**eParking Garage System**

**Team Members:** Jason Maxwell (Team Leader and Documentation), Caleb Ooten (Access System Coder), Timothy Harness (Database Coder), Brendan Harvey (Machine Learning and Documentation)

**Description**

This project is being developed to provide an electronic parking system that can allow for users to check in and out of a parking garage using their UTC ID or phone. Our main focus will be offering this service to universities that utilize parking garages in order to help facilitate a better parking experience for their students and easy integration into their current student database and parking systems.

**Goals and Objectives**

* Database Utilization: MariaDB
* Parking Space Tracking: OpenCV, Python, Web Camera
* Online Access System: Javascript, PHP, HTML, Swift
* Swipe Card System

**Non-technical Description**

Parking garages provide a great utilization of space when it comes to student parking on campus. Our system provides a way of integrating the IDs that are issued to students along with the current parking system that a campus may already use. Students will be able to swipe in when they enter the parking garage. This swipe will cause a database lookup to make sure that the student is allowed to use this parking area and also provide them with an area that they will be recommended to park in.

The focus of our system will be confirmed parking that utilizes cameras and a card swipe system to allow for confirmed parking. Students will also be able to take advantage of the website/app in order to see which parking garage is available for parking. This eases worries on whether or not they have to get to campus early in order to get the best spot.

Our secure and easy-to-use system will allow for students to better utilize their time by enhancing their parking experience. One single swipe or just opening of the website/app will ease their parking worries.

**Requirements**

|  |  |
| --- | --- |
| Preface | This document was designed to describe the depicted system in detail. This document is useful for those seeking further information on the functionality of the system, as well as those interested in implementing the system in order to implement an e-parking system for a parking garage. This document may be updated as the project continues. |
| Introduction | This project is being developed to provide an electronic parking system that can allow for users to check in and out of a parking garage using their UTC ID or phone. Our main focus will be offering this service to universities that utilize parking garages in order to help facilitate a better parking experience for their students and easy integration into their current student database and parking systems. |
| Glossary | **Cross Entropy** - Entropy is a measure of unpredictability of the  state, or equivalently, of its average information content. (Shannon  1948) Alternatively, the cross entropy between two probability  distributions p and q over the same underlying set of events  measures the average number of bits needed to identify an event  drawn from the set, if a coding scheme is used that is optimized for  an "unnatural" probability distribution q , rather than the "true"  distribution p . (Ian Goodfellow Yoshua Bengio 2017) Put another  way, if we think of a distribution as the tool we use to encode  symbols, then entropy measures the number of bits we'll need if we use the correct tool y . This is optimal, in that we can't encode the symbols using fewer bits on average. In contrast, cross entropy is the number of bits we'll need if we encode symbols from y using the wrong tool ŷ . This consists of encoding the i -th symbol using instead log( ŷ) of bits. We of 1 log( ) y 1 course still take the expected value to the true distribution y , since it's the distribution that truly generates the symbols:  H(y, ŷ) = Σ log(1/y ) = -Σ y log(ŷ) (DiPietro 2016)  The closer cross entropy is to 0, the closer your algorithm is to ideal  at, in the context of this system, accurately predicting the nature of  an object.  **Entropy** - (see Cross Entropy)  **Machine Learning** ​- an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed (Machine Learning).  **Python** ​- an open-source, high-level programming language known for its readability and support for multiple programming styles, and, due to its many libraries, a large range of applications (Python).  **JavaScript** - a high-level, interpreted programming language characterized as dynamic, weakly typed, prototype-based, and multi-paradigm. Considered one of the three technologies of the World Wide Web  **PHP** - a server-side scripting language designed for Web development but also used as a general-purpose programming language that can be embedded into HTML code or used in combination with various web tools.  **MariaDB** - a community-developed fork of the MySQL relational database management system intended to remain free under the GNU GPL.  **OpenCV** - a library of programming functions mainly aimed at real-time computer vision. |
| User Requirements Definition | The system will provide a way of integrating a college’s ID system with their parking system. It provides this integration by allowing for students to swipe into the parking garage using their student IDs, causing a database lookup to ensure that the student is allowed to park there and to show them the available parking spots. Confirmation of available parking spots will be made by camera(s) identifying whether a car is parking inside each spot within the parking garage by processing the video through a trained neural network and relaying this information to each parking lot’s database table.  Performance is important, as this needs to be functional as a live  monitoring system. The system should efficiently provide prompt  feedback for a rapid response time. It should be reliable, and  operate on minimal hardware. To aid in that, the system should also have light code that is functional on smaller devices, such as  Raspberry Pi’s. The system should be scalable to adapt to larger,  more complex hardware setups and increasingly specific user needs. It should ultimately be available and open source for its growth and development. |
| System Architecture | The architecture of this system is a remotely hosted server that contains the necessary software (MariaDB, PHP files, Javascript files, etc) and a laptop that contains the Python code that interacts with a webcam to provide the neural network the video feed and subsequent queries to the database on the remote server. |
| System Requirements Specifications | This system operates based on, at a bare minimum, a laptop, a camera, a card swipe reader, and an interface to access the internet.  Video data acquired from the camera is parsed by the trained protocol buffer files, using a Python script. This generates data related to the likelihood of the tracked object fitting a specific category, in this system: cars and motorcycles. Once a threshold is met for likelihood of it being a car or motorcycle, the script outputs a SQL query to the database and updates the relevant parking lot table. When the system does not detect a car or motorcycle in the parking spot, another SQL query will be sent to maintain the proper status of the database.  A web page which allows for the students (users) to check into the parking garage will be hosted on the remote server. A card swipe reader will interact with the webpage, reading the code attached to the user’s (student’s) ID card and communicating this with the attached user database. Once confirmation is received that the user (student) is allowed to park in the parking garage, an interactive map that is kept updated through queries sent to the database will be shown to the user (student), allowing them to see which parking lot would be best for them to park at. |
| Appendices | Hardware Requirements:  The system has been designed to run on most modern computers that are capable of processing video footage. For out prototype, we plan to use a Dell laptop with an external USB webcam. The laptop has the following specs:  Operating System: Windows 10 Pro 64-bit System Manufacturer: Dell Inc. System Model: Latitude E7440 BIOS: 1.7.1 Processor: Intel(R) Core(TM)i5-4300U CPU @ 1.90GHz, 2.50GHz Memory: 12.0 GB RAM |

**Requirement Specifications:**

|  |  |
| --- | --- |
| **Detection of Vehicle in Parking Space** | |
| Function | Accurately identify motor vehicles in the designated parking space. |
| Description | When an object is identified as a motor vehicle with a percentage likelihood greater than a set parameter, a query is sent to the database. |
| Inputs | Frame data. |
| Source | Frames come from video data recorded from a camera within the system. |
| Outputs | SQL query to MariaDB Database table for designated parking lot/space. |
| Destination | MariaDB database table for designated parking lot/space. |
| Action | If the object identified matches or exceeds the level of likelihood decided by a user set parameter, then an SQL query is sent to the database table for that particular parking lot and parking space. This query will designate that this parking space is occupied inside of this particular parking lot. |
| Requirements | The system, specifically the trained machine, must be installed properly. |
| Pre-condition | The parking space is empty and corresponding database table reflects as much. |
| Post-condition | The database entry will show that the spot is occupied which will be reflected on the interactive map for that parking lot. |
| Side effects | None. |

|  |  |
| --- | --- |
| **Successful Card Swipe at Entrance to Parking Garage** | |
| Function | Checking of the database to make sure User (student) is allowed to park in parking garage. |
| Description | When a user (student) swipes their ID at the parking garage entrance. |
| Inputs | ID card data |
| Source | User (student) ID |
| Outputs | 16 digit data code from User (student) ID |
| Destination | MariaDB Database |
| Action | User’s (student’s) 16 digit data code is checked against the database for the parking garage to check that they are allowed to use the parking facility. |
| Requirements | The system and a kiosk with an attached swipe card reader properly installed. |
| Pre-condition | The login screen with an empty login field will be shown on the kiosk screen. |
| Post-condition | The availability screen will be shown on the kiosk screen. |
| Side effects | None. |

|  |  |
| --- | --- |
| **Unsuccessful Card Swipe at Entrance to Parking Garage** | |
| Function | Checking of the database to make sure User (student) is allowed to park in parking garage. |
| Description | When a user (student) swipes their ID at the parking garage entrance and they are not allowed to park there. |
| Inputs | ID card data |
| Source | User (student) ID |
| Outputs | 16 digit data code from User (student) ID |
| Destination | MariaDB Database |
| Action | User’s (student’s) 16 digit data code is checked against the database for the parking garage to check that they are allowed to use the parking facility. If not, they are brought back to the login page. |
| Requirements | The system and a kiosk with an attached swipe card reader properly installed. |
| Pre-condition | The login page with an empty login field will be shown on the kiosk screen. |
| Post-condition | The login page with an empty login field and an error message that student is not allowed in the parking garage. |
| Side effects | None. |

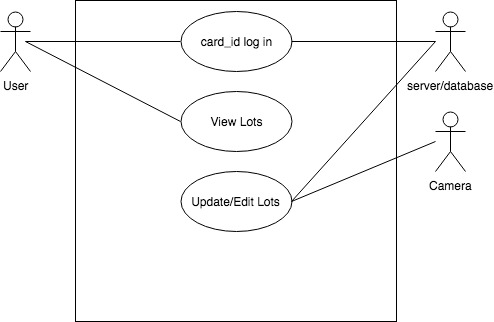
**Design Pattern**

**State Design**

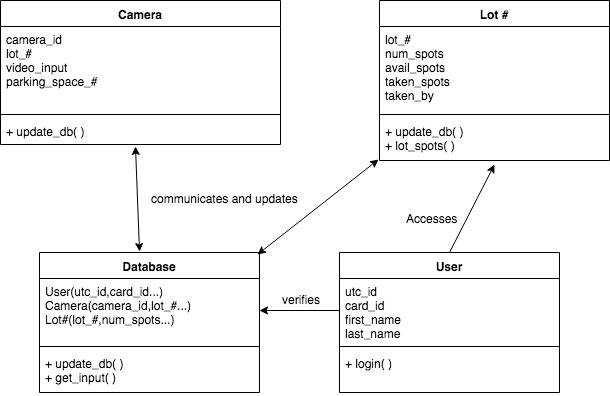
A State Design Pattern is useful for us in this application because ultimately, changing from a open parking spot to a taken parking spot is the most basic of our functionality. Switching states allows a different range of functionality to be available, and it will cause less overhead overall to implement instead of having to resort to extensive conditional statements. For example, say we have a problem with a parking lot screen not displaying properly. With a state pattern approach, we can interface with the specific parking lot table and camera(s), and more easily maintain the software. There are drawbacks - it makes the program somewhat inflexible and things are separated from one another. Adding other states would involve a significant code overhaul. We seek to employ it as a design philosophy without feeling chained specifically to every facet of the pattern.

**Diagrams:**

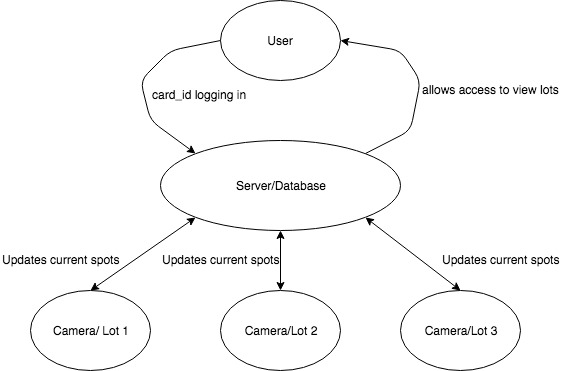
**Use Case Diagram:**



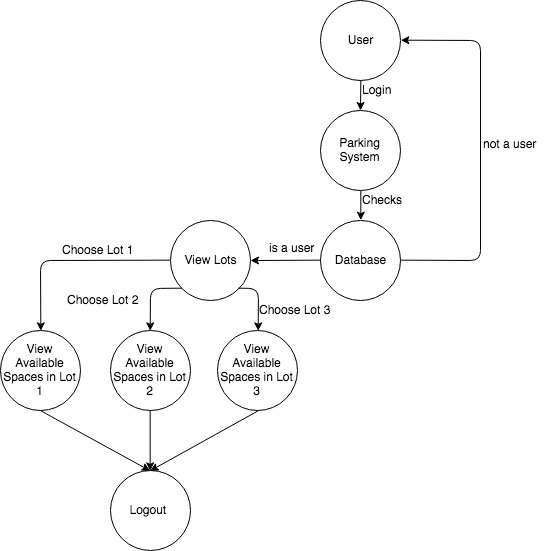
**UML Class Diagram:**



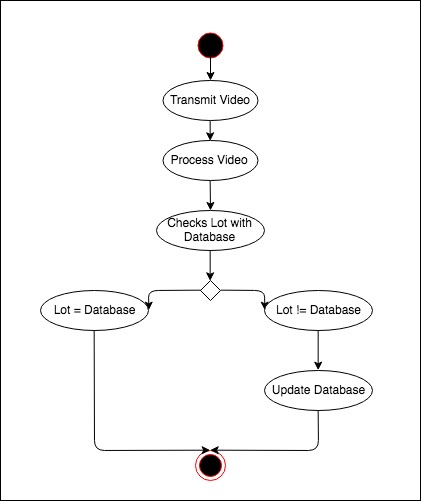
**Dataflow Diagram:**



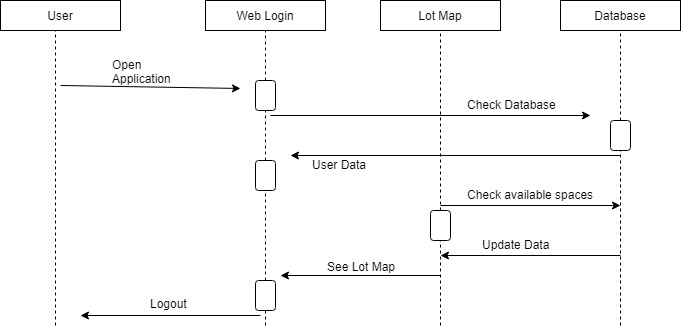
**State Diagram:**



**Activity Diagram:**



**Sequence Diagram**



**Implementation**

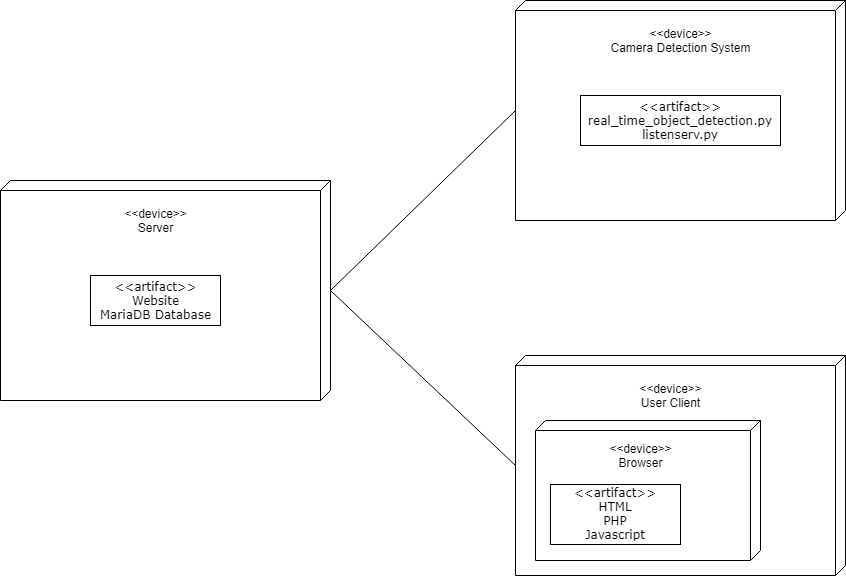
To implement the project we we first had to answer what the project was and what problem we were going to try to fix. Once we decided on UTC eParking, we had to decide on what the best answer to the problem was. We came up with using a camera and machine learning to help with the parking issues every student encounters. We then built the database and code around the needs of the student to make parking on campus stress-free.

We assigned roles within our project group to help with handling all the responsibilities and jobs that came with this project. We chose Jason as our team leader to help maintain and complete goals the project needed to reach at specific times. We also chose Jason and Brendan to write the code to process the video feed and use machine learning to specify the objects within a given space and update the database accordingly. We chose Caleb and Timothy to develop the web application and database. They developed a working application that accurately depicted the parking lots on campus and showed the spots available and unavailable within a lot. This allowed users to login and logout, view parking lot availabilities, and have a stress free experience.

The main improvement that needs to be made is stronger hardware to run the camera and process the video being uploaded. This was the biggest issue we encountered trying to process multiple feeds to multiple spaces within a lot and the computer would crash from the massive load.

**Deployment**

The project will be deployed by showing UTC the amount of time saved and satisfaction of students when using a demo of our project. Then we will begin with three lots to start and see how the campus reacts. Then we will eventually move to all lots giving the campus a full visual of parking on the campus as well as peak times and average wait time for another spot. This will allow students proper data to make decisions on how early they should get there and best options to park that fits their schedule and allows more time in the classroom rather than parking.



**Maintenance**

**Install Instructions**

Download and install Anaconda, Python v6, OpenCV, and MariaDB, and PHP version 7.2.7. Each website’s HTML file can be modified for the specific college that the system will be installed at. If you have any difficulty, please reference the wiki on their github pages or help pages on their website.

**User Manual**

Use of the system is designed to be easy to avoid the inconvenience or training, and the total manpower required to monitor multiple instances of the system if necessary. The remote server can be accessed at any time to make sure that the database is secure and operating correctly as well as communicating with the web camera monitoring the parking spaces.

**Administrator Manual**

Make sure your hardware meets the baseline requirements (camera, place to store it, way to

transmit it) then you are ready to install wherever you need to monitor. The requirements are

deliberately designed to be minimal such that an extensive system could be built relatively

inexpensively.

**Patch Process**

If any issues arise, contact support (the administrators of the github) and provide a full incident

report of what happened. You may actively put in requests online, and patches will roll out when developers alleviate the issue.