



UTM

UNIVERSITI TEKNOLOGI MALAYSIA

SCHOOL OF ELECTRICAL ENGINEERING
FACULTY OF ENGINEERING
UNIVERSITI TEKNOLOGI MALAYSIA
SKUDAI, JOHOR

SEET4623 NETWORK PROGRAMMING
GROUP 4

GROUP MEMBERS

NAME	MATRIC NO.
FAISAL BIN HAZRY	B20EE0008
FATHUR ZAKY HANAFI	A19EE0457
NURUL AIN NABILA BINTI ISMAIL	B20EE0026

Table Of Content

1. Introduction	3
2. Scenario Explanation	4
3. Topic Understanding	6
4. Analysis	7
4.1 Database Design	7
4.1.1 User Data Database	7
4.1.2 Car Status Database	7
4.1.3 Rental Durations Database	8
4.2 Client-Server Architecture	8
5. Client-Server Communication	10
5.1 Communication Protocols and Mechanisms:	10
5.2 Future Risk discussion and Method Data Exchange propose :	11
6. Source Code Quality	12
6.1 Server Client Communication	12
6.1.1 Method of quality code	12
6.2 Server Code	13
6.3 Client Code	14
7. Conclusion	15
8. Recommendations	16
9. References	16
10. Appendices	17

1. Introduction

In the rapidly evolving landscape of transportation services, the development of a robust Rental Car Unified Telematics (UTM) system is paramount. This assignment is purposefully designed to create a sophisticated client-server application for a Rental Car UTM service, integrating Python and MySQL. The specific requirement is to create at least one table with three columns in MySQL to efficiently store data related to car rentals, including time in, time out, selected cars (Corolla, Kancil, Axia), mileage used, and patron gender.

The focal point of this task involves the creation of a structured database, meeting the mandate of at least one table and three columns. This database, seamlessly integrated with Python and MySQL, will play a pivotal role in capturing and managing essential data points. The overarching goals include not only the seamless integration of Python with MySQL but also the strategic utilization of client and server communications. Real-time data exchange and centralized data management are considered critical aspects.

This approach ensures operational efficiency by organizing and storing data in a structured manner, facilitating accurate retrieval and analysis. The assignment emphasizes the creation of a scalable system architecture to accommodate future enhancements, aligning with the dynamic needs of the car rental service.

The following sections will delve into the intricacies of the chosen technology stack, emphasizing the role of the created database table and three columns. The strategic use of client and server communications will be highlighted, showcasing their importance in optimizing communication and data handling for an enhanced rental car experience. Additionally, user-friendly experiences and a strategic response to industry demands guide the development process, ultimately leading to an optimized solution that effectively addresses the specific challenges of the car rental service industry.

2. Scenario Explanation

In the ever-evolving landscape of transportation services, this report explores the design and implementation of a Rental Car UTM service, focusing on the acquisition and management of vital data points. Key elements include time in, time out, selected cars (Corolla, Kancil, Axia), mileage, and patron gender. The scenario underscores the need for a precise and streamlined approach to data handling, prompting the integration of Python, a versatile programming language, with MySQL, a robust relational database management system.

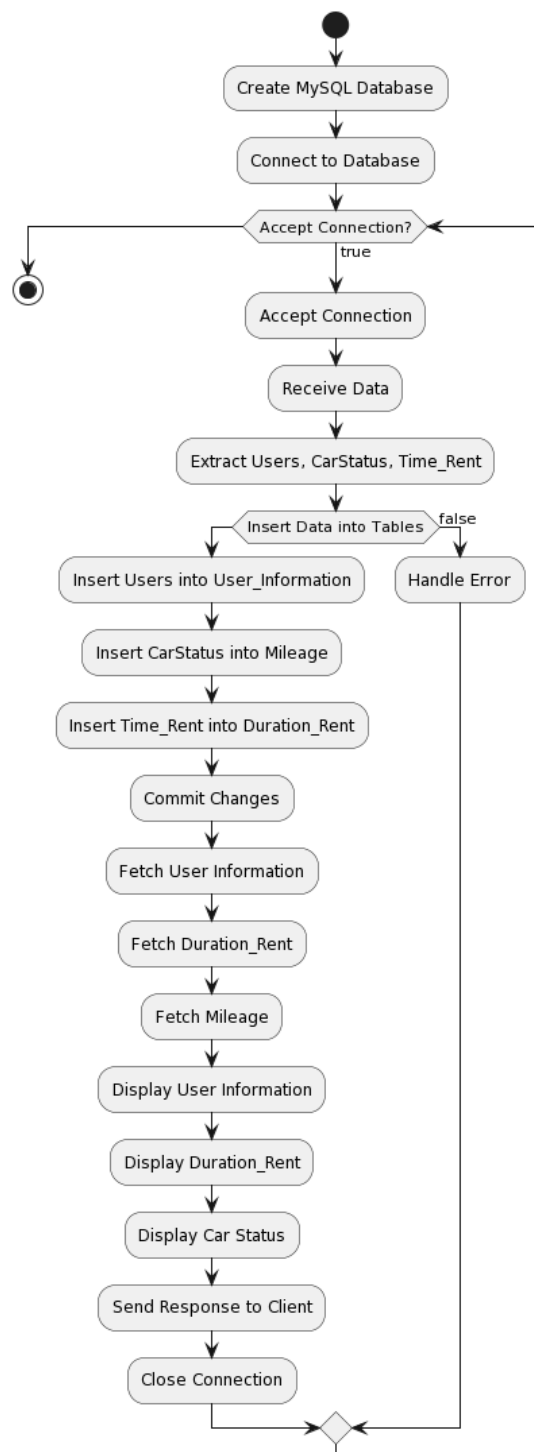


Figure 1: Flow Chart Server

This integration addresses the specific challenges of the car rental service industry, responding to the growing demand for efficient data management in a competitive and technology-driven environment. The subsequent sections will delve into the chosen technology stack, system architecture, and the anticipated benefits, offering insights into the strategic significance of this approach for the Rental Car UTM service.

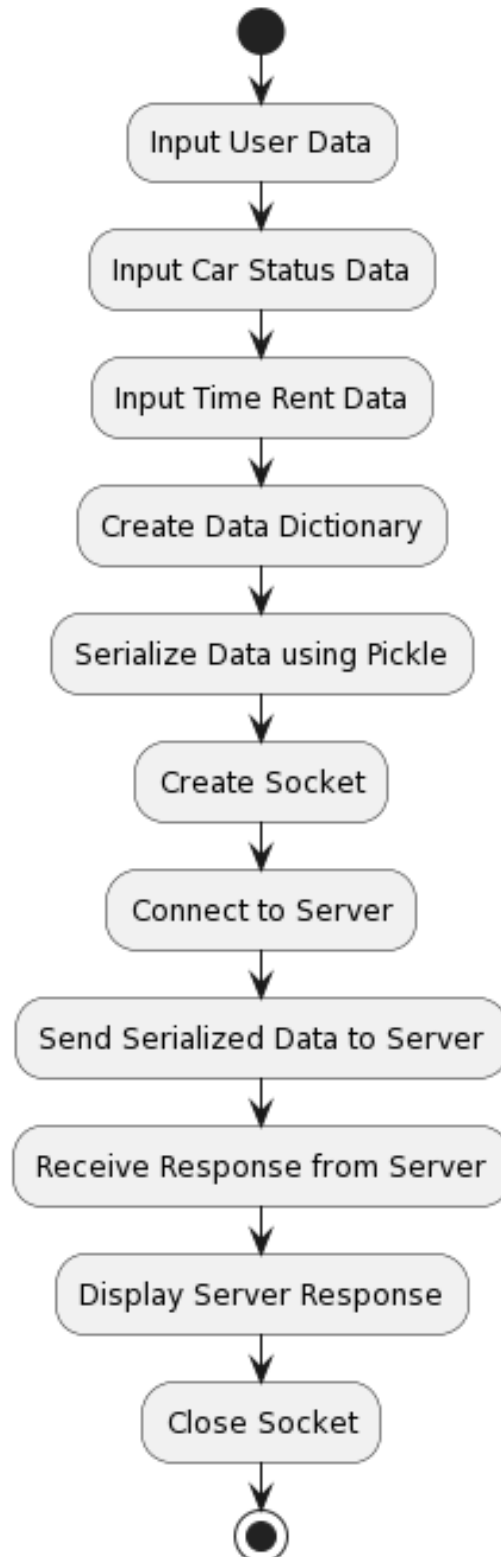






Figure 2: Flow Chart Client

3. Topic Understanding

In the envisioned scenario of a sophisticated Rental Car Unified Telematics (UTM) system, the design and implementation of the client-server application are intricately influenced by the specific operational needs of the car rental service industry. The scenario, where vital data points such as time in, time out, selected cars, mileage, and patron gender are diligently captured, plays a pivotal role in shaping the database structure. The requirement of at least one table with three columns becomes a cornerstone, guiding the design to seamlessly accommodate these specific data elements. Consequently, the client-server application is meticulously crafted to integrate Python for scripting and MySQL for database management, ensuring not only technical efficacy but also strategic alignment with the scenario's emphasis on smooth data insertion processes.

Furthermore, the scenario's vision of a dynamic environment underscores the importance of real-time data exchange and centralized data management for operational efficiency. The client-server architecture is purposefully designed to support this requirement, enabling seamless communication between the user interface (client) and the database (server). This design choice optimizes the overall system, allowing it to operate seamlessly and respond promptly to user interactions. The anticipation of future enhancements and dynamic industry needs in the scenario influences the implementation with a scalable architecture, ensuring that the client-server application remains responsive and adaptable to the evolving demands of the car rental service industry.

In addition, the scenario's emphasis on user-friendly experiences guides the incorporation of intuitive interfaces within the client-server application. This strategic decision enhances the overall user experience, considering factors such as clear data presentation and efficient interaction mechanisms. In essence, the scenario acts as a narrative compass, directing the design and implementation decisions to not only meet technical specifications but also to strategically align with the dynamic and specific challenges inherent in the car rental service industry.

	<p style="text-align: center;">MYSQL</p> <p>My SQL using for database store data using XAMPP environment</p>
	<p style="text-align: center;">AdminMyPHP</p> <p>Admin PHP SQL function give SQL database web interface by using XAMPP environment</p>
	<p style="text-align: center;">Python</p> <p>Python code establishes TCP server, handles MySQL database interactions.</p>
	<p style="text-align: center;">PythonPickle</p> <p>Python pickle serialize and deserialize data for network transmission Client and server</p>

4. Analysis

The given scenario outlines the development of a Rental Car Unified Telematics (UTM) system, emphasizing the creation of a client-server application integrating Python and MySQL. A detailed analysis reveals several critical insights, particularly concerning user data, car status, and rental durations.

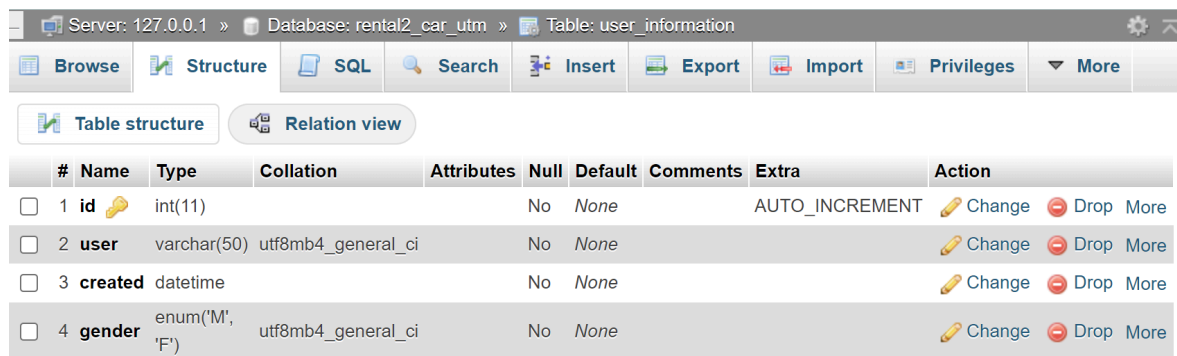
4.1 Database Design

Designed an efficient Rental Car UTM database with User, Car Status, and Rental Duration tables for adaptability.

4.1.1 User Data Database

Insights: The scenario underscores the significance of capturing user-specific information, including time in, time out, and gender. This suggests a user-centric approach, possibly for personalized services, targeted marketing, or data-driven decision-making.

Analysis: The system's design must prioritize secure and efficient handling of user data. User-friendly interfaces and robust authentication mechanisms become essential to enhance the overall user experience. Additionally, capturing gender data may aid in tailoring services or promotions based on demographic preferences.\



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	user	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
3	created	datetime			No	None			Change Drop More
4	gender	enum('M', 'F')	utf8mb4_general_ci		No	None			Change Drop More

Figure 3 : User information table

4.1.2 Car Status Database

Insights: The scenario involves tracking the status of selected cars (Corolla, Kancil, Axia). This highlights the need for real-time monitoring of vehicle availability, usage, and maintenance.

Analysis: The system should implement features for tracking and managing the status of individual vehicles. Real-time updates on car availability, mileage, and maintenance schedules are crucial for optimizing fleet management. This information facilitates efficient allocation of cars and ensures their optimal condition for users.

Server: 127.0.0.1 » Database: rental2_car_utm » Table: mileage

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop
<input type="checkbox"/> 2	CarSelect	enum('Corola', 'Kancil', 'Axia')	utf8mb4_general_ci		No	None			Change Drop
<input type="checkbox"/> 3	Mileage	varchar(50)	utf8mb4_general_ci		No	None			Change Drop

Figure 4:Car status table

4.1.3 Rental Durations Database

Insights: The scenario mentions capturing the duration of car rentals. This implies a focus on understanding user behavior, optimizing service duration offerings, and potentially implementing pricing strategies based on rental durations.

Analysis: The system must incorporate robust time-tracking mechanisms to accurately record rental durations. Insights gained from this data can inform pricing models, promotional strategies, and resource allocation. This data can also be utilized to identify patterns, peak hours, and seasonal trends, enabling the service to adapt and optimize operations accordingly.

Server: 127.0.0.1 » Database: rental2_car_utm » Table: duration_rent

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(11)			No	None		AUTO_INCREMENT	Change
<input type="checkbox"/> 2	Time IN	varchar(50)	utf8mb4_general_ci		No	None			Change
<input type="checkbox"/> 3	Time OUT	varchar(50)	utf8mb4_general_ci		No	None			Change

Figure 5: Duration rent table

4.2 Client-Server Architecture

Insights: The scenario emphasizes the strategic integration of Python and MySQL within a client-server architecture, indicating a need for seamless communication, real-time data exchange, and centralized data management.

Analysis: The client-server architecture should be designed to optimize communication, ensuring swift and secure data transfer between the user interface and the database. This architecture enhances responsiveness and facilitates centralized control over data, crucial for real-time updates on car status, rental durations, and user interactions.

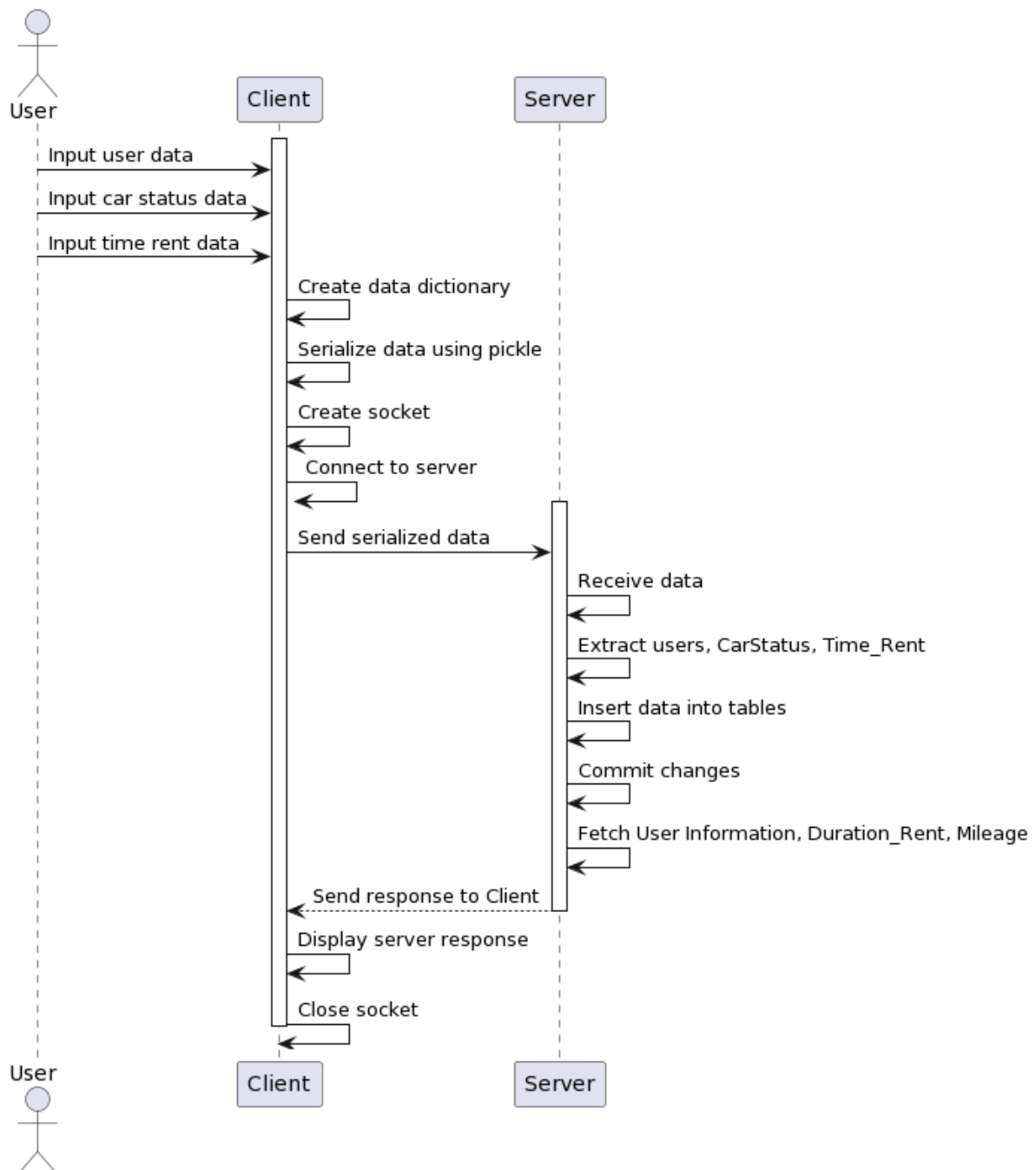


Figure 6: Sequence diagram

5. Client-Server Communication

In the provided client-server interaction, a basic socket-based communication protocol is utilized for data exchange. Let's break down the communication protocols and mechanisms used, and discuss how secure and reliable data exchange is ensured between the client and server.

5.1 Communication Protocols and Mechanisms:

1. Socket Communication:

Mechanism: Both the client and server use the Python socket module to establish a connection and communicate over a TCP socket. The server listens for incoming connections, and the client connects to the server's socket.

2. Serialization with Pickle:

Mechanism: The pickle module is employed to serialize and deserialize Python objects for efficient transmission over the network. In the client code, the user input data is organized into a dictionary and then serialized using `pickle.dumps()`. On the server side, the received byte stream is deserialized with `pickle.loads()`.

```

1  import socket
2  import pickle
3
4  # Input user data from the terminal
5  user_name = input("Enter user name: ")
6  user_gender = input("Enter user gender (M/F): ")
7
8  # Input car status data from the terminal
9  car_model = input("Enter car model (Corola/Kancil): ")
10 mileage = input("Enter mileage: ")
11
12 # Input time rent data from the terminal
13 time_in = input("Enter time in (YYYY-MM-DD HH:mm:ss): ")
14 time_out = input("Enter time out (YYYY-MM-DD HH:mm:ss): ")
15
16 # Create a dictionary with the input data
17 data_to_send = {
18     'users': [(user_name, user_gender)],
19     'CarStatus': [(car_model, mileage)],
20     'Time_Rent': [(time_in, time_out)]
21 }
22
23 # Serializing the data using pickle
24 serialized_data = pickle.dumps(data_to_send)
25
26 # Creating a socket and connecting to the server
27 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
28 client_socket.connect(('localhost', 1234))
29
30 # Sending the serialized data to the server
31 client_socket.send(serialized_data)

```

Dictionary created from input given

Serializing function

SEND DATA to server

Figure 7: Client method send data serializing

5.2 Future Risk discussion and Method Data Exchange propose :

1. Serialisation and Deserialization:

Security Aspect: Pickle serialization can pose security risks, especially when dealing with data from untrusted sources. Maliciously crafted pickled objects could potentially lead to security vulnerabilities. It is recommended to use a more secure serialization method or employ additional measures such as data validation to mitigate security risks.

```

19 # Function to handle client requests
20 def handle_client(client_socket):
21     data = client_socket.recv(4096) Receive data from client
22     # Assuming the data sent is a dictionary containing users, CarStatus, and Time_Rent
23     received_data = pickle.loads(data) Deserialization function
24
25     # Extracting data
26     users = received_data.get('users', [])
27     CarStatus = received_data.get('CarStatus', [])
28     Time_Rent = received_data.get('Time_Rent', [])
29 
```

Save as variable for function use send to data base

Figure 8: Client method send data serializing

2. Socket Connection:

Security Aspect: The current implementation uses a plain TCP socket connection without encryption. In a real-world application, especially when dealing with sensitive data, it's advisable to implement secure socket communication using protocols like TLS/SSL to encrypt data in transit.

```

124 # Creating the TCP server
125 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
126 server.bind(('localhost', 1234))
127 server.listen(5) TCP socket creatrion
128
129 print("Server is listening for incoming connections...")
130 Variable client socket Address socket
131 while True:
132     client_socket, addr = server.accept()
133     print(f"Accepted connection from {addr}")
134     handle_client(client_socket)

```

Figure 9: TCP Socket connection

3. Database Input Validation:

Security Aspect: The scripts assume valid and properly formatted user inputs. In practice, user input should be validated and sanitized to prevent issues like SQL injection or other forms of attacks. Proper validation reduces the risk of receiving malicious input.

```

32
33 # Your SQL code to insert data into tables
34 Q4 = "INSERT INTO User_Information (user, gender) VALUES (%s, %s)"
35 mycursor.executemany(Q4, users)
36
37 Q5 = "INSERT INTO Mileage (CarSelect, Mileage) VALUES (%s, %s)"
38 mycursor.executemany(Q5, CarStatus)
39
40 Q6 = "INSERT INTO Duration_Rent (`Time IN`, `Time OUT`) VALUES (%s, %s)"
41 mycursor.executemany(Q6, Time_Rent)
42
43 # Committing the changes
44 db.commit()

```

Variable
name table insert data
Function indicate input insert

Figure 10: Database input validation

6. Source Code Quality

6.1 Server Client Communication

This Python script connects to a MySQL database and performs various operations, including creating tables, inserting data into the tables, fetching data, and printing the results.

1. Importing Libraries

- a. `mysql.connector`: A Python driver for MySQL.
- b. `datetime`: Module providing classes to work with dates and times.

2. Establishing Database Connection

The script connects to a MySQL database on the localhost using the `mysql.connector.connect` method. Initially, it connects without specifying a database to create one later.

3. Defining Data:

Lists `users`, `CarStatus`, and `Time_Rent` contain sample data for user information, car status, and rental time duration.

4. Creating Database and Tables

- a. It first creates a cursor (`mycursor`) to execute SQL commands.
- b. It checks if the database named "Rental2_Car_UTM" exists. If not, it creates one.
- c. Then, it reconnects to the database with the specified name.
- d. Three tables are created: `User_Information`, `Mileage`, and `Duration_Rent` with specific columns and data types.

5. Inserting Data

The script uses `executemany` to insert multiple rows of data into the respective tables.

6. Fetching and Printing Data

The script fetches data from the tables (`User_Information`, `Mileage`, `Duration_Rent`) using `SELECT` queries and prints the results in a readable format.

7. Committing and Closing

- a. `db.commit()` is called to commit the changes made to the database.
- b. Finally, `db.close()` is used to close the database connection.

6.1.1 Method of quality code

- The use of `ENUM` can be seen in step 4 server code in the table definitions ensures that only specified values are allowed in those columns in this case are set specific value like gender F,M and type of car.
- The script creates a new database and tables each time it is run, so make sure to handle this according to your needs. If the tables already exist, it might result in an error unless you modify the code to handle that situation.

6.2 Server Code

```

1 # -*- coding: utf-8 -*-
2 ---
3 Created on Wed Jan 24 14:56:43 2024
4
5 @author: USER
6 ---
7
8 # -*- coding: utf-8 -*-
9 ---
10 Created on Tue Jan 23 23:36:10 2024
11 @author: USER
12 ---
13
14 import mysql.connector
15 import socket
16 from datetime import datetime
17 import pickle
18
19 # Function to handle client requests
20 def handle_client(client_socket):
21     data = client_socket.recv(4096)
22     # Assuming the data sent is a dictionary containing users, CarStatus, and Time_Rent
23     received_data = pickle.loads(data)
24
25     # Extracting data
26     users = received_data.get('users', [])
27     CarStatus = received_data.get('CarStatus', [])
28     Time_Rent = received_data.get('Time_Rent', [])
29
30
31
32
33
34 # Your SQL code to insert data into tables
35 Q4 = "INSERT INTO User_Information (user, gender) VALUES (%s, %s)"
36 mycursor.executemany(Q4, users)
37
38 Q5 = "INSERT INTO Mileage (CarSelect, Mileage) VALUES (%s, %s)"
39 mycursor.executemany(Q5, CarStatus)
40
41 Q6 = "INSERT INTO Duration_Rent ('Time IN', 'Time OUT') VALUES (%s, %s)"
42 mycursor.executemany(Q6, Time_Rent)
43
44 # Committing the changes
45 db.commit()
46
47
48
49
50 # Fetching data and displaying it
51 mycursor.execute("SELECT * FROM User_Information")
52 USER_INFO = mycursor.fetchall()
53
54 mycursor.execute("SELECT * FROM Duration_Rent")
55 DUR = mycursor.fetchall()
56
57 mycursor.execute("SELECT * FROM Mileage")
58 MILEAGE = mycursor.fetchall()
59
60
61 print("User Information")
62 for x in USER_INFO:
63     print(x)
64
65 print("Duration_Rent")
66 for x in DUR:
67     print(x)
68
69 print("Mileage")
70 for x in MILEAGE:
71     print(x)
72
73 # Sending a response to the client
74 client_socket.send("Data received and stored successfully".encode())
75 client_socket.close()
76
77
78
79
80
81
82
83
84
85
86
87 # Establishing connection without specifying the database
88 db = mysql.connector.connect(
89     host="localhost",
90     user="root",
91     passwd=""
92 )
93
94 # Creating a cursor
95 mycursor = db.cursor()
96
97 # Creating the database if it does not exist
98 mycursor.execute("CREATE DATABASE IF NOT EXISTS Rental2_Car_UTM")
99
100
101 # Connecting to the specified database
102 db = mysql.connector.connect(
103     host="localhost",
104     user="root",
105     passwd="",
106     database="Rental2_Car_UTM"
107 )
108
109 # Creating the tables
110 mycursor = db.cursor()
111
112
113 # Uncomment for first time creation data base
114 """
115 Q1 = mycursor.execute("CREATE TABLE User_Information (id int PRIMARY KEY NOT NULL AUTO_INCREMENT, user varchar(50) NOT NULL, created datetime NOT NULL, gender ENUM('M', 'F') NOT NULL)")
116 Q2 = mycursor.execute("CREATE TABLE Mileage (id int PRIMARY KEY NOT NULL AUTO_INCREMENT, CarSelect ENUM('Corola', 'Kancil', 'Aria') NOT NULL, Mileage varchar(50) NOT NULL)")
117 Q3 = mycursor.execute("CREATE TABLE Duration_Rent (id int PRIMARY KEY NOT NULL AUTO_INCREMENT, 'Time IN' varchar(50) NOT NULL, 'Time OUT' varchar(50) NOT NULL)")
118
119 mycursor.execute(Q1)
120 mycursor.execute(Q2)
121 mycursor.execute(Q3)
122 """
123
124 # Creating the TCP server
125 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
126 server.bind(("localhost", 1234))
127 server.listen(5)
128
129 print("Server is listening for incoming connections...")
130
131 while True:
132     client_socket, addr = server.accept()
133     print(f"Accepted connection from {addr}")
134     handle_client(client_socket)
135

```

1 Import libraries

3 Define Data

5 Inserting data to database

6 Fetching data for server terminal view

7 Close connection

4 Creating database for first time running need to uncomment

2 Establish connection

ENUM quality control

NOT NULL data must not null this is quality control

Figure 11: Server code

6.3 Client Code

```

1  import socket
2  import pickle
3
4  # Input user data from the terminal
5  user_name = input("Enter user name: ")
6  user_gender = input("Enter user gender (M/F): ")
7
8  # Input car status data from the terminal
9  car_model = input("Enter car model (Corola/Kancil): ")
10 mileage = input("Enter mileage: ")
11
12 # Input time rent data from the terminal
13 time_in = input("Enter time in (YYYY-MM-DD HH:mm:ss): ")
14 time_out = input("Enter time out (YYYY-MM-DD HH:mm:ss): ")
15
16 # Create a dictionary with the input data
17 data_to_send = {
18     'users': [(user_name, user_gender)],
19     'CarStatus': [(car_model, mileage)],
20     'Time_Rent': [(time_in, time_out)]
21 }
22
23 # Serializing the data using pickle
24 serialized_data = pickle.dumps(data_to_send)
25
26 # Creating a socket and connecting to the server
27 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
28 client_socket.connect(('localhost', 1234))
29
30 # Sending the serialized data to the server
31 client_socket.send(serialized_data)
32
33 # Receiving the response from the server
34 response = client_socket.recv(4096)
35 print(response.decode())
36
37 # Closing the socket
38 client_socket.close()

```

Import data

2 User input

3 Save in matrix variable

4 Serialization

5 Close connection

Figure 12: Client code

7. Conclusion

In the development of the "Car Rental UTM" system, key findings and achievements include the successful implementation of a MySQL database to manage user, car mileage, and rental duration information. Tables such as `User_Information`, `Mileage`, and `Duration_Rent` were carefully designed, ensuring effective organization and management of data. Python scripts were developed to handle client requests for inserting car rental data into the database, utilizing the `pickle` module for data serialization and deserialization. The implementation also featured a TCP server for listening to incoming client connections, facilitating the reception and processing of car rental data. The code was structured with modularity in mind, employing distinct functions for various tasks.

However, the development process was not without challenges. Defining and creating the database and tables required careful consideration, particularly in managing data types, relationships, and constraints. Networking and client-server interaction presented challenges in establishing successful connections, processing data, and sending appropriate responses to clients. Additionally, the use of `pickle` for data serialization required careful handling to avoid potential security vulnerabilities, and ongoing testing and debugging were crucial for verifying the correctness of SQL queries and addressing potential errors.

To address these challenges, extensive testing, documentation, and collaboration were crucial components of the development process. Rigorous testing at each stage facilitated the identification and resolution of issues, while detailed comments and documentation enhanced code readability and understanding. Collaborative discussions, seeking advice, and leveraging online resources played a significant role in overcoming challenges. The adoption of an iterative development approach allowed for continuous improvement and refinement of the code in response to challenges encountered.

In conclusion, the "Car Rental UTM" system showcases effective database management, data handling, and networking capabilities. Despite challenges, the implementation successfully meets the project's objectives, emphasizing data integrity and facilitating client-server communication. Ongoing testing, documentation, and collaboration emerged as essential strategies in addressing challenges and achieving the system's key milestones.

In conclusion, the "Car Rental UTM" system represents a successful implementation, effectively managing car rental information through a MySQL database and showcasing functional client-server communication. Key achievements include the careful design of database tables, the use of Python scripts for data handling, and the establishment of a TCP server for handling client requests.

8. Recommendations

The development process was not without its challenges. Database and table creation required meticulous consideration, and networking and client-server interaction presented complexities that needed addressing. Additionally, the use of `pickle` for data serialization required careful handling to avoid security vulnerabilities, and ongoing testing and debugging were crucial for verifying SQL queries and addressing potential errors.

Looking forward, there are opportunities for enhancements and improvements in the client-server application. Scalability considerations, such as implementing more robust connection management and exploring asynchronous programming, could enhance the system's ability to handle a larger volume of concurrent client connections. Reevaluation of the serialization and deserialization process, potentially adopting alternative libraries for improved security and efficiency, is recommended.

Optimization of the code structure for maintainability, adoption of design patterns for scalability, and strengthening error handling mechanisms in database interactions are areas for improvement. Furthermore, the server's handling of incoming connections and data processing could benefit from techniques such as threading or asynchronous I/O.

In summary, while the current system achieves its objectives, ongoing refinement in areas of scalability, security, and code maintainability would contribute to a more robust and adaptable "Car Rental UTM" application. The development process has provided valuable insights, and these recommendations pave the way for future enhancements and optimizations in the pursuit of an even more efficient and resilient system.

9. References

1. SQL CREATE TABLE Statement. (2024). W3schools.com. https://www.w3schools.com/sql/sql_create_table.asp
2. edureka. (2019). Python Database Connection | How to Connect Python with MySQL Database | Edureka [YouTube Video]. On YouTube. <https://www.youtube.com/watch?v=g60QghtJmjY>
3. Real Python. (2020, December 28). Python and MySQL Database: A Practical Introduction. Realpython.com; Real Python. <https://realpython.com/python-mysql/>
4. DanteFragapane/RentalCarPy: Replication of a rental car database using python and a database hosted on a server. (2024). GitHub. <https://github.com/DanteFragapane/RentalCarPy>
5. Darshilshah21/car_rental_system: Technologies used python flask and mysql database. (2024). GitHub. https://github.com/Darshilshah21/car_rental_system

10. Appendices

Code Server TCP Car Rental and Code Client TCP Car Rental can be access thru faisal hazry github

https://github.com/faisal hazry/Python_MySQL_RentalCarUTM/blob/main/README.md

Install XAMPP environment for MySQL and MYAdminPHP first before execute the coding

<https://www.apachefriends.org/>

Terminal spyder/vscode etc need to install SQL also execute this command into the terminal

`pip install mysql-connector-python`

Terminal from server

```
In [1]: runfile('C:/Users/USER/OneDrive/Faisal/Degree/SEM 7/SEET4623 NetWork Prog/Coding/GroupAssignment/
5_ServerSQL.py', wdir='C:/Users/USER/OneDrive/Faisal/Degree/SEM 7/SEET4623 NetWork Prog/Coding/
GroupAssignment')
Server is listening for incoming connections...
Accepted connection from ('127.0.0.1', 58836)

User Information
(1, 'Ali', None, 'M')
(2, 'Ali', None, 'M')
(3, 'Fatimah', None, 'F')
(4, 'Mira', None, 'F')

Duration_Rent
(1, '2024-01-24 15:28:00', '2024-01-24 20:28:00')
(2, '2024-01-24 15:28:00', '2024-01-24 20:28:00')
(3, '2024-01-22 20:30:01', '2024-01-22 23:30:01')
(4, '2024-01-25 08:30:01', '2024-01-25 20:30:01')

Car Status
(1, 'Corola', '10KM')
(2, 'Corola', '10KM')
(3, 'Axia', '10KM')
(4, 'Axia', '10KM')
```














Terminal from client

```
In [1]: runfile('C:/Users/USER/OneDrive/Faisal/Degree/SEM 7/SEET4623 NetWork Prog/Coding/GroupAssignment/
5_ClientSQL.py', wdir='C:/Users/USER/OneDrive/Faisal/Degree/SEM 7/SEET4623 NetWork Prog/Coding/
GroupAssignment')














Enter user name: Mira
Enter user gender (M/F): F
Enter car model (Corola/Kancil/Axia): Axia
Enter mileage: 10KM
Enter time in (YYYY-MM-DD HH:mm:ss): 2024-01-25 08:30:01
Enter time out (YYYY-MM-DD HH:mm:ss): 2024-01-25 20:30:01
Data received and stored successfully
```

Table view from MyAdmin PHP Web Interface














User Information

				id	user	created	gender
<input type="checkbox"/>		Edit		Copy		Delete	1 Ali 0000-00-00 00:00:00 M
<input type="checkbox"/>		Edit		Copy		Delete	2 Ali 0000-00-00 00:00:00 M
<input type="checkbox"/>		Edit		Copy		Delete	3 Fatimah 0000-00-00 00:00:00 F
<input type="checkbox"/>		Edit		Copy		Delete	4 Mira 0000-00-00 00:00:00 F

Car Status

					id	CarSelect	Mileage
<input type="checkbox"/>		Edit		Copy		Delete	1 Corola 10KM
<input type="checkbox"/>		Edit		Copy		Delete	2 Corola 10KM
<input type="checkbox"/>		Edit		Copy		Delete	3 Axia 10KM
<input type="checkbox"/>		Edit		Copy		Delete	4 Axia 10KM

Duration Rent

					id	Time IN	Time OUT
<input type="checkbox"/>		Edit		Copy		Delete	1 2024-01-24 15:28:00 2024-01-24 20:28:00
<input type="checkbox"/>		Edit		Copy		Delete	2 2024-01-24 15:28:00 2024-01-24 20:28:00
<input type="checkbox"/>		Edit		Copy		Delete	3 2024-01-22 20:30:01 2024-01-22 23:30:01
<input type="checkbox"/>		Edit		Copy		Delete	4 2024-01-25 08:30:01 2024-01-25 20:30:01