Report on Vehicle Rental Management System

Introduction

Goal of the Coursework

The aim of this course is to create a car rental system, this system facilitates the Vehicle Rental Management System and helped me improve my basic software development skills, including object-oriented programming, data validation and design patterns.

Topic Description

The topic of this coursework is the development of a Vehicle Rental Management System. This system facilitates the management of user information, vehicle selection, and appointment scheduling for a vehicle rental business. The system ensures data integrity through validation checks and supports data storage and retrieval using CSV files.

What is your application?

It is a Python application written to simplify user registration, schedule car rental appointments and manage vehicle data. The application includes:

-Register new users with validated personal information.

-Schedule rental appointments for various types of vehicles, ensuring that the dates and times are valid.

-Save and load user and appointment data to and from CSV files, ensuring data persistence.

How to run the program?

To run the program, follow these steps:

-Ensure Python is installed: If you don't have Python on your computer, download Python.

-Save the code: Copy the provided code and save it in a file named vehicle\_rental.py.

-Open a terminal or command prompt: Navigate to the directory where vehicle\_rental.py is saved.

-Run the program: Execute the command python vehicle\_rental.py to start the program.

How to use the program?

User Registration:

- We need the user's name, surname, date of birth and driver's license date.

Vehicle Selection:

-The program will ask you to choose a vehicle type (Car or Motorcycle).

-Select the brand and model from the provided options. If an invalid option is chosen, the program will prompt you to try again.

Appointment Scheduling:

- enter the date and time of the appointment

Data Persistence:

-After successfully adding a user and scheduling an appointment, the program will save the data to CSV files (users.csv and appointments.csv).

Analysis and Explanation of the Program's Implementation

Vehicle Rental Management System is designed to manage user records, plan vehicle rental appointments and ensure data continuity. This section provides an analysis of how the implementation of the program meets these goals and functional requirements

User Management

Objective: Register new users with validated personal information.

The Person and User classes handle user registration and validation.

Code Snippet: User Registration and Validation

class Person:  
 def \_\_init\_\_(self, first\_name, last\_name, birth\_date):  
 self.first\_name = first\_name  
 self.last\_name = last\_name  
 self.birth\_date = self.\_validate\_birth\_date(birth\_date)  
  
 def \_validate\_birth\_date(self, birth\_date):  
 birth\_date\_dt = datetime.datetime.strptime(birth\_date, '%Y-%m-%d')  
 if birth\_date\_dt > datetime.datetime.now():  
 raise ValueError("Birth date cannot be in the future.")  
 return birth\_date\_dt  
  
 def \_\_str\_\_(self):  
 return f"{self.first\_name} {self.last\_name}, Birth Date: {self.birth\_date.strftime('%Y-%m-%d')}"

The User class extends the Person class and includes license date validation.

class User(Person):  
 def \_\_init\_\_(self, first\_name, last\_name, birth\_date, license\_date):  
 super().\_\_init\_\_(first\_name, last\_name, birth\_date)  
 self.license\_date = self.\_validate\_license\_date(license\_date)  
  
 def \_validate\_license\_date(self, license\_date):  
 license\_date\_dt = datetime.datetime.strptime(license\_date, '%Y-%m-%d')  
 if license\_date\_dt > datetime.datetime.now():  
 raise ValueError("License date cannot be in the future.")  
 if license\_date\_dt < self.birth\_date:  
 raise ValueError("License date cannot be before birth date.")  
 return license\_date\_dt  
  
 def \_\_str\_\_(self):  
 return f"{super().\_\_str\_\_()}, License Date: {self.license\_date.strftime('%Y-%m-%d')}"

These classes ensure that users are registered with valid birth and license dates, preventing future dates and ensuring logical consistency (e.g., the license date cannot be before the birth date).

Appointment Scheduling

Objective: Schedule rental appointments with validated dates and times.

The class is responsible for creating and validating appointments.Appointment

Code Snippet: Appointment Creation and Validation

class Appointment:  
 def \_\_init\_\_(self, user, vehicle\_type, appointment\_date, appointment\_time):  
 self.user = user  
 self.vehicle\_type = vehicle\_type  
 self.appointment\_datetime = self.\_validate\_appointment\_datetime(appointment\_date, appointment\_time)  
  
 def \_validate\_appointment\_datetime(self, appointment\_date, appointment\_time):  
 appointment\_date\_dt = datetime.datetime.strptime(appointment\_date, '%Y-%m-%d')  
 appointment\_time\_dt = datetime.datetime.strptime(appointment\_time, '%H:%M')  
 appointment\_datetime = datetime.datetime.combine(appointment\_date\_dt.date(), appointment\_time\_dt.time())  
 if appointment\_datetime < datetime.datetime.now():  
 raise ValueError("Appointment date cannot be in the past.")  
 return appointment\_datetime  
  
 def \_\_str\_\_(self):  
 return (  
 f"Appointment: {self.appointment\_datetime.strftime('%Y-%m-%d %H:%M')} - "  
 f"{self.user.first\_name} {self.user.last\_name} - Vehicle Type: {self.vehicle\_type}"  
 )

The class ensures that appointments are only scheduled for future dates and times, preventing the scheduling of past appointments.Appointment

Data Persistence

Objective: Save and load user and appointment data to/from CSV files.

The class, implemented as a Singleton, manages the list of users and appointments. It provides methods to save and load data to and from CSV files.VehicleRental

Code Snippet: Data Persistence

@singleton  
class VehicleRental:  
 def \_\_init\_\_(self):  
 self.users = []  
 self.appointments = []  
  
 def add\_user(self, first\_name, last\_name, birth\_date, license\_date):  
 user = UserFactory.create\_user(first\_name, last\_name, birth\_date, license\_date)  
 self.users.append(user)  
 return user  
  
 def add\_appointment(self, user, vehicle\_type, appointment\_date, appointment\_time):  
 appointment = Appointment(user, vehicle\_type, appointment\_date, appointment\_time)  
 self.appointments.append(appointment)  
 return appointment  
  
 def save\_data\_to\_csv(self, users\_file="users.csv", appointments\_file="appointments.csv"):  
 with open(users\_file, 'w', newline='') as ufile:  
 writer = csv.writer(ufile)  
 writer.writerow(["First Name", "Last Name", "Birth Date", "License Date"])  
 for user in self.users:  
 writer.writerow([user.first\_name, user.last\_name, user.birth\_date.strftime('%Y-%m-%d'), user.license\_date.strftime('%Y-%m-%d')])  
  
 with open(appointments\_file, 'w', newline='') as afile:  
 writer = csv.writer(afile)  
 writer.writerow(["First Name", "Last Name", "Vehicle Type", "Appointment DateTime"])  
 for appointment in self.appointments:  
 writer.writerow([appointment.user.first\_name, appointment.user.last\_name, appointment.vehicle\_type, appointment.appointment\_datetime.strftime('%Y-%m-%d %H:%M')])  
  
 def load\_data\_from\_csv(self, users\_file="users.csv", appointments\_file="appointments.csv"):  
 self.users.clear()  
 self.appointments.clear()  
  
 with open(users\_file, 'r') as ufile:  
 reader = csv.DictReader(ufile)  
 for row in reader:  
 self.add\_user(row["First Name"], row["Last Name"], row["Birth Date"], row["License Date"])  
  
 with open(appointments\_file, 'r') as afile:  
 reader = csv.DictReader(afile)  
 for row in reader:  
 user = next((u for u in self.users if u.first\_name == row["First Name"] and u.last\_name == row["Last Name"]), None)  
 if user:  
 self.add\_appointment(user, row["Vehicle Type"], row["Appointment DateTime"].split()[0], row["Appointment DateTime"].split()[1])

These methods ensure that user and appointment data are accurately saved and can be reloaded, maintaining the consistency and integrity of the data across different sessions.

User Interface

Objective: Provide a user-friendly command-line interface for interacting with the system.

The function serves as the entry point of the program, interacting with the user through the command line. It collects input for user and appointment details and handles any validation errors.main

Code Snippet: User Interaction

def main():  
 system = VehicleRental()  
  
 first\_name = input("First Name: ")  
 last\_name = input("Last Name: ")  
  
 while True:  
 try:  
 birth\_date = input("Birth Date (YYYY-MM-DD): ")  
 license\_date = input("License Date (YYYY-MM-DD): ")  
 user = system.add\_user(first\_name, last\_name, birth\_date, license\_date)  
 break  
 except ValueError as ve:  
 print(f"Invalid input: {ve}")  
  
 cars = {"Toyota": ["Corolla", "Camry", "RAV4"], "Honda": ["Civic", "Accord", "CR-V"], "Ford": ["Focus", "Mustang", "Explorer"]}  
 motorcycles = {"Yamaha": ["MT-07", "YZF-R6", "FZ-10"], "Honda": ["CBR500R", "Africa Twin", "CB500X"], "Kawasaki": ["Ninja 400", "Z650", "Versys 650"]}  
 vehicle\_type = None  
  
 while True:  
 choice = input("Select vehicle type (1: Car, 2: Motorcycle): ")  
 if choice == "1":  
 brand = input(f"Select car brand ({', '.join(cars.keys())}): ")  
 if brand in cars:  
 model = input(f"Select model ({', '.join(cars[brand])}): ")  
 if model in cars[brand]:  
 vehicle\_type = f"{brand} {model}"  
 break  
 else:  
 print("Invalid model. Please try again.")  
 else:  
 print("Invalid brand. Please try again.")  
 elif choice == "2":  
 brand = input(f"Select motorcycle brand ({', '.join(motorcycles.keys())}): ")  
 if brand in motorcycles:  
 model = input(f"Select model ({', '.join(motorcycles[brand])}): ")  
 if model in motorcycles[brand]:  
 vehicle\_type = f"{brand} {model}"  
 break  
 else:  
 print("Invalid model. Please try again.")  
 else:  
 print("Invalid brand. Please try again.")  
 else:  
 print("Invalid choice. Please try again.")  
  
 while True:  
 try:  
 appointment\_date = input("Appointment Date (YYYY-MM-DD): ")  
 appointment\_time = input("Appointment Time (HH:MM): ")  
 appointment = system.add\_appointment(user, vehicle\_type, appointment\_date, appointment\_time)  
 break  
 except ValueError as ve:  
 print(f"Invalid input: {ve}")  
  
 system.save\_data\_to\_csv()  
 print("User and appointment data successfully saved.")

The function ensures that users interact with the system in a step-by-step manner, providing clear prompts and handling any validation errors gracefully.main

By implementing these features, the Vehicle Rental Management System effectively meets the defined objectives and functional requirements, providing a reliable and user-friendly solution for managing vehicle rentals.

Results

Successful User and Appointment Management: The Vehicle Rental Management System effectively manages user registrations and appointment scheduling with validated inputs, ensuring data integrity and logical consistency.

Data Persistence Achieved: The system successfully saves and loads user and appointment data to and from CSV files, providing data persistence across different sessions.

User-Friendly Interface: The command-line interface guides users through the process of entering their details and scheduling appointments, with clear prompts and error handling for invalid inputs.

Challenges Faced: During implementation, challenges included ensuring robust date validation to prevent future dates for birth and license dates, as well as handling user input errors gracefully in the command-line interface.

Design Patterns Utilized: The use of Singleton and Factory design patterns contributed to a clean, maintainable codebase, but also required careful consideration to ensure correct implementation and avoid potential pitfalls such as multiple instance creation.

Conclusions

Summary of Key Findings and Outcomes

Effective User and Appointment Management: The system successfully manages user records and rental appointments with robust validation, preventing logical inconsistencies. This allows users to register and book appointments with confidence.

Robust Data Persistence: Data is stored securely by saving and loading into CSV files. This method prevents data from being lost and provides easy access, which is very practical for real-world use.

User-Friendly Design: The command line interface makes the system accessible to everyone by giving clear instructions to users and handling errors gracefully. In this way, users with limited technical knowledge

Achievements

Validated Input Handling: Robust verification process ensures that all user information and appointment dates are in a logical and accurate format.

Single Instance Management: Using the singleton scheme ensures that users and appointments are managed by a single instance, preventing data inconsistencies and duplication. This allows the system to operate more efficiently.

Modular and Maintainable Code: The use of design patterns like Singleton and Factory promotes a clean and maintainable codebase, making future extensions and maintenance more straightforward.

Future Prospects

Graphical User Interface (GUI): In the future, we may consider developing a GUI using frameworks such as Tkinter or PyQt to make our application more intuitive and visually appealing. This can make users have an easier and more enjoyable experience.

Database Integration: We may plan to replace the CSV file storage we are currently using with a relational database such as SQLite or MySQL. This change can increase the scalability of the application by providing a system that is more efficient in data processing and supports complex queries.

Advanced Features: We can add some additional features to make car rental management more comprehensive. For example, features such as tracking vehicle availability, authenticating users, and sending email notifications for appointments can be included in the app.

Web-Based Application: We may consider turning our application into a web-based system using frameworks such as Flask or Django. In this way, users can provide remote access and the system can appeal to a wider audience by offering multi-user support.