

PERSONAL INCOME TAX SYSTEM (PITS)

Assignment 3: Distributed Systems

Unit Code: CSI3344D

Due Date: Fri, 25 Oct 2024 21:29

Personal Income Tax System (PITS)

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

Executive Summary	1
1. Introduction.....	4
2. Background	5
3. Scope.....	6
3.1 Functional Scope.....	6
3.2 Technical Scope	6
3.3 Project Scope	7
4. Main Body	8
4.1 Understanding of Concepts/Techniques	8
5. Remote Procedure Call	9
5.1 What is RPC?	9
5.2 RPC Process.....	9
6. RMI Diagram	10
6.1 Explanation	11
6.2 RPC Flow.....	12
7. Application Requirements	13
8. Application Design and Implementation Procedure (SDLC)	16
9. Implementation Procedure	20
9.1 Phase 1 Implementation	20
System Overview	20
Architecture.....	20
User Manual.....	21
Code Documentation	22
1. Phase 1: Test Cases	24
2. Phase 2 Implementation.....	26
Tax System as an Ordinary user.....	26
Tax system as an Administrative User	31
Bonus Marks Criteria	40
Implementing the Database Schema	40
Display all Available data records.....	41
CRUD operation for data records for Admin users	42
User Registration	45
Code Documentation Phase 2	48

Personal Income Tax System (PITS)

User Manual.....	71
4. Phase 2: Test Cases	82
10 Conclusion	85
11 References	86
12 Appendix	87

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.)

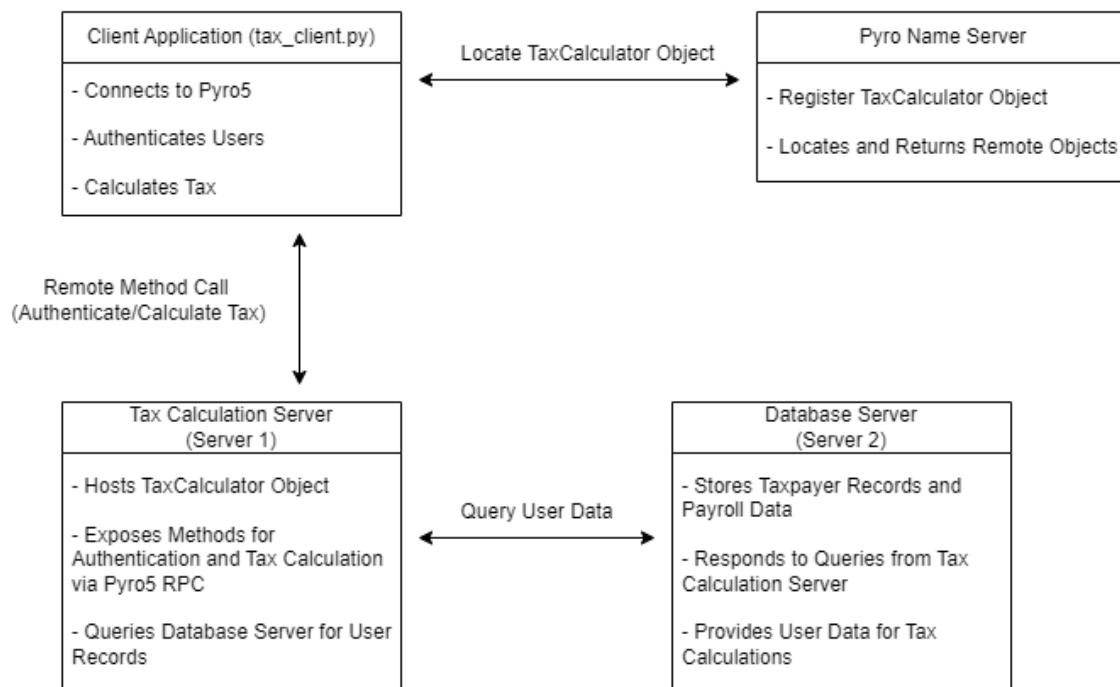
5.2 RPC Process

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.

Personal Income Tax System (PITS)

- 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> █
```

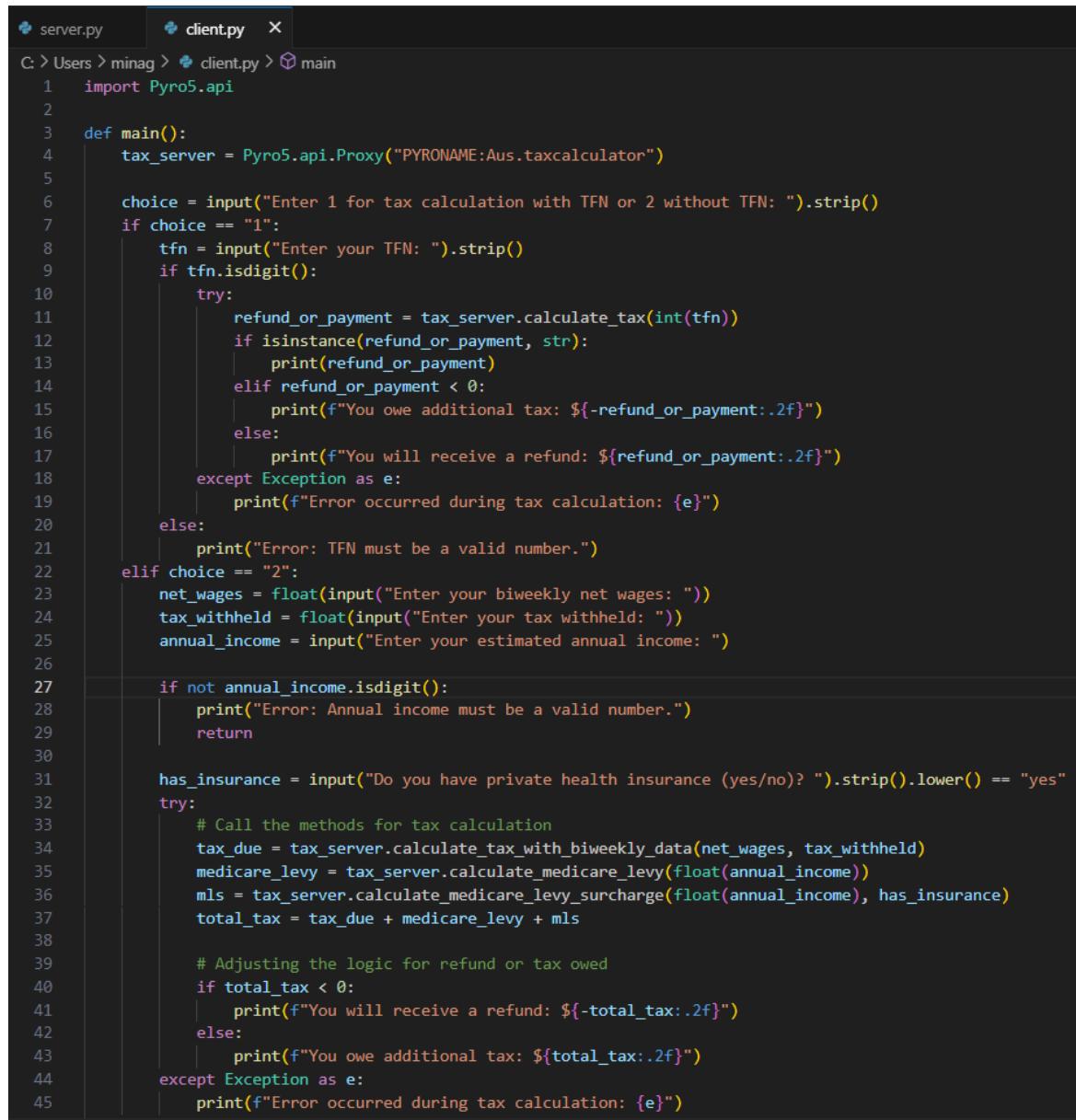
Code Documentation

server.py

```
server.py  x  client.py
C > Users > minag > server.py > ...
1  import Pyro5.api
2
3  @Pyro5.api.expose
4  class TaxCalculator:
5      def calculate_tax(self, tfn):
6          # Simulated calculation logic for TFN-based tax calculation
7          if tfn == 123456:
8              return 1000 # Example refund amount
9          else:
10              return "Error: TFN not found."
11
12      def calculate_tax_with_biweekly_data(self, net_wages, tax_withheld):
13          annual_income = net_wages * 26
14          tax_due = annual_income * 0.325 # Tax rate applied based on the taxable income
15          total_tax_due = tax_due - tax_withheld
16          return total_tax_due
17
18      def calculate_medicare_levy(self, annual_income):
19          return annual_income * 0.02
20
21      def calculate_medicare_levy_surcharge(self, annual_income, has_insurance):
22          if not has_insurance and annual_income > 90000:
23              if annual_income <= 105000:
24                  return annual_income * 0.01
25              elif annual_income <= 140000:
26                  return annual_income * 0.0125
27              else:
28                  return annual_income * 0.015
29          return 0
30
31  def main():
32      daemon = Pyro5.api.Daemon()
33      ns = Pyro5.api.locate_ns()
34      uri = daemon.register(TaxCalculator)
35      ns.register("Aus.taxcalculator", uri)
36      print("Server 1 is ready and waiting for clients.")
37      daemon.requestLoop()
38
39  if __name__ == "__main__":
40      main()
```

Personal Income Tax System (PITS)

client.py



```
C:\> Users > minag > client.py > main
1 import Pyro5.api
2
3 def main():
4     tax_server = Pyro5.api.Proxy("PYRONAME:Aus.taxcalculator")
5
6     choice = input("Enter 1 for tax calculation with TFN or 2 without TFN: ").strip()
7     if choice == "1":
8         tfn = input("Enter your TFN: ").strip()
9         if tfn.isdigit():
10             try:
11                 refund_or_payment = tax_server.calculate_tax(int(tfn))
12                 if isinstance(refund_or_payment, str):
13                     print(refund_or_payment)
14                 elif refund_or_payment < 0:
15                     print(f"You owe additional tax: ${-refund_or_payment:.2f}")
16                 else:
17                     print(f"You will receive a refund: ${refund_or_payment:.2f}")
18             except Exception as e:
19                 print(f"Error occurred during tax calculation: {e}")
20         else:
21             print("Error: TFN must be a valid number.")
22     elif choice == "2":
23         net_wages = float(input("Enter your biweekly net wages: "))
24         tax_withheld = float(input("Enter your tax withheld: "))
25         annual_income = input("Enter your estimated annual income: ")
26
27         if not annual_income.isdigit():
28             print("Error: Annual income must be a valid number.")
29             return
30
31         has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
32         try:
33             # Call the methods for tax calculation
34             tax_due = tax_server.calculate_tax_with_biweekly_data(net_wages, tax_withheld)
35             medicare_levy = tax_server.calculate_medicare_levy(float(annual_income))
36             mls = tax_server.calculate_medicare_levy_surcharge(float(annual_income), has_insurance)
37             total_tax = tax_due + medicare_levy + mls
38
39             # Adjusting the logic for refund or tax owed
40             if total_tax < 0:
41                 print(f"You will receive a refund: ${-total_tax:.2f}")
42             else:
43                 print(f"You owe additional tax: ${total_tax:.2f}")
44         except Exception as e:
45             print(f"Error occurred during tax calculation: {e}")
```

Personal Income Tax System (PITS)

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> █
```

Personal Income Tax System (PITS)

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

Personal Income Tax System (PITS)

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.

Personal Income Tax System (PITS)

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 | 75000.00 | 10000.00 | 1 |
| 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      for
Do you have private health insurance (yes/no)? yes
You will receive a refund: $510.00
```

Personal Income Tax System (PITS)

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)
```

Personal Income Tax System (PITS)

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: |
```

Personal Income Tax System (PITS)

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_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_income': '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', 'annual_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', 'annual_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', '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: 1
```

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: 1
```

Personal Income Tax System (PITS)

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): yes
User updated successfully.
```

Snapshot of the database updated ordinary user to admin role by an 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
```

Personal Income Tax System (PITS)

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', 'annual_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
```

Personal Income Tax System (PITS)

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)
```

Personal Income Tax System (PITS)

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)
```

Personal Income Tax System (PITS)

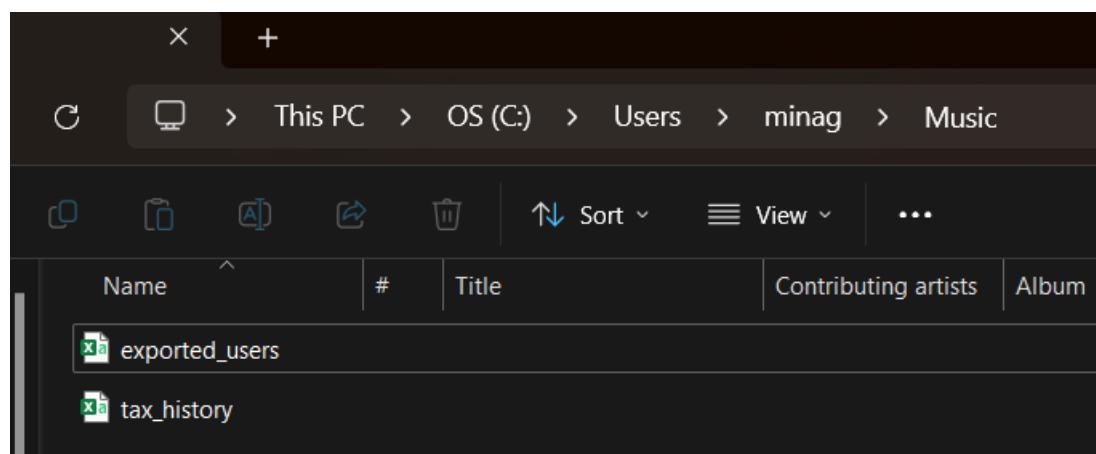
If the user chooses option 11 or 12,

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.
```



Personal Income Tax System (PITS)

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: )
```

Personal Income Tax System (PITS)

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.

```
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_income": "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", "annual_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", "annual_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", "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}, {"id": 17, "person_id": "user107", "tfn": "47392871", "name": "John Doe", "email": "john.doe@gmail.com", "password": "userpass107", "role": "user", "annual_income": "100000.00", "tax_withheld": "10000.00", "has_health_insurance": 1}]
```

Bonus Marks Criteria

Implementing the Database Schema

For viewing ability for Admin User

```
mysql> EXIT;
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: )
```

Personal Income Tax System (PITS)

Display all Available data records

, from any table Admin user wish to view

users table information

```
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_income": "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", "annual_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", "annual_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", "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}
{"id": 15, "person_id": "user107", "tfn": "47392071", "name": "John Doe", "email": "john.doe@gmail.com", "password": "userpass107", "role": "user", "annual_income": "100000.00", "tax_withheld": "10000.00", "has_health_insurance": 1}
{"id": 16, "person_id": "user108", "tfn": "47502438", "name": "Kelly Reynolds", "email": "kelly.reynolds@gmail.com", "password": "userpass108", "role": "user", "annual_income": None, "tax_withheld": None, "has_health_insurance": None}
```

TFN_free_users table information

```
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"}
```

payroll_records table information

```
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"}
```

tax_history table information

```
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": "15875.00", "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}
```

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)
```

Personal Income Tax System (PITS)

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)
```

Personal Income Tax System (PITS)

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): olive.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 | olive.green@gmail.com | 2024-10-23 19:04:16 |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Personal Income Tax System (PITS)

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

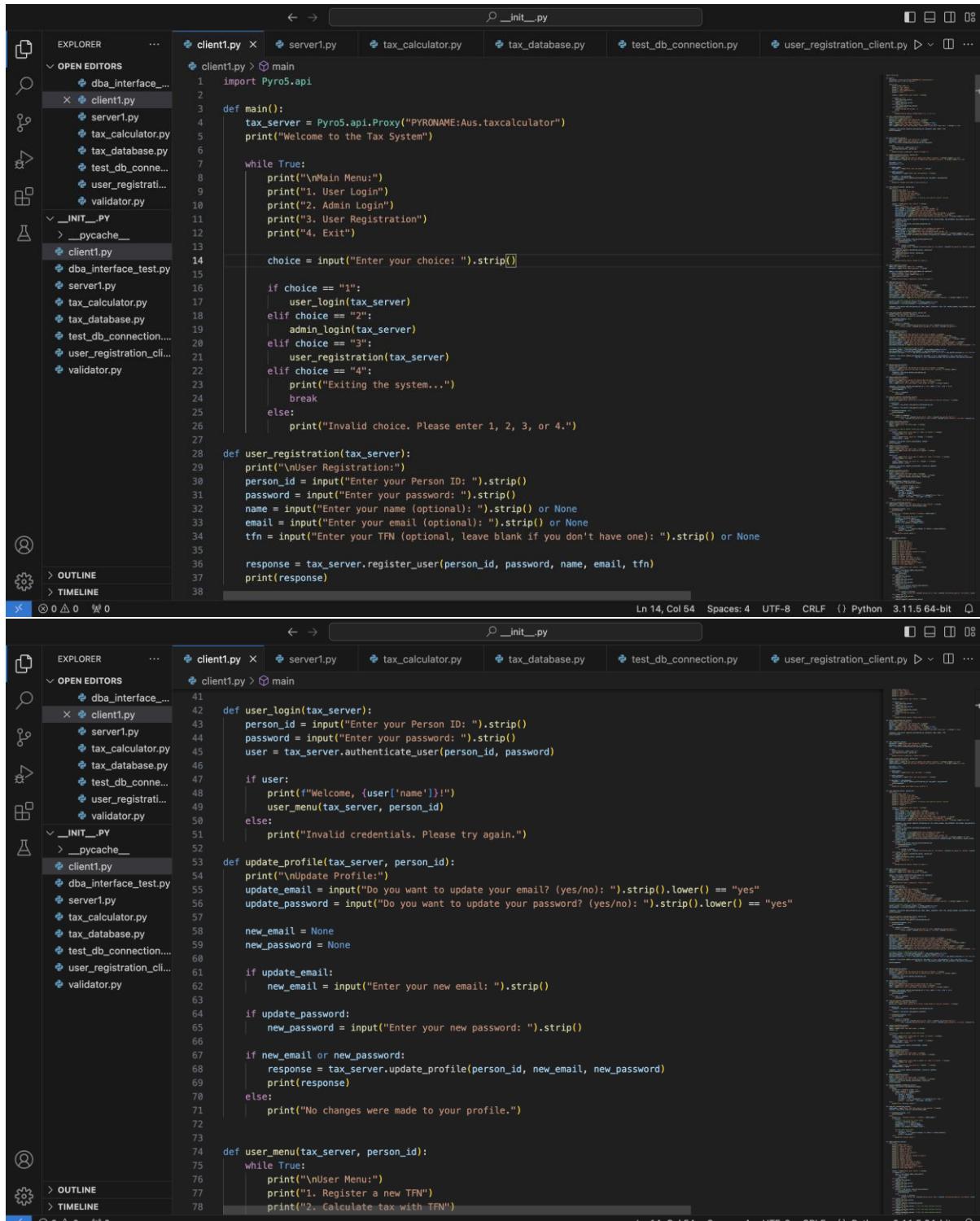
Enter your TFN (optional, leave blank if you don't have one):

Error: User with person_id 'user101' already exists.

Personal Income Tax System (PITS)

Code Documentation Phase 2

Client1.py



```
client1.py > main
1  import Pyro5.api
2
3  def main():
4      tax_server = Pyro5.api.Proxy("PYRONAME:Aus.taxcalculator")
5      print("Welcome to the Tax System")
6
7      while True:
8          print("\nMain Menu:")
9          print("1. User Login")
10         print("2. Admin Login")
11         print("3. User Registration")
12         print("4. Exit")
13
14         choice = input("Enter your choice: ").strip()
15
16         if choice == "1":
17             user_login(tax_server)
18         elif choice == "2":
19             admin_login(tax_server)
20         elif choice == "3":
21             user_registration(tax_server)
22         elif choice == "4":
23             print("Exiting the system...")
24             break
25         else:
26             print("Invalid choice. Please enter 1, 2, 3, or 4.")
27
28     def user_registration(tax_server):
29         print("\nUser Registration:")
30         person_id = input("Enter your Person ID: ").strip()
31         password = input("Enter your password: ").strip()
32         name = input("Enter your name (optional): ").strip() or None
33         email = input("Enter your email (optional): ").strip() or None
34         tfn = input("Enter your TFN (optional, leave blank if you don't have one): ").strip() or None
35
36         response = tax_server.register_user(person_id, password, name, email, tfn)
37         print(response)
38
```



```
client1.py > main
41
42     def user_login(tax_server):
43         person_id = input("Enter your Person ID: ").strip()
44         password = input("Enter your password: ").strip()
45         user = tax_server.authenticate_user(person_id, password)
46
47         if user:
48             print(f"Welcome, {user['name']}!")
49             user_menu(tax_server, person_id)
50         else:
51             print("Invalid credentials. Please try again.")
52
53     def update_profile(tax_server, person_id):
54         print("\nUpdate Profile:")
55         update_email = input("Do you want to update your email? (yes/no): ").strip().lower() == "yes"
56         update_password = input("Do you want to update your password? (yes/no): ").strip().lower() == "yes"
57
58         new_email = None
59         new_password = None
60
61         if update_email:
62             new_email = input("Enter your new email: ").strip()
63
64         if update_password:
65             new_password = input("Enter your new password: ").strip()
66
67         if new_email or new_password:
68             response = tax_server.update_profile(person_id, new_email, new_password)
69             print(response)
70         else:
71             print("No changes were made to your profile.")
72
73
74     def user_menu(tax_server, person_id):
75         while True:
76             print("\nUser Menu:")
77             print("1. Register a new TFN")
78             print("2. Calculate tax with TFN")
```

Personal Income Tax System (PITS)

```
client1.py > main
74 def user_menu(tax_server, person_id):
75     while True:
76         print("\nUser Menu:")
77         print("1. Register a new TFN")
78         print("2. Calculate tax with TFN")
79         print("3. Calculate tax without TFN")
80         print("4. View tax history")
81         print("5. View payroll records") # Display user-specific payroll records
82         print("6. Update profile")
83         print("7. Logout")
84
85         choice = input("Enter your choice: ").strip()
86         if choice == "1":
87             tfn = input("Enter your new TFN: ").strip()
88             gross_income = float(input("Enter your gross income: "))
89             tax_withheld = float(input("Enter your tax withheld: "))
90             net_income = float(input("Enter your net income: "))
91             pay_period_start = input("Enter pay period start date (YYYY-MM-DD): ").strip()
92             pay_period_end = input("Enter pay period end date (YYYY-MM-DD): ").strip()
93             has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
94
95             response = tax_server.register_tfn(person_id, tfn, gross_income, tax_withheld, net_income, pay_period_start)
96             print(response)
97         elif choice == "2":
98             response = tax_server.calculate_tax(person_id)
99             print(response)
100        elif choice == "3":
101            biweekly_wages = float(input("Enter your biweekly net wages: "))
102            tax_withheld = float(input("Enter your tax withheld: "))
103            annual_income = float(input("Enter your estimated annual income: "))
104            has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
105            response = tax_server.calculate_tax_without_tfn(person_id, biweekly_wages, tax_withheld, annual_income, has_insurance)
106            print(response)
107        elif choice == "4":
108            history = tax_server.view_tax_history(person_id)
109            if isinstance(history, str):
110                print(history)
111            else:
112                for record in history:
113                    print(f"\nDate: {record['calculation_date']}, Tax Amount: {record['tax_amount']}, Refund: {record['is_refund']}")
114
115        elif choice == "5":
116            view_user_payroll_records(tax_server, person_id)
117
118        elif choice == "6":
119            update_profile(tax_server, person_id)
120
121        elif choice == "":
122            print("Logging out...")
123            break
124
125        else:
126            print("Invalid choice. Please try again.")
```

Personal Income Tax System (PITS)

```
client1.py
124
125     def admin_login(tax_server):
126         admin_id = input("Enter Admin ID: ").strip()
127         password = input("Enter Admin password: ").strip()
128
129         admin = tax_server.authenticate_user(admin_id, password)
130         if admin and admin['role'] == 'admin':
131             print("Welcome, Admin (admin['name'])")
132             admin_menu(tax_server)
133         else:
134             print("Invalid admin credentials. Please try again.")
135
136     def add_user(tax_server):
137         print("\nAdd New User:")
138         person_id = input("Enter the person ID: ").strip()
139         name = input("Enter the name: ").strip()
140         email = input("Enter the email: ").strip()
141         password = input("Enter the password: ").strip()
142         role = input("Enter the role (user/admin): ").strip().lower()
143         tfn = input("Enter the TFN (optional): ").strip() or None
144         annual_income = input("Enter the annual income (optional): ").strip()
145         tax_withheld = input("Enter the tax withheld (optional): ").strip()
146         has_health_insurance = input("Do you have private health insurance (yes/no): ").strip().lower() == "yes"
147
148         # Convert optional inputs to None if empty
149         annual_income = float(annual_income) if annual_income else None
150         tax_withheld = float(tax_withheld) if tax_withheld else None
151
152         response = tax_server.add_user(person_id, name, email, password, role, tfn, annual_income, tax_withheld, has_health_insurance)
153         print(response)
154
155
156     def view_user_payroll_records(tax_server, person_id):
157         print("\nView Your Payroll Records:")
158         response = tax_server.view_payroll_records(person_id)
159
160         if isinstance(response, str):
161             print(response)
162
163
164     def update_user(tax_server):
165         print("\nUpdate User:")
166         person_id = input("Enter the person ID of the user to update: ").strip()
167         new_email = input("Enter the new email (leave blank to keep unchanged): ").strip()
168         new_password = input("Enter the new password (leave blank to keep unchanged): ").strip()
169         new_role = input("Enter the new role (user/admin, leave blank to keep unchanged): ").strip().lower()
170         new_tfn = input("Enter the new TFN (leave blank to keep unchanged): ").strip()
171         new_annual_income = input("Enter the new annual income (leave blank to keep unchanged): ").strip()
172         new_tax_withheld = input("Enter the new tax withheld (leave blank to keep unchanged): ").strip()
173         has_health_insurance = input("Do you have private health insurance (yes/no, leave blank to keep unchanged): ").strip()
174
175         # Convert inputs to appropriate types or None
176         new_annual_income = float(new_annual_income) if new_annual_income else None
177         new_tax_withheld = float(new_tax_withheld) if new_tax_withheld else None
178         has_health_insurance = True if has_health_insurance == "yes" else False if has_health_insurance == "no" else None
179
180         response = tax_server.update_user(person_id, new_email or None, new_password or None, new_role or None,
181                                         new_tfn or None, new_annual_income, new_tax_withheld, has_health_insurance)
182
183         print(response)
184
185
186     def delete_user(tax_server):
187         print("\nDelete User:")
188         person_id = input("Enter the person ID of the user to delete: ").strip()
189
190
191
192
```

__init__.py

```
client1.py > main
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
```

Personal Income Tax System (PITS)

The image shows two screenshots of a code editor interface, likely PyCharm, displaying Python code for a Personal Income Tax System (PITS). The top screenshot shows the `client1.py` file, and the bottom screenshot shows the `__init__.py` file.

client1.py (Top Screenshot):

```
190     def delete_user(tax_server):
191         print("\nDelete User:")
192         person_id = input("Enter the person ID of the user to delete: ").strip()
193         confirm = input("Are you sure you want to delete this user? (yes/no): ").strip().lower()
194         if confirm == "yes":
195             response = tax_server.delete_user(person_id)
196             print(response)
197
198     def search_users(tax_server):
199         print("\nSearch Users:")
200         person_id = input("Enter person ID (leave blank for any): ").strip()
201         email = input("Enter email (leave blank for any): ").strip()
202         role = input("Enter role (user/admin, leave blank for any): ").strip().lower()
203
204         response = tax_server.search_users(person_id or None, email or None, role or None)
205         if isinstance(response, str):
206             print(response)
207         else:
208             for user in response:
209                 print(user)
210
211     def view_all_payroll_records(tax_server):
212         print("\nView All Payroll Records:")
213         person_id = input("Enter person ID to filter (leave blank to view all records): ").strip()
214
215         if person_id:
216             response = tax_server.view_payroll_records(person_id)
217         else:
218             response = tax_server.view_payroll_records()
219
220         if isinstance(response, str):
221             print(response)
222         else:
223             for record in response:
224                 print(f"Person ID: {record['person_id']}, Start: {record['pay_period_start']}, "
225                      f"End: {record['pay_period_end']}, Gross Income: {record['gross_income']}, Tax Paid: {record['tax_paid']}
```

__init__.py (Bottom Screenshot):

```
227     def insert_record(tax_server):
228         print("\nInsert Record:")
229         table = input("Enter the table name: ").strip()
230         fields = {}
231
232         # Assuming we need to gather fields and values
233         while True:
234             field = input("Enter field name (or 'done' to finish): ").strip()
235             if field.lower() == 'done':
236                 break
237             value = input("Enter value for '{field}': ").strip()
238             fields[field] = value
239
240         response = tax_server.insert_record(table, fields)
241         print(response)
242
243     def update_record(tax_server):
244         print("\nUpdate Record:")
245         table = input("Enter the table name: ").strip()
246         record_id = input("Enter the record ID to update: ").strip()
247         updates = {}
248
249         while True:
250             field = input("Enter field name to update (or 'done' to finish): ").strip()
251             if field.lower() == 'done':
252                 break
253             value = input("Enter new value for '{field}': ").strip()
254             updates[field] = value
255
256         response = tax_server.update_record(table, record_id, updates)
257         print(response)
258
259     def delete_record(tax_server):
260         print("\nDelete Record:")
261         table = input("Enter the table name: ").strip()
262         record_id = input("Enter the record ID to delete: ").strip()
263         response = tax_server.delete_record(table, record_id)
264         print(response)
```

Personal Income Tax System (PITS)

The screenshot shows two code editors side-by-side. Both editors have the same interface with tabs at the top, a sidebar on the left, and status bars at the bottom.

Top Editor (client1.py):

```
266     def display_database_schema(tax_server):
267         schema = tax_server.view_detailed_schema()
268         if schema:
269             print("\n--- Database Schema ---")
270             for table, details in schema.items():
271                 print(f"\nTable: {table}")
272                 for column in details:
273                     col_name = column[0]
274                     col_type = column[1]
275                     col_key = f"(Key: {column[3]})" if column[3] else "(Key: )"
276                     print(f"  {col_name} - {col_type} {col_key}")
277             else:
278                 print("Error fetching schema.")
279
280     def view_all_records(tax_server):
281         table_name = input("Enter the table name to view records: ").strip()
282         records = tax_server.view_all_records(table_name)
283
284         if isinstance(records, str):
285             print(records)
286         else:
287             print(f"\n--- Database Records ---\nTable: {table_name}")
288             if records:
289                 # Extract and print the column names
290                 columns = records[0].keys()
291                 col_headers = " | ".join(columns)
292                 header_line = "-" * len(col_headers)
293                 print(f"\n{col_headers}\n{header_line}")
294
295                 # Print each row of data
296                 for record in records:
297                     row_data = " | ".join(str(value) for value in record.values())
298                     print(f"\n{row_data}")
299             else:
300                 print("No records found.")
301
302     def admin_menu(tax_server):
303         def admin_menu(tax_server):
```

Bottom Editor (__init__.py):

```
303     def admin_menu(tax_server):
304         while True:
305             print("\nAdmin Menu:")
306             print("1. View all users")
307             print("2. Add a new user")
308             print("3. Update a user")
309             print("4. Delete a user")
310             print("5. View all tax history")
311             print("6. Search users")
312             print("7. Display payroll records of users")
313             print("8. Insert record")
314             print("9. Update record")
315             print("10. Delete record")
316             print("11. Export user data to CSV")
317             print("12. Export tax history to CSV")
318             print("13. View Database Schema")
319             print("14. View All Records in a Table")
320             print("15. Exit Admin Menu")
321
322             choice = input("Enter your choice: ").strip()
323             if choice == "1":
324                 users = tax_server.admin_view_users()
325                 for user in users:
326                     print(user)
327             elif choice == "2":
328                 add_user(tax_server)
329             elif choice == "3":
330                 update_user(tax_server)
331             elif choice == "4":
332                 delete_user(tax_server)
333             elif choice == "5":
334                 history = tax_server.view_all_tax_history()
335                 if isinstance(history, str):
336                     print(history)
337                 else:
338                     for record in history:
339                         print(f"Person ID: {record['person_id']}, Date: {record['calculation_date']}, Tax Amount: {record['tax_amount']}")
340             elif choice == "6":
```

Personal Income Tax System (PITS)

```
client1.py
303 def admin_menu(tax_server):
304     print("Person ID: {record['person_id']}, Date: {record['calculation_date']}, Tax Amount: {record['tax_amount']}")
305     elif choice == "6":
306         search_users(tax_server)
307     elif choice == "":
308         view_all_payroll_records(tax_server)
309     elif choice == "8":
310         insert_record(tax_server) # Call the newly defined function
311     elif choice == "9":
312         update_record(tax_server) # Call the newly defined function
313     elif choice == "10":
314         delete_record(tax_server) # Call the newly defined function
315     elif choice == "11":
316         file_path = input("Enter the file path for the CSV export (default: exported_users.csv): ").strip()
317         file_path = file_path or "exported_users.csv"
318         response = tax_server.export_users_to_csv(file_path)
319         print(response)
320     elif choice == "12":
321         file_path = input("Enter the file path for the CSV export (default: exported_tax_history.csv): ").strip()
322         file_path = file_path or "exported_tax_history.csv"
323         response = tax_server.export_tax_history_to_csv(file_path)
324         print(response)
325     elif choice == "13":
326         display_database_schema(tax_server)
327     elif choice == "14":
328         table_name = input("Enter the table name to view records: ").strip()
329         records = tax_server.view_all_records(table_name)
330         if isinstance(records, str):
331             print(records)
332         else:
333             for record in records:
334                 print(record)
335     elif choice == "15":
336         print("Exiting Admin Menu...")
337         break
338     else:
339         print("Invalid choice. Please try again.")

Ln 14, Col 54 Spaces: 4 UTF-8 CRLF () Python 3.11.5 64-bit
```

```
server1.py
303 def admin_menu(tax_server):
304     print("Person ID: {record['person_id']}, Date: {record['calculation_date']}, Tax Amount: {record['tax_amount']}")
305     elif choice == "6":
306         search_users(tax_server)
307     elif choice == "":
308         view_all_payroll_records(tax_server)
309     elif choice == "8":
310         insert_record(tax_server) # Call the newly defined function
311     elif choice == "9":
312         update_record(tax_server) # Call the newly defined function
313     elif choice == "10":
314         delete_record(tax_server) # Call the newly defined function
315     elif choice == "11":
316         file_path = input("Enter the file path for the CSV export (default: exported_users.csv): ").strip()
317         file_path = file_path or "exported_users.csv"
318         response = tax_server.export_users_to_csv(file_path)
319         print(response)
320     elif choice == "12":
321         file_path = input("Enter the file path for the CSV export (default: exported_tax_history.csv): ").strip()
322         file_path = file_path or "exported_tax_history.csv"
323         response = tax_server.export_tax_history_to_csv(file_path)
324         print(response)
325     elif choice == "13":
326         display_database_schema(tax_server)
327     elif choice == "14":
328         table_name = input("Enter the table name to view records: ").strip()
329         records = tax_server.view_all_records(table_name)
330         if isinstance(records, str):
331             print(records)
332         else:
333             for record in records:
334                 print(record)
335     elif choice == "15":
336         print("Exiting Admin Menu...")
337         break
338     else:
339         print("Invalid choice. Please try again.")

378 if __name__ == "__main__":
379     main()

Ln 14, Col 54 Spaces: 4 UTF-8 CRLF () Python 3.11.5 64-bit
```

server1.py

Personal Income Tax System (PITS)

The screenshot shows a code editor interface with the following details:

- EXPLORER View:** Shows the project structure with files like client1.py, server1.py, tax_calculator.py, tax_database.py, test_db_connection.py, user_registration_client.py, and validator.py.
- OPEN EDITORS View:** Shows the current open file is server1.py.
- Code Editor Content:** The server1.py file contains Python code:

```
import Pyro5.api
from tax_calculator import TaxCalculator

def main():
    daemon = Pyro5.api.Daemon()
    ns = Pyro5.api.locate_ns()
    uri = daemon.register(TaxCalculator)
    ns.register("Aus.taxcalculator", uri)
    print("Server is running with database integration...")
    daemon.requestLoop()

if __name__ == "__main__":
    main()
```
- Bottom Status Bar:** Displays information such as Line 1, Column 1, Spaces: 4, UTF-8, CRLF, Python 3.11.5 64-bit, and a file icon.

tax_calculator.py

Personal Income Tax System (PITS)

```

tax_calculator.py > TaxCalculator
1  import Pyro5.api
2  import csv
3  import os
4  from tax_database import TaxDatabaseServer
5  from validator import Validator
6  from mysql.connector import Error
7
8
9  @Pyro5.api.expose
10 class TaxCalculator:
11     def __init__(self):
12         self.db = TaxDatabaseServer()
13         self.validator = Validator() # Make sure the validator is properly initialized
14
15     def view_detailed_schema(self):
16         """Expose the detailed schema view through the TaxCalculator class."""
17         # Call the method from the db (TaxDatabaseServer instance)
18         if not self.db.connection:
19             return "Database connection failed. Please try again later."
20         return self.db.view_detailed_schema()
21
22     def view_all_records(self, table_name):
23         """Expose the method to view all records in a given table."""
24         return self.db.view_all_records(table_name)
25
26     def authenticate_user(self, person_id, password):
27         if not self.db.connection:
28             return "Database connection failed. Please try again later."
29         cursor = self.db.connection.cursor(dictionary=True)
30         query = "SELECT * FROM users WHERE person_id = %s AND password = %s"
31         cursor.execute(query, (person_id, password))
32         result = cursor.fetchone()
33         cursor.close()
34
35         if result:
36             return result
37         else:
38             return None

```

Ln 14, Col 1 Spaces: 4 UTF-8 CRLF () Python 3.11.5 64-bit


```

tax_calculator.py > TaxCalculator
10
11     class TaxCalculator:
12
13         def register_user(self, person_id, password, name=None, email=None, tfn=None):
14             if not self.db.connection:
15                 return "Database connection failed. Please try again later."
16
17             # Check if the user already exists in either table
18             if self.db.get_user_by_person_id(person_id) or self.db.get_tfn_free_user_by_person_id(person_id):
19                 return f"Error: User with person_id '{person_id}' already exists."
20
21             try:
22                 if tfn:
23                     # Insert into the main users table
24                     self.db.insert_record('users', {
25                         'person_id': person_id,
26                         'password': password,
27                         'name': name,
28                         'email': email,
29                         'tfn': tfn,
30                         'role': 'user'
31                     })
32                 else:
33                     # Insert into the TFN-free users table
34                     self.db.insert_record('tfn_free_users', {
35                         'person_id': person_id,
36                         'password': password,
37                         'name': name,
38                         'email': email
39                     })
40             except Exception as e:
41                 return f"An error occurred during registration: {e}"
42
43             return f"User '{person_id}' registered successfully."
44
45         def get_tfn_free_user_by_person_id(self, person_id):
46             self.ensure_connection()
47             try:
48                 cursor = self.connection.cursor(dictionary=True)
49                 query = "SELECT * FROM tfn_free_users WHERE person_id = %s"
50                 cursor.execute(query, (person_id,))
51                 result = cursor.fetchone()
52                 cursor.close()
53                 return result
54             except Exception as e:
55                 return f"An error occurred during retrieval: {e}"
56
57     def ensure_connection(self):
58         if not self.db.connection:
59             self.db.connect()
60
61     def disconnect(self):
62         self.db.disconnect()
63
64     def __del__(self):
65         self.disconnect()

```

Ln 14, Col 1 Spaces: 4 UTF-8 CRLF () Python 3.11.5 64-bit

Personal Income Tax System (PITS)

```
tax_calculator.py > TaxCalculator
10  class TaxCalculator:
11      def get_tfn_free_user_by_person_id(self, person_id):
12          cursor = self.connection.cursor(dictionary=True)
13          query = "SELECT * FROM tfn_free_users WHERE person_id = %s"
14          cursor.execute(query, (person_id,))
15          user = cursor.fetchone()
16          cursor.close()
17          return user
18      except Error as e:
19          print(f"Error fetching TFN-free user by person ID: {e}")
20          return None
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
```

```
__init__.py
tax_calculator.py
tax_database.py
test_db_connection.py
user_registration_client.py
```

Personal Income Tax System (PITS)

```
tax_calculator.py > TaxCalculator
10  class TaxCalculator:
11      def calculate_tax(self, person_id):
12          # Convert the database values to float
13          annual_income = float(user['annual_income'])
14          tax_withheld = float(user['tax_withheld'])
15          has_insurance = user['has_health_insurance']
16
17          medicare Levy = self.calculate_medicare Levy(annual_income)
18          mls = self.calculate_medicare Levy_surcharge(annual_income, has_insurance)
19          total_tax = annual_income * 0.325 + medicare Levy + mls - tax_withheld
20
21          if total_tax > 0:
22              self.log_tax_calculation(person_id, annual_income, tax_withheld, total_tax, False)
23              return f"You owe additional tax: ${total_tax:.2f}"
24          else:
25              self.log_tax_calculation(person_id, annual_income, tax_withheld, -total_tax, True)
26              return f"You will receive a refund: ${-total_tax:.2f}"
27
28      def calculate_tax_without_tfn(self, person_id, biweekly_wages, tax_withheld, annual_income, has_insurance):
29          try:
30              # Calculate total net wages based on biweekly wages (assuming 26 pay periods in a year)
31              total_net_wages = biweekly_wages * 26
32
33              # Validate inputs
34              self.validator.validate_wage(total_net_wages)
35              self.validator.validate_wage(tax_withheld)
36              self.validator.validate_wage(annual_income)
37
38              medicare Levy = self.calculate_medicare Levy(annual_income)
39              mls = self.calculate_medicare Levy_surcharge(annual_income, has_insurance)
40
41              # Calculate total tax due based on the inputs
42              total_tax_due = (total_net_wages * 0.325) + medicare Levy + mls - tax_withheld
43
44              if total_tax_due > 0:
45                  self.log_tax_calculation(person_id, annual_income, tax_withheld, total_tax_due, False)
46                  return f"You owe additional tax: ${total_tax_due:.2f}"
47              else:
48                  self.log_tax_calculation(person_id, annual_income, tax_withheld, -total_tax_due, True)
49                  return f"You will receive a refund: ${-total_tax_due:.2f}"
50
51      def log_tax_calculation(self, person_id, annual_income, tax_withheld, tax_amount, is_refund):
52          if not self.db.connection:
53              return "Database connection failed. Please try again later."
54          try:
55              cursor = self.db.connection.cursor()
56              query = """
57                  INSERT INTO tax_history (person_id, calculation_date, annual_income, tax_withheld, tax_amount, is_refund)
58                  VALUES (%s, NOW(), %s, %s, %s, %s)
59              """
60
61              cursor.execute(query, (person_id, annual_income, tax_withheld, tax_amount, is_refund))
62              self.db.connection.commit()
63              cursor.close()
64          except Exception as e:
65              print(f"Error logging tax calculation: {e}")
66
67      def view_tax_history(self, person_id):
68          if not self.db.connection:
69              return "Database connection failed. Please try again later."
70          try:
71              cursor = self.db.connection.cursor(dictionary=True)
72              query = "SELECT * FROM tax_history WHERE person_id = %s ORDER BY calculation_date DESC"
73              cursor.execute(query, (person_id,))
74              history = cursor.fetchall()
75              cursor.close()
76          if not history:
77              return "No tax history found for this user."
78
79
__init__.py
1  from .tax_calculator import TaxCalculator
```

Personal Income Tax System (PITS)

```
tax_calculator.py > TaxCalculator
10  class TaxCalculator:
199      def view_tax_history(self, person_id):
200          if person_id == None:
201              return "No tax history found for this user."
202          return history
203      except Exception as e:
204          return f"Error retrieving tax history: {e}"
205
206      def view_payroll_records(self, person_id=None):
207          if not self.db.connection:
208              return "Database connection failed. Please try again later."
209          try:
210              cursor = self.db.connection.cursor(dictionary=True)
211              if person_id:
212                  query = "SELECT * FROM payroll_records WHERE person_id = %s ORDER BY pay_period_start DESC"
213                  cursor.execute(query, (person_id,))
214              else:
215                  query = "SELECT * FROM payroll_records ORDER BY pay_period_start DESC"
216                  cursor.execute(query)
217
218              records = cursor.fetchall()
219              cursor.close()
220              if not records:
221                  return "No payroll records found."
222              return records
223          except Exception as e:
224              return f"Error retrieving payroll records: {e}"
225
226
227      def update_profile(self, person_id, new_email=None, new_password=None):
228          if not self.db.connection:
229              return "Database connection failed. Please try again later."
230          try:
231              if new_email is not None:
232                  cursor = self.db.connection.cursor()
233                  query = "UPDATE users SET email = %s WHERE person_id = %s"
234                  cursor.execute(query, (new_email, person_id))
235                  self.db.connection.commit()
236                  cursor.close()
237
238              return "Profile updated successfully."
239          except Exception as e:
240              return f"Error updating profile: {e}"
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
```

```
tax_calculator.py > TaxCalculator
10  class TaxCalculator:
235      def update_profile(self, person_id, new_email=None, new_password=None):
244
245          if new_password is not None:
246              cursor = self.db.connection.cursor()
247              query = "UPDATE users SET password = %s WHERE person_id = %s"
248              cursor.execute(query, (new_password, person_id))
249              self.db.connection.commit()
250              cursor.close()
251
252
253              return "Profile updated successfully."
254          except Exception as e:
255              return f"Error updating profile: {e}"
256
257      def insert_record(self, table, fields):
258          if not self.db.connection:
259              return "Database connection failed. Please try again later."
260          try:
261              cursor = self.db.connection.cursor()
262              placeholders = ', '.join(['%s'] * len(fields))
263              query = f'INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})'
264              cursor.execute(query, list(fields.values()))
265              self.db.connection.commit()
266              cursor.close()
267
268              return "Record inserted successfully."
269          except Exception as e:
270              return f"Error inserting record: {e}"
271
272      def update_record(self, table, record_id, updates):
273          if not self.db.connection:
274              return "Database connection failed. Please try again later."
275          try:
276              cursor = self.db.connection.cursor()
277              set_clause = ', '.join([f'{key} = %s' for key in updates.keys()])
278              query = f'UPDATE {table} SET {set_clause} WHERE id = %s'
279              cursor.execute(query, list(updates.values()) + [record_id])
280              self.db.connection.commit()
281              cursor.close()
```

Personal Income Tax System (PITS)

```
tax_calculator.py > TaxCalculator
10  class TaxCalculator:
271      def update_record(self, table, record_id, updates):
281          return "Record updated successfully."
282      except Exception as e:
283          return f"Error updating record: {e}"
284
285      def delete_record(self, table, record_id):
286          if not self.db.connection:
287              return "Database connection failed. Please try again later."
288          try:
289              cursor = self.db.connection.cursor()
290              query = f"DELETE FROM {table} WHERE id = %s"
291              cursor.execute(query, (record_id,))
292              self.db.connection.commit()
293              cursor.close()
294              return "Record deleted successfully."
295          except Exception as e:
296              return f"Error deleting record: {e}"
297
298
299
300      def calculate_tax_with_pay_period(self, person_id):
301          if not self.db.connection:
302              return "Database connection failed. Please try again later."
303          try:
304              cursor = self.db.connection.cursor(dictionary=True)
305              query = """
306                  SELECT * FROM payroll_records WHERE person_id = %s
307                  ORDER BY pay_period_start DESC
308              """
309
310              cursor.execute(query, (person_id,))
311              records = cursor.fetchall()
312              cursor.close()
313
314              if not records:
315                  return "No payroll records found for this user."
316
317              total_gross_income = sum(record['gross_income'] for record in records)
318
319              medicare Levy = self.calculate_medicare_levy(total_gross_income)
320              mls = self.calculate_medicare Levy.surcharge(total_gross_income, self.has_health_insurance(person_id))
321              total_tax_due = (total_gross_income * 0.325) + medicare Levy + mls - total_tax_paid
322
323              if total_tax_due > 0:
324                  self.log_tax_calculation(person_id, total_gross_income, total_tax_paid, total_tax_due, False)
325                  return f"Total tax due based on payroll records: ${total_tax_due:.2f}"
326              else:
327                  self.log_tax_calculation(person_id, total_gross_income, total_tax_paid, -total_tax_due, True)
328                  return f"You will receive a refund based on payroll records: ${-total_tax_due:.2f}"
329
330          except Exception as e:
331              return f"Error calculating tax: {e}"
332
333      def has_health_insurance(self, person_id):
334          if not self.db.connection:
335              return False
336          try:
337              cursor = self.db.connection.cursor(dictionary=True)
338              query = "SELECT has_health_insurance FROM users WHERE person_id = %s"
339              cursor.execute(query, (person_id,))
340              result = cursor.fetchone()
341              cursor.close()
342
343              if result and result['has_health_insurance']:
344                  return True
345              return False
346
347          except Exception as e:
348              print(f"Error checking health insurance status: {e}")
349              return False
350
351
352      def add_user(self, person_id, name, email, password, role='user', tfn=None, annual_income=None, tax_withheld=None, has_health_insurance=False):
353          if not self.db.connection:
354              return
355
356
357
358      def __init__(self):
359          self.db = DatabaseConnection()
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
```

```
Ln 14, Col 1  Spaces: 4  UTF-8  CRLF  () Python  3.11.5 64-bit
```

```
tax_calculator.py > TaxCalculator
10  class TaxCalculator:
300      def calculate_tax_with_pay_period(self, person_id):
316          total_gross_income = sum(record['gross_income'] for record in records)
317          total_tax_paid = sum(record['tax_paid'] for record in records)
318
319          medicare Levy = self.calculate_medicare_levy(total_gross_income)
320          mls = self.calculate_medicare Levy.surcharge(total_gross_income, self.has_health_insurance(person_id))
321          total_tax_due = (total_gross_income * 0.325) + medicare Levy + mls - total_tax_paid
322
323          if total_tax_due > 0:
324              self.log_tax_calculation(person_id, total_gross_income, total_tax_paid, total_tax_due, False)
325              return f"Total tax due based on payroll records: ${total_tax_due:.2f}"
326          else:
327              self.log_tax_calculation(person_id, total_gross_income, total_tax_paid, -total_tax_due, True)
328              return f"You will receive a refund based on payroll records: ${-total_tax_due:.2f}"
329
330      except Exception as e:
331          return f"Error calculating tax: {e}"
332
333      def has_health_insurance(self, person_id):
334          if not self.db.connection:
335              return False
336          try:
337              cursor = self.db.connection.cursor(dictionary=True)
338              query = "SELECT has_health_insurance FROM users WHERE person_id = %s"
339              cursor.execute(query, (person_id,))
340              result = cursor.fetchone()
341              cursor.close()
342
343              if result and result['has_health_insurance']:
344                  return True
345              return False
346
347          except Exception as e:
348              print(f"Error checking health insurance status: {e}")
349              return False
350
351
352      def add_user(self, person_id, name, email, password, role='user', tfn=None, annual_income=None, tax_withheld=None, has_health_insurance=False):
353          if not self.db.connection:
354              return
355
356
357
358      def __init__(self):
359          self.db = DatabaseConnection()
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
409
```

```
Ln 14, Col 1  Spaces: 4  UTF-8  CRLF  () Python  3.11.5 64-bit
```

Personal Income Tax System (PITS)

The image shows a code editor with two tabs open, both displaying Python code for a `TaxCalculator` class.

Top Tab: `tax_calculator.py`

```
10  class TaxCalculator:
349
350      def add_user(self, person_id, name, email, password, role='user', tfn=None, annual_income=None, tax_withheld=None, has_health_insurance=False):
351          if not self.db.connection:
352              return "Database connection failed. Please try again later."
353          try:
354              cursor = self.db.connection.cursor()
355              query = """
356                  INSERT INTO users (person_id, name, email, password, role, tfn, annual_income, tax_withheld, has_health_insurance)
357                  VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
358              """
359              cursor.execute(query, (person_id, name, email, password, role, tfn, annual_income, tax_withheld, has_health_insurance))
360              self.db.connection.commit()
361              cursor.close()
362              return "User added successfully."
363          except Exception as e:
364              return f"Error adding user: {e}"
365
366
367      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=None):
368          if not self.db.connection:
369              return "Database connection failed. Please try again later."
370          try:
371              # Update fields if new values are provided
372              if new_email:
373                  cursor = self.db.connection.cursor()
374                  query = "UPDATE users SET email = %s WHERE person_id = %s"
375                  cursor.execute(query, (new_email, person_id))
376                  self.db.connection.commit()
377                  cursor.close()
378
379              if new_password:
380                  cursor = self.db.connection.cursor()
381                  query = "UPDATE users SET password = %s WHERE person_id = %s"
382                  cursor.execute(query, (new_password, person_id))
383                  self.db.connection.commit()
384                  cursor.close()
385
386              if new_role:
387                  cursor = self.db.connection.cursor()
388                  query = "UPDATE users SET role = %s WHERE person_id = %s"
389                  cursor.execute(query, (new_role, person_id))
390                  self.db.connection.commit()
391                  cursor.close()
392
393              if new_tfn:
394                  cursor = self.db.connection.cursor()
395                  query = "UPDATE users SET tfn = %s WHERE person_id = %s"
396                  cursor.execute(query, (new_tfn, person_id))
397                  self.db.connection.commit()
398                  cursor.close()
399
400              if new_annual_income:
401                  cursor = self.db.connection.cursor()
402                  query = "UPDATE users SET annual_income = %s WHERE person_id = %s"
403                  cursor.execute(query, (new_annual_income, person_id))
```

Bottom Tab: `tax_calculator.py`

```
10  class TaxCalculator:
367      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=None):
368          if not self.db.connection:
369              return "Database connection failed. Please try again later."
370          try:
371              # Update fields if new values are provided
372              if new_email:
373                  cursor = self.db.connection.cursor()
374                  query = "UPDATE users SET email = %s WHERE person_id = %s"
375                  cursor.execute(query, (new_email, person_id))
376                  self.db.connection.commit()
377                  cursor.close()
378
379              if new_password:
380                  cursor = self.db.connection.cursor()
381                  query = "UPDATE users SET password = %s WHERE person_id = %s"
382                  cursor.execute(query, (new_password, person_id))
383                  self.db.connection.commit()
384                  cursor.close()
385
386              if new_role:
387                  cursor = self.db.connection.cursor()
388                  query = "UPDATE users SET role = %s WHERE person_id = %s"
389                  cursor.execute(query, (new_role, person_id))
390                  self.db.connection.commit()
391                  cursor.close()
392
393              if new_tfn:
394                  cursor = self.db.connection.cursor()
395                  query = "UPDATE users SET tfn = %s WHERE person_id = %s"
396                  cursor.execute(query, (new_tfn, person_id))
397                  self.db.connection.commit()
398                  cursor.close()
399
400              if new_annual_income:
401                  cursor = self.db.connection.cursor()
402                  query = "UPDATE users SET annual_income = %s WHERE person_id = %s"
403                  cursor.execute(query, (new_annual_income, person_id))
```

Personal Income Tax System (PITS)

```
tax_calculator.py
10  class TaxCalculator:
367      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=None):
400          if new_annual_income:
401              cursor = self.db.connection.cursor()
402              query = "UPDATE users SET annual_income = %s WHERE person_id = %s"
403              cursor.execute(query, (new_annual_income, person_id))
404              self.db.connection.commit()
405              cursor.close()
406
407          if new_tax_withheld:
408              cursor = self.db.connection.cursor()
409              query = "UPDATE users SET tax_withheld = %s WHERE person_id = %s"
410              cursor.execute(query, (new_tax_withheld, person_id))
411              self.db.connection.commit()
412              cursor.close()
413
414          if has_health_insurance is not None:
415              cursor = self.db.connection.cursor()
416              query = "UPDATE users SET has_health_insurance = %s WHERE person_id = %s"
417              cursor.execute(query, (has_health_insurance, person_id))
418              self.db.connection.commit()
419              cursor.close()
420
421          return "User updated successfully."
422      except Exception as e:
423          return f"Error updating user: {e}"
424
425
426      def delete_user(self, person_id):
427          if not self.db.connection:
428              return "Database connection failed. Please try again later."
429          try:
430              cursor = self.db.connection.cursor()
431              query = "DELETE FROM users WHERE person_id = %s"
432              cursor.execute(query, (person_id,))
433              self.db.connection.commit()
434              cursor.close()
435          return "User deleted successfully."
436
437      except Exception as e:
438          return f"Error deleting user: {e}"
439
440      def view_all_tax_history(self):
441          if not self.db.connection:
442              return "Database connection failed. Please try again later."
443          try:
444              cursor = self.db.connection.cursor(dictionary=True)
445              query = "SELECT * FROM tax_history ORDER BY calculation_date DESC"
446              cursor.execute(query)
447              history = cursor.fetchall()
448              cursor.close()
449              if not history:
450                  return "No tax history found."
451              return history
452          except Exception as e:
453              return f"Error retrieving tax history: {e}"
454
455      def search_users(self, person_id=None, email=None, role=None):
456          if not self.db.connection:
457              return "Database connection failed. Please try again later."
458          try:
459              cursor = self.db.connection.cursor(dictionary=True)
460              query = "SELECT * FROM users WHERE TRUE"
461              params = []
462
463              if person_id:
464                  params.append(f"person_id = {person_id}")
465
466              if email:
467                  params.append(f"email = '{email}'")
468
469              if role:
470                  params.append(f"role = '{role}'")
471
472              query += " AND ".join(params)
473
474              cursor.execute(query)
475              users = cursor.fetchall()
476              cursor.close()
477
478              if not users:
479                  return "No users found."
480              return users
481
482      except Exception as e:
483          return f"Error searching users: {e}"
```

```
__init__.py
10  from .tax_calculator import TaxCalculator
```

Personal Income Tax System (PITS)

The image shows a code editor interface with two tabs open: `tax_calculator.py` and `__init__.py`. The `tax_calculator.py` tab is active, displaying Python code for a `TaxCalculator` class. The `__init__.py` tab is visible at the bottom.

```
10  class TaxCalculator:
454
455     def search_users(self, person_id=None, email=None, role=None):
456         if not self.db.connection:
457             return "Database connection failed. Please try again later."
458         try:
459             cursor = self.db.connection.cursor(dictionary=True)
460             query = "SELECT * FROM users WHERE TRUE"
461             params = []
462
463             if person_id:
464                 query += " AND person_id = %s"
465                 params.append(person_id)
466
467             if email:
468                 query += " AND email = %s"
469                 params.append(email)
470
471             if role:
472                 query += " AND role = %s"
473                 params.append(role)
474
475             cursor.execute(query, params)
476             users = cursor.fetchall()
477             cursor.close()
478
479             if not users:
480                 return "No users found matching the criteria."
481             return users
482         except Exception as e:
483             return f"Error searching users: {e}"
484
485     def export_users_to_csv(self, file_path="exported_users.csv"):
486         if not self.db.connection:
487             return "Database connection failed. Please try again later."
488         try:
489             # If the file path is a directory, set a default filename
490             if os.path.isdir(file_path):
491                 file_path = os.path.join(file_path, "exported_users.csv")
492
493             cursor = self.db.connection.cursor(dictionary=True)
494             query = "SELECT * FROM users"
495             cursor.execute(query)
496             users = cursor.fetchall()
497             cursor.close()
498
499             if not users:
500                 return "No user data found."
501
502             # Export data to CSV
503             with open(file_path, mode='w', newline='') as file:
504                 writer = csv.DictWriter(file, fieldnames=users[0].keys())
505                 writer.writeheader()
506                 writer.writerows(users)
507
508             return f"User data exported successfully to {file_path}."
509         except Exception as e:
510             return f"Error exporting user data: {e}"
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()
```

The `tax_calculator.py` file contains methods for searching users based on person ID, email, and role, and for exporting user data to a CSV file. The `__init__.py` file is partially visible at the bottom.

Personal Income Tax System (PITS)

```
tax_calculator.py
10 class TaxCalculator:
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()
521
522             if not history:
523                 return "No tax history data found."
524
525             # Export data to CSV
526             with open(file_path, mode='w', newline='') as file:
527                 writer = csv.DictWriter(file, fieldnames=history[0].keys())
528                 writer.writeheader()
529                 writer.writerows(history)
530
531             return f"Tax history data exported successfully to {file_path}."
532         except Exception as e:
533             return f"Error exporting tax history data: {e}"
534
535     def admin_view_users(self):
536         users = self.db.view_all_users()
537         return users
538
539     def admin_view_schema(self, table_name):
540         schema = self.db.view_schema(table_name)
541         return schema
542
543     def calculate_medicare Levy(self, annual_income):
544         return annual_income * 0.02
545
546     def calculate_medicare Levy surcharge(self, annual_income, has_insurance):
547         if not has_insurance and annual_income > 90000:
548             if annual_income <= 105000:
549                 return annual_income * 0.01
550             elif annual_income <= 140000:
551                 return annual_income * 0.0125
552             else:
553                 return annual_income * 0.015
554
555     return 0
```

```
tax_database.py
10 class TaxDatabase:
535     def __init__(self, db_type="sqlite", db_name="tax_db"):
536         self.db_type = db_type
537         self.db_name = db_name
538
539     def connect(self):
540         if self.db_type == "sqlite":
541             self.connection = sqlite3.connect(self.db_name)
542             self.cursor = self.connection.cursor()
543
544     def disconnect(self):
545         if self.connection:
546             self.connection.close()
547
548     def view_all_users(self):
549         query = "SELECT * FROM users"
550         self.cursor.execute(query)
551         users = self.cursor.fetchall()
552         return users
553
554     def view_schema(self, table_name):
555         query = f"PRAGMA table_info({table_name});"
556         self.cursor.execute(query)
557         schema = self.cursor.fetchall()
558         return schema
559
560     def insert_user(self, user_data):
561         columns = ", ".join(user_data.keys())
562         values = ", ".join(["?"] * len(user_data))
563         query = f"INSERT INTO users ({columns}) VALUES ({values});"
564         self.cursor.execute(query, tuple(user_data.values()))
565         self.connection.commit()
566
567     def update_user(self, user_id, updated_data):
568         set_clause = ", ".join([f"{key} = ?" for key in updated_data])
569         query = f"UPDATE users SET {set_clause} WHERE id = ?"
570         self.cursor.execute(query, (*tuple(updated_data.values()), user_id))
571         self.connection.commit()
572
573     def delete_user(self, user_id):
574         query = f"DELETE FROM users WHERE id = ?"
575         self.cursor.execute(query, (user_id,))
576         self.connection.commit()
```

tax_database.py

Personal Income Tax System (PITS)

```
tax_database.py > TaxDatabaseServer
1 import mysql.connector
2 from mysql.connector import Error
3
4 class TaxDatabaseServer:
5     def __init__(self):
6         self.connection = self.create_connection()
7
8     def create_connection(self):
9         try:
10             connection = mysql.connector.connect(
11                 host="localhost",
12                 user="root",
13                 password="mysql",
14                 database="tax_system"
15             )
16             if connection.is_connected():
17                 print("Connected to MySQL Database")
18             return connection
19         except Error as e:
20             print(f"Error connecting to the database: {e}")
21             return None
22
23     def ensure_connection(self):
24         if not self.connection or not self.connection.is_connected():
25             self.connection = self.create_connection()
26
27     def user_exists(self, person_id):
28         self.ensure_connection()
29         try:
30             cursor = self.connection.cursor(dictionary=True)
31             query = "SELECT 1 FROM users WHERE person_id = %s"
32             cursor.execute(query, (person_id,))
33             user = cursor.fetchone()
34             cursor.close()
35             return user is not None
36         except Error as e:
37             print(f"Error checking if user exists: {e}")
38             return False
```

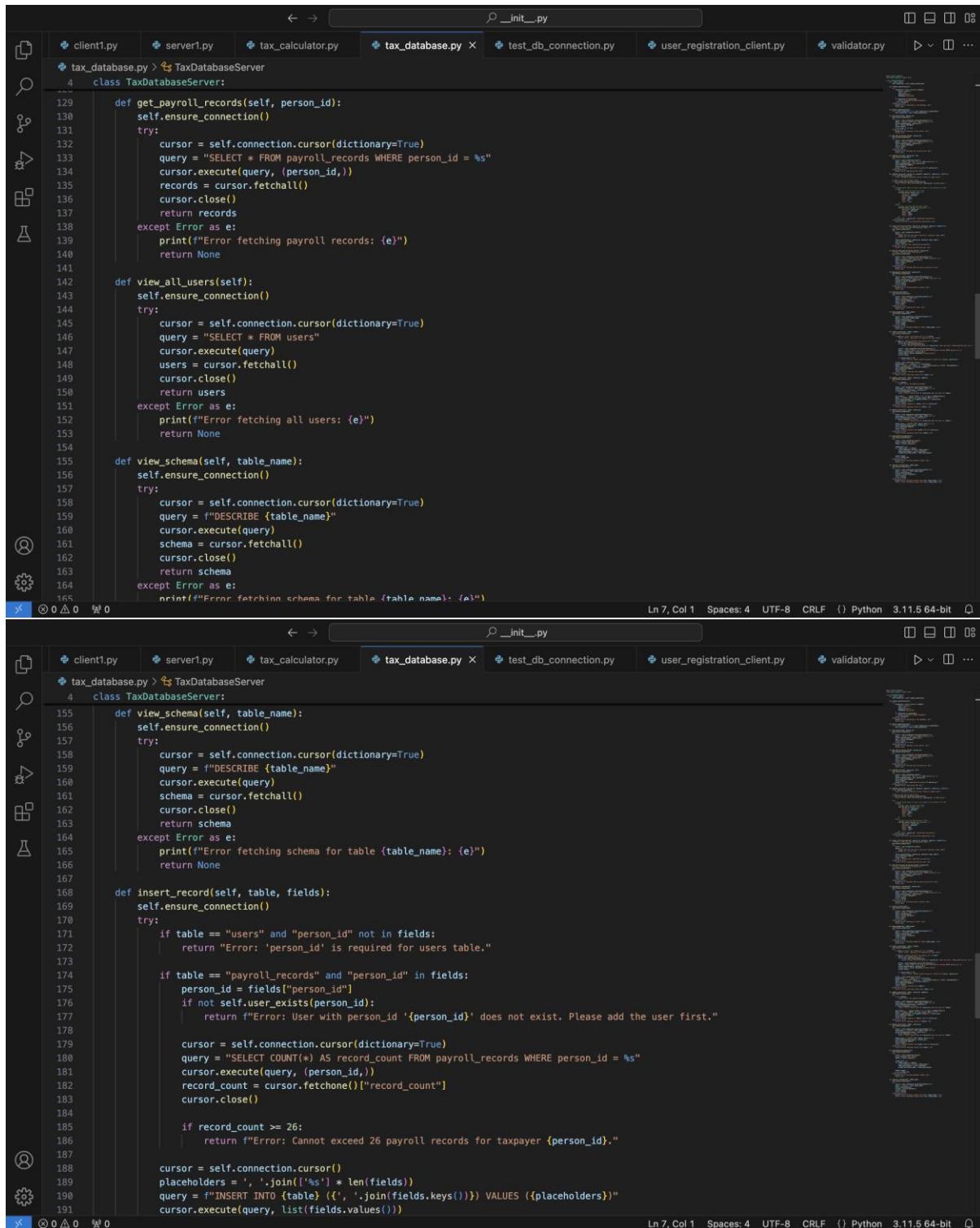


```
tax_database.py > TaxDatabaseServer
4 class TaxDatabaseServer:
39
40     def get_user_by_person_id(self, person_id):
41         self.ensure_connection()
42         try:
43             cursor = self.connection.cursor(dictionary=True)
44             query = "SELECT * FROM users WHERE person_id = %s"
45             cursor.execute(query, (person_id,))
46             user = cursor.fetchone()
47             cursor.close()
48             return user
49         except Error as e:
50             print(f"Error fetching user by person ID: {e}")
51             return None
52
53     def register_tfn(self, person_id, tfn):
54         self.ensure_connection()
55         try:
56             cursor = self.connection.cursor()
57             query = "UPDATE users SET tfn = %s WHERE person_id = %s"
58             cursor.execute(query, (tfn, person_id))
59             self.connection.commit()
60             cursor.close()
61             print(f"TFN {tfn} registered for person ID {person_id}")
62         except Error as e:
63             print(f"Error registering TFN: {e}")
64
65     def register_user(self, person_id, password, name=None, email=None, tfn=None):
66         if not self.db.connection:
67             return "Database connection failed. Please try again later."
68
69         # Check if the user already exists
70         if self.db.get_user_by_person_id(person_id):
71             return f"Error: User with person_id '{person_id}' already exists."
72
73         try:
74             # Decide which table to insert into based on the presence of a TFN
75             if tfn:
76                 # Insert into the main users table
```

Personal Income Tax System (PITS)

```
tax_database.py
4  class TaxDatabaseServer:
55      def register_user(self, person_id, password, name=None, email=None, tfn=None):
56          if not self.db.connection:
57              return "Database connection failed. Please try again later."
58
59          # Check if the user already exists
60          if self.db.get_user_by_person_id(person_id):
61              return f"Error: User with person_id '{person_id}' already exists."
62
63          try:
64              # Decide which table to insert into based on the presence of a TFN
65              if tfn:
66                  # Insert into the main users table
67                  self.db.insert_record('users', {
68                      'person_id': person_id,
69                      'password': password,
70                      'name': name,
71                      'email': email,
72                      'tfn': tfn,
73                      'role': 'user'
74                  })
75              else:
76                  # Insert into the TFN-free users table
77                  self.db.insert_record('tfn_free_users', {
78                      'person_id': person_id,
79                      'password': password,
80                      'name': name,
81                      'email': email,
82                      'role': 'user'
83                  })
84
85          return f"User '{person_id}' registered successfully."
86      except Exception as e:
87          return f"An error occurred during registration: {e}"
88
89
90      def insert_tfn_free_user(self, person_id, password, name=None, email=None):
91          """Insert a user into the TFN-free_users table."""
92          self.ensure_connection()
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
__init__.py
```

Personal Income Tax System (PITS)



```
client1.py server1.py tax_calculator.py tax_database.py test_db_connection.py user_registration_client.py validator.py

# tax_database.py

class TaxDatabaseServer:
    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:
            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()))
        except Error as e:
            print(f"Error inserting record: {e}")

    def user_exists(self, person_id):
        cursor = self.connection.cursor()
        query = "SELECT COUNT(*) AS count FROM users WHERE person_id = %s"
        cursor.execute(query, (person_id,))
        count = cursor.fetchone()["count"]
        cursor.close()
        return count > 0
```

Personal Income Tax System (PITS)

```
client1.py server1.py tax_calculator.py tax_database.py test_db_connection.py user_registration_client.py validator.py
```

```
tax_database.py > TaxDatabaseServer
```

```
168     def insert_record(self, table, fields):
169         self.ensure_connection()
170         try:
171             if table == "users" and "person_id" not in fields:
172                 return "Error: 'person_id' is required for users table."
173
174             if table == "payroll_records" and "person_id" in fields:
175                 person_id = fields["person_id"]
176                 if not self.user_exists(person_id):
177                     return f"Error: User with person_id '{person_id}' does not exist. Please add the user first."
178
179             cursor = self.connection.cursor(dictionary=True)
180             query = f"SELECT COUNT(*) AS record_count FROM payroll_records WHERE person_id = %s"
181             cursor.execute(query, (person_id,))
182             record_count = cursor.fetchone()["record_count"]
183             cursor.close()
184
185             if record_count >= 26:
186                 return f"Error: Cannot exceed 26 payroll records for taxpayer {person_id}."
187
188             cursor = self.connection.cursor()
189             placeholders = ', '.join(['%s'] * len(fields))
190             query = f"INSERT INTO {table} ({', '.join(fields.keys())}) VALUES ({placeholders})"
191             cursor.execute(query, list(fields.values()))
192             self.connection.commit()
193             cursor.close()
194             return f"Record inserted into {table}."
195         except Error as e:
196             return f"Error inserting record into {table}: {e}"
197
198     def update_record(self, table, record_id, updates):
199         self.ensure_connection()
200         try:
201             if not updates:
202                 return "Error: No updates provided."
203
204             cursor = self.connection.cursor(dictionary=True)
205             check_query = f"SELECT 1 FROM {table} WHERE id = %s"
206             cursor.execute(check_query, (record_id,))
207             if cursor.fetchone() is None:
208                 return f"Error: Record with ID {record_id} does not exist in {table}."
209
210             set_clause = ', '.join([f"{key} = %s" for key in updates.keys()])
211             query = f"UPDATE {table} SET {set_clause} WHERE id = %s"
212             cursor.execute(query, list(updates.values()) + [record_id])
213             self.connection.commit()
214             cursor.close()
215             return f"Record updated in {table} with ID {record_id}."
216         except Error as e:
217             return f"Error updating record in {table}: {e}"
218
219     def delete_record(self, table, record_id):
220         self.ensure_connection()
221         try:
222             cursor = self.connection.cursor(dictionary=True)
223             check_query = f"SELECT 1 FROM {table} WHERE id = %s"
224             cursor.execute(check_query, (record_id,))
225             if cursor.fetchone() is None:
226                 return f"Error: Record with ID {record_id} does not exist in {table}."
227
228             delete_query = f"DELETE FROM {table} WHERE id = %s"
229             cursor.execute(delete_query, (record_id,))
230             self.connection.commit()
231             cursor.close()
232             return f"Record deleted from {table} with ID {record_id}."
233         except Error as e:
234             return f"Error deleting record from {table}: {e}"
```

```
Ln 7, Col 1 Spaces: 4 UTF-8 CRLF () Python 3.11.5 64-bit
```

Personal Income Tax System (PITS)

The image shows two side-by-side code editors. Both editors have tabs for client.py, server1.py, tax_calculator.py, test_db_connection.py, user_registration_client.py, and validator.py. The top editor's title bar says '_init_.py'. The bottom editor's title bar also says '_init_.py'. The left sidebar of each editor has icons for file operations like open, save, and close, as well as search and refresh.

tax_database.py (Top Editor):

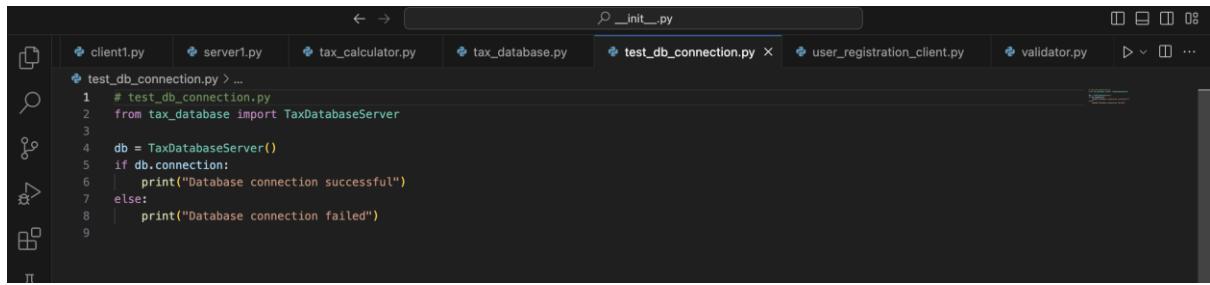
```
219     def delete_record(self, table, record_id):
220         self.ensure_connection()
221         try:
222             cursor = self.connection.cursor(dictionary=True)
223             check_query = f"SELECT 1 FROM {table} WHERE id = %s"
224             cursor.execute(check_query, (record_id,))
225             if cursor.fetchone() is None:
226                 return f"Error: Record with ID {record_id} does not exist in {table}."
227
228             delete_query = f"DELETE FROM {table} WHERE id = %s"
229             cursor.execute(delete_query, (record_id,))
230             self.connection.commit()
231             cursor.close()
232             return f"Record deleted from {table} with ID {record_id}."
233         except Error as e:
234             return f"Error deleting record from {table}: {e}"
235
236     def view_detailed_schema(self):
237         self.ensure_connection()
238         try:
239             cursor = self.connection.cursor()
240             cursor.execute("SHOW TABLES")
241             tables = cursor.fetchall()
242
243             schema_info = {}
244             for (table_name,) in tables:
245                 cursor.execute(f"DESCRIBE {table_name}")
246                 table_description = cursor.fetchall()
247                 schema_info[table_name] = table_description
248
249             cursor.close()
250             return schema_info
251         except Error as e:
252             print(f"Error fetching database schema: {e}")
253             return None
254
255     def view_all_records(self, table_name):
```

test_db_connection.py (Bottom Editor):

```
255     def view_all_records(self, table_name):
256         self.ensure_connection()
257         try:
258             cursor = self.connection.cursor(dictionary=True)
259             query = f"SELECT * FROM {table_name}"
260             cursor.execute(query)
261             records = cursor.fetchall()
262             cursor.close()
263             return records
264         except Error as e:
265             print(f"Error fetching records from table {table_name}: {e}")
266             return f"Error fetching records from table {table_name}: {e}"
267
268
```

test_db_connection.py

Personal Income Tax System (PITS)



A screenshot of a code editor window titled "Personal Income Tax System (PITS)". The window shows several tabs at the top: "client1.py", "server1.py", "tax_calculator.py", "tax_database.py", "test_db_connection.py", "user_registration_client.py", and "validator.py". The "test_db_connection.py" tab is currently active. The code in the editor is:

```
1 # test_db_connection.py ...
2 from tax_database import TaxDatabaseServer
3
4 db = TaxDatabaseServer()
5 if db.connection:
6     print("Database connection successful")
7 else:
8     print("Database connection failed")
```

validator.py

Personal Income Tax System (PITS)

The image shows two side-by-side screenshots of a code editor, likely PyCharm, displaying Python code for a Validator class. Both screenshots show the same code structure but with different content in the body of the validate_pay_period method.

```
client1.py server1.py tax_calculator.py tax_database.py test_db_connection.py user_registration_client.py validator.py
```

```
validator.py > ...
```

```
1 import re
2 from datetime import datetime
3
4 class Validator:
5     def validate_tfn(self, tfn):
6         if re.fullmatch(r'\d{8,9}', str(tfn)):
7             return True
8         raise ValueError("Invalid TFN: TFN must be an 8 or 9-digit number.")
9
10    def validate_date(self, date_str):
11        if re.fullmatch(r'\d{4}-\d{2}-\d{2}', date_str):
12            return True
13        raise ValueError("Invalid date format: Date must be in the format YYYY-MM-DD.")
14
15    def validate_wage(self, wage):
16        return self.validate_positive_number(wage, "wage")
17
18    def validate_income(self, income):
19        return self.validate_positive_number(income, "income")
20
21    def validate_tax(self, tax):
22        return self.validate_positive_number(tax, "tax amount")
23
24    def validate_positive_number(self, value, field_name="value"):
25        try:
26            number = float(value)
27            if number > 0:
28                return True
29            raise ValueError(f"Invalid {field_name}: It must be a positive number.")
30        except ValueError:
31            raise ValueError(f"Invalid {field_name}: It must be a numeric value.")
32
33    def validate_pay_period(self, start_date, end_date):
34        try:
35            start = datetime.strptime(start_date, "%Y-%m-%d")
36            end = datetime.strptime(end_date, "%Y-%m-%d")
37            if end >= start:
38                return True
39        except ValueError:
40            raise ValueError("Invalid pay period: End date cannot be earlier than the start date.")
41
42    def validate_boolean_input(self, value):
43        if value.lower() in ['yes', 'no']:
44            return True
45        raise ValueError("Invalid input: Please enter 'yes' or 'no'.")
46
47
```

The top screenshot shows the original implementation where the validation logic for dates and numbers is combined in the validate_positive_number method. The bottom screenshot shows a modified implementation where the validation logic for dates is moved into its own validate_pay_period method, and the validate_positive_number method is simplified to only handle numeric values.

User Manual

Installation

1. Install Python

- Download and install Python from the official website- [python.org/downloads/](https://www.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.nameserverdownload](https://dev.mysql.com/get/mysql-5.7.32-linux-glibc2.12-x86_64.tar.gz)
- Follow the setup instructions and set up a root password.

Verify MySQL Installation by entering the following command

- mysql –version

Personal Income Tax System (PITS)

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
```

Personal Income Tax System (PITS)

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)
);
```

Personal Income Tax System (PITS)

```
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)
);
```

Personal Income Tax System (PITS)

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 –

```
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: 1
```

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

Personal Income Tax System (PITS)

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

Personal Income Tax System (PITS)

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  x  client.py
C:\Users\minag> server.py ...
1 import Pyro5.api
2
3 @Pyro5.api.expose
4 class TaxCalculator:
5     def calculate_tax(self, tfn):
6         # Simulated calculation logic for TFN-based tax calculation
7         if tfn == 123456:
8             return 1000 # Example refund amount
9         else:
10            return "Error: TFN not found."
11
12     def calculate_tax_with_biweekly_data(self, net_wages, tax_withheld):
13         annual_income = net_wages * 26
14         tax_due = annual_income * 0.325 # Tax rate applied based on the taxable income
15         total_tax_due = tax_due - tax_withheld
16         return total_tax_due
17
18     def calculate_medicare_levy(self, annual_income):
19         return annual_income * 0.02
20
21     def calculate_medicare_surcharge(self, annual_income, has_insurance):
22         if not has_insurance and annual_income > 90000:
23             if annual_income <= 105000:
24                 return annual_income * 0.01
25             elif annual_income <= 140000:
26                 return annual_income * 0.0125
27             else:
28                 return annual_income * 0.015
29         return 0
30
31     def main():
32         daemon = Pyro5.api.Daemon()
33         ns = Pyro5.api.locate_ns()
34         uri = daemon.register(TaxCalculator)
35         ns.register("Aus.taxcalculator", uri)
36         print("Server 1 is ready and waiting for clients.")
37         daemon.requestLoop()
38
39 if __name__ == "__main__":
40     main()
```

Personal Income Tax System (PITS)

```
server.py  client.py  x
C:\Users\minag> python client.py
1  import Pyro5.api
2
3  def main():
4      tax_server = Pyro5.api.Proxy("PYRONAME:Aus.taxcalculator")
5
6      choice = input("Enter 1 for tax calculation with TFN or 2 without TFN: ").strip()
7      if choice == "1":
8          tfn = input("Enter your TFN: ").strip()
9          if tfn.isdigit():
10             try:
11                 refund_or_payment = tax_server.calculate_tax(int(tfn))
12                 if isinstance(refund_or_payment, str):
13                     print(refund_or_payment)
14                 elif refund_or_payment < 0:
15                     print(f"You owe additional tax: ${-refund_or_payment:.2f}")
16                 else:
17                     print(f"You will receive a refund: ${refund_or_payment:.2f}")
18             except Exception as e:
19                 print(f"Error occurred during tax calculation: {e}")
20         else:
21             print("Error: TFN must be a valid number.")
22     elif choice == "2":
23         net_wages = float(input("Enter your biweekly net wages: "))
24         tax_withheld = float(input("Enter your tax withheld: "))
25         annual_income = input("Enter your estimated annual income: ")
26
27     if not annual_income.isdigit():
28         print("Error: Annual income must be a valid number.")
29         return
30
31     has_insurance = input("Do you have private health insurance (yes/no)? ").strip().lower() == "yes"
32
33     try:
34         # Call the methods for tax calculation
35         tax_due = tax_server.calculate_tax_with_biweekly_data(net_wages, tax_withheld)
36         medicare Levy = tax_server.calculate_medicare_levy(float(annual_income))
37         mls = tax_server.calculate_medicare_levy_surcharge(float(annual_income), has_insurance)
38         total_tax = tax_due + medicare Levy + mls
39
40         # Adjusting the logic for refund or tax owed
41         if total_tax < 0:
42             print(f"You will receive a refund: ${-total_tax:.2f}")
43         else:
44             print(f"You owe additional tax: ${total_tax:.2f}")
45     except Exception as e:
46         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>
```

Personal Income Tax System (PITS)

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)
```

Personal Income Tax System (PITS)

```
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
```

Personal Income Tax System (PITS)

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: 1

Personal Income Tax System (PITS)

```
Enter your choice: 1
{'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_income': '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', 'annual_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', 'annual_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', '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: 1
```

```
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: 1
```

```
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)
```

Personal Income Tax System (PITS)

```
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): no
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', 'annual_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
```

Personal Income Tax System (PITS)

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

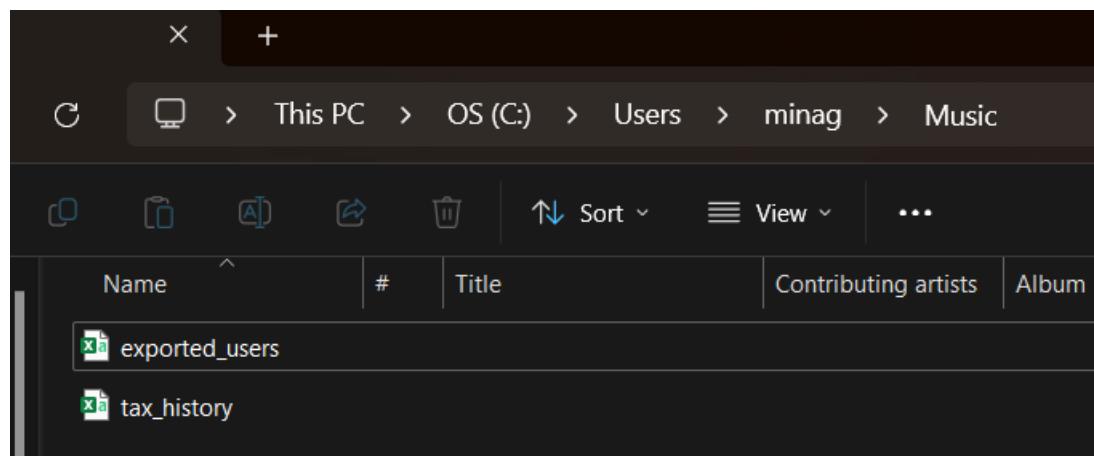
Delete Record:
Enter the table name: users
Enter the record ID to delete: 18
Record deleted successfully.

Personal Income Tax System (PITS)

```
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.
```



Personal Income Tax System (PITS)

```
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_income": "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", "annual_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", "annual_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", "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}, {"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": "10000.00", "has_health_insurance": 1}]
```

Bonus Marks:

```
101 EXIT Admin menu
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: )
```

Personal Income Tax System (PITS)

```
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_income": "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", "annual_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", "annual_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", "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}, {"id": 15, "person_id": "user107", "tfn": "U7392071", "name": "John Doe", "email": "john.doe@gmail.com", "password": "userpass107", "role": "user", "annual_income": "100000.00", "tax_withheld": "10000.00", "has_health_insurance": 1}, {"id": 20, "person_id": "user110", "tfn": "47502438", "name": "Kelly Reynolds", "email": "kelly.reynolds@gmail.com", "password": "userpass110", "role": "user", "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", "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)
```

Personal Income Tax System (PITS)

```
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)
```

```
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:

Personal Income Tax System (PITS)

```
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): olive.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 | olive.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.
```

Personal Income Tax System (PITS)

```
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.

Personal Income Tax System (PITS)

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)
);
```

Personal Income Tax System (PITS)