Final Project Stage 4: Design

SUU Electronic Learning Center Parking App (Lots 14, 17, 18, & 19)

Author Jie Liang

Course CS2450

Course Professor Dr. Gang Liu

Contents

1. Project Overview

1.1 Project Overview

The ELC Parking App is a real-time parking management system designed to help user who will park near Southern Utah University's Electronic Learning Center. The system provides students, staff, and visitors with instant access to parking information across four lots (14, 17, 18, 19), helping users make parking decisions and reducing time spent searching for available spaces.

1.2 User requirements

Many computer sciences course classroom are in the this building first and third floor. The students attend computer courses want to find parking spaces near the building

The testing center is in the second floor of this building. Future students or exist students want to park nearby to take tests in testing center.

Many professors have offices on the fourth floor. Staff and also the students who want to visit professors in office want to park nearby.

But most of time the students' parking lots, No 17 is full after 9am. And the staff parking lot No. 18 is a parking lot with very limited parking space.

2. System Architecture design

2.1 Architecture Pattern: Client-Server Model

The system follows a client-server architecture with clear separation of concerns:

The parking lot administrator is a part of university administration. It provides the service to provide information of the parking lot availability to make the user to easy and effective to access the parking lot.

The users (students, staff and visitors who need to access the ELC building) use the app to make parking easy.

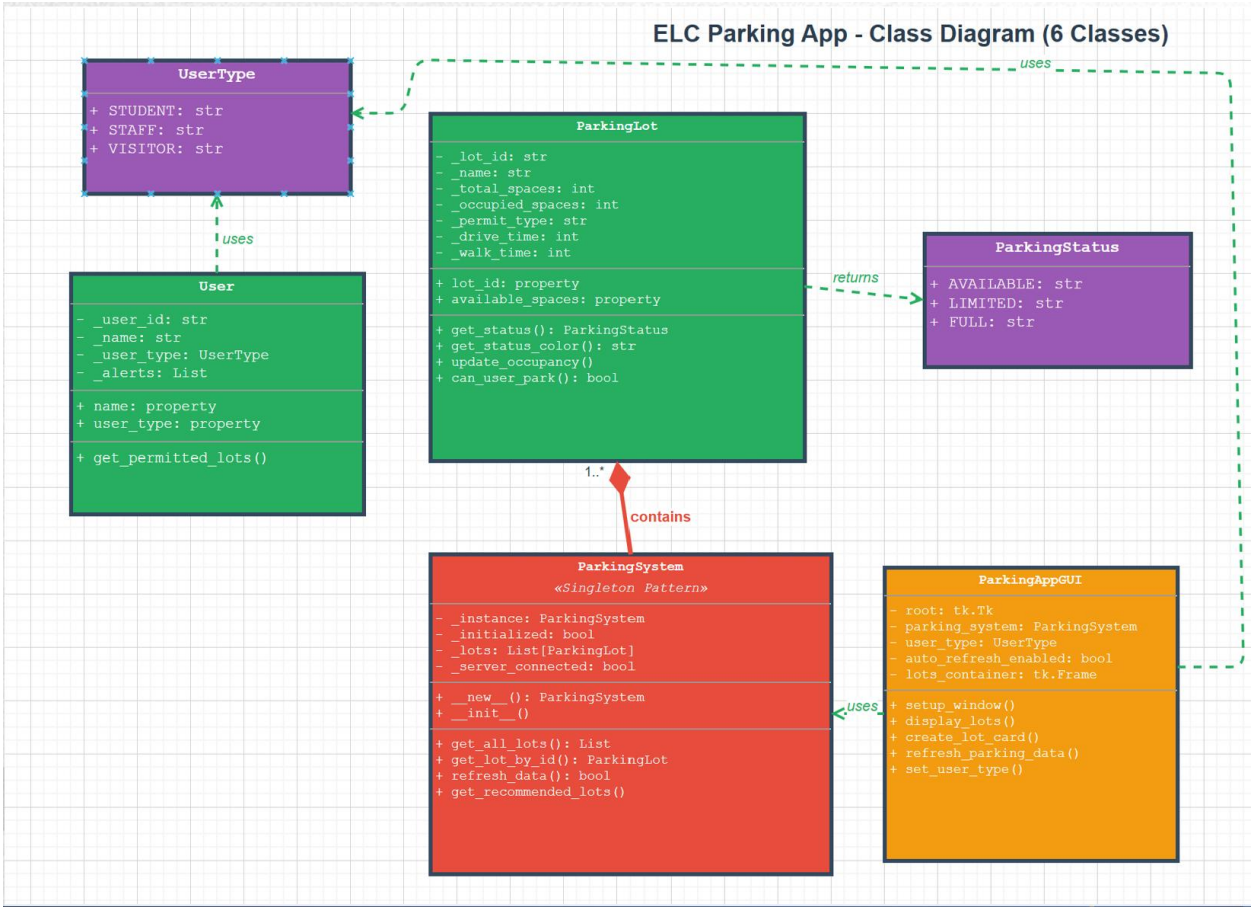
Below are the tables shows the functionality of different part of this application system.

Component	Technology	Purpose
Client Application	Python + Tkinter	User interface for viewing parking availability
Server/Backend	Python + Flask	REST API providing parking availability information and managing the accessibility
Data Storage	JSON	storage of parking lot states
Admin Interface	HTML	Web-based parking lot management

2.2 Major Class

There is total 6 classes which connect the client and server side.

2.2.1 class relationship structure



2..2.2 Class functionality

Class	Purpose	Key Responsibilities
UserType	Enum defining user roles	• STUDENT • STAFF • VISITOR
User	Represents system user	• Store user information • Manage permit types • Filter allowed parking lots
ParkingStatus	Enum for lot availability	• AVAILABLE (>20% free) • LIMITED (5-20% free) • FULL (<5% free)
ParkingLot	Individual parking lot	• Track capacity and occupancy • Calculate availability • Determine status • Validate occupancy bounds
ParkingSystem	System controller (Singleton)	• Manage all parking lots • Provide centralized access • Ensure single instance
ParkingAppGUI	Desktop user interface	• Display parking data • Handle user interactions • Auto-refresh every 10s • Show color-coded status

3. Applied design patterns and their purpose

3.1. Encapsulation Pattern

3.1.1 Purpose:

To protect parking lot data from outsider watch and change, use encapsulation to hide internal details and provide controlled access.

3.1.2 Implementation:

Private Attributes: All data uses underscore prefix (_lot_id, _name, _occupied_spaces)

Property Decorators: Provide read-only access to private data

No Setters: Users can view data but cannot modify directly

Validated Methods: update_occupancy() includes bounds checking

3.1.3 coding example from ParkingLot class:

```
class ParkingLot:
    def __init__(self, lot_id, name):
        self._lot_id = lot_id          # Private
        self._name = name              # Private

    @property
    def lot_id(self):
        return self._lot_id            # Read-only

    # No setter - lot_id cannot be changed

    def update_occupancy(self, occupied):
        # Controlled modification with validation
        self._occupied_spaces = max(0, min(occupied, self._total_spaces))
```

3.2. Singleton Pattern

3.2.1 Purpose

Ensure only ONE instance of ParkingSystem exists throughout the application, protect two or three parking lots data update at the same time and cause confuse.

3.2.2 Implementation:

Class Variable: _instance = None stores the single instance

__new__ Method: Controls object creation

Initialization Guard: _initialized flag prevents re-initialization

3.2.3 coding example from ParkingSystem class:

```
class ParkingSystem:
    _instance = None  # Class variable

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance._initialized = False
        return cls._instance  # Always return same instance

    def __init__(self):
        if self._initialized:
            return  # Don't re-initialize
        self._initialized = True
        self._lots = []
```

4. Testing design and completement

Total 15 tests are designed and list in the table below. All tests are passed

Test Category	Tests	Purpose
User Tests	2	Verify user initialization and permission filtering
ParkingLot Tests	8	Test capacity, status calculation, bounds checking, permit logic
ParkingSystem Tests	4	Validate Singleton pattern and lot retrieval
Integration Tests	1	Test complete user workflow from selection to display

5. UI design and completement

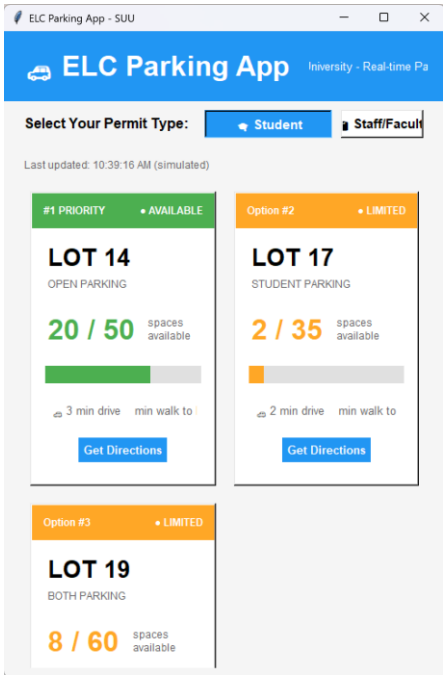
5.1 UI design and completement for 4 user case

5.1.1 UC1 - Student Parking Selection

Implemented user type filtering. Students see Lots 17, 19, 14

with real-time availability.

Code: User.get_permitted_lots() for STUDENT type

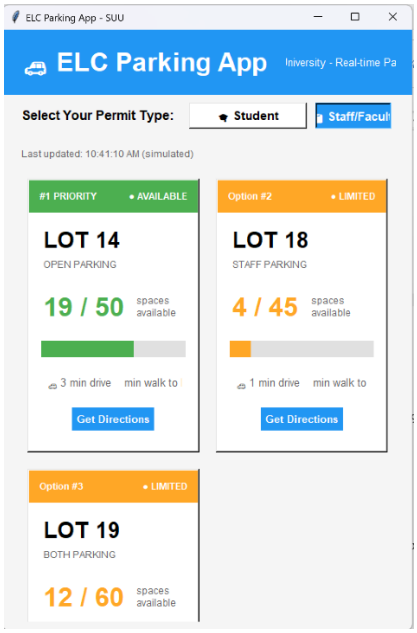


5.1.2 UC2 - Staff Priority Parking

Implemented user type filtering. Staff see Lots 18, 19, 14

with Lot 18 shown first as priority.

Code: User.get_permitted_lots() for STAFF type

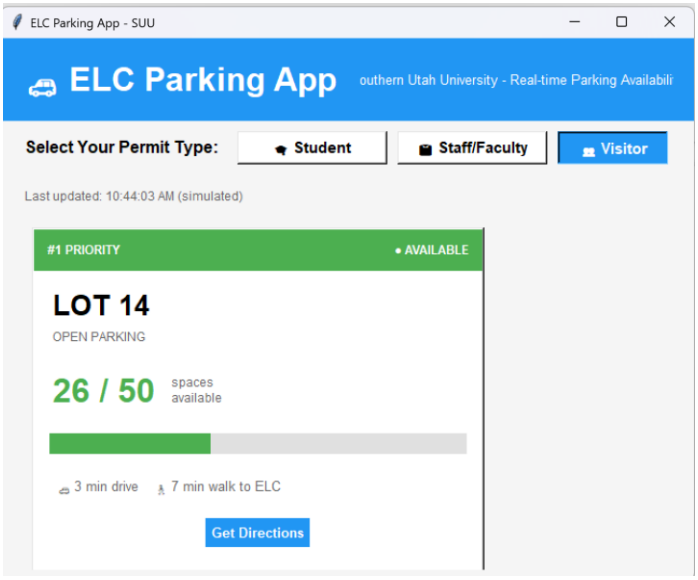


5.1.3 UC3 - Visitor/Guest Parking

Implemented user type filtering. Visitors see only Lot 14

(open parking, no permit required).

Code: User.get_permitted_lots() for VISITOR type

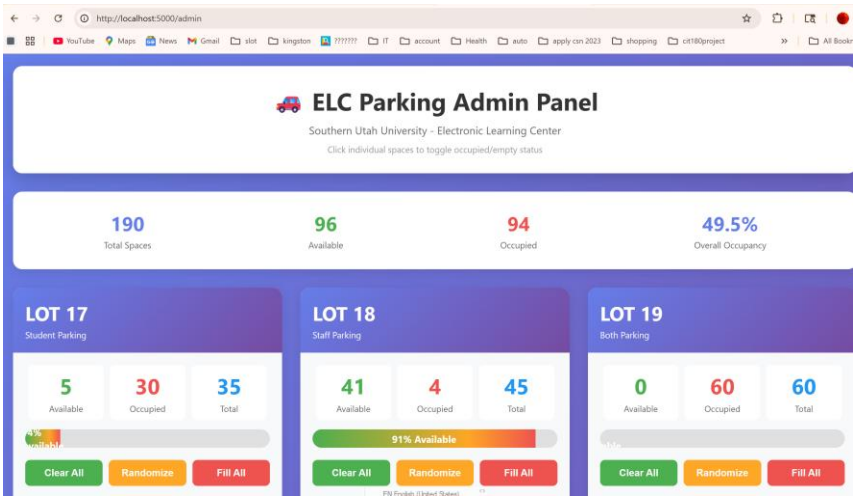


5.1.4 UC4 - System Administration:

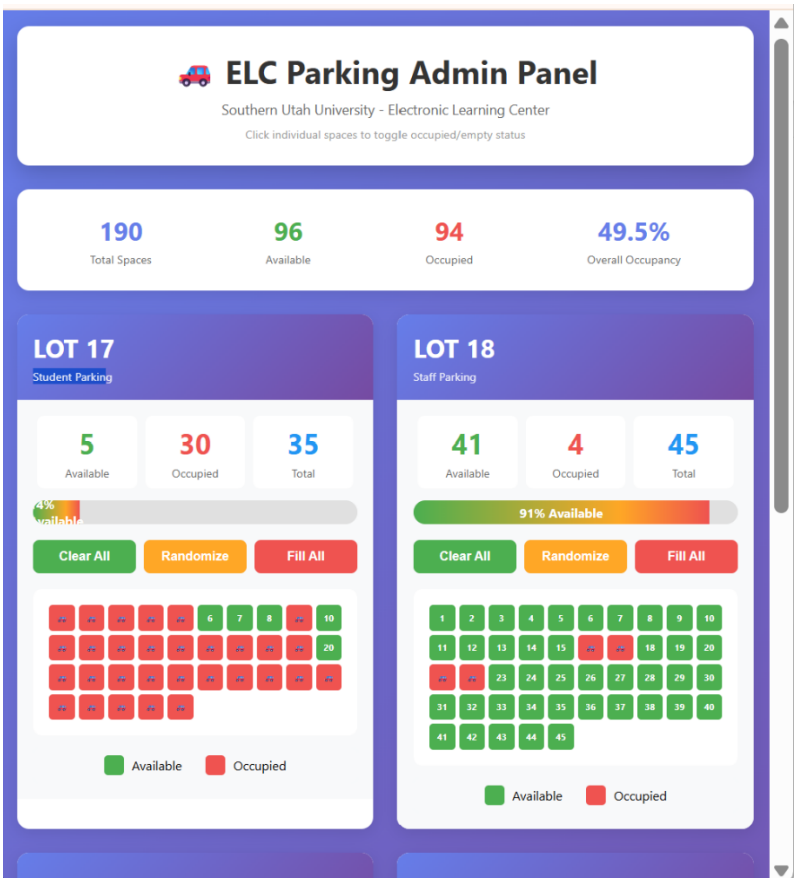
Implemented web interface at localhost:5000 for lot configuration and sensor simulation.

Code: parking_server.py with admin.html

Parking lot administration panel windows version



Parking lot administration panel mobile devices friendly version



5.2. UI Implementation from technical view

Main Dashboard- Tkinter:

✓ Color-coded status (Green/Orange/Red)

✓ User type selector (Student/Staff/Visitor)

✓ Real-time availability display (e.g., "25/35 available")

✓ Drive time and walk time to ELC

✓ Auto-refresh (10 seconds)

✓ Manual refresh button

✓ 2-column grid layout

Administration Panel - HTML:

✓ Web interface at localhost:5000

✓ Individual parking space toggle

✓ Bulk operations (fill/clear/full/randomize)

✓ Real-time parking grid visualization

6. Reflections and Learnings

6.1 Technical Skills Developed

In this project, I practice what I learn in CS1410 OOP design and design pattern, first try client-Servier architecture using html as administration control panel and combine the software engineering concept what I learn in CS2450 to control the time I put on in this project and improve my testing design.

Below are the details

Skill Area	What I Learned
Object-Oriented Design	Implementing design patterns (Singleton, Encapsulation) in a very practical case.
Software Architecture	Designing client-server systems with clear separation of concerns; choosing appropriate technologies for each layer (Tkinter for client, Flask for server
API Development	Building APIs with Flask; handling HTTP requests and responses; JSON data serialization and deserialization
GUI Programming	Creating responsive interfaces with Tkinter, this project user interface is for mobile device; implementing auto-refresh mechanisms; designing user-friendly layouts with real time updated contents
Testing Methodology	Practice the unit test skill I learned in CS1410 and begin to build the integrate test
Data Validation	Implementing bounds checking; ensuring data integrity through encapsulation; preventing invalid states

6.2 Software engineering principles learned and practiced

6.2.1. Requirements Analysis

In this project, I followed the professor's request use user case and serinior to analyze the requirement. This is new for me but it improves to get the clear picture about the requirement

6.2.2 Iterative Development:

In this project, I write the requirement and build the code and testing which can run then I come back to write the clearer implementation plan and modify code to make the user interface more friendly. It is a iterative development (Agile method).

6.2.3. User-Centered Design:

This project is based on the user actual needs: (students arriving on time to attending class, staff needing quick office access, visitors finding permit-free parking) kept features practical and useful rather than just for learning and courses.

7. Challenges Faced

7.1 hardware and software interface

parking lot sensor simulator system IO with computer software. I have not try to build the hardware and software IO and I believe there are still a lot of uncertainty, also limited by which software language I can use to control the hardware.

7.2 security protection for the administration (server):

At this moment, I have not built the security (password or passkey) protection for the administration side.

8. Future work

8.1 using Raspberry Pi to simulate the parking lot admin system

I am interested in software and hardware connections which can make the software be practical, especially in mobile devices.

8.2 build the security layer for administration control side

This is necessary to make this application more practical. I need to do research to see how the data breach happened in recent years to balance the cost and the security to build the right security layer for this system.

8.3 User Preferences & Profiles

In the user app (client side) build an setting up page to let the user to put his position: student/one time user/staff so he uses need one click to check the parking lots availability specially for him.

In the server side, record the parking lot available history so it is easy to provide smart choose for the user.