Here is the humanized version of your content:

Coding Standard – Car Rental System

It explains how Car Rental System complies with coding standards expected in such a project.

Modularity and Encapsulation

The system is separated into distinct layers — UI, Business, Domain, and Data. There is only one responsibility per class:

- `CarRepository` only deals with saving &
- `CarService` is responsible for rules about cars.
- DTOs and Mappers help to pass data safely without exposing database information.

It is easier to maintain, to test, and to extend.

Performance Considerations

It uses **SQLAlchemy** with SQLite.

- It is a **Singleton** so that the app does not waste time reconnecting whenever it needs a database.
- For a project of this size, setup is fast and efficient.
- It doesn't yet include advanced optimizations such as caching but is set to handle larger improvements down the road.

Performance is good for project scope.

Commenting and Documentation

There are informative **docstrings** to comment about methods and classes. In addition to that, the project supplies:

- A comprehensive **README** file with information about installation and usage.
- Separately **User** and **System Documentation** PDFs.
- **UML diagrams** (Class, Sequence, Use Case) to explicitly present design.

Good documentation is one of the project's strengths.

Indentation and Formatting

It follows **PEP8 conventions**:

- Consistent 4-space indentation.
- Clear and easy-to-read formatting.

- No issues with mixed spaces/tabs.

It's a tidy-looking code.

Naming Conventions

It uses systematic and unambiguous naming throughout:

- Classes also employ **PascalCase** (e.g., `CarService`, `RentalRepository`).
- Functions and methods use **snake_case** (e.g., `list_users`, `create_rental`).
- Interfaces also contain a prefix `I` (e.g., `ICarService`, `IRentalService`).
- DTOs and Mappers follow a predictable pattern (`UserDto`, `UserMapper`).

It simplifies a code to make it easier to understand.

Final Evaluation

Guideline	Rating	Notes
Modularity & Encapsulation	Strong	Clear division of concerns
Performance Considerations	Good	Optimized with Singleton DBManager, could add caching later
Commenting & Documentation	Outstanding	Extensive documentation and diagrams
Indentation & Formatting	Clean	PEP8-compliant and consistent
Naming Conventions	Inconsistent	Random with no overall strategy or standard

General: It's a good example of good coding practices. It is easy to read and document as a module. Advanced performance tuning is where it can be improved but for its scope it is good enough.