College of Electrical and Mechanical Engineering

Department of Electrical Engineering

Smart EV Charging Network Simulation

Talha Ahmad
522438

M. Bilal Chaudhry
501887

EE-46-A

CS-107 Project

Submitted to: Professor Soyiba Jawed
Lab Instructor: Mam Eman Fatima

May 14, 2025

Contents

Chapter 1

Smart EV Charging Network

1.1 Introduction

Electric vehicles (EVs) are becoming increasingly popular due to their environmental benefits and government incentives. However, their rising adoption also demands smarter and more scalable charging infrastructure. Smart charging ensures efficient energy usage, prevents system overloads, and provides convenient services to users.

Before building real infrastructure, simulation helps test and optimize designs. This project simulates a smart EV charging network using C++ object-oriented programming to model real-world entities and interactions.

1.2 Problem Statement

This project aims to build a simulation of a city-wide smart EV charging system that must:

- Manage users and their EVs.
- Allow users to book charging time slots and stations.
- Simulate charging stations with multiple docks.
- Use solar or grid energy based on availability.
- Track energy delivered to each EV.
- Prevent overload using dynamic load balancing.
- Calculate bills using real-time pricing rules.
- Handle many users, EVs, and stations to reflect a city-scale deployment.

1.3 Methodology / System Design

The system is implemented in C++ using object-oriented design, enabling modularity, reuse, and scalability.

Main Classes

- **User**: Represents a customer. Can register EVs and make bookings.
- **EV**: An electric vehicle with ID, make, battery capacity, current charge, and port type.
- **ChargingStation**: Represents a location with multiple **ChargingDocks**.
- **ChargingDock**: A charging port with specific power output and availability status.
- **Booking**: Records an EV's scheduled charging session, including start/end time and energy used.
- **EnergySource (abstract)**: Base class for different energy providers.
  - **SolarPower**: Cleaner and cheaper, but limited during night.
  - **GridPower**: Always available but more costly.
- **Invoice**: Calculates and displays billing information after a charging session.
- **PricingEngine**: Applies pricing logic based on charging speed, membership, time of day, and energy source.
- **LoadBalancer**: Distributes load across stations and prevents overload.
- **AnalyticsEngine**: Collects data such as energy usage, session duration, and station popularity.
- **NotificationManager**: Sends simulated alerts like "charging complete" or "bill generated".

How Booking Works

1. User selects EV, station, and time.
2. System finds an available dock.
3. A booking is created and the dock is marked as occupied.
4. Charging session uses solar first, then grid if needed.
5. Session duration and energy usage are tracked.
6. Price is calculated, invoice is printed, and analytics are updated.

Polymorphism

The energy source selection uses polymorphism: both 'SolarPower' and 'GridPower' share a common interface ('EnergySource'), allowing flexible switching without altering core logic.

1.4 Implementation Details

- **Object Relationships**: Users own EVs and Bookings. Bookings link EVs to Docks and track energy used.
- **Charging Process**: ChargingDock checks availability, and Booking manages the session.
- **Energy Flow**: System first tries SolarPower; if not sufficient, it uses GridPower for the remaining energy.
- **Dynamic Pricing**: ‘PricingEngine‘ calculates rates using a ‘switch‘-based model based on:
  - Charging speed (FAST/SLOW)
  - Time of day (PEAK/OFF-PEAK)
  - Membership level (PREMIUM/SILVER/NONE)
  - Energy source (SOLAR/GRID)
  - Usage-based surcharges
- **Load Management**: ‘LoadBalancer‘ keeps track of station capacities to avoid system overload.
- **Analytics**: Logs total energy used, average session duration, and most-used station.

- **Memory Safety**: All objects are managed using ‘std::unique$_ptr$‘$forautomaticandsafememorydealloca$
$Messagesaresimulatedusing$‘$NotificationManager$‘$tomimicuseralerts$.

1.5 Results / Output Screenshots

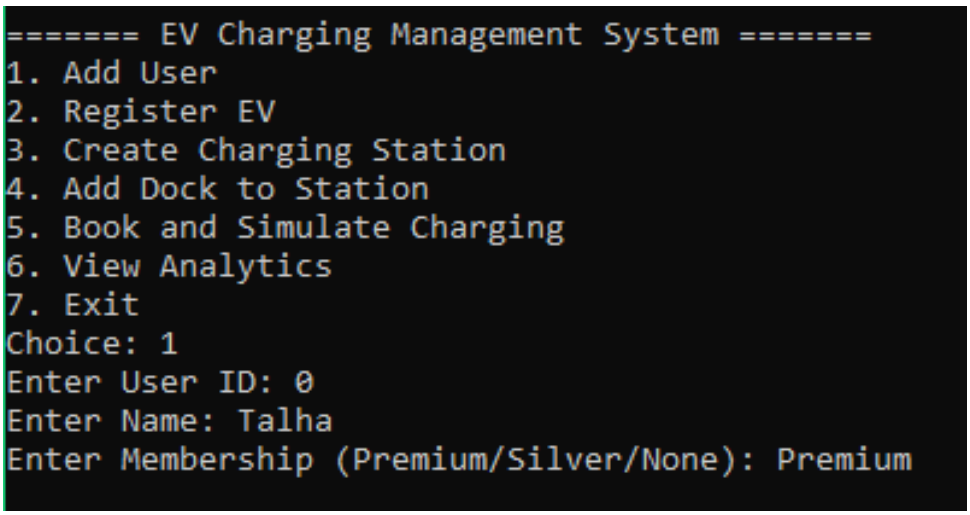

Figure 1.1: User Registration Screen. This screenshot shows the registration of a new user in the system. Users can input their details, such as ID, name, and membership level, which are saved for subsequent EV registrations and bookings.

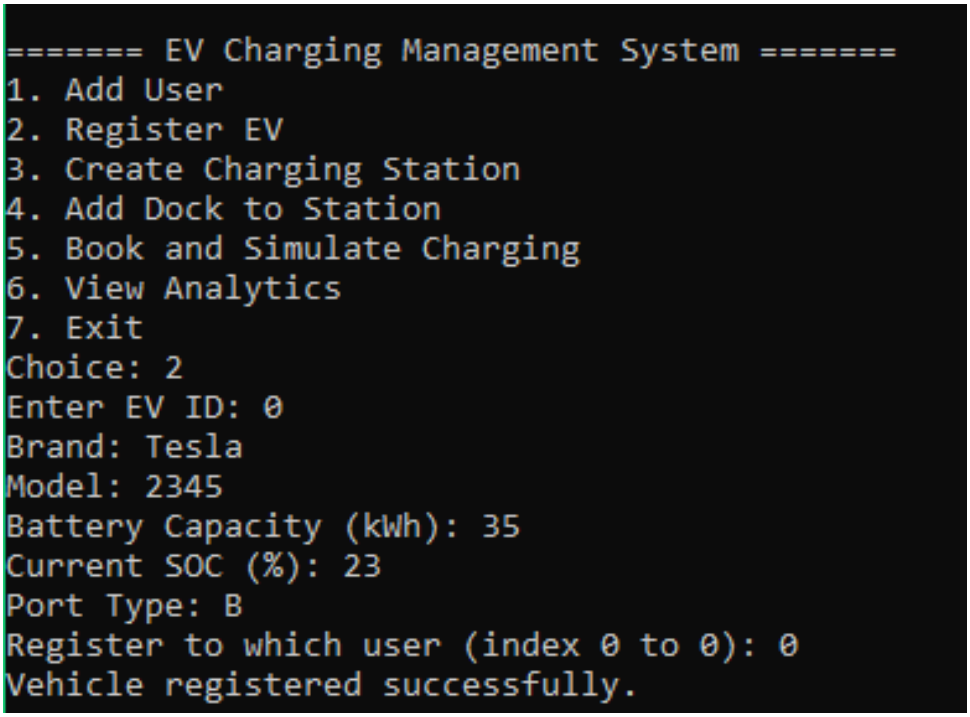

Figure 1.2: EV Registration Screen. Here, a user registers their electric vehicle by entering details such as EV ID, make, model, battery capacity, and port type. The EV is then linked to the selected user.

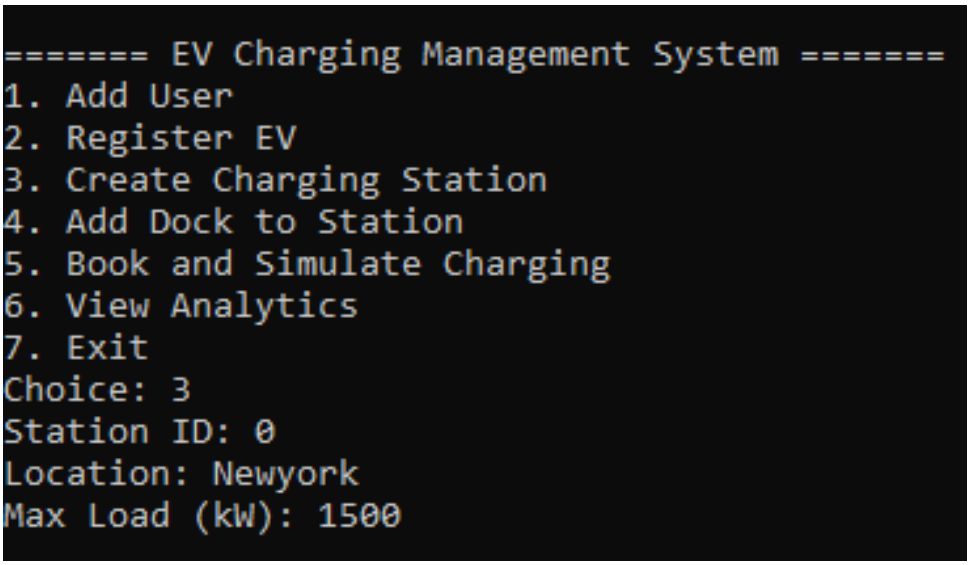

Figure 1.3: Create Charging Station Screen. This screenshot shows the process of creating a new charging station, where the user inputs the station ID, location, and maximum load (kW).

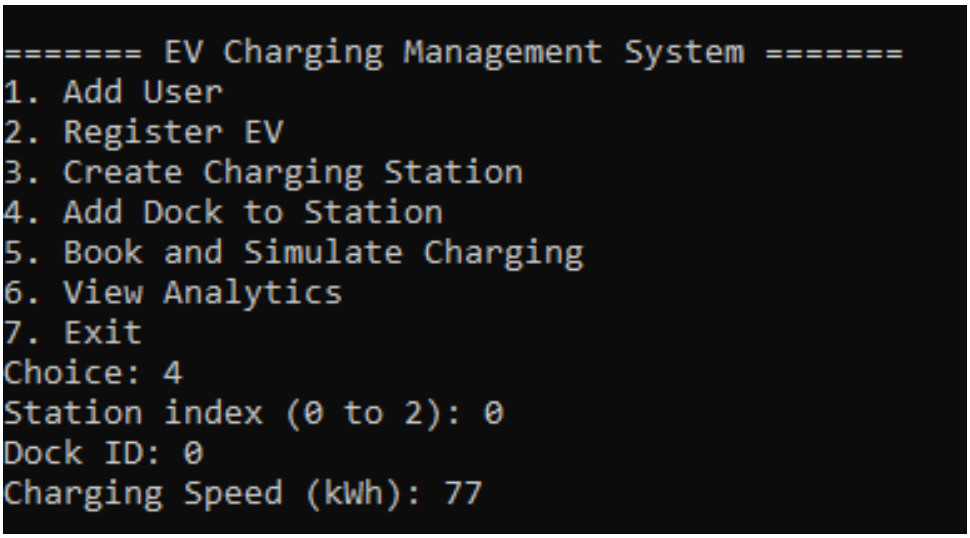

Figure 1.4: Add Dock to Charging Station Screen. This screenshot demonstrates the process of adding a charging dock to a station, where the user provides the dock ID and charging speed.

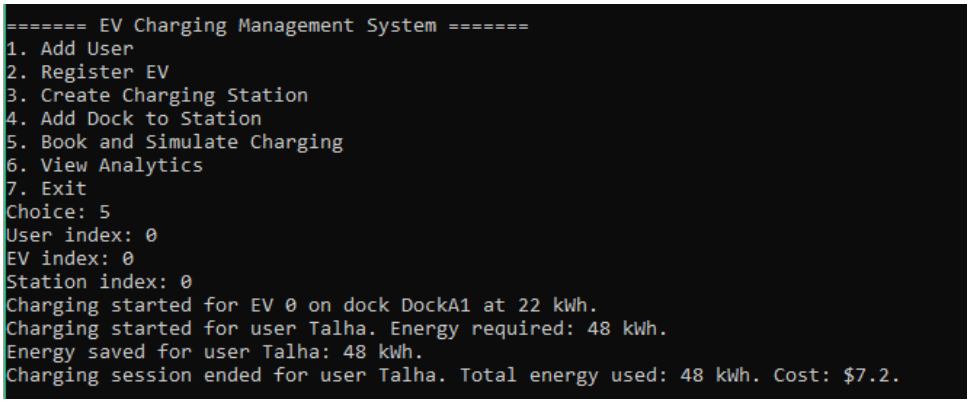

Figure 1.5: Charging Booking Screen. This screenshot shows the booking process where a user selects a charging station, available dock, and time slot for charging their EV. The system checks for dock availability and proceeds with the booking.

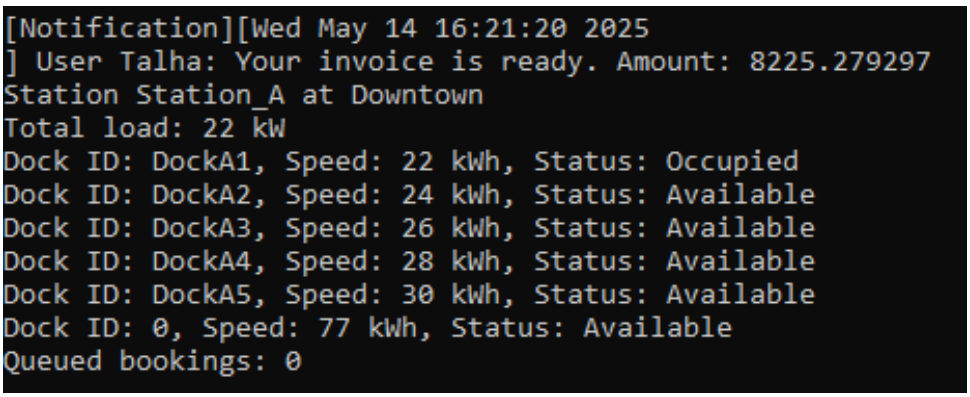

Figure 1.6: This screenshot gives notification about many things including Invoice generation, queueing process etc.

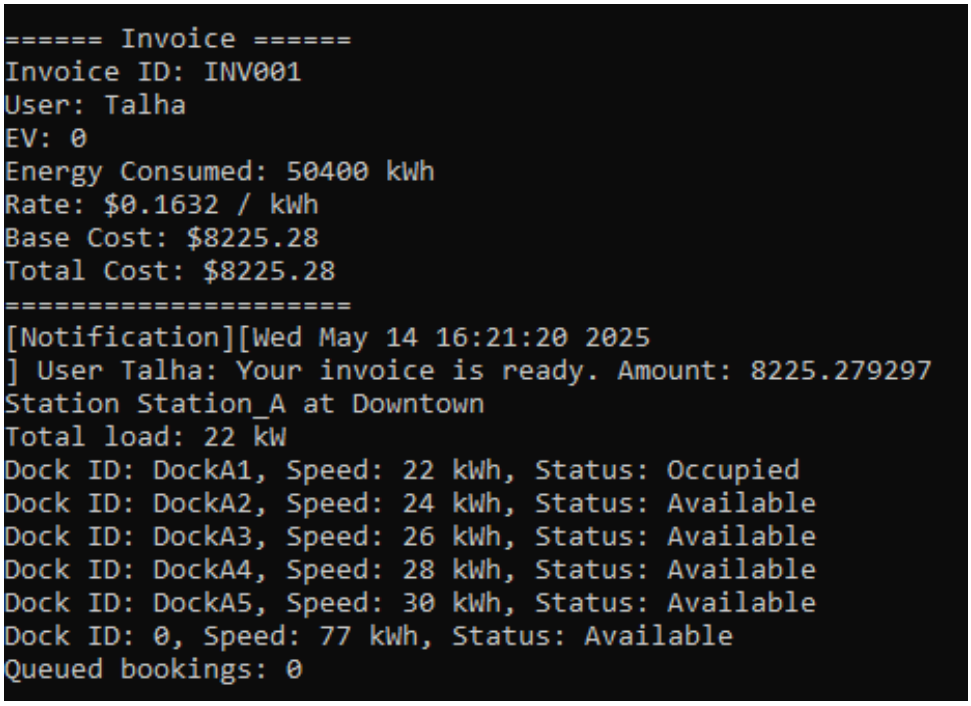

Figure 1.7: Invoice Generation Screen. This screenshot shows the generated invoice for a completed charging session. The system calculates the price based on various factors like membership, energy source, and charging speed.

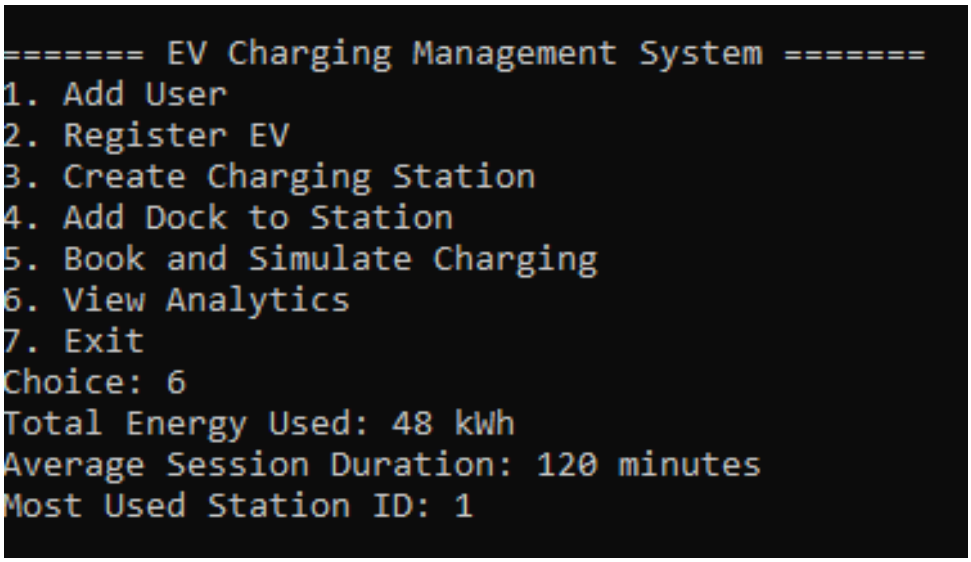

Figure 1.8: Analytics Dashboard Screen. This figure illustrates the analytics dashboard that displays key data such as energy usage, station performance, and session durations. It helps monitor system efficiency and user behavior.

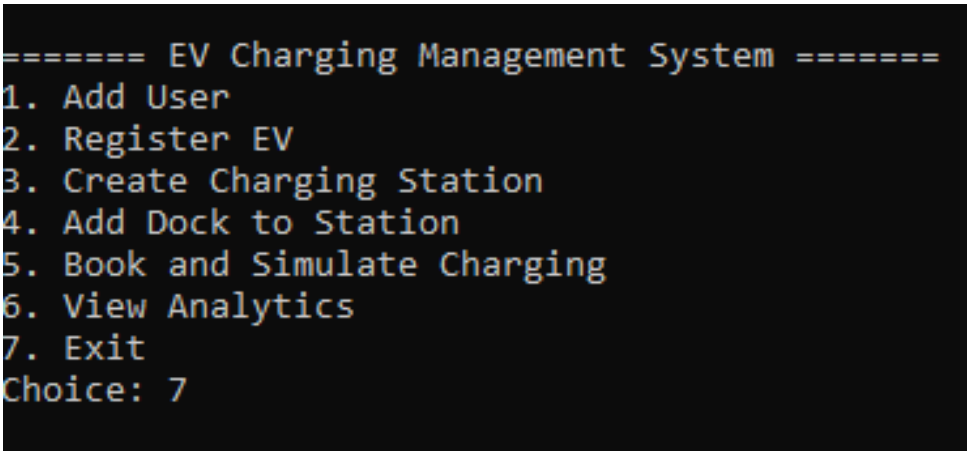

Figure 1.9: EXITS.

1.6 Challenges Faced

- **Complex Interactions**: Keeping relationships between EVs, Users, Bookings, and Stations consistent.
- **Smart Pointers**: Preventing memory leaks and handling ownership without circular dependencies.
- **Polymorphic Design**: Designing flexible classes for energy sources while keeping the system scalable.
- **Dynamic Load Balancing**: Ensuring fair energy distribution without overloads during peak hours.
- **Real-Time Operations**: Simulating a real-time experience of booking and charging.

1.7 Conclusion

This project successfully simulates a smart EV charging system, demonstrating object-oriented programming skills and core principles like encapsulation, inheritance, and polymorphism. The simulation efficiently handles multiple users, vehicles, stations, and dynamic pricing, providing an efficient model for city-wide EV charging networks.

1.8 References

- NUST CS-107 Project Files
- Lab Manuals
- C++ OOP Concepts – cpp.com
- OpenAI ChatGPT for debugging and suggestions
- Youtube.com