

CS 3354.001 Software Engineering

Final Project Deliverable 2

Smarter Parking for a University Campus

Patrick O'Boyle

David Hiser

Gaurang Goel

Saumya Karia

1. Tasks (Deliverable 2)

Claiming tasks:

- Claim a section of the slides (for the live presentation. **3-4 min** per person)
 - More info in the deliverable document. Select from **Step 8's** template
- Step 7 (References): Include whatever you cite here in IEEE

Patrick O'Boyle:

- ~~Step 1, Step 2 + Setup and organize document/tasks (done)~~
- ~~Step 5 (comparisons) (done)~~
- ~~Step 6 (conclusions) (done)~~
- ~~Slide Duties (done)~~
 - ~~Title slide~~
 - ~~Project objective / Background~~
 - ~~UI "demo"~~
- Step 10 (check to make sure everything is on github)

David Hiser:

- Step4
- ~~Slide Duties~~
 - ~~Use case diagram~~
 - ~~Sequence Diagram~~

Gaurang Goel:

- ~~Slide Duties (Done)~~
 - ~~Class diagram~~
 - ~~Architectural design~~

Saumya Karia:

- ~~Part 3 (Project Scheduling, Cost, Effort and Price Estimation)~~
- ~~Part 7 (IEEE Reference)~~
- ~~Slide Duties (Done)~~
 - ~~Cost Estimation~~
 - ~~Project Timeline~~
 - ~~Functional and Non-functional Requirements~~

2. Project Deliverable 1 Content

1. Final Project Draft Description

Good choice for a topic! Watching movies is such a popular activity, particularly during these times. Your design is promising to be a very useful one for determining a good one to see, as there are so many of them.

Your team should consider a more detailed break down of the tasks as there are more of them (please see the project specs). There is no need to resubmit this proposal, though. Just make sure that you don't miss any task. Also include the "had to be added late" new member to your team.

In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.

Fair delegation of tasks.

Please share this feedback with your group members.

You are good to go. Have fun with the project and hope everyone enjoys the collaboration.

Complying with Changes:

- More detailed tasks: We provided a detailed breakdown for tasks in part 3 and will continue to update it for future deliverables
- Late member: They are from a different section, so that news didn't apply.
- Comparison with similar applications + differentiation: We'll make sure to cover this in our final report.

2. Github:

Link: https://github.com/saumyaK512/3354-Smart_Parking

Split tasks:

- 1.3: Completed by Saumya
- 1.4: First was accidentally completed by Saumya. Refer to the new commit by Gaurang.
- 1.5: Completed by Patrick

Note on the 1.4 mistake: Retroactively messing with the Git history is a bad idea. It's better to layer new commits over old ones as we did.

3. Tasks (Deliverable 1)

Patrick O'Boyle:

- Setup document
- Part 1
- Part 2 (1.5)
- Part 9

David Hiser:

- Part 6
- Part 7

Gaurang Goel:

- Part 2 (1.4)
- Part 8

Saumya Karia:

- Part 2 (1.2,1.3)
- Part 4
- Part 5

4. Software Process

We have chosen to employ the Spiral model as our software process model for this project. The project's overall size and complexity are the major factors. It makes intuitive sense to adopt the spiral approach since we are developing an entire IoT system that will communicate with various camera hardware and sensors throughout multiple parking structures.

Although this project is currently only focusing exclusively on the University of Texas at Dallas campus, if it needs to be expanded to multiple campuses across the country, the spiral model is the most practical option because it combines the controlled and systematic aspects of the waterfall model with the iterative nature of prototyping. Since no two campuses are the same, as the project grows, different stakeholders will have different requirements. For this reason, being able to prototype iterations quickly is very beneficial because it would only make sense to implement the project during the summer semesters when universities typically have fewer students and traffic on campus.

Additionally, this aids in developing a rapid prototype of the system that stakeholders can use to offer input, but which won't impact the whole campus if it breaks or needs significant adjustments.

5. Software Requirements

Functional Requirements:

1. Create a log file at the end of the day which includes the summary for each parking structure
2. Allow user to manually set occupancy for each parking structure
3. Check if cars license plate has been registered with a parking pass to allow entry into parking structure
4. Accurately deduce the number of parking spots occupied in a parking lot based on the parking category
5. Parking occupancy of each parking structure should get updated accurately based on a car occupying a parking spot or leaving the parking spot.
6. Check if a car is parked in the correct parking spot based on the type of parking pass linked to its license plate

Non-functional Requirements:

1. Product Requirements:

- a. Usability Requirements: The user should be easily be able to enter the garage without actually physically having use the system since everything is done by the sensors and cameras. Additionally the app should show the show the users the capacity for all the parking lots and a combined capacity of the campus
- b. Efficiency Requirements:
 - i. Performance Requirements: The software needs to run 24/7 since the parking garages are available all the time. For peak hours the system should be able to respond within 5 seconds of an availability of the parking lot or if a space is occupied, this time would be our 99th percentile. For the rest of the time it would be acceptable if the system takes upto a minute to respond. For the phone software, it should be able to run on android OS 7 and upwards and for IOS 10 and above.
 - ii. Space Requirement: In terms of the space the software should not take up more than 100 mb on the users phone, for the server we would require a bare minimum of 250 gb of storage with 16 gbs of ram space and 8 CPU Cores, all of this would include the storage space for the database of each parking space, ubuntu and other dependencies such as dockers, kubernetes etc. Additionally since the software uses some level of machine learning its good to have a good processor in the server.
- c. Dependability Requirements: The Worldwide Electrotechnical Commission (IEC) sets and maintains international standards through its Technical Committee TC 56, which provides systematic methodologies and tools for assessing and managing the dependability of equipment, services, and systems throughout their life cycles. Therefore this system should be able to hold up to those standards.
- d. Security Requirements: This system should hold up to the security standards created by the OWASP Application Security Verification Standards.

2. Organizational Requirements:

- a. Environmental Requirements: The servers should be housed in a dry, clean, well-ventilated, and temperature-controlled environment. The client should perform tests provided by the ASHRAE report, “Thermal Guidelines for Data Processing Environments” on a regular basis in order to maintain the servers and get the most performance out of the servers. The cameras and sensors used in the parking structures should also be cleaned on regular intervals and even be replaced if needed on a regular basis.
- b. Operational Requirements: The system should follow the standards and protocols set by the NASA Systems Engineering Handbook and the Department of Home Land Security: Developing Operational Requirements Standards and should check for the following sections in the operational requirements:
 - i. Electrical—Power consumption, efficiency, signal integrity
 - ii. Mechanical—Strength, motion required
 - iii. Structural—Capability to withstand environments in mission profiles
 - iv. Optical
- c. Development Requirements: The Software would be a culmination of multiple softwares such as Python for the machine learning aspect and communicating with the different hardware, the device softwares would use either Javascript, Kotlin and Swift and the server could be managed on javascript with node.js and mongodb as its dependencies. Additionally the software development would follow the agile framework.

3. External Requirements:

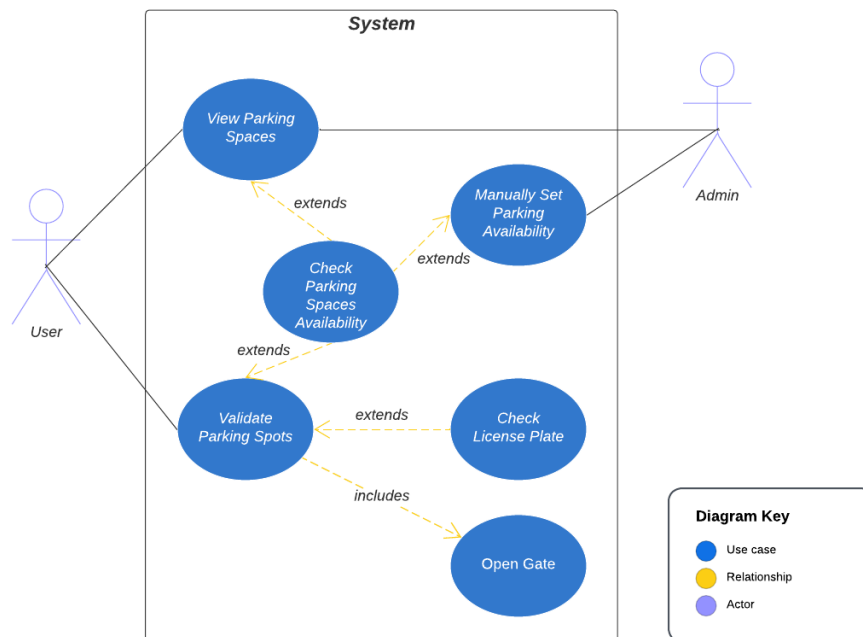
- a. Regulatory Requirements: The software needs to comply with the following regulations:
 - i. EU GDPR (General Data Protection Regulation)
 - ii. GLBA (Gramm-Leach-Bliley Act)
 - iii. PIPEDA (Personal Information Protection and Electronic Documents Act)
 - iv. CCPA (California Consumer Privacy Act)
- b. Ethical Requirements: While developing the software and maintaining it, any one and everyone involved will follow the Code of Ethics set up by

the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices.

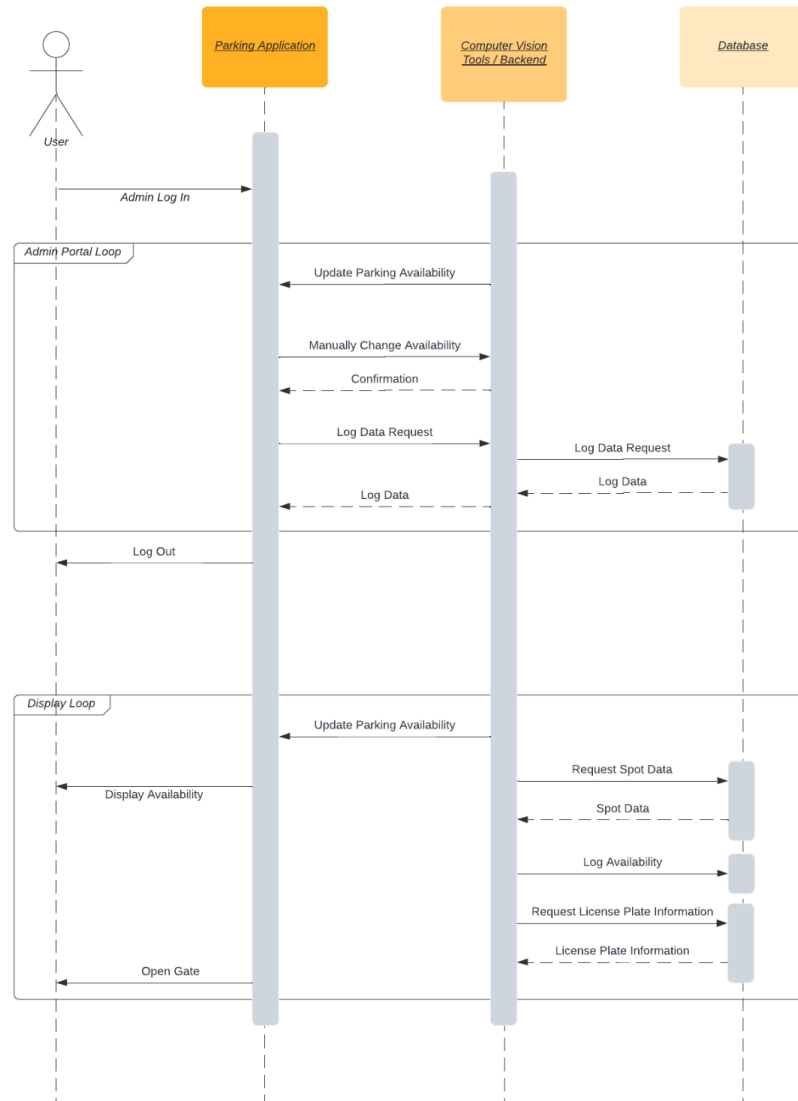
c. Legislative Requirements:

- i. Accounting Requirements: The cost of the software development and anything else related to it should follow guidelines under US GAAP and GASB created by the US Government.
- ii. Safety/Security Requirements: The development of the software should follow the standards for software safety created by IEEE SA - P1228

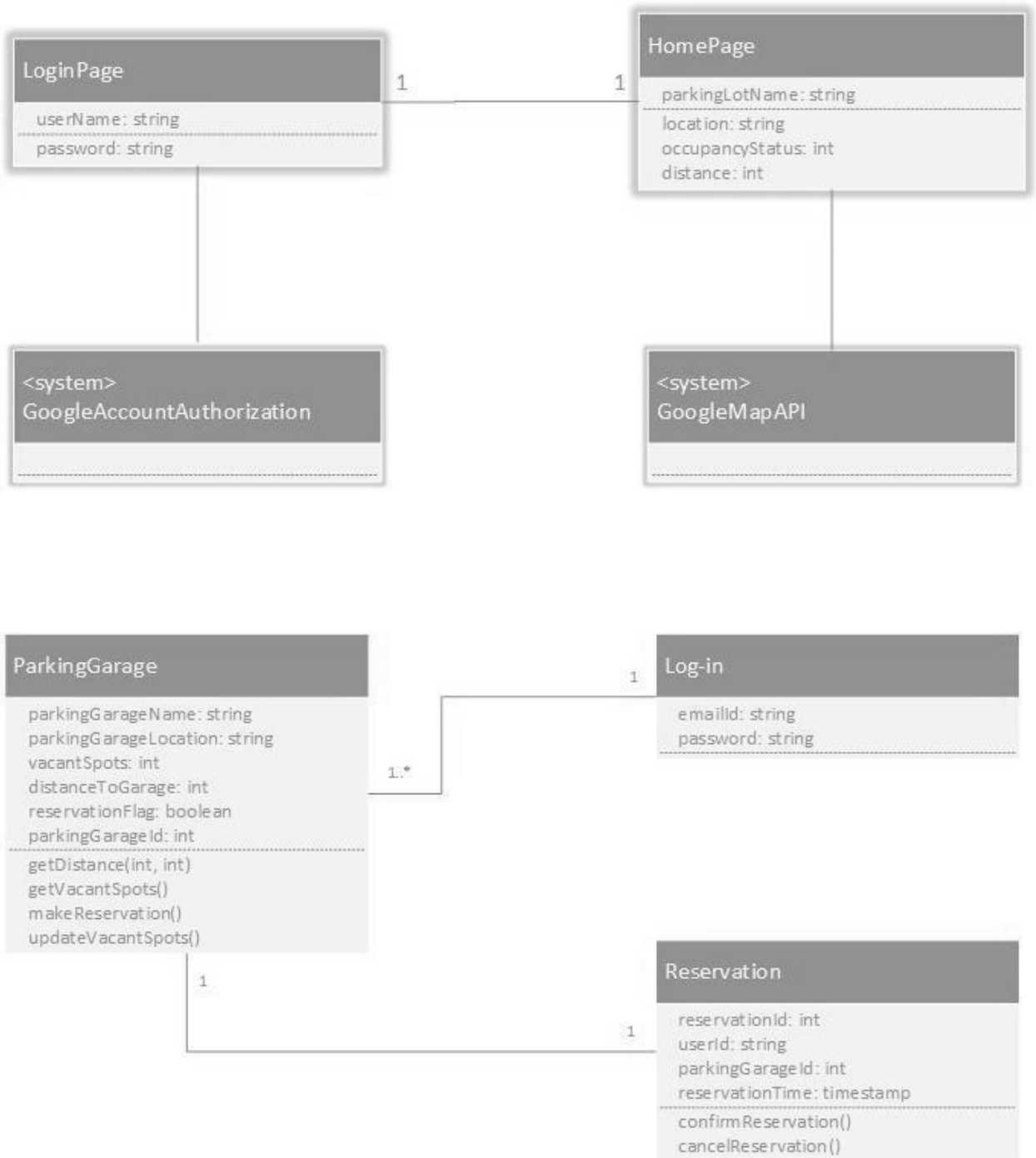
6. Use Case Diagram



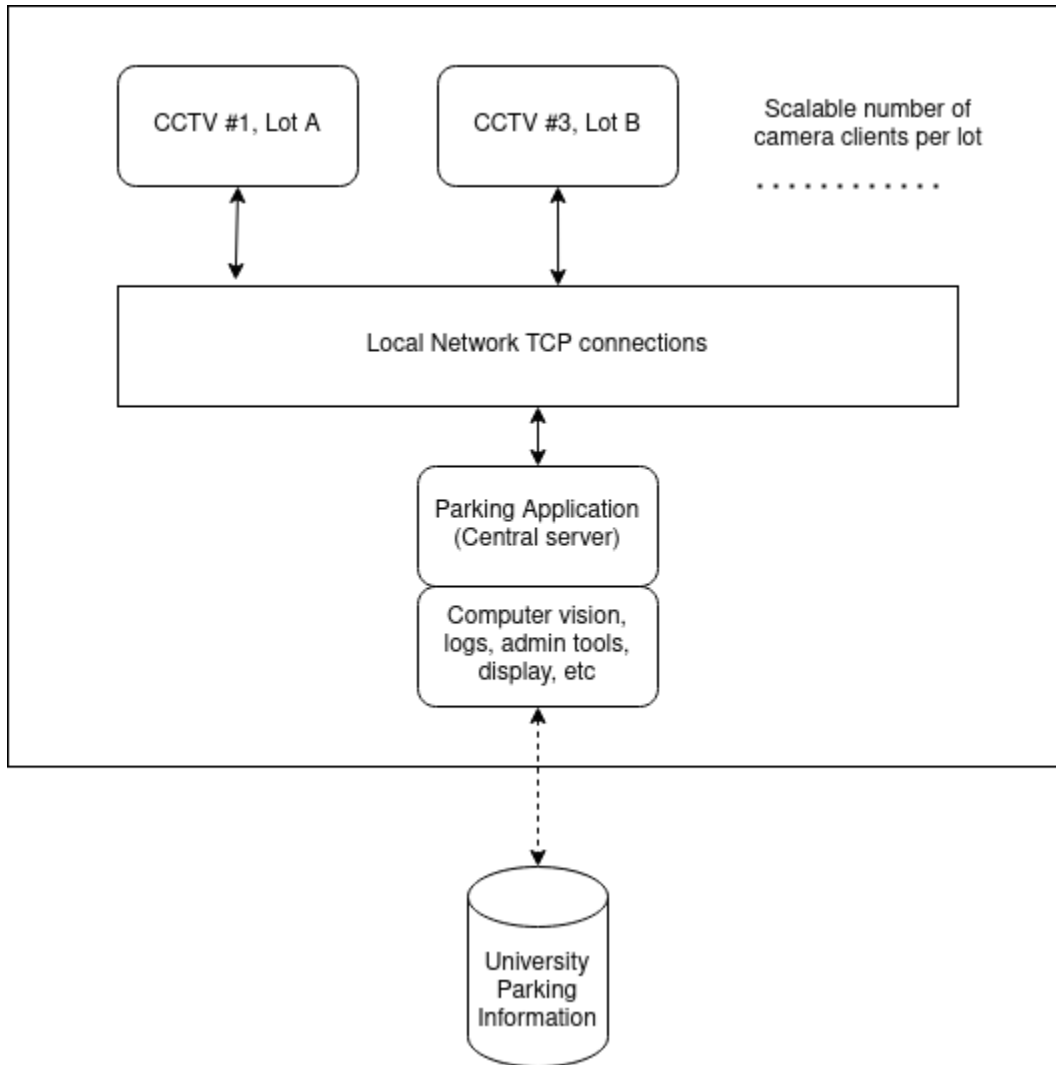
7. Sequence Diagram



8. Class Diagram



9. Architectural Design (Client-Server)



The above diagram is based on Figure 6.13 from the course textbook.

The bounding box defines the overall project. It is agnostic to the specific information provided by the given university, and can still operate without it.

10. Project Scope (also on Github)

Smart Parking Application

- Monitoring
 - Display occupancy information for each lot
 - tiled window displays for cameras at each lot
 - video feed, detected vehicles, car counts
- Administration
 - Ability to manually set/reset count at a lot
 - Access to log data and other relevant databases for parking information
- Logs
 - Collected at fixed intervals or whenever the count changes
 - Contains time stamps, occupancy count, license plates, parking pass ID's
 - Output an additional end-of-day summary log
- Parking Database
 - Need access to information to link license plates to parking pass types
 - Provided by university
- Object detection/tracking
 - Analyze video feed to detect and track cars and identify them within parking spaces
 - Cross-check license plates (or sticker passes) and parking spots
- Regulating Entry (if lot has a bar that opens/closes)
 - Only allow entry to lot/structure if the license plate is associated with the correct parking pass type for the lot/structure

3. Project Scheduling & Estimation

3.1 Project Scheduling:

Since our project requires us to implement hardware and softwares on a university campus the best time to do so would be during the summer when any university campus has the least amount of students faculty and traffic around the campus. This is helpful because if there is a need to close any parking lots or testing out equipment there won't be any major inconvenience to the university. While the majority of the work would be done during the summer, the project would start earlier. This schedule does not include weekends and the employees would be working 35 hour weeks with maybe some additional hours needed based on the timeline.

Phase 1: (February - May)

The first task of the project would involve scoping out the parking lots and structures to figure out which parking structures have the most amount of traffic, peak hours and average number of cars coming in and coming out of the parking space. This will be a labor intensive job since people would either have to physically go around and check different parking spots or monitor them through CCTV's if possible. Essential this should not take more than One and a Half month.

The second task that needs to be completed during this phase is checking the inventory of the campus. This includes checking for usable cameras in parking lots and structures, if they either need to be upgraded, repositioned or removed, and checking to see if the parking lots would need to have additional cameras and the most optimal position for each camera. This can be done simultaneously with the previous task but could take up to the end of May.

Finally we would also need to make sure how the cameras would communicate with the main server. Which would mean seeing if we can have them connected wired or wirelessly to the main server. This should not take more than a month and can be done during March.

Phase 2: (May - September)

This phase involves installing all of the new equipment, building our software and database connecting the server and training the ML model to recognise the different license plates and parking spots.

The Code based development would take around 3 months to build this also includes the time needed to train the ML Model. Additionally the justification for this is so that we can have a test

prototype for the university campus so that we can make sure our software works as we need to and have ample of time for the testing and bug fixing before the fall semester.

Installation of the server and additional hardware should not take more than a month and can be done simultaneously.

The final month of this phase would be left for testing out the software with the hardware and fixing bugs.

3.2 Cost Effort and Price Estimation:

The team has decided to use the Function Point Analysis Method for estimating the cost of the project.

	Function Category	Count	Complexity			Count x Complexity
			Simple	Average	Complex	
1	Number of User Input	10	3	4	6	40
2	Number of User Output	4	4	5	7	20
3	Number of User Queries	9	3	4	6	36
4	Number of Data files and Relational tables	16	7	10	15	160
5	Number of External interfaces	4	5	7	10	28
GFP						284

Inputs – enter license plate and check for a parking spot, camera input for parking spots, additional admin functionality, parking spot computer vision, and license plate recognition

Output – show parking spots, total spots per level, and cumulative parking space, map with parking location

Queries – Check if the parking pass is valid, and check if they can park in that location, show which space is available, computation of parking, and additional queries that can be made by the admin

Data files and Relational Tables – Main database holding license plate info, a database with parking spots per parking structure this database is separate for each parking lot and not combined, a daily log of parking, a relational table holding data of parking spots, location of parking space, and user authentication

External Interface – 1 screen per parking lot which runs the same code, 1 screen for admin, cameras communicating with the server, main server

We considered the entire complexity as average

Does the system require reliable backup and recovery?

5

Are data communications required?

5

Are there distributed processing functions?

1

Is performance critical?

5

Will the system run in an existing, heavily utilized operational environment?

3

Does the system require online data entry?

3

Does the online data entry require the input transaction to be built over multiple screens or operations?

1

Are the master files updated online?

0

Are the inputs, outputs, files, or inquiries complex?

1

Is the internal processing complex?

4

Is the code designed to be reusable?

4

Are conversion and installation included in the design?

3

Is the system designed for multiple installations in different organizations?

5

Is the application designed to facilitate change and ease of use by the user?

3

$$PCA = 0.65 + 0.01(5 + 5 + 1 + 5 + 3 + 3 + 1 + 0 + 1 + 4 + 4 + 3 + 5 + 3) = 1.08$$

$$FP = 284 \times 1.08 = 306.72$$

This calculation below is adapted based on the calculation in [1]

Assuming a team of four works at 4.2 hours/FP and there are 4 developers on the team.

$$\text{Total hours needed} = 4.5 \text{ hours/FP} \times 306.72 \text{ FP} = 1380.24 \text{ Hours}$$

So hours per developer would be = 345.06 hours

Based on section 3.1 if a team works 35 hours per week, the amount of time to finish the project would be $345.06 / 35 = 9.85$ weeks = 2.46 months

3.3 Estimated cost of Hardware Products:

For now the system would need a single server for for Storage and Specialized tasks which on average would cost around \$20000 and around \$732 per year to maintain the server. [2,3]

Cost of installing 1080p IP surveillance cameras would be around \$90 - \$200 dollars, the exact amount can only be determined after a detail inspection of the parking lot is done.

Outdoor Wifi Access point Routers can cost anywhere from \$1000 - \$3000 the price varies on its IP rating and quality, etc.

Good outdoor video screen panels can cost about \$1500 - \$2500 per panel and each parking lot entrance would need around 10 - 15 panels

Additional Hardware resources such as computers and other important office supplies can be shared with the university resources.

3.4 Estimate Cost of Software Products

Since we would be using all open-source software there wont be any cost for those. There might be a cost for the software license for the cameras, but those would be included with the purchase of the cameras.

3.5 Labor cost

According to LinkedIn and Indeed the average salary of a software engineer in Texas is \$94,168 which leads to an hourly pay of \$45.3. So to hire 4 software engineers who would work roughly 345 hours per person it would cost \$62,514. Additionally this system can be integrated into the universities Office of Information and Technology and therefore training cost can essentially become zero since the university already hires employees for the Office of Information and Technology, so essentially we can train those employees to operate and maintain the system. [4]

4. Test Plan

For testing units of our software, we plan to use black box unit testing techniques. We have chosen to use purely black box testing techniques because of their simplicity and straightforwardness while still being highly effective, which will aid in programming our project efficiently so it can be delivered on time. We will use both equivalence partition testing and boundary value analysis to ensure the code works properly. However, we will not use cause-effect testing since it is not needed for many of our units. We do not believe the inclusion of cause-effect testing into this plan would not provide enough benefit to outway the additional cost it would add to developing units.



As an example, code and tests for a single unit are provided. This unit is a basic method that makes a database call to update the occupancy information of a parking space. This method takes the lot designation as a string, the parking space number as an integer, and an occupancy value as a boolean. It must first call the database to get the number of parking spots in the specific lot. Then it checks if the entered spot number is within the range, then it updates the database. The return code will be -1 if the lot is invalid, -2 if the spot number is invalid, and -3 if the database update returns an error (ex. it is disconnected). It returns 0 on success.

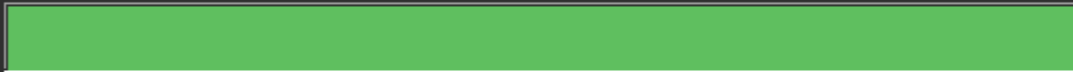
The equivalence partitions for the lot would be "PS4" and any other string, since this example method only works for PS4. For the parking space number, the equivalence partitions for this demo are 1-148 as a valid input partition, negative numbers and 0 as an invalid partition, and numbers above 148 as a positive partition. Our boundary values for this are 1 and 148. They should both work to pass the test. The method should work with both true and false, but as this change does not affect any logic in the method, it should not need to be tested. We will also test all combinations of valid and invalid inputs for each input variable.


On the following page is the code snippet for the tests and a screenshot of them all passing.


```
public void tests() {
    assertEquals("Test for all valid", 0, updateOccupancy("PS4",10,true));
    assertEquals("Test for invalid lot", -1,
        updateOccupancy("test",10,true));
    assertEquals("Test for valid lot, negative space", -2,
        updateOccupancy("PS4",-10,true));
    assertEquals("Test for valid lot, space too high", -2,
        updateOccupancy("PS4",1000,true));
    assertEquals("Test for valid space lower boundary", 0,
        updateOccupancy("PS4",1,true));
    assertEquals("Test for invalid space lower boundary", -2,
        updateOccupancy("PS4",0,true));
    assertEquals("Test for valid space upper boundary", 0,
        updateOccupancy("PS4",148,true));
    assertEquals("Test for invalid space upper boundary", -2,
        updateOccupancy("PS4",149,true));
}
```

Finished after 0.039 seconds

Runs: 1/1  Errors: 0  Failures: 0



▼  main [Runner: JUnit 4] (0.002 s)

 tests (0.002 s)

5. Existing Market

1. Smart Parking Vendors

Vendors integrate many features into a cohesive smart parking solution that provides parking lot usage information in any easy to use format, like an app [5,6]. They also offer additional features such as reservation systems, parking enforcement, and dynamic pricing that make for an efficient parking experience. The background research discusses these features within the context of vendor services like SPARK and IPark [5,6,7].

Vendor systems can be costly for several reasons. The first is fees for tasks like maintenance and support [6,7]. Vendor lock-in is another problem to consider, which is a serious issue. This means that important university infrastructure (parking) would be dependent on proprietary systems that the university ultimately has little control over. The fully integrated solutions that work “out of the box” may not be so appealing after all.

2. Open Source Computer Vision

There are simpler alternatives to vendor solutions. These alternatives may not work fully out of the box, but they do utilize robust projects like OpenCV for computer vision needs [8]. There are many similar open source projects that are capable of simple object detecting and tracking, but they aren’t integrated much further [6,7].

Computer vision is a great approach for solving the problem of detecting vehicles. At a high-level, It functions similarly to more direct means of detecting cars: microwave or pressure sensing. These are the choice approaches in mainstream smart parking vendor solutions [5]. A camera can accomplish the same task for a much lower cost since decent CCTV cameras cost \$30 at most [9]. You can place a single camera high up above an open lot and have it send frames to a server that analyzes the footage to detect which cars are parked instead of having many individual, more expensive pressure or microwave sensors on the ground [5]. This works as described, because cameras can capture footage that’s used to detect many more cars per unit than a microwave or pressure sensor, for example, which have more limited ranges.

Clearly, taking this approach requires hooking up all the parts yourself if you want to build a more complete smart parking suite like the one proposed by this project. It is worth noting that the university already has relevant databases and systems in place for parking information and enforcement, because lot usage information is provided in parking structures. The workers who go around to verify that students are parked in the correct spaces also have to use some kind of database to verify parking stickers. Even if a vendor solution is chosen, all these existing systems

would still have to be hooked up to whatever the vendor provides, so there's no avoiding the problem entirely.

Implementing such a system yourself is definitely feasible considering that small research teams and other groups have pulled off similar feats [6,7,8].

Comparing to Our Design

We do not specify the specific tools or libraries to be used for the project (out of scope), but the overall scheme is similar to both vendored smart parking and the simpler approaches that use open source projects and tools. While our project does not have all the features of a completely integrated smart parking system with reservation, dynamic pricing, enforcement features, and integration with apps, it does achieve more than some of the simple approaches outlined above.

The project is unique in the market due to both the lack of vendor solutions that use computer vision and lack of open source fully integrated parking systems. It combines the best of both approaches in a way that is cost-effective [7].

6. Conclusion & Evaluation

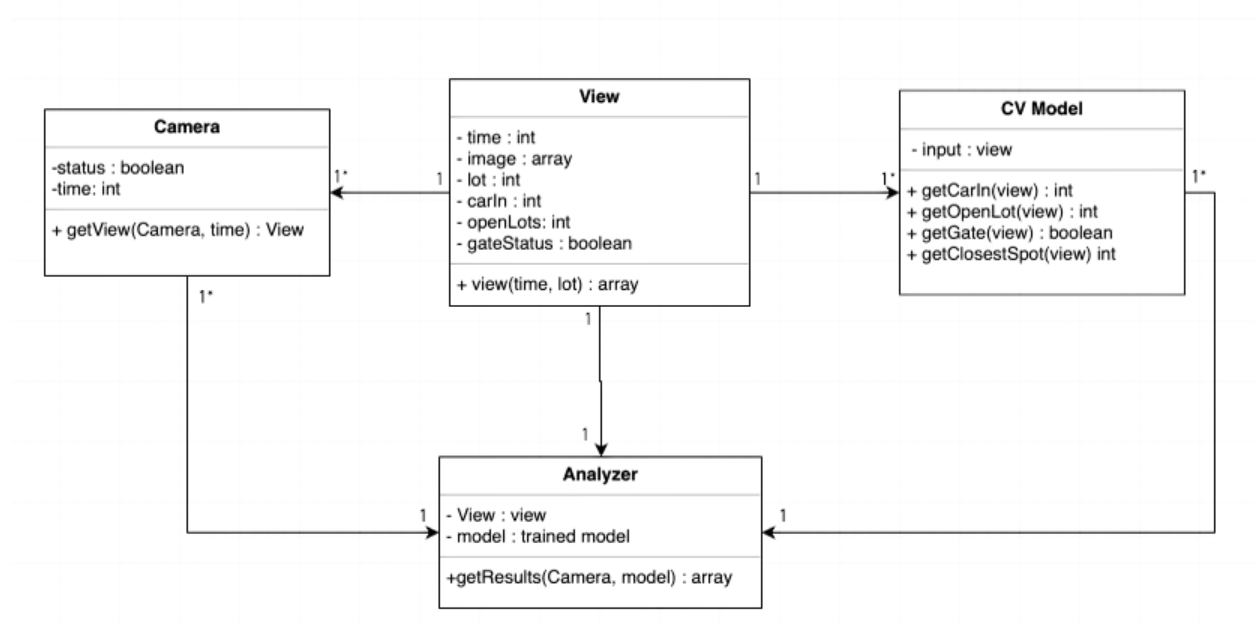
We have no major changes to report, since our design as it stands does not look like it will have significant problems with development. There are plenty of similar open source designs to pull from, although they aren't as integrated (refer to previous section "Existing Market"), so there aren't too many unknowns that could influence the cost/design. Our cost analysis is reasonable and the results strongly suggest that the project does not require major reevaluations or changes in order to be completed in a reasonable time frame.

The sections below discuss some important factors to consider that may warrant minor changes, but nothing too significant from the original design. Many of these points are somewhat out of scope, but still important to consider for deployment. All previous documentation is still relevant.

Minor Changes / Extra Factors (For Consideration)

1. Camera Issues
 - a. Reliability, Quality. If the incoming frames (server side) are too stuttery or pixelated, the system may fail to detect vehicles.
 - b. Policy issues (discussed in more detail below)
2. Conflicts in interfacing with university infrastructure (existing databases, websites, etc).
 - a. This can be problematic due to university policies relating to security and privacy (as noted in our non-functional requirements). Therefore, certain features may have to hold off until further approvals are made (such as controlling gates). Beginning iterations may have "stub" databases and interfaces in place of real ones that are managed by the university. Using the university's CCTV cameras would be ideal and would help bring cost down, but this may be denied for similar reasons.
3. **Using UDP instead of TCP** for sending frames
 - a. Each frame does not have to be sent and processed. Some loss is fine, which is the standard approach for video/audio over a networked connection. The speedup gains here would help in addressing some of the issues noted under "Camera Issues" related to video quality. Many socket programs default to TCP though and for an initial design, it likely won't have too much of a negative impact if this change isn't made.
4. Bigger change: New Class Diagram (more closely aligns with final design)
 - a. See next page

New Class Diagram



7. References


- [1] A. Alexander, *Estimating using FPA and hours/FP*.
https://alvinalexander.com/fpa/cost-estimating/Estimating_using_FPA_Hours_.shtml.
- [2] K. Mindanao, “How much does a server cost? [all factors explained],” *Intelligent Technical Solutions*, 28-Mar-2022. <https://www.itsasap.com/blog/server-cost>.
- [3] T. S. Team, “Total cost of ownership of servers,” *Sherweb*, 08-Jun-2022.
<https://www.sherweb.com/blog/cloud-server/total-cost-of-ownership-of-servers-iaas-vs-on-premises/>.
- [4] “Software engineer salary in Texas - indeed,” *Indeed*.
<https://www.indeed.com/career/software-engineer/salaries/TX>.
- [5] E. Polycarpou, L. Lambrinos, and E. Protopapadakis. “Smart Parking Solutions for Urban Areas.” IEEE 14th International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2013,
<https://doi.org/10.1109/WoWMoM.2013.6583499.1>
- [6] R. Grodi, D. Rawat, and F. Rios-Gutierrez. “Smart Parking: Parking Occupancy Monitoring and Visualization System for Smart Cities.” IEEE Southeastcon, pp. 1–5, July 2016,
<https://doi.org/10.1109/SECON.2016.7506721>.
- [7] K. Hassoune, W. Dachry, F. Moutaouakkil, and H. Medromi. “Smart Parking Systems: A Survey.” SITA, 2016. <https://doi.org/10.1109/SITA.2016.7772297>
- [8] J. Garcia. “People Counting with OpenCV, Python & Ubidots.” Ubidots.
<https://help.ubidots.com/en/articles/1674356-people-counting-systems-with-opencv-python-and-ubidots>
- [9] “GE CCTV camera,” *Amazon*, 1974. [Online]. Available:
<https://www.amazon.com/cctv-camera/s?k=cctv%2Bcamera>.

8. Slides

Smarter Parking for a University Campus

Patrick O'Boyle, David Hiser, Saumya Karia, Guarang Goel

Objective / Background

- Problem: Campus parking
 - Solution: Smart Parking
 - Approach: Computer Vision
- 
- Implementation: Demo that demonstrates the necessary features to help students and staff find parking spots
 - Scope: Our project focuses on the *back-end* to stay modular
 - Can send our data to any interface that displays where open spots are

UI Demo

- Multiplexed screen
 - Grid of camera views
- Timestamp
- Per-view occupancy counts
 - # of cars in spots
 - Out of total spots
- Linux Terminal (**admin**)
 - Read-Execute-Print Loop (like fdisk, sftp, etc)
 - Updating occupancy values
 - Log access
- No screen transitions
 - Main use case: **monitoring**

Lot A <timestamp>	Lot B	Lot B
<camera1 view>	<camera2 view>	<camera3 view>
Vehicles: 22/25	Vehicles: 13/30	Vehicles: 55/60
Lot B	Lot C	Lot C
<camera4 view>	<camera5 view>	<camera6 view>
Vehicles: 32/38	Vehicles: 31/35	Vehicles: 27/34

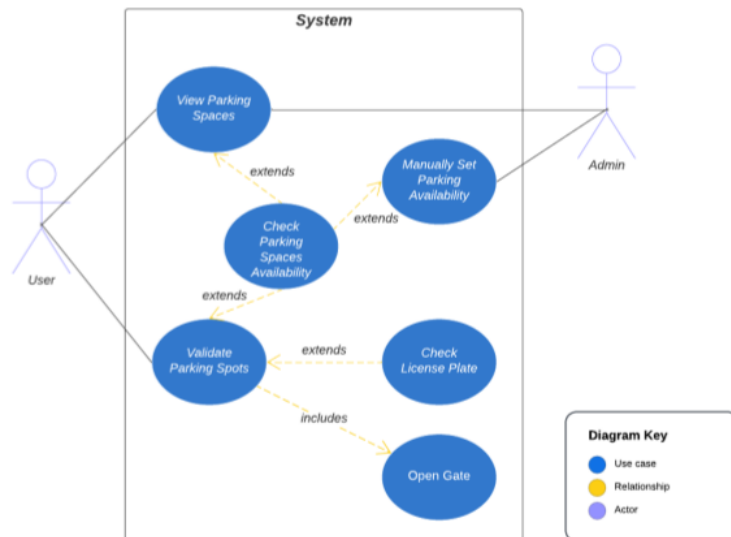
Fundamental Requirements

- Create End of Day Summarized Log Files
- Allow Admins to modify Parking Occupancy
- License Plate Validation
- Detect Available Parking Spots using Vision
- Validate Correct Parking based on Parking Permit Tier

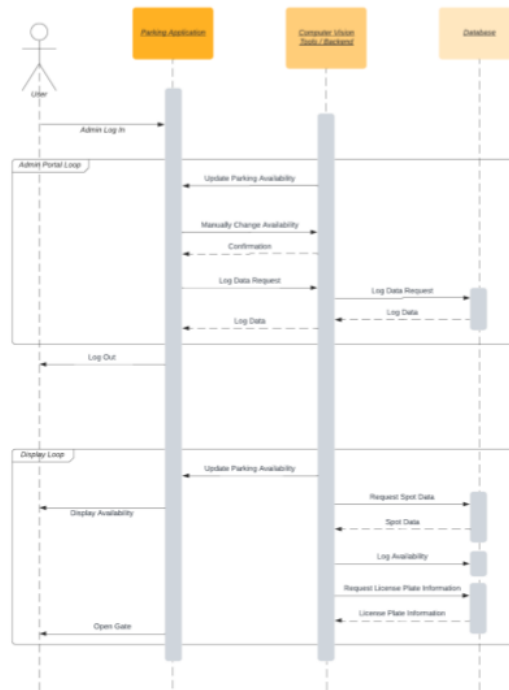
Non-Fundamental Requirements

- Most important Non-Functional Requirement
- Efficiency - System Response time > 5 seconds
- Security - Should pass OWASP Application Security Verification Standards
- Regulatory - Should Comply with EU GDPR, GLBA Act, PIPEDA, and CCPA
- Additional requirement included in the report with in-depth explanation

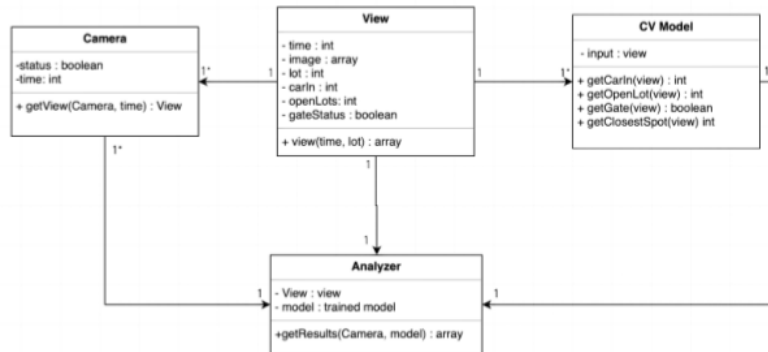
Use Case Diagram



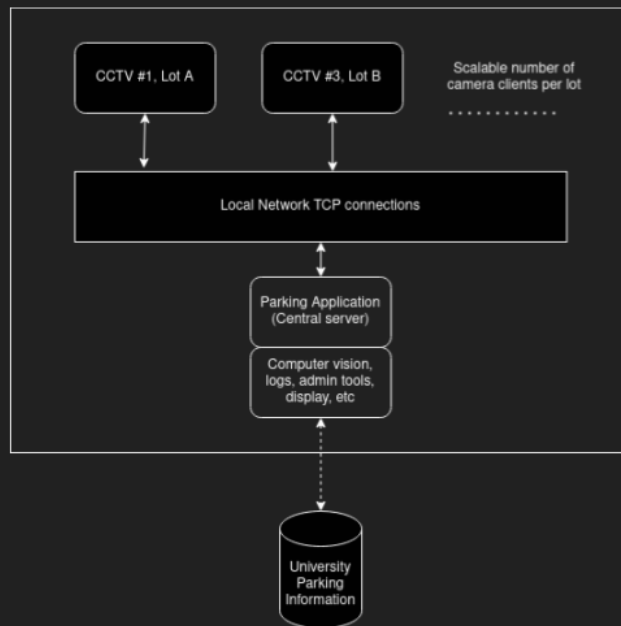
Sequence Diagram



Class Diagram



Architectural Design (Client-Server)



Project Scheduling

- Split into 2 Phases
- Phase 1 (February - May) :
 - Research, On-site Evaluations, Verify Inventory, Price Quotations and Planning
- Phase 2 (May - September):
 - Develop Software, Establish Database, Train Vision and ML Model
 - Installation of Hardware Components, Build and Verify Prototype, Bug Fixing, Staff Training

Cost Estimation

- Calculated using Function Point Estimation
- Estimated Function Points of about 307 points
- Required time to complete - 1380 hours
- At 35 hours/week with a 4 person team can be completed in 2.5 months *

* Software Development

Cost Estimation

Products	Cost
Labor (Software Developers + training)	~\$250,056
Hardware	~\$285,250
Server	~\$20,000
Total :	~ \$560,000

Hardware and Software Cost Estimation

- One time Hardware Cost based on research and Guestimation
- Costs includes Servers, Outdoor Cameras, WIFI Access point routers and Outdoor Video Screen Panels and Labor
- Majority of Software is Open Source

Labor Cost Estimation

- Average Texas Software Developer Pay - \$45.3 /hr
- Software can be integrated into University Office of Information and Technology
- University staff and Student Workers can be trained to use software at no additional cost.

Conclusions

- Comparisons
 - Many research papers that build low-cost simple smart parking systems with website and app interfaces (lacking features)
 - Industry smart parking systems (costly)
- Future Work / Changes
 - Integrate with interface(s) to display information
 - Addressing policy issues related to using a university's cameras and infrastructure
 - Use UDP-based protocol instead of TCP

10. Github

Same repo link as deliverable 1, included earlier in this document.

Contains:

- Your final project deliverable2 report
- Unit test code for a sample unit of your project
- Implementation code (if you have implemented your project)
- Presentation slides