AMERICAN UNIVERSITY OF BEIRUT
MAROUN SEMAAN FACULTY OF ENGINEERING & ARCHITECTURE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
EECE 351 - Computing Networks and Services

Networking Programming Project
AUBus – Technical Report

Professor: Ayman Tajeddine

<u>Team Members:</u>

Ayat Nassar (ain10@mail.aub.edu , % of effort : 33.33)

Aya Halawi (ahh100@mail.aub.edu , % of effort : 33.33)

Chaza Kazem (cik03@mail.aub.edu , % of effort : 33.33)

The full code is provided in the submission.

I. Description of the System Architecture and the Protocol Used Between Communicating Entities

System Architecture

The AUBus platform is designed as a hybrid client-server and peer-to-peer architecture using Python's socket programming for network communication. The architecture consists of a centralized server and multiple clients that interact with each other both through the server and directly in peer-to-peer manner for chat functionality.

Server

The server handles all client requests related to user authentication, profile management, ride requests, driver schedules, ratings, notifications, and emergency contacts. It supports multithreading to handle multiple clients simultaneously. The server uses SQLite as its database, with a comprehensive schema defined in init_database.py storing user accounts, profiles, ride requests, ratings, driver schedules, emergency contacts, and chat messages.

Clients

Clients connect to the server through localhost:8888 by default. The client application is built using PyQt5, providing a GUI for users to register, log in, create profiles, manage driver schedules, request rides, exchange P2P messages, rate other users, view ride history, manage emergency contacts, and view real-time weather.

Peer-to-Peer Communication

For real-time chat, the platform implements P2P architecture. Once a ride is accepted, driver and passenger exchange messages directly without routing through the server. The P2P chat module (p2p_chat.py) establishes direct TCP connections. Each client runs a chat server on port 9000 and supports text messages, images (base64-encoded), voice messages, and location sharing (GPS coordinates), Multimedia P2P chat is implemented on its own, but not integrated with database.

Protocol

The communication protocol is a custom application-layer protocol over TCP using pipe-delimited format where commands and responses are structured as strings with fields separated by the pipe character (|).

Client-to-Server Protocol Format

Format: COMMAND_TYPE|parameter1|parameter2|parameter3|...

Examples:

- Registration: REGISTER|username|email|password|first_name|last_name|area|photo_path|is_driver
- Login: LOGIN|username|password
- Ride Request: RIDE_REQUEST_CREATE|passenger_id|pickup_area|destination|request_time
- Rating: RATING_SUBMIT|request_id|rater_id|target_id|target_role|rating|comment

Server-to-Client Protocol Format

Responses: SUCCESS|message|data or ERROR|error_message

Examples:

- Success: SUCCESS|User registered successfully|user_id|123
- Error: ERROR|Invalid username or password

Transport Layer Protocol

TCP is used for client-server communication due to reliability, ordered delivery, and connection-oriented nature—critical for authentication and ride requests. UDP is used for premature GPS tracking due to low latency and acceptable packet loss in continuous location streams. This hybrid approach is used to optimize reliability and performance.

II. Tabular Presentation of All Project Features

AUBus Features

Feature	Status	Description
User Registration	Successfully Implemented	Users register with username, password, name, and area
User Login	Successfully Implemented	Secure authentication with username/password
Driver Profile Creation	Successfully Implemented	Drivers specify vehicle info and availability
Passenger Profile Creation	Successfully Implemented	Passengers create profiles with personal info
Driver Schedule Management	Successfully Implemented	Drivers add weekly commuting schedules
Ride Request Creation	Successfully Implemented	Passengers request rides with area and time
Ride Request Notification	Successfully Implemented	Server notifies matching drivers
Ride Accept/Decline	Successfully Implemented	Drivers accept or decline ride requests
P2P Text Chat	Successfully Implemented	Direct messaging without server intermediation
Rating System	Successfully Implemented	Users rate each other after rides
Minimum Rating Filter	Successfully Implemented	Set minimum rating thresholds
Weather Display	Successfully Implemented	Current weather via OpenWeatherMap API
Server Multithreading	Successfully Implemented	Handles multiple concurrent connections
Database Management	Successfully Implemented	SQLite stores all data persistently
Session Management	Successfully Implemented	Tracks active user sessions

Premium AUBus Features

Feature	Status	Description
Update Profile	Successfully Implemented	User can update his/her info
Google Maps Integration	Successfully Implemented	Interactive map within Application
Emergency Contacts	Successfully Implemented	Add emergency contacts
Emergency Trigger	Successfully Implemented	Panic button alerts contacts
Ride History	Successfully Implemented	View complete ride history
Driver Statistics	Successfully Implemented	Performance stats and earnings
Passenger Statistics	Successfully Implemented	Ride patterns and spending
Active Ride Monitoring	Successfully Implemented	Real-time status updates
Driver Availability Toggle	Successfully Implemented	Go online/offline dynamically
Notification System	Successfully Implemented	In-app notifications
Profile Photo Upload	Successfully Implemented	Upload custom pictures
Rating History	Successfully Implemented	View detailed rating history
Ride Cancellation	Successfully Implemented	Cancel rides with reason tracking

III. Description of the Implementation of Different Functionalities

User Registration and Login

Users register through the GUI by providing username, password, name, and area. The client constructs a REGISTER command and sends it to the server via ServerIntegration class. The server validates input using the validator module, checks username uniqueness, and inserts the new user into the database. For login, users provide username and password. The server authenticates credentials against the database and maintains sessions using in-memory dictionaries mapping user IDs to session tokens and socket connections to user IDs.

Profile Management

After login, users create or update profiles containing phone number, area, driver status, and profile photo. The server handles PROFILE_CREATE and PROFILE_UPDATE commands, validating input and updating the profiles table. Profile photos can be uploaded and stored in the database.

Driver Schedule Management

Drivers specify weekly schedules via GUI (driver_schedule.ui) with checkboxes for each day and time windows. Data is stored in driver_schedules table. When passengers create ride requests, the server queries this table to find drivers with matching areas and schedules for notification.

Ride Request System

Passengers create ride requests specifying pickup area and desired time. The client sends RIDE_REQUEST_CREATE command. The server inserts the request into ride_requests table with status 'pending', then immediately notifies relevant drivers by querying for drivers with matching area and schedule availability. Drivers can accept or decline requests. The server implements a locking mechanism (is_locked flag) to prevent race conditions where multiple drivers accept the same ride.

Peer-to-Peer Chat Communication

Once a ride is accepted, driver and passenger communicate directly. The P2P chat (p2p_chat.py) uses TCP sockets. Each client runs a chat server on port 9000 to accept connections and can connect to other users as a client. Messages are exchanged in JSON format.

Rating System

After ride completion, passenger rates driver. Ratings (1-5 stars) are stored in ride_ratings table. Both parties see each other’s rating upon accepting the ride and will be updated to User Statistics.

Weather Service Integration

The platform displays current weather for Beirut using OpenWeatherMap API. The WeatherService class (weather_service.py) updates automatically every 30 minutes and emits Qt signals to update the GUI.

Google Maps Integration

Google Maps API is integrated to display interactive maps. The MapWidget class uses QWebEngineView to embed the map (google_map.html) in PyQt5. External API used : Google Cloud API

GPS Tracking

Real-time GPS tracking uses UDP sockets. The GPSTracker class sends location updates every 3 seconds via UDP to minimize overhead. The tracker can both share location (sendto) and receive location updates (recvfrom). Premature GPS tracking was implement using JavaScript; however, System uses hardcoded coordinates to avoid asking for user permission.

Emergency Contact System

Users can add emergency contacts (phone/email) stored in emergency_contacts table. When an emergency is triggered during a ride, the system creates an emergency event in emergency_events table, retrieves all contacts for the user, and sends notifications (SMS/email) with the user's current location.

Server Multithreading

The server uses Python's threading module to handle multiple concurrent connections. Each client connection runs in a separate daemon thread. Thread-safe database access is ensured using SQLite connections with check_same_thread=False. The server supports both JSON and pipe-delimited protocols for interoperability.

Database Management

SQLite database (aubus.db) is comprehensively defined in init_database.py with tables for users, profiles, sessions, ride_requests, driver_status, driver_schedules, driver_cars, driver_routes, ride_ratings, notifications, ride_locations, emergency_contacts, emergency_events, chat_messages, ride_declines, ride_cancellations, and rides. Indexes on frequently queried fields optimize performance.

IV. Testing and Debugging

Testing Methodology

To ensure reliability and correctness, we implemented a comprehensive testing strategy:

1. Dummy User Testing

We created 5-10 driver accounts with different areas (Hamra, Achrafieh, Verdun) and various schedules to test matching logic. We created 5-10 passenger accounts from different areas. Some accounts were both drivers and passengers to test role-switching. We registered users with valid and invalid data, tested username uniqueness constraints, tested login with correct/incorrect credentials, and verified authentication requirements for protected operations.

2. Component-Level Testing

We tested each major component independently: Authentication (registration, login, session management), Profile Management (create/update profiles, photo uploads), Ride Requests (creation, cancellation, status tracking), Driver Matching (schedule configuration, matching logic), P2P Chat (connections, text messages, network interruptions), and Rating System (submission, validation).

3. User Interface Testing

We opened and tested each UI page independently to identify errors. All UI files (welcomescreen.ui, loginnew.ui, createaccnew.ui, main2.ui, profile.ui, driver_schedule.ui, ride_request.ui, activeride3.ui, activeride_driver.ui, ridehistory.ui, emergency_contacts.ui) were loaded in Qt Designer to verify layouts. We manually tested all buttons, input fields, interactive elements, navigation between screens, and error message displays.

4. Print Statement Debugging

Throughout development, we placed print statements to track execution flow and identify issues. In server.py, we printed received commands, user IDs being processed, and responses sent. In ride.py, we printed ride request creation, matching driver counts, and driver notifications. In p2p_chat.py, we printed connection status, messages sent/received, and errors. These print statements helped us identify where execution stops when errors occur, trace data flow through the system, verify commands reach intended handlers, and monitor thread activity.

5. Network Testing

We tested network functionality by starting the server and connecting multiple clients simultaneously, testing server behavior with client disconnections, verifying graceful handling of malformed messages, testing P2P connections on local network, verifying firewall doesn't block connections.

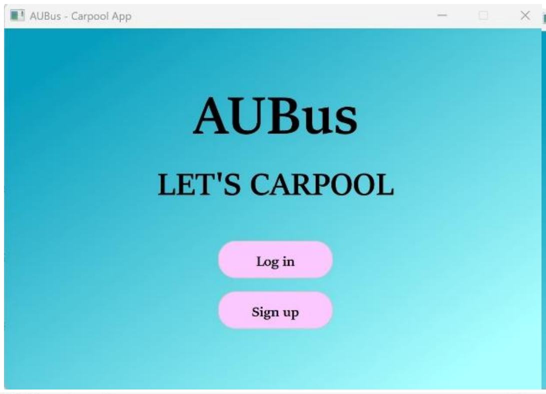
6. Database Testing

We tested all database operations: user registration/login, profile CRUD operations, ride request operations, rating submissions and queries, emergency contact management, and session management. We verified foreign key constraints, tested concurrent writes from multiple threads, verified transaction rollback on errors, and tested database connection pooling

V. Appendix I: Screenshots and Usage Examples

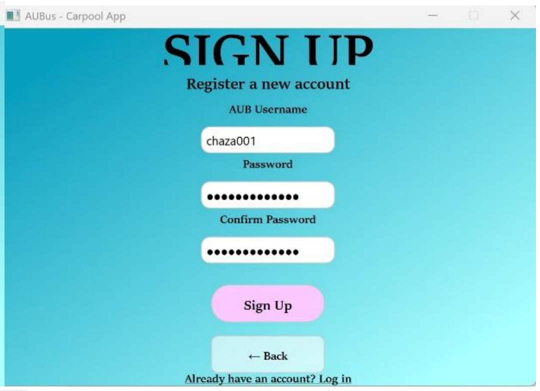
Instructions for Screenshot Demonstration

- User Registration and Login


AUBus - Carpool App

AUBus

LET'S CARPOOL


AUBus - Carpool App

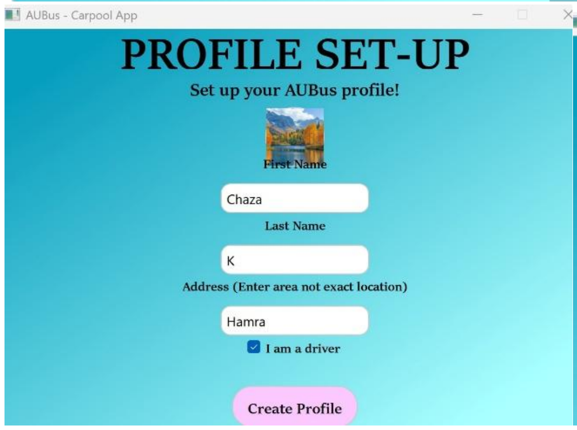
SIGN UP

Register a new account

AUB Username: chaza001

Password: •••••••••••••

Confirm Password: •••••••••••••


AUBus - Carpool App

PROFILE SET-UP

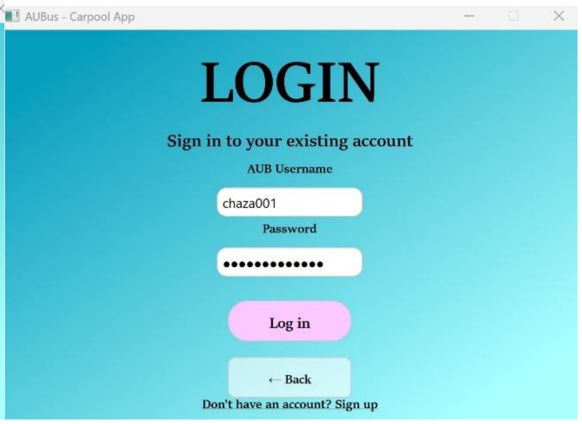
Set up your AUBus profile!

First Name: Chaza

Last Name: K

Address (Enter area not exact location): Hamra

[✓] I am a driver


AUBus - Carpool App

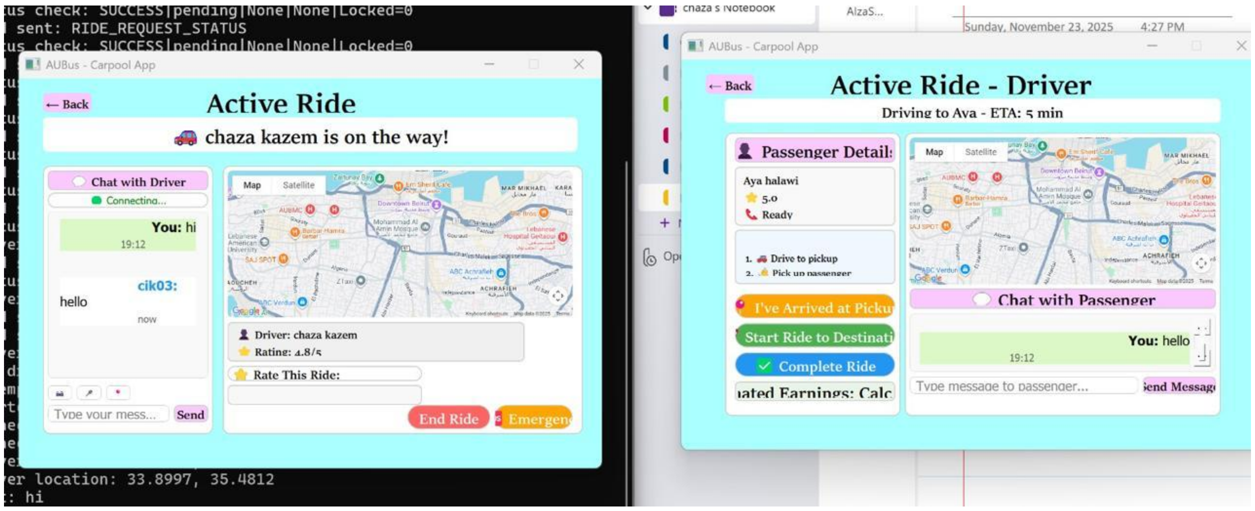
LOGIN

Sign in to your existing account

AUB Username: chaza001

Password: •••••••••••••

Active Ride:



```
us check: SUCCESS|pending|None|None|Locked=0
 sent: RIDE_REQUEST_STATUS
us check: SUCCESS|pending|None|None|Locked=0
er location: 33.8997, 35.4812
: hi
```

AUBus - Carpool App

Active Ride

🚗 chaza kazem is on the way!

Chat with Driver

Connecting...

You: hi
19:12

cik03:
hello
now

Driver: chaza kazem
Rating: 4.8/5

Rate This Ride:

chaza's Notebook

Sunday, November 23, 2025 4:27 PM

AUBus - Carpool App

Active Ride - Driver

Driving to Ava - ETA: 5 min

Passenger Detail:

Aya halawi
⭐ 5.0
📞 Ready

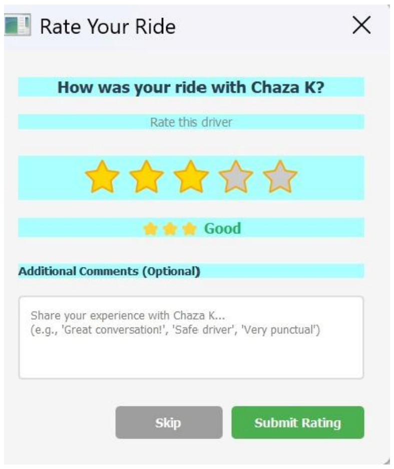
1. 🚗 Drive to pickup
2. Pick up passenger

Chat with Passenger

You: hello
19:12

ated Earnings: Calc

Rating :



Rate Your Ride

How was your ride with Chaza K?

Rate this driver

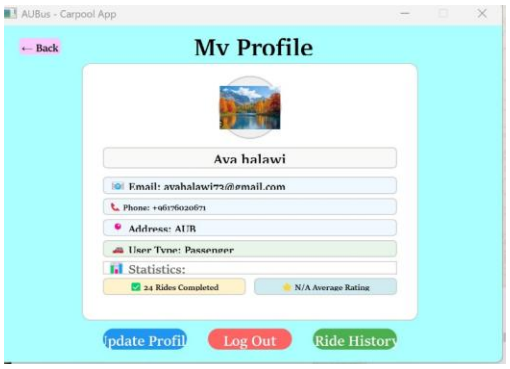
★★★☆☆

⭐⭐⭐ Good

Additional Comments (Optional)

Share your experience with Chaza K...
(e.g., 'Great conversation!', 'Safe driver', 'Very punctual')

Profile :



AUBus - Carpool App

My Profile

Aya halawi

Email: ayahalawi73@gmail.com

Phone: +96176020671

Address: AUB

User Type: Passenger

Statistics:

✓ 24 Rides Completed

⭐ N/A Average Rating

VI. Appendix II: Task Distribution

VI. Appendix II: Task Distribution

	Chaza	Aya	Ayat
Creating a TCP-based client-server architecture			X
Developed and integrated the ride rating system		X	
Built a peer-to-peer (P2P) chat system			X
Testing With Dummy Data	X	X	
Implemented premature code of GPS tracking System			X
Implemented the entire Front-End interface (ui files)		X	
Finalized Two-way P2P between driver and passenger	X		
Integrated External Weather API		X	
Integrated External Google Maps API		X	
Debugged P2P Chat System and integrated it with GUI	X	X	
Debugged Update Profile Feature	X	X	
Background multi-threading Server Concurrency	X		
Fixed issues in the P2P chat system			X
Database integration	X		
HTML formatting, status indicators in P2P chat	X		
Validator, Password Hashing	X		
connection management, server pipelines, JSON <-> pipe conversion	X		

Report	X	X	X

Conclusion

The AUBus platform successfully implements a comprehensive carpooling solution for AUB students with both standard and premium features. The system demonstrates robust client-server architecture with multithreading, hybrid communication model combining server-mediated matching with P2P chat, rich feature set emergency alerts, user-friendly PyQt5 GUI, scalable modular design, and thorough testing methodology.