CAR RENTAL SYSTEM

Daniel Heřt

Contents

INTRODUCTION

1.1 Purpose

This program was created as an outcome of the class NPRG041 at the Charles University in Prague at the faculty of Mathematics and Physics.

The purpose of this program is to facilitate the daily operations of a car rental service by providing a comprehensive and efficient system for managing car rentals, customer information, and vehicle inventory. This program aims to manage the workflow for employees, ensuring accurate tracking and timely management of rented, available, and under-maintenance vehicles. It also provides automated functionalities for contract generation and overdue alerts, enhancing both operational efficiency and customer service.

1.2 Scope

This program covers all critical aspects of car rental management, including:

Inventory Management: Tracking the status of cars (available, rented, under maintenance, reserved) and updating their status based on rental activities.

Customer Management: Storing and managing customer information, allowing for easy access and updates.

Contract Generation: Automatically generating rental contracts with necessary details, simplifying the rental process.

Time Management: Monitoring rental periods and providing alerts for overdue vehicles.

User Role Management: Differentiating between regular users and admin users, ensuring appropriate access control and functionality restrictions.

Data Storage: Managing data through text files, ensuring easy manipulation and retrieval of information.

The program is designed to interact with users via a console interface, making it user-friendly and accessible without the need for external libraries.

1.3 Audience

This documentation is intended for programmers who will be working with the code. It provides a overview of the system's architecture, installation, configuration, and usage instructions. By following this documentation, programmers will be able to efficiently understand, modify, and enhance the codebase to support and extend the car rental service's operational capabilities.

SYSTEM OVERVIEW

2.1 Architecture

The system operates entirely on the client's computer without requiring internet access or connection to any external systems. It is a straightforward, local application without any specific architectural patterns. The system comprises two main components:

Code: The main program logic is implemented in C++.

Data Storage: Text files are used to store all relevant data, including car inventory, customer information, rental records, and more. The program reads from and writes to these text files as needed.

2.2 Technologies Used

The program is developed in C++ and utilizes several standard libraries from the C++ Standard Library to handle various functionalities:

`#include <fstream>`: For file handling operations.

`#include <iostream>`: For input and output stream operations.

`#include <sstream>`: For string stream operations.

`#include <ctime>`: For handling date and time.

`#include <chrono>`: For precise timekeeping.

`#include <filesystem>`: For interacting with the file system.

`#include <string>`: For string manipulation.

`#include <vector>`: For using dynamic arrays.

`#include <stdexcept>`: For standard exceptions.

`#include <ostream>`: For output stream operations.

`#include <algorithm>`: For standard algorithms (e.g., sorting, searching).

`#include <cstring>`: For C-style string manipulation.

`#include <limits>`: For handling numeric limits.

`#include <climits>`: For defining the properties of fundamental types.

The development environment used for this project is Visual Studio, configured as a CMake project. The program has been tested using both Windows Subsystem for Linux (WSL) and native Windows environments, ensuring compatibility across these platforms.

2.3 Dependencies

The program does not rely on any external dependencies, making it lightweight and easy to deploy. However, there are some specific requirements and configurations:

CMake: The project requires CMake version greater than 3.12 for building the project.

C++ Standard: The code is written to comply with the C++20 standard, so the compiler used must support this version of the language.

There are no other dependencies or special environment configurations needed to run the program. This simplicity in setup ensures that the program can be easily compiled and executed on any compatible system with the required tools.

INSTALLATION

3.1 Prerequisites

The program is designed to run on any operating system that has a compatible C++ compiler. While it has been tested on both Linux and Windows, it should function correctly on any system meeting the following requirements:

Operating System: Linux, Windows, or any other OS with a compatible C++ compiler.

Disk Space: Minimal (600 kB for the program).

For development purposes, you will need additional tools:

Development Tools: Visual Studio (for Windows) or any other IDE/text editor for C++ development.

CMake: Version greater than 3.12.

C++ Compiler: Must support the C++20 standard.

3.2 Installation Steps

For Usage:

Download and Unzip:

Download the program package (zip file) from the provided source.

Unzip the file to your desired directory.

Run the Program:

Navigate to the unzipped directory.

Execute the program. This can be done by running an executable file provided in the package.

USAGE

4.1 User Interface

The program features a console-based user interface, as shown in the provided screenshot.

The interface comprises three main sections:

Header: Displays a decorative header with a title indicating the current section of the program.

Content Area: Displays the main content, which changes based on the user's selections and current tasks. This area shows different menus and prompts for input depending on the selected operation.

Footer: Displays status messages, such as notifications about delayed contracts or prompts for further actions.

Users interact with the program by selecting numbered options from the displayed menus. The interface operates in two primary modes:

Browsing Mode: Users navigate through menus by entering the corresponding number of their choice (e.g., 1 for "Cars", 2 for "Customers").

Input Mode: When adding new records or performing specific operations, users are prompted to enter the required information (e.g., license plate number, customer phone number, car make).

4.2 User Authentication

User authentication is handled via a simple login system that assigns a status to each user, either admin or non-admin. This status is represented in the user file, where `1` indicates an admin and `0` indicates a non-admin. Upon login, the program checks the user's status and customizes the menu accordingly:

Admin Users: Have access to additional functionalities such as adding new users and viewing archived contracts.

Non-Admin Users: Have a restricted menu with basic functionalities required for daily operations.

4.3 Core Functionality

The core functionality of the program is organized into several key areas, each with specific menu options and operations. Below is a detailed description of each area and its functionalities:

Cars:

Show available cars

Show rented cars

Show cars in service

Show permanently unavailable cars

Admin Only:

Add new car

Move a car (between different status categories)

Sub-menus for adding or moving cars:

Add an available car

Add a serviced car

Add a rented car

Add a permanently unavailable car

Move a car to available, rented, in service, or permanently unavailable status

Customers:

Show customers

Add a customer

Contracts:

Show active contracts

New contract: Create a new rental contract by entering required details.

End active contract: Conclude a rental contract and update the car's status.

Admin Only:

Show archived contracts: View contracts that have been completed and archived.

User Management (Admin Only):

Add a user: Add a new user with a specific status (admin or non-admin).

Each operation involves navigating through the menus and entering necessary information when prompted. The program ensures that all actions are recorded and stored in the appropriate text files, maintaining up-to-date records of car statuses, customer details, and contract information.

By following these menus and prompts, users can efficiently manage the car rental service's daily operations, ensuring a smooth and organized workflow.

Upon initial setup, the program comes with a default user account pre-configured:

Username: admin

Password: 1234

This default user has administrative privileges, allowing full access to all functionalities of the program, including user management and administrative tasks. It is recommended to log in using this account first and then create additional users as needed. For security purposes, consider changing the password of the default admin account after your first login.

HEADERS

5.1 Headers Description

The program is organized into several header files, each serving a specific purpose. Below is a detailed description of each header file:

Objects.h

- **Purpose**: This header defines the core objects used in the program, such as Car, Customer, and User. Each class represents an entity within the car rental system and includes methods for manipulating the properties of these entities.
- **Key Classes and Functions**:
  - **Car**: Represents a car with attributes like make, model, year, color, license plate, motorization, gearbox, seats, and cost per hour.
  - **Customer**: Represents a customer with attributes like name, surname, email address, phone number, and address.
  - **User**: Represents a user with attributes like username, password, and admin status.

FileHandler.h

- **Purpose**: This header provides functionality for reading from and writing to text files. It includes classes and methods for managing the storage and retrieval of car, customer, and user data.
- **Key Classes and Functions**:
  - **FileReader**: Contains methods for retrieving data from files, such as lists of cars, customers, users, active contracts, and archived contracts.
  - **FileWriter**: Contains methods for writing data to files, including adding and deleting records, creating contracts, and moving files between directories.

ConsoleController.h

- **Purpose**: This header manages the console input and output operations, providing a user interface for interacting with the car rental system. It includes methods for displaying menus, capturing user input, and showing various data.
- **Key Classes and Functions**:
  - **ConsoleController**: Contains methods for displaying menus, lists of cars, customers, contracts, and handling user inputs.

System.h

- **Purpose**: This header manages the overall functionality of the car rental system. It integrates the console controller, file handler, and core objects to provide a cohesive system.
- **Key Classes and Functions**:
    - **System**: Manages the main loop of the program, user authentication, and high-level operations such as adding cars, customers, users, and managing contracts.

CarRentalSystem.h

- **Purpose**: This header serves as an entry point for the car rental system, including necessary headers and initializing the system.
- **Key Classes and Functions**:
    - **CarRentalSystem**: Standard system files and project-specific includes.

5.1.1 Headers member functions description

Objects.h:

Car class:

- Car(const Properties& props)
    - The Car constructor initializes a Car object using a vector of properties. It expects the vector to contain exactly nine elements representing the car's make, model, year, color, license plate, motorization, gearbox, number of seats, and cost per hour. If the vector does not contain exactly nine elements, the constructor throws an std::invalid_argument exception. The relevant properties are then parsed and assigned to the member variables of the Car class.

- Properties GetProperties() const
    - The GetProperties function returns a Properties vector containing all the attributes of the Car object. The attributes include make, model, year, color, license plate, motorization, gearbox, number of seats, and cost per hour, with numerical values converted to strings. This function encapsulates the car's data into a single vector for easy access and manipulation.

- • std::string GetMake() const
    - The GetMake function returns the make of the car as a string. This allows external access to the Make attribute of the Car object.

- std::string GetModel() const
    - The GetModel function returns the model of the car as a string. This allows external access to the Model attribute of the Car object.

- int GetYear() const
    - The GetYear function returns the year of manufacture of the car as an integer. This allows external access to the Year attribute of the Car object.

- std::string GetColor() const
    - The GetColor function returns the color of the car as a string. This allows external access to the Color attribute of the Car object.

- std::string GetLicencePlate() const
    - The GetLicencePlate function returns the license plate of the car as a string. This allows external access to the LicencePlate attribute of the Car object.

- std::string GetMotorization() const
    - The GetMotorization function returns the motorization type of the car as a string. This allows external access to the Motorization attribute of the Car object.
- std::string GetGearbox() const
    - The GetGearbox function returns the gearbox type of the car as a string. This allows external access to the Gearbox attribute of the Car object.

- int GetSeats() const
    - The GetSeats function returns the number of seats in the car as an integer. This allows external access to the Seats attribute of the Car object.

- int GetCostPerHour() const

    - The GetCostPerHour function returns the cost per hour to rent the car as an integer. This allows external access to the CostPerHour attribute of the Car object.

Customer class:

- Customer(const Properties& props)
    - The Customer constructor initializes a Customer object using a vector of properties. It expects the vector to contain exactly five elements representing the customer's name, surname, email address, phone number, and address. If the vector does not contain exactly five elements, the constructor throws an std::invalid_argument exception. The relevant properties are then parsed and assigned to the member variables of the Customer class.

- Properties GetProperties() const
    - The GetProperties function returns a Properties vector containing all the attributes of the Customer object. The attributes include name, surname, email address, phone number, and address. This function encapsulates the customer's data into a single vector for easy access and manipulation.

- std::string GetName() const
    - The GetName function returns the name of the customer as a string. This allows external access to the Name attribute of the Customer object.

- std::string GetSurname() const
    - The GetSurname function returns the surname of the customer as a string. This allows external access to the Surname attribute of the Customer object.

- std::string GetEmailAdress() const
    - The GetEmailAdress function returns the email address of the customer as a string. This allows external access to the EmailAdress attribute of the Customer object.

- std::string GetPhone() const

    - The GetPhone function returns the phone number of the customer as a string. This allows external access to the Phone attribute of the Customer object.

- std::string GetAdress() const
    - The GetAdress function returns the address of the customer as a string. This allows external access to the Adress attribute of the Customer object.

User class:

- User(const Properties& props)
    - The User constructor initializes a User object using a vector of properties. It expects the vector to contain exactly three elements representing the user's username, password, and admin status. If the vector does not contain exactly three elements, the constructor throws an std::invalid_argument exception. The relevant properties are then parsed and assigned to the member variables of the User class, with the admin status being set based on the value of the third property.

- Properties GetProperties() const
    - The GetProperties function returns a Properties vector containing all the attributes of the User object. The attributes include username, password, and admin status. This function encapsulates the user's data into a single vector for easy access and manipulation.

- bool GetAdminStatus() const
    - The GetAdminStatus function returns a boolean indicating whether the user is an admin. This allows external access to the Admin attribute of the User object.

FileHandler.h:

FileReader class:

- StorageVector GetServicedCars()
    - The GetServicedCars function retrieves all the cars currently being serviced by reading the properties from the corresponding text file. It returns a StorageVector containing the properties of all serviced cars.

- StorageVector GetRentedCars()
    - The GetRentedCars function retrieves all the cars currently rented to customers by reading the properties from the corresponding text file. It returns a StorageVector containing the properties of all rented cars.

- StorageVector GetAvailableCars()
    - The GetAvailableCars function retrieves all the cars currently available for rent by reading the properties from the corresponding text file. It returns a StorageVector containing the properties of all available cars.

- StorageVector GetPermanentlyUnavailableCars()
    - The GetPermanentlyUnavailableCars function retrieves all the cars that are permanently unavailable by reading the properties from the corresponding text file. It returns a StorageVector containing the properties of all permanently unavailable cars.

- StorageVector GetCustomers()
    - The GetCustomers function retrieves information about all customers by reading the properties from the corresponding text file. It returns a StorageVector containing the properties of all customers.

- StorageVector GetUsers()
    - The GetUsers function retrieves information about all users by reading the properties from the corresponding text file. It returns a StorageVector containing the properties of all users.

- StorageVector GetActiveContracs()
    - The GetActiveContracs function retrieves the names of files with active contracts by reading the directory containing these files. It returns a StorageVector containing the names of files with active contracts.

- StorageVector GetArchivedContracs()
    - The GetArchivedContracs function retrieves the names of files with archived contracts by reading the directory containing these files. It

returns a StorageVector containing the names of files with archived contracts.

- Properties FindRecordByToken(const std::string& filename, const std::string& token)
    - The FindRecordByToken function searches for a record in a given file where one of the properties matches the provided token. If a matching record is found, it returns a Properties vector containing the entire record. If no match is found, it returns an empty Properties vector.

- StorageVector GetInfo(const std::string& what)
    - The GetInfo function is a private helper function that retrieves information from a specified file. It reads each line of the file, splits it by a delimiter, and stores the resulting tokens in a StorageVector. This function is used to gather data about cars, customers, and users.

- StorageVector GetNamesOfFiles(const std::string& directoryPath)
    - The GetNamesOfFiles function is a private helper function that retrieves the names of files in a specified directory. It returns a StorageVector containing the names of the files without their extensions. This function is used to list active and archived contract files.

FileWriter class:

- AddServicedCar(const Car& car)
    - The AddServicedCar function adds a car to the serviced list by appending its properties to the corresponding text file. It uses the AddInfo private helper function to write the car's properties.

- AddRentedCar(const Car& car)
    - The AddRentedCar function adds a car to the rented list by appending its properties to the corresponding text file. It uses the AddInfo private helper function to write the car's properties.

- AddAvailableCar(const Car& car)
    - The AddAvailableCar function adds a car to the available list by appending its properties to the corresponding text file. It uses the AddInfo private helper function to write the car's properties.

- AddPermanentlyUnavailableCar(const Car& car)
    - The AddPermanentlyUnavailableCar function adds a car to the permanently unavailable list by appending its properties to the corresponding text file. It uses the AddInfo private helper function to write the car's properties.

- AddCustomer(const Customer& customer)
    - The AddCustomer function adds a customer to the customer list by appending their properties to the corresponding text file. It uses the AddInfo private helper function to write the customer's properties.

- AddUser(const User& user)
    - The AddUser function adds a user to the user list by appending their properties to the corresponding text file. It uses the AddInfo private helper function to write the user's properties.

- DeleteRecordInFile(const std::string& filename, const std::string& token)
    - The DeleteRecordInFile function deletes a record from a specified file by searching for a line that contains the given token. If the token is found, the corresponding line is removed from the file.

- CreateNewContract(const Customer& customer, const Car& car, const std::chrono::system_clock::time_point& dueDate)
    - The CreateNewContract function creates a new contract between a customer and a car by generating a text file with the contract details, including customer information, car information, due date, rental duration, and total price.

- MoveFileToFolder(const std::string& sourceFolder, const std::string& destinationFolder, const std::string& fileName)
    - The MoveFileToFolder function moves a file from one folder to another by renaming its path. This operation is performed only if the source file exists and is a regular file.

- AddInfo(const Properties& props, const std::string& what)
    - The AddInfo function is a private helper function that appends a set of properties to a specified file. It writes the properties as a delimited string, with each property separated by a delimiter.

- TimePointToString(const std::chrono::system_clock::time_point& timePoint, const std::string& delimiter)

    - The TimePointToString function converts a given time point to a string using a specified delimiter. The string format includes the year, month, day, hour, and minute, separated by the provided delimiter.

ConsoleController.h:

ConsoleController class:

- DisplayMenu(std::ostream& os, const std::vectorstd::string& options, bool inMainMenu, int numberOfDelayedContracts)
    - The DisplayMenu function displays a menu with a list of options. It shows the header, each menu option, and the footer. The number of delayed contracts is displayed if applicable.

- DisplayAvailableCars(std::ostream& os)
    - The DisplayAvailableCars function displays the list of available cars by retrieving their properties from the corresponding file and displaying the content.

- DisplayRentedCars(std::ostream& os)
    - The DisplayRentedCars function displays the list of rented cars by retrieving their properties from the corresponding file and displaying the content.

- DisplayServicedCars(std::ostream& os)
    - The DisplayServicedCars function displays the list of cars currently under service by retrieving their properties from the corresponding file and displaying the content.

- DisplayPermanentlyUnavailableCars(std::ostream& os)
    - The DisplayPermanentlyUnavailableCars function displays the list of permanently unavailable cars by retrieving their properties from the corresponding file and displaying the content.

- DisplayCustomers(std::ostream& os)
    - The DisplayCustomers function displays the list of customers by retrieving their properties from the corresponding file and displaying the content.

- DisplayActiveContracts(std::ostream& os)
    - The DisplayActiveContracts function displays the list of active contracts by retrieving their names from the corresponding directory and displaying the content.

- DisplayArchivedContracts(std::ostream& os)
    - The DisplayArchivedContracts function displays the list of archived contracts by retrieving their names from the corresponding directory and displaying the content.

- DisplayGoodByeMessage(std::ostream& os)
    - The DisplayGoodByeMessage function displays a goodbye message with a simple ASCII art figure and a farewell message.

- DisplayAddingMessage(std::ostream& os, std::string namesOfColumns)
    - The DisplayAddingMessage function displays a message prompting the user to add a new record. It shows the required columns for the new record.

- DisplayLogInMenu(std::ostream& os)
    - The DisplayLogInMenu function displays the login menu with a welcome message.

- ClearConsole()
    - The ClearConsole function clears the console screen. It uses system-specific commands to clear the console on Windows and POSIX systems.

- GetIntInput(std::ostream& os, std::istream& is, int min, int max)
    - The GetIntInput function gets an integer input from the user within a specified range. It validates the input and handles invalid entries by prompting the user again.

- GetStringInput(std::istream& is)
    - The GetStringInput function gets a string input from the user.

- DisplayContent(std::ostream& os, const std::vector<std::vectorstd::string>& content, bool displayFileNames)
    - The DisplayContent function displays the content of a list of records. It formats the content for display, ensuring proper spacing and alignment.

- DisplayContentHeader(std::ostream& os, const std::string& title)
    - The DisplayContentHeader function displays the header for a content section, centering the title.

- DisplayHeader(std::ostream& os)
    - The DisplayHeader function displays the header of the console output, including a decorative border and the name of the system.

- DisplayFooter(std::ostream& os, bool inMainMenu, int numberOfDelayedContracts, bool showExitOption)
    - The DisplayFooter function displays the footer of the console output. It shows different messages based on whether the user is in the main menu or a sub-menu, and it can display the number of delayed contracts if applicable.

- DisplayMenuOption(const std::string& option, std::ostream& os)
    - The DisplayMenuOption function displays a single menu option, centering it within the console output.

- CalculateSpacing(size_t lengthOfContent)
    - The CalculateSpacing function calculates the spacing required to center content within the console output. It ensures the content is properly aligned and spaced.

System.h:

System class:

- System(std::ostream& output, std::istream& input)
    - The System constructor initializes a new System object, setting the output and input streams for displaying messages and receiving user input. It also initializes a default user.

- void Run()
    - The Run function starts the system's main loop, allowing the user to interact with the system through various menu options. It handles user login, displays menus, processes user input, and executes corresponding actions based on the selected options. The loop continues until the user chooses to exit.

- bool LogIn()
    - The LogIn function handles user authentication by prompting for a username and password. It checks the entered credentials against the stored user data. If the credentials match, the user is logged in, and the function returns true; otherwise, it prompts the user to try again or cancel.

- void AddAvailableCar() const
    - The AddAvailableCar function prompts the user to enter the properties of a new available car and adds it to the corresponding list by saving the properties to a file.

- void AddServicedCar() const
    - The AddServicedCar function prompts the user to enter the properties of a new car currently in service and adds it to the corresponding list by saving the properties to a file.

- void AddRentedCar() const
    - The AddRentedCar function prompts the user to enter the properties of a new rented car and adds it to the corresponding list by saving the properties to a file.

- void AddPermanentlyUnavailableCar() const
    - The AddPermanentlyUnavailableCar function prompts the user to enter the properties of a new permanently unavailable car and adds it to the corresponding list by saving the properties to a file.

- void AddCustomer() const
    - The AddCustomer function prompts the user to enter the properties of a new customer and adds them to the customer list by saving the properties to a file.

- void AddUser() const
    - The AddUser function prompts the user to enter the properties of a new user and adds them to the user list by saving the properties to a file.

- void CarsMenu(const std::vectorstd::string& chosenCarMenuOptions) const

    - The CarsMenu function displays car menu options based on the user's role (admin or regular). It processes the selected option and executes the corresponding action, such as displaying available cars, adding a new car, or moving a car.

- void AddNewCarMenu() const
    - The AddNewCarMenu function displays the menu for adding a new car and processes the selected option to add an available, serviced, rented, or permanently unavailable car.

- void CustomersMenu() const
    - The CustomersMenu function displays customer menu options and processes the selected option to display customers or add a new customer.

- void CreateNewContract() const
    - The CreateNewContract function creates a new rental contract by prompting the user to choose a car and a customer and enter the contract's due date. It then saves the contract details to a file and updates the car's status.

- void ArchiveContract() const
    - The ArchiveContract function archives an existing contract by moving it from the active contracts directory to the archived contracts directory. It also updates the car's status to available.

- void MoveACar() const
    - The MoveACar function moves a car from one status to another (e.g., from available to rented). It prompts the user to choose the car and the destination status, updates the car's properties, and saves the changes to the corresponding files.

- int CheckContractDates() const
    - The CheckContractDates function checks the dates of all active contracts and returns the number of overdue contracts by comparing the due dates with the current date and time.

- Properties GetPropsFromInput(const char* namesOfProps) const
    - The GetPropsFromInput function prompts the user to enter properties for a new record based on the provided names of properties and returns the entered properties as a vector.

- std::vector<int> GetContractProperties() const
    - The GetContractProperties function prompts the user to enter the properties for a new contract's due date and returns the entered properties as a vector of integers.

- int CountWords(const char* str) const
    - The CountWords function counts the number of words in a given string by counting the spaces and returns the word count.

- int CalculateRentDurationHours(const std::chrono::system_clock::time_point& currentTime, const std::chrono::system_clock::time_point& returnTime) const
    - The CalculateRentDurationHours function calculates the duration of a car rent in hours by computing the difference between the return time and the current time.

- std::chrono::system_clock::time_point CreateManualTimePoint(int year, int month, int day, int hour, int min) const
    - The CreateManualTimePoint function creates a time point based on manual input of year, month, day, hour, and

minute, converting the input into a time point that can be used for time calculations.

5.2 Headers Linking

The headers in the program are linked together in a hierarchical manner, with dependencies clearly defined. Below is an overview of how these headers are interconnected:

1. **CarRentalSystem.h**: The entry point of the program, which includes System.h to start the car rental system.
2. **System.h**:
    - Includes ConsoleController.h to handle user interface operations.
    - Manages the main loop, user authentication, and primary functionalities.
3. **ConsoleController.h**:
    - Includes FileHandler.h to access file reading and writing functionalities.
    - Manages the display of menus, user inputs, and data representation.
4. **FileHandler.h**:
    - Includes Objects.h to manipulate the core objects (Car, Customer, User).
    - Provides functions for reading and writing data related to cars, customers, users, and contracts.
5. **Objects.h**:
    - Defines the core objects used throughout the program.
    - Provides constructors and methods for manipulating these objects.

This organization ensures that the program is modular, with clear separation of concerns, making it easier to maintain and extend. Each header plays a specific role and interacts with others through well-defined interfaces.

MAIN ALGORITHM

6.1 Description

The main algorithm of the car rental system manages the interaction between the user and the system. It handles user authentication, menu navigation, and execution of core functionalities like adding cars, managing customers, creating contracts, and checking overdue contracts. The algorithm ensures that the system operates smoothly, guiding the user through various tasks in a structured manner.

Steps

Initialization:

The program starts by initializing a System object, passing the standard output (cout) and input (cin) streams for user interaction.

The Run method of the System object is called to begin the main loop.

User Authentication:

The LogIn method prompts the user to enter their username and password.

The method checks the credentials against stored user data.

If the credentials are valid, the user is logged in; otherwise, they are prompted to try again or cancel.

Main Loop:

The system displays the main menu options based on the user's role (admin or regular user).

The user selects an option by entering the corresponding number.

The system processes the selected option and performs the appropriate action.

Menu Navigation:

Cars Menu: Displays options for managing cars (e.g., showing available cars, adding a new car, moving a car).

Customers Menu: Displays options for managing customers (e.g., showing customers, adding a new customer).

Contracts: Allows the user to create a new contract, show active contracts, or archive a contract.

Action Execution:

Based on the selected menu option, the system executes the corresponding action:

Add Car: Prompts the user to enter car details and adds the car to the appropriate list.

Add Customer: Prompts the user to enter customer details and adds the customer to the list.

Create Contract: Prompts the user to choose a car and a customer, enter the due date, and create a rental contract.

Archive Contract: Moves a contract from the active to the archived list and updates the car's status.

Move Car: Moves a car from one status to another (e.g., available to rented).

Exit:

The user can exit the main loop by selecting the exit option from the main menu, which displays a goodbye message and terminates the program.

Key Functions

System::Run(): Starts the main loop and handles user interactions and menu navigation.

System::LogIn(): Manages user authentication.

System::AddAvailableCar(), System::AddServicedCar(), System::AddRentedCar(), System::AddPermanentlyUnavailableCar(): Functions for adding cars to different lists.

System::AddCustomer(): Adds a new customer to the system.

System::AddUser(): Adds a new user to the system.

System::CreateNewContract(): Creates a new rental contract.

System::ArchiveContract(): Archives an existing contract.

System::MoveACar(): Moves a car between different status lists.

Example

Creating a New Contract:

The user selects the option to create a new contract from the main menu.

The system displays the list of available cars.

The user selects a car by entering its license plate number.

The system displays the list of customers.

The user selects a customer by entering their phone number.

The system prompts the user to enter the due date for the contract.

The contract details are saved to a new file, and the car's status is updated to rented.

The system moves the car from the available list to the rented list.

This structured approach ensures that the system is easy to navigate and use, providing a seamless experience for managing car rentals.

DATABASE SCHEMA

7.1 Overview

The car rental system utilizes a simple file-based storage mechanism to manage and store data. It stores data in plain text files organized into directories. Each type of entity (e.g., cars, customers, users, contracts) has its own file or set of files where records are stored in a structured format.

7.2 File Structure

Source Files Directory (SOURCEFILES):

This directory contains subdirectories and files for storing various types of data related to the car rental system.

Subdirectories and Files:

/Cars/:

available.txt: Stores data for cars that are available for rent.

rented.txt: Stores data for cars that are currently rented.

repair_shop.txt: Stores data for cars that are under service.

permanently_unavailable.txt: Stores data for cars that are permanently unavailable.

/Users/:

users.txt: Stores data for system users.

/Customers/:

customers.txt: Stores data for customers.

/Customers/Contracts/Active/:

Contains files for active rental contracts.

/Customers/Contracts/Archived/:

Contains files for archived rental contracts.

7.3 File Format

Each file stores records in a structured format where each record is represented by a line of text. Fields within each record are separated by a delimiter (e.g., #). Below are the details for each file format:

Cars Files:

Fields: Make, Model, Year, Color, License Plate, Motorization, Gearbox, Seats, Cost Per Hour

Example Record: Toyota#Camry#2020#Red#ABC123#Gasoline 1.5#Automatic#5#100

Users File (users.txt):

Fields: Username, Password, Admin Status

Example Record: admin#1234#1

Customers File (customers.txt):

Fields: Name, Surname, Email Address, Phone, Address

Example Record: Jan#Novak#jan.novak@example.com#123456789#Parizska

Contracts Files:

Example File Name: Novak_ABC123_2023_12_31_14_00.txt

Example File Content:

Contract Details:

Customer: Jan Novak

Car: Toyota Camry, License Plate: ABC123

Due Date: 31. 12. 2023

Due Hours: 14:00

Hours of rent: 48

Total price: 4800 Kc

Signed on: 01. 12. 2023

Customers signature:.....

Representative signature:.....

7.4 Data Handling

Reading Data:

The system uses the FileReader class to read data from the text files. It reads each line, splits the line into fields using the delimiter, and stores the fields in a Properties vector.

Writing Data:

The system uses the FileWriter class to write data to the text files. It converts the fields of a record into a delimited string and appends it to the appropriate file.

Updating Data:

To update a record, the system reads all records into memory, modifies the relevant record, and writes all records back to the file.

Deleting Data:

To delete a record, the system reads all records into memory, removes the relevant record, and writes the remaining records back to the file.