Technical Report: Car Rental Management System

Project Title: Car Rental Management System

Team Members:

علي محسن الحبشي 2221809256

عبدالعزيز اشرف الجازوي 2221808195

محمد المهدي اللوشة 2221810969

Course: Advanced Programming with Java

1. Overview of the System

The Car Rental Management System is a desktop application designed to streamline and automate the core operations of a car rental business. The system provides a user-friendly graphical interface for managing the company's fleet of vehicles, customer information, and all rental transactions. It enables staff to efficiently add new cars and customers to the database, track vehicle availability, process new rentals, complete returns, and generate insightful reports on business performance.

The primary objective of this system is to replace manual, paper-based processes with a centralized, digital solution. This reduces the risk of errors, improves operational efficiency, and provides management with valuable data-driven insights through its reporting features. The application is built to be intuitive, allowing users with minimal technical training to perform their daily tasks effectively.

2. Technologies and Tools Used

This project was developed using the following technologies and tools:

- **Programming Language**: Java 11+
- **IDE:** NetBeans IDE
- **Build Tool:** Apache Ant (integrated withing netBeans)
- **Database:** SQLite, alightweight, serverless database engine ideal for desktop applications
- **JDBC Driver:** sqlite-jdbc-3.36.0.3.jar for Java Database Connectivity.
- **GUI Framework:** Java Swing (AWT) for creating the graphical user interface.
- **Key Java APIs:**
- **JDBC**: For database connectivity and CRUD operations.
- **Collections Framework:** Java(ArrayList,HashMap) for dynamic data management
- **I/O Streams:** For file operations, including exporting data and logging system actions
- **Multithreading:**(SwingWorker,ScheduledExecutorService) for performing background tasks and keeping the UI responsive.

3. Key Features Implemented

The system is modularized into several key functional areas:

- **Car Management:**
  - Add new vehicles with details like make, model, year, license plate, and daily rate.
  - Update existing car information.
  - Delete cars from the fleet.
  - View all cars or filter to see only available vehicles.
  - Automatic tracking of car availability (rented vs. available).
- **Customer Management:**
  - Register new customers with their contact details.
  - Update customer information.
  - Delete customer records.
  - View a complete list of all registered customers.
- **Rental Management:**
  - Create new rental agreements by selecting a car, customer, and rental period.
  - Automatic calculation of total rental cost based on the car's daily rate and rental duration.
  - View all rental records, including active and completed rentals.
  - Mark rentals as "Completed" upon vehicle return, which automatically updates the car's availability status.
  - Delete rental records if necessary.
- **Reporting and Export:**
  - **Summary Report:** Provides a high-level overview of total cars, customers, rentals, and revenue.
  - **Car Usage Report:** Details how many times each car has been rented and the total revenue generated per vehicle.
  - **Customer Activity Report:** Shows rental history and total spending for each customer.
  - **Data Export:** Functionality to export car, customer, and rental data to CSV files for external analysis or record-keeping.
- **System Utilities:**
  - **Action Logging:** All significant actions (e.g., adding a car, creating a rental) are automatically logged to a text file for auditing and debugging.
  - **Auto-Refresh:** A configurable feature that automatically refreshes the rental data at a set interval, ensuring the displayed information is always up-to-date.

4. Sample Screenshots

Figure 1: Main Application Window This screenshot shows the main interface with the tabbed pane for navigating between Cars, Customers, Rentals, and Reports.

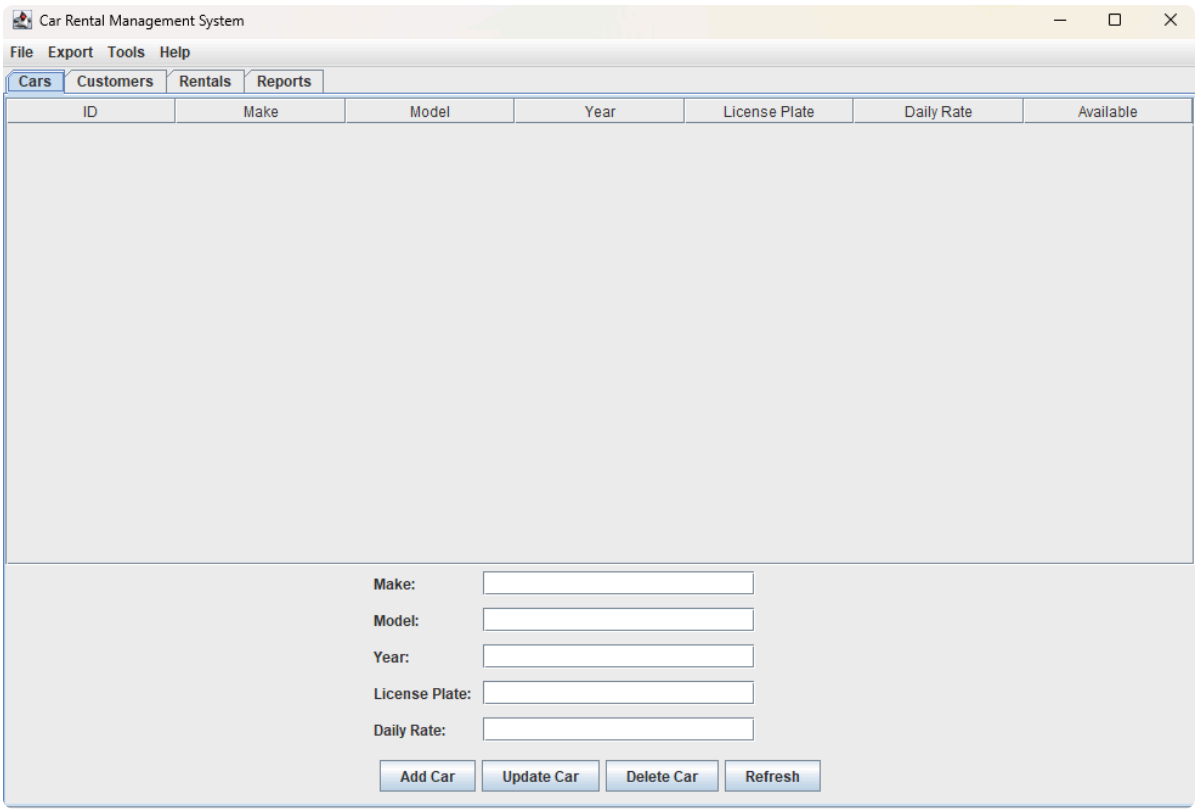


Figure 2: Car Management Panel This screenshot demonstrates the form for adding a new car and the table displaying the current car inventory.

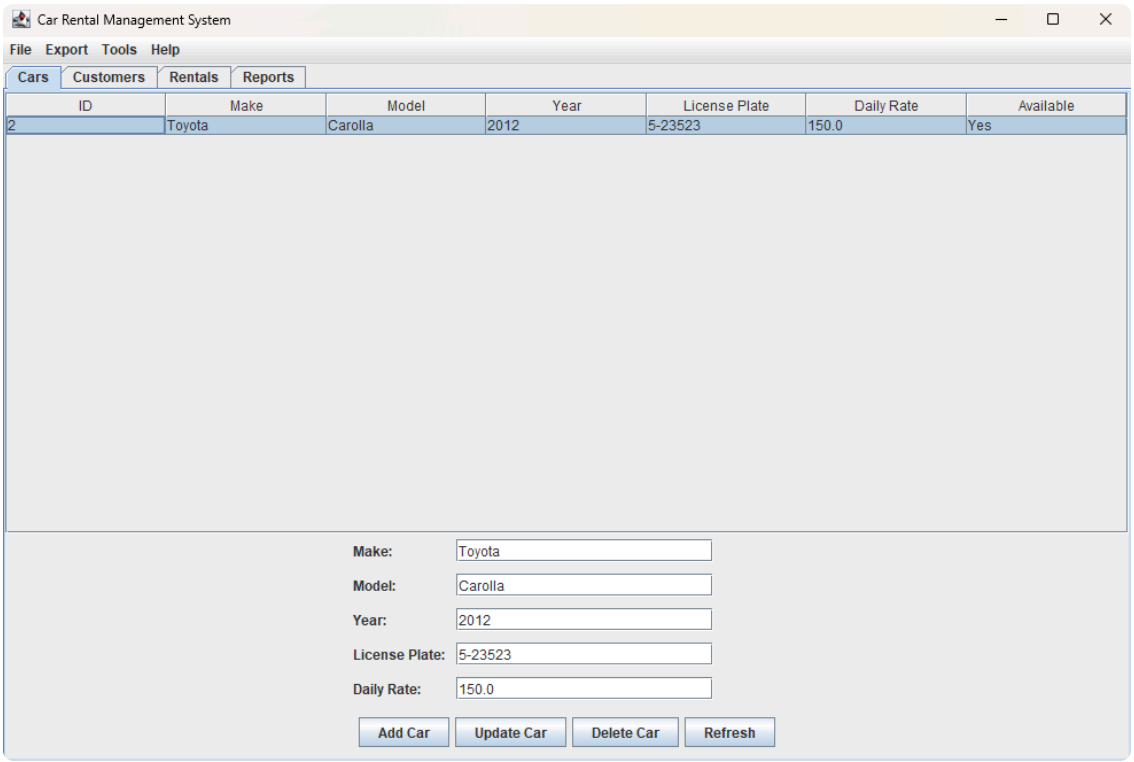


Figure 3: Creating a New Rental This screenshot shows the process of creating a new rental, selecting an available car, a customer, and the rental dates.

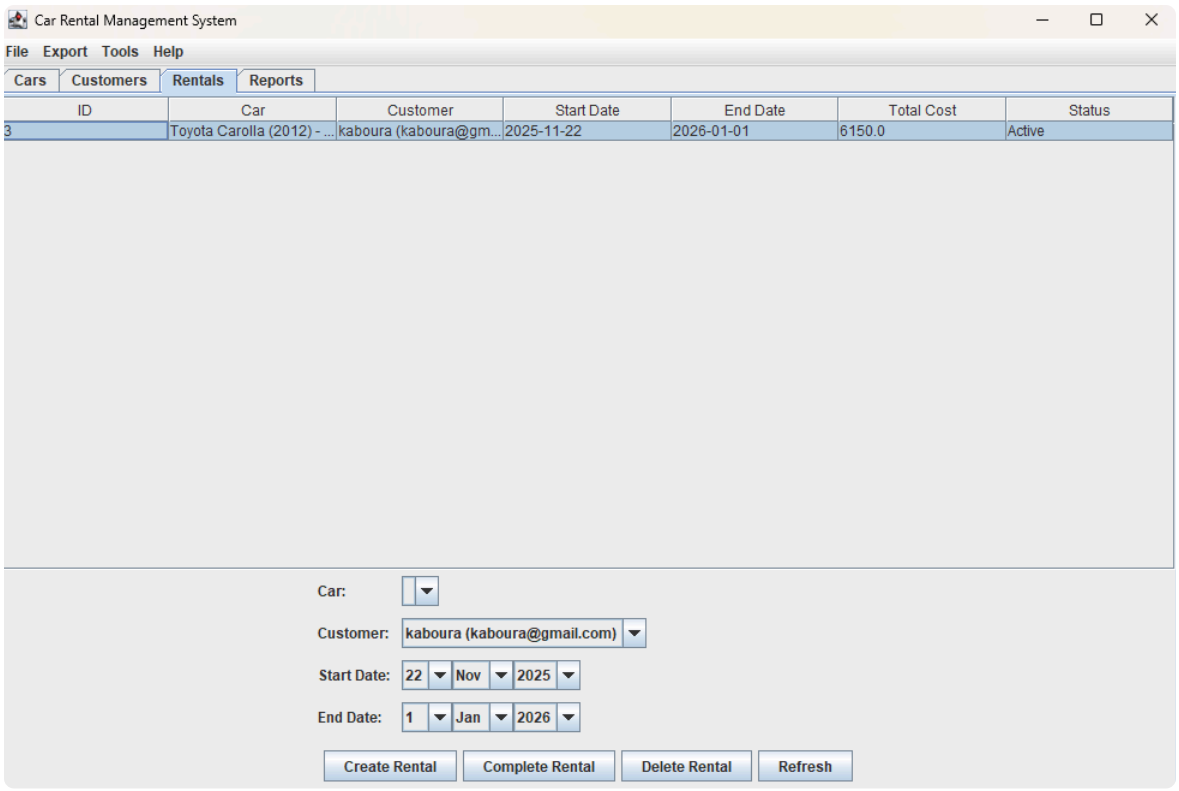
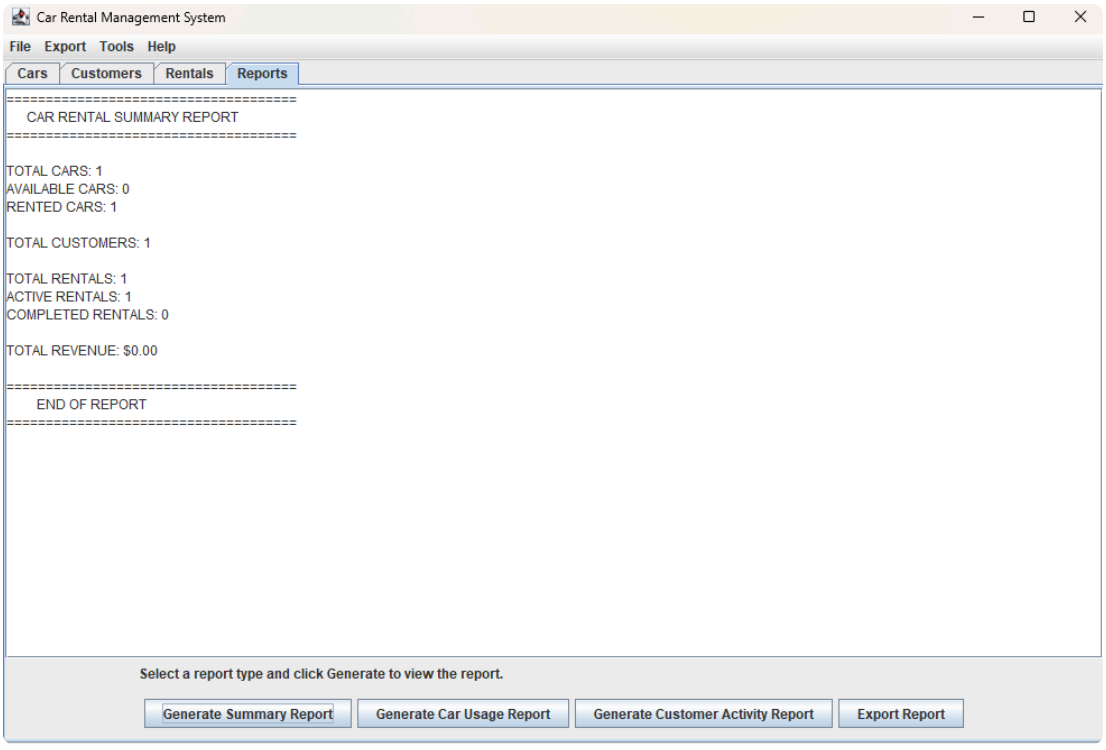


Figure 4: Generated Summary Report This screenshot displays the output of the summary report, showing key business metrics.



5. Explanation of How Each Required Topic Was Implemented

1. Exception Handling: Proper exception handling is implemented throughout the application to ensure robustness.

- **Database Operations:** All methods in the DAO (Data Access Object) classes, such as CarDAO and RentalDAO, use try-catch-finally blocks. The finally block ensures that

database resources (Connection, Statement, ResultSet) are always closed, preventing resource leaks.
- **User Input:** Service layer classes (e.g., CarService) validate user input before it reaches the DAO layer. For example, they check if a car's make or a customer's name is empty, preventing invalid data from being processed.
- **File I/O:** The FileManager class uses try-catch blocks to handle potential IOExceptions when writing to log files or exporting data, ensuring the application doesn't crash if a file is inaccessible.

2. Collections Framework: Java Collections are used extensively for dynamic data management.

- **ArrayList:** Used in all DAO classes to retrieve and store multiple records from the database before displaying them in the GUI. For example, CarDAO.getAllCars() returns an ArrayList<Car>.
- **HashMap:** Used in the ReportsPanel to efficiently aggregate data. For instance, a HashMap<Integer, Double> is used to map each car_id to its total revenue, allowing for quick calculations without repeatedly querying the database.

3. JDBC - Java Database Connectivity: JDBC is the core technology for all database interactions.

- **Database Initialization:** The DatabaseManager class uses JDBC to create the database tables (cars, customers, rentals) if they don't already exist when the application starts.
- **CRUD Operations:** The DAO pattern is fully implemented. Classes like CarDAO, CustomerDAO, and RentalDAO contain methods for Create, Read, Update, and Delete operations. We use PreparedStatement for all SQL commands to prevent SQL injection attacks and improve performance.

4. GUI Using Swing + Event Handling: The entire user interface is built using Java Swing components and follows an event-driven programming model.

- **Components:** We used JFrame for the main window, JTabbedPane for organizing modules, JTable for displaying tabular data, JPanel for layout, JButton for actions, JTextField and JTextArea for input, and JComboBox for selections.
- **Event Handling:** ActionListeners are registered on all buttons to handle user clicks. For example, the "Add Car" button has an ActionListener that triggers the addCar() method, which validates input and calls the service layer.

5. I/O Streams: I/O Streams are used for logging and data export features.

- **Logging:** The FileManager.logAction() method uses a BufferedWriter wrapped around a FileWriter to append timestamped messages to a log file (car_rental_log.txt). This provides an audit trail of system usage.
- **Data Export:** Methods like FileManager.exportCarsToCSV() use BufferedWriter to write the contents of a data collection to a CSV file in a structured format, allowing data to be opened in spreadsheet applications.

6. Multithreading: Multithreading is implemented to enhance user experience and enable background processing.

- **Background Tasks:** All database operations (loading, adding, updating data) are executed in a background thread using the BackgroundTask class, which extends SwingWorker. This prevents the GUI from freezing during long-running tasks. The SwingWorker handles the communication between the background thread and the Event Dispatch Thread (EDT) safely.
- **Auto-Refresh Service:** The AutoRefreshService uses a ScheduledExecutorService to run a task at a fixed interval. This task can automatically refresh the data in a JTable, ensuring the user sees near real-time information without manual intervention.

6. Task Distribution Among Team Members

Team Member	Responsibilities
علي محسن الحبشي	- Project Lead & Database Design- Implemented DatabaseManager, DAO classes (CarDAO, CustomerDAO, RentalDAO)- Implemented Service layer logic- Integrated JDBC and handled all database connectivity.
عبدالعزيز اشرف الجازوي	- GUI Development- Designed and implemented all UI panels (CarManagementPanel, CustomerManagementPanel, etc.)- Implemented event handling and user input validation- Worked on layout and user experience
محمد المهدي اللوشة	- Advanced Features & Reporting- Implemented multithreading (BackgroundTask, AutoRefreshService)- Developed the reporting module and FileManager for I/O operations- Conducted testing and debugging- Prepared technical documentation

7. Challenges and Lessons Learned

- **Challenge:** Designing a logical and efficient database schema that properly linked cars, customers, and rentals.

    - **Lesson Learned:** The importance of planning relationships and foreign keys early on. A well-designed schema is the foundation of a stable application and simplifies data retrieval significantly.
- **Challenge:** Keeping the GUI responsive while performing time-consuming database queries.
    - **Lesson Learned:** The power of multithreading in desktop applications. Using SwingWorker was crucial for providing a smooth user experience. We learned that any long-running task must be moved off the Event Dispatch Thread.
- **Challenge:** Implementing comprehensive error handling that was both robust for the application and informative for the user.
    - **Lesson Learned:** Exception handling is more than just try-catch. It involves validating user input proactively, providing clear error messages, and logging technical details for debugging purposes.
- **Challenge:** Coordinating different parts of the system (GUI, business logic, data access) to work together seamlessly.
    - **Lesson Learned:** The value of a layered architecture (UI -> Service -> DAO). This separation of concerns made the code more organized, easier to test, and simpler for multiple team members to work on simultaneously without conflicts.

Technical Report: Car Rental Management System

Project Title: Car Rental Management System

Team Members:

علي محسن الحبشي 2221809256

عبدالعزيز اشرف الجازوي 2221808195

محمد المهدي اللوشة 2221810969

Course: Advanced Programming with Java

1. Overview of the System

The Car Rental Management System is a desktop application designed to streamline and automate the core operations of a car rental business. The system provides a user-friendly graphical interface for managing the company's fleet of vehicles, customer information, and all rental transactions. It enables staff to efficiently add new cars and customers to the database, track vehicle availability, process new rentals, complete returns, and generate insightful reports on business performance.

The primary objective of this system is to replace manual, paper-based processes with a centralized, digital solution. This reduces the risk of errors, improves operational efficiency, and provides management with valuable data-driven insights through its reporting features. The application is built to be intuitive, allowing users with minimal technical training to perform their daily tasks effectively.

2. Technologies and Tools Used

This project was developed using the following technologies and tools:

- **Programming Language**: Java 11+
- **IDE:** NetBeans IDE
- **Build Tool:** Apache Ant (integrated withing netBeans)
- **Database:** SQLite, alightweight, serverless database engine ideal for desktop applications
- **JDBC Driver:** sqlite-jdbc-3.36.0.3.jar for Java Database Connectivity.
- **GUI Framework:** Java Swing (AWT) for creating the graphical user interface.
- **Key Java APIs:**
- **JDBC**: For database connectivity and CRUD operations.
- **Collections Framework:** Java(ArrayList,HashMap) for dynamic data management
- **I/O Streams:** For file operations, including exporting data and logging system actions
- **Multithreading:**(SwingWorker,ScheduledExecutorService) for performing background tasks and keeping the UI responsive.

3. Key Features Implemented

The system is modularized into several key functional areas:

- **Car Management:**
  - Add new vehicles with details like make, model, year, license plate, and daily rate.
  - Update existing car information.
  - Delete cars from the fleet.
  - View all cars or filter to see only available vehicles.
  - Automatic tracking of car availability (rented vs. available).
- **Customer Management:**
  - Register new customers with their contact details.
  - Update customer information.
  - Delete customer records.
  - View a complete list of all registered customers.
- **Rental Management:**
  - Create new rental agreements by selecting a car, customer, and rental period.
  - Automatic calculation of total rental cost based on the car's daily rate and rental duration.
  - View all rental records, including active and completed rentals.
  - Mark rentals as "Completed" upon vehicle return, which automatically updates the car's availability status.
  - Delete rental records if necessary.
- **Reporting and Export:**
  - **Summary Report:** Provides a high-level overview of total cars, customers, rentals, and revenue.
  - **Car Usage Report:** Details how many times each car has been rented and the total revenue generated per vehicle.
  - **Customer Activity Report:** Shows rental history and total spending for each customer.
  - **Data Export:** Functionality to export car, customer, and rental data to CSV files for external analysis or record-keeping.
- **System Utilities:**
  - **Action Logging:** All significant actions (e.g., adding a car, creating a rental) are automatically logged to a text file for auditing and debugging.
  - **Auto-Refresh:** A configurable feature that automatically refreshes the rental data at a set interval, ensuring the displayed information is always up-to-date.

4. Sample Screenshots

Figure 1: Main Application Window This screenshot shows the main interface with the tabbed pane for navigating between Cars, Customers, Rentals, and Reports.

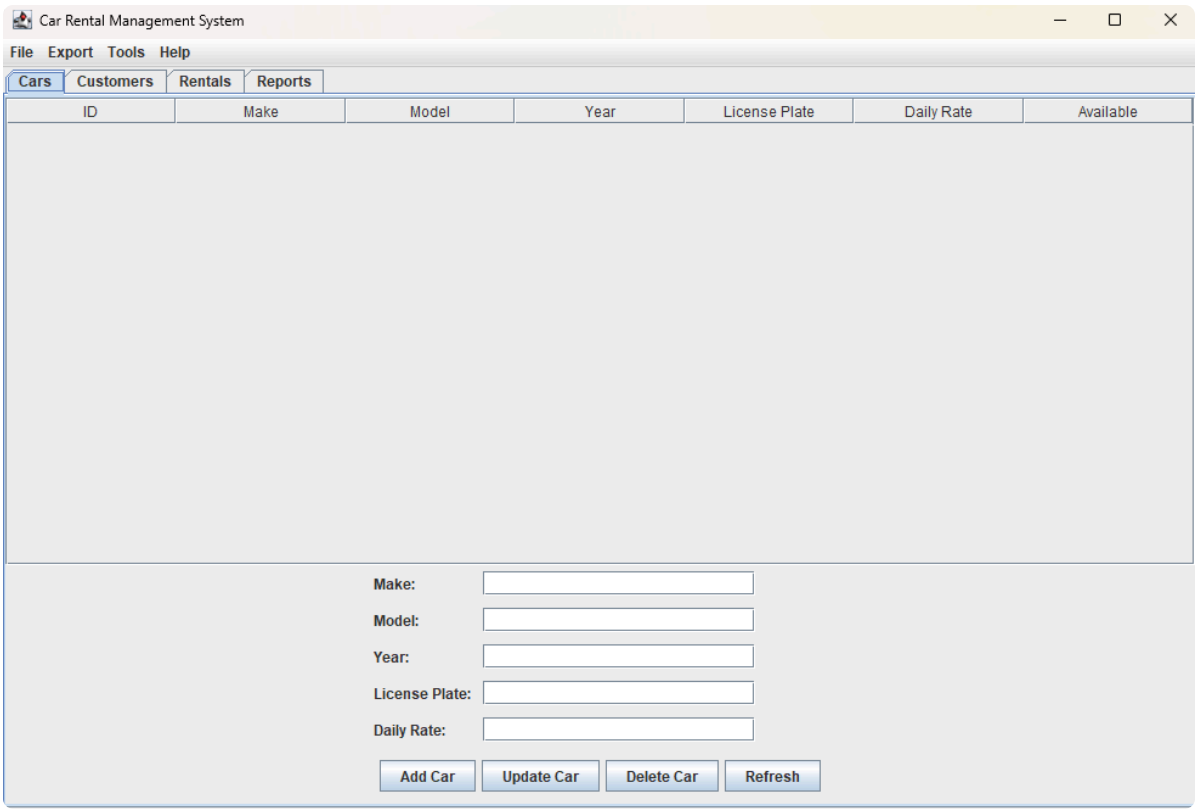


Figure 2: Car Management Panel This screenshot demonstrates the form for adding a new car and the table displaying the current car inventory.

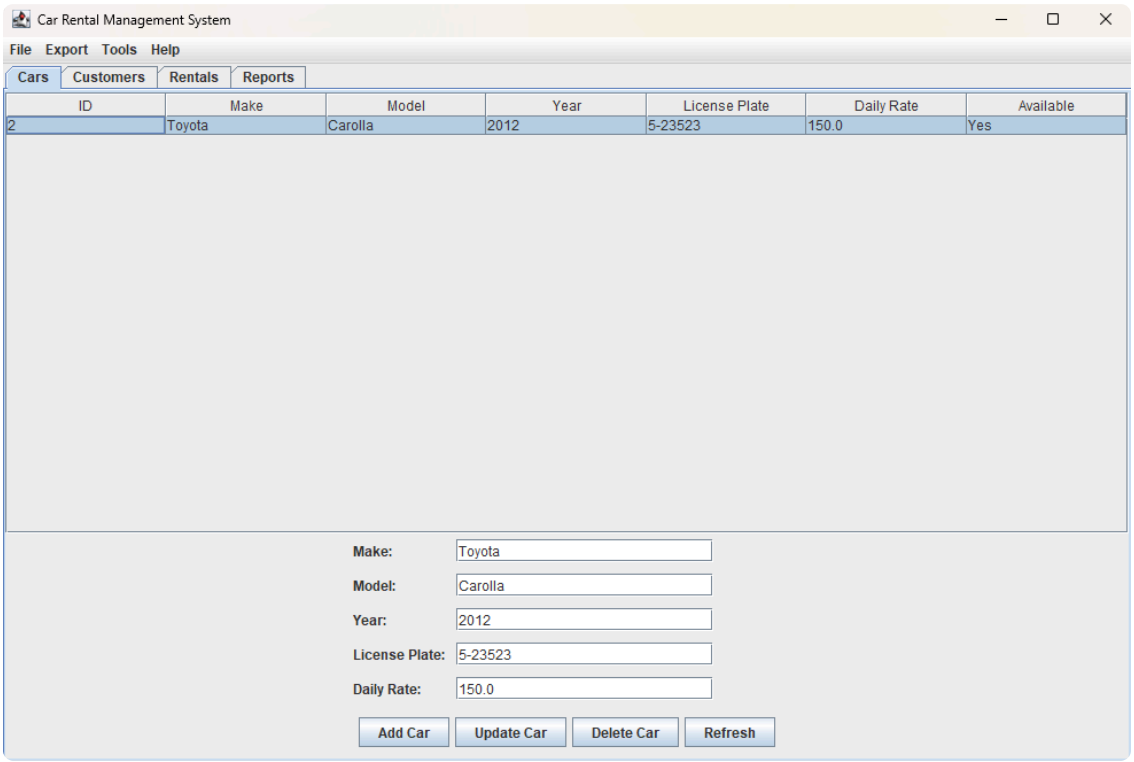


Figure 3: Creating a New Rental This screenshot shows the process of creating a new rental, selecting an available car, a customer, and the rental dates.

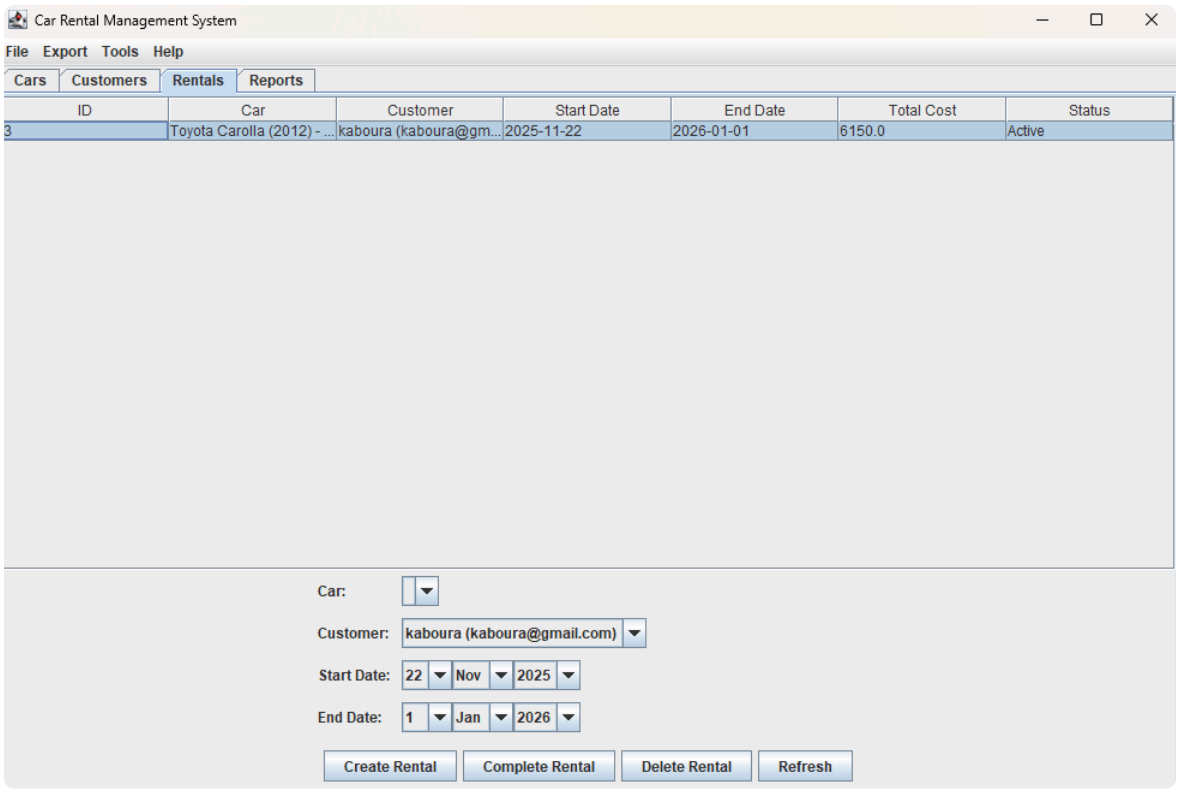
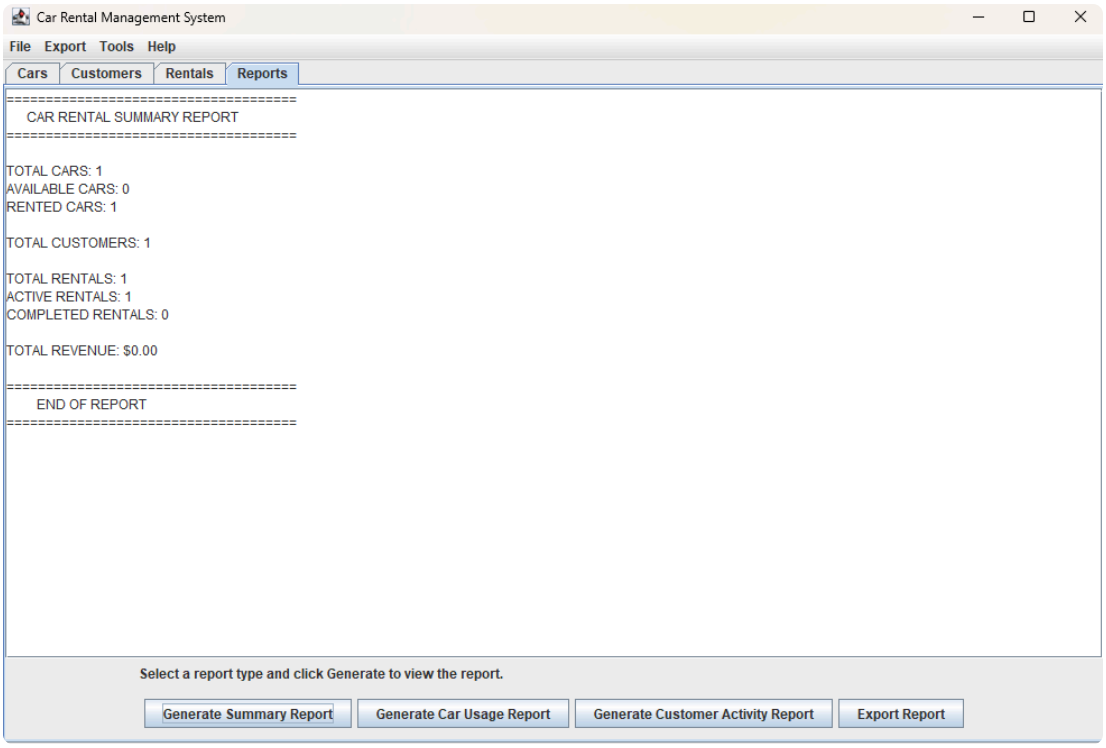


Figure 4: Generated Summary Report This screenshot displays the output of the summary report, showing key business metrics.



5. Explanation of How Each Required Topic Was Implemented

1. Exception Handling: Proper exception handling is implemented throughout the application to ensure robustness.

- **Database Operations:** All methods in the DAO (Data Access Object) classes, such as CarDAO and RentalDAO, use try-catch-finally blocks. The finally block ensures that database resources (Connection, Statement, ResultSet) are always closed, preventing

resource leaks.

```
// Example from CarDAO.java
try (Connection conn = DatabaseManager.getConnection();
     PreparedStatement pstmt = conn.prepareStatement(sql)) {
    // ... database operations ...
} catch (SQLException e) {
    System.err.println("Error adding car: " + e.getMessage());
    return false;
}
```

- **User Input:** Service layer classes (e.g., CarService) validate user input before it reaches the DAO layer. For example, they check if a car's make or a customer's name is empty, preventing invalid data from being processed.
- **File I/O:** The FileManager class uses try-catch blocks to handle potential IOExceptions when writing to log files or exporting data, ensuring the application doesn't crash if a file is inaccessible.

2. Collections Framework: Java Collections are used extensively for dynamic data management.

- **ArrayList:** Used in all DAO classes to retrieve and store multiple records from the database before displaying them in the GUI. For example, CarDAO.getAllCars() returns an ArrayList<Car>.

```
// Example from CarDAO.java
List<Car> cars = new ArrayList<>();
String sql = "SELECT * FROM cars";
// ... populate the list from the database ...
return cars;
```

- **HashMap:** Used in the ReportsPanel to efficiently aggregate data. For instance, a HashMap<Integer, Double> is used to map each car_id to its total revenue, allowing for quick calculations without repeatedly querying the database.

3. JDBC - Java Database Connectivity: JDBC is the core technology for all database interactions.

- **Database Initialization:** The DatabaseManager class uses JDBC to create the database tables (cars, customers, rentals) if they don't already exist when the application starts.
- **CRUD Operations:** The DAO pattern is fully implemented. Classes like CarDAO, CustomerDAO, and RentalDAO contain methods for Create, Read, Update, and Delete operations. We use PreparedStatement for all SQL commands to prevent SQL injection attacks and improve performance.

```
// Example from RentalDAO.java
String sql = "INSERT INTO rentals(car_id, customer_id, start_date, end_date, total_cost, status) VALUES(?, ?, ?, ?, ?, ?)";
try (Connection conn = DatabaseManager.getConnection();
     PreparedStatement pstmt = conn.prepareStatement(sql)) {
    pstmt.setInt(1, rental.getCarId());
    pstmt.setInt(2, rental.getCustomerId());
```

```
    // ... set other parameters ...
    pstmt.executeUpdate();
}
```

4. GUI Using Swing + Event Handling: The entire user interface is built using Java Swing components and follows an event-driven programming model.

- **Components:** We used JFrame for the main window, JTabbedPane for organizing modules, JTable for displaying tabular data, JPanel for layout, JButton for actions, JTextField and JTextArea for input, and JComboBox for selections.
- **Event Handling:** ActionListeners are registered on all buttons to handle user clicks. For example, the "Add Car" button has an ActionListener that triggers the addCar() method, which validates input and calls the service layer.

```
// Example from CarManagementPanel.java
addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        addCar(); // This method is called when the button is clicked
    }
});
```

5. I/O Streams: I/O Streams are used for logging and data export features.

- **Logging:** The FileManager.logAction() method uses a BufferedWriter wrapped around a FileWriter to append timestamped messages to a log file (car_rental_log.txt). This provides an audit trail of system usage.
- **Data Export:** Methods like FileManager.exportCarsToCSV() use BufferedWriter to write the contents of a data collection to a CSV file in a structured format, allowing data to be opened in spreadsheet applications.

6. Multithreading: Multithreading is implemented to enhance user experience and enable background processing.

- **Background Tasks:** All database operations (loading, adding, updating data) are executed in a background thread using the BackgroundTask class, which extends SwingWorker. This prevents the GUI from freezing during long-running tasks. The SwingWorker handles the communication between the background thread and the Event Dispatch Thread (EDT) safely.

```
// Example from CarManagementPanel.java
new BackgroundTask<List<Car>, Void>("Load Cars") {
    @Override
    protected List<Car> executeInBackground() throws Exception {
        return carService.getAllCars(); // Runs on a background thread
    }
    @Override
    protected void onSuccess(List<Car> cars) {
        // Update the UI on the EDT
    }
```

11. **}.execute();**

- **Auto-Refresh Service:** The AutoRefreshService uses a ScheduledExecutorService to run a task at a fixed interval. This task can automatically refresh the data in a JTable, ensuring the user sees near real-time information without manual intervention.

6. Task Distribution Among Team Members

Team Member	Responsibilities
علي محسن الحبشي	- Project Lead & Database Design- Implemented DatabaseManager, DAO classes (CarDAO, CustomerDAO, RentalDAO)- Implemented Service layer logic- Integrated JDBC and handled all database connectivity.
عبدالعزيز اشرف الجازوي	- GUI Development- Designed and implemented all UI panels (CarManagementPanel, CustomerManagementPanel, etc.)- Implemented event handling and user input validation- Worked on layout and user experience
محمد المهدي اللوشة	- Advanced Features & Reporting- Implemented multithreading (BackgroundTask, AutoRefreshService)- Developed the reporting module and FileManager for I/O operations- Conducted testing and debugging- Prepared technical documentation

7. Challenges and Lessons Learned

- **Challenge:** Designing a logical and efficient database schema that properly linked cars, customers, and rentals.
  - **Lesson Learned:** The importance of planning relationships and foreign keys early on. A well-designed schema is the foundation of a stable application and simplifies data retrieval significantly.
- **Challenge:** Keeping the GUI responsive while performing time-consuming database queries.
  - **Lesson Learned:** The power of multithreading in desktop applications. Using SwingWorker was crucial for providing a smooth user experience. We learned that any

long-running task must be moved off the Event Dispatch Thread.

- **Challenge:** Implementing comprehensive error handling that was both robust for the application and informative for the user.
  - **Lesson Learned:** Exception handling is more than just try-catch. It involves validating user input proactively, providing clear error messages, and logging technical details for debugging purposes.
- **Challenge:** Coordinating different parts of the system (GUI, business logic, data access) to work together seamlessly.
  - **Lesson Learned:** The value of a layered architecture (UI -> Service -> DAO). This separation of concerns made the code more organized, easier to test, and simpler for multiple team members to work on simultaneously without conflicts.