Barak Babayov, 206920100
Daniel Kozlov, 316523778
Nave Tal, 208518456

EyePark

What is it?

EyePark is a simple device that allows automatic detection of licence plate numbers of incoming cars and allowing car barriers to only let specific cars to pass. The system can be used both by a private user, allowing him/her to connect it to his/her garage door so it'll only open for specific cars, or by local municipalities or institutes with their own parking lots, allowing drivers to reserve a spot for themselves for a specific period of time for a small fee.

What is it made of?

Hardware:

Raspberry Pi 2:

We're using a Raspberry Pi 2 running a Windows 10 IoT Core operating system as the heart of our project, the Raspberry Pi will be connected to a usb camera and will run a UWP app.
According to input from the server (the Raspberry pi will be connected to the internet via a Wi-Fi dongle) it will handle opening and closing the gate connected to it.

Camera:

We're using a Microsoft Webcam: LifeCam Studio (connected to the Raspberry Pi via USB) to take photos of the surrounding and check whether an expected car is approaching which will be passed by the Raspberry Pi to the cloud.

Other:

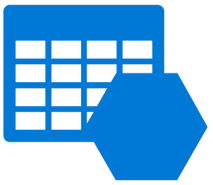
When put to practical use, the device will be connected to a real car barrier or garage door. The demo device only connects to a simple LED bulb instead, since barriers/doors are expensive and problematic to demonstrate with inside a university classroom.

Barak Babayov, 206920100
Daniel Kozlov, 316523778
Nave Tal, 208518456

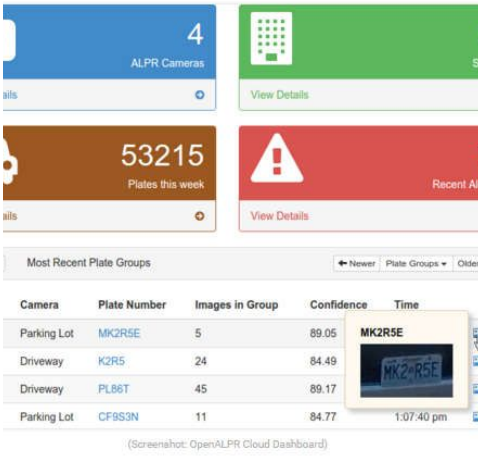
Software:

Server-side:

The server side is made up of 3 main components:
The **IOT Hub** is a component made by and hosted on Microsoft Azure. It receives and stores all messages sent from the different devices in the system.



The **Storage Account** is used to store the devices and reservations tables, which store all the needed information about the devices and all the reservations made by the clients.
What we call the "**Main server**" is the part of the server code that we wrote and would run in a **virtual machine** or as a **function app** on azure (see "What's Next?" section later). It pulls the messages from the IOT Hub, processes them and decides if it should signal the device if it should raise its barrier or just ignore it.
The processing of the images sent from the Raspberry Pi is done with **OpenALPR** (automatic license plate recognition) with its Cloud API which is a web service running in the cloud that analyzes images of vehicles and responds with license plate data, as well as vehicle color, make, model and body type.



(Screenshot: OpenALPR Cloud Dashboard)

Device-side:

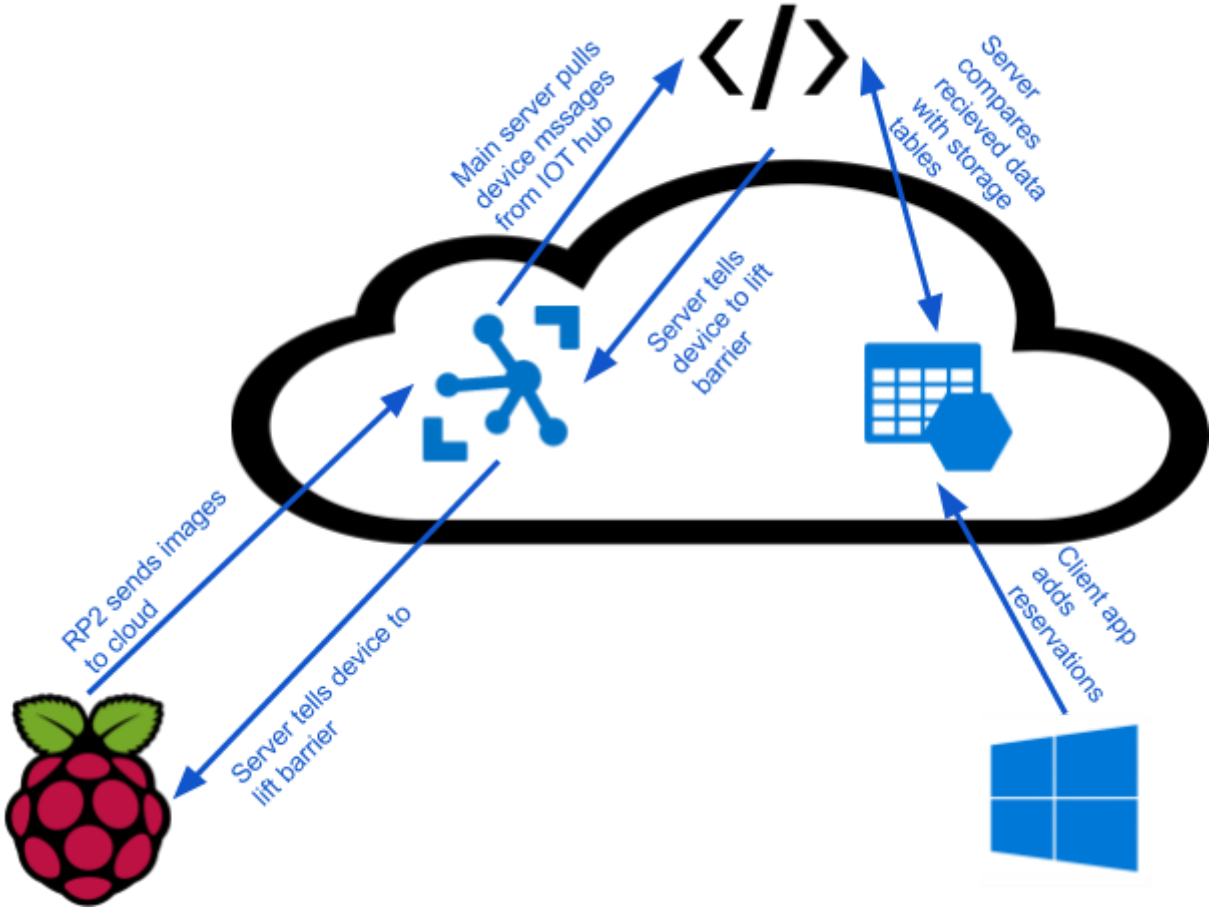
The device runs a simple program, written in c#, that takes pictures in equal, configurable intervals and sends it to the cloud (by converting it to a Base64 string, which is decoded on the server-side). Additionally, a separate task is running in the background, waiting for the server to send it a signal that means it found a matching reservation to the plate image it received, raising the barrier for a configurable period of time.

Client-side:

The client can interface with the system, seeing available parking locations and adding reservations, using our UWP app which finds the closest parking spot available given an address.

Barak Babayov, 206920100
Daniel Kozlov, 316523778
Nave Tal, 208518456

The whole picture:



Barak Babayov, 206920100
Daniel Kozlov, 316523778
Nave Tal, 208518456

What next?

While we are proud with the project we have made, we acknowledge it's not perfect. Here are a few ideas we came up with to improve upon this system in the future but didn't have time to implement:

Move the main server to Azure cloud

For ease of debugging and demonstrating, the "Main server" currently runs on a local machine rather than on the cloud. Thus, the first thing to do in order to improve this system would be to move this part to Azure as well, either by loading it to a virtual machine or by converting it to a function app, or a series of those apps, with the 2nd option allowing for much better performance of the server but requiring re-writing most of it's code.

Move the OCR to Azure cloud

Right now the project uses an external system to do the OCR, which is fine for the Demo but would not be fine if this system was to actually be implemented in the real world. Thus, the OCR part needs to run on Azure as well. In order to do that we have 2 options:
If the main server were to run in a virtual machine on Azure we could run the OCR system as well, since it's an open-source system and compiling it ourselves for this purpose is possible.
The other option would be to switch to using Azure's computer vision services. This option should also be compatible with a main server part that runs as a series of function apps, and we won't have to worry about updating it ourselves to the latest version. However, unlike the first option Azure's CV API is not optimized for reading license plate numbers, and thus will require to re-write large portions of the code, and may still produce inferior results.

Add support for private usage

The current version of the system only works for public parking spots reservations, but it shouldn't be hard to make a private version for personal garages, or maybe even make one version for both personal and public usage (Though we don't really like the 2nd idea since it could confuse users who want to use the system on both their garages and to reserve public parking spots.)