```
mysql> SELECT * FROM payroll_records;
+----+-----------+-------------------+-----------------+--------------+----------+
| id | person_id | pay_period_start | pay_period_end | gross_income | tax_paid |
+----+-----------+-------------------+-----------------+--------------+----------+
| 28 | user101   | 2024-01-01       | 2024-12-12     |      5000.00 |   100.00 |
+----+-----------+-------------------+-----------------+--------------+----------+
1 row in set (0.00 sec)
```

User chooses option number 2 to calculate tax with a TFN number, automatically does the calculation because the user is logged in

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Delete account
8. Logout
Enter your choice: 2
You owe additional tax: $15875.00
```

If user chooses option number 3 to calculate tax without a TFN number, system asks for biweekly net wages, tax withheld, estimated annual income and asks the user if they do have private health insurance and calculate tax and return with additional tax or refund

```
Enter your choice: 3
Enter your biweekly net wages: 1000
Enter your tax withheld: 100
Enter your estimated annual income: 48000
Do you have private health insurance (yes/no)? yes
You owe additional tax: $9310.00
```

```
Enter your biweekly net wages: 1000
Enter your tax withheld: 10000
Enter your estimated annual income: 52000       fou
Do you have private health insurance (yes/no)? yes
You will receive a refund: $510.00
```

When the user chooses option number 4 View tax history, the system retrieves and displays the history of tax calculations previously performed for that user account, including details like the date of calculation, tax amount, and whether it was a refund or owed tax.

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Delete account
8. Logout
Enter your choice: 4
Date: 2024-10-23T13:21:44, Tax Amount: 9310.00, Refund: 0
Date: 2024-10-23T13:12:39, Tax Amount: 15875.00, Refund: 0
```

If the user chooses option number 5 user can retrieve their own personal payroll records

```
Enter your choice: 5

View Your Payroll Records:
Start: 2024-01-01, End: 2024-12-12, Gross Income: 5000.00, Tax Paid: 100.00
```

If the user chooses option number 6 user can update their basic profile information like email and the password.

```
Enter your choice: 6

Update Profile:
Do you want to update your email? (yes/no): yes
Do you want to update your password? (yes/no): yes
Enter your new email: alice.johnson@gmail.com
Enter your new password: userpass101
Profile updated successfully.
```

Database snapshot of user updated profile information

```
mysql> SELECT * FROM users WHERE person_id = 'user101';
+----+-----------+----------+---------------+-----------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name          | email                 | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+---------------+-----------------------+-------------+------+---------------+--------------+----------------------+
|  9 | user101   | 47394759 | Alice Johnson | alice.johnson@gmail.com | userpass101 | user |      75000.00 |     10000.00 |                    1 |
+----+-----------+----------+---------------+-----------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

Removed the delete account due to it involves sensitive financial information and records that usually require retention for compliance and legal purposes. As it a function like that should require administrative authentication.

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Logout
Enter your choice: 7
Logging out...
```

Tax system as an Administrative User

Admin login

Admin user chooses option 2 from the main menu and proceed with the admin ID and password to login.

```
Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 2
Enter Admin ID: admin001
Enter Admin password: adminpass
Welcome, Admin Admin User!

Admin Menu:
1. View all users
2. Add a new user
3. Update a user
4. Delete a user
5. View all tax history
6. Search users
7. Display payroll records of users
8. Insert record
9. Update record
10. Delete record
11. Export user data to CSV
12. Export tax history to CSV
13. View Database Schema
14. View All Records in a Table
15. Exit Admin Menu
Enter your choice:
```

If user chooses option 1 from the menu, user can view all the user information from the users table

```
Enter your choice: 1
{'id': 8, 'person_id': 'admin001', 'tfn': None, 'name': 'Admin User', 'email'
: 'admin@example.com', 'password': 'adminpass', 'role': 'admin', 'annual_inco
me': None, 'tax_withheld': None, 'has_health_insurance': None}
{'id': 9, 'person_id': 'user101', 'tfn': '47394759', 'name': 'Alice Johnson',
 'email': 'alice.johnson@gmail.com', 'password': 'userpass101', 'role': 'user
', 'annual_income': '75000.00', 'tax_withheld': '10000.00', 'has_health_insur
ance': 1}{'id': 10, 'person_id': 'user102', 'tfn': None, 'name': 'Bob Smith',
 'email': 'bob.smith@example.com', 'password': 'UserPass102', 'role': 'user',
 'annual_income': '85000.00', 'tax_withheld': '12000.00', 'has_health_insuran
ce': 0}
{'id': 11, 'person_id': 'user103', 'tfn': None, 'name': 'Charlie Brown', 'ema
il': 'charlie.brown@example.com', 'password': 'UserPass103', 'role': 'user',
'annual_income': '95000.00', 'tax_withheld': '15000.00', 'has_health_insuranc
e': 1}
{'id': 12, 'person_id': 'user104', 'tfn': None, 'name': 'Diana Evans', 'email
': 'diana.evans@example.com', 'password': 'UserPass104', 'role': 'user', 'ann
ual_income': '65000.00', 'tax_withheld': '8000.00', 'has_health_insurance': 0
}
{'id': 13, 'person_id': 'user105', 'tfn': None, 'name': 'Edward Wright', 'ema
il': 'edward.wright@example.com', 'password': 'UserPass105', 'role': 'user',
'annual_income': '60000.00', 'tax_withheld': '6000.00', 'has_health_insurance
': 1}
{'id': 14, 'person_id': 'user106', 'tfn': None, 'name': 'Fiona Lee', 'email':
 'fiona.lee@example.com', 'password': 'UserPass106', 'role': 'user', 'annual_
income': '70000.00', 'tax_withheld': '7000.00', 'has_health_insurance': 0}
Enter your choice:
```

If user chooses option 2, the admin user can add a user by entering user credentials – ID, name, email, password, offer the new user admin or the user role, TFN, annual income, tax withheld and if the user has health insurance.

```
Add New User:
Enter the person ID: user107
Enter the name: John Doe
Enter the email: john.doe@gmail.com
Enter the password: userpass107
Enter the role (user/admin): user
Enter the TFN (optional): 47392071
Enter the annual income (optional): 100000
Enter the tax withheld (optional): 1000
Do you have private health insurance (yes/no): yes
User added successfully.
Enter your choice:
```

Database snapshot of the added new user information

```
mysql> SELECT * FROM users WHERE person_id = 'user107';
+----+-----------+----------+----------+--------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name     | email              | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+----------+--------------------+-------------+------+---------------+--------------+----------------------+
| 17 | user107   | 47392071 | John Doe | john.doe@gmail.com | userpass107 | user |     100000.00 |      1000.00 |                    1 |
+----+-----------+----------+----------+--------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

If user chooses option 3 admin user can update information of any user in the system, as same as the add new user function as well offer an admin role to an ordinary user.

```
Enter your choice: 3

Update User:
Enter the person ID of the user to update: user102
Enter the new email (leave blank to keep unchanged):
Enter the new password (leave blank to keep unchanged):
Enter the new role (user/admin, leave blank to keep unchanged): admin
Enter the new TFN (leave blank to keep unchanged):
Enter the new annual income (leave blank to keep unchanged):
Enter the new tax withheld (leave blank to keep unchanged):
Do you have private health insurance (yes/no, leave blank to keep unchanged):
User updated successfully.
```

Snapshot of the database updated ordinary user to admin role by and existing Admin user by the user update function

```
mysql> SELECT * FROM users WHERE person_id = 'user102';
+----+-----------+------+-----------+-----------------------+-------------+-------+---------------+--------------+----------------------+
| id | person_id | tfn  | name      | email                 | password    | role  | annual_income | tax_withheld | has_health_insurance |
+----+-----------+------+-----------+-----------------------+-------------+-------+---------------+--------------+----------------------+
| 10 | user102   | NULL | Bob Smith | bob.smith@example.com | UserPass102 | admin |      85000.00 |     12000.00 |                    0 |
+----+-----------+------+-----------+-----------------------+-------------+-------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

If user chooses option 4, can delete an account after confirming yes or no, by simply entering the user ID

```
Enter your choice: 4

Delete User:
Enter the person ID of the user to delete: user102
Are you sure you want to delete this user? (yes/no): no
```

If the user chooses option 5, user can view tax history, the system retrieves and displays the history of tax calculations previously performed by all users, including details like the date of calculation, tax amount, and whether it was a refund or owed tax .

```
Enter your choice: 5
Person ID: user101, Date: 2024-10-23T13:21:44, Tax Amount: 9310.00, Refund: 0
Person ID: user101, Date: 2024-10-23T13:12:39, Tax Amount: 15875.00, Refund:
0
```

If the user chooses option 6, user can search an existing user in the system database simply by any credential like – ID, email, role they serve in the system, using the ID directly will return the exact user because it is unique.

```
Enter your choice: 6

Search Users:
Enter person ID (leave blank for any): user104
Enter email (leave blank for any):
Enter role (user/admin, leave blank for any):
{'id': 12, 'person_id': 'user104', 'tfn': None, 'name': 'Diana Evans', 'email
': 'diana.evans@example.com', 'password': 'UserPass104', 'role': 'user', 'ann
ual_income': '65000.00', 'tax_withheld': '8000.00', 'has_health_insurance': 0
}
```

If the user chooses option 7, user can view any payroll records related from the payroll_records table, user can enter a specific user ID or view all the records with ID, start date, end date, gross income and tax paid

```
Enter your choice: 7

View All Payroll Records:
Enter person ID to filter (leave blank to view all records):
Person ID: user101, Start: 2024-01-01, End: 2024-12-12, Gross Income: 5000.00
, Tax Paid: 100.00
```

If user chooses option 8, the system prompts the admin to enter a table name and the details of the fields to insert a new record into the specified table. This allows the admin to add new data entries directly into the database.

```
Enter your choice: 8

Insert Record:
Enter the table name: users
Enter field name (or 'done' to finish): name
Enter value for 'name': Roman Row
Enter field name (or 'done' to finish): person_id
Enter value for 'person_id': user108
Enter field name (or 'done' to finish): tfn
Enter value for 'tfn': 57309264
Enter field name (or 'done' to finish): email
Enter value for 'email': roman.row@gmail.com
Enter field name (or 'done' to finish): password
Enter value for 'password': userpass107
Enter field name (or 'done' to finish): role
Enter value for 'role': user
Enter field name (or 'done' to finish): annual_income
Enter value for 'annual_income': 80000
Enter field name (or 'done' to finish): tax_withheld
Enter value for 'tax_withheld': 5000
Enter field name (or 'done' to finish): has_health_insurance
Enter value for 'has_health_insurance': 1
Enter field name (or 'done' to finish): done
Record inserted successfully.
```

Database snapshot of the newly inserted record

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email               | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |      80000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

If the user chooses option 9, the system asks the admin for the table name, record ID, and the fields that need to be updated. It then updates the specified fields of the identified record, enabling the admin to modify existing data in the database.

```
Enter your choice: 9

Update Record:
Enter the table name: users
Enter the record ID to update: 18
Enter field name to update (or 'done' to finish): annual_income
Enter new value for 'annual_income': 100000
Enter field name to update (or 'done' to finish): done
Record updated successfully.
```

Database record of the updated user information

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+--------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email              | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+--------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |     100000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+--------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

If the user chooses option 10, the system prompts the admin to specify the table and the record ID they want to delete. It then deletes the identified record from the database, allowing the admin to remove unwanted or incorrect data.

```
Enter your choice: 10

Delete Record:
Enter the table name: users
Enter the record ID to delete: 18
Record deleted successfully.
```

Database snapshot of the user record deleted from the database

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
Empty set (0.00 sec)
```

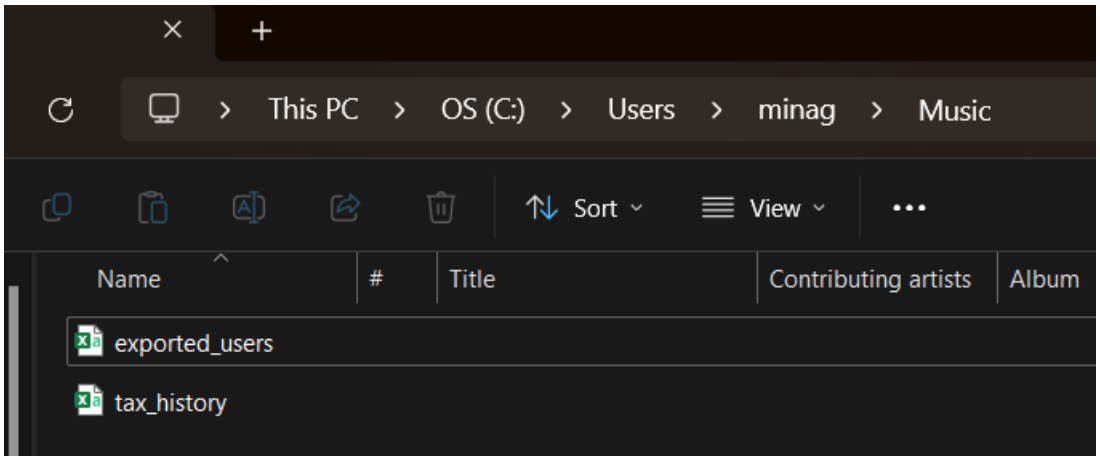
If the user chooses option 11 or 12,

Export User Data to CSV - The system exports all user data from the database into a CSV file. The admin can specify the file path or use the default file name. This feature provides a convenient way to back up or analyse user information.

Export Tax History to CSV - The system exports the entire tax history data from the database into a CSV file. The admin can set the file path or use the default name, allowing for backup, review, or analysis of tax records.

```
Enter your choice: 11
Enter the file path for the CSV export (default: exported_users.csv): C:\Users\minag\Music\exported_users.csv
User data exported successfully to C:\Users\minag\Music\exported_users.csv.
```

```
Enter your choice: 12
Enter the file path for the CSV export (default: exported_tax_history.csv): C:\Users\minag\Music\tax_history.csv
Tax history data exported successfully to C:\Users\minag\Music\tax_history.csv.
```



If user chooses option 13, the system retrieves and displays the schema of the database, showing the structure of all tables, including columns, data types, and keys. This helps the admin understand the organization and structure of the database.

```
Enter your choice: 13

--- Database Schema ---

Table: payroll_records
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  pay_period_start - date (Key: )
  pay_period_end - date (Key: )
  gross_income - decimal(10,2) (Key: )
  tax_paid - decimal(10,2) (Key: )

Table: tax_history
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  calculation_date - datetime (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  tax_amount - decimal(10,2) (Key: )
  is_refund - tinyint(1) (Key: )

Table: users
  id - int (Key: PRI)
  person_id - varchar(10) (Key: UNI)
  tfn - varchar(9) (Key: )
  name - varchar(100) (Key: )
  email - varchar(100) (Key: )
  password - varchar(50) (Key: )
  role - enum('user','admin') (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  has_health_insurance - tinyint(1) (Key: )
```

If user chooses option 14, the system asks the admin for a table name and retrieves all records from that table. It then displays the records, providing the admin with a complete view of the data stored within a specific table.

Bonus Marks Criteria

Implementing the Database Schema

For viewing ability for Admin User

```
Enter your choice: 13

--- Database Schema ---

Table: payroll_records
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  pay_period_start - date (Key: )
  pay_period_end - date (Key: )
  gross_income - decimal(10,2) (Key: )
  tax_paid - decimal(10,2) (Key: )

Table: tax_history
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  calculation_date - datetime (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  tax_amount - decimal(10,2) (Key: )
  is_refund - tinyint(1) (Key: )

Table: tfn_free_users
  id - int (Key: PRI)
  person_id - varchar(50) (Key: UNI)
  password - varchar(255) (Key: )
  name - varchar(255) (Key: )
  email - varchar(255) (Key: )
  created_at - timestamp (Key: )

Table: users
  id - int (Key: PRI)
  person_id - varchar(10) (Key: UNI)
  tfn - varchar(9) (Key: )
  name - varchar(100) (Key: )
  email - varchar(100) (Key: )
  password - varchar(50) (Key: )
  role - enum('user','admin') (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  has_health_insurance - tinyint(1) (Key: )
```

Display all Available data records

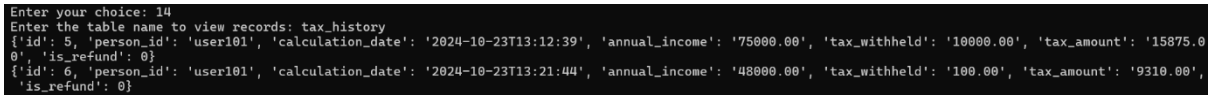
, from any table Admin user wish to view

users table information

TFN_free_users table information

payroll_records table information

tax_history table information



CRUD operation for data records for Admin users

Insert Record -

The system prompts the admin to enter a table name and the details of the fields to insert a new record into the specified table. This allows the admin to add new data entries directly into the database.

```
Enter your choice: 8

Insert Record:
Enter the table name: users
Enter field name (or 'done' to finish): name
Enter value for 'name': Roman Row
Enter field name (or 'done' to finish): person_id
Enter value for 'person_id': user108
Enter field name (or 'done' to finish): tfn
Enter value for 'tfn': 57309264
Enter field name (or 'done' to finish): email
Enter value for 'email': roman.row@gmail.com
Enter field name (or 'done' to finish): password
Enter value for 'password': userpass107
Enter field name (or 'done' to finish): role
Enter value for 'role': user
Enter field name (or 'done' to finish): annual_income
Enter value for 'annual_income': 80000
Enter field name (or 'done' to finish): tax_withheld
Enter value for 'tax_withheld': 5000
Enter field name (or 'done' to finish): has_health_insurance
Enter value for 'has_health_insurance': 1
Enter field name (or 'done' to finish): done
Record inserted successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+--------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email              | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+--------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |      80000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+--------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

Update Record -

The system asks the admin for the table name, record ID, and the fields that need to be updated. It then updates the specified fields of the identified record, enabling the admin to modify existing data in the database.

```
Enter your choice: 9

Update Record:
Enter the table name: users
Enter the record ID to update: 18
Enter field name to update (or 'done' to finish): annual_income
Enter new value for 'annual_income': 100000
Enter field name to update (or 'done' to finish): done
Record updated successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email               | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |     100000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

Delete Record -

The system prompts the admin to specify the table and the record ID they want to delete. It then deletes the identified record from the database, allowing the admin to remove unwanted or incorrect data.

```
Enter your choice: 10

Delete Record:
Enter the table name: users
Enter the record ID to delete: 18
Record deleted successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
Empty set (0.00 sec)
```

Integrated CRUD functions keeping track of the number of payroll tax data records and ensure they do not exceed 26 or 52 records per taxpayer.

```
def insert_record(self, table, fields):
    self.ensure_connection()
    try:
        if table == "users" and "person_id" not in fields:
            return "Error: 'person_id' is required for users table."

        if table == "payroll_records" and "person_id" in fields:
            person_id = fields["person_id"]
            if not self.user_exists(person_id):
                return f"Error: User with person_id '{person_id}' does not exist. Please add the user first."

            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT COUNT(*) AS record_count FROM payroll_records WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            record_count = cursor.fetchone()["record_count"]
            cursor.close()

            if record_count >= 26:
                return f"Error: Cannot exceed 26 payroll records for taxpayer {person_id}."

        cursor = self.connection.cursor()
        placeholders = ', '.join(['%s'] * len(fields))
        query = f"INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})"
        cursor.execute(query, list(fields.values()))
        self.connection.commit()
        cursor.close()
        return f"Record inserted into {table}."
    except Error as e:
        return f"Error inserting record into {table}: {e}"
```

User Registration

Created a user registration section to the main menu according to the appendix B section

First snapshot depicts a user registration without a TFN

```
PS C:\Users\minag> cd C:\Users\minag\__init__.py
PS C:\Users\minag\__init__.py> python client1.py
Welcome to the Tax System

Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 3

User Registration:
Enter your Person ID: user109
Enter your password: userpass109
Enter your name (optional): Olive Green
Enter your email (optional): oilve.green@gmail.com
Enter your TFN (optional, leave blank if you don't have one):
User 'user109' registered successfully.
```

Database Snapshot

```
mysql> SELECT * FROM TFN_free_users WHERE person_id = 'user109';
+----+-----------+-------------+-------------+----------------------+---------------------+
| id | person_id | password    | name        | email                | created_at          |
+----+-----------+-------------+-------------+----------------------+---------------------+
|  1 | user109   | userpass109 | Olive Green | oilve.green@gmail.com | 2024-10-23 19:04:16 |
+----+-----------+-------------+-------------+----------------------+---------------------+
1 row in set (0.00 sec)
```

Created a database table for TFN free users to register as a user and stored the data in the table

```
mysql> DESCRIBE TFN_free_users;
+------------+--------------+------+-----+-------------------+-------------------+
| Field      | Type         | Null | Key | Default           | Extra             |
+------------+--------------+------+-----+-------------------+-------------------+
| id         | int          | NO   | PRI | NULL              | auto_increment    |
| person_id  | varchar(50)  | NO   | UNI | NULL              |                   |
| password   | varchar(255) | NO   |     | NULL              |                   |
| name       | varchar(255) | YES  |     | NULL              |                   |
| email      | varchar(255) | YES  |     | NULL              |                   |
| created_at | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+--------------+------+-----+-------------------+-------------------+
6 rows in set (0.01 sec)
```

The second snapshot depicts a user registration with a TFN, and it will be stored to the users table because the user has a TFN

```
Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 3

User Registration:
Enter your Person ID: user110
Enter your password: userpass110
Enter your name (optional): Kelly Reynolds
Enter your email (optional): kelly.reynolds@gmail.com
Enter your TFN (optional, leave blank if you don't have one): 47502438
User 'user110' registered successfully.
```

Database Snapshot

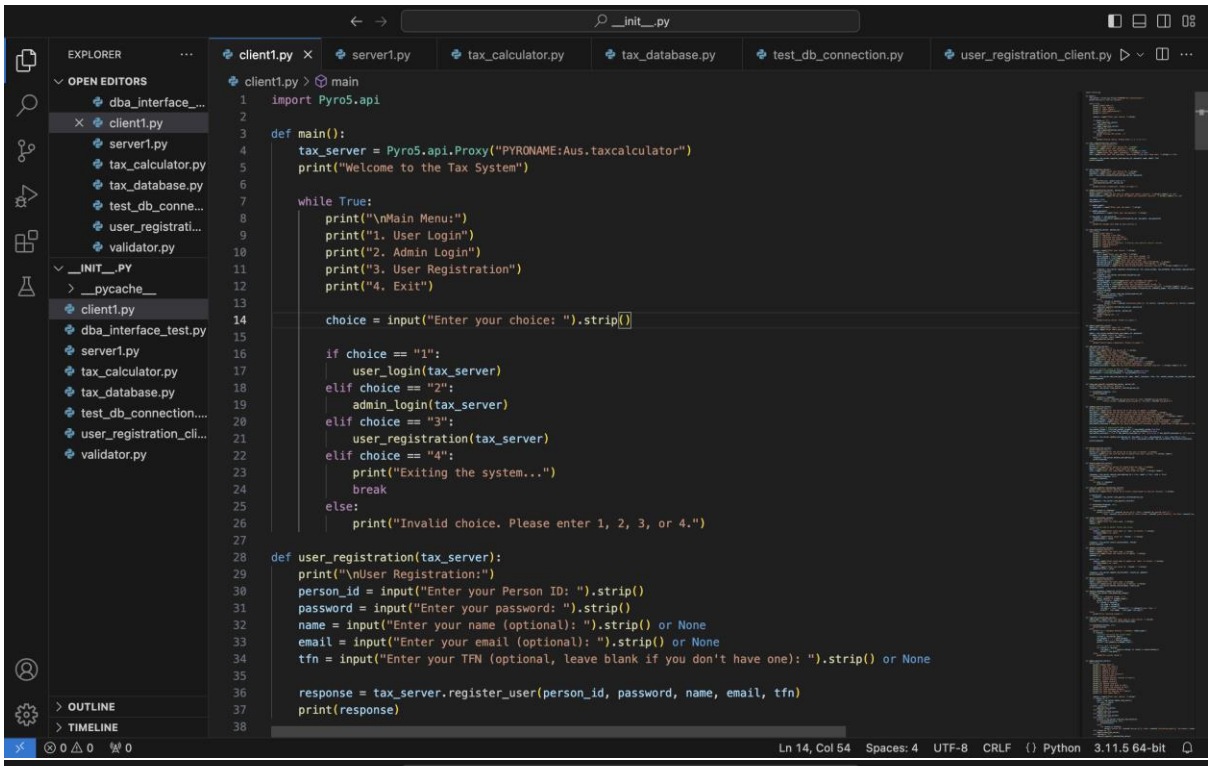
```
mysql> SELECT * FROM users WHERE person_id = 'user110';
+----+-----------+----------+---------------+------------------------+------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name          | email                  | password   | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+---------------+------------------------+------------+------+---------------+--------------+----------------------+
| 20 | user110   | 47502438 | Kelly Reynolds | kelly.reynolds@gmail.com | userpass110 | user |          NULL |         NULL |                 NULL |
+----+-----------+----------+---------------+------------------------+------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

Third Snapshot depicts a user registration with a TFN, who is already registered in the database as a TFN holder returns an error if the TFN number of the user already exists

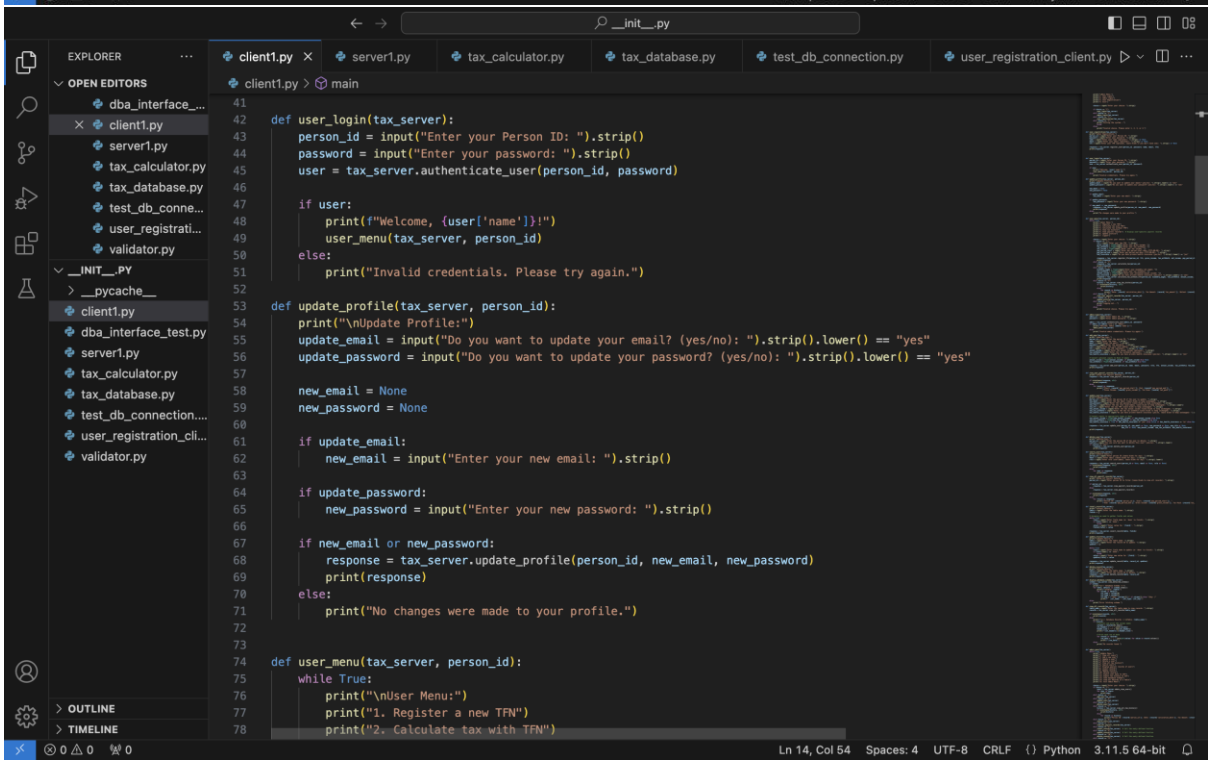
```
Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 3

User Registration:
Enter your Person ID: user101
Enter your password: userpass101
Enter your name (optional): Alice Johnson
Enter your email (optional): alice.johnson@gmail.com
Enter your TFN (optional, leave blank if you don't have one):
Error: User with person_id 'user101' already exists.
```

Code Documentation Phase 2

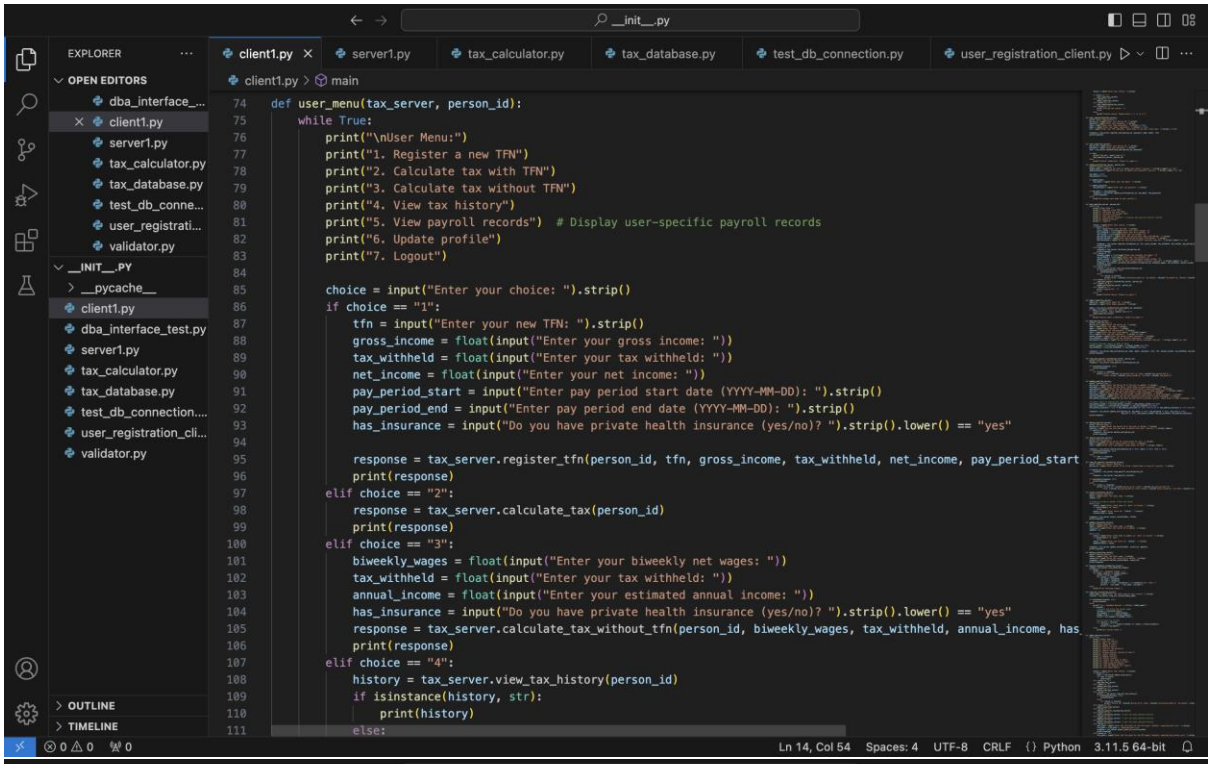
Client1.py



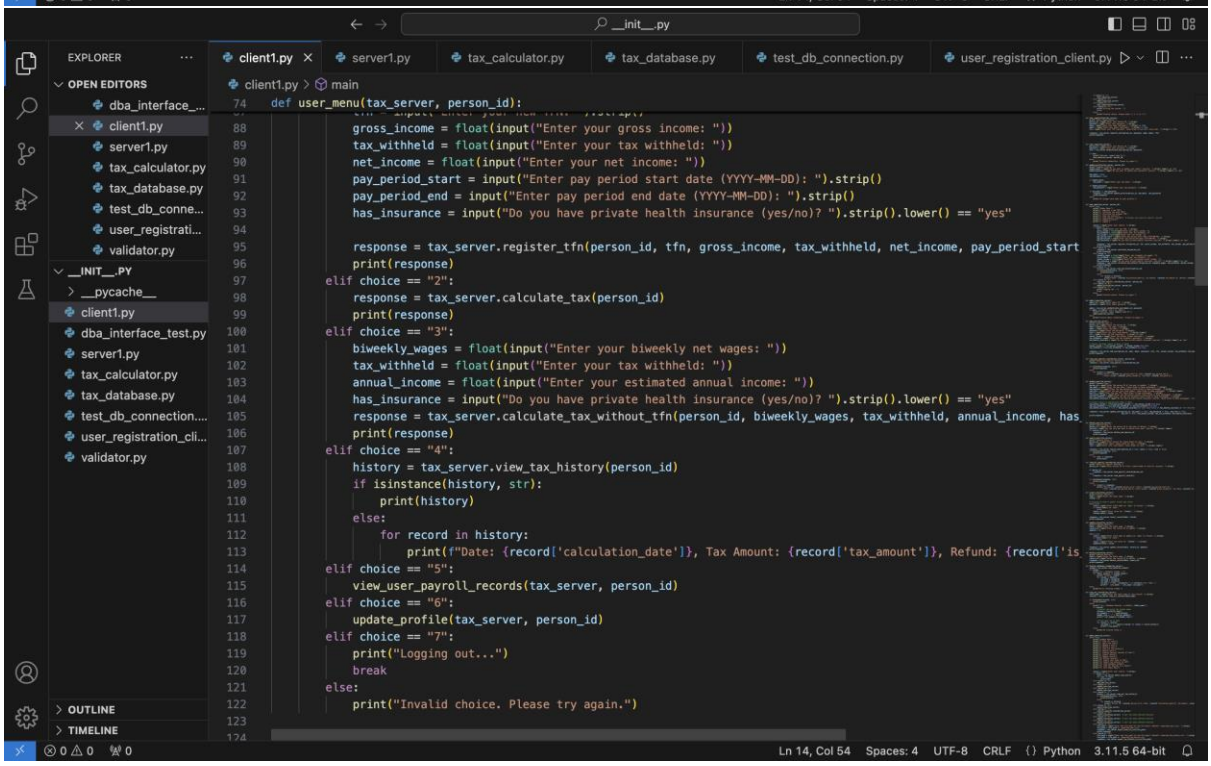
```python
import Pyro5.api

def main():
    tax_server = Pyro5.api.Proxy("PYRONAME:Aus.taxcalculator")
    print("Welcome to the Tax System")

    while True:
        print("\nMain Menu:")
        print("1. User Login")
        print("2. Admin Login")
        print("3. User Registration")
        print("4. Exit")

        choice = input("Enter your choice: ").strip()

        if choice == "1":
            user_login(tax_server)
        elif choice == "2":
            admin_login(tax_server)
        elif choice == "3":
            user_registration(tax_server)
        elif choice == "4":
            print("Exiting the system...")
            break
        else:
            print("Invalid choice. Please enter 1, 2, 3, or 4.")

def user_registration(tax_server):
    print("\nUser Registration:")
    person_id = input("Enter your Person ID: ").strip()
    password = input("Enter your password: ").strip()
    name = input("Enter your name (optional): ").strip() or None
    email = input("Enter your email (optional): ").strip() or None
    tfn = input("Enter your TFN (optional, leave blank if you don't have one): ").strip() or None

    response = tax_server.register_user(person_id, password, name, email, tfn)
    print(response)
```



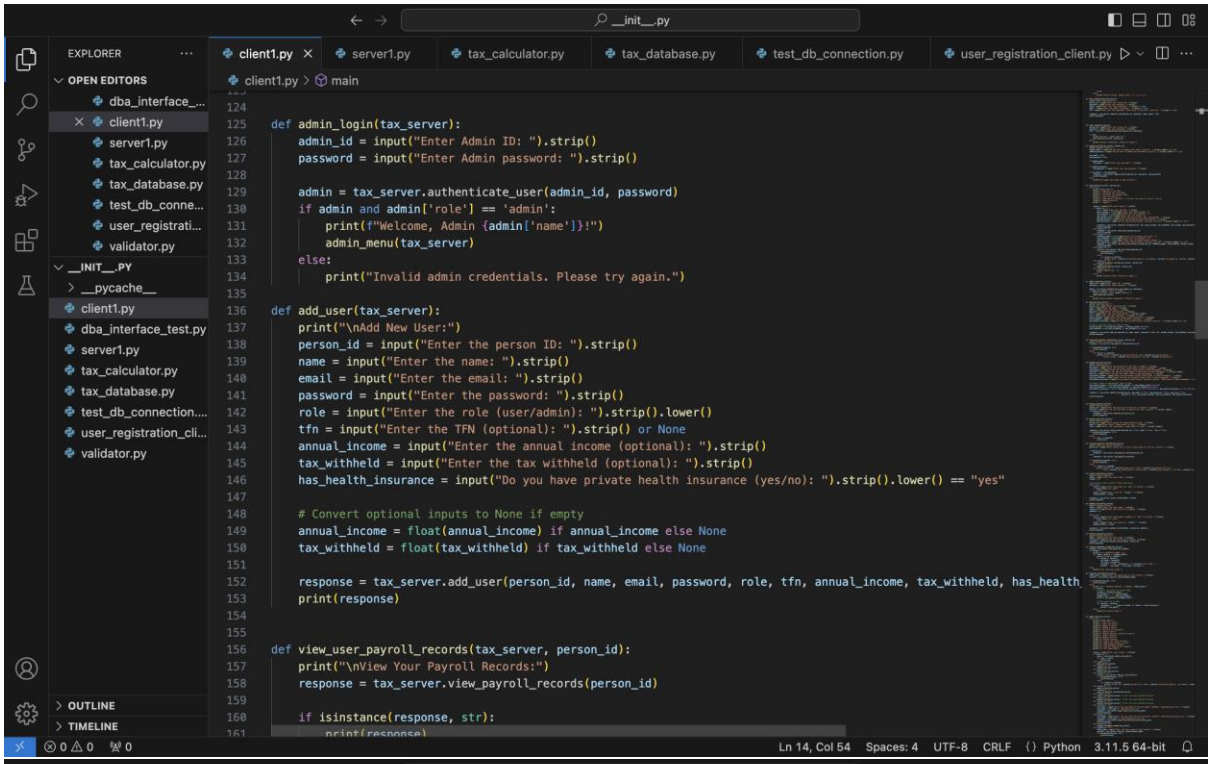
```python
def user_login(tax_server):
    person_id = input("Enter your Person ID: ").strip()
    password = input("Enter your password: ").strip()
    user = tax_server.authenticate_user(person_id, password)

    if user:
        print(f"Welcome, {user['name']}!")
        user_menu(tax_server, person_id)
    else:
        print("Invalid credentials. Please try again.")

def update_profile(tax_server, person_id):
    print("\nUpdate Profile:")
    update_email = input("Do you want to update your email? (yes/no): ").strip().lower() == "yes"
    update_password = input("Do you want to update your password? (yes/no): ").strip().lower() == "yes"

    new_email = None
    new_password = None

    if update_email:
        new_email = input("Enter your new email: ").strip()

    if update_password:
        new_password = input("Enter your new password: ").strip()

    if new_email or new_password:
        response = tax_server.update_profile(person_id, new_email, new_password)
        print(response)
    else:
        print("No changes were made to your profile.")


def user_menu(tax_server, person_id):
    while True:
        print("\nUser Menu:")
        print("1. Register a new TFN")
        print("2. Calculate tax with TFN")
```



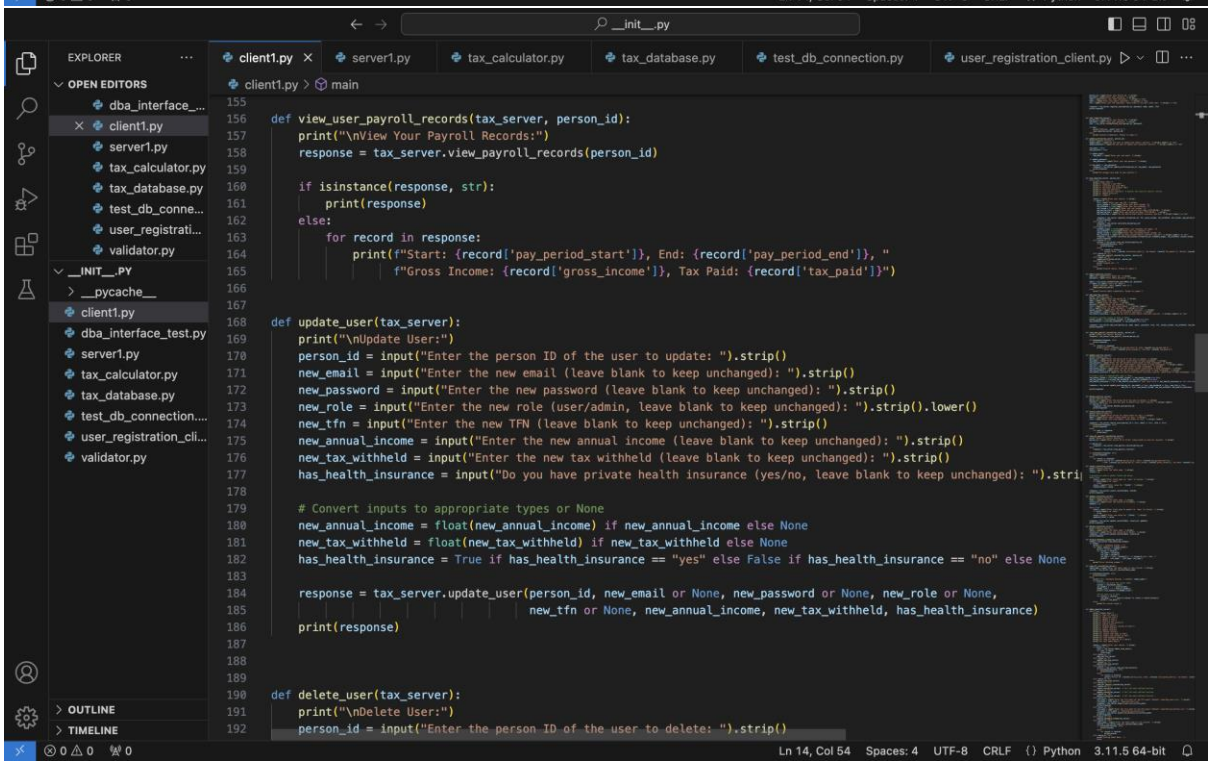
```python
def user_menu(tax_server, person_id):
    while True:
        print("\nUser Menu:")
        print("1. Register a new TFN")
        print("2. Calculate tax with TFN")
        print("3. Calculate tax without TFN")
        print("4. View tax history")
        print("5. View payroll records")  # Display user-specific payroll records
        print("6. Update profile")
        print("7. Logout")

        choice = input("Enter your choice: ").strip()
        if choice == "1":
            tfn = input("Enter your new TFN: ").strip()
            gross_income = float(input("Enter your gross income: "))
            tax_withheld = float(input("Enter your tax withheld: "))
            net_income = float(input("Enter your net income: "))
            pay_period_start = input("Enter pay period start date (YYYY-MM-DD): ").strip()
            pay_period_end = input("Enter pay period end date (YYYY-MM-DD): ").strip()
            has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"

            response = tax_server.register_tfn(person_id, tfn, gross_income, tax_withheld, net_income, pay_period_start
            print(response)
        elif choice == "2":
            response = tax_server.calculate_tax(person_id)
            print(response)
        elif choice == "3":
            biweekly_wages = float(input("Enter your biweekly net wages: "))
            tax_withheld = float(input("Enter your tax withheld: "))
            annual_income = float(input("Enter your estimated annual income: "))
            has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
            response = tax_server.calculate_tax_without_tfn(person_id, biweekly_wages, tax_withheld, annual_income, has
            print(response)
        elif choice == "4":
            history = tax_server.view_tax_history(person_id)
            if isinstance(history, str):
                print(history)
            else:
```



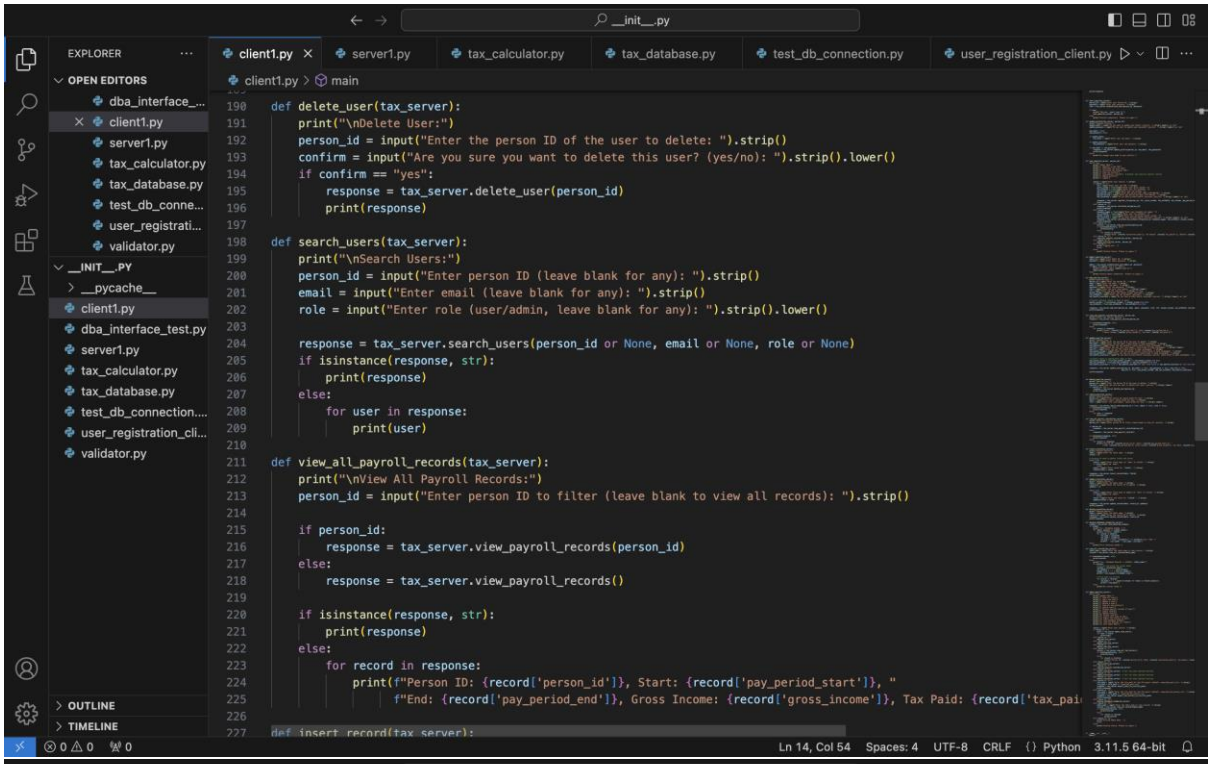
```python
def user_menu(tax_server, person_id):
            gross_income = float(input("Enter your gross income: "))
            tax_withheld = float(input("Enter your tax withheld: "))
            net_income = float(input("Enter your net income: "))
            pay_period_start = input("Enter pay period start date (YYYY-MM-DD): ").strip()
            pay_period_end = input("Enter pay period end date (YYYY-MM-DD): ").strip()
            has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"

            response = tax_server.register_tfn(person_id, tfn, gross_income, tax_withheld, net_income, pay_period_start
            print(response)
        elif choice == "2":
            response = tax_server.calculate_tax(person_id)
            print(response)
        elif choice == "3":
            biweekly_wages = float(input("Enter your biweekly net wages: "))
            tax_withheld = float(input("Enter your tax withheld: "))
            annual_income = float(input("Enter your estimated annual income: "))
            has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
            response = tax_server.calculate_tax_without_tfn(person_id, biweekly_wages, tax_withheld, annual_income, has
            print(response)
        elif choice == "4":
            history = tax_server.view_tax_history(person_id)
            if isinstance(history, str):
                print(history)
            else:
                for record in history:
                    print(f"Date: {record['calculation_date']}, Tax Amount: {record['tax_amount']}, Refund: {record['is
        elif choice == "5":
            view_user_payroll_records(tax_server, person_id)
        elif choice == "6":
            update_profile(tax_server, person_id)
        elif choice == "7":
            print("Logging out...")
            break
        else:
            print("Invalid choice. Please try again.")
```



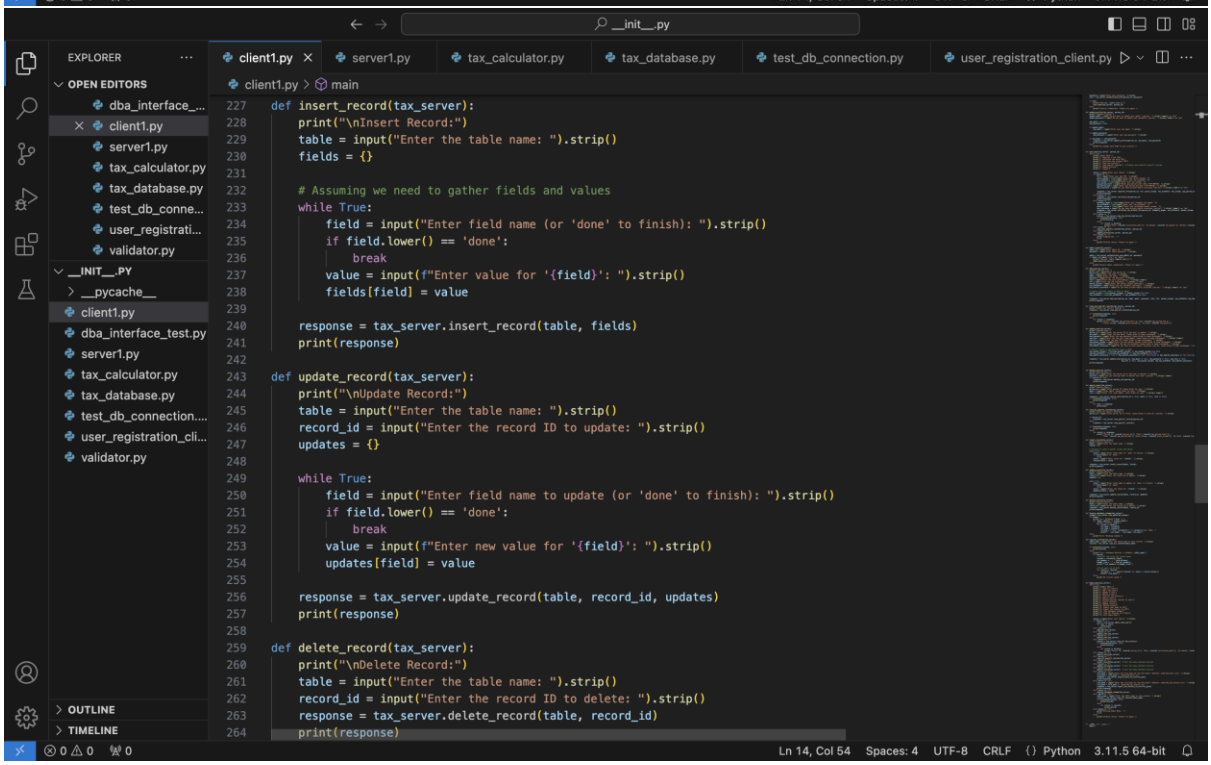
```python
def admin_login(tax_server):
    admin_id = input("Enter Admin ID: ").strip()
    password = input("Enter Admin password: ").strip()

    admin = tax_server.authenticate_user(admin_id, password)
    if admin and admin['role'] == 'admin':
        print(f"Welcome, Admin {admin['name']}!")
        admin_menu(tax_server)
    else:
        print("Invalid admin credentials. Please try again.")

def add_user(tax_server):
    print("\nAdd New User:")
    person_id = input("Enter the person ID: ").strip()
    name = input("Enter the name: ").strip()
    email = input("Enter the email: ").strip()
    password = input("Enter the password: ").strip()
    role = input("Enter the role (user/admin): ").strip().lower()
    tfn = input("Enter the TFN (optional): ").strip() or None
    annual_income = input("Enter the annual income (optional): ").strip()
    tax_withheld = input("Enter the tax withheld (optional): ").strip()
    has_health_insurance = input("Do you have private health insurance (yes/no): ").strip().lower() == "yes"

    # Convert optional inputs to None if empty
    annual_income = float(annual_income) if annual_income else None
    tax_withheld = float(tax_withheld) if tax_withheld else None

    response = tax_server.add_user(person_id, name, email, password, role, tfn, annual_income, tax_withheld, has_health
    print(response)


def view_user_payroll_records(tax_server, person_id):
    print("\nView Your Payroll Records:")
    response = tax_server.view_payroll_records(person_id)

    if isinstance(response, str):
```



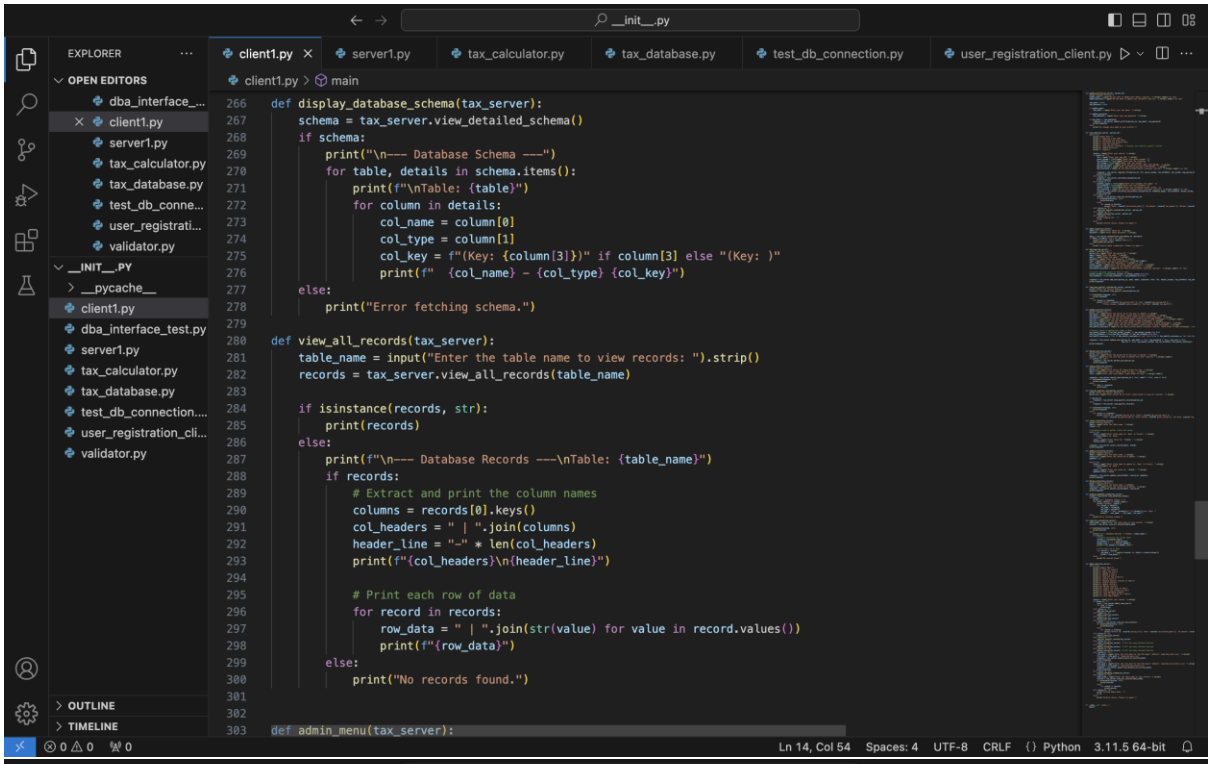
```python
def view_user_payroll_records(tax_server, person_id):
    print("\nView Your Payroll Records:")
    response = tax_server.view_payroll_records(person_id)

    if isinstance(response, str):
        print(response)
    else:
        for record in response:
            print(f"Start: {record['pay_period_start']}, End: {record['pay_period_end']}, "
                  f"Gross Income: {record['gross_income']}, Tax Paid: {record['tax_paid']}")


def update_user(tax_server):
    print("\nUpdate User:")
    person_id = input("Enter the person ID of the user to update: ").strip()
    new_email = input("Enter the new email (leave blank to keep unchanged): ").strip()
    new_password = input("Enter the new password (leave blank to keep unchanged): ").strip()
    new_role = input("Enter the new role (user/admin, leave blank to keep unchanged): ").strip().lower()
    new_tfn = input("Enter the new TFN (leave blank to keep unchanged): ").strip()
    new_annual_income = input("Enter the new annual income (leave blank to keep unchanged): ").strip()
    new_tax_withheld = input("Enter the new tax withheld (leave blank to keep unchanged): ").strip()
    has_health_insurance = input("Do you have private health insurance (yes/no, leave blank to keep unchanged): ").stri

    # Convert inputs to appropriate types or None
    new_annual_income = float(new_annual_income) if new_annual_income else None
    new_tax_withheld = float(new_tax_withheld) if new_tax_withheld else None
    has_health_insurance = True if has_health_insurance == "yes" else False if has_health_insurance == "no" else None

    response = tax_server.update_user(person_id, new_email or None, new_password or None, new_role or None,
                                      new_tfn or None, new_annual_income, new_tax_withheld, has_health_insurance)
    print(response)


def delete_user(tax_server):
    print("\nDelete User:")
    person_id = input("Enter the person ID of the user to delete: ").strip()
```



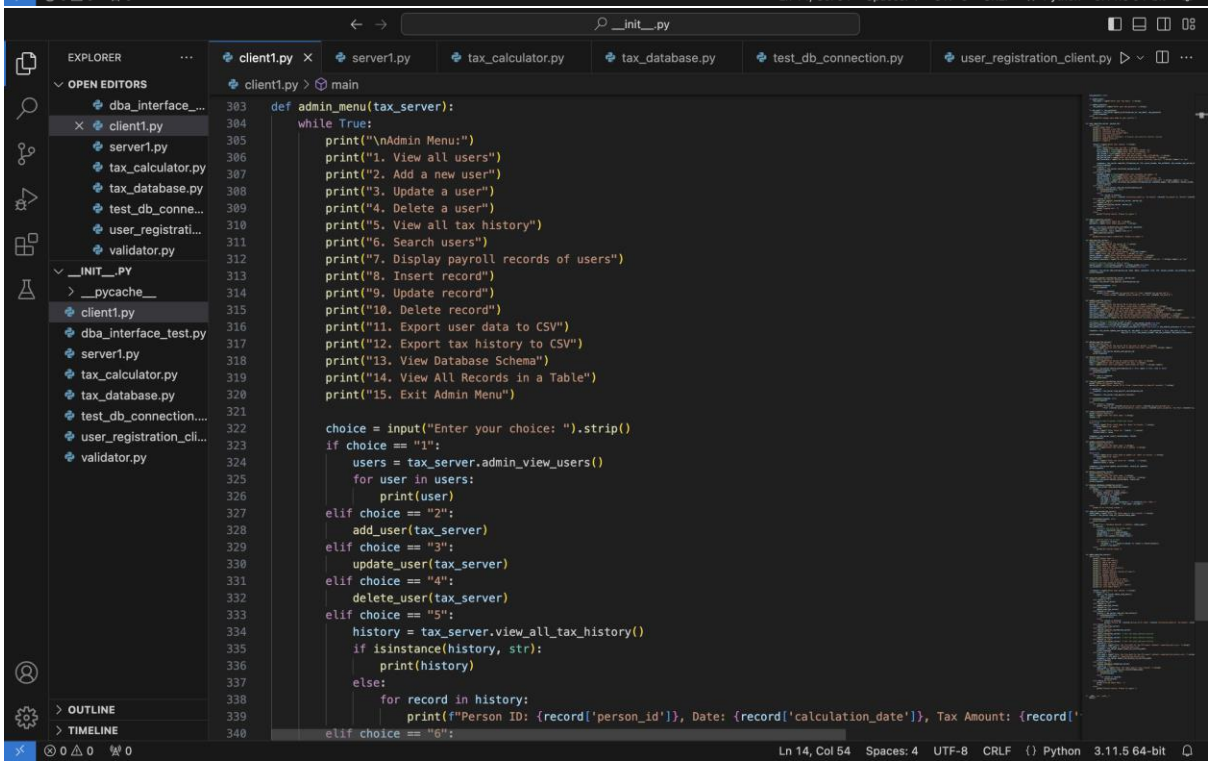
```python
def delete_user(tax_server):
    print("\nDelete User:")
    person_id = input("Enter the person ID of the user to delete: ").strip()
    confirm = input("Are you sure you want to delete this user? (yes/no): ").strip().lower()
    if confirm == "yes":
        response = tax_server.delete_user(person_id)
        print(response)

def search_users(tax_server):
    print("\nSearch Users:")
    person_id = input("Enter person ID (leave blank for any): ").strip()
    email = input("Enter email (leave blank for any): ").strip()
    role = input("Enter role (user/admin, leave blank for any): ").strip().lower()

    response = tax_server.search_users(person_id or None, email or None, role or None)
    if isinstance(response, str):
        print(response)
    else:
        for user in response:
            print(user)

def view_all_payroll_records(tax_server):
    print("\nView All Payroll Records:")
    person_id = input("Enter person ID to filter (leave blank to view all records): ").strip()

    if person_id:
        response = tax_server.view_payroll_records(person_id)
    else:
        response = tax_server.view_payroll_records()

    if isinstance(response, str):
        print(response)
    else:
        for record in response:
            print(f"Person ID: {record['person_id']}, Start: {record['pay_period_start']}, "
                  f"End: {record['pay_period_end']}, Gross Income: {record['gross_income']}, Tax Paid: {record['tax_pai
```



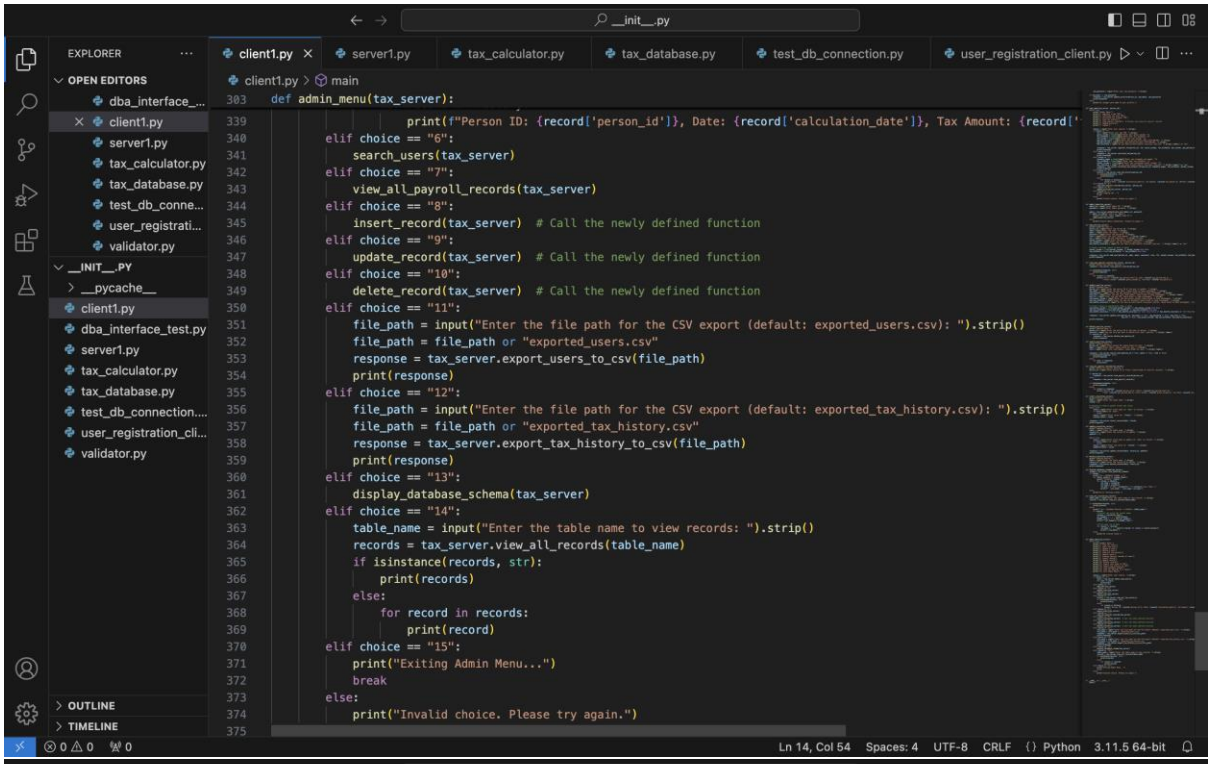
```python
def insert_record(tax_server):
    print("\nInsert Record:")
    table = input("Enter the table name: ").strip()
    fields = {}

    # Assuming we need to gather fields and values
    while True:
        field = input("Enter field name (or 'done' to finish): ").strip()
        if field.lower() == 'done':
            break
        value = input(f"Enter value for '{field}': ").strip()
        fields[field] = value

    response = tax_server.insert_record(table, fields)
    print(response)

def update_record(tax_server):
    print("\nUpdate Record:")
    table = input("Enter the table name: ").strip()
    record_id = input("Enter the record ID to update: ").strip()
    updates = {}

    while True:
        field = input("Enter field name to update (or 'done' to finish): ").strip()
        if field.lower() == 'done':
            break
        value = input(f"Enter new value for '{field}': ").strip()
        updates[field] = value

    response = tax_server.update_record(table, record_id, updates)
    print(response)

def delete_record(tax_server):
    print("\nDelete Record:")
    table = input("Enter the table name: ").strip()
    record_id = input("Enter the record ID to delete: ").strip()
    response = tax_server.delete_record(table, record_id)
    print(response)
```



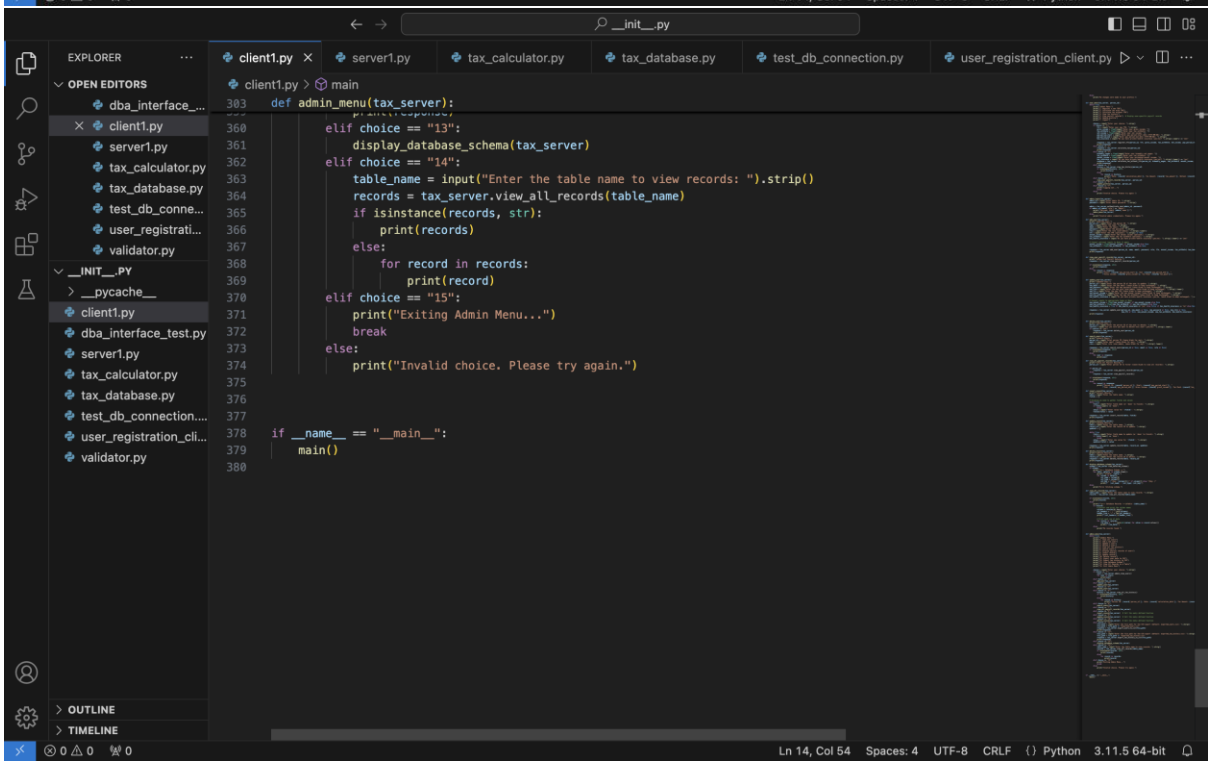
```python
def display_database_schema(tax_server):
    schema = tax_server.view_detailed_schema()
    if schema:
        print("\n--- Database Schema ---")
        for table, details in schema.items():
            print(f"\nTable: {table}")
            for column in details:
                col_name = column[0]
                col_type = column[1]
                col_key = f"(Key: {column[3]})" if column[3] else "(Key: )"
                print(f"  {col_name} - {col_type} {col_key}")
    else:
        print("Error fetching schema.")

def view_all_records(tax_server):
    table_name = input("Enter the table name to view records: ").strip()
    records = tax_server.view_all_records(table_name)

    if isinstance(records, str):
        print(records)
    else:
        print(f"\n--- Database Records ---\nTable: {table_name}")
        if records:
            # Extract and print the column names
            columns = records[0].keys()
            col_headers = " | ".join(columns)
            header_line = "-" * len(col_headers)
            print(f"{col_headers}\n{header_line}")

            # Print each row of data
            for record in records:
                row_data = " | ".join(str(value) for value in record.values())
                print(f"{row_data}")
        else:
            print("No records found.")


def admin_menu(tax_server):
```



```python
def admin_menu(tax_server):
    while True:
        print("\nAdmin Menu:")
        print("1. View all users")
        print("2. Add a new user")
        print("3. Update a user")
        print("4. Delete a user")
        print("5. View all tax history")
        print("6. Search users")
        print("7. Display payroll records of users")
        print("8. Insert record")
        print("9. Update record")
        print("10. Delete record")
        print("11. Export user data to CSV")
        print("12. Export tax history to CSV")
        print("13. View Database Schema")
        print("14. View All Records in a Table")
        print("15. Exit Admin Menu")

        choice = input("Enter your choice: ").strip()
        if choice == "1":
            users = tax_server.admin_view_users()
            for user in users:
                print(user)
        elif choice == "2":
            add_user(tax_server)
        elif choice == "3":
            update_user(tax_server)
        elif choice == "4":
            delete_user(tax_server)
        elif choice == "5":
            history = tax_server.view_all_tax_history()
            if isinstance(history, str):
                print(history)
            else:
                for record in history:
                    print(f"Person ID: {record['person_id']}, Date: {record['calculation_date']}, Tax Amount: {record['
        elif choice == "6":
```

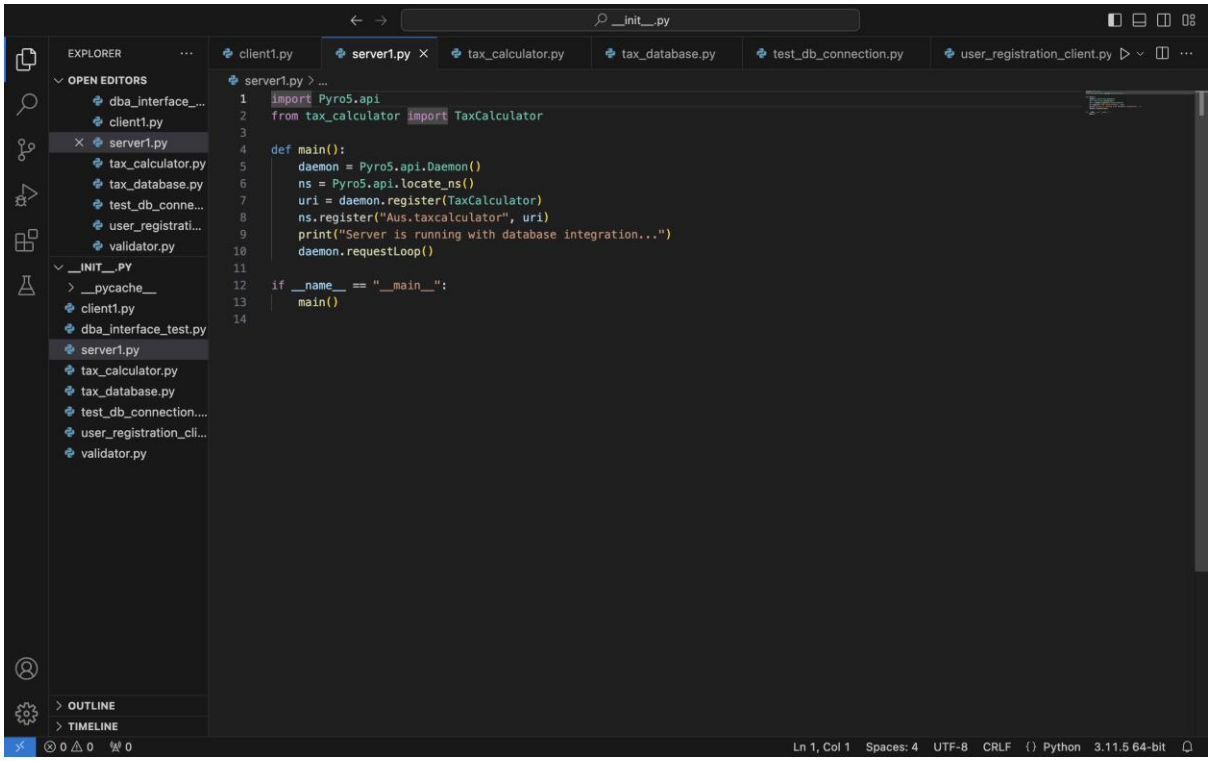


```python
def admin_menu(tax_server):
                print(f"Person ID: {record['person_id']}, Date: {record['calculation_date']}, Tax Amount: {record['
        elif choice == "6":
            search_users(tax_server)
        elif choice == "7":
            view_all_payroll_records(tax_server)
        elif choice == "8":
            insert_record(tax_server)  # Call the newly defined function
        elif choice == "9":
            update_record(tax_server)  # Call the newly defined function
        elif choice == "10":
            delete_record(tax_server)  # Call the newly defined function
        elif choice == "11":
            file_path = input("Enter the file path for the CSV export (default: exported_users.csv): ").strip()
            file_path = file_path or "exported_users.csv"
            response = tax_server.export_users_to_csv(file_path)
            print(response)
        elif choice == "12":
            file_path = input("Enter the file path for the CSV export (default: exported_tax_history.csv): ").strip()
            file_path = file_path or "exported_tax_history.csv"
            response = tax_server.export_tax_history_to_csv(file_path)
            print(response)
        elif choice == "13":
            display_database_schema(tax_server)
        elif choice == "14":
            table_name = input("Enter the table name to view records: ").strip()
            records = tax_server.view_all_records(table_name)
            if isinstance(records, str):
                print(records)
            else:
                for record in records:
                    print(record)
        elif choice == "15":
            print("Exiting Admin Menu...")
            break
        else:
            print("Invalid choice. Please try again.")
```

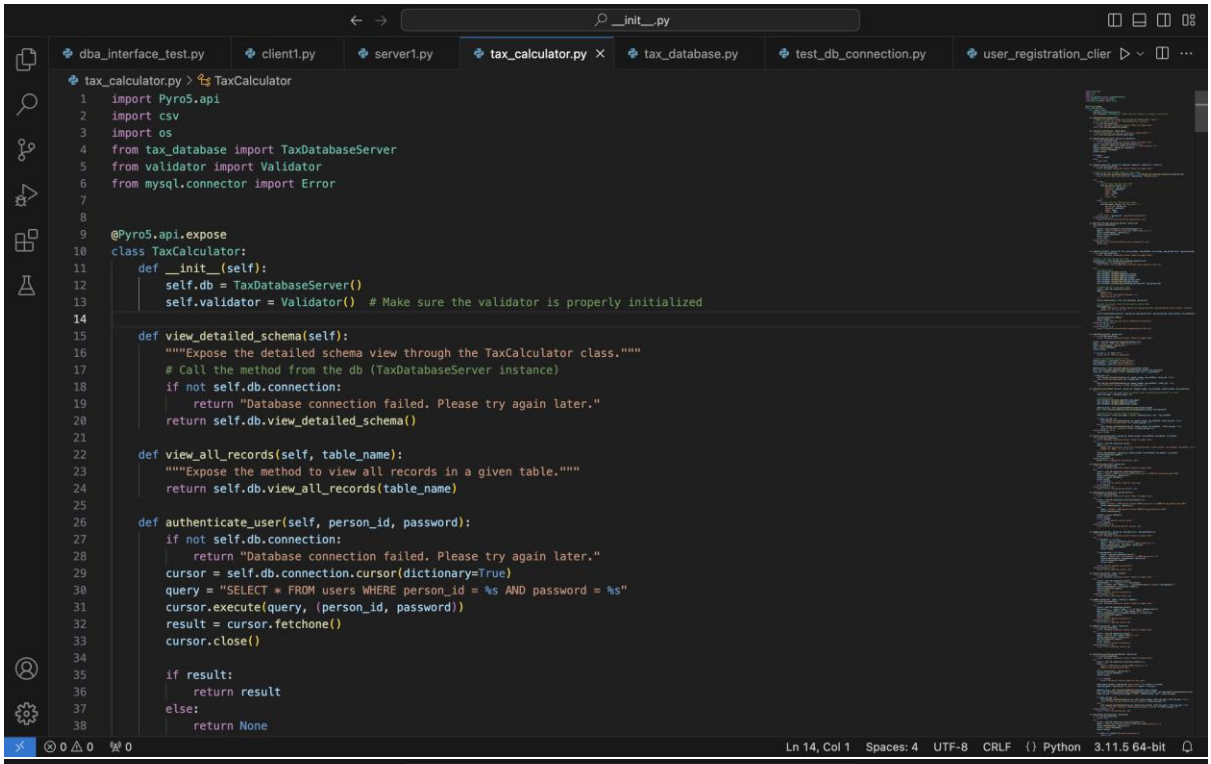


```python
def admin_menu(tax_server):
        elif choice == "13":
            display_database_schema(tax_server)
        elif choice == "14":
            table_name = input("Enter the table name to view records: ").strip()
            records = tax_server.view_all_records(table_name)
            if isinstance(records, str):
                print(records)
            else:
                for record in records:
                    print(record)
        elif choice == "15":
            print("Exiting Admin Menu...")
            break
        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    main()
```

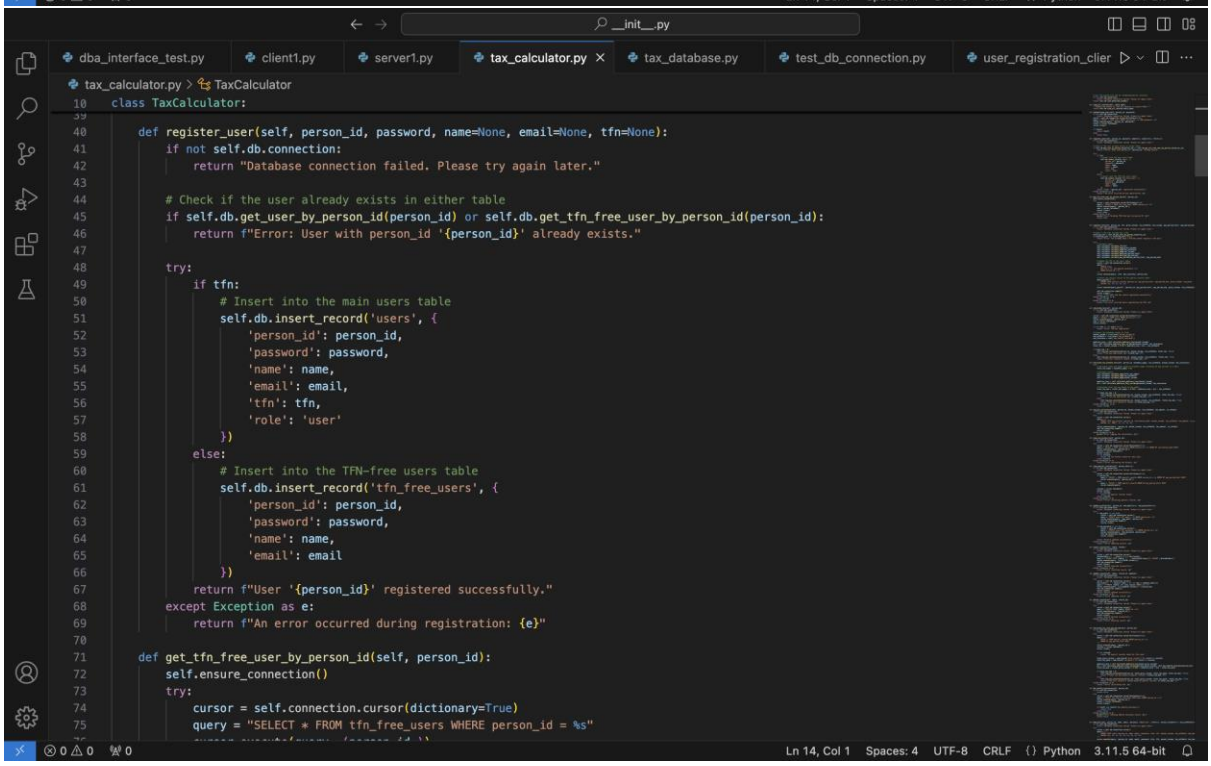
server1.py



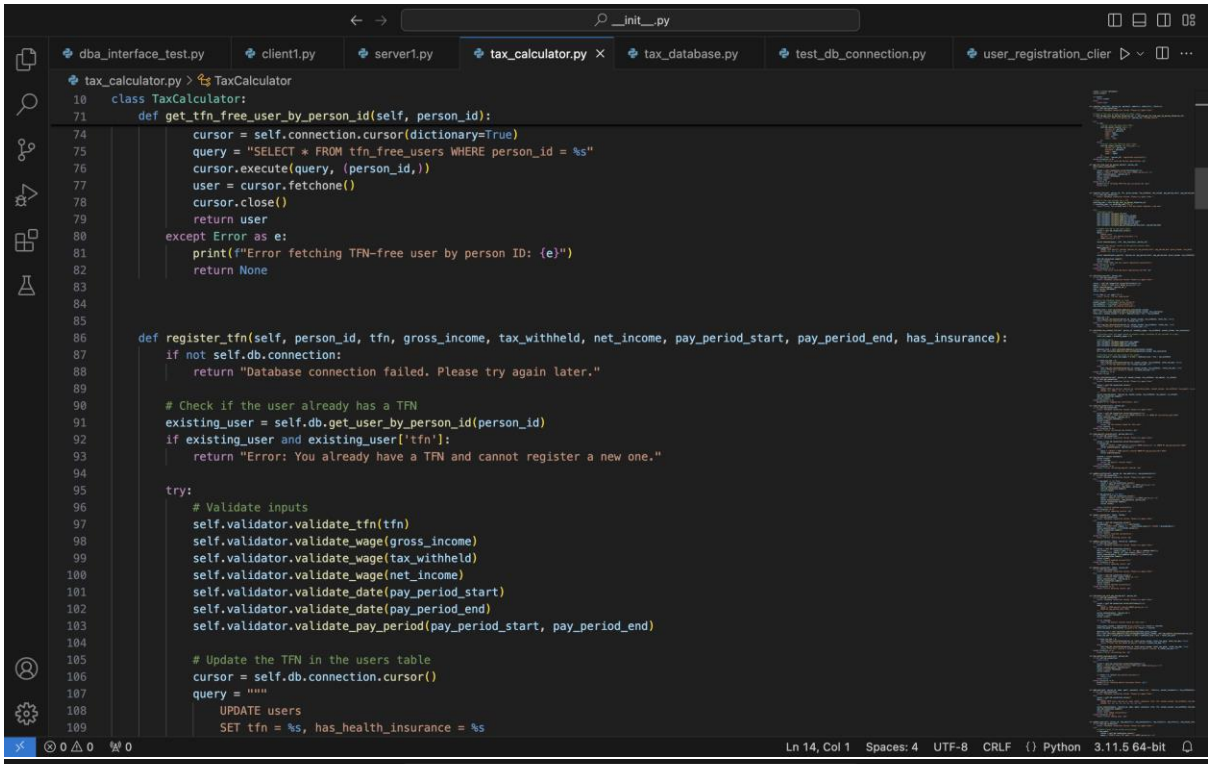
tax_calculator.py



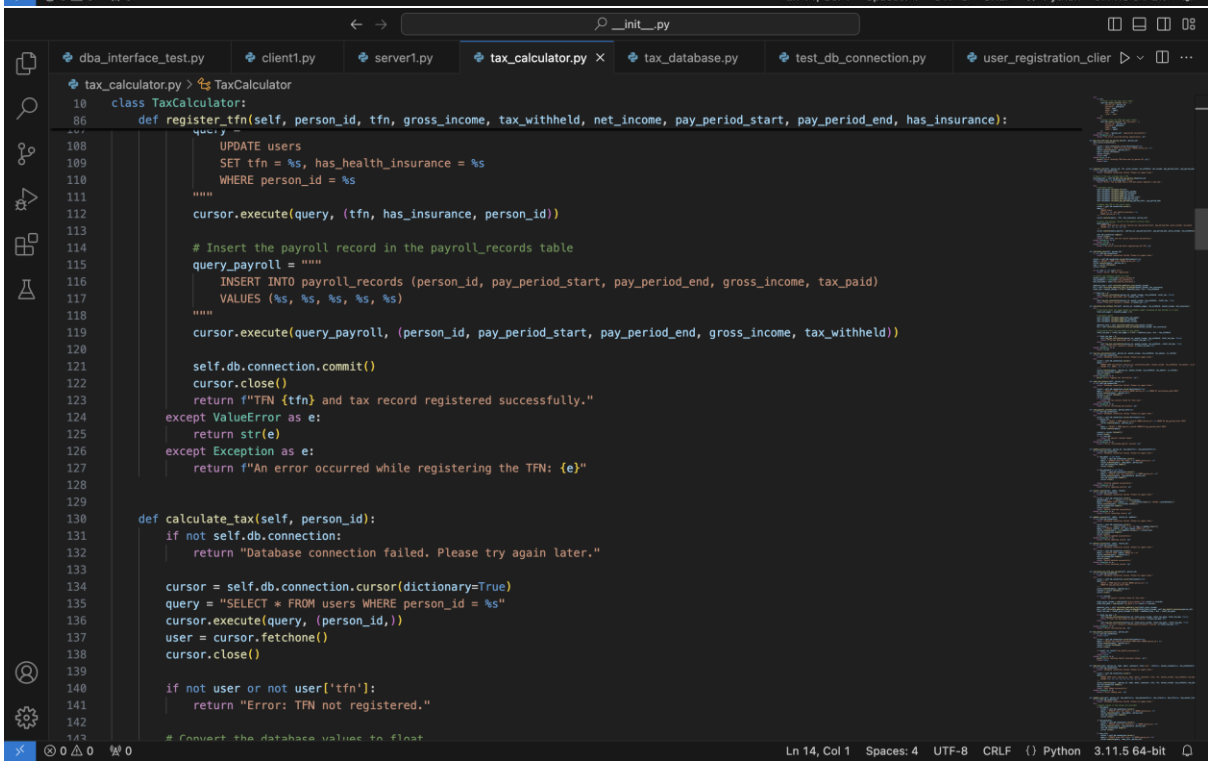
```python
import Pyro5.api
import csv
import os
from tax_database import TaxDatabaseServer
from validator import Validator
from mysql.connector import Error


@Pyro5.api.expose
class TaxCalculator:
    def __init__(self):
        self.db = TaxDatabaseServer()
        self.validator = Validator()  # Make sure the validator is properly initialized

    def view_detailed_schema(self):
        """Expose the detailed schema view through the TaxCalculator class."""
        # Call the method from the db (TaxDatabaseServer instance)
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        return self.db.view_detailed_schema()

    def view_all_records(self, table_name):
        """Expose the method to view all records in a given table."""
        return self.db.view_all_records(table_name)

    def authenticate_user(self, person_id, password):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        cursor = self.db.connection.cursor(dictionary=True)
        query = "SELECT * FROM users WHERE person_id = %s AND password = %s"
        cursor.execute(query, (person_id, password))
        result = cursor.fetchone()
        cursor.close()

        if result:
            return result
        else:
            return None
```



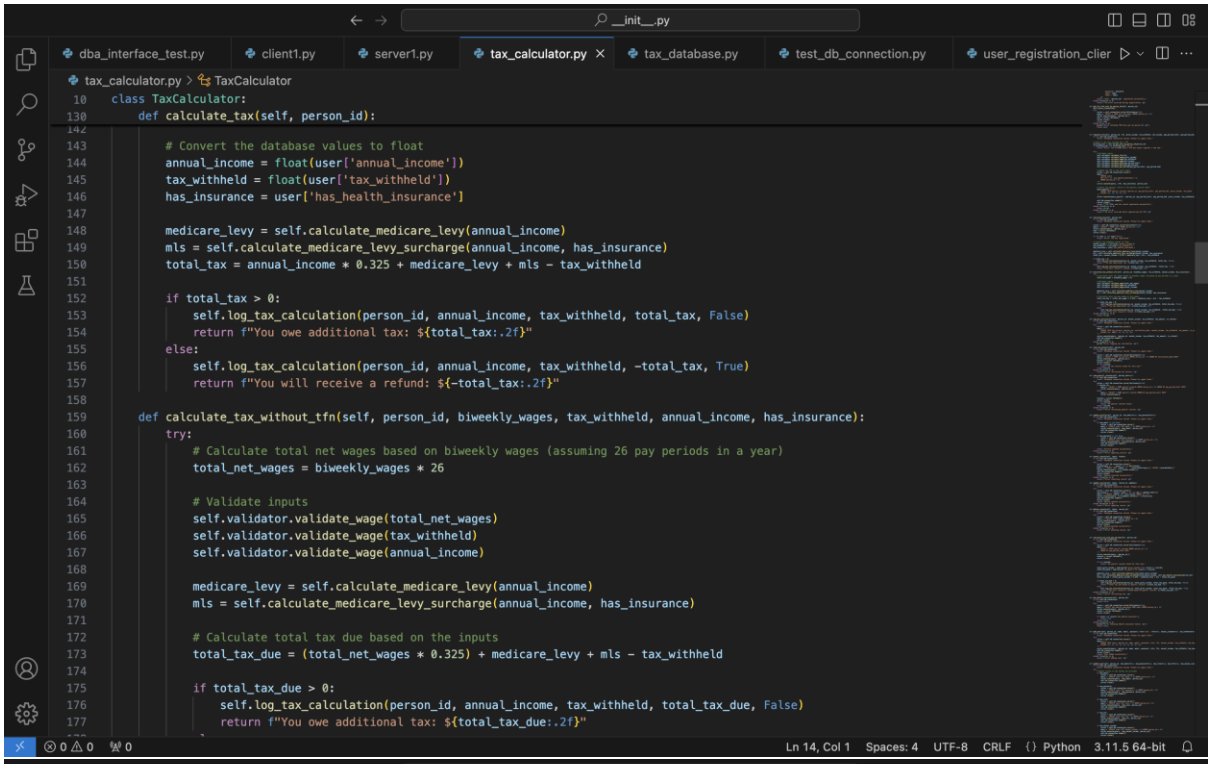
```python
class TaxCalculator:

    def register_user(self, person_id, password, name=None, email=None, tfn=None):
        if not self.db.connection:
            return "Database connection failed. Please try again later."

        # Check if the user already exists in either table
        if self.db.get_user_by_person_id(person_id) or self.db.get_tfn_free_user_by_person_id(person_id):
            return f"Error: User with person_id '{person_id}' already exists."

        try:
            if tfn:
                # Insert into the main users table
                self.db.insert_record('users', {
                    'person_id': person_id,
                    'password': password,
                    'name': name,
                    'email': email,
                    'tfn': tfn,
                    'role': 'user'
                })
            else:
                # Insert into the TFN-free users table
                self.db.insert_record('tfn_free_users', {
                    'person_id': person_id,
                    'password': password,
                    'name': name,
                    'email': email
                })
            return f"User '{person_id}' registered successfully."
        except Exception as e:
            return f"An error occurred during registration: {e}"

    def get_tfn_free_user_by_person_id(self, person_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM tfn_free_users WHERE person_id = %s"
```



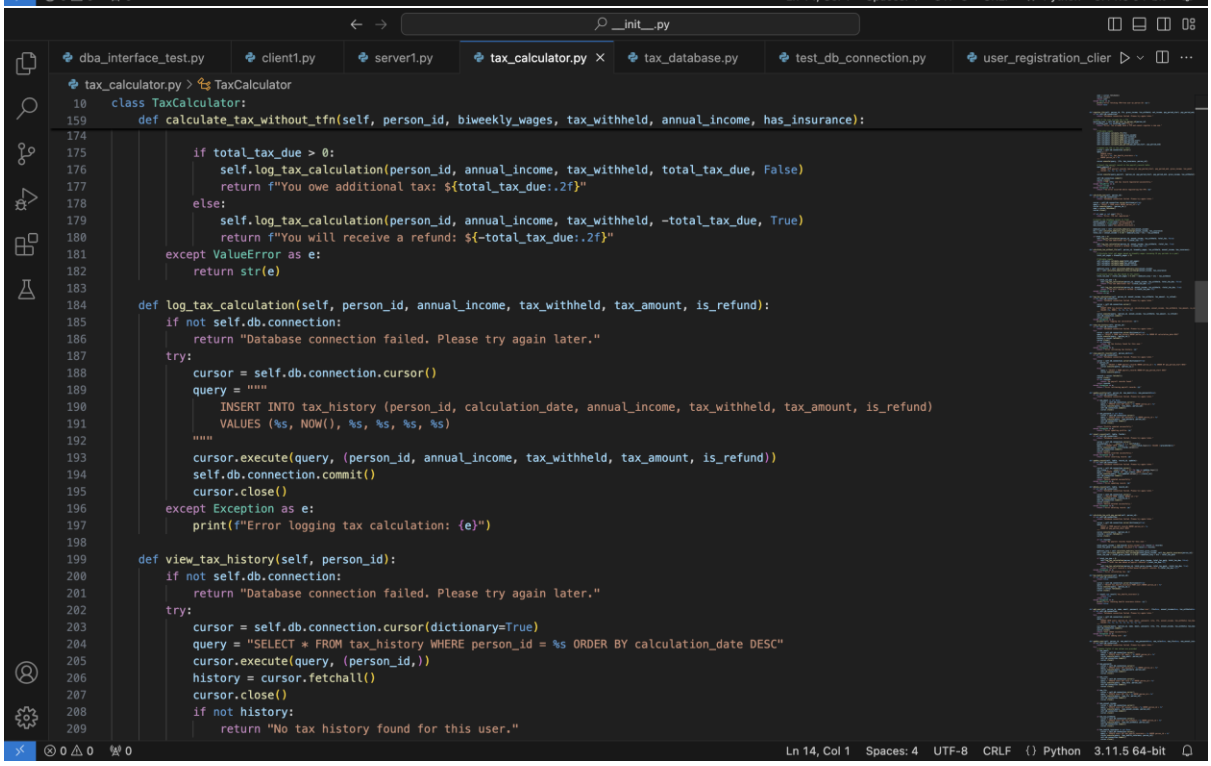
```python
class TaxCalculator:
    def get_tfn_free_user_by_person_id(self, person_id):
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM tfn_free_users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            user = cursor.fetchone()
            cursor.close()
            return user
        except Error as e:
            print(f"Error fetching TFN-free user by person ID: {e}")
            return None



    def register_tfn(self, person_id, tfn, gross_income, tax_withheld, net_income, pay_period_start, pay_period_end, has_insurance):
        if not self.db.connection:
            return "Database connection failed. Please try again later."

        # Check if the user already has a TFN
        existing_user = self.db.get_user_by_person_id(person_id)
        if existing_user and existing_user['tfn']:
            return "Error: You already have a TFN and cannot register a new one."

        try:
            # Validate inputs
            self.validator.validate_tfn(tfn)
            self.validator.validate_wage(gross_income)
            self.validator.validate_wage(tax_withheld)
            self.validator.validate_wage(net_income)
            self.validator.validate_date(pay_period_start)
            self.validator.validate_date(pay_period_end)
            self.validator.validate_pay_period(pay_period_start, pay_period_end)

            # Update the TFN in the users table
            cursor = self.db.connection.cursor()
            query = """
                UPDATE users
                SET tfn = %s, has_health_insurance = %s
```



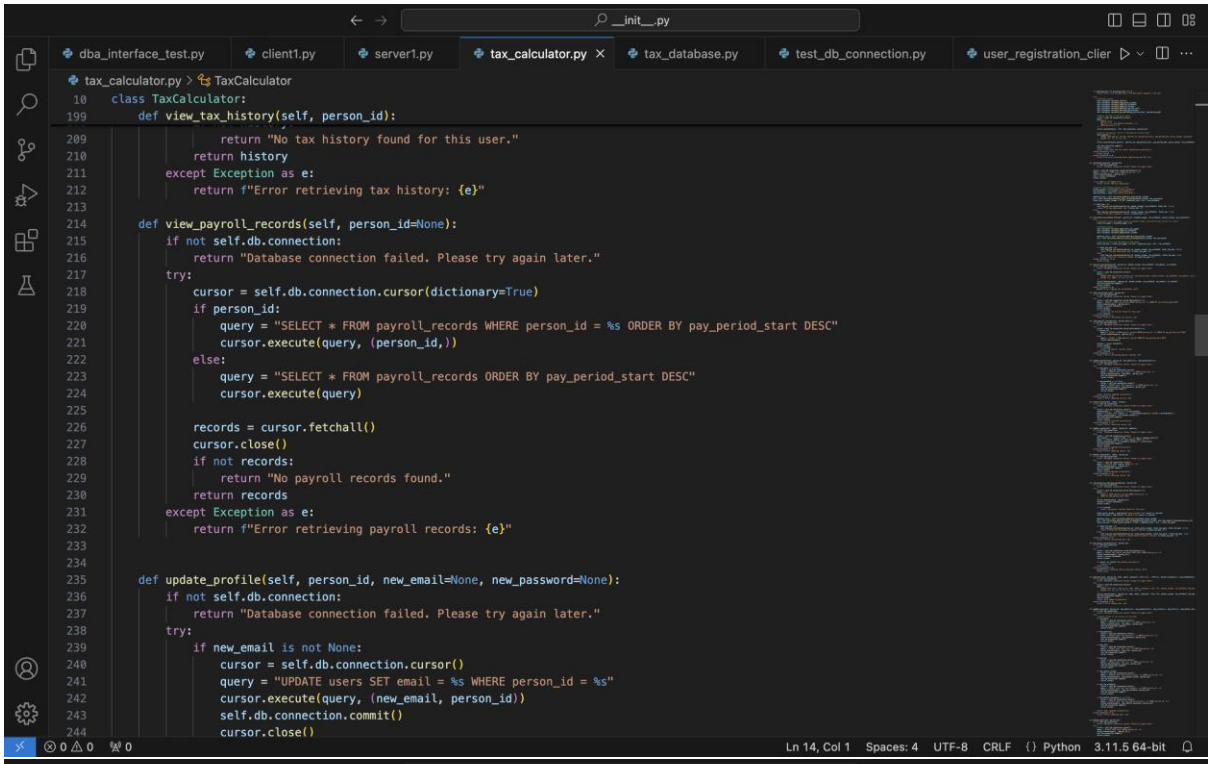
```python
class TaxCalculator:
    def register_tfn(self, person_id, tfn, gross_income, tax_withheld, net_income, pay_period_start, pay_period_end, has_insurance):
                UPDATE users
                SET tfn = %s, has_health_insurance = %s
                WHERE person_id = %s
            """
            cursor.execute(query, (tfn, has_insurance, person_id))

            # Insert the payroll record in the payroll_records table
            query_payroll = """
                INSERT INTO payroll_records (person_id, pay_period_start, pay_period_end, gross_income, tax_paid)
                VALUES (%s, %s, %s, %s, %s)
            """
            cursor.execute(query_payroll, (person_id, pay_period_start, pay_period_end, gross_income, tax_withheld))

            self.db.connection.commit()
            cursor.close()
            return f"TFN {tfn} and tax record registered successfully."
        except ValueError as e:
            return str(e)
        except Exception as e:
            return f"An error occurred while registering the TFN: {e}"


    def calculate_tax(self, person_id):
        if not self.db.connection:
            return "Database connection failed. Please try again later."

        cursor = self.db.connection.cursor(dictionary=True)
        query = "SELECT * FROM users WHERE person_id = %s"
        cursor.execute(query, (person_id,))
        user = cursor.fetchone()
        cursor.close()

        if not user or not user['tfn']:
            return "Error: TFN not registered."

        # Convert the database values to float
```



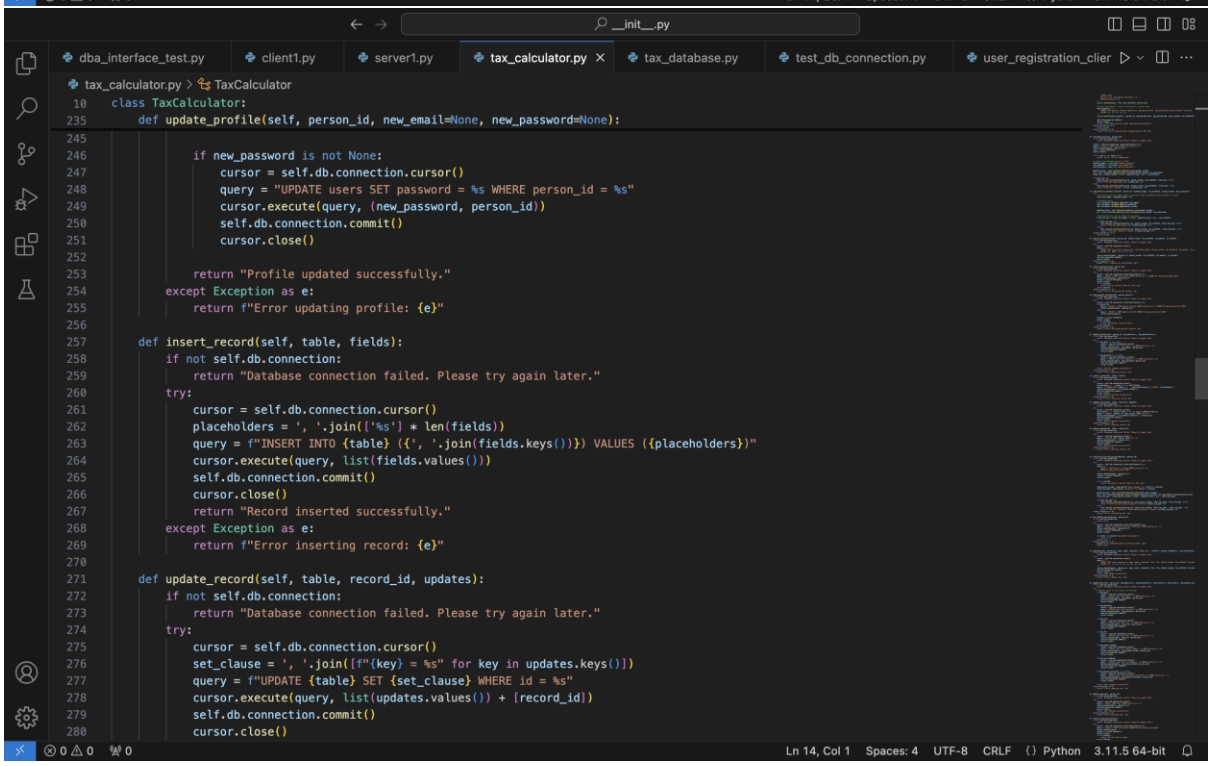
```
class TaxCalculator:
    def calculate_tax(self, person_id):

        # Convert the database values to float
        annual_income = float(user['annual_income'])
        tax_withheld = float(user['tax_withheld'])
        has_insurance = user['has_health_insurance']

        medicare_levy = self.calculate_medicare_levy(annual_income)
        mls = self.calculate_medicare_levy_surcharge(annual_income, has_insurance)
        total_tax = annual_income * 0.325 + medicare_levy + mls - tax_withheld

        if total_tax > 0:
            self.log_tax_calculation(person_id, annual_income, tax_withheld, total_tax, False)
            return f"You owe additional tax: ${total_tax:.2f}"
        else:
            self.log_tax_calculation(person_id, annual_income, tax_withheld, -total_tax, True)
            return f"You will receive a refund: ${-total_tax:.2f}"

    def calculate_tax_without_tfn(self, person_id, biweekly_wages, tax_withheld, annual_income, has_insurance):
        try:
            # Calculate total net wages based on biweekly wages (assuming 26 pay periods in a year)
            total_net_wages = biweekly_wages * 26

            # Validate inputs
            self.validator.validate_wage(total_net_wages)
            self.validator.validate_wage(tax_withheld)
            self.validator.validate_wage(annual_income)

            medicare_levy = self.calculate_medicare_levy(annual_income)
            mls = self.calculate_medicare_levy_surcharge(annual_income, has_insurance)

            # Calculate total tax due based on the inputs
            total_tax_due = (total_net_wages * 0.325) + medicare_levy + mls - tax_withheld

            if total_tax_due > 0:
                self.log_tax_calculation(person_id, annual_income, tax_withheld, total_tax_due, False)
                return f"You owe additional tax: ${total_tax_due:.2f}"
```



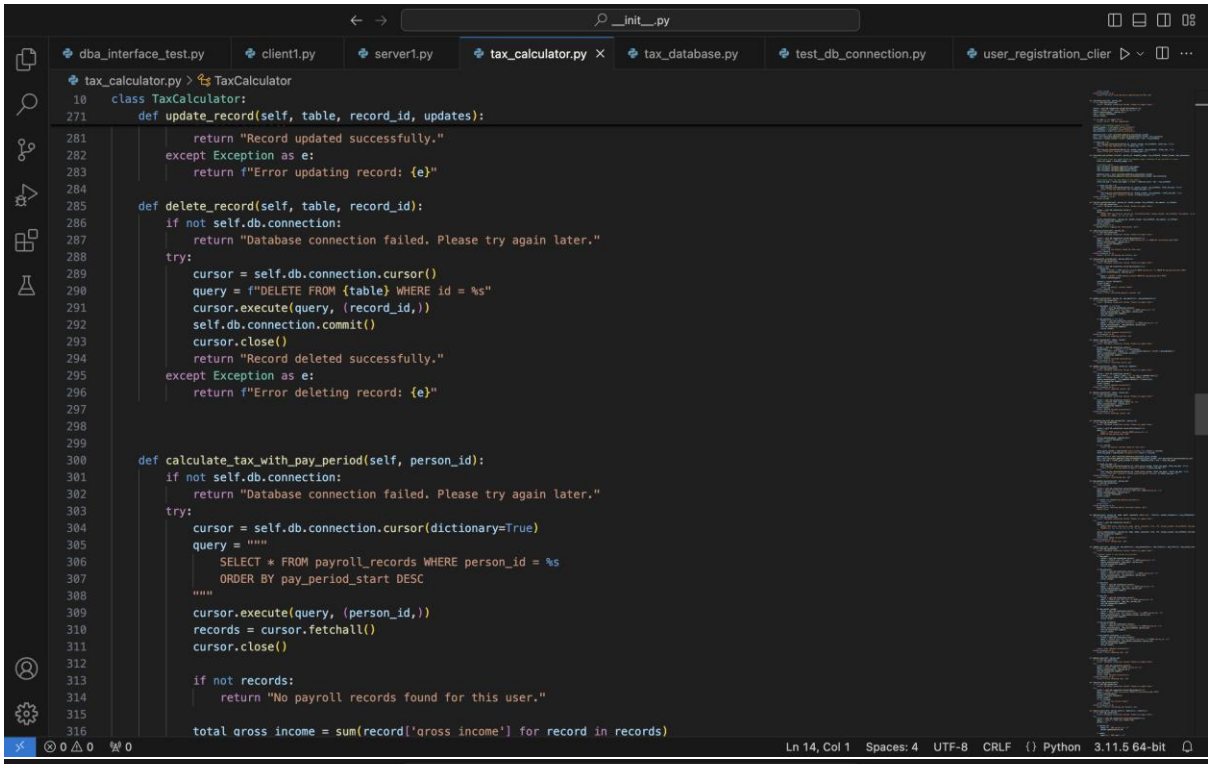
```
class TaxCalculator:
    def calculate_tax_without_tfn(self, person_id, biweekly_wages, tax_withheld, annual_income, has_insurance):

            if total_tax_due > 0:
                self.log_tax_calculation(person_id, annual_income, tax_withheld, total_tax_due, False)
                return f"You owe additional tax: ${total_tax_due:.2f}"
            else:
                self.log_tax_calculation(person_id, annual_income, tax_withheld, -total_tax_due, True)
                return f"You will receive a refund: ${-total_tax_due:.2f}"
        except ValueError as e:
            return str(e)

    def log_tax_calculation(self, person_id, annual_income, tax_withheld, tax_amount, is_refund):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor()
            query = """
                INSERT INTO tax_history (person_id, calculation_date, annual_income, tax_withheld, tax_amount, is_refund)
                VALUES (%s, NOW(), %s, %s, %s, %s)
            """
            cursor.execute(query, (person_id, annual_income, tax_withheld, tax_amount, is_refund))
            self.db.connection.commit()
            cursor.close()
        except Exception as e:
            print(f"Error logging tax calculation: {e}")

    def view_tax_history(self, person_id):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor(dictionary=True)
            query = "SELECT * FROM tax_history WHERE person_id = %s ORDER BY calculation_date DESC"
            cursor.execute(query, (person_id,))
            history = cursor.fetchall()
            cursor.close()
            if not history:
                return "No tax history found for this user."
```



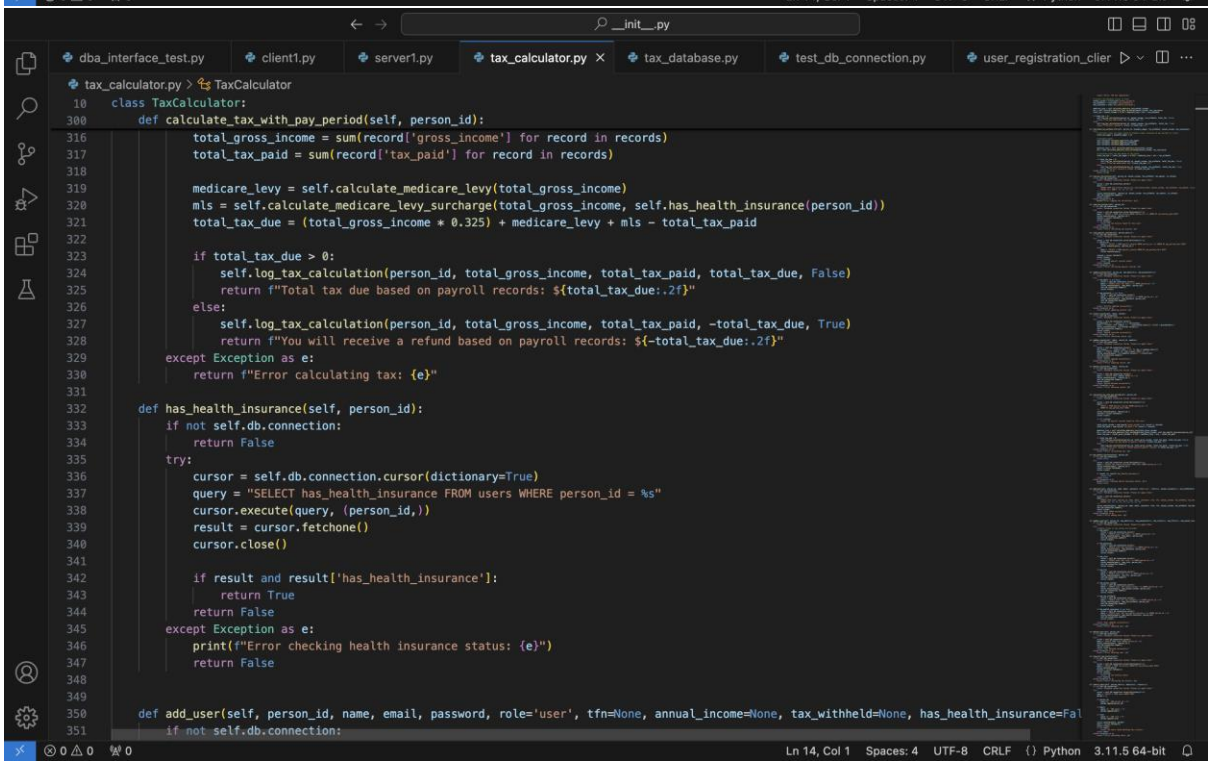
```
class TaxCalculator:
    def view_tax_history(self, person_id):
                    return "No tax history found for this user."
                return history
            except Exception as e:
                return f"Error retrieving tax history: {e}"

    def view_payroll_records(self, person_id=None):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor(dictionary=True)
            if person_id:
                query = "SELECT * FROM payroll_records WHERE person_id = %s ORDER BY pay_period_start DESC"
                cursor.execute(query, (person_id,))
            else:
                query = "SELECT * FROM payroll_records ORDER BY pay_period_start DESC"
                cursor.execute(query)

            records = cursor.fetchall()
            cursor.close()
            if not records:
                return "No payroll records found."
            return records
        except Exception as e:
            return f"Error retrieving payroll records: {e}"


    def update_profile(self, person_id, new_email=None, new_password=None):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            if new_email is not None:
                cursor = self.db.connection.cursor()
                query = "UPDATE users SET email = %s WHERE person_id = %s"
                cursor.execute(query, (new_email, person_id))
                self.db.connection.commit()
                cursor.close()
```



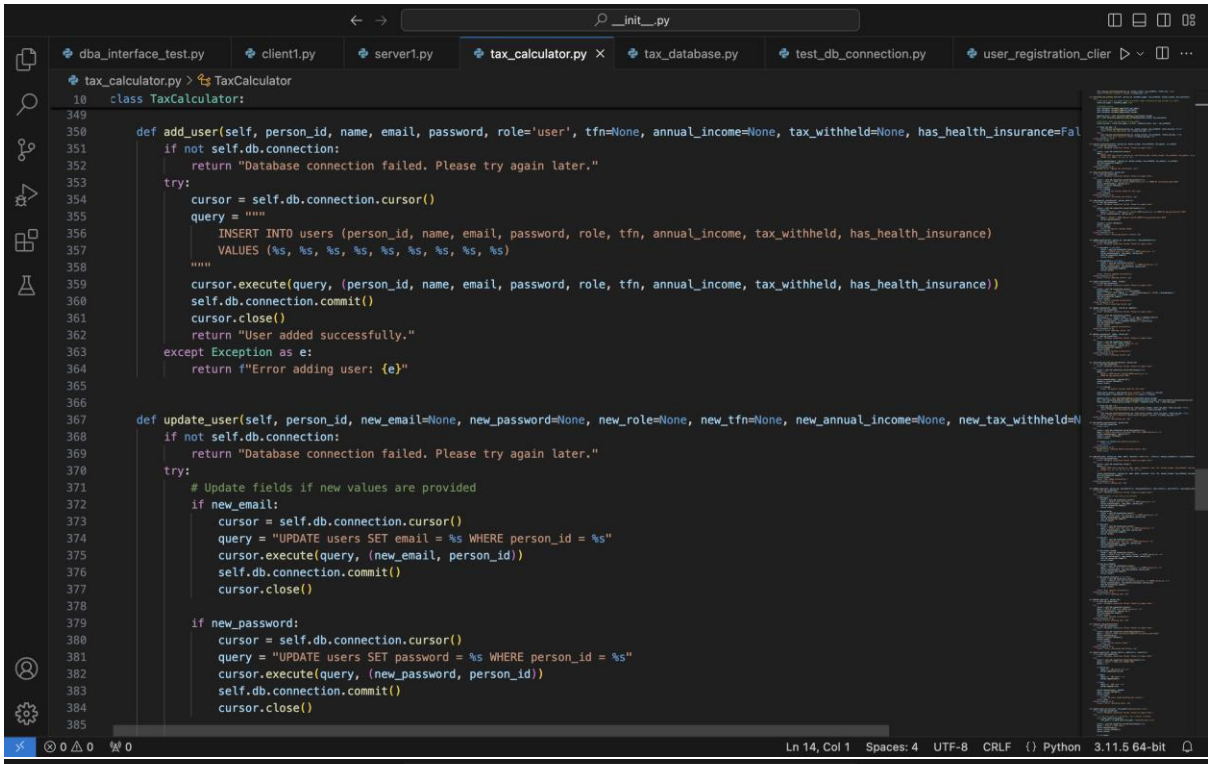
```
class TaxCalculator:
    def update_profile(self, person_id, new_email=None, new_password=None):

            if new_password is not None:
                cursor = self.db.connection.cursor()
                query = "UPDATE users SET password = %s WHERE person_id = %s"
                cursor.execute(query, (new_password, person_id))
                self.db.connection.commit()
                cursor.close()

            return "Profile updated successfully."
        except Exception as e:
            return f"Error updating profile: {e}"

    def insert_record(self, table, fields):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor()
            placeholders = ', '.join(['%s'] * len(fields))
            query = f"INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})"
            cursor.execute(query, list(fields.values()))
            self.db.connection.commit()
            cursor.close()
            return "Record inserted successfully."
        except Exception as e:
            return f"Error inserting record: {e}"

    def update_record(self, table, record_id, updates):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor()
            set_clause = ', '.join([f"{key} = %s" for key in updates.keys()])
            query = f"UPDATE {table} SET {set_clause} WHERE id = %s"
            cursor.execute(query, list(updates.values()) + [record_id])
            self.db.connection.commit()
            cursor.close()
```

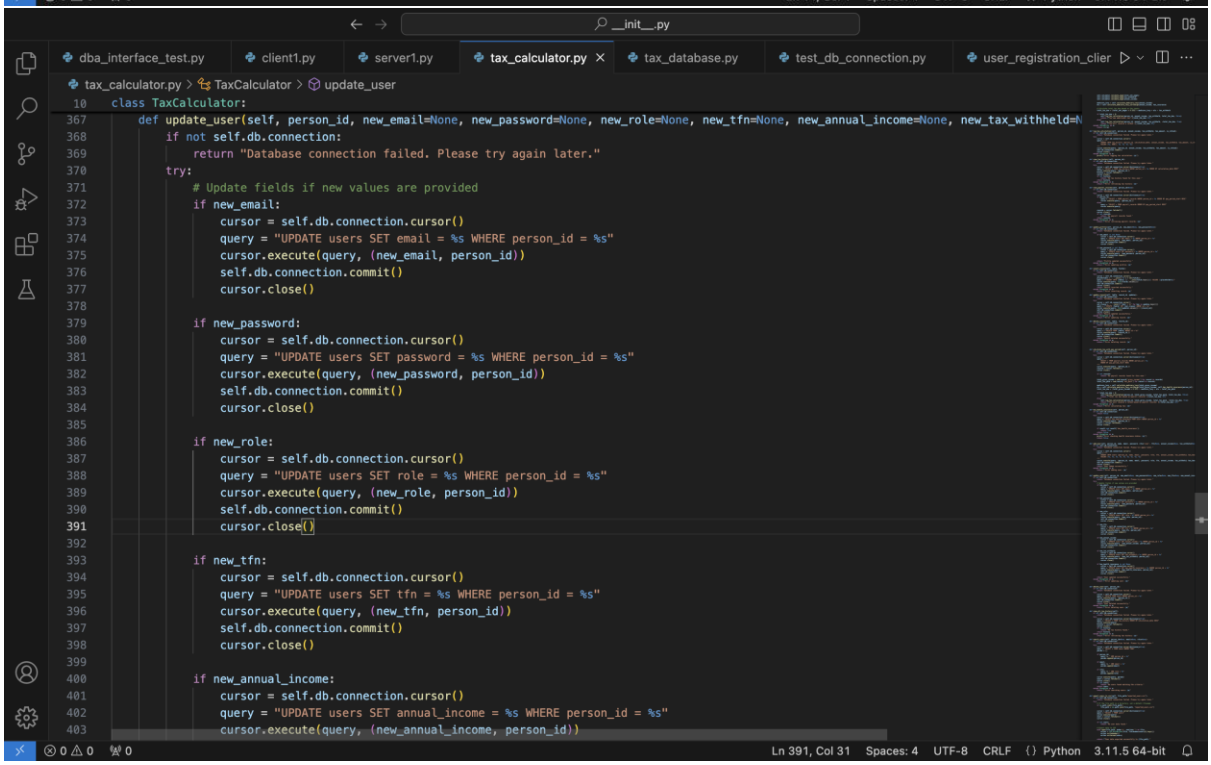


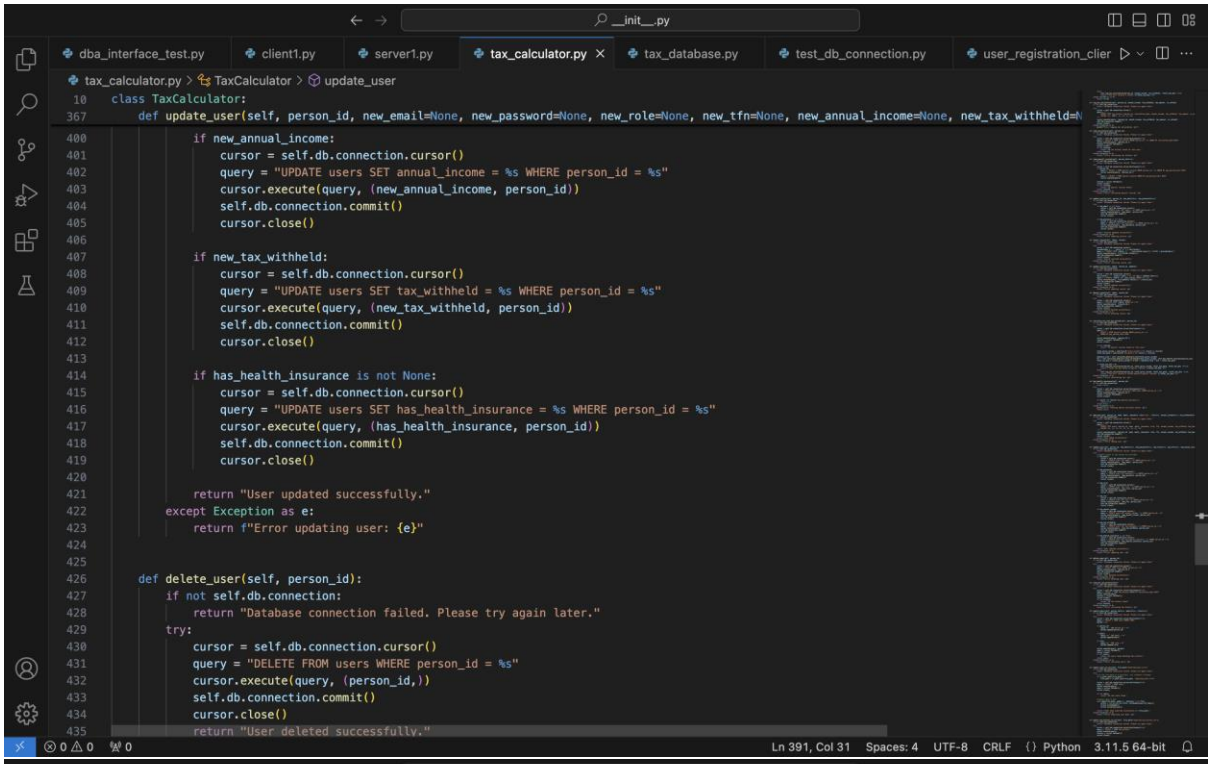
```
    def update_record(self, table, record_id, updates):
            return "Record updated successfully."
        except Exception as e:
            return f"Error updating record: {e}"

    def delete_record(self, table, record_id):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor()
            query = f"DELETE FROM {table} WHERE id = %s"
            cursor.execute(query, (record_id,))
            self.db.connection.commit()
            cursor.close()
            return "Record deleted successfully."
        except Exception as e:
            return f"Error deleting record: {e}"



    def calculate_tax_with_pay_period(self, person_id):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor(dictionary=True)
            query = """
                SELECT * FROM payroll_records WHERE person_id = %s
                ORDER BY pay_period_start DESC
            """
            cursor.execute(query, (person_id,))
            records = cursor.fetchall()
            cursor.close()

            if not records:
                return "No payroll records found for this user."

            total_gross_income = sum(record['gross_income'] for record in records)
```



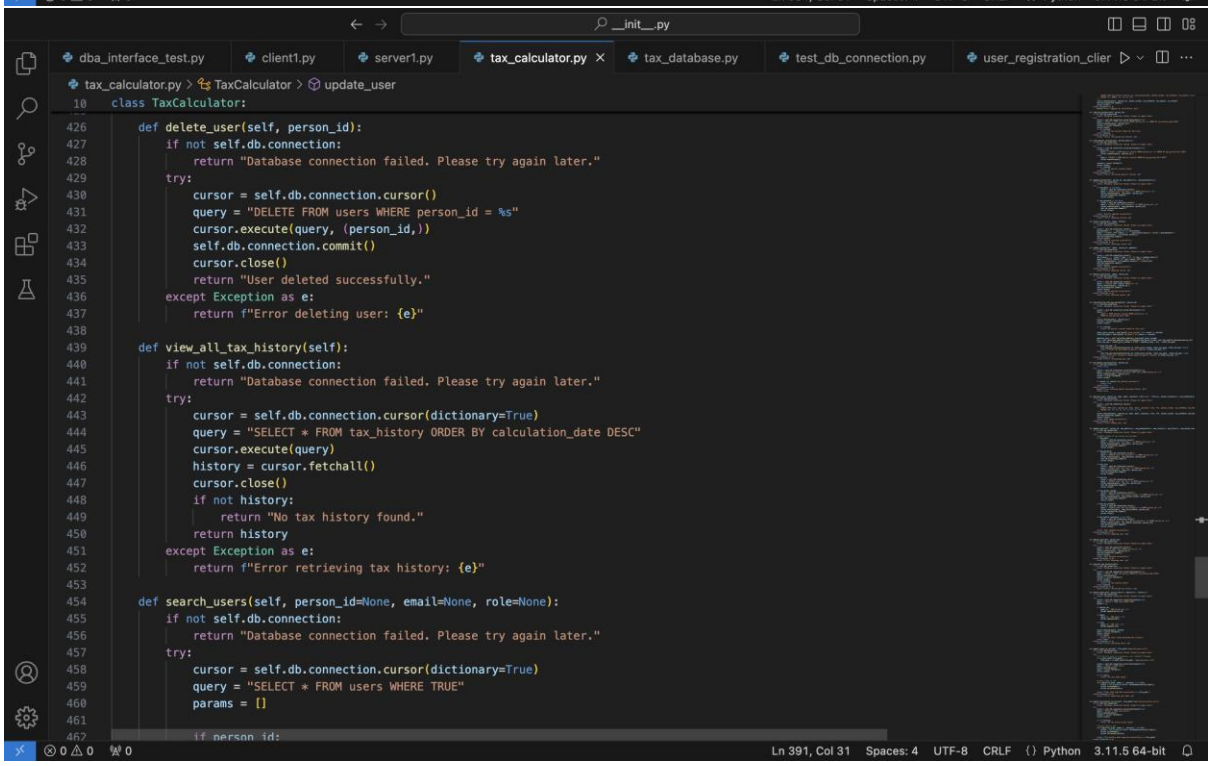
```
    def calculate_tax_with_pay_period(self, person_id):
            total_gross_income = sum(record['gross_income'] for record in records)
            total_tax_paid = sum(record['tax_paid'] for record in records)

            medicare_levy = self.calculate_medicare_levy(total_gross_income)
            mls = self.calculate_medicare_levy_surcharge(total_gross_income, self.has_health_insurance(person_id))
            total_tax_due = (total_gross_income * 0.325) + medicare_levy + mls - total_tax_paid

            if total_tax_due > 0:
                self.log_tax_calculation(person_id, total_gross_income, total_tax_paid, total_tax_due, False)
                return f"Total tax due based on payroll records: ${total_tax_due:.2f}"
            else:
                self.log_tax_calculation(person_id, total_gross_income, total_tax_paid, -total_tax_due, True)
                return f"You will receive a refund based on payroll records: ${-total_tax_due:.2f}"
        except Exception as e:
            return f"Error calculating tax: {e}"

    def has_health_insurance(self, person_id):
        if not self.db.connection:
            return False
        try:
            cursor = self.db.connection.cursor(dictionary=True)
            query = "SELECT has_health_insurance FROM users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            result = cursor.fetchone()
            cursor.close()

            if result and result['has_health_insurance']:
                return True
            return False
        except Exception as e:
            print(f"Error checking health insurance status: {e}")
            return False



    def add_user(self, person_id, name, email, password, role='user', tfn=None, annual_income=None, tax_withheld=None, has_health_insurance=Fal
        if not self.db.connection:
```


__init__.py
dba_interface_test.py
client1.py
server1.py
tax_calculator.py
tax_database.py
test_db_connection.py
user_registration_clier
tax_calculator.py > TaxCalculator
class TaxCalculator:
def add_user(self, person_id, name, email, password, role='user', tfn=None, annual_income=None, tax_withheld=None, has_health_insurance=Fal
if not self.db.connection:
return "Database connection failed. Please try again later."
try:
cursor = self.db.connection.cursor()
query = """
INSERT INTO users (person_id, name, email, password, role, tfn, annual_income, tax_withheld, has_health_insurance)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
"""
cursor.execute(query, (person_id, name, email, password, role, tfn, annual_income, tax_withheld, has_health_insurance))
self.db.connection.commit()
cursor.close()
return "User added successfully."
except Exception as e:
return f"Error adding user: {e}"
def update_user(self, person_id, new_email=None, new_password=None, new_role=None, new_tfn=None, new_annual_income=None, new_tax_withheld=N
if not self.db.connection:
return "Database connection failed. Please try again later."
try:
# Update fields if new values are provided
if new_email:
cursor = self.db.connection.cursor()
query = "UPDATE users SET email = %s WHERE person_id = %s"
cursor.execute(query, (new_email, person_id))
self.db.connection.commit()
cursor.close()
if new_password:
cursor = self.db.connection.cursor()
query = "UPDATE users SET password = %s WHERE person_id = %s"
cursor.execute(query, (new_password, person_id))
self.db.connection.commit()
cursor.close()
Ln 14, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.11.5 64-bit

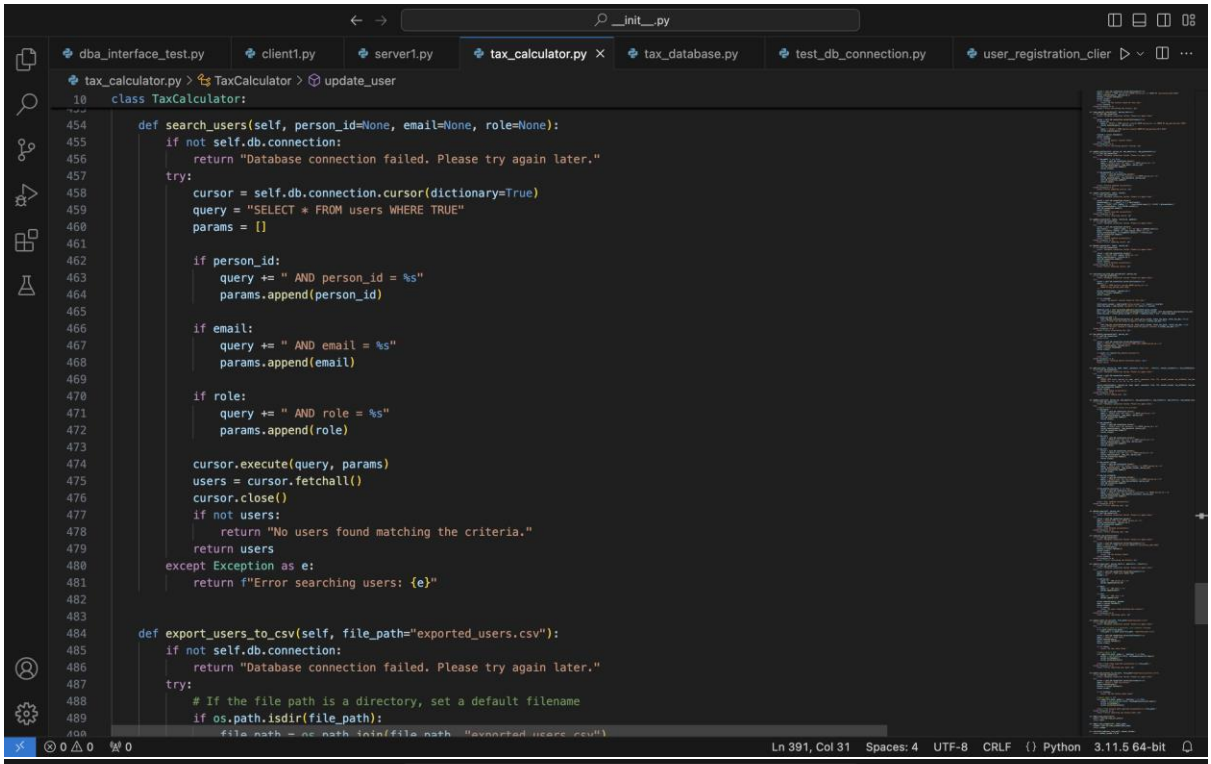

__init__.py
dba_interface_test.py
client1.py
server1.py
tax_calculator.py
tax_database.py
test_db_connection.py
user_registration_clier
tax_calculator.py > TaxCalculator > update_user
class TaxCalculator:
def update_user(self, person_id, new_email=None, new_password=None, new_role=None, new_tfn=None, new_annual_income=None, new_tax_withheld=N
if not self.db.connection:
return "Database connection failed. Please try again later."
try:
# Update fields if new values are provided
if new_email:
cursor = self.db.connection.cursor()
query = "UPDATE users SET email = %s WHERE person_id = %s"
cursor.execute(query, (new_email, person_id))
self.db.connection.commit()
cursor.close()
if new_password:
cursor = self.db.connection.cursor()
query = "UPDATE users SET password = %s WHERE person_id = %s"
cursor.execute(query, (new_password, person_id))
self.db.connection.commit()
cursor.close()
if new_role:
cursor = self.db.connection.cursor()
query = "UPDATE users SET role = %s WHERE person_id = %s"
cursor.execute(query, (new_role, person_id))
self.db.connection.commit()
cursor.close()
if new_tfn:
cursor = self.db.connection.cursor()
query = "UPDATE users SET tfn = %s WHERE person_id = %s"
cursor.execute(query, (new_tfn, person_id))
self.db.connection.commit()
cursor.close()
if new_annual_income:
cursor = self.db.connection.cursor()
query = "UPDATE users SET annual_income = %s WHERE person_id = %s"
cursor.execute(query, (new_annual_income, person_id))
Ln 391, Col 31
Spaces: 4
UTF-8
CRLF
Python
3.11.5 64-bit



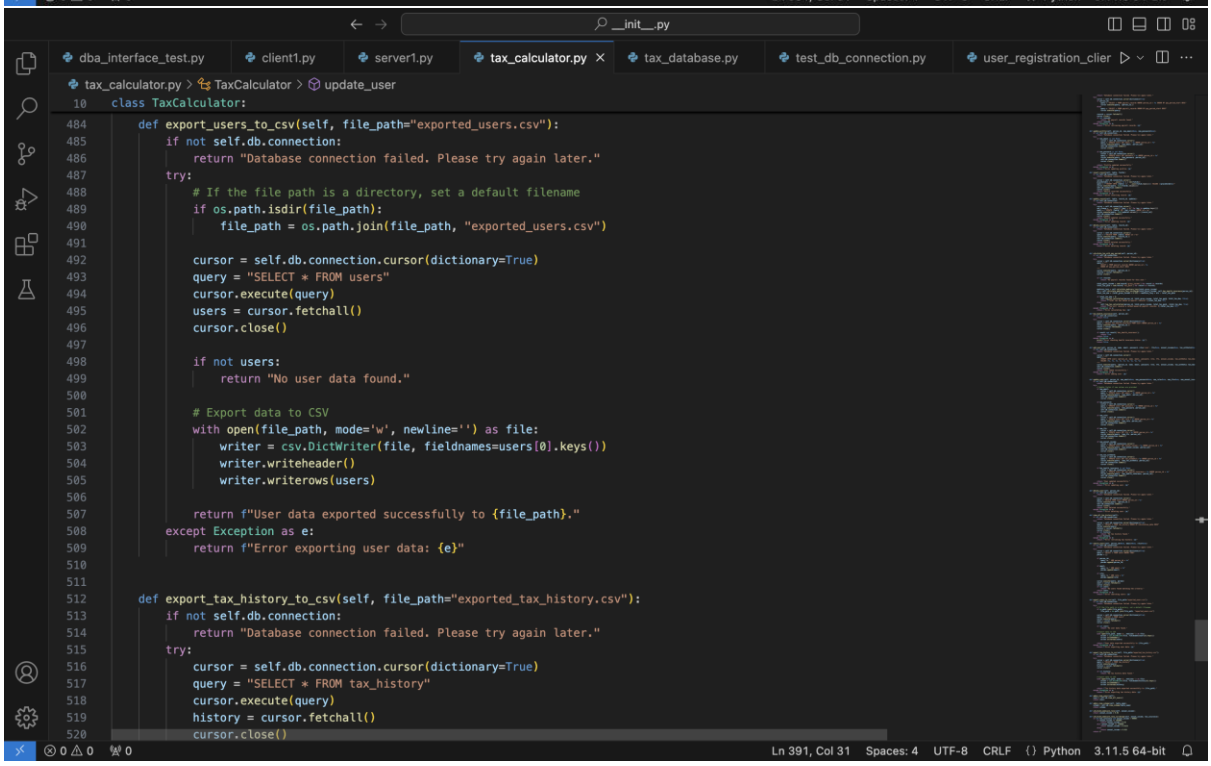
```
            if new_annual_income:
                cursor = self.db.connection.cursor()
                query = "UPDATE users SET annual_income = %s WHERE person_id = %s"
                cursor.execute(query, (new_annual_income, person_id))
                self.db.connection.commit()
                cursor.close()

            if new_tax_withheld:
                cursor = self.db.connection.cursor()
                query = "UPDATE users SET tax_withheld = %s WHERE person_id = %s"
                cursor.execute(query, (new_tax_withheld, person_id))
                self.db.connection.commit()
                cursor.close()

            if has_health_insurance is not None:
                cursor = self.db.connection.cursor()
                query = "UPDATE users SET has_health_insurance = %s WHERE person_id = %s"
                cursor.execute(query, (has_health_insurance, person_id))
                self.db.connection.commit()
                cursor.close()

            return "User updated successfully."
        except Exception as e:
            return f"Error updating user: {e}"


    def delete_user(self, person_id):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor()
            query = "DELETE FROM users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            self.db.connection.commit()
            cursor.close()
            return "User deleted successfully."
```



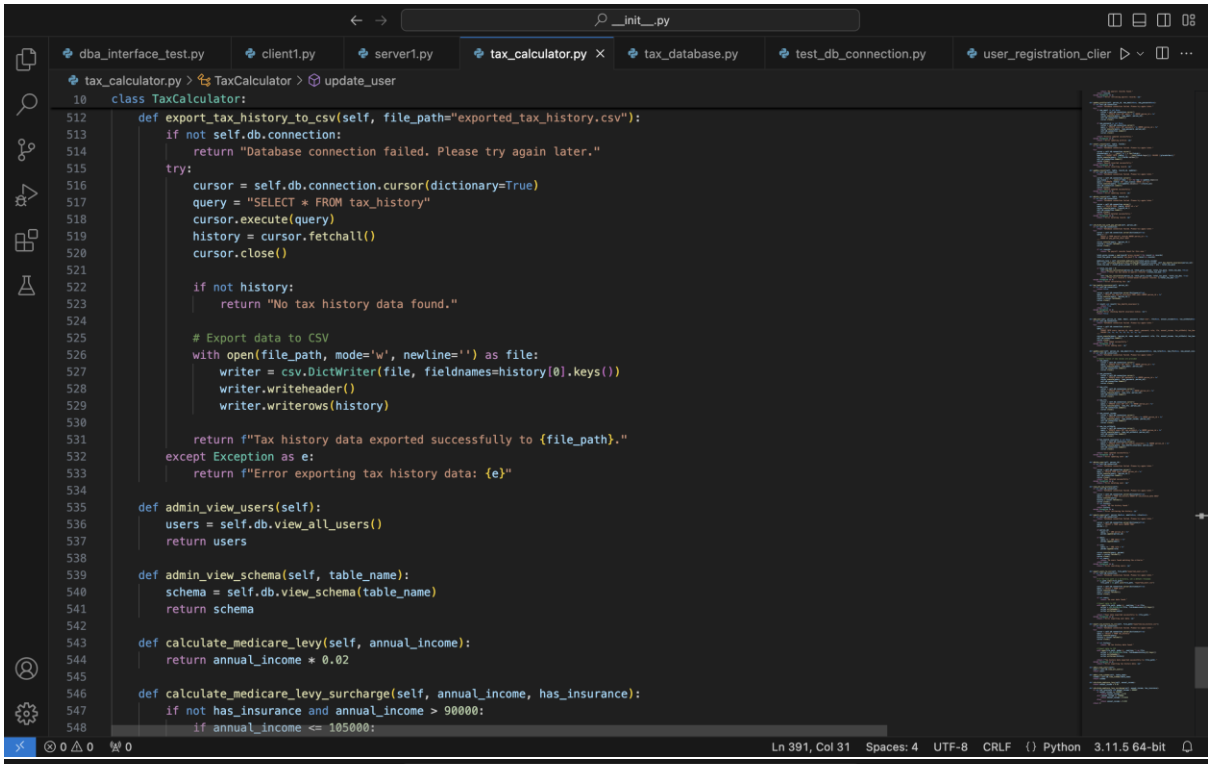
```
    def delete_user(self, person_id):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor()
            query = "DELETE FROM users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            self.db.connection.commit()
            cursor.close()
            return "User deleted successfully."
        except Exception as e:
            return f"Error deleting user: {e}"

    def view_all_tax_history(self):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor(dictionary=True)
            query = "SELECT * FROM tax_history ORDER BY calculation_date DESC"
            cursor.execute(query)
            history = cursor.fetchall()
            cursor.close()
            if not history:
                return "No tax history found."
            return history
        except Exception as e:
            return f"Error retrieving tax history: {e}"

    def search_users(self, person_id=None, email=None, role=None):
        if not self.db.connection:
            return "Database connection failed. Please try again later."
        try:
            cursor = self.db.connection.cursor(dictionary=True)
            query = "SELECT * FROM users WHERE TRUE"
            params = []
```

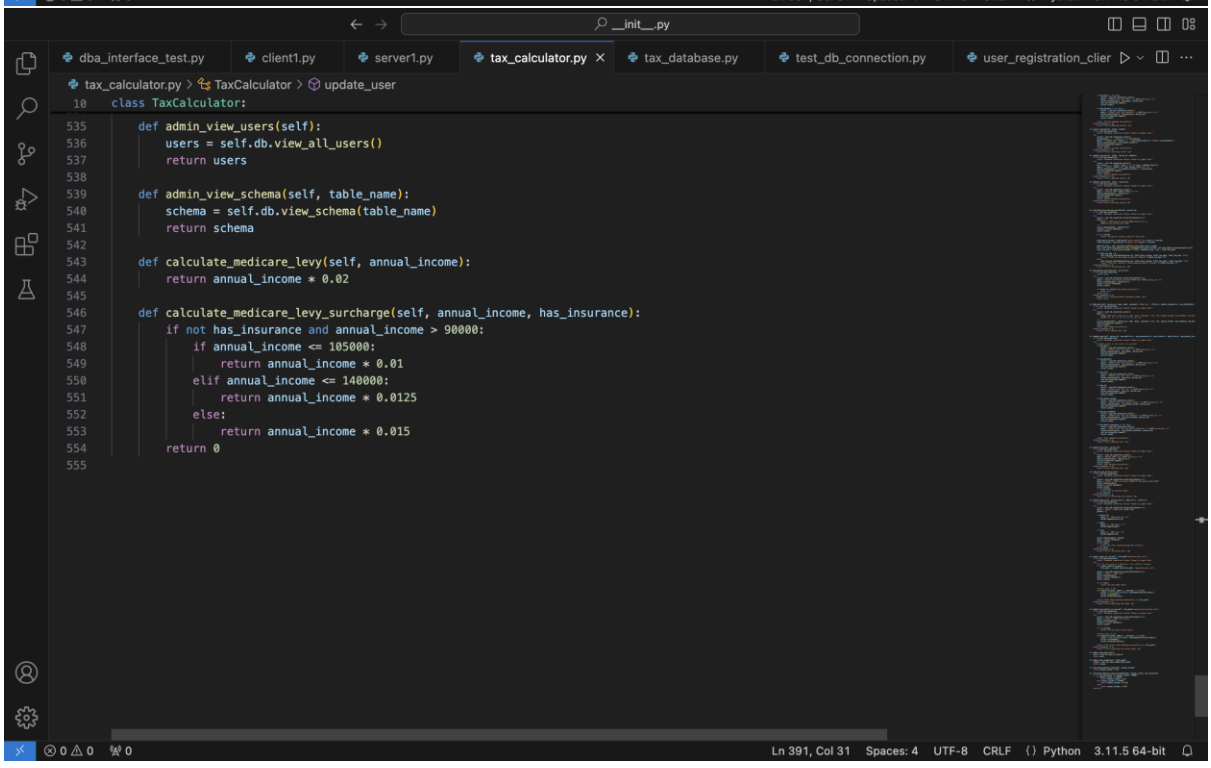


```
454     def search_users(self, person_id=None, email=None, role=None):
455         if not self.db.connection:
456             return "Database connection failed. Please try again later."
457         try:
458             cursor = self.db.connection.cursor(dictionary=True)
459             query = "SELECT * FROM users WHERE TRUE"
460             params = []
461
462             if person_id:
463                 query += " AND person_id = %s"
464                 params.append(person_id)
465
466             if email:
467                 query += " AND email = %s"
468                 params.append(email)
469
470             if role:
471                 query += " AND role = %s"
472                 params.append(role)
473
474             cursor.execute(query, params)
475             users = cursor.fetchall()
476             cursor.close()
477             if not users:
478                 return "No users found matching the criteria."
479             return users
480         except Exception as e:
481             return f"Error searching users: {e}"
482
483
484     def export_users_to_csv(self, file_path="exported_users.csv"):
485         if not self.db.connection:
486             return "Database connection failed. Please try again later."
487         try:
488             # If the file path is a directory, set a default filename
489             if os.path.isdir(file_path):
```

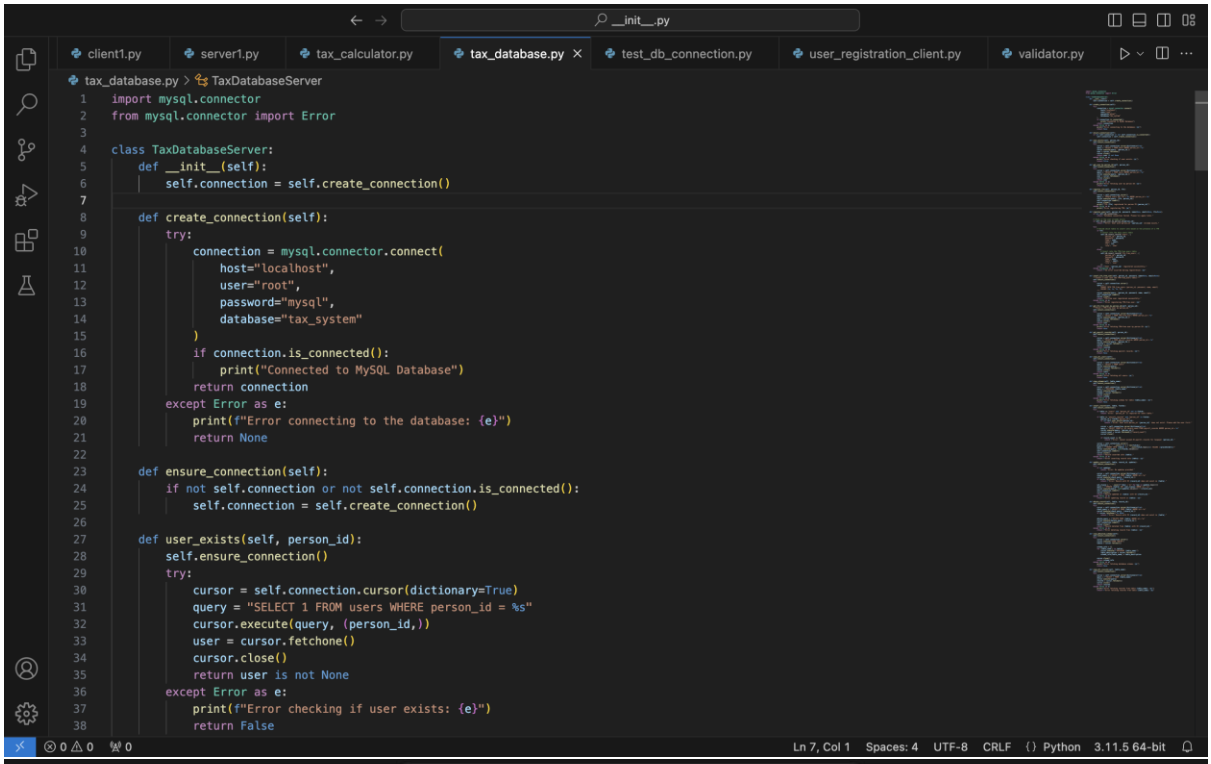


```
484     def export_users_to_csv(self, file_path="exported_users.csv"):
485         if not self.db.connection:
486             return "Database connection failed. Please try again later."
487         try:
488             # If the file path is a directory, set a default filename
489             if os.path.isdir(file_path):
490                 file_path = os.path.join(file_path, "exported_users.csv")
491
492             cursor = self.db.connection.cursor(dictionary=True)
493             query = "SELECT * FROM users"
494             cursor.execute(query)
495             users = cursor.fetchall()
496             cursor.close()
497
498             if not users:
499                 return "No user data found."
500
501             # Export data to CSV
502             with open(file_path, mode='w', newline='') as file:
503                 writer = csv.DictWriter(file, fieldnames=users[0].keys())
504                 writer.writeheader()
505                 writer.writerows(users)
506
507             return f"User data exported successfully to {file_path}."
508         except Exception as e:
509             return f"Error exporting user data: {e}"
510
511
512     def export_tax_history_to_csv(self, file_path="exported_tax_history.csv"):
513         if not self.db.connection:
514             return "Database connection failed. Please try again later."
515         try:
516             cursor = self.db.connection.cursor(dictionary=True)
517             query = "SELECT * FROM tax_history"
518             cursor.execute(query)
519             history = cursor.fetchall()
520             cursor.close()
```

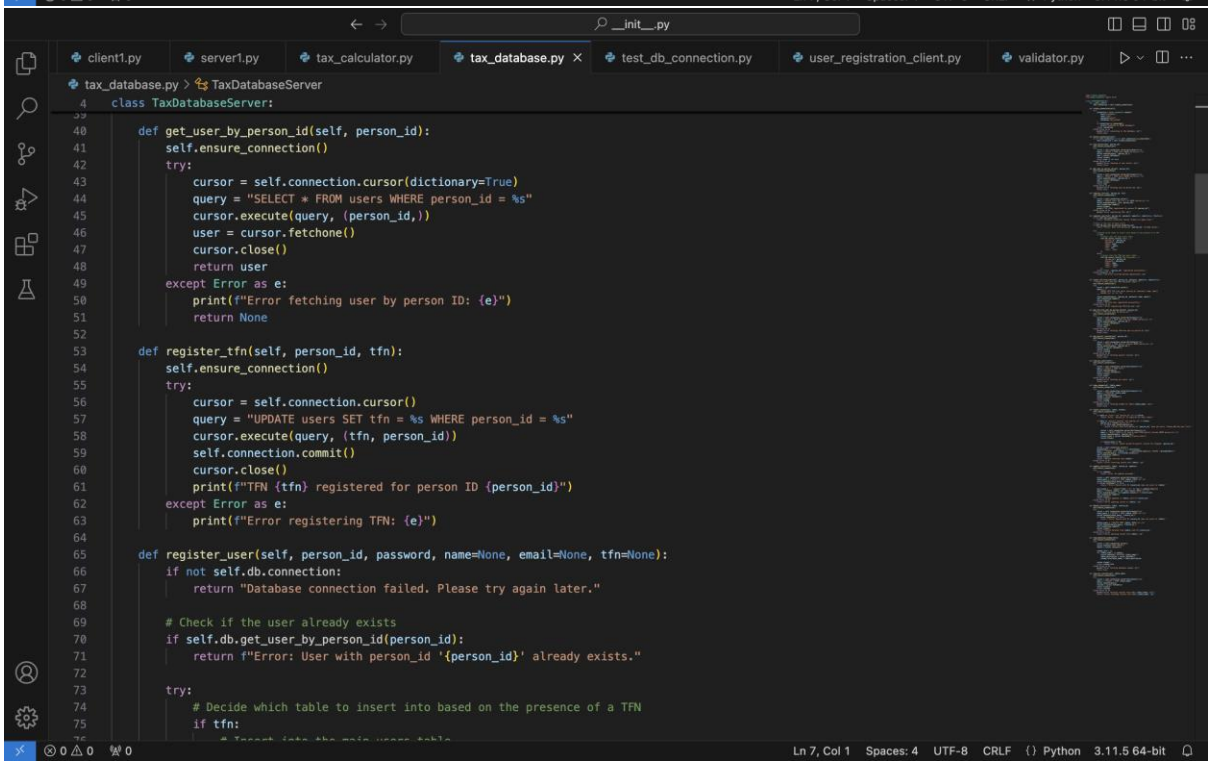




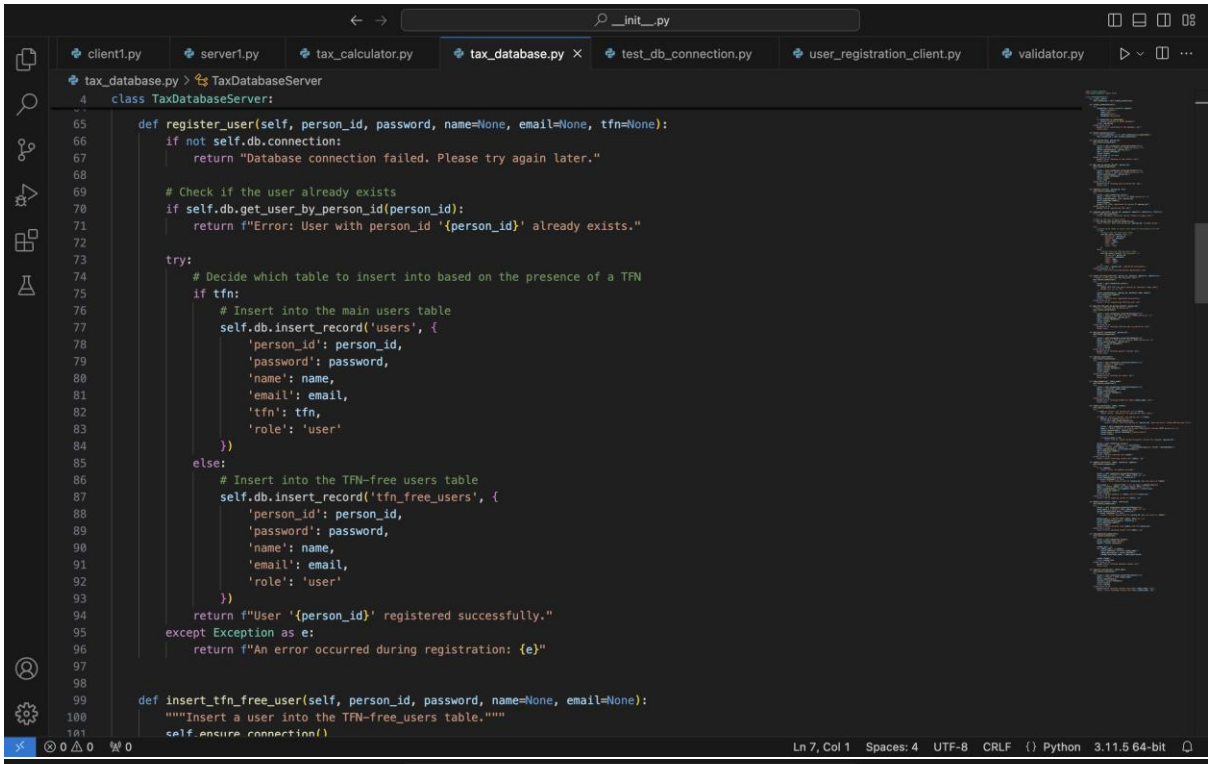
tax_database.py



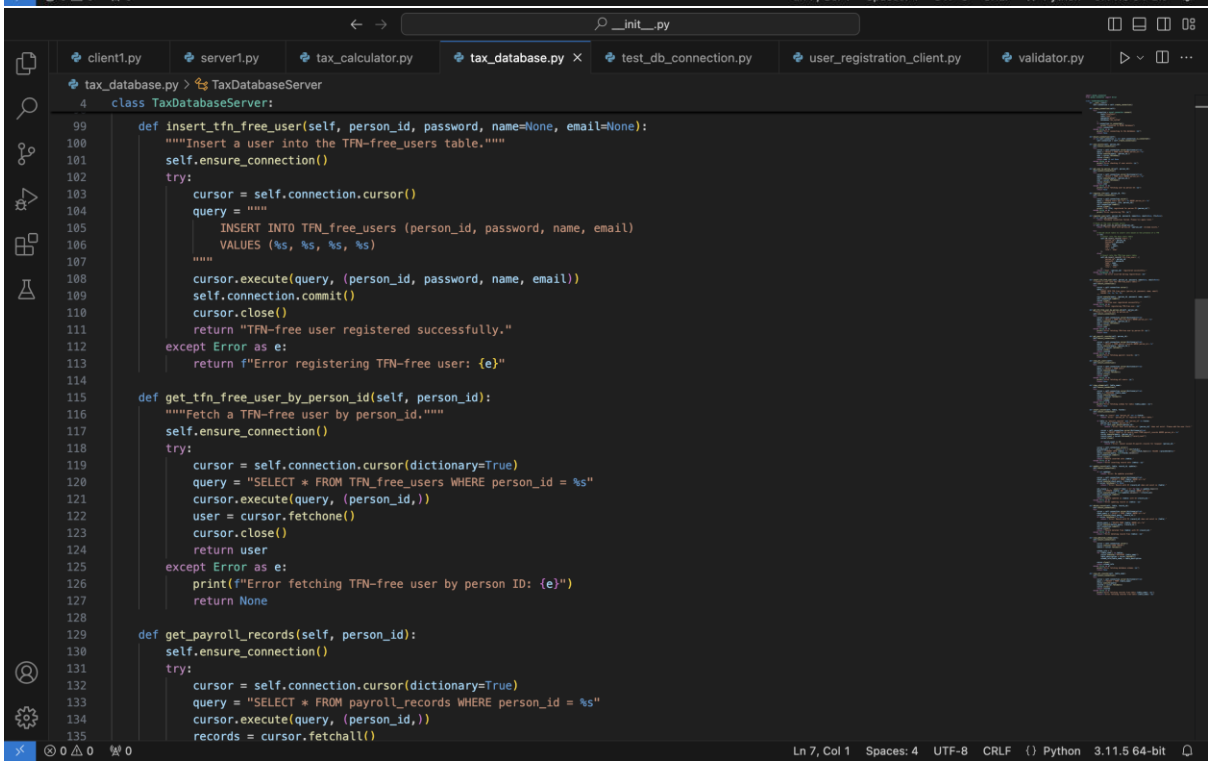
```python
import mysql.connector
from mysql.connector import Error

class TaxDatabaseServer:
    def __init__(self):
        self.connection = self.create_connection()

    def create_connection(self):
        try:
            connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="mysql",
                database="tax_system"
            )
            if connection.is_connected():
                print("Connected to MySQL Database")
            return connection
        except Error as e:
            print(f"Error connecting to the database: {e}")
            return None

    def ensure_connection(self):
        if not self.connection or not self.connection.is_connected():
            self.connection = self.create_connection()

    def user_exists(self, person_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT 1 FROM users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            user = cursor.fetchone()
            cursor.close()
            return user is not None
        except Error as e:
            print(f"Error checking if user exists: {e}")
            return False
```



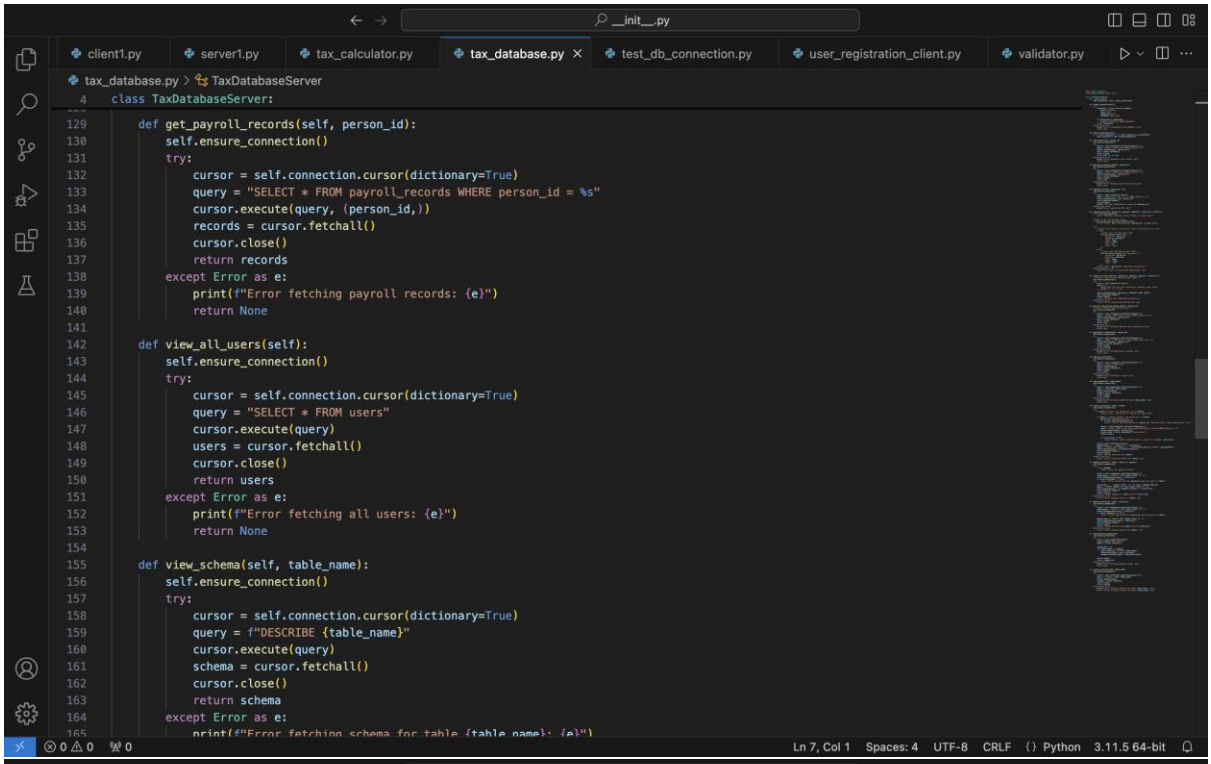
```python
class TaxDatabaseServer:

    def get_user_by_person_id(self, person_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            user = cursor.fetchone()
            cursor.close()
            return user
        except Error as e:
            print(f"Error fetching user by person ID: {e}")
            return None

    def register_tfn(self, person_id, tfn):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor()
            query = "UPDATE users SET tfn = %s WHERE person_id = %s"
            cursor.execute(query, (tfn, person_id))
            self.connection.commit()
            cursor.close()
            print(f"TFN {tfn} registered for person ID {person_id}")
        except Error as e:
            print(f"Error registering TFN: {e}")

    def register_user(self, person_id, password, name=None, email=None, tfn=None):
        if not self.db.connection:
            return "Database connection failed. Please try again later."

        # Check if the user already exists
        if self.db.get_user_by_person_id(person_id):
            return f"Error: User with person_id '{person_id}' already exists."

        try:
            # Decide which table to insert into based on the presence of a TFN
            if tfn:
```



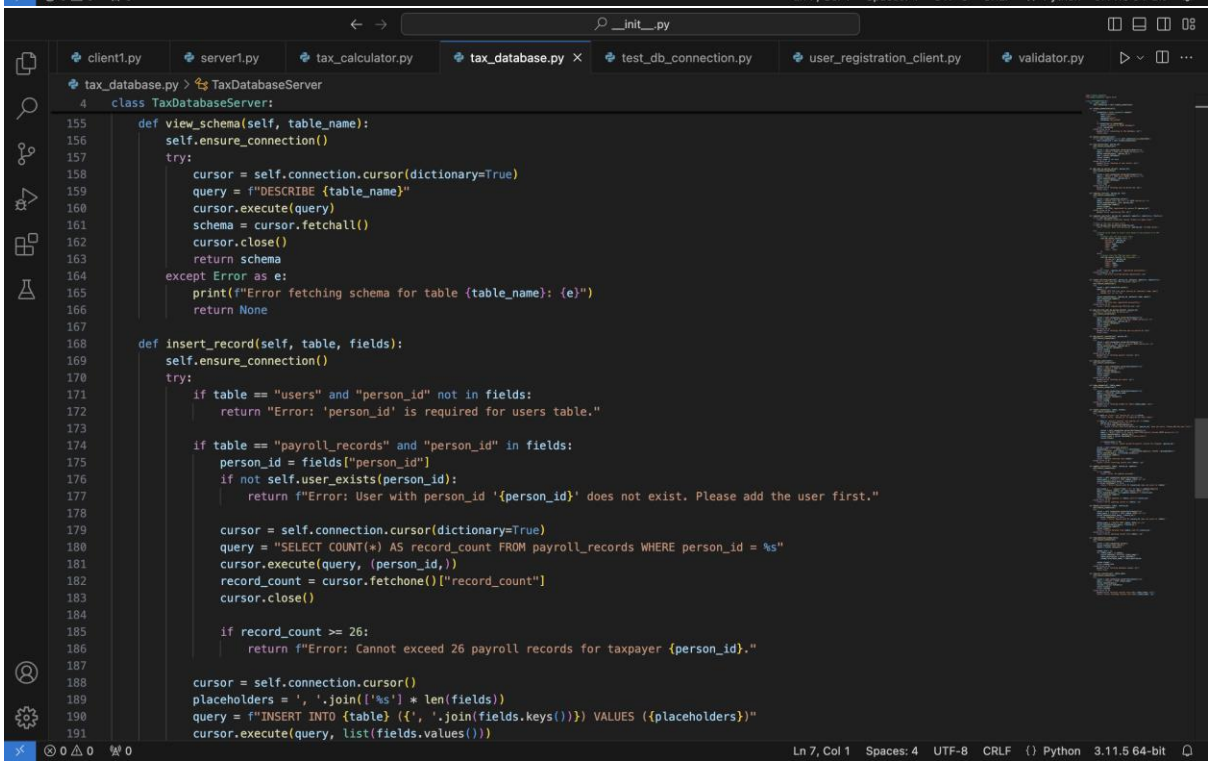
```python
class TaxDatabaseServer:
    def register_user(self, person_id, password, name=None, email=None, tfn=None):
        if not self.db.connection:
            return "Database connection failed. Please try again later."

        # Check if the user already exists
        if self.db.get_user_by_person_id(person_id):
            return f"Error: User with person_id '{person_id}' already exists."

        try:
            # Decide which table to insert into based on the presence of a TFN
            if tfn:
                # Insert into the main users table
                self.db.insert_record('users', {
                    'person_id': person_id,
                    'password': password,
                    'name': name,
                    'email': email,
                    'tfn': tfn,
                    'role': 'user'
                })
            else:
                # Insert into the TFN-free users table
                self.db.insert_record('tfn_free_users', {
                    'person_id': person_id,
                    'password': password,
                    'name': name,
                    'email': email,
                    'role': 'user'
                })
            return f"User '{person_id}' registered successfully."
        except Exception as e:
            return f"An error occurred during registration: {e}"


    def insert_tfn_free_user(self, person_id, password, name=None, email=None):
        """Insert a user into the TFN-free_users table."""
        self.ensure_connection()
```



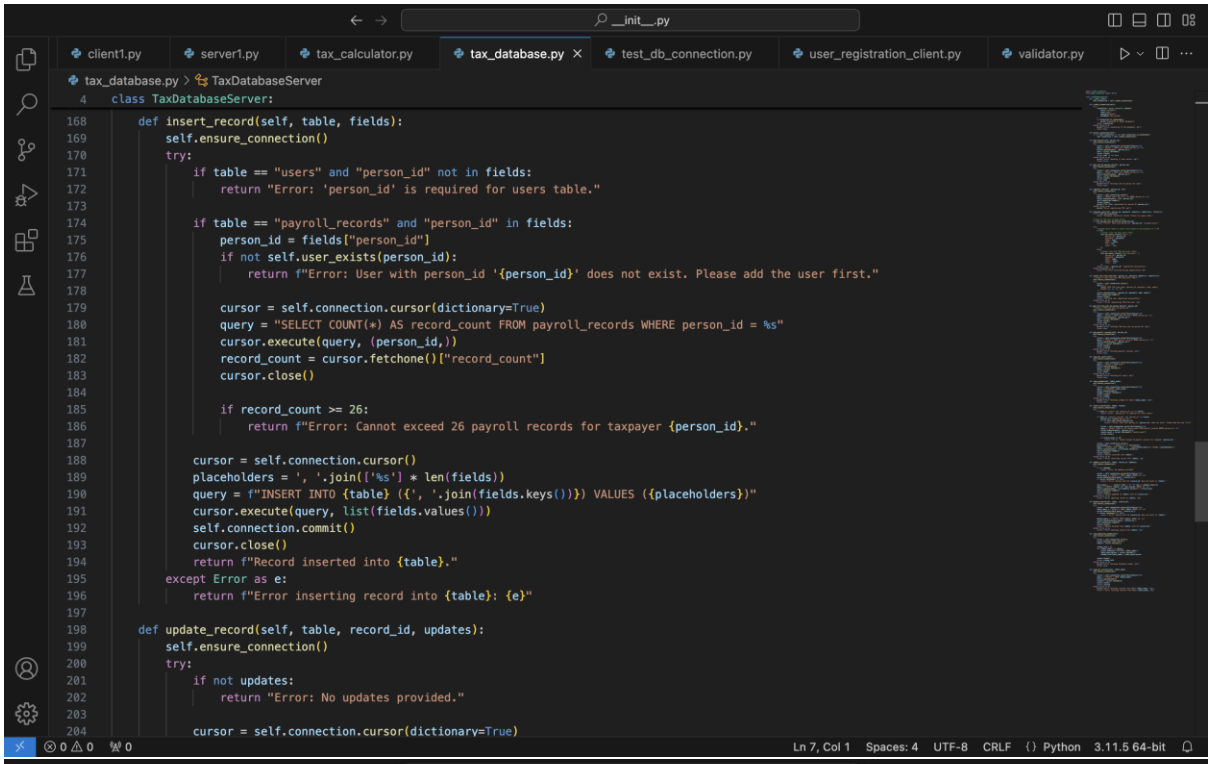
```python
class TaxDatabaseServer:
    def insert_tfn_free_user(self, person_id, password, name=None, email=None):
        """Insert a user into the TFN-free_users table."""
        self.ensure_connection()
        try:
            cursor = self.connection.cursor()
            query = """
                INSERT INTO TFN_free_users (person_id, password, name, email)
                VALUES (%s, %s, %s, %s)
            """
            cursor.execute(query, (person_id, password, name, email))
            self.connection.commit()
            cursor.close()
            return "TFN-free user registered successfully."
        except Error as e:
            return f"Error registering TFN-free user: {e}"

    def get_tfn_free_user_by_person_id(self, person_id):
        """Fetch a TFN-free user by person_id."""
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM TFN_free_users WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            user = cursor.fetchone()
            cursor.close()
            return user
        except Error as e:
            print(f"Error fetching TFN-free user by person ID: {e}")
            return None

    def get_payroll_records(self, person_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM payroll_records WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            records = cursor.fetchall()
```



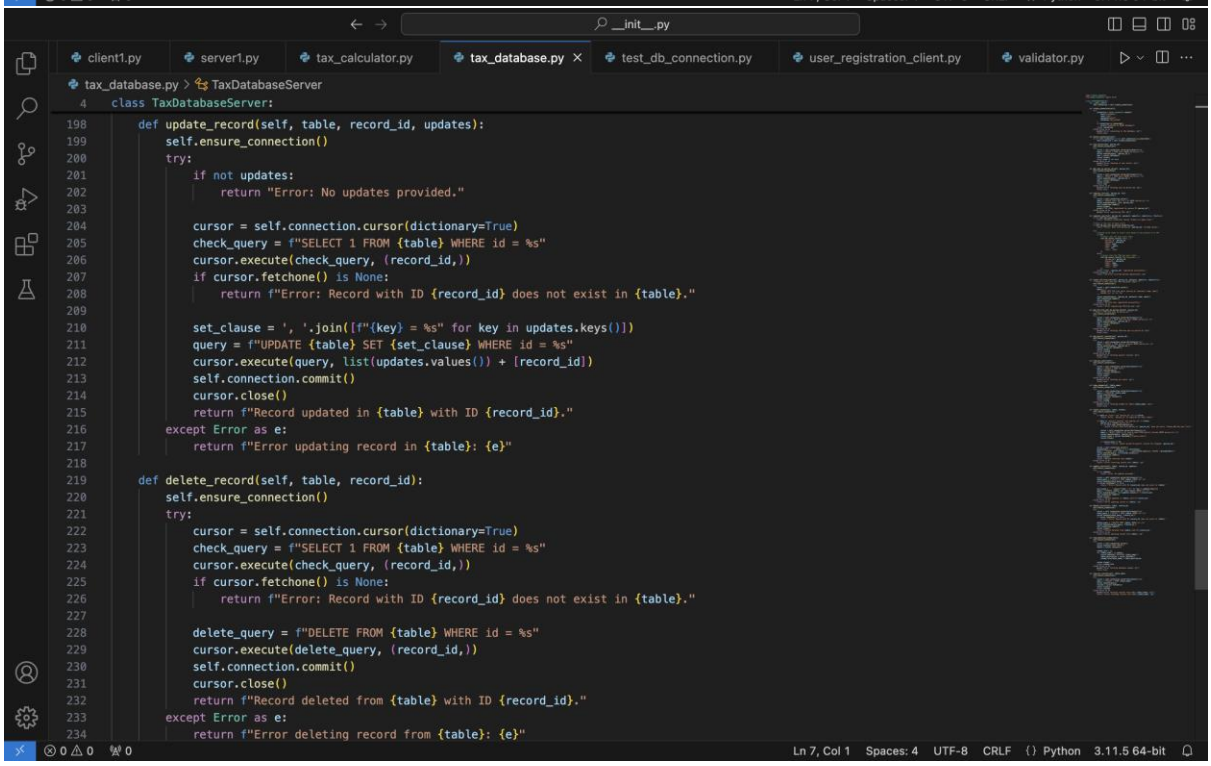
```python
    def get_payroll_records(self, person_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM payroll_records WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            records = cursor.fetchall()
            cursor.close()
            return records
        except Error as e:
            print(f"Error fetching payroll records: {e}")
            return None

    def view_all_users(self):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM users"
            cursor.execute(query)
            users = cursor.fetchall()
            cursor.close()
            return users
        except Error as e:
            print(f"Error fetching all users: {e}")
            return None

    def view_schema(self, table_name):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = f"DESCRIBE {table_name}"
            cursor.execute(query)
            schema = cursor.fetchall()
            cursor.close()
            return schema
        except Error as e:
```



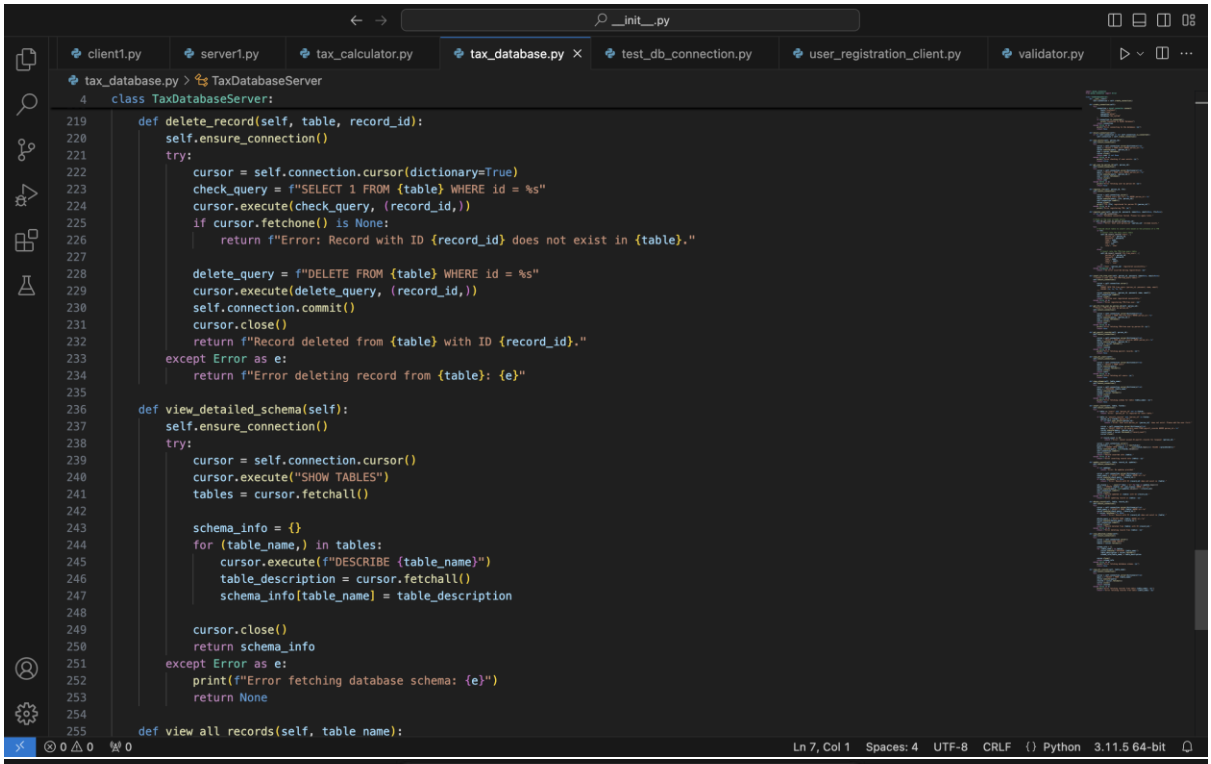
```python
    def view_schema(self, table_name):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = f"DESCRIBE {table_name}"
            cursor.execute(query)
            schema = cursor.fetchall()
            cursor.close()
            return schema
        except Error as e:
            print(f"Error fetching schema for table {table_name}: {e}")
            return None

    def insert_record(self, table, fields):
        self.ensure_connection()
        try:
            if table == "users" and "person_id" not in fields:
                return "Error: 'person_id' is required for users table."

            if table == "payroll_records" and "person_id" in fields:
                person_id = fields["person_id"]
                if not self.user_exists(person_id):
                    return f"Error: User with person_id '{person_id}' does not exist. Please add the user first."

                cursor = self.connection.cursor(dictionary=True)
                query = "SELECT COUNT(*) AS record_count FROM payroll_records WHERE person_id = %s"
                cursor.execute(query, (person_id,))
                record_count = cursor.fetchone()["record_count"]
                cursor.close()

                if record_count >= 26:
                    return f"Error: Cannot exceed 26 payroll records for taxpayer {person_id}."

            cursor = self.connection.cursor()
            placeholders = ', '.join(['%s'] * len(fields))
            query = f"INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})"
            cursor.execute(query, list(fields.values()))
```



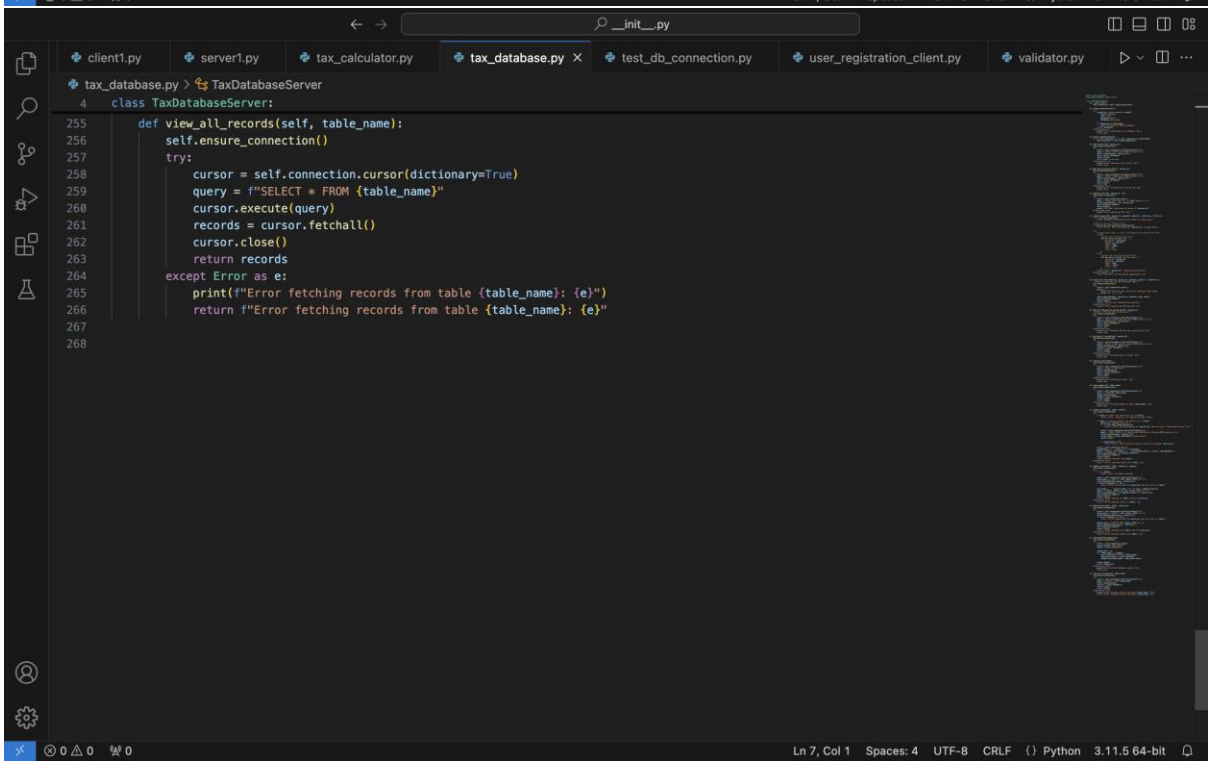
```python
class TaxDatabaseServer:
    def insert_record(self, table, fields):
        self.ensure_connection()
        try:
            if table == "users" and "person_id" not in fields:
                return "Error: 'person_id' is required for users table."

            if table == "payroll_records" and "person_id" in fields:
                person_id = fields["person_id"]
                if not self.user_exists(person_id):
                    return f"Error: User with person_id '{person_id}' does not exist. Please add the user first."

                cursor = self.connection.cursor(dictionary=True)
                query = "SELECT COUNT(*) AS record_count FROM payroll_records WHERE person_id = %s"
                cursor.execute(query, (person_id,))
                record_count = cursor.fetchone()["record_count"]
                cursor.close()

                if record_count >= 26:
                    return f"Error: Cannot exceed 26 payroll records for taxpayer {person_id}."

            cursor = self.connection.cursor()
            placeholders = ', '.join(['%s'] * len(fields))
            query = f"INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})"
            cursor.execute(query, list(fields.values()))
            self.connection.commit()
            cursor.close()
            return f"Record inserted into {table}."
        except Error as e:
            return f"Error inserting record into {table}: {e}"

    def update_record(self, table, record_id, updates):
        self.ensure_connection()
        try:
            if not updates:
                return "Error: No updates provided."

            cursor = self.connection.cursor(dictionary=True)
```



```python
class TaxDatabaseServer:
    def update_record(self, table, record_id, updates):
        self.ensure_connection()
        try:
            if not updates:
                return "Error: No updates provided."

            cursor = self.connection.cursor(dictionary=True)
            check_query = f"SELECT 1 FROM {table} WHERE id = %s"
            cursor.execute(check_query, (record_id,))
            if cursor.fetchone() is None:
                return f"Error: Record with ID {record_id} does not exist in {table}."

            set_clause = ', '.join([f"{key} = %s" for key in updates.keys()])
            query = f"UPDATE {table} SET {set_clause} WHERE id = %s"
            cursor.execute(query, list(updates.values()) + [record_id])
            self.connection.commit()
            cursor.close()
            return f"Record updated in {table} with ID {record_id}."
        except Error as e:
            return f"Error updating record in {table}: {e}"

    def delete_record(self, table, record_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            check_query = f"SELECT 1 FROM {table} WHERE id = %s"
            cursor.execute(check_query, (record_id,))
            if cursor.fetchone() is None:
                return f"Error: Record with ID {record_id} does not exist in {table}."

            delete_query = f"DELETE FROM {table} WHERE id = %s"
            cursor.execute(delete_query, (record_id,))
            self.connection.commit()
            cursor.close()
            return f"Record deleted from {table} with ID {record_id}."
        except Error as e:
            return f"Error deleting record from {table}: {e}"
```

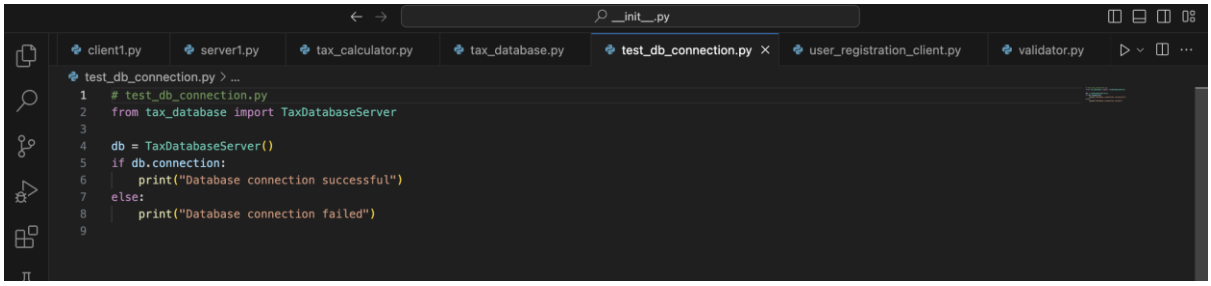


```python
    def delete_record(self, table, record_id):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            check_query = f"SELECT 1 FROM {table} WHERE id = %s"
            cursor.execute(check_query, (record_id,))
            if cursor.fetchone() is None:
                return f"Error: Record with ID {record_id} does not exist in {table}."

            delete_query = f"DELETE FROM {table} WHERE id = %s"
            cursor.execute(delete_query, (record_id,))
            self.connection.commit()
            cursor.close()
            return f"Record deleted from {table} with ID {record_id}."
        except Error as e:
            return f"Error deleting record from {table}: {e}"

    def view_detailed_schema(self):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor()
            cursor.execute("SHOW TABLES")
            tables = cursor.fetchall()

            schema_info = {}
            for (table_name,) in tables:
                cursor.execute(f"DESCRIBE {table_name}")
                table_description = cursor.fetchall()
                schema_info[table_name] = table_description

            cursor.close()
            return schema_info
        except Error as e:
            print(f"Error fetching database schema: {e}")
            return None

    def view all records(self, table name):
```

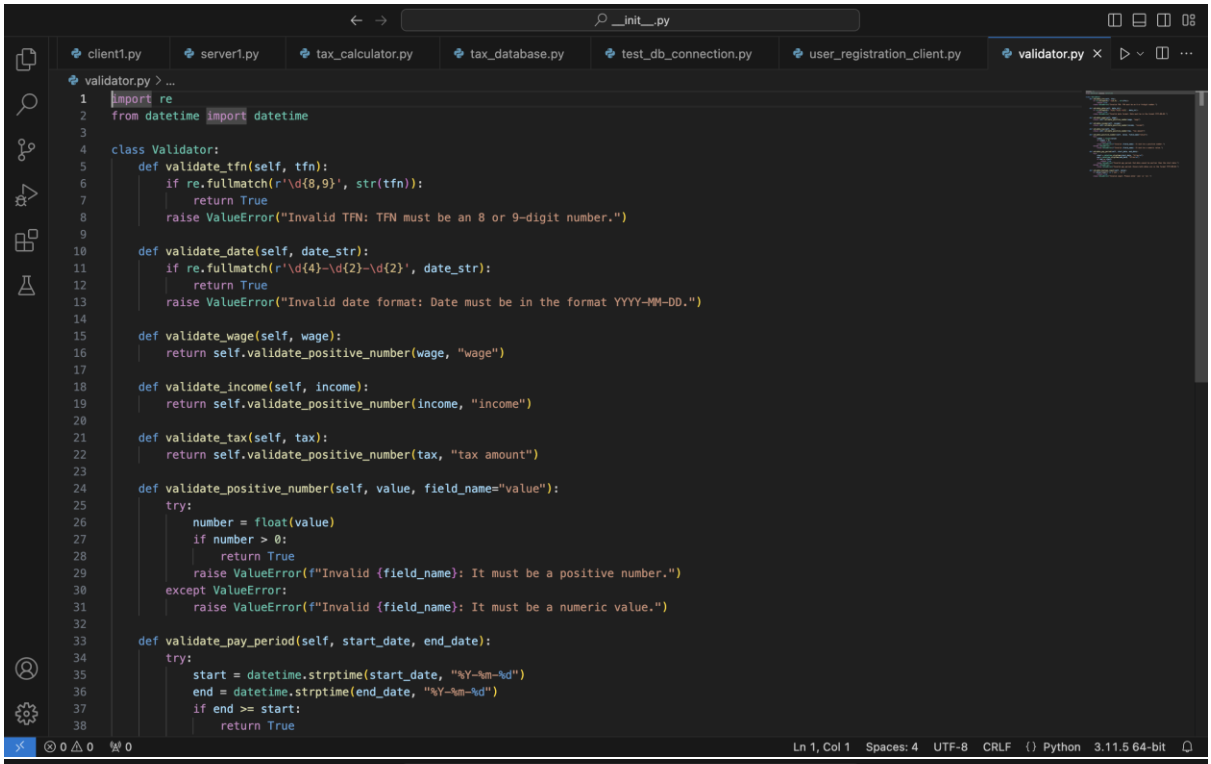


```python
    def view_all_records(self, table_name):
        self.ensure_connection()
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = f"SELECT * FROM {table_name}"
            cursor.execute(query)
            records = cursor.fetchall()
            cursor.close()
            return records
        except Error as e:
            print(f"Error fetching records from table {table_name}: {e}")
            return f"Error fetching records from table {table_name}: {e}"
```

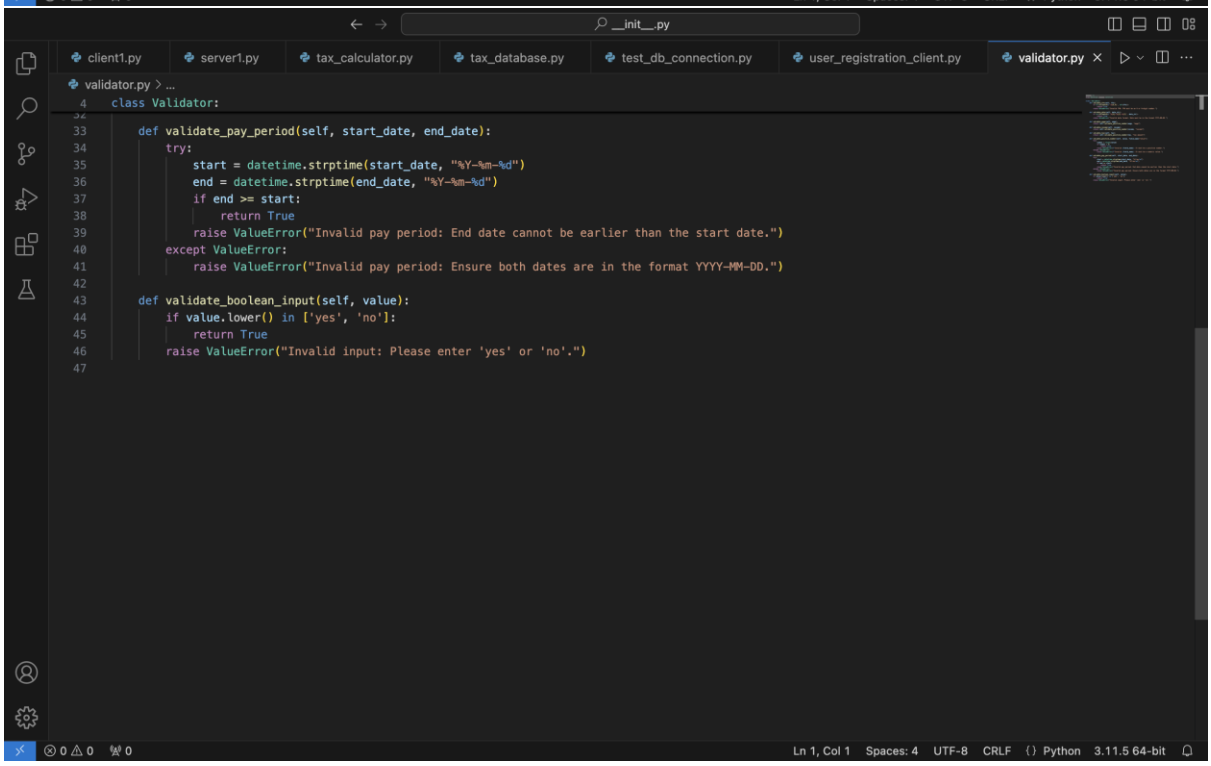
test_db_connection.py



validator.py



```python
import re
from datetime import datetime

class Validator:
    def validate_tfn(self, tfn):
        if re.fullmatch(r'\d{8,9}', str(tfn)):
            return True
        raise ValueError("Invalid TFN: TFN must be an 8 or 9-digit number.")

    def validate_date(self, date_str):
        if re.fullmatch(r'\d{4}-\d{2}-\d{2}', date_str):
            return True
        raise ValueError("Invalid date format: Date must be in the format YYYY-MM-DD.")

    def validate_wage(self, wage):
        return self.validate_positive_number(wage, "wage")

    def validate_income(self, income):
        return self.validate_positive_number(income, "income")

    def validate_tax(self, tax):
        return self.validate_positive_number(tax, "tax amount")

    def validate_positive_number(self, value, field_name="value"):
        try:
            number = float(value)
            if number > 0:
                return True
            raise ValueError(f"Invalid {field_name}: It must be a positive number.")
        except ValueError:
            raise ValueError(f"Invalid {field_name}: It must be a numeric value.")

    def validate_pay_period(self, start_date, end_date):
        try:
            start = datetime.strptime(start_date, "%Y-%m-%d")
            end = datetime.strptime(end_date, "%Y-%m-%d")
            if end >= start:
                return True
```



```python
class Validator:

    def validate_pay_period(self, start_date, end_date):
        try:
            start = datetime.strptime(start_date, "%Y-%m-%d")
            end = datetime.strptime(end_date, "%Y-%m-%d")
            if end >= start:
                return True
            raise ValueError("Invalid pay period: End date cannot be earlier than the start date.")
        except ValueError:
            raise ValueError("Invalid pay period: Ensure both dates are in the format YYYY-MM-DD.")

    def validate_boolean_input(self, value):
        if value.lower() in ['yes', 'no']:
            return True
        raise ValueError("Invalid input: Please enter 'yes' or 'no'.")
```

User Manual

Installation

1. Install Python

    - Download and install Python from the official website- python.org/downloads/
    - Make sure to add Python to the system PATH during installation.

    Verify Python Installation by entering the below command any 3. version

    - python –version

2. Install MySQL Server

    - Download and install MySQL Server from - MySQL d.nameserverownload
    - Follow the setup instructions and set up a root password.

    Verify MySQL Installation by entering the following command

    - mysql –version

3. Set Up the MySQL Database

   Open MySQL command-line tool -

   - mysql -u root -p

   Create the database -

   - CREATE DATABASE tax_system;

   Switch to the database -

```
mysql> USE tax_system;
Database changed
```

4. Install Pyro5 package using pip
   - pip install Pyro5

```
Collecting Pyro5
  Downloading Pyro5-5.15-py3-none-any.whl.metadata (4.9 kB)
Collecting serpent>=1.41 (from Pyro5)
  Downloading serpent-1.41-py3-none-any.whl.metadata (5.8 kB)
Downloading Pyro5-5.15-py3-none-any.whl (79 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 146.8 kB/s eta 0:00:00
Downloading serpent-1.41-py3-none-any.whl (9.6 kB)
Installing collected packages: serpent, Pyro5
```

5. Install MySQL Connector for Python

   - pip install mysql-connector-python

```
Collecting mysql-connector-python
  Downloading mysql_connector_python-9.1.0-cp311-cp311-win_amd64.whl.metadata (6.2 kB)
Downloading mysql_connector_python-9.1.0-cp311-cp311-win_amd64.whl (16.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.1/16.1 MB 116.2 kB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.1.0

[notice] A new release of pip is available: 24.0 -> 24.2
```

6. Create necessary tables for the system in the database

Database Creation for the Phase 2 Implementation

```
CREATE TABLE users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(10) NOT NULL UNIQUE,
        tfn VARCHAR(9),
        name VARCHAR(100),
        email VARCHAR(100),
        password VARCHAR(50),
         role ENUM('user', 'admin') NOT NULL DEFAULT 'user',
         annual_income DECIMAL(10,2),
         tax_withheld DECIMAL(10,2),
         has_health_insurance TINYINT(1),
        UNIQUE (person_id)
);

CREATE TABLE payroll_records (
        id INT AUTO_INCREMENT PRIMARY KEY,
         person_id VARCHAR(10),
         pay_period_start DATE,
        pay_period_end DATE,
        gross_income DECIMAL(10,2),
        tax_paid DECIMAL(10,2),
        FOREIGN KEY (person_id) REFERENCES users(person_id)
);
```

```
CREATE TABLE tfn_free_users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(50) NOT NULL UNIQUE,
        password VARCHAR(255) NOT NULL,
        name VARCHAR(255),
        email VARCHAR(255),
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE tax_history (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(10),
        calculation_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        annual_income DECIMAL(10,2),
        tax_withheld DECIMAL(10,2),
        tax_amount DECIMAL(10,2),
        is_refund TINYINT(1),
        FOREIGN KEY (person_id) REFERENCES users(person_id)
);
```

Application Run Status

Run Pyro5 name server

```
PS C:\Users\minag> cd C:\Users\minag\__init__.py
PS C:\Users\minag\__init__.py> python -m Pyro5.nameserver
Not starting broadcast server for IPv6.
NS running on localhost:9090 (::1)
URI = PYRO:Pyro.NameServer@localhost:9090
```

Run the Server code

```
PS C:\Users\minag> cd C:\Users\minag\__init__.py
PS C:\Users\minag\__init__.py> python server1.py
Server is running with database integration...
Connected to MySQL Database
```

Run the Client code

```
PS C:\Users\minag> cd C:\Users\minag\__init__.py
PS C:\Users\minag\__init__.py> python client1.py
Welcome to the Tax System
```

Main Menu Options

When the application starts, users are presented with the following options –

```
Welcome to the Tax System

Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 1
```

1. User Login

- Allows registered users to log into their accounts using their person_id and password.
- **Steps** -
    1. Enter your person_id.
    2. Enter your password.
    3. If the credentials are correct, you are logged in and directed to the user menu.
    4. If the credentials are incorrect, an error message is displayed.

2. Admin Login

- Allows admins to access the admin menu using their admin credentials.
- **Steps** -
    1. Enter your Admin ID.
    2. Enter your Admin password.
    3. If the credentials are correct and you have admin privileges, you are logged in and directed to the admin menu.

3. User Registration

- New users can register with or without a TFN.
- **Steps** -
    1. Enter your person_id.
    2. Enter a password.
    3. Enter your name (optional).
    4. Enter your email (optional).
    5. Enter your TFN if you have one or leave it blank if you do not.
    6. If the registration is successful, a confirmation message is displayed.

4. Exit

- Exits the application.

User Menu Options

Once logged in, users have the following options –

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Logout
Enter your choice: 7
```

1. Register a New TFN

- Users who have not registered their TFN can do so.
- **Steps** -
    1. Enter your new TFN.
    2. Enter your gross income.
    3. Enter your tax withheld.
    4. Enter your net income.
    5. Enter the pay period start date (YYYY-MM-DD).
    6. Enter the pay period end date (YYYY-MM-DD).
    7. Indicate if you have private health insurance (yes/no).
    8. If successful, a message confirms the TFN and tax record registration.

2. Calculate Tax with TFN

- Users who have registered their TFN can calculate their tax based on their annual income and other data.
- The system displays the tax amount owed or the refund due.

3. Calculate Tax without TFN

- Users without a TFN can calculate their tax based on biweekly wages and other details.
- **Steps** -
    1. Enter your biweekly net wages.
    2. Enter your tax withheld.
    3. Enter your estimated annual income.
    4. Indicate if you have private health insurance (yes/no).
    5. The system calculates and displays the tax owed or refund amount.

4. View Tax History

- Users can view the history of their past tax calculations.
- The system displays a list of tax records, including the calculation date, tax amount, and refund information.

5. View Payroll Records

- Displays payroll records specific to the user, including details like pay period start and end dates, gross income, and tax paid.

6. Update Profile

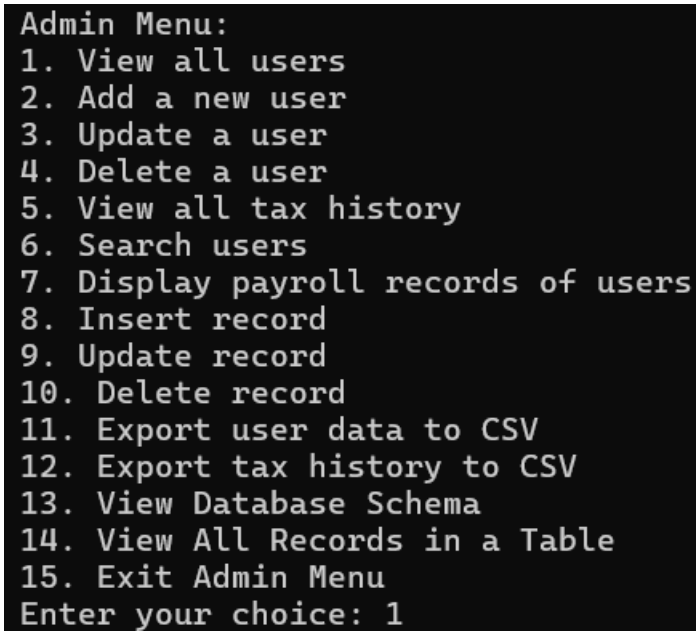
- Users can update their email and password.
- **Steps** -
    1. Indicate if you want to update your email.
    2. Enter your new email if applicable.
    3. Indicate if you want to update your password.
    4. Enter your new password if applicable.
    5. The system confirms if the profile update is successful.

7. Logout

- Logs the user out of the system and returns to the main menu.

Admin Menu Options

Admins have additional functionalities for managing user accounts and system data –



1. View All Users

- Displays a list of all users registered in the system.

2. Add a New User

- Allows the admin to manually add a new user.
- **Steps**:
    1. Enter user details such as person_id, name, email, password, role (user/admin), and TFN (optional).
    2. Enter optional details like annual income and tax withheld.
    3. The system confirms if the user has been added successfully.

3. Update a User

- Admins can update details of any user, including email, password, role, and TFN.

4. Delete a User

- Admins can delete user accounts from the system.
- **Steps -**
    1. Enter the person_id of the user to be deleted.
    2. Confirm the deletion.
    3. The system confirms if the user was successfully deleted.

5. View All Tax History

- Displays the entire tax history for all users in the system.

6. Search Users

- Allows admins to search for users based on person_id, email, or role.

7. Display Payroll Records of Users

- Admins can view payroll records of all users or a specific user.

8. Insert Record

- Admins can manually insert records into any table in the database.
- The system confirms successful insertion.

9. Update Record

- Admins can update existing records in any table.
- The system confirms successful update.

10. Delete Record

- Admins can delete records from any table.
- The system confirms successful deletion.

11. Export User Data to CSV

- Exports user data into a CSV file.
- Admins are prompted to specify a file path for saving.

12. Export Tax History to CSV

- Exports tax history into a CSV file.
- Admins are prompted to specify a file path for saving.

13. View Database Schema

- Displays the structure of all tables in the database, including column names and data types.

14. View All Records in a Table

- Allows admins to view all records in a specified table.

15. Exit Admin Menu

- Logs out the admin and returns to the main menu.

4. Phase 2: Test Cases

Test Case ID	Scenario	User Input	Expected Output	Actual Output	Status
TC-01	Valid user login	Person ID: user123 Password: pass123	Login successful, welcome message shown	Login successful	Pass
TC-02	Invalid user login	Person ID: user123 Password: wrong	Error message: "Invalid credentials"	Error shown	Pass
TC-03	User registration with TFN	Person ID: newUser TFN: 123456789	User registered successfully	User registered	Pass
TC-04	User registration without TFN	Person ID: noTFNUser	User registered successfully (TFN-free)	User registered	Pass
TC-05	Tax calculation for user with TFN	Person ID: user123	Tax amount or refund displayed	Tax calculated	Pass
TC-06	Tax calculation for user without TFN	Person ID: noTFNUser Wages: 3000	Tax amount or refund displayed	Tax calculated	Pass
TC-07	TFN registration for existing user	Person ID: user123 TFN: 987654321	TFN updated, confirmation message shown	TFN updated	Pass
TC-08	Payroll record entry with invalid date	Start Date: 2024-15-01	Error message: "Invalid date format"	Error shown	Pass
TC-09	Tax history retrieval for valid user	Person ID: user123	Tax history records displayed	Records displayed	Pass

Test Case ID	Scenario	User Input	Expected Output	Actual Output	Status
TC-10	Tax history retrieval for non-existing user	Person ID: noRecordUser	Error message: "No tax history found"	Error shown	Pass
TC-11	Invalid data entry (negative wage)	Wages: -500	Error message: "Invalid wage amount"	Error shown	Pass
TC-12	Database connection failure	None	Error message: "Database connection failed"	Error shown	Pass
TC-13	Admin view all users	Admin credentials	List of all users displayed	Users listed	Pass
TC-14	Admin inserts new record	Table: users Data: Valid data	Record inserted successfully	Record inserted	Pass
TC-15	Admin deletes a user	User ID: userToDelete	Confirmation message shown, user deleted	User deleted	Pass
TC-16	User attempts to access system without login	None	Error message: "Please log in first"	Error shown	Pass
TC-17	Tax calculation for user with invalid TFN	Person ID: invalidUser TFN: abc123	Error message: "Invalid TFN format"	Error shown	Pass
TC-18	Admin exports user data to CSV	File path: validPath	CSV file created successfully	CSV created	Pass
TC-19	User profile update	New Email: new@example.com	Profile updated successfully	Profile updated	Pass

Test Case ID	Scenario	User Input	Expected Output	Actual Output	Status
TC-20	Admin tries to update a non-existent user	Person ID: nonExistent	Error message: "User does not exist"	Error shown	Pass

10 Conclusion

The development of the Personal Income Tax Return Estimate (PITRE) system, implemented as a distributed application using Pyro5 for Remote Procedure Call (RPC), demonstrates the effective use of distributed systems architecture in tax estimation processes. The project successfully showcases the integration of client-server communication, database management, and modular design principles, enabling scalable and efficient interaction between the different components of the system

Through the implementation of a multi-tier architecture, the system achieves a separation of concerns, which enhances maintainability and scalability. The client application allows users to input tax-related information or retrieve data stored in the MySQL database, with the Pyro5 framework handling the RPC interactions seamlessly. The Tax Calculation Server processes user data and interacts with the Database Server to fetch and store necessary records, ensuring accurate and real-time tax calculations based on the Australian tax system.

By leveraging Pyro5's capabilities, the system achieves reliable and asynchronous communication, which is crucial for handling client requests in real-time. The integration of MySQL further enhances the system's functionality by ensuring secure and persistent data storage, allowing users to access their tax history and payroll records when needed.

Overall, the PITRE system successfully fulfills the project objectives by demonstrating the essential concepts of distributed systems, such as remote method invocation (RMI via Pyro5 RPC), data management across multiple tiers, and client-server architecture. The modular nature of the system allows for future enhancements, such as expanding the database capabilities or integrating more complex tax rules and deductions, thus highlighting the flexibility and robustness of distributed systems in real-world applications.

11 References

Pyro - Python Remote Objects - 5.14-dev — Pyro 5.14-dev documentation. (n.d.). Pyro5.Readthedocs.io. https://pyro5.readthedocs.io/en/latest/index.html

RPC implementation using Python | Vrishabhdhwaj. (n.d.). Coderspacket.com. https://coderspacket.com/rpc-implementation-using-python

Clients: Calling remote objects — Pyro 5.16.dev0 documentation. (2024). Readthedocs.io. https://pyro5.readthedocs.io/en/latest/clientcode.html

Freedman, M., & Lloyd, W. (n.d.). Network Communication and Remote Procedure Calls (RPCs) COS 418 + 518: (Advanced) Distributed Systems Lecture 2. Retrieved October 25, 2024, from https://www.cs.princeton.edu/courses/archive/fall19/cos418/docs/L2-rpc.pdf

12 Appendix

Phase 1:

```
PS C:\Users\minag> python -m Pyro5.nameserver
>>
Not starting broadcast server for IPv6.
NS running on localhost:9090 (::1)
URI = PYRO:Pyro.NameServer@localhost:9090
```

```
PS C:\Users\minag> python server.py
Server 1 is ready and waiting for clients.
```

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 1
Enter your TFN: 123456
You will receive a refund: $1000.00
PS C:\Users\minag>
```

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 2
Enter your biweekly net wages: 1500
Enter your tax withheld: 5000
Enter your estimated annual income: 60000
Do you have private health insurance (yes/no)? yes
You owe additional tax: $8875.00
PS C:\Users\minag>
```

server.py | client.py

C: > Users > minag > server.py > ...

```python
import Pyro5.api

@Pyro5.api.expose
class TaxCalculator:
    def calculate_tax(self, tfn):
        # Simulated calculation logic for TFN-based tax calculation
        if tfn == 123456:
            return 1000  # Example refund amount
        else:
            return "Error: TFN not found."

    def calculate_tax_with_biweekly_data(self, net_wages, tax_withheld):
        annual_income = net_wages * 26
        tax_due = annual_income * 0.325  # Tax rate applied based on the taxable income
        total_tax_due = tax_due - tax_withheld
        return total_tax_due

    def calculate_medicare_levy(self, annual_income):
        return annual_income * 0.02

    def calculate_medicare_levy_surcharge(self, annual_income, has_insurance):
        if not has_insurance and annual_income > 90000:
            if annual_income <= 105000:
                return annual_income * 0.01
            elif annual_income <= 140000:
                return annual_income * 0.0125
            else:
                return annual_income * 0.015
        return 0

def main():
    daemon = Pyro5.api.Daemon()
    ns = Pyro5.api.locate_ns()
    uri = daemon.register(TaxCalculator)
    ns.register("Aus.taxcalculator", uri)
    print("Server 1 is ready and waiting for clients.")
    daemon.requestLoop()

if __name__ == "__main__":
    main()
```

server.py client.py

C: > Users > minag > client.py > main

```python
import Pyro5.api

def main():
    tax_server = Pyro5.api.Proxy("PYRONAME:Aus.taxcalculator")

    choice = input("Enter 1 for tax calculation with TFN or 2 without TFN: ").strip()
    if choice == "1":
        tfn = input("Enter your TFN: ").strip()
        if tfn.isdigit():
            try:
                refund_or_payment = tax_server.calculate_tax(int(tfn))
                if isinstance(refund_or_payment, str):
                    print(refund_or_payment)
                elif refund_or_payment < 0:
                    print(f"You owe additional tax: ${-refund_or_payment:.2f}")
                else:
                    print(f"You will receive a refund: ${refund_or_payment:.2f}")
            except Exception as e:
                print(f"Error occurred during tax calculation: {e}")
        else:
            print("Error: TFN must be a valid number.")
    elif choice == "2":
        net_wages = float(input("Enter your biweekly net wages: "))
        tax_withheld = float(input("Enter your tax withheld: "))
        annual_income = input("Enter your estimated annual income: ")

        if not annual_income.isdigit():
            print("Error: Annual income must be a valid number.")
            return

        has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
        try:
            # Call the methods for tax calculation
            tax_due = tax_server.calculate_tax_with_biweekly_data(net_wages, tax_withheld)
            medicare_levy = tax_server.calculate_medicare_levy(float(annual_income))
            mls = tax_server.calculate_medicare_levy_surcharge(float(annual_income), has_insurance)
            total_tax = tax_due + medicare_levy + mls

            # Adjusting the logic for refund or tax owed
            if total_tax < 0:
                print(f"You will receive a refund: ${-total_tax:.2f}")
            else:
                print(f"You owe additional tax: ${total_tax:.2f}")
        except Exception as e:
            print(f"Error occurred during tax calculation: {e}")
```

Phase 1 TESTs

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 1
Enter your TFN: 123456
You will receive a refund: $1000.00
PS C:\Users\minag>
```

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 1
Enter your TFN: 654321
Error: TFN not found.
PS C:\Users\minag>
```

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 2
Enter your biweekly net wages: 1000
Enter your tax withheld: 10000
Enter your estimated annual income: 52000       found.
Do you have private health insurance (yes/no)? yes
You will receive a refund: $510.00
```

```
PS C:\Users\minag> python client.py
Enter 1 for tax calculation with TFN or 2 without TFN: 2
Enter your biweekly net wages: 1500
Enter your tax withheld: 5000
Enter your estimated annual income: 100000
Do you have private health insurance (yes/no)? no
You owe additional tax: $10675.00
PS C:\Users\minag>
```

Phase 2:

```
PS C:\Users\minag\__init__.py> python client1.py
Welcome to the Tax System

Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 1
Enter your Person ID: user101
Enter your password: UserPass101
Welcome, Alice Johnson!
```

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Delete account
8. Logout
Enter your choice: 1
Enter your new TFN: 47394759
Enter your gross income: 5000
Enter your tax withheld: 100
Enter your net income: 50000
Enter pay period start date (YYYY-MM-DD): 2024-01-01
Enter pay period end date (YYYY-MM-DD): 2024-12-12
Do you have private health insurance (yes/no)? yes
TFN 47394759 and tax record registered successfully.
```

```
mysql> SELECT * FROM users;
+----+-----------+----------+--------------+-------------------------+------------+-------+---------------+--------------+---------------------+
| id | person_id | tfn      | name         | email                   | password   | role  | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+--------------+-------------------------+------------+-------+---------------+--------------+---------------------+
|  8 | admin001  | NULL     | Admin User   | admin@example.com       | adminpass  | admin |          NULL |
NULL |                NULL |
|  9 | user101   | 47394759 | Alice Johnson | alice.johnson@example.com | UserPass101 | user  |      75000.00 |     10000.00 |                   1 |
```

```
mysql> SELECT * FROM payroll_records;
+----+-----------+-------------------+----------------+--------------+-----------+
| id | person_id | pay_period_start | pay_period_end | gross_income | tax_paid |
+----+-----------+-------------------+----------------+--------------+-----------+
| 28 | user101   | 2024-01-01       | 2024-12-12     |      5000.00 |   100.00 |
+----+-----------+-------------------+----------------+--------------+-----------+
1 row in set (0.00 sec)
```

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Delete account
8. Logout
Enter your choice: 2
You owe additional tax: $15875.00
```

```
Enter your choice: 3
Enter your biweekly net wages: 1000
Enter your tax withheld: 100
Enter your estimated annual income: 48000
Do you have private health insurance (yes/no)? yes
You owe additional tax: $9310.00
```

```
Enter your biweekly net wages: 1000
Enter your tax withheld: 10000
Enter your estimated annual income: 52000       fou
Do you have private health insurance (yes/no)? yes
You will receive a refund: $510.00
```

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Delete account
8. Logout
Enter your choice: 4
Date: 2024-10-23T13:21:44, Tax Amount: 9310.00, Refund: 0
Date: 2024-10-23T13:12:39, Tax Amount: 15875.00, Refund: 0
```

```
Enter your choice: 5

View Your Payroll Records:
Start: 2024-01-01, End: 2024-12-12, Gross Income: 5000.00, Tax Paid: 100.00
```

```
Enter your choice: 6

Update Profile:
Do you want to update your email? (yes/no): yes
Do you want to update your password? (yes/no): yes
Enter your new email: alice.johnson@gmail.com
Enter your new password: userpass101
Profile updated successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user101';
+----+-----------+----------+---------------+-----------------------+-------------+------+---------------+--------------+-----------------------+
| id | person_id | tfn      | name          | email                 | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+---------------+-----------------------+-------------+------+---------------+--------------+-----------------------+
|  9 | user101   | 47394759 | Alice Johnson | alice.johnson@gmail.com | userpass101 | user |      75000.00 |     10000.00 |                     1 |
+----+-----------+----------+---------------+-----------------------+-------------+------+---------------+--------------+-----------------------+
1 row in set (0.00 sec)
```

```
User Menu:
1. Register a new TFN
2. Calculate tax with TFN
3. Calculate tax without TFN
4. View tax history
5. View payroll records
6. Update profile
7. Logout
Enter your choice: 7
Logging out...
```

```
Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 2
Enter Admin ID: admin001
Enter Admin password: adminpass
Welcome, Admin Admin User!

Admin Menu:
1. View all users
2. Add a new user
3. Update a user
4. Delete a user
5. View all tax history
6. Search users
7. Display payroll records of users
8. Insert record
9. Update record
10. Delete record
11. Export user data to CSV
12. Export tax history to CSV
13. View Database Schema
14. View All Records in a Table
15. Exit Admin Menu
Enter your choice:
```

```
Enter your choice: 1
{'id': 8, 'person_id': 'admin001', 'tfn': None, 'name': 'Admin User', 'email'
: 'admin@example.com', 'password': 'adminpass', 'role': 'admin', 'annual_inco
me': None, 'tax_withheld': None, 'has_health_insurance': None}
{'id': 9, 'person_id': 'user101', 'tfn': '47394759', 'name': 'Alice Johnson',
 'email': 'alice.johnson@gmail.com', 'password': 'userpass101', 'role': 'user
', 'annual_income': '75000.00', 'tax_withheld': '10000.00', 'has_health_insur
ance': 1}{'id': 10, 'person_id': 'user102', 'tfn': None, 'name': 'Bob Smith',
 'email': 'bob.smith@example.com', 'password': 'UserPass102', 'role': 'user',
 'annual_income': '85000.00', 'tax_withheld': '12000.00', 'has_health_insuran
ce': 0}
{'id': 11, 'person_id': 'user103', 'tfn': None, 'name': 'Charlie Brown', 'ema
il': 'charlie.brown@example.com', 'password': 'UserPass103', 'role': 'user',
'annual_income': '95000.00', 'tax_withheld': '15000.00', 'has_health_insuranc
e': 1}
{'id': 12, 'person_id': 'user104', 'tfn': None, 'name': 'Diana Evans', 'email
': 'diana.evans@example.com', 'password': 'UserPass104', 'role': 'user', 'ann
ual_income': '65000.00', 'tax_withheld': '8000.00', 'has_health_insurance': 0
}
{'id': 13, 'person_id': 'user105', 'tfn': None, 'name': 'Edward Wright', 'ema
il': 'edward.wright@example.com', 'password': 'UserPass105', 'role': 'user',
'annual_income': '60000.00', 'tax_withheld': '6000.00', 'has_health_insurance
': 1}
{'id': 14, 'person_id': 'user106', 'tfn': None, 'name': 'Fiona Lee', 'email':
 'fiona.lee@example.com', 'password': 'UserPass106', 'role': 'user', 'annual_
income': '70000.00', 'tax_withheld': '7000.00', 'has_health_insurance': 0}
Enter your choice:
```

```
Add New User:
Enter the person ID: user107
Enter the name: John Doe
Enter the email: john.doe@gmail.com
Enter the password: userpass107
Enter the role (user/admin): user
Enter the TFN (optional): 47392071
Enter the annual income (optional): 100000
Enter the tax withheld (optional): 1000
Do you have private health insurance (yes/no): yes
User added successfully.
Enter your choice:
```

```
mysql> SELECT * FROM users WHERE person_id = 'user107';
+----+-----------+----------+----------+-------------------+------------+------+---------------+-------------+---------------------+
| id | person_id | tfn      | name     | email             | password   | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+----------+-------------------+------------+------+---------------+-------------+---------------------+
| 17 | user107   | 47392071 | John Doe | john.doe@gmail.com | userpass107 | user |     100000.00 |      1000.00 |                   1 |
+----+-----------+----------+----------+-------------------+------------+------+---------------+-------------+---------------------+
1 row in set (0.00 sec)
```

```
Enter your choice: 3

Update User:
Enter the person ID of the user to update: user102
Enter the new email (leave blank to keep unchanged):
Enter the new password (leave blank to keep unchanged):
Enter the new role (user/admin, leave blank to keep unchanged): admin
Enter the new TFN (leave blank to keep unchanged):
Enter the new annual income (leave blank to keep unchanged):
Enter the new tax withheld (leave blank to keep unchanged):
Do you have private health insurance (yes/no, leave blank to keep unchanged):
User updated successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user102';
+----+-----------+------+-----------+----------------------+-------------+-------+---------------+--------------+----------------------+
| id | person_id | tfn  | name      | email                | password    | role  | annual_income | tax_withheld | has_health_insurance |
+----+-----------+------+-----------+----------------------+-------------+-------+---------------+--------------+----------------------+
| 10 | user102   | NULL | Bob Smith | bob.smith@example.com | UserPass102 | admin |      85000.00 |     12000.00 |                    0 |
+----+-----------+------+-----------+----------------------+-------------+-------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

```
Enter your choice: 4

Delete User:
Enter the person ID of the user to delete: user102
Are you sure you want to delete this user? (yes/no): no
```

```
Enter your choice: 5
Person ID: user101, Date: 2024-10-23T13:21:44, Tax Amount: 9310.00, Refund: 0
Person ID: user101, Date: 2024-10-23T13:12:39, Tax Amount: 15875.00, Refund:
0
```

```
Enter your choice: 6

Search Users:
Enter person ID (leave blank for any): user104
Enter email (leave blank for any):
Enter role (user/admin, leave blank for any):
{'id': 12, 'person_id': 'user104', 'tfn': None, 'name': 'Diana Evans', 'email
': 'diana.evans@example.com', 'password': 'UserPass104', 'role': 'user', 'ann
ual_income': '65000.00', 'tax_withheld': '8000.00', 'has_health_insurance': 0
}
```

```
Enter your choice: 7

View All Payroll Records:
Enter person ID to filter (leave blank to view all records):
Person ID: user101, Start: 2024-01-01, End: 2024-12-12, Gross Income: 5000.00
, Tax Paid: 100.00
```

```
Enter your choice: 8

Insert Record:
Enter the table name: users
Enter field name (or 'done' to finish): name
Enter value for 'name': Roman Row
Enter field name (or 'done' to finish): person_id
Enter value for 'person_id': user108
Enter field name (or 'done' to finish): tfn
Enter value for 'tfn': 57309264
Enter field name (or 'done' to finish): email
Enter value for 'email': roman.row@gmail.com
Enter field name (or 'done' to finish): password
Enter value for 'password': userpass107
Enter field name (or 'done' to finish): role
Enter value for 'role': user
Enter field name (or 'done' to finish): annual_income
Enter value for 'annual_income': 80000
Enter field name (or 'done' to finish): tax_withheld
Enter value for 'tax_withheld': 5000
Enter field name (or 'done' to finish): has_health_insurance
Enter value for 'has_health_insurance': 1
Enter field name (or 'done' to finish): done
Record inserted successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email               | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |      80000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

```
Enter your choice: 9

Update Record:
Enter the table name: users
Enter the record ID to update: 18
Enter field name to update (or 'done' to finish): annual_income
Enter new value for 'annual_income': 100000
Enter field name to update (or 'done' to finish): done
Record updated successfully.
```

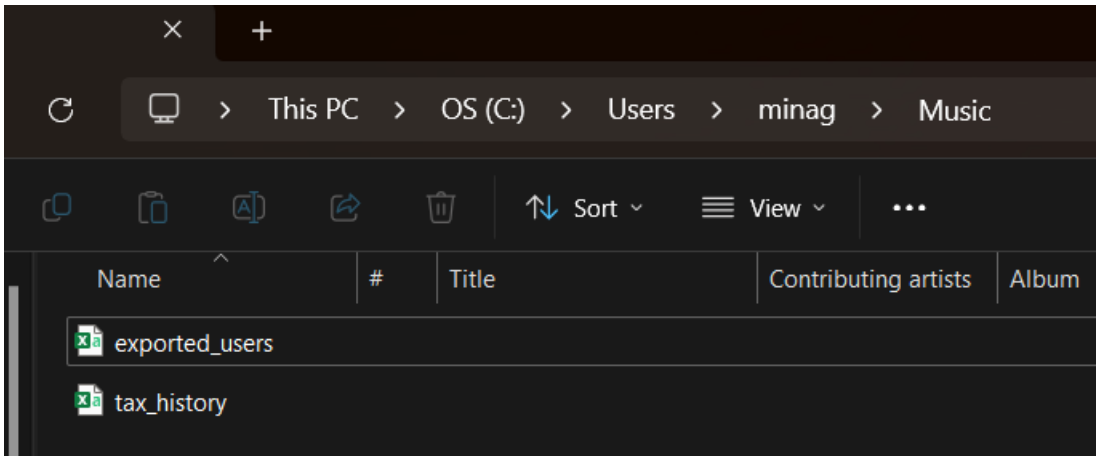
```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email               | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |     100000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

```
Enter your choice: 10

Delete Record:
Enter the table name: users
Enter the record ID to delete: 18
Record deleted successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
Empty set (0.00 sec)
```

```
Enter your choice: 11
Enter the file path for the CSV export (default: exported_users.csv): C:\Users\minag\Music\exported_users.csv
User data exported successfully to C:\Users\minag\Music\exported_users.csv.
```

```
Enter your choice: 12
Enter the file path for the CSV export (default: exported_tax_history.csv): C:\Users\minag\Music\tax_history.csv
Tax history data exported successfully to C:\Users\minag\Music\tax_history.csv.
```



```
Enter your choice: 13

--- Database Schema ---

Table: payroll_records
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  pay_period_start - date (Key: )
  pay_period_end - date (Key: )
  gross_income - decimal(10,2) (Key: )
  tax_paid - decimal(10,2) (Key: )

Table: tax_history
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  calculation_date - datetime (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  tax_amount - decimal(10,2) (Key: )
  is_refund - tinyint(1) (Key: )

Table: users
  id - int (Key: PRI)
  person_id - varchar(10) (Key: UNI)
  tfn - varchar(9) (Key: )
  name - varchar(100) (Key: )
  email - varchar(100) (Key: )
  password - varchar(50) (Key: )
  role - enum('user','admin') (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  has_health_insurance - tinyint(1) (Key: )
```

```
Enter your choice: 14
Enter the table name to view records: users
{'id': 8, 'person_id': 'admin001', 'tfn': None, 'name': 'Admin User', 'email': 'admin@example.com', 'password': 'adminpass', 'role': 'admin', 'annual_income
': None, 'tax_withheld': None, 'has_health_insurance': None}
{'id': 9, 'person_id': 'user101', 'tfn': '47394759', 'name': 'Alice Johnson', 'email': 'alice.johnson@gmail.com', 'password': 'userpass101', 'role': 'user',
 'annual_income': '75000.00', 'tax_withheld': '10000.00', 'has_health_insurance': 1}
{'id': 10, 'person_id': 'user102', 'tfn': None, 'name': 'Bob Smith', 'email': 'bob.smith@example.com', 'password': 'UserPass102', 'role': 'user', 'annual_in
come': '85000.00', 'tax_withheld': '12000.00', 'has_health_insurance': 0}
{'id': 11, 'person_id': 'user103', 'tfn': None, 'name': 'Charlie Brown', 'email': 'charlie.brown@example.com', 'password': 'UserPass103', 'role': 'user', 'a
nnual_income': '95000.00', 'tax_withheld': '15000.00', 'has_health_insurance': 1}
{'id': 12, 'person_id': 'user104', 'tfn': None, 'name': 'Diana Evans', 'email': 'diana.evans@example.com', 'password': 'UserPass104', 'role': 'user', 'annua
l_income': '65000.00', 'tax_withheld': '8000.00', 'has_health_insurance': 0}
{'id': 13, 'person_id': 'user105', 'tfn': None, 'name': 'Edward Wright', 'email': 'edward.wright@example.com', 'password': 'UserPass105', 'role': 'user', 'a
nnual_income': '60000.00', 'tax_withheld': '6000.00', 'has_health_insurance': 1}
{'id': 14, 'person_id': 'user106', 'tfn': None, 'name': 'Fiona Lee', 'email': 'fiona.lee@example.com', 'password': 'UserPass106', 'role': 'user', 'annual_in
come': '70000.00', 'tax_withheld': '7000.00', 'has_health_insurance': 0}
{'id': 17, 'person_id': 'user107', 'tfn': '47392071', 'name': 'John Doe', 'email': 'john.doe@gmail.com', 'password': 'userpass107', 'role': 'user', 'annual_
income': '100000.00', 'tax_withheld': '1000.00', 'has_health_insurance': 1}
```

Bonus Marks:

```
Enter your choice: 13

--- Database Schema ---

Table: payroll_records
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  pay_period_start - date (Key: )
  pay_period_end - date (Key: )
  gross_income - decimal(10,2) (Key: )
  tax_paid - decimal(10,2) (Key: )

Table: tax_history
  id - int (Key: PRI)
  person_id - varchar(10) (Key: MUL)
  calculation_date - datetime (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  tax_amount - decimal(10,2) (Key: )
  is_refund - tinyint(1) (Key: )

Table: tfn_free_users
  id - int (Key: PRI)
  person_id - varchar(50) (Key: UNI)
  password - varchar(255) (Key: )
  name - varchar(255) (Key: )
  email - varchar(255) (Key: )
  created_at - timestamp (Key: )

Table: users
  id - int (Key: PRI)
  person_id - varchar(10) (Key: UNI)
  tfn - varchar(9) (Key: )
  name - varchar(100) (Key: )
  email - varchar(100) (Key: )
  password - varchar(50) (Key: )
  role - enum('user','admin') (Key: )
  annual_income - decimal(10,2) (Key: )
  tax_withheld - decimal(10,2) (Key: )
  has_health_insurance - tinyint(1) (Key: )
```

```
Enter your choice: 14
Enter the table name to view records: users
{'id': 8, 'person_id': 'admin001', 'tfn': None, 'name': 'Admin User', 'email': 'admin@example.com', 'password': 'adminpass', 'role': 'admin', 'annual_income
': None, 'tax_withheld': None, 'has_health_insurance': None}
{'id': 9, 'person_id': 'user101', 'tfn': '47394759', 'name': 'Alice Johnson', 'email': 'alice.johnson@gmail.com', 'password': 'userpass101', 'role': 'user',
 'annual_income': '75000.00', 'tax_withheld': '10000.00', 'has_health_insurance': 1}
{'id': 10, 'person_id': 'user102', 'tfn': None, 'name': 'Bob Smith', 'email': 'bob.smith@example.com', 'password': 'UserPass102', 'role': 'admin', 'annual_i
ncome': '85000.00', 'tax_withheld': '12000.00', 'has_health_insurance': 0}
{'id': 11, 'person_id': 'user103', 'tfn': None, 'name': 'Charlie Brown', 'email': 'charlie.brown@example.com', 'password': 'UserPass103', 'role': 'user', 'a
nnual_income': '95000.00', 'tax_withheld': '15000.00', 'has_health_insurance': 1}
{'id': 12, 'person_id': 'user104', 'tfn': None, 'name': 'Diana Evans', 'email': 'diana.evans@example.com', 'password': 'UserPass104', 'role': 'user', 'annua
l_income': '65000.00', 'tax_withheld': '8000.00', 'has_health_insurance': 0}
{'id': 13, 'person_id': 'user105', 'tfn': None, 'name': 'Edward Wright', 'email': 'edward.wright@example.com', 'password': 'UserPass105', 'role': 'user', 'a
nnual_income': '60000.00', 'tax_withheld': '6000.00', 'has_health_insurance': 1}
{'id': 14, 'person_id': 'user106', 'tfn': None, 'name': 'Fiona Lee', 'email': 'fiona.lee@example.com', 'password': 'UserPass106', 'role': 'user', 'annual_in
come': '70000.00', 'tax_withheld': '7000.00', 'has_health_insurance': 0}
{'id': 17, 'person_id': 'user107', 'tfn': '47392071', 'name': 'John Doe', 'email': 'john.doe@gmail.com', 'password': 'userpass107', 'role': 'user', 'annual_
income': '100000.00', 'tax_withheld': '1000.00', 'has_health_insurance': 1}
{'id': 20, 'person_id': 'user110', 'tfn': '47502438', 'name': 'Kelly Reynolds', 'email': 'kelly.reynolds@gmail.com', 'password': 'userpass110', 'role': 'use
r', 'annual_income': None, 'tax_withheld': None, 'has_health_insurance': None}
```

```
Enter your choice: 14
Enter the table name to view records: TFN_free_users
{'id': 1, 'person_id': 'user109', 'password': 'userpass109', 'name': 'Olive Green', 'email': 'oilve.green@gmail.com', 'created_at': '2024-10-23T19:04:16'}
```

```
Enter your choice: 14
Enter the table name to view records: payroll_records
{'id': 28, 'person_id': 'user101', 'pay_period_start': '2024-01-01', 'pay_period_end': '2024-12-12', 'gross_income': '5000.00', 'tax_paid': '100.00'}
```

```
Enter your choice: 14
Enter the table name to view records: tax_history
{'id': 5, 'person_id': 'user101', 'calculation_date': '2024-10-23T13:12:39', 'annual_income': '75000.00', 'tax_withheld': '10000.00', 'tax_amount': '15875.0
0', 'is_refund': 0}
{'id': 6, 'person_id': 'user101', 'calculation_date': '2024-10-23T13:21:44', 'annual_income': '48000.00', 'tax_withheld': '100.00', 'tax_amount': '9310.00',
 'is_refund': 0}
```

```
Enter your choice: 8

Insert Record:
Enter the table name: users
Enter field name (or 'done' to finish): name
Enter value for 'name': Roman Row
Enter field name (or 'done' to finish): person_id
Enter value for 'person_id': user108
Enter field name (or 'done' to finish): tfn
Enter value for 'tfn': 57309264
Enter field name (or 'done' to finish): email
Enter value for 'email': roman.row@gmail.com
Enter field name (or 'done' to finish): password
Enter value for 'password': userpass107
Enter field name (or 'done' to finish): role
Enter value for 'role': user
Enter field name (or 'done' to finish): annual_income
Enter value for 'annual_income': 80000
Enter field name (or 'done' to finish): tax_withheld
Enter value for 'tax_withheld': 5000
Enter field name (or 'done' to finish): has_health_insurance
Enter value for 'has_health_insurance': 1
Enter field name (or 'done' to finish): done
Record inserted successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email               | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |      80000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

```
Enter your choice: 9

Update Record:
Enter the table name: users
Enter the record ID to update: 18
Enter field name to update (or 'done' to finish): annual_income
Enter new value for 'annual_income': 100000
Enter field name to update (or 'done' to finish): done
Record updated successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name      | email               | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
| 18 | user108   | 57309264 | Roman Row | roman.row@gmail.com | userpass107 | user |     100000.00 |      5000.00 |                    1 |
+----+-----------+----------+-----------+---------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

```
Enter your choice: 10

Delete Record:
Enter the table name: users
Enter the record ID to delete: 18
Record deleted successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user108';
Empty set (0.00 sec)
```

```python
def insert_record(self, table, fields):
    self.ensure_connection()
    try:
        if table == "users" and "person_id" not in fields:
            return "Error: 'person_id' is required for users table."

        if table == "payroll_records" and "person_id" in fields:
            person_id = fields["person_id"]
            if not self.user_exists(person_id):
                return f"Error: User with person_id '{person_id}' does not exist. Please add the user first."

            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT COUNT(*) AS record_count FROM payroll_records WHERE person_id = %s"
            cursor.execute(query, (person_id,))
            record_count = cursor.fetchone()["record_count"]
            cursor.close()

            if record_count >= 26:
                return f"Error: Cannot exceed 26 payroll records for taxpayer {person_id}."

        cursor = self.connection.cursor()
        placeholders = ', '.join(['%s'] * len(fields))
        query = f"INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})"
        cursor.execute(query, list(fields.values()))
        self.connection.commit()
        cursor.close()
        return f"Record inserted into {table}."
    except Error as e:
        return f"Error inserting record into {table}: {e}"
```

User registration:

```
PS C:\Users\minag> cd C:\Users\minag\__init__.py
PS C:\Users\minag\__init__.py> python client1.py
Welcome to the Tax System

Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 3

User Registration:
Enter your Person ID: user109
Enter your password: userpass109
Enter your name (optional): Olive Green
Enter your email (optional): oilve.green@gmail.com
Enter your TFN (optional, leave blank if you don't have one):
User 'user109' registered successfully.
```

```
mysql> SELECT * FROM TFN_free_users WHERE person_id = 'user109';
+----+-----------+-------------+-------------+----------------------+---------------------+
| id | person_id | password    | name        | email                | created_at          |
+----+-----------+-------------+-------------+----------------------+---------------------+
|  1 | user109   | userpass109 | Olive Green | oilve.green@gmail.com | 2024-10-23 19:04:16 |
+----+-----------+-------------+-------------+----------------------+---------------------+
1 row in set (0.00 sec)
```

```
mysql> DESCRIBE TFN_free_users;
+------------+--------------+------+-----+-------------------+-------------------+
| Field      | Type         | Null | Key | Default           | Extra             |
+------------+--------------+------+-----+-------------------+-------------------+
| id         | int          | NO   | PRI | NULL              | auto_increment    |
| person_id  | varchar(50)  | NO   | UNI | NULL              |                   |
| password   | varchar(255) | NO   |     | NULL              |                   |
| name       | varchar(255) | YES  |     | NULL              |                   |
| email      | varchar(255) | YES  |     | NULL              |                   |
| created_at | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+--------------+------+-----+-------------------+-------------------+
6 rows in set (0.01 sec)
```

```
Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 3

User Registration:
Enter your Person ID: user110
Enter your password: userpass110
Enter your name (optional): Kelly Reynolds
Enter your email (optional): kelly.reynolds@gmail.com
Enter your TFN (optional, leave blank if you don't have one): 47502438
User 'user110' registered successfully.
```

```
mysql> SELECT * FROM users WHERE person_id = 'user110';
+----+-----------+----------+----------------+-------------------------+-------------+------+---------------+--------------+----------------------+
| id | person_id | tfn      | name           | email                   | password    | role | annual_income | tax_withheld | has_health_insurance |
+----+-----------+----------+----------------+-------------------------+-------------+------+---------------+--------------+----------------------+
| 20 | user110   | 47502438 | Kelly Reynolds | kelly.reynolds@gmail.com | userpass110 | user |          NULL |         NULL |                 NULL |
+----+-----------+----------+----------------+-------------------------+-------------+------+---------------+--------------+----------------------+
1 row in set (0.00 sec)
```

```
Main Menu:
1. User Login
2. Admin Login
3. User Registration
4. Exit
Enter your choice: 3

User Registration:
Enter your Person ID: user101
Enter your password: userpass101
Enter your name (optional): Alice Johnson
Enter your email (optional): alice.johnson@gmail.com
Enter your TFN (optional, leave blank if you don't have one):
Error: User with person_id 'user101' already exists.
```

Database:

```
CREATE TABLE tfn_free_users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(50) NOT NULL UNIQUE,
        password VARCHAR(255) NOT NULL,
        name VARCHAR(255),
        email VARCHAR(255),
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE tax_history (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(10),
        calculation_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        annual_income DECIMAL(10,2),
        tax_withheld DECIMAL(10,2),
        tax_amount DECIMAL(10,2),
        is_refund TINYINT(1),
        FOREIGN KEY (person_id) REFERENCES users(person_id)
);
```

```
CREATE TABLE users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(10) NOT NULL UNIQUE,
        tfn VARCHAR(9),
        name VARCHAR(100),
        email VARCHAR(100),
        password VARCHAR(50),
        role ENUM('user', 'admin') NOT NULL DEFAULT 'user',
        annual_income DECIMAL(10,2),
        tax_withheld DECIMAL(10,2),
        has_health_insurance TINYINT(1),
        UNIQUE (person_id)
);

CREATE TABLE payroll_records (
        id INT AUTO_INCREMENT PRIMARY KEY,
        person_id VARCHAR(10),
        pay_period_start DATE,
        pay_period_end DATE,
        gross_income DECIMAL(10,2),
        tax_paid DECIMAL(10,2),
        FOREIGN KEY (person_id) REFERENCES users(person_id)
);
```