Support & Maintenance of Car Rental System

Current Stable Release: v1.0.0

This plan discusses steps that will keep all of the software, hardware and processes utilized to run the Car Rental System up to date over its lifecycle.

---

1. Software Maintenance Management

Layered Architecture It allows you to break down each system into UI layer, Business Layer, Domain Layers and Data layers so that with minimal impact the components can be updated without much impact on the application as a whole.

Bug Fixes Defects can be addressed in their seprare layer (for example fixing validation in RentalService does not touch the database logic).

Documentation Good documentation – such as User Guides, System Documentation, UML diagrams and Coding Standards all assist the new developer to slot into the development process faster.

Testing Unit tests, in the pytest format, make it difficult for developers to break features with refactoring.

Together, these practices provide a solid basis for the long-term maintenance of software.

---

2. Versioning

The project follows the guidelines of Semantic Versioning (SemVer):

- **MAJOR version** - when you make incompatible API changes, or very rarely for structural changes that break backwards compatibility.
- **MINOR version** – when you add functionality in a backwards compatible manner.
- **PATCH version** - for fixing bugs and making small improvements.

Example versions:

- v1. 0. 0 – Initial stable release
- v1. 1. 0 - new feature (rental fee calculator or any other)
- v1. 1. 1- Fixed a bug

Each version is well marked with a tag for easy tracking in github.

How to Add a New Version

To ensure that the Car Rental System follows a clear and structured versioning strategy, each release should be assigned a version number using **Semantic Versioning (SemVer)**.

Steps to add or update the version:

1. **Update Version File**

- Edit the `version.py` file in the project root:

```
# version.py
__version__ = "1.0.0"
```

- This file keeps track of the current release version.

2. **Display Version in Application**

- In `main.py` (or your entry point), import and print the version:

```
from version import __version__
print(f"Car Rental System - Version {__version__}")
```

- This makes the version visible when the system runs.

3. **Update Documentation**

- Add the version number at the top of the `README.md` file:

```
# Car Rental System
**Version:** 1.0.0 (Stable Release)
```

- Update `MaintenanceAndSupport.md` to reflect the current stable version.

4. **Tag the Version in GitHub** (if using Git)

```
git tag -a v1.0.0 -m "Stable Release v1.0.0"
git push origin v1.0.0
```

- Tags allow anyone to download the exact version of the project.

Example:

- Current stable version: **v1.0.0**
- Next feature release: **v1.1.0**
- Bug fix release: **v1.1.1**

By following this process, the Car Rental System can evolve in a controlled way, with each change clearly tracked and documented.

---

3. Backward Compatibility

To be compatible with prior versions as the system changes:

DTOs and Mappers DTO acts as a middleware layer between the internal db and your external interfaces (UI, API), protecting them from changes in the internal storage.

Database Migration Strategy Schema changes will be managed with Alembic to enable controlled and incremental upgrades without any loss of data.

API Stability Older versions will continue to be available in case RESTful endpoints are added (for instance, via FastAPI) as well, and users can transition.

Deprecation Policy Deprecated features shall be documented explicitly and supported for at least one minor release before removal.

This method enables contiuing use of an old terminal even when new functionalities are developed.

Summary

Car Rental System is designed for sustainability in the long term:

- Maintenance is straightforward as a modular structure and comprehensive documentation has been utilized.
- SemVer is employed for versioning in a straightforward, senseful manner.
- It maintains compatibility through protective data layers, cautious database migrations and thoughtful deprecation policies.

All of the major components discussed on the rubric will be addressed in this plan to help keep your system maintainable, extensible, and reliable.