Car Rental System

A Project in Object-Oriented Design and Principled Architecture

Elvin Musayev & Eltun Jalilli
Faculty of Information and Computer Technologies

The Mandate: A Challenge in Principled Software Design

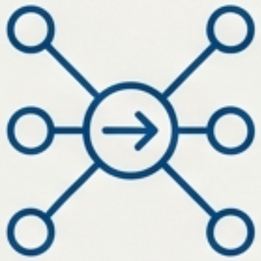
Our objective was to design and implement a Python application for a Car Rental System, demonstrating mastery over fundamental and advanced software design principles.

OOP Principles

Abstraction, Encapsulation, Inheritance, and Polymorphism.

SOLID Principles

Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion.

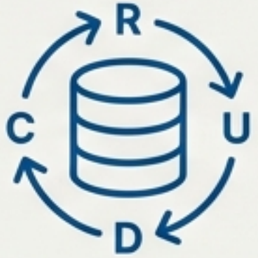


GRASP Principles

Controller, Creator, High Cohesion, and Low Coupling.

CUPID Principles

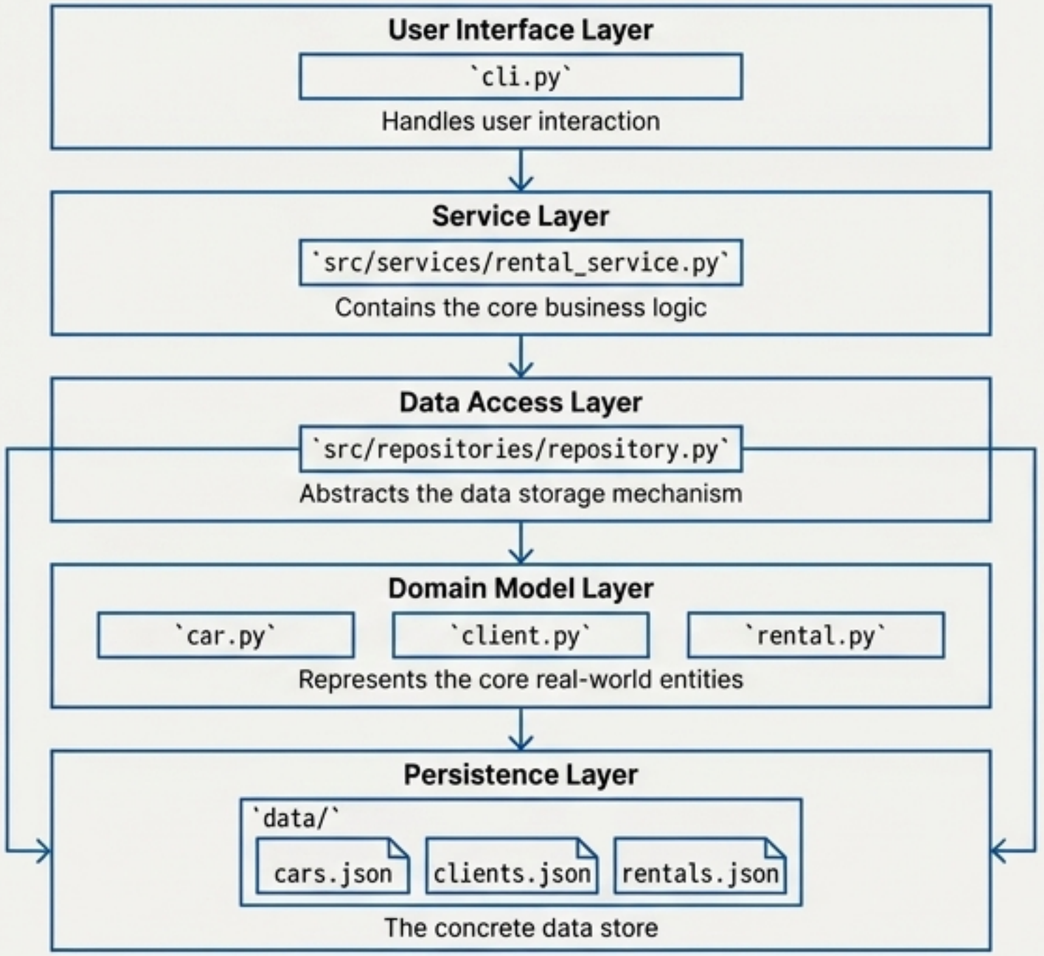
Composable, Understandable, Predictable, Idiomatic, and Domain-based.



Data Management

Full CRUD operations using JSON for data persistence.

A Layered Architecture for Modularity and Scalability



**Key Takeaway:* This separation of concerns was a foundational decision, ensuring low coupling and high cohesion from the start.

Sprint 1: Building a SOLID Foundation

Sprint Goal: To establish a robust architectural foundation, implement core domain models, and integrate design principles from day one, resulting in a working prototype with partial CRUD.

Key Accomplishments

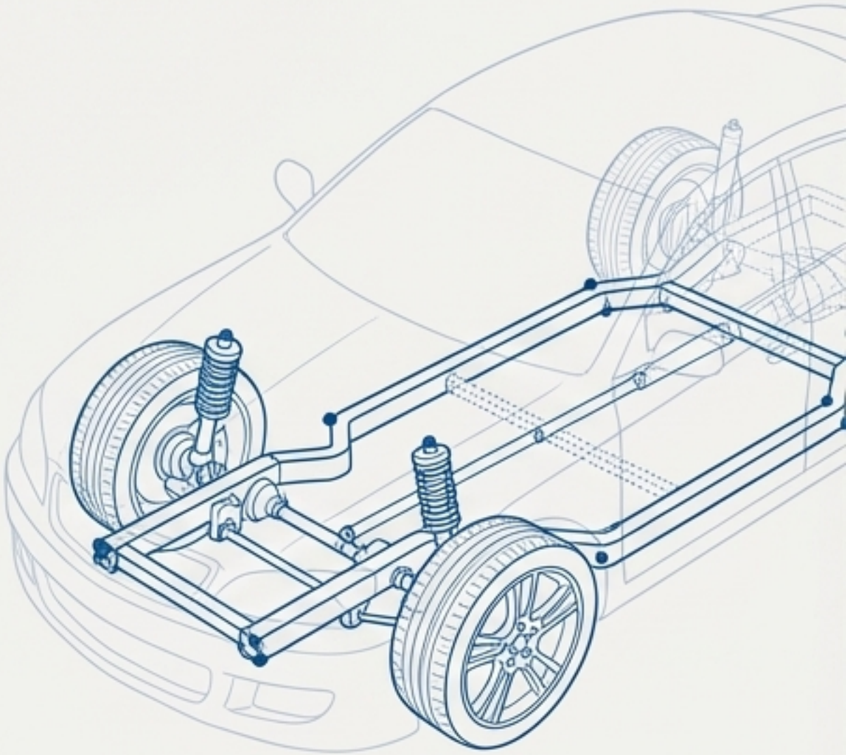
Project Structure: Created the modular directory structure (`src/`, `tests/`, `docs/`).

Core Models Implemented: `Car`, `Client`, and `Rental` classes were designed to represent the system's core entities.

Data Layer Established: A generic `Repository` class was implemented to handle data persistence with JSON, abstracting file I/O operations.

Create
Read

Initial Functionality: Create and Read operations were successfully implemented and tested.

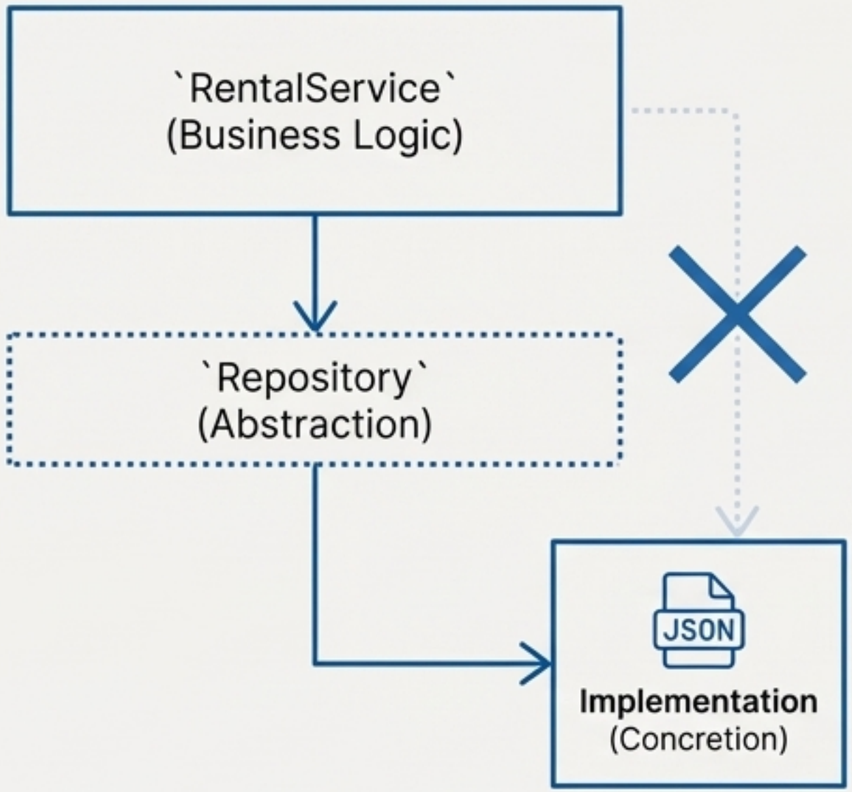


Principle in Practice: Dependency Inversion

1. The Principle (SOLID):

>“Depend on abstractions, not on concretions.”

2. How We Applied It:



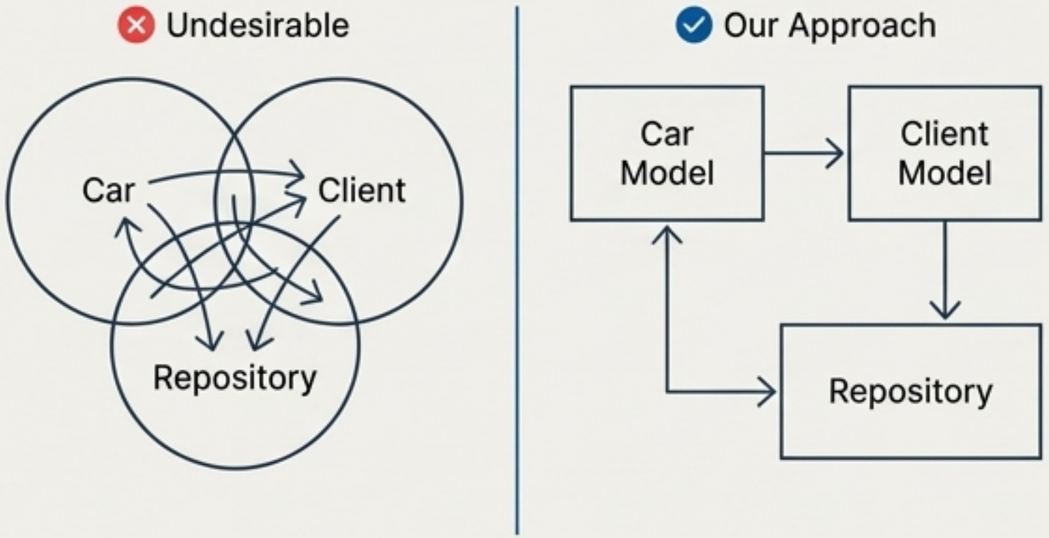
- Our `RentalService` (high-level business logic) does not directly interact with `json` files.
- Instead, it depends on the abstract `Repository` class.
- This ‘inverts’ the dependency, decoupling our core logic from the specific data storage mechanism.”

3. The Benefit:

This design allows the data storage to be swapped (e.g., from JSON to an SQL database) in the future without requiring any changes to the `RentalService` class, adhering to the Open/Closed principle.

Engineering for CC Cohesion, Clarity, and Control

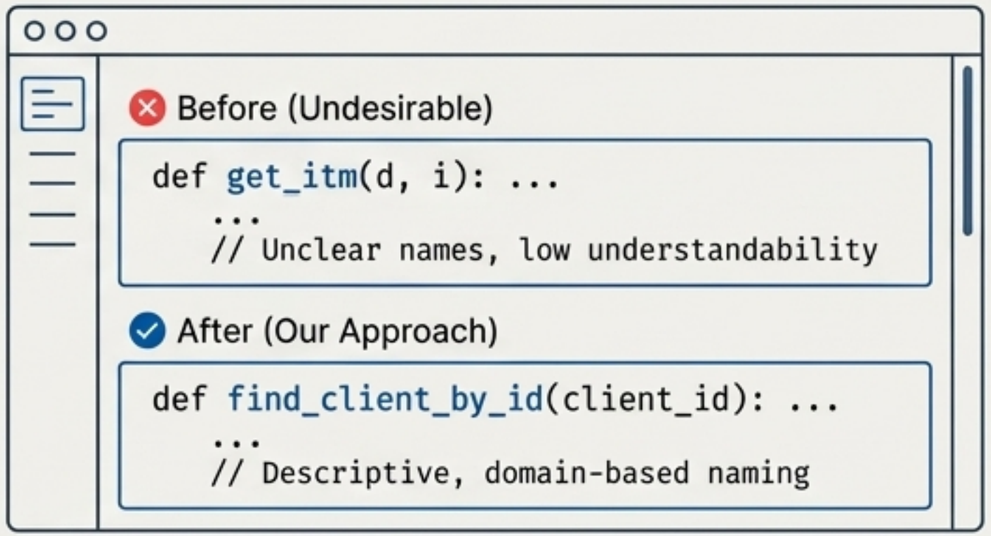
GRASP in Action: High Cohesion & Low Coupling



Each class has a single, well-defined purpose. The `Car` model only manages car attributes. The `Repository` only manages data persistence. They don't know about each other's internal workings.

This makes individual components easier to understand, test, and maintain independently.

CUPID in Action: Domain-Based & Understandable Code



We used clear, descriptive names for classes and methods (`add_car`, `find_client_by_id`) that directly mirror the real-world domain of car rentals.

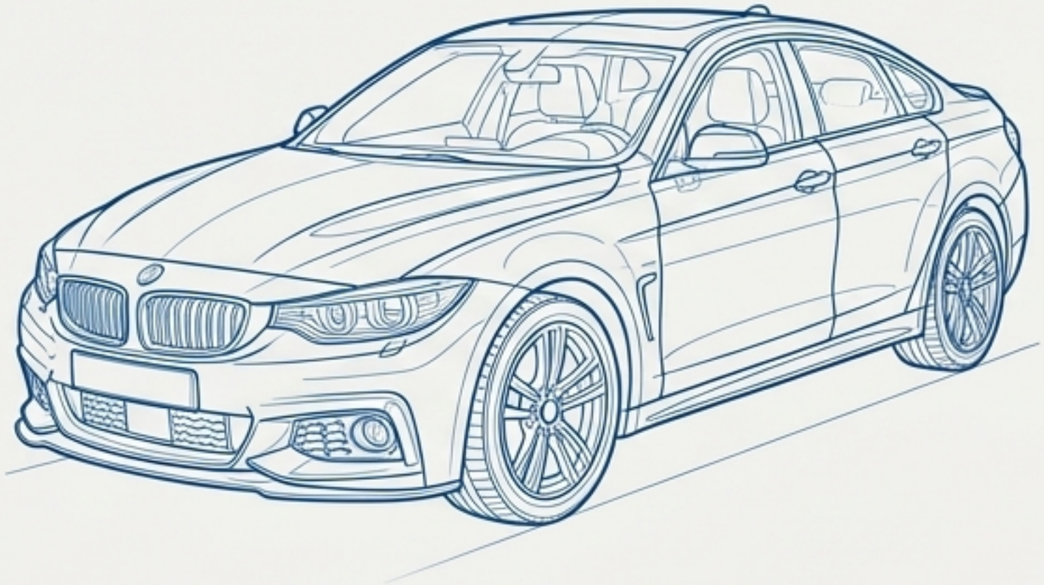
The code becomes self-documenting and intuitive for other developers.

Sprint 2: From Core Logic to a Complete Application

Sprint Goal: To complete all functionality, enhance code quality through comprehensive testing and refinement, and finalize all project deliverables.

Functionality Completed

- ☑ **Full CRUD:** Implemented the remaining Update and Delete operations.
- ☑ **User Interface:** Built a functional Command Line Interface (CLI) for user interaction.
- ☑ **Modularity:** Refined the separation between models, services, and repositories.
- ☑ **Robustness:** Strengthened exception handling and logging across the application.



The Interface: Managing the System via CLI

A clean, command-driven interface allows for managing cars, clients, and rentals directly from the terminal. The design is modular, with intuitive commands for all CRUD operations.

Logically grouped commands for managing each entity.

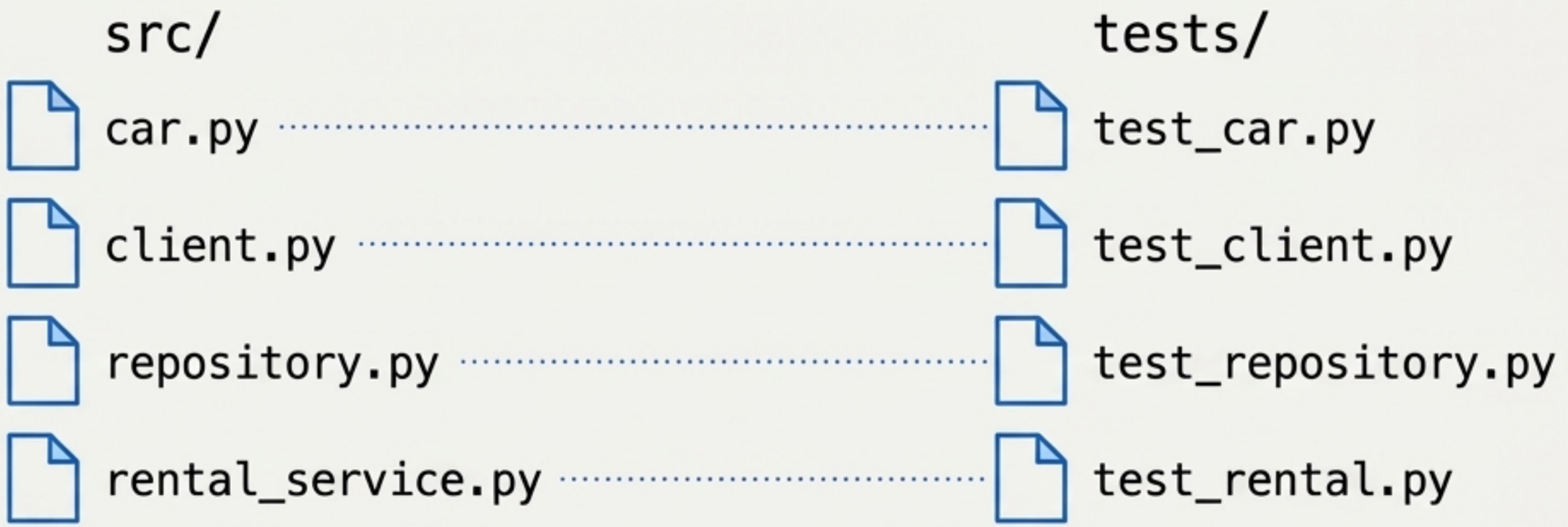
Intuitive and consistent syntax for actions (add, update, delete, list).

Clear, user-friendly feedback for successful operations.

```
PS C:\Users\acer\OneDrive\iş masası\oop-son\car-rental-system> python cli.py
Usage: cli.py [OPTIONS] COMMAND [ARGS]...

  Car Rental Management CLI

Options:
  --help  Show this message and exit.

Commands:
  car     Manage cars
  client  Manage clients
  rental  Manage rentals
PS C:\Users\acer\OneDrive\iş masas\oop-son\car-rental-system> python cli.py car add
Brand: Lada
Model: Niva
Daily Rate: 100
Car Type: SUV
Seats: 4
INFO:src.repositories.concrete_repository:Item created: 0ea01705-f1cf-4d21-b9fb-7758c03f6d2c
Car added: Lada Niva
PS C:\Users\acer\OneDrive\iş masas\oop-son\car-rental-system>
```

Ensuring Robustness Through Comprehensive Testing

Testing Framework: We utilized `pytest` to write unit tests for each component in isolation.



`tests/test_car.py` : Validates the `Car` model's logic and constraints.
`tests/test_client.py` : Validates the `Client` model's behavior.
`tests/test_repository.py`: Ensures the JSON data persistence layer works as expected.
`tests/test_rental.py` : Verifies the business logic within the `RentalService`.

Sprint 2 Goal Met: The test suite was expanded to cover edge cases and error scenarios, achieving our goal of **over 80% code coverage.**

A Complete Project Includes a Clear Guidebook

1. Technical Documentation

`docs/technical_documentation.md`

- Provides a deep dive into the system's architecture.
- Includes detailed Class Diagrams and ER diagrams.
- Explains key design choices and the specific application of SOLID, GRASP, and CUPID principles.

2. User Guide

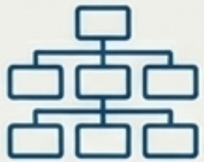
`docs/user_guide.md`

- Offers step-by-step instructions for setting up and running the application.
- Contains practical examples for every CLI command to demonstrate full CRUD functionality.

Project Delivered: A Synthesis of Theory and Practice

Summary of Achievements

A Fully Functional System:
We successfully built a complete Car Rental Management system with full CRUD operations and a persistent JSON data store.



Principled by Design: We demonstrated the practical application of OOP, SOLID, GRASP, and CUPID principles throughout the architecture.

A Structured Process: We followed a methodical, two-sprint agile methodology, moving from a solid foundation to a polished final product.

A Complete Deliverable: The final project is high-quality, thoroughly tested, and comprehensively documented.

GitHub Repository: The complete source code is available for review at:
`github.com/M4sayev/car-rental-system`