SDLC Documentation for Smart Parking Lot Manager (Console App)

# 1. Introduction

This project is a console-based Smart Parking Lot Manager developed in Python. It simulates a modern parking management system for cars and two-wheelers. The system allows users to park vehicles, unpark them, check slot availability, and calculate parking charges based on vehicle type and duration of stay.

# 2. SDLC Model Used – Waterfall

We used the Waterfall SDLC Model, suitable for academic projects where:

• Requirements are well-defined from the start  
• Development proceeds in sequential phases  
• Testing occurs after implementation

Phases in Waterfall:  
- Requirement Analysis  
- System Design  
- Implementation  
- Testing  
- Deployment  
- Maintenance

# 3. Requirement Analysis

Functional Requirements:

• Park and unpark vehicles (cars and bikes)  
• View available parking slots by type  
• Search parked vehicle by number  
• Calculate parking fee based on time and vehicle type

Non-Functional Requirements:

• Console-based interface  
• Modular code (separated into files)  
• In-memory operation (no external database)

# 4. System Design

The system is broken into multiple modules for clarity:

|  |  |
| --- | --- |
| File | Purpose |
| vehicle.py | Represents a vehicle and its details |
| slot.py | Represents a parking slot |
| parking\_lot.py | Manages parking, unparking, searching, and availability |
| main.py | User interface via console menu |

Architecture Diagram:  
main.py --> parking\_lot.py --> [slot.py, vehicle.py]

Class Relationships:  
• ParkingLot has a list of ParkingSlot objects  
• ParkingSlot may have a Vehicle object assigned  
• main.py takes user input and routes actions to ParkingLot

# 5. Implementation

vehicle.py

class Vehicle:  
 def \_\_init\_\_(self, license\_plate, vehicle\_type, entry\_time):  
 self.license\_plate = license\_plate  
 self.vehicle\_type = vehicle\_type  
 self.entry\_time = entry\_time

slot.py

class ParkingSlot:  
 def \_\_init\_\_(self, slot\_id, slot\_type):  
 self.slot\_id = slot\_id  
 self.slot\_type = slot\_type  
 self.is\_occupied = False  
 self.parked\_vehicle = None

parking\_lot.py

class ParkingLot:  
 def \_\_init\_\_(self):  
 self.slots = []  
 self.vehicle\_map = {}  
 self.setup\_slots()

main.py

def main():  
 lot = ParkingLot()  
 while True:  
 print("\n1. Park Vehicle\n2. Unpark Vehicle\n3. View Available Slots\n4. Search Vehicle\n5. Exit")  
 choice = input("Enter your choice: ")  
 ...

# 6. Testing

|  |  |  |
| --- | --- | --- |
| Test Case | Input | Expected Output |
| Park car | Car, MH01AB1234 | Allocated to available car slot |
| Park bike | Bike, MH02XY7890 | Allocated to available bike slot |
| Unpark | MH01AB1234 | Parking time & charge displayed |
| View slots | Option 3 | All free slots by type |
| Search | MH02XY7890 | Shows slot number and type |

# 7. Deployment

To run the project:  
1. Place all .py files in one folder  
2. Open terminal in that folder  
3. Run the program:  
 python main.py  
  
No installation or dependencies required.

# 8. Maintenance

Future updates could include:  
• Saving and loading data to/from a file  
• Adding admin panel for reporting  
• Converting console app to GUI (Tkinter) or Web (Flask)  
• Real-time clock-based parking fee calculation

# 9. Summary Table of File Roles

|  |  |  |
| --- | --- | --- |
| File | Role | Description |
| vehicle.py | Data model | Holds vehicle details |
| slot.py | Data model | Manages slot occupancy |
| parking\_lot.py | Logic/controller | Handles operations like park/unpark/search |
| main.py | Interface | User menu and command routing |