МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»
Кафедра «IT – инжиниринг»

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой
PhD, доцент
_____ Т.С. Картбаев
«___» _____ 2019г.

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Автоматическая система регистрации парковочных мест для ТОО «Almaz-Group»

Специальность 5В070400 – «Вычислительная техника и программное обеспечение»

Выполнил Бакиев Мухаммед Группа ВТ-15-2
Научный руководитель: к.т.н., доцент Мусапирова Г. Д.

Консультанты:
по экономической части: к.э.н., профессор _____ Ж. Г. Аренбаева
«13» мая 2019г.

по безопасности
жизнедеятельности: д.т.н., ст. преп. _____ Ш.Ш. Бекбасаров
«14» Мая 2019г.

по применению
вычислительной техники: ст. преп. _____ М.Н. Майкотов
«7» Мая 2019г.

Нормоконтролер: ст. преп. _____ Айткazина А. А.
«15» мая 2019г.

Рецензент: д.т.н., профессор, _____ Р. К. Ускенбаева
«___» _____ 2019г

Алматы 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»

Институт систем управления и информационных технологий

Кафедра IT-инжиниринг

Специальность 5В070400 – «Вычислительная техника и программное обеспечение»

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Бакиеву Мухаммеду

Тема проекта: Автоматическая система регистрации парковочных мест для ТОО «Almaz-Group»

Утверждена приказом по университету № 124 от «26» октября 2018 г.

Срок сдачи законченного проекта «24» мая 2019 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): Руководство системы менеджмента качества на предприятии; международные стандарты ИСО-9001, данные преддипломной практики.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта:

- аналитическая часть;
- проектная часть;
- практическая часть;
- экономическая часть;
- безопасность жизнедеятельности;
- приложение А. Техническое задание;
- приложение Б. Листинг программы;
- приложение В. Акты внедрения.

Перечень графического материала (с точным указанием обязательных чертежей): представлены 12 таблиц, 50 иллюстрации.

Основная рекомендуемая литература:

1 «Введение в модель данных SQL» / С.Д. Кузнецов – М.: Национальный Открытый Университет «ИНТУИТ», 2016.

2 Jatuporn C., Udomporn S., Satien T. Smart Parking: An Application of Optical Wireless Sensor Network / URL: https://ieeexplore.ieee.org/document/4090136

3 Faheem, S.A. Mahmud, G.M. Khan, M. Rahman and H. Zafar. Intelligent Car Parking Systems / URL: http://www.scielo.org.mx/pdf/jart/v11n5/v11n5a11.pdf

4 Mark Lutz / «Programming Python» - четвертый выпуск серии «Powerful Object-Oriented Programming», 2011.

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Экономическая часть	Аренбаева Ж.Г.	04.03.2019-13.05.2019	
Безопасность жизнедеятельности	Бекбасаров Ш.Ш.	04.03.2019-14.05.2019	
Программное обеспечение	Майкотов М.Н.	04.03.2019 07.05.2019	
Нормоконтролер	Айтказина А.А.	02.04.19-15.05.19	

ГРАФИК
подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Аналитическая часть	05.11.2018 – 21.12.2018	
Описание разрабатываемого проекта	07.01.2019 – 31.01.2019	
Проектирование системы	04.02.2019 – 12.03.2019	
Разработка системы	13.03.2019 – 12.04.2019	

Дата выдачи задания «19» октября 2018 г.

Заведующий кафедрой _____ Т.С. Картбаев

Научный руководитель проекта _____ Г.Д. Мусапирова

Задание принял к
исполнению студент Muhammed Bakyev М. Бакиев

Аңдатпа

Дипломдық жобаның тақырыбы: «Almaz-Group» ЖШС үшін тұрақ орындарын тіркеудің автоматты жүйесі.

Бұл дипломдық жобада автотұраққа кіретін және шығатын автокөліктерді бақылап, оларға есеп жүргізетін автотұрақ орындарына арналған жүйе құрастырылған. Сонымен қатар ол жүйе жобасының шығынын, құнын және экономикалық тұрғыдан тиімділігі есептелген, жоба аумағында еңбек шарттарын жақсарту бойынша іс шаралар ұсынылған.

Дипломдық жобаның мақсаттарына жету үшін Open Source компьютердің көру кітапханасы, Google Cloud Platform платформасы, MySQL, Python, бірыңғай платалық компьютер Raspberry Pi қолданылды.

Аннотация

Тема дипломного проекта: Автоматическая система регистрации парковочных мест для ТОО «Almaz-Group».

В данном дипломном проекте разрабатывается система для парковочных стоянок, цель которой заключается в том, чтобы вести учет и контроль за автомобилями, которые въезжают и выезжают с парковочной стоянки. Также был проведен экономический расчет затрат и стоимости разработки системы, оценка экономической целесообразности разрабатываемого проекта и были предложены мероприятия по улучшению условий труда в рамках реализуемого проекта.

Для достижения цели разрабатываемого дипломного проекта были применены такие технологии, как open-source библиотеки компьютерного виденья, облачная платформа Google Cloud Platform, MySQL, Python, одноплатный компьютер Raspberry Pi.

Annotation

The subject of this graduation project: Automatic car park registration system for LLP Almaz-Group.

This graduation project is about the development of a system, designed for logging and controlling cars coming in and out of parking lots. An economic analysis regarding the development costs was carried out, as well as an evaluation of the economic feasibility of the project being developed. Some measures to improve the working conditions within the ongoing project were also proposed.

Technologies, such as open-source computer vision libraries, Google Cloud Platform, MySQL, Python, single-board computer Raspberry Pi.

Содержание

Введение

Последние десятилетия наша страна активно развивается. Сейчас человечество находится в таком времени, где у каждого есть свой собственный компьютер, который можно в миг достать из кармана, и задать ему любой вопрос, на который он с легкостью ответит. Увеличилось количество автомобилей, и даже начали появляться электрокары, о которых люди только недавно могли мечтать. Количество роскоши увеличивается с каждым днем, и с тем же темпом растет вопрос о безопасности. В крупных городах, таких как Астана, Алматы и др. люди стали приобретать более дорогие, эксклюзивные автомобили, которые очень часто покупают в кредит, либо на накопленные средства. Поэтому хозяин автомобиля должен быть уверен в сохранности своего имущества.

Однако, с появлением большого количества автомобилей в Казахстане растет и уровень преступности: воровство различных ценностей из автомобиля, а также наружных авто аксессуаров, таких как фары, колеса, зеркала бокового вида и т. д. Поэтому люди хотят использовать парковочные площадки, которые должны предоставить определенный уровень безопасности и предохранить от таких преступлений. Многие жилищные комплексы и многоэтажные дома уже предоставляют частные парковочные гаражи, где резиденты комплекса могут припарковать машину, и с чувством безопасности и сохранности оставить свой автомобиль до следующей поездки. Но и такие мероприятия тоже имеют свои недостатки и уязвимости. И причина чаще всего — это «человеческий фактор». Например, в жилищном комплексе на пропуске чаще всего сидит человек-охранник, и если подъезжает неизвестный автомобиль, то не всегда проверка проходит строго, т. е. на парковку может попасть любой посторонний.

На сегодняшний день существует разные системы безопасности для парковочных площадок, например, сервис «Кузет», пропуск по специальным ID-картам или брелокам. В более престижных комплексах существуют системы распознавания автомобиля по встроенным микросхемам, которые с помощью радиосигналов передают необходимую информацию для идентификации транспортного средства.

Технология позволяет создавать более безопасные системы идентификации, которые достаточно трудно обойти. Только высококвалифицированные специалисты, которые являются профессионалами своего дела, могут обойти безопасность такой системы, и то не всегда.

С каждым днем электронные приборы становятся неотъемлемой частью жизни человека. Они внедрены во все отрасли деятельности человечества, и человеческий мир уже не сможет функционировать без них. Электронные устройства, а именно роботы, во многих случаях, считаются надежнее чем человек, у них отсутствует «человеческий фактор». Они выполняют только то, для чего их создали, и следуют строго запрограммированным в них инструкциям.

Если сравнить робота с человеком, то можно сразу же определить ключевые отличия, которые будут иметь большую выгоду при построении системы, которая будет обслуживать парковочную площадку, отвечать за ее безопасность, распределение автомобилей по местам и идентификацию автомобилей владельцев парковочных мест. Робот не устает, он будет работать круглосуточно, он всегда сосредоточен на своей задаче, его нельзя подкупить. Конечно же не обойтись без человеческого контроля, так как на сегодняшний день не существуют роботов, которые бы могли оценить критичность ситуации и принять необходимые меры. В любом случае необходимо иметь человека-супервайзера, который будет следить за всеми происшествиями, но робот будет выполнять основную часть трудоёмкой работы.

Целью данной дипломной работы является разработка автоматической системы регистрации и контроля парковочных мест жилых комплексов, бизнес центров и др. наземных, подземных паркингов с применением шлагбаума и стойки управления. Данная система будет идентифицировать и пропускать автомобили резидентов жилых комплексов, сотрудников предприятий, посетителей и т. д., вести учет о их прибытии/выбытии, а также регистрировать наличие припаркованного автомобиля. Система разрабатывается для компании ТОО «Almaz-Group».

1 Аналитический обзор современного состояния исследуемой проблемы

Вопрос о автоматизации процесса «паркинга» возник давно. На сегодняшний день существует разные методы и подходы автоматизации парковочных процессов, но нет единого «решения» всех проблем. Обстановка на парковочных стоянках может быть разной, в зависимости от места, предназначения и посещаемости. Поэтому владельцы парковочных стоянок ищут наибольшее подходящий метод решения присутствующих проблем, и таким образом определяют наилучший метод автоматизации.

Для автостоянки, которая находится на территории торгово-развлекательного центра, нет смысля устанавливать строгий контроль посещаемых автомобилей, т. к. клиентами и посетителям негде будет парковать автомобили. Соответственно будет снижаться количество посещающих, а также будет беспорядок на дорогах, из-за припаркованных, в неположенных местах, автомобилей. То же самое можно и сказать о жилых комплексах. Если на паркинг жилого комплекса будут заезжать все подряд, то самим резидентам негде будет парковаться. Таким образом падает уровень безопасности автомобилей жителей, и жители не будут уверены в сохранности своего имущества.

Для каждого вида предприятия, где имеется парковочная стоянка, нужно находить свой подход к автоматизации паркинга, который будет решать определенные задачи на данном паркинге. Нет единого решения, которое решит все проблемы.

1.1 История паркинга

С появлением первых автомобилей и растущим количеством их владельцев, возник и вопрос о его хранении. Основная масса владельцев данной роскоши парковали свои транспортные средства там, где им было угодно, и на сколько угодно. Автомобили парковали вдоль тротуара, на тротуаре, на пешеходных зонах, на дороге, и т. д. Причина этому была проста: инфраструктура парковок автомобилей еще не была внедрена в ежедневную жизнь обычного человека. Это привело к беспорядку на дорогах и в общественных местах [3]. Нужно было решить данную проблему как можно было скорее, т. к. уровень жизни становился выше и соответственно численность владельцев авто возрастал вместе с ним.

Чтобы решить проблему о парковке, на западе, в таких странах, как США и Великобритания были внедрены парковочные часы, которые по сути являются таймером, предназначенным для мониторинга времени парковки транспортного средства на обусловленном месте. Водитель должен был самостоятельно установить длительность своей остановки и внести необходимую оплату за установленное время. При проверке инспекторы парковочных зон могли просто посмотреть на наличие красного флажка на

парковочных часах и оставшегося времени на них. Данная система применяется по сей день в западных странах, несмотря на появление более эффективных и высокотехнологичных решений [4].

Частные парковочные зоны, которые были предназначенные для жителей квартирных комплексов, были очень простые. Территории были открытыми и без особой охраны. Парковочные гаражи находились под землей, где каждому жителю выделялось место и ключ от ворот. Безопасность подземного паркинга держалась полностью на доверии резидентов и владельцев территории [5].

Интеллектуальные системы управления парковками начали внедрять по всей Европе, Великобритания и Японии в начале 1970-х годов. Первоначальные системы предоставляли информацию, такую как статус доступности и/или количество свободных мест [1].

Платные парковки оплачивались через кассовые ларьки, а контроль осуществляли парковщики или охранники, т.е. когда автомобиль заезжает на парковочную площадку, его встречает человек, который его выпустит только при наличии квитанции, которую предоставляет кассовый ларек. Позже начали использовать шлагбаумы и гаражные ворота для более удобного и надежного контроля автомобилей.

1.2 Анализ существующих систем

В настоящее время существуют разные решения по обеспечению регистрации и контроля парковочных площадок и гаражей. Современные системы, по большей степени, являются частично автоматизированным. Такие системы в основном применяются на территории крупных корпорациях, бизнес центров, государственных учреждениях, торгово-развлекательных центрах и т. д.

По способу контроля и регистрации различают следующие системы:

– автоматизированные платежно-пропускные системы – самые распространенные системы, на сегодняшний день в Казахстане, и популярны среди парковок торгово-развлекательных центров, торговых центров, бизнес-центров, гостинец, аэропортов, вокзалов и т. п., где на въезде автостоянки стоит шлагбаум, и оплата производится на основе парковочных билетов. Такие системы называются «Системы автоматизированного контроля въезда на автомобильные стоянки» и решают такие задачи, как [7]:

– автоматизация контроля въезда/выезда транспортных средств на территорию автостоянки;

– автоматизация системы оплаты на основе билетов со штрих-кодом и бесконтактных карт доступа;

– удобный контроль и анализ финансовой деятельности парковочного комплекса;

– сокращение числа злоупотреблений со стороны клиентов и персонала автостоянки.

– автоматизированные системы частной парковки, которые чаще всего применяются бизнес-центрами и различными предприятиями. У таких систем существует много разных методов регистрации и контроля автомобилей, например: охранник, который сидит в кабинке и работает вручную, нажимая на кнопку, чтобы поднять шлагбаум; пропуск по пластиковым бесконтактным картам; системы с распознаванием биометрических характеристик посетителя; системы с распознавания государственного регистрационного номера автомобиля;

– автоматизированные «умные» системы паркинга, которые работают без шлагбаума [6]. Принцип работы очень простой (см. рисунок 1.2.1). Такие системы очень широко применяются в европейских странах, таких как Норвегия, Швеция, Дания, Германия;

Рисунок 1.2.1 – Автоматизированный «умный паркинг»

ParkLink. Автомобиль регистрируется по государственному регистрационному номеру при въезде и выезде. Оплата производится через мобильное приложение или терминал. Если же водитель покинул парковочную зону, не оплатив парковку, то выписывается штраф на владельца автотранспорта.

– автономные системы паркинга. Подобные системы используются только на государственных учреждениях, например на автостоянках аэропортов или милиции. Автомобиль регистрируется как в предыдущей системе, с помощью распознавания государственного регистрационного номера автомобиля, но отличие в том, что данный вид системы работает полностью автономно, т.е. после того, как автомобиль покидает зону парковки оформляется счет-фактура на владельца автомобиля и высылается ему в виде физического или электронного письма [8]. Таким способом нет необходимости заморочки со штрафами, если водитель не заплатил за парковку.

Все перечисленные системы имеют свои преимущества и недостатки, но по большей степени у каждой системы имеется своя сфера применения.

Автономные и «умные» паркинги облегчают контроль и менеджмент парковок государственных и корпоративных предприятий. Они значительно уменьшают количество «шагов» необходимых для завершения цикла паркинга. Водителям больше не нужно беспокоиться о парковочном билете, отсутствие денег при оплате парковки, или о том, что у них заканчивается время парковки. Им просто нужно заехать на парковку, припарковаться, выехать с парковки. Все остальные процессы, например, такие, как мониторинг парковки, контроль парковки, оплата парковки, и др. управляются роботом. Такие системы паркинга очень удобны для парковочных стоянок предприятий, у которых ежедневно более 10000 посещений, например, аэропорты, торгово-развлекательные центры, городская милиция, центры обслуживания населения и т. п. С такой системой минимизируется износ оборудования, создаются удобные условия для посетителей парковки и упрощается сам процесс паркинга. Конечно, есть и минусы, где самым главным является высокая стоимость внедрения, но, в долгосрочной перспективе, оно того стоит [9].

1.2.1 CirPark

CirPark Platform – программно-инженерная инфраструктура, в основе которой используется база программного обеспечения CirPark Scada (англ. Supervisory Control And Data Acquisition — диспетчерское управление и сбор данных). Платформа CirPark разработана компанией Circontrol, и включает в себя не только возможности контроля и учета автомобилей, но также комплекс инструментов для автоматизации и улучшения условий паркингов:

– интеллектуальная система навигации на парковке;

– светодиодная система освещения парковочных объектов;

– станции зарядки электромобилей для внутренних и наружных, парковочных объектов;

– система мониторинга потребления электроэнергии зарядными устройствами и любыми иными энергопотребителями;

– система поиска автотранспорта.

На данный момент Circontol – единственная компания в мире, которая имеет одну программно-аппаратную платформу для управления — системой навигации на парковке, светодиодным освещением и электрозарядными станциями для электромобилей. Данная компания самостоятельно разрабатывает и производит всю аппаратную и программную часть для своей платформы [10].

1.2.2 EasyPark

EasyPark Group – шведская компания, которая разработала систему «EasyPark». EasyPark – это аппаратное и облачное решение, цель которой

является цифровизация и автоматизация процесса паркинга. EasyPark предоставляет следующие возможности [11]:

- поиск автомобилей;
- учет и контроль парковочных зон;
- аналитику автостоянок;
- удобные решения оплаты паркинга;
- бронирование и регистрация парковочных мест;
- возможность коммуникации с посетителями парковок;
- снижение затрат на оборудование для паркингов.

1.2.3 ParkLink

ParkLink – «умная» система контроля и учета парковочных стоянок, разработанная скандинавской компанией АО «Bember». Данная система используется на парковочных стоянках в торговых центрах, государственных учреждениях, а также на уличных и крытых паркингах. Целью ParkLink является автоматизация паркинга в крупных городах и снижение затрат на расходы, связанные с существующими решениями для паркингов. Система предоставляет все необходимые инструменты для контроля, учета и оплаты паркинга и, по большей части, работает в автономном режиме [6].

1.2.4 Контроль паркингов жилых комплексов

Основная часть существующих «умных» систем для парковочных стоянок разрабатываются и применяются в корпоративных целях для торговых центров и государственных предприятий. Рынок систем паркингов для жилых комплексов, бизнес центров и небольших предприятий не большой. В основном используют решения, где при въезде на парковку стоит шлагбаум и управляющая кабинка, в которой может находиться охранник (см. рисунок 2).

На рисунке 2 видно въезд к подземной парковочной стоянке, которая предназначена для жителей жилого комплекса «Солнечная долина». Данный жилой комплекс считается «престижным», но безопасность парковочной стоянки не соответствует данному статусу. Весь контроль и учет ведется охранником, который, в большинстве случаев, не имеет понятия какие автомобили принадлежат резидентам жилого комплекса, и какие принадлежат гостям или просто посетителям, или даже злоумышленникам.

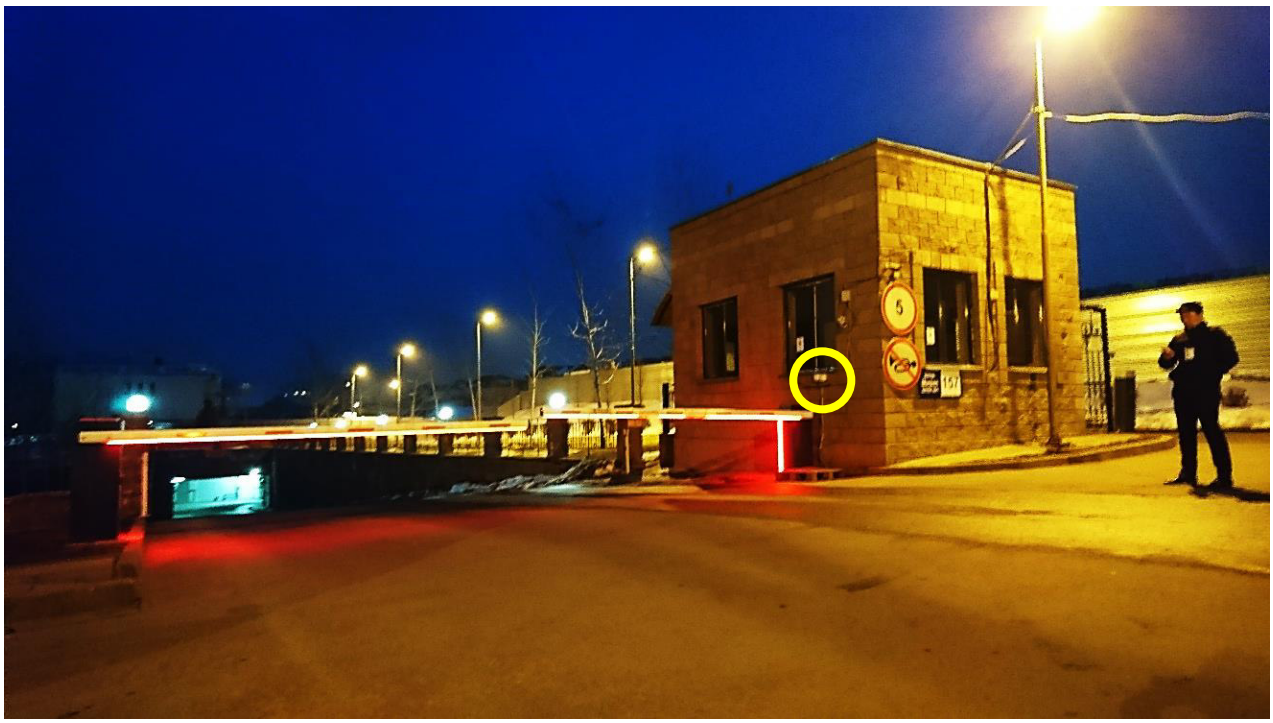

Рисунок 1.2.2 – Въезд на подземный паркинг жилого комплекса «Солнечная долина» (переключатели, которые управляют шлагбаум отмечены желтым кругом)

2 Описание проекта

На сегодняшний день проблема безопасности и автоматизации паркингов становится все более актуальной. Это важно для пользователей автостоянок, а также для их владельцев. В зависимости от того какая система установлена на паркинге будет формироваться сам процесс паркинга.

2.1 Выбор средств и технологий

В данном подразделе описываются какие технологии и аппаратные средства были использованы при разработке дипломного проекта, их ключевые достоинства и недостатки, почему были выбраны именно эти технологии, и за какую часть они отвечают.

2.1.1 Google Cloud Platform

Google Cloud Platform — предоставляемый компанией Google набор облачных служб, которые выполняются на той же самой инфраструктуре, которую Google использует для своих продуктов, предназначенных для конечных потребителей, таких как Google Search и YouTube. Кроме инструментов для управления, также предоставляется ряд модульных облачных служб, таких как облачные вычисления, хранение данных, анализ данных и машинное обучение [15].

В дипломном проекте Google Cloud Platform используется для распознавания текста в графических изображениях (Google Vision) и для услуги Google Cloud SQL. Google Cloud SQL – это услуга Google Cloud Platform, которая позволяет создать облачную базу данных с возможностью доступа с любой точки мира. Google Vision – услуга компьютерного зрения, позволяющая, на основе машинного обучения, создавать специализированные модели нейронных сетей, предназначенные для распознавания объектов в графических изображениях, и использовать готовые модели для таких задач, как распознавание текста, распознавание лиц, распознавание логотипов, и т. д.

В качестве аналогов существует Amazon Web Service, которые предоставляют похожие услуги с Google Cloud Platform.

Преимущества Google Cloud Platform в том, что Google имеет дата-центры по всему миру, в отличие от Amazon которые имеют дата-центры в Северной Америке, Европе и небольших регионах Азии.

2.1.2 BitBucket

Bitbucket («ведро битов») — веб-сервис для хостинга проектов и их совместной разработки, основанный на системе контроля версий Mercurial и Git. По назначению и основным предлагаемым функциям аналогичен GitHub, от которого отличается с одной стороны меньшей пользовательской базой, а с

другой, имеет определённые преимущества в плане размещения непубличных репозиториев — возможностью их бесплатного хостинга с ограничением на размер команды не более пяти человек и меньшая арендная плата при большем размере команды, а также управление правами доступа на уровне отдельных ветвей проекта [16].

При разработке любого программного продукта необходимо вести учет стадии разработки. Это можно осуществлять с помощью отдельных папок, которые находятся в корневой папке проекта. Но это не является самым оптимальным вариантом, так как очень просто можно запутаться в огромном количестве папок. Поэтому используется такой инструмент как BitBucket. Данный инструмент позволяет создавать разные версии стадии разработки и при необходимости возвращаться в любой версии без потери текущей версии. Также этот инструмент способствует улучшению рабочего процесса.

2.1.3 Python 3.6

Python – это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций [17].

Python 3.6 поддерживает огромное количество библиотек и модулей, которые используются при разработке разных проектов. Его синтаксис прост и имеет сходство с другими языками программирования высокого уровня. Также с пакетом Python идёт менеджер пакетов PIP. PIP – это удобный инструмент, который позволяет устанавливать, редактировать и вести учёт версии пакетов для Python.

2.1.4 PyCharm

PyCharm - Интегрированная среда разработки программного обеспечения специально для всех версий Python.

PyCharm Community Edition - бесплатный пакет самой умной и оптимизированной среды для разработки на языке Python на основе открытого кода.

PyCharm имеет такие функции как:

– умное автодополнение, инструменты для анализа кода, удобная навигация по файлам проекта, расширенные инструменты рефакторинга;
– возможность создания виртуальной среды для разработки проектов;
– возможность создания отдельных интерпретаторов для Python;
– удобное добавление и установление модулей с помощью PIP;
– инструменты тестирования и отладки;
– интеграция с системами управления и контроля версий Git и Mercurial.

2.1.5 OpenCV

OpenCV (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях [18].

В данном дипломном проекте OpenCV используется для обработки графических изображений. Данная библиотека удобна тем, что она имеет огромное сообщество программистов, которые поддерживают. Также имеется много литературы и большое количество информации в сети.

2.1.6 MySQL

MySQL – это свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle [19].

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы [19].

MySQL входит в состав облачных решений Google Cloud SQL. Данная услуга позволяет:

– очень просто создавать пользователей, базы данных, резервные копировании;

– вести журналирование всех операций;

– размещать сервер в любой точке мира для оптимальной связи с клиентом;

– подключить функцию автоматического резервного копирования, автоматического увеличения пространства памяти, цифровое журналирование;

– подключить автоматическое размещение базы данных на дата-центры Google, которые расположены по всему миру.

Google Cloud SQL позволяет пользователям и разработчикам подключаться к базе данных через облачный терминал либо интегрированный виртуальной машины. Также можно иметь подключение через удаленный доступ через внешний IP-адрес, SSH, SSL.

2.1.7 Raspberry PI 3 Model B

Raspberry Pi – одноплатный компьютер, то есть различные части компьютера, которые обычно располагаются на отдельных платах, здесь

представлены на одной. К тому же эта плата имеет относительно небольшой размер — примерно 8,5*5,5 см (примерно, как кредитная карта) [20].

Несмотря на небольшой размер, Raspberry Pi Model B имеет следующие характеристики:

– 1.4 ГГц четырехъядерный 64-разрядный процессор A53, основан на архитектуре ARM V8;

– графический процессор Broadcom Videocore-IV;

– ОЗУ, 1 Гб DDR2 SDRAM;

– сетевой чипсет Gigabit Ethernet, со встроенным 2.4ГГц и 5ГГц 802.11b/g/n/ac Wi-Fi;

– bluetooth 4.2, Bluetooth Low Energy (BLE);

– слот для SD-карты;

– 40 пинов для ввода/вывода.

Благодаря небольшому размеру Raspberry Pi можно интегрировать практически в любые проекты.

Операционная система Raspberry Pi называется «Rasbian». Данная операционная система основана на Debian, дистрибутив операционной системы Linux, и специально разработана для Raspberry Pi. Существует несколько версии данной операционной системы. Операционная система Rasbian включает в себя все необходимые инструменты для работы с Raspberry Pi. На Raspberry Pi можно также устанавливать другие версии Linux, включая Ubuntu, Debian и Archlinux.

В данном проекте Raspberry Pi служит как основной компьютер, который будет обрабатывать графические изображения, работать с базой данных и управлять шлагбаумом.

2.1.8 IP-камера Ubiquiti Unifi

IP-камера Ubiquiti Unifi по своим характеристикам отлично подходит к данному проекту, т. к. она способна передавать высокочастотный видео с частотой 30 кадров в секунду в HD, имеет низкую потребляемую мощность (4.5 Вт) и предназначена для наружного применения.

Основное предназначение камеры будет снимать снимок передней части автомобиля, где находится регистрационный номер, при въезде и выезде из паркинга.

2.1.9 Ультразвуковой дальномер HC-SR04

Ультразвуковой дальномер используется для обнаружения присутствия автомобиля. Работа данного модуля основана на принципе эхолокации. Модуль посылает ультразвуковой сигнал и принимает его отражение от объекта. Если измерить время между отправкой и получением импульса, то можно определить расстояние до препятствия.

Дальномер HC-SR04 может измерять длину до 5 метров. Он имеет четыре контакта:

- питание (5В);
- земля;
- триггер, для активации дальномера;
- эхо, для считывания полученных данных.

2.2 Постановка цели и задачи

Целью дипломной проекта заключается в разработке системы автоматической регистрации и контроля парковочных мест жилых комплексов, бизнес центров, наземных, подземных паркингов и иных территорий, требующих автоматизированный контроль и безопасность с применением шлагбаума и стойки управления.

Данная системы особенно удобная для подземного паркинга жилых комплексов. Пропуск на паркинг будет производиться строго по зарегистрированным автомобилям, что обеспечит определенный уровень безопасности. Если на паркинге имеются гостевые места, то при наличии свободных мест можно будет зарегистрировать автомобиль гостя, и он также получит пропуск к подземному паркингу.

В данной системе будет иметься база данных зарегистрированных автомобилей, которые будут допускаться к пропуску через шлагбаум. Система будет идентифицировать автомобиль по государственному регистрационному номеру автомобиля, с помощью видео камеры и сервиса Google Cloud API, и будет вести учет прибывших/выбывших автомобилей, данные которого будут записываться в базу данных.

Для реализации данного проекта необходимо осуществить следующие требования:

- анализ существующих аналогов систем управления парковками;
- определение основных требований к системе учета и регистрации.
- проектирование базы данных, программной части, аппаратной;
- выбор и обоснование технологий для разработки;
- выбор и обоснование комплектующих для сборки;
- разработка системы контроля и учета;
- настройка сервера и облачных услуг;
- обоснование экономической целесообразности разрабатываемого продукта;
- предложение мероприятий по улучшению условий труда в рамках реализуемого проекта.

3 Проектирование

В данной главе описывается процесс проектирования базы данных, программной и аппаратной части проекта.

3.1 Проектирование аппаратной части

Прежде чем приступать к сборке какой-либо аппаратуры нужно определиться со схемой подключения. Так как в данном проекте аппаратная часть является самой чувствительной к внешним факторами и для того, чтобы не переподключать контакты лишний раз, нужно тщательно продумать подключение.

Имеется пять основных компонента – это Raspberry Pi, сетевая камера, ультразвуковой дальномер, шлагбаум, и 4G WI-FI модем. Прежде чем физически расключать компоненты необходимо построить диаграмму, на которой будет отображаться схема подключения (см рисунок 3.1.1).

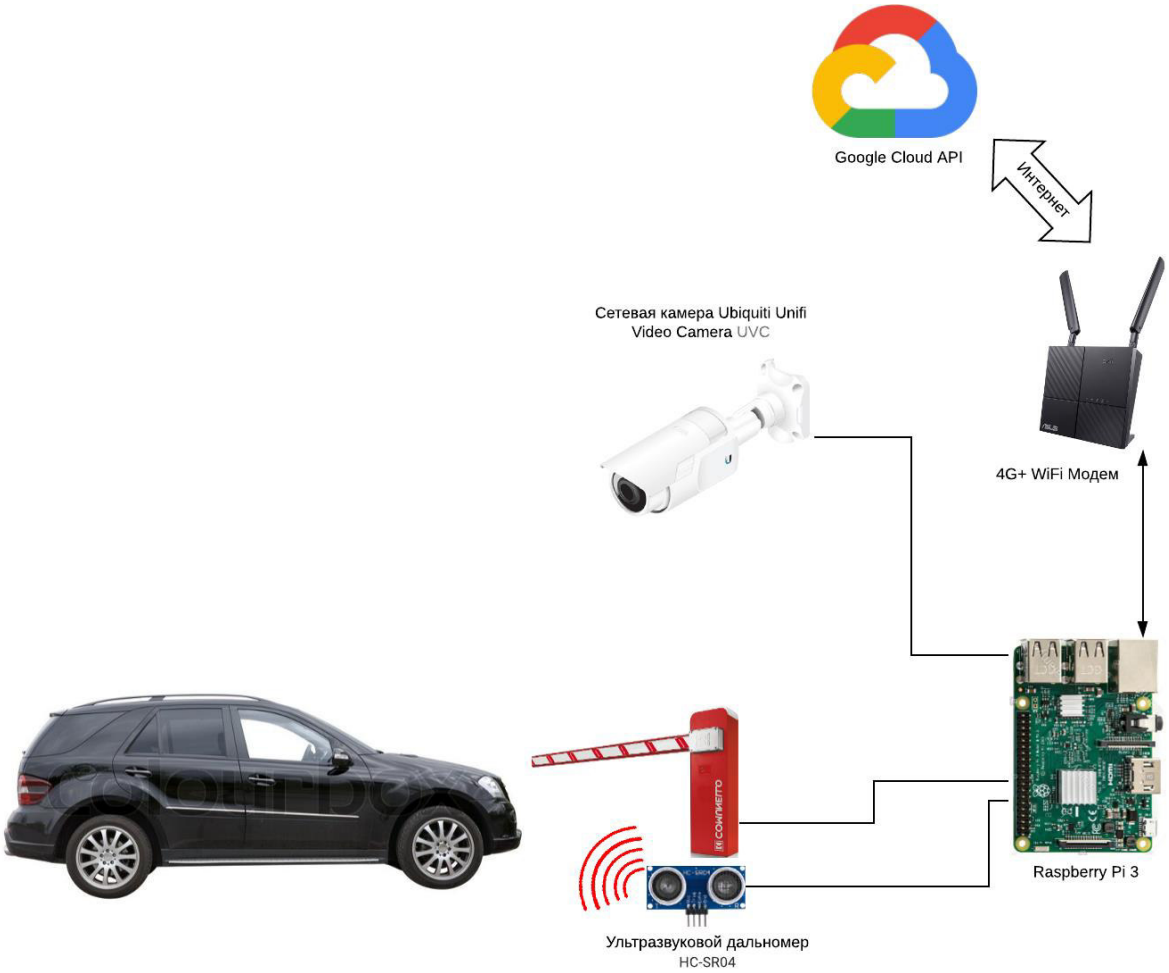

Рисунок 3.1.1 – Схема подключения компонентов

Как видно на рисунке 3.1.1 сетевая камера и модем подключены к портам Raspberry Pi. Шлагбаум и ультразвуковой дальномер подключены к интерфейсу ввода-вывода.

Когда автомобиль подъезжает в пределе двух метров к ультразвуковому дальномеру, фиксируется снимок переда автомобиля, с помощью сетевой камеры. Снимок далее обрабатывается и, с помощью модема, отправляется в облако для распознавания текста. При получении ответа начинается обработка входного текста. Ведется поиск регистрационного номера и при его наличии производится сверка с базой данных. Если регистрационный номер не находится в черном списке и имеется в базе данных то шлагбаум поднимается и автомобиль свободно проезжает на парковку. Как только автомобиль уходит поля зрения ультразвукового дальномера, шлагбаум опускается вниз, спустя двух секунд.

Так как имеется въезд и выезд на парковочную зону, то количество компонентов для расключения нужно удвоить, за исключением 4G WI-FI модема.

В случае подключение ультразвукового дальномера и шлагбаума к Raspberry Pi необходимо составить схему подключения выводов компонентов. Для этого создаются более подробная схема подключения элементов (см рисунок 3.1.2).

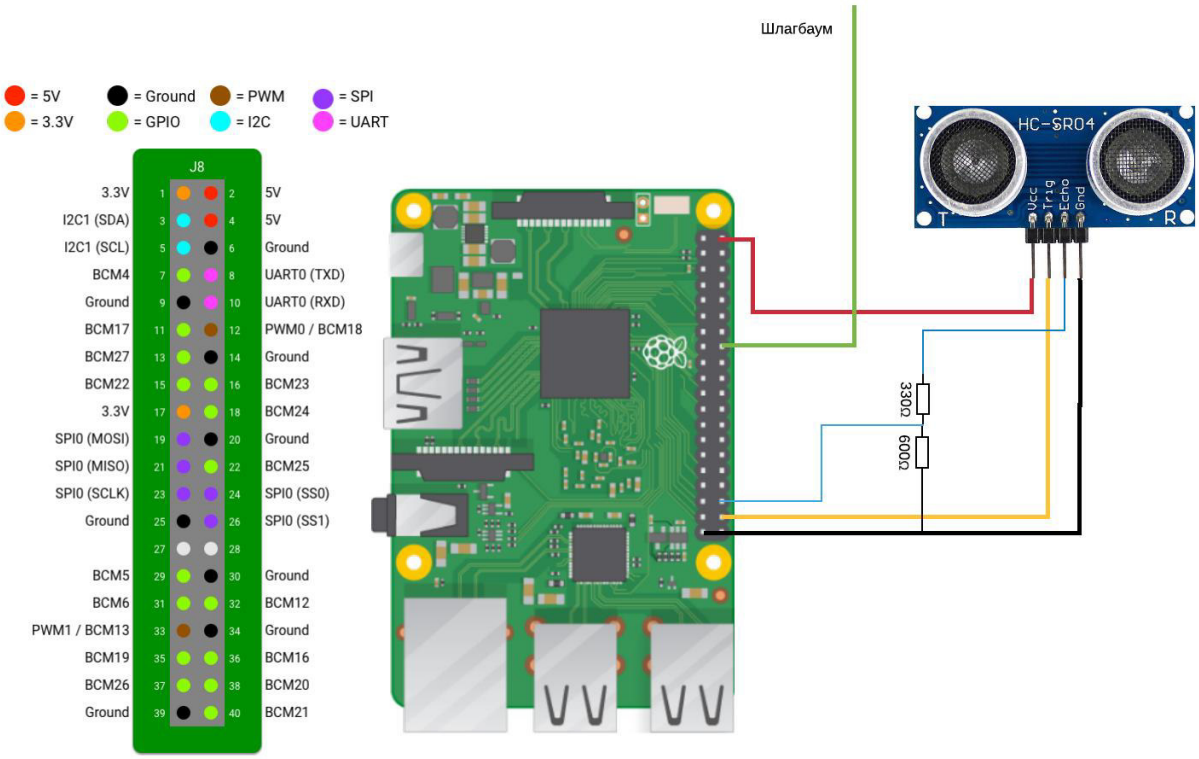

Рисунок 3.1.2 – Схема подключения шлагбаума и ультразвукового дальномера к Raspberry Pi

Как видно на рисунке 3.1.2 вывод шлагбаума подключен к контакту GPIO BCM23.

Ультразвуковой дальномер питается напрямую от источника питания Raspberry Pi, т. к. контакты 5В и земли подключены напрямую к источнику питания самого Raspberry Pi. Триггер ультразвукового дальномера подключён к GPIO BCM20, а эхо подключён к GPIO BCM16. Также на схеме видно, что имеется делитель напряжения между контактом «эхо» ультразвукового дальномера и контактом Raspberry Pi. Причина этому является то, что логика ультразвукового дальномера оперирует при 5 вольт, а логика Raspberry Pi оперирует при 3,3 вольт. Поэтому устанавливается делитель напряжения на этом контакте.

Чтобы рассчитать значение резисторов для делителя напряжения можно использовать следующую формулу (3.1):

$$В_{вых} = В_{вх} * \frac{R2}{R1+R2}, \quad (3.1)$$

где $В_{вых}$ – выходное напряжение;
$В_{вх}$ – входное напряжение;
$R1$ – резистор первый;
$R2$ – резистор второй;

Значение резистора R1 выбирается произвольно, а R2 вычисляется по формуле.

3.2 Проектирование программной части

Одним из самых важных этапов разработки программного обеспечения – это написание программного кода. Необходимо продумать последовательность действий, которые могут выполняться в программе. Логику программы можно часто расписать на листке бумаги, либо построить блок-схему.

На рисунке 3.2.1 представлена блок-схема алгоритма работы программы, которая будет размещаться на Raspberry Pi при въезде на паркинг. На рисунке отображен основной процесс программы проекта. Данный процесс будет работать в циклическом режиме.

На рисунке 3.2.2 представлена блок схема алгоритма работы программы, которая будет размещена на Raspberry Pi при выезде из паркинга. Данный алгоритма содержит меньше шагов, т. к. нет необходимости проверки.

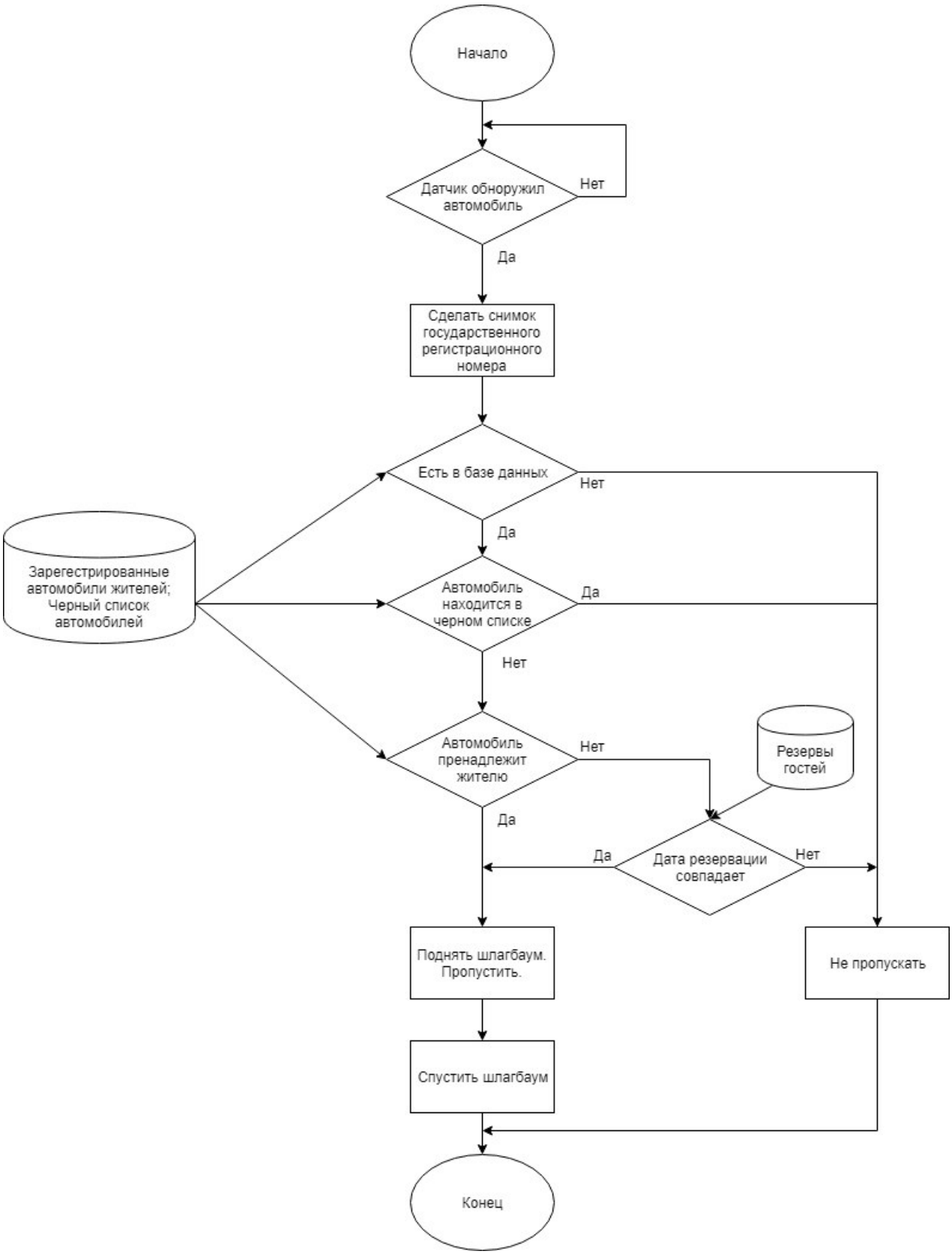

Рисунок 3.2.1 – Блок-схема алгоритма программы при въезде на парковочную стоянку

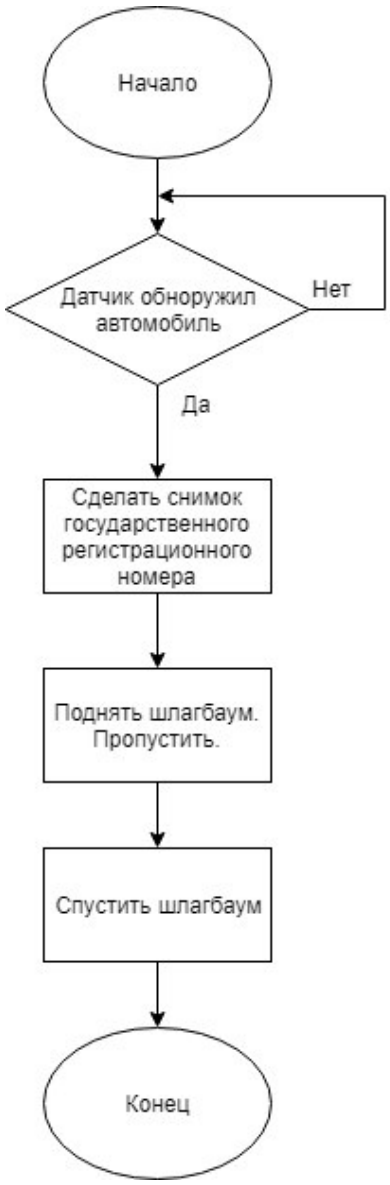

Рисунок 3.2.3 – Блок-схема алгоритма программы при выезде из парковочной стоянки

3.3 Проектирование базы данных

Очень часто при разработке программного продукта, где есть необходимость хранить информацию, используются базы данных. База данных может состоять из отдельных файлов, в которые записывается информация, либо как дополнительное программное обеспечение. В зависимости от разработчика выбирается проходящий метод хранения данных.

База данных может находиться локально либо в облаке. В обеих случаях есть свои преимущества и недостатки. Если база данных находится локально на устройстве, где работает приложение, то приложение будет иметь непосредственную связь с базой данных. В противном случае, если база данных находится в облаке, то приложению придётся общаться с базой данных через

интернет. Несмотря на то, что приложению придётся проходить дополнительные слои «интернета», преимущества облачной базы данных превосходят все преимущества локальных баз данных. Но это зависит и предназначения базы данных. Если нет необходимости хранить огромное количество данных или чувствительных данных, то локальные базы данных – это самый оптимальный вариант. Преимущества облачных решений заключается в том, что гарантируется доступность, надежность и безопасность.

В зависимости от того, как разработчик построить структуру базы данных будет зависеть производительность и работоспособность приложения и насколько просто ему будет получать из базы данных данные. Поэтому процесс проектирование базы — это данных довольно сложный процесс. Разработчику необходимо тщательно продумать логику и структуру базы данных.

При разработке автоматической системы регистрации парковочных мест необходимо создать базу данных для хранения данных пользователей автостоянок, резервации мест для гостей, загруженность парковочных мест и чёрный список автомобилей. Также будет вестись журналирование посещаемых автомобилей.

Для того чтобы не создавать новые базы данных для каждой точки применения, необходимо будет построить таблицы и логику так, чтобы можно было вести учёт разных парковочных стоянок в разных локациях.

Данные в базе данных нужно хранить в структурированном виде, иначе будет замедляться процесс выполнения запросов в базу данных. В базе данных необходимо хранить только те данные, которые действительно нужны для работы системы, т. к. наличие лишних данных способствует к засорению памяти, а также к замедлению работы самой базы данных. Если же есть необходимость добавить какой-либо вид данных в базу данных, то у каждой СУБД есть функционал, который позволяет любой момент времени добавить нужные поля в любую из таблиц базы данных. Данное свойство позволяет разработчику дополнять и усовершенствовать свою базу данных, но основная структура должна присутствовать при начальной проектировке базы данных.

В данном проекте используются в СУБД MySQL. MySQL имеет ряд недостатков и отсутствие функционала, который присутствуют других СУБД [22]:

– транзакции - позволяют объединить несколько SQL-запросов в одну единицу работы и в случае сбоя любого из запросов, входящего в эту единицу, выполнить откат, чтобы вернуть данные в исходное состояние. Но так, как в данном проекте нет для этого необходимости, то здесь много не теряется;

– триггеры - служат для автоматизации контроля за состоянием и работой базы данных. Триггер хранится в базе и срабатывает, когда происходит определенное событие;

– хранимые процедуры – это несколько SQL-команд, которые хранятся в базе данных под неким именем и в совокупности выполняют некую функцию.

При помощи хранимых процедур можно расширить синтаксис SQL так, что он будет похож на обычный язык программирования (например, Oracle PL/SQL);

– инструкция UNION – простыми словами, она объединяет вывод нескольких запросов в один, с возможностью исключить дубликаты строк;

– каскадное обновление данных - позволяет удалять и обновлять связанные данные. Например, при удалении из базы данных записи о пользователе из связанных таблиц автоматически удаляются все записи о его зарегистрированных автомобилях.

Все перечисленные недостатки можно симулировать в программном коде. Это займет немного больше времени, но много здесь не теряется.

Польза от MySQL, в данном проекте, заключается в ее преимуществах. Ключевое преимущество – это ее быстродействие. Это позволит довольно быстро пробегаться по таблицам для получения информации. Другие преимущества MySQL включают: безопасность, надежность, экономия ресурсов, переносимость.

Анализ предметной области является один из самых важных этапов разработки системы. Это позволяет выявить все данные, которые потребуется для правильной работы системы, и какие данные будут храниться в базе данных.

Для того чтобы позже не добавлять данные в базу данных необходимо поэтапно переходить к каждому функционалу системы от начала взаимодействия пользователем с системой и анализировать, какие данные будут нужны для реализации данного функционала.

Самое первое взаимодействие пользователь системы – это заполнение анкеты, где будут указываться данные о автомобиле пользователя. Для того чтобы пользователь имел пропуск на парковочную стоянку, системе нужно знать данные в автомобиле, а именно его регистрационный номер, т. к. идентификация автомобилей будет производиться по регистрационному номеру. Также необходимые данные такие как фамилия, имя, отчество пользователя, номер телефона и электронный адрес, для того чтобы можно было мыть связь с пользователем.

Для того чтобы система могла разбираться в какой зоне она работает, необходимо иметь данные о локации, где расположена парковочная стоянка.

Каждая парковочная стоянка имеет разные зоны, где могут парковаться автомобили. Если автостоянка крытая, то в каждой зоне скорее всего имеется несколько этажей, и на каждом этаже имеется определённое количество парковочных мест.

Каждое парковочное место может иметь определенное предназначение, например, место для инвалидов. Если парковочная стоянка принадлежит жилому комплексу или бизнес центру, то скорее всего у некоторых пользователей будут свои собственные парковочные места.

Парковочное место может иметь только два состояния: занятое или свободное.

Если намечается встреча и на парковочную стоянку должен прибыть автомобиль, которого нет в системе, то ему необходимо зарезервировать место. Для этого необходимо заполнить анкету со следующими данными: фамилия, имя, номер телефона и государственный регистрационный номер автомобиля посещаемого, а также назначить ему парковочное место.

У каждой автостоянки может также иметься чёрный список автомобилей, которые нельзя пропускать на автостоянку из-за определенной причины.

Для того, чтобы можно было вести учет и журналирование событий на автостоянке, необходимо иметь данные о въезде и выезде автомобилей с автостоянки, а также о наличии припаркованных автомобилей на стоянке и на каком парковочном месте они находятся.

3.3.1 ER-диаграмма

Собрав все проанализированные данные, можно построить диаграмму «сущность-связь» (ER-диаграмма) (см. рисунок 3.3.1).

ER-диаграмма – это тип блок-схемы, которая отображает как сущности, такие как автомобили, объекты или концепции, относится друг с другом в пределах определенной системы. ER-диаграммы используется для проектирования или отладки реляционных баз данных в области разработки программного обеспечения, информационных систем для бизнеса, образования и исследований. Также известные как ERD или ER-модели, они используют определенный набор символов, таких как прямоугольники, ромбы, овалы и соединительные линии, чтобы изобразить взаимосвязь сущностей, отношений и их атрибутов [12].

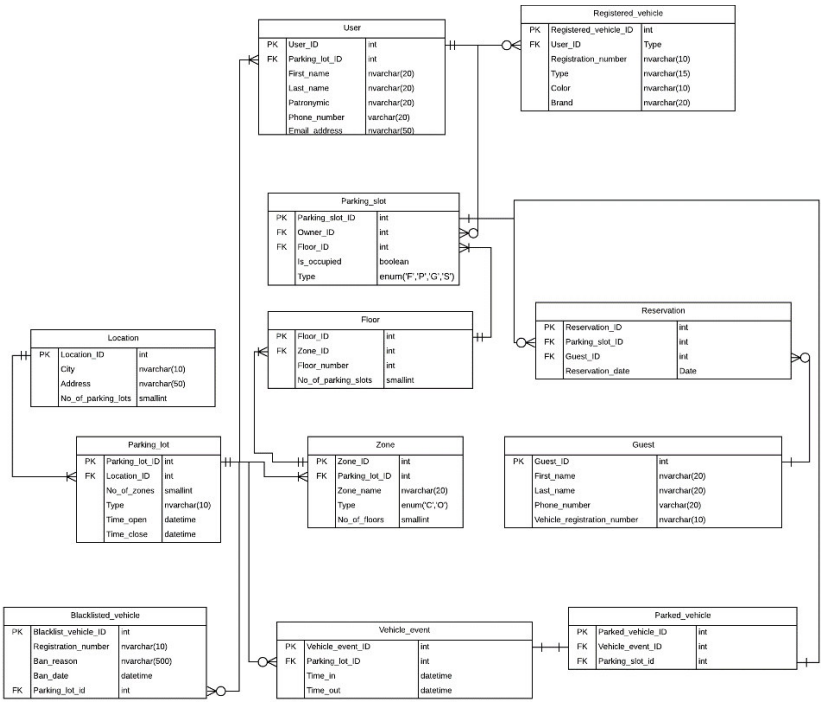

Рисунок 3.3.1 – ER-диаграмма, на основе которой строится база данных

3.4 Создание базы данных

В данном подразделе будут приведены запросы, которые применялись для создания таблиц базы данных, а также краткое пояснение к каждой таблице.

Первая таблица базы данных имеет название «Location» (локация). Эта таблица будет хранить в себе данные, связанные с определенной локацией, где находится автостоянка: город, адрес, и сколько автостоянок имеется по заданному адресу (см. рисунок 3.4.1).

```
CREATE TABLE `location` (
  `location_id` int NOT NULL AUTO_INCREMENT,
  `city` nvarchar(10) NOT NULL,
  `address` nvarchar(50) NOT NULL,
  `no_of_parking_lots` smallint,
  PRIMARY KEY (`location_id`)
);
```

Рисунок 3.4.1 – Запрос создания таблицы «Location»

Следующая таблица имеет название «Blacklisted vehicles» (черный список транспортных средств). Основное предназначение таблица хранить данные о автомобилях, которые нельзя пропускать на территорию парковочной стоянки. Таблица будет иметь такие поля как: регистрационный номер автомобиля, причина внесения в чёрный список и дата внесения в чёрный список (см. рисунок 3.4.2).

```
CREATE TABLE `blacklisted_vehicle` (
  `blacklist_vehicle_id` int NOT NULL AUTO_INCREMENT,
  `parking_lot_id` int NOT NULL,
  `registration_number` nvarchar(10) NOT NULL,
  `ban_reason` nvarchar(500) NOT NULL,
  `ban_date` datetime NOT NULL,
  PRIMARY KEY (`blacklist_vehicle_id`),
  FOREIGN KEY (parking_lot_id) REFERENCES parking_lot(parking_lot_id)
);
```

Рисунок 3.4.2 – Запрос создания таблицы «Blacklisted vehicle»

Третья таблица базы данных называется «Parking lot» (парковочная стоянка). В данной таблице имеются данные, которые описывают определенную парковочную стоянку. В ней будут такие данные, как количество зон, вид парковочной стоянки, время открытия, и время закрытия (см. рисунок 3.4.3).

```
CREATE TABLE `parking_lot` (
  `parking_lot_id` int NOT NULL AUTO_INCREMENT,
  `location_id` int NOT NULL,
  `no_of_zones` smallint,
  `type` nvarchar(10) NOT NULL,
  `time_open` datetime,
  `time_close` datetime,
  PRIMARY KEY (`Parking_lot_ID`),
  FOREIGN KEY (`location_id`) REFERENCES location(location_id)
);
```

Рисунок 3.4.3 – Запрос создания таблицы «Parking lot»

Четвертая таблица называется «Zone» (зона). Эта таблица будет хранить в себе все данные относительно определенной зоны на парковочной стоянке: название зоны, вид зоны, и количество этажей (см. рисунок 3.4.4).

```
CREATE TABLE `zone` (
  `zone_id` int NOT NULL AUTO_INCREMENT,
  `parking_lot_id` int NOT NULL,
  `zone_name` nvarchar(20) NOT NULL,
  `type` enum('C','O') NOT NULL,
  `no_of_floors` smallint,
  PRIMARY KEY (`zone_id`),
  FOREIGN KEY (`parking_lot_id`) REFERENCES parking_lot(parking_lot_id)
);
```

Рисунок 3.4.4 – Запрос создания таблицы «Zone»

Следующая таблица называется «Floor» (этаж). Данная таблица будет содержать информацию определенного этажа, его номер и количество парковочных мест (см. рисунок 3.4.5).

```
CREATE TABLE `floor` (
  `floor_id` int NOT NULL AUTO_INCREMENT,
  `zone_id` int NOT NULL,
  `floor_number` int NOT NULL,
  `no_of_parking_slots` smallint,
  PRIMARY KEY (`Floor_ID`),
  FOREIGN KEy (`Zone_ID`) REFERENCES zone(zone_id
);
```

Рисунок 3.4.5 – Запрос создания таблицы «Floor»

Следующая таблица базы данных называется «User» (пользователь). Это одна из основных таблиц базы данных т. к. в ней будет храниться информация

пользователя. Пользователь имеет: фамилия, имя, отчество, номер телефона и почтового адрес (см. рисунок 3.4.6).

```
CREATE TABLE `user` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `parking_lot_id` int NOT NULL,
  `first_name` nvarchar(20) NOT NULL,
  `last_name` nvarchar(20) NOT NULL,
  `patronymic` nvarchar(20),
  `phone_number` varchar(20) NOT NULL,
  `email_address` nvarchar(50) NOT NULL,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (`paring_lot_id`) REFERENCES parking_lot(parking_lot_id)
);
```

Рисунок 3.4.6 – Запрос создания таблицы «User»

Седьмая таблица связана с предыдущей таблицей. Таблица называется «Registered vehicle» (зарегистрированное транспортное средство) и будет хранить в себе основные характеристики автомобилей пользователей: государственный регистрационный номер, тип кузова, цвет, и марку транспортного средста (см. рисунок 3.4.7).

```
CREATE TABLE `registered_vehicle` (
  `registered_vehicle_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `registration_number` nvarchar(10) NOT NULL,
  `type` nvarchar(15) NOT NULL,
  `color` nvarchar(10) NOT NULL,
  `brand` nvarchar(20) NOT NULL,
  temporary bit NOT NULL,
  PRIMARY KEY (`registered_vehicle_id`),
  FOREIGN KEY (`user_id`) REFERENCES user(user_id)
);
```

Рисунок 3.4.7 – Запрос создания таблицы «Registered vehicle»

Таблица, с названием «Parking slot» (парковочное место) будет хранить в себе данные парковочных местах: кто владелец парковочного места, если он имеется, на каком этаже расположено парковочное место, занято или свободное парковочное место, тип парковочного места (см. рисунок 3.4.8).

Для резервации мест для посетителей или гостей необходимо создать таблицу «Guest» (гость). Эта таблица будет хранить в себе информации о посетителях: имя, фамилия, номер телефона и регистрационный номер автомобиля, на котором гость приедет (см. рисунок 3.4.9).

```
CREATE TABLE `parking_slot` (
  `parking_slot_id` int NOT NULL AUTO_INCREMENT,
  `owner_id` int ,
  `floor_id` int NOT NULL,
  `is_occupied` boolean NOT NULL,
  `type` enum('F','P','G','S') NOT NULL,
  PRIMARY KEY (`parking_slot_id`),
  FOREIGN KEY (`owner`) REFERENCES user(user_id),
  FOREIGN KEY (`floor_id`) REFERENCES floor(floor_id)
);
```

Рисунок 3.4.8 – Запрос создания таблицы «Parking slot»

```
CREATE TABLE `guest` (
  `guest_id` int NOT NULL AUTO_INCREMENT,
  `first_name` nvarchar(20) NOT NULL,
  `last_name` nvarchar(20) NOT NULL,
  `phone_number` varchar(20) NOT NULL,
  vehicle_registration_number nvarchar(10) NOT NULL,
  PRIMARY KEY (`guest_id`)
);
```

Рисунок 3.4.9 – Запрос создания таблицы «Guest»

Следующая таблица отвечает за резервацию парковочного места посещаемого. Она называется «Reservation» (резервации). Эта таблица будет хранить в себе данные резервации: дата резервации, на кого зарезервирована, и какой какое место зарезервировано (см. рисунок 3.4.10).

```
CREATE TABLE `reservation` (
  `reservation_id` int NOT NULL AUTO_INCREMENT,
  `parking_slot_id` int NOT NULL,
  `guest_id` int NOT NULL,
  `reservation_date` datetime NOT NULL,
  PRIMARY KEY (`reservation_id`),
  FOREIGN KEY (`parking_slot_id`) REFERENCES parking_slot(parking_slot_id),
  FOREIGN KEY (`guest_id`) REFERENCES guest(guest_id)
);
```

Рисунок 3.4.10 – Запрос создания таблицы «Reservation»

Для ведения учёта и журналирования «событий» на парковочной стоянке необходимо создать таблицу для событий. Таблица будет иметь название «Vehicle event» (события транспортных средств). В этой таблице будут храниться данные которые относятся к времени въезда и выезда автомобилей с автостоянки, а также к какой стоянке относится событие (см. рисунок 3.4.11).

```
CREATE TABLE `vehicle_event` (
  `vehicle_event_id` int NOT NULL AUTO_INCREMENT,
  `parking_lot_id` int NOT NULL,
  `time_in` datetime NOT NULL,
  `time_out` datetime,
  PRIMARY KEY (`vehicle_event_id`),
  FOREIGN KEY (`parking_lot_id`) REFERENCES parking_lot(parking_lot_id)
);
```

Рисунок 3.4.11 – Запрос создания таблицы «Vehicle event»

Последняя таблица будет называться «Parked vehicle» (припаркованные транспортные средства). Эта таблица будет хранить в себе информацию о припаркованных автомобилях на автостоянке. Таблица хранит в себе данные о том, какой автомобиль припаркован на каком парковочном месте и к какому временному событию относится автомобиль (см. рисунок 3.4.12).

```
CREATE TABLE `parked_vehicle` (
  `parked_vehicle_id` int NOT NULL AUTO_INCREMENT,
  `vehicle_event_id` int NOT NULL,
  `parking_slot_id` int NOT NULL,
  PRIMARY KEY (`parked_vehicle_id`),
  FOREIGN KEY (`vehicle_event_id`) REFERENCES vehicle_event(vehicle_event_id),
  FOREIGN KEY (`parking_lot_id`) REFERENCES parking_slot(parking_slot_id)
);
```

Рисунок 3.4.12 – Запрос создания таблицы «Parked vehicle»

3.5 Инструменты для работы с базой данных

При разработке базы данных для данного проекта и при непосредственной работе с базой данных были применены два основных инструмента – это Google Cloud Shell и Lucidchart (lucidchart.com)

Google Cloud Shell является интегрированным модулем Google Cloud Platform. Данный модуль позволяет соединятся с удаленными серверами Google Cloud Platform и подключаться к экземплярам баз данных. Google Cloud Shell, простыми словами – это интегрированный терминал в браузере. На рисунках 3.5.1 и 3.5.2 приведен пример как открыть терминал Google Cloud Shell, и подключиться к базе данных.

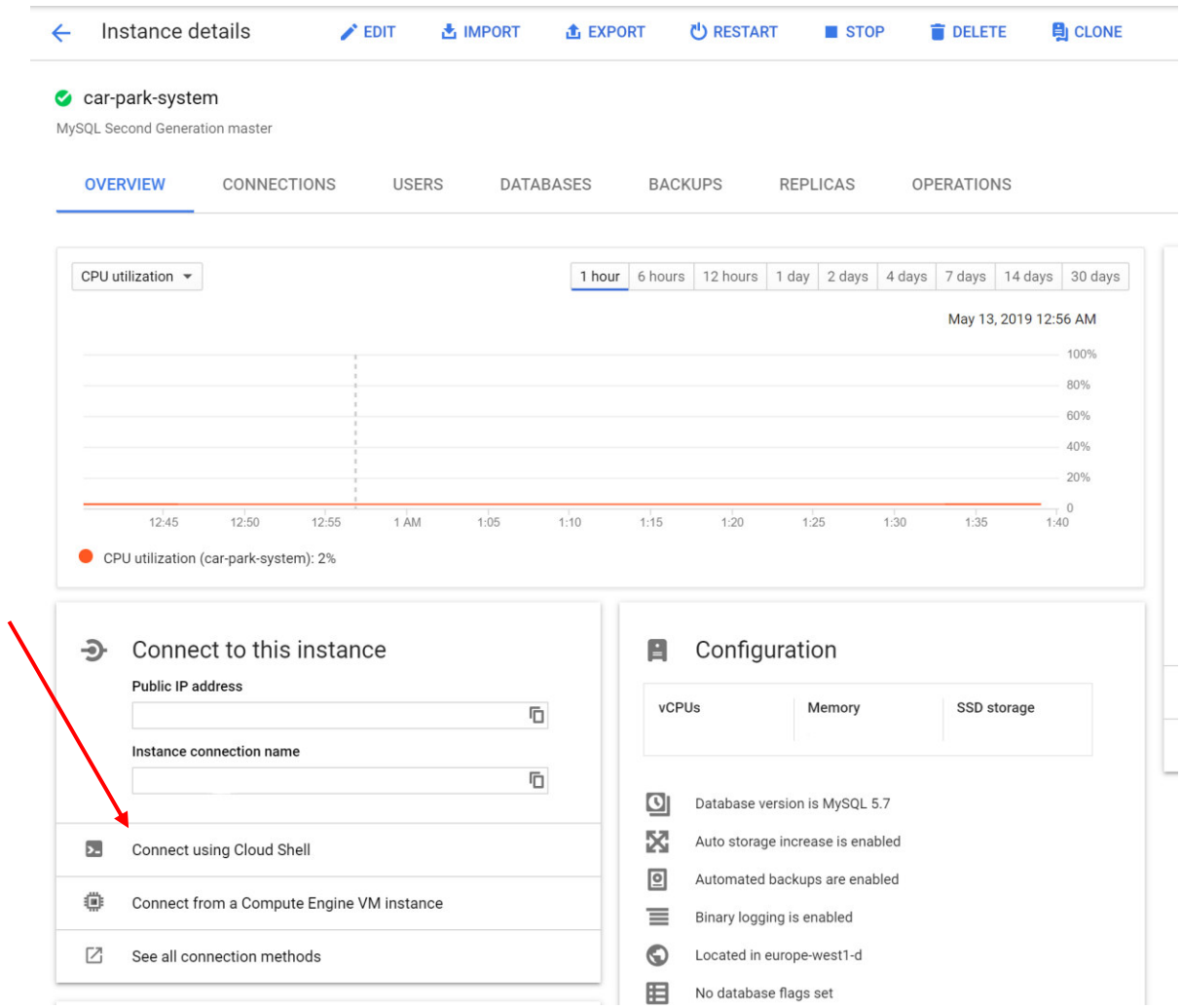

Рисунок 3.5.1 – основной интерфейс для работы с сервером Google Cloud SQL (красной стрелкой отмечена кнопка «Connect using Cloud Shell»)

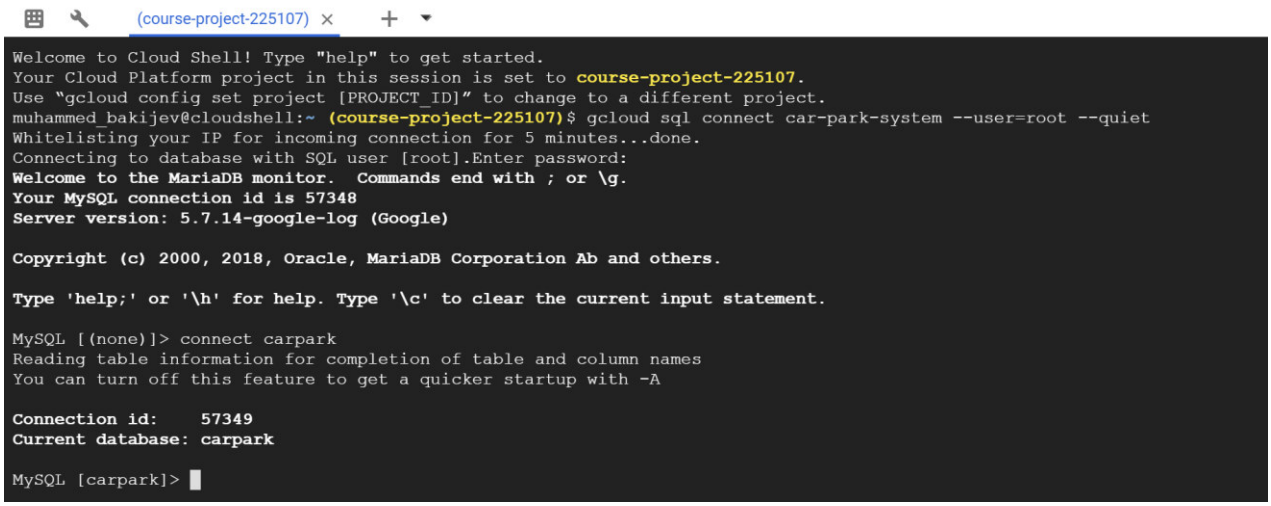

Рисунок 3.5.2 – Терминал Google Cloud Shell, который располагается в нижней части браузера

Для того чтобы подключиться к базе данных через терминал Google Cloud Shell, необходимо выполнить следующие действия:

– открыть терминал;

– ввести следующую команду: *gcloud sql connect* <название экземпляра Google Cloud SQL> *--user*= <имя пользователя> *--quiet;*

– ввести пароль пользователя;

– после подключения к серверу необходимо подключиться к самой базе данных. Для этого используется команда: *connect* <название базы данных>.

Выполнение перечисленных шагов отображены на рисунке 3.5.2.

Lucidchart - это идеальный инструмент для диаграмм для мозгового штурма и управления проектами. Инструмент также хорошо работает с популярными веб-приложениями и бизнес-системами, включая Google Apps. Lucidchart настолько интуитивно понятен, что используется во многих отраслях, включая инжиниринг, веб-дизайн и разработка, а также в бизнес-секторах.

Также, Lucidchart является совершенным инструментом для построения диаграмм. Данный инструмент предоставляет компаниям и профессионалам удобную для пользователя веб-платформу с множеством функций и возможностей.

Lucidchart также предоставляет возможность генерации SQL-кода на основе ER-диаграмм. Это не только ускоряет процесс создания баз данных, но и дает возможность быстро и удобно вносить изменения в структуру базы данных.

4 Разработка

Разрабатываемая система представляет собой клиент-серверное приложение. Клиент серверное приложение – это приложение, которое имеет две основные части: клиент и сервер.

В данном проекте в качестве сервера выступает сервер, который расположен на в дата-центрах Google Cloud Platform, а в качестве клиента - Raspberry Pi, которые будут размещаться при въезде парковочных стоянок в разных локациях.

Единственное взаимодействие пользователей с клиентской частью будет при въезде на парковочную стоянку. Поэтому нет необходимости создания графического интерфейса для данного проекта.

С серверной частью пользователи не будут иметь никакого контакта, т. к. в этом в принципе нет никакой необходимости. Назначение сервера в данном случае – это хранение данных и распознавание текст в изображениях.

В начале разработки необходимо создать проект, который будет включать в себя все необходимые средства для разработки данного проекта.

Так как в данном проекте используются разные библиотеки для Python 3.6, то их нужно установить на систему, где будет располагаться программа. Для этого используется менеджер пакетов PIP. Пакеты для Python 3.6 устанавливается с помощью следующей команды: *pip install* <название пакета>, которую необходимо ввести в командную строку.

Для правильного функционирования данного проекта необходимо установить следующие пакеты:

– OpenCV, пакет «opencv-python»;

– библиотеку для работы с базами данных MySQL, пакет «mysqlclient»;

– библиотеку для работы с Google Cloud Vision API, пакет «google-cloud-vision».

Выполнение команд установки модулей приведено на рисунке 4.1.

```
C:\Users\Muham>pip install opencv-python
Collecting opencv-python
  Using cached https://files.pythonhosted.org/packages/dc/54/a6b7727c67d4e14194549a9e1a1acd7902ebae2f4a688d84b658ae40b5f
b/opencv_python-4.1.0.25-cp36-cp36m-win_amd64.whl
Requirement already satisfied: numpy>=1.11.3 in c:\users\muham\appdata\local\programs\python\python36\lib\site-packages
(from opencv-python) (1.16.3)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.1.0.25
WARNING: You are using pip version 19.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\Muham>pip install mysqlclient
Collecting mysqlclient
  Using cached https://files.pythonhosted.org/packages/c1/15/83f1444c0bbeeff9e14b0ee2cfbc985949404e325938a1b1c20adcd462a
c/mysqlclient-1.4.2.post1-cp36-cp36m-win_amd64.whl
Installing collected packages: mysqlclient
Successfully installed mysqlclient-1.4.2.post1
WARNING: You are using pip version 19.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Рисунок 4.1 – Установка модулей с помощью менеджера пакетов PIP

4.1 Настройка Google Cloud Vision

Одним из важнейших элементов разрабатываемой системы – это корректное функционирование услуги Google Cloud Vision API.

Для этого нужно зайти на страницу управления пользователя Google Cloud Platform и из левой панели выбрать Cloud Vision (см рисунок 4.1.1).

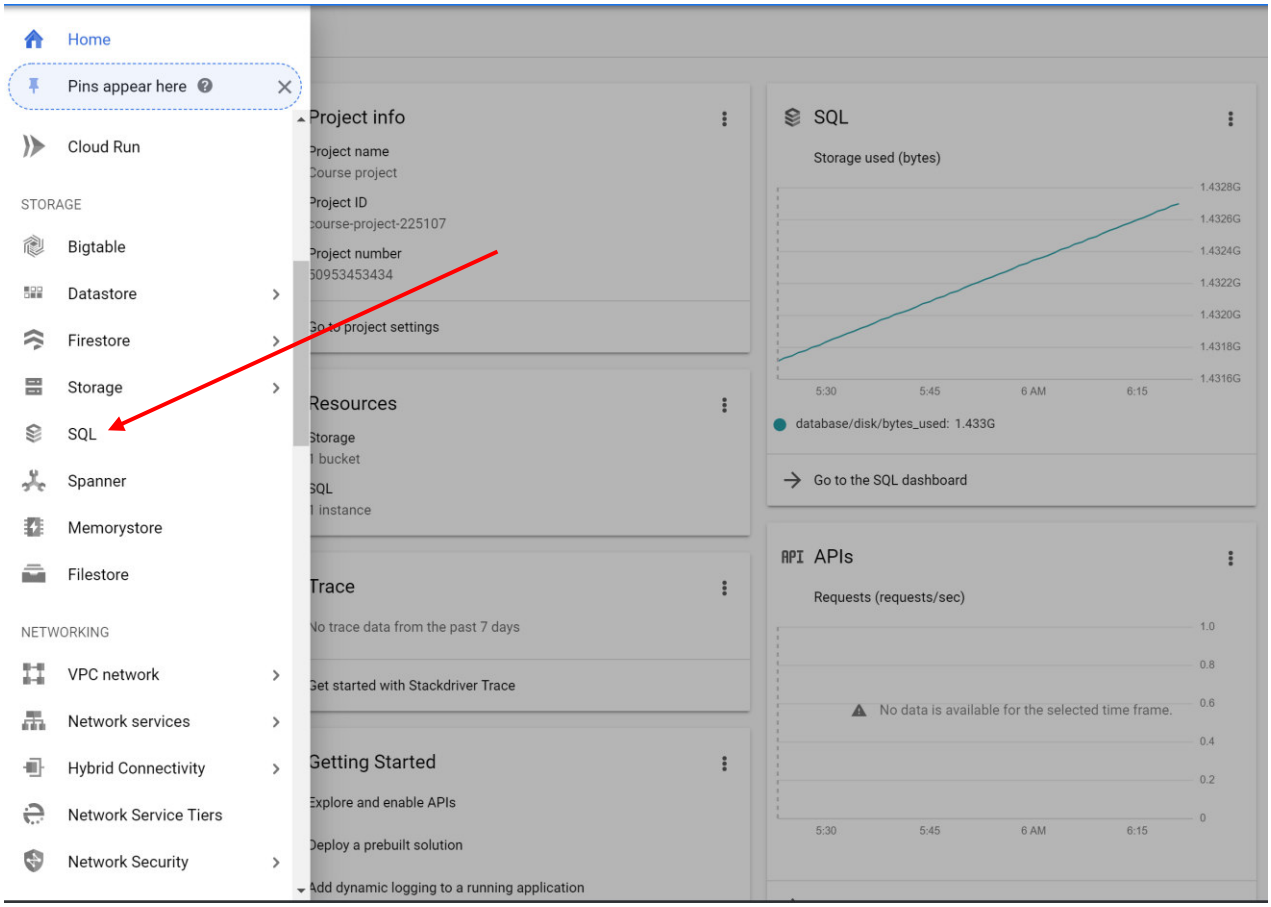

Рисунок 4.1.1 – Интерфейс Google Cloud Platform (красной стрелкой отмечена ссылка, по которой нужно перейти)

Необходимо активировать данную услугу, нажав на кнопку «Activate API». После успешной активации нужно сгенерировать пользовательский файл, который будет использоваться для аутентификации приложения при поступлении запросов на сервер. Для этого необходимо создать служебную учетную запись для Google Cloud Platform, перейдя на страницу создания служебных учетных записей и создать новую. После создания учетной записи можно будет скачать файл (см. рисунок 4.1.3). Данный файл необходимо сохранить в корневую папку проекта. Важно отметить, что данный файл годится только для аутентификации Google Vision API. Для других услуг он не будет работать.

Рисунок 4.1.2 – Вот так выглядит файл аутентификации

4.2 Настройка Google Cloud SQL и создание базы данных

Самым важным элементом данного проекта – это база данных. Для хостинга базы данных применяется услуга Google Cloud SQL. Поэтому послу успешной настройки услуги Google Cloud Vision API нужно настроить сервер для хостинга и хранения базы данных.

Для этого из левой панели нужно найти Google Cloud SQL и перейти по ссылке. Далее нужно создать экземпляр сервера (см. рисунок 4.2.1).

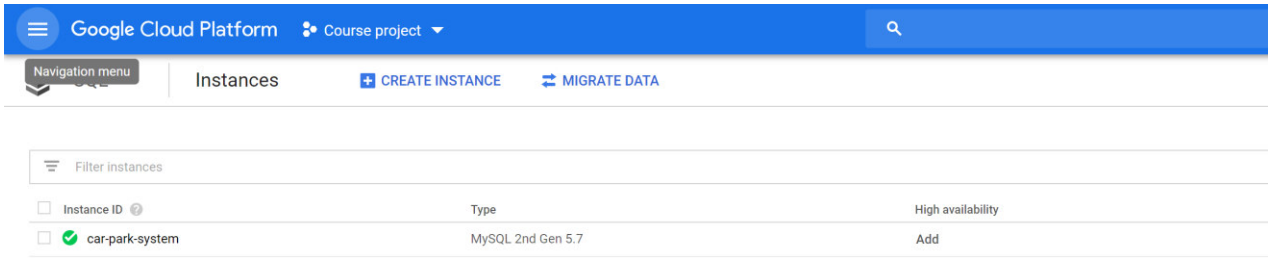

Рисунок 4.2.1 – Страница создания экземпляра сервера для Google Cloud SQL

При создании сервера предлагаются разные характеристики, которые можно указать по желанию. Важно отметить, что чем лучше характеристики сервера, тем выше стоимость аренды сервера. При создании предлагаются следующие характеристики и параметры (см. рисунок 4.2.2):

- выбор региона, для хостинга;
- версия СУБД;
- методы подключения;
- возможность автономного резервного копирования;
- настройка графика обслуживания;
- выбор процессора сервера;
- выбор вид и объем накопителя;
- выбор объема ОЗУ;
- установка меток.

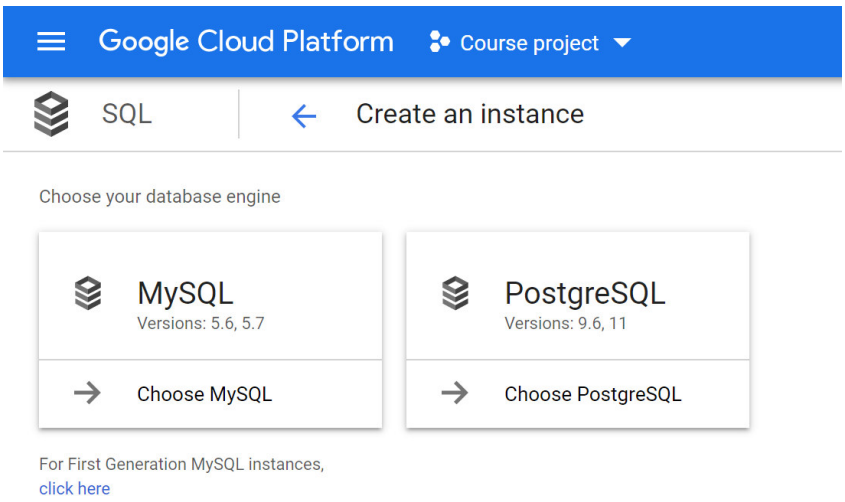

Рисунок 4.2.2 – Выбор СУБД.

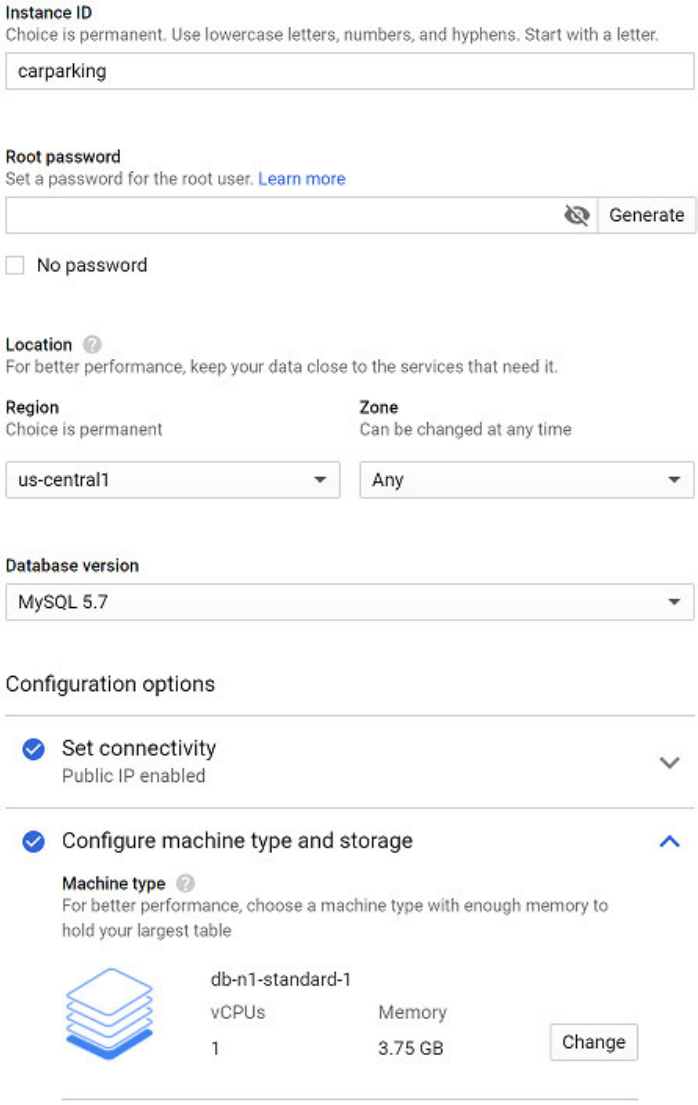

Рисунок 4.2.2 – Настройка характеристик сервера для базы данных

После завершения настройки экземпляра сервера (виртуальной машины) можно приступить к созданию пользователей, которые будут иметь права на подключение к серверу. Для этого нужно перейти по ссылке «USERS» (см. рисунок 4.2.3), и создать пользователя с необходимыми правами и привилегиями.

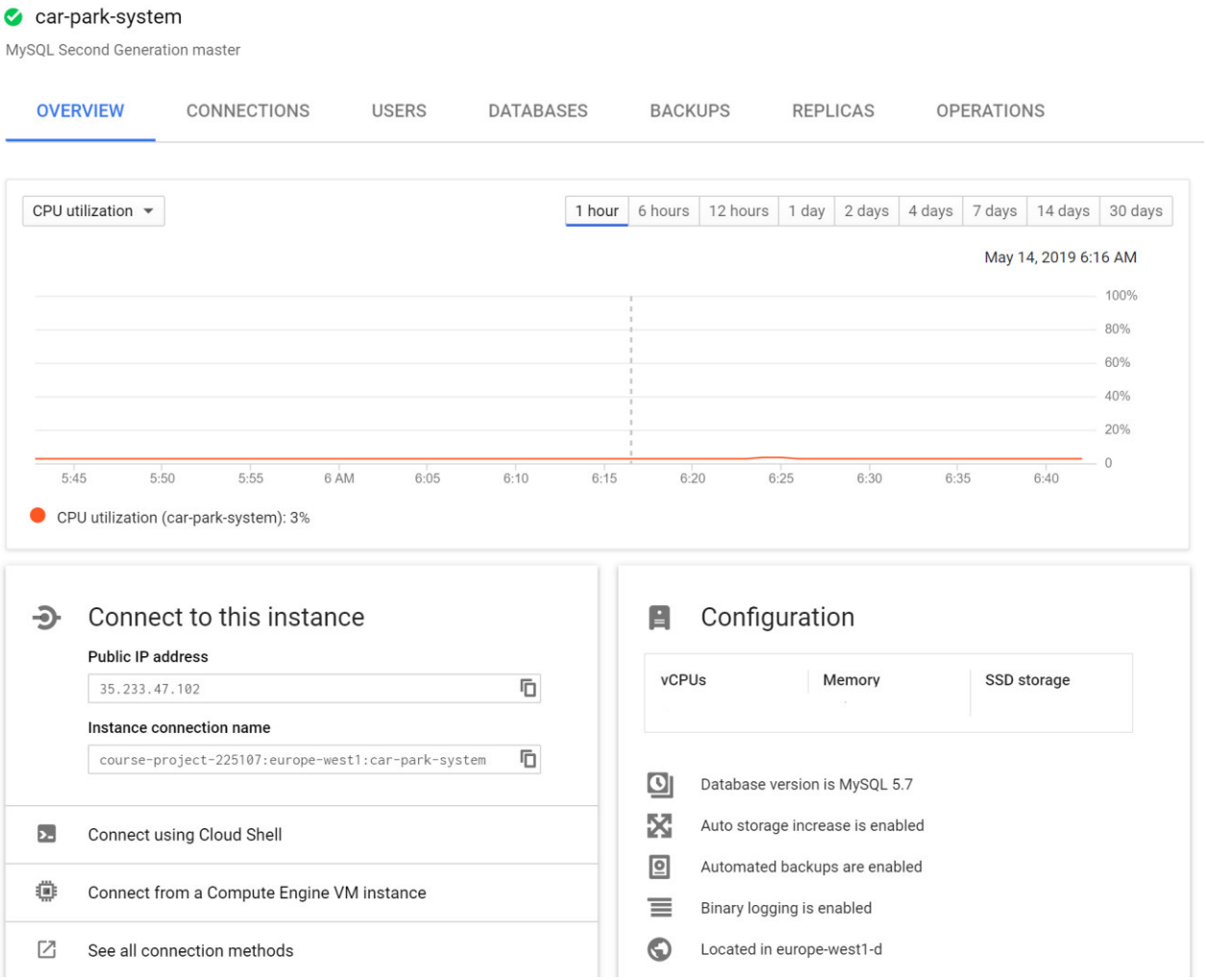

Рисунок 4.2.3 – Панель управления экземпляра сервера

При завершении создания пользователя необходимо перейти по ссылке «CONNECTIONS» (см. рисунок 4.2.3).

На странице для создания подключений можно выбрать какой метод подключения к базе данных будет осуществляться через клиент. Есит два основных метода – это с помощью ключа SSH (Secure Shell) и внешнего IP-адреса.

Основные настройки экземпляра сервера Google Cloud SQL завершены. Теперь можно приступить к созданию новой базы данных. Это можно сделать, перейдя по ссылки «DATABASES» (см. рисунок 4.2.3).

Для создания базы данных требуется лишь указать имя базы и кодировку (см. рисунок 4.2.4).

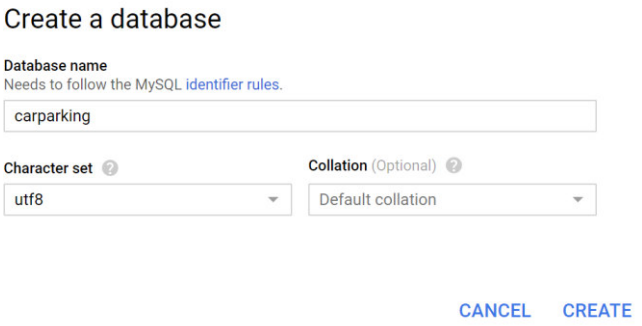

Рисунок 4.2.4 – Создание базы данных

С помощью терминала Cloud Shell можно теперь подключиться к серверу, и далее подключиться к самой базе данных (пример подключения выполнялся в 3 главе, подраздел 5). Нужно приступить к созданию таблиц базы данных. Это делается с помощью запросов, которые были переведены в предыдущей главе.

Сейчас можно сказать, что процесс создания базы данных завершен, т. к. выполнены все необходимые требования. Для того, чтобы можно было проверить работоспособность созданной базы данных необходимо заполнить ее таблицы и выполнить несколько запросов. Пример запроса для заполнения таблицы «User» приведен ниже. Выполнение запроса приведено на рисунке 4.2.8. Аналогично заполняются остальные таблицы.

Пример запроса:

INSERT INTO user(parking_lot_id, first_name, last_name, phone_number, email_address)
VALUES(1, "Testname", "Testsurname", "87777777777", "test@gmail.com");

```
MySQL [carpark]> INSERT INTO user(parking_lot_id, first_name, last_name, phone_number, email_address)
    -> VALUES(1, "Testname", "Testsurname", "87777777777", "test@gmail.com");
Query OK, 1 row affected (0.26 sec)
```

Рисунок 4.2.5 – Выполнение запроса вставки в таблицу «User», где видно, что выполнение запроса прошло успешно

4.3 Разработка основной программы

Основная программа представляет собой набор функций и алгоритмов, которые выполняют основную работу системы. Начиная с обнаружения автомобиля и заканчивая с его пропуском на парковочную стоянку.

После завершения настройки серверной части можно приступить к локальной части, а именно разработки программы, которая будет работать на одноплатном компьютере Raspberry Pi.

Для этого нужно создать новый проект в среде разработки и указать желаемый и интерпретатор. Можно указать существующий интерпретатор,

либо создать новый. Но т. к. прежде были установлены пакеты на существующий интерпретатор, то выбирается существующий.

Для проверки работы пакетов можно выполнить простые команды, которые присутствуют в данных пакетах, либо увидеть это визуально с помощью графического интерфейса интегрированной среды разработки PyCharm (см. рисунок 4.3.2).

```
import cv2
import MySQLdb
from google.cloud import vision
```

Рисунок 4.3.1 – Так как импортируемые модули не подчеркиваются красным, можно быть уверенным, что они корректно установлены

Прежде чем начинать писать головную программу нужно создать класс, в котором будут содержаться статические методы для обработки графических изображений, текстовой информации и других данных (см. рисунок 4.3.2).

```
class Processor:

    def __init__(self):
        pass
```

Рисунок 4.3.2 – Класс «Processor», который будет содержать все методы для обработки информации

Созданный класс будет иметь следующие методы.

```
@staticmethod
def get_bytestring(image):
    return cv2.imencode('.jpg', image)[1].tostring()
```

Рисунок 4.3.3 – Конвертация графического изображения с формата JPG в строку байтов

```
@staticmethod
def get_binary_image(image):
    gray = cv2.cvtColor(image.copy(), cv2.COLOR_RGB2GRAY)
    # cv2.imshow('111', gray)
    smooth = cv2.bilateralFilter(gray, 11, 21, 21)
    # cv2.imshow('1111', smooth)
    binary = cv2.adaptiveThreshold(smooth, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 55, 5)
    return binary
```

Рисунок 4.3.4 – Обработка цветного трехканального изображения в «бинарное» (двухцветное, черный и белый) изображение

```python
@staticmethod
def canny(image):
    gray = cv2.cvtColor(image.copy(), cv2.COLOR_RGB2GRAY)
    # blur = cv2.bilateralFilter(gray, 9, 17, 17)
    # eq_histogram = cv2.equalizeHist(gray)
    canny_img = cv2.Canny(gray, 79, 100)
    return canny_img
```

Рисунок 4.3.5 – Выявление углов и контуров цветного изображения

```python
@staticmethod
def get_plate_text(text):
    result = re.findall(r'\d{3}\s?[A-Z]+\s?\d{2}|'
                        r'[A-Z]\s\d{1,3}\s[A-Z]{3}|'
                        r'[A-Z]\s?\d{1,4}\s\d{1,2}', text)
    if len(result) < 1:
        print('No plates found')
    if len(result) > 1:
        return -1
    else:
        result = result[0].replace(' ', '')
        return result
```

Рисунок 4.3.6 – Парсинг текста для получения государственных регистрационных номеров с помощью RegEx

```python
@staticmethod
def get_formatted_datetime():
    return datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

Рисунок 4.3.7 – Метод, который возвращает текущую дату и время в формате, который подходит для запросов SQL

```python
@staticmethod
def get_mktime(string):
    return time.mktime(time.strptime(string, '%Y-%m-%d %H:%M:%S'))
```

Рисунок 4.3.8 – Конвертация строки с датой и временем в формат времени UNIX

```
@staticmethod
def has_reservations_expired(reservation_date):
    reservation_mktime = Processor.get_mktime(reservation_date)
    now = time.time()
    return True if reservation_mktime < now else False
```

Рисунок 4.3.9 – Данный метод для проверки резервации. Истек ли срок резервации

В любом языка программирования, при работе с базами данных, необходимо создать переменную, в которой будет храниться подключение к самой базе. Язык программирования Python 3.6 имеет много разных библиотек для работы с разными СУБД. В данном проекта используется СУБД MySQL и для этого можно применить библиотеку MySQLdb, в которой есть все необходимое для корректной работы с базами данных.

Для того, чтобы подключиться к облачной базе данных необходимо знать:

– IP-адрес сервера, на котором хранится база данных;

– имя пользователья, который имеет права на редактирование базы данных;

– пароль пользователя;

– название базы данных.

Если при создании базы данных указывалась кодировка, то ее тоже нужно знать, т. к. при выполнении запросов могут произойти некорректные действия.

Пример подключения к базе данных приведен на рисунке 4.3.10.

```
dbconnect = MySQLdb.connect(HOST, USER, PASSWORD, DB, charset='utf8')
```

Рисунок 4.3.10 – Подключение к базе данных с помощью библиотеки MySQLdb

Выполнение запросов осуществляется через «курсор». Класс курсора содержит все необходимые метода для того, чтобы можно было полноценно работать с базами данных. Например, с помочью курсора выполняются запросы SELECT, INSERT, DELETE, UPDATE, CREATE.

Курсор можно только создать для текущей сессии подключения с базой данных. Если срок сессии истек, то курсор уже не действителен. Также один курсор нельзя применить для сессии подключения другой базы данных.

Пример создания курсора приведено на рисунке 4.3.11.

```
cursor = dbconnect.cursor()
```

Рисунок 4.3.11 – Создание курсора

Чтобы не переписывать запросы, которые необходимо выполнить для того, чтобы получить данные с базы данных, нужно создать отдельный файл, где будут храниться эти запросы. Это не только очистит головную программу от лишнего кода, но и также создаст удобный метод для выполнения запросов с помощью библиотеки MySQLdb.

После выполнения любого запроса, который вносит изменения в базу данных, нужно выполнить коммит. Коммит – это подтверждение вноса изменений, который требуется чтобы не нарушить целостность базы данных. Если произойдет сбой программы и она зациклится или начнет выполнять запроса удаления данных, то без коммита эти действия не будут вноситься в саму базу данных.

Коммит выполняется с помощью функции «commit()». Нужно писать код так, чтобы если при выполнении запроса произойдет ошибка, то произойдет откат действий. Это обеспечивает безопасность и целостность данных базы данных. Откат действий выполняется с помощью функции «rollback()».

Примеры работы с запросами приведены ниже:

```
QUERY_SELECT_USER_BY_USER_ID = 'SELECT u.first_name, u.last_name, u.parking_lot_id, ps.parking_slot_id, ' \
                               'ps.is_occupied ' \
                               'FROM user u ' \
                               'INNER JOIN parking_slot ps ON ps.owner_id = u.user_id ' \
                               'WHERE user_id={};'
```

Рисунок 4.3.12 – Пример кода, где хранится запрос выборки

```
def select_user_by_user_id(cursor, user_id):
    user_name = ''
    parking_lot_id = 0
    parking_slot_id = 0
    is_occupied = False
    try:
        query_user = QUERY_SELECT_USER_BY_USER_ID.format(user_id)
        cursor.execute(query_user)
        result = cursor.fetchone()
        user_name = result[0] + ' ' + result[1]
        parking_lot_id = result[2]
        parking_slot_id = result[3]
        is_occupied = result[4]
    except MySQLdb.Error as error:
        print("ERROR ON GETTING USER INFORMATION: {}".format(error))
    finally:
        return user_name, parking_lot_id, parking_slot_id, is_occupied
```

Рисунок 4.3.13 – Пример выполнения запроса

```
QUERY_UPDATE_VEHICLE_EVENT_OUT_BY_REG_NUM = 'UPDATE vehicle_event ' \
                                            'SET time_out = "{0}" ' \
                                            'WHERE vehicle_event_id = ( ' \
                                            'SELECT pv.vehicle_event_id FROM parked_vehicle pv ' \
                                            'WHERE pv.parking_slot_id = ( ' \
                                            'SELECT ps.parking_slot_id FROM parking_slot ps ' \
                                            'INNER JOIN reservation r ' \
                                            'ON r.parking_slot_id = ps.parking_slot_id ' \
                                            'INNER JOIN guest g ' \
                                            'ON g.guest_id = r.guest_id ' \
                                            'WHERE g.vehicle_registration_number = "{1}") ' \
                                            'OR pv.parking_slot_id = ( ' \
                                            'SELECT ps.parking_slot_id FROM parking_slot ps ' \
                                            'INNER JOIN user u ' \
                                            'ON u.user_id = ps.owner_id ' \
                                            'INNER JOIN registered_vehicle rv ' \
                                            'ON u.user_id = rv.user_id ' \
                                            'WHERE rv.registration_number = "{1}"));'
```

Рисунок 4.3.14 – Пример запроса UPDATE

```
def update_vehicle_event(dbconnect, cursor, registration_number):
    try:
        query = QUERY_UPDATE_VEHICLE_EVENT_OUT_BY_REG_NUM.format(Processor.get_formatted_datetime(),
                                                                 registration_number)
        cursor.execute(query)
        dbconnect.commit()
    except MySQLdb.Error as error:
        dbconnect.rollback()
        print('ERROR UPDATE VEHICLE EVENT: {}'.format(error))
```

Рисунок 4.3.15 – Пример выполнения запроса

В головной программе необходимо создать функцию инициации Google API (см. рисунок 4.3.16). Создается «сессия» с помощью аутентификационного файла, который был создан ранее. Также создать несколько функции для упрощения работы с Google Cloud Vision API. Данные функции принимают в параметрах графическое изображение, которое загружается на сервер. От сервера поступает ответ в виде JSON, где хранится распознанный текст и его координаты на картинке. Из JSON вынимается текст, и функция его возвращает. Эти функции не обязательны, но таким образом код головной программы становится более читабельным (см. рисунок 4.3.17).

```
def init_googleapi():
    global VISION_API_CLIENT
    # Instantiates a client
    VISION_API_CLIENT = vision.ImageAnnotatorClient()
```

Рисунок 4.3.16 – Инициация Google Vision API

```
def detect_text(client, img):
    response = client.text_detection(image=img)
    texts = response.text_annotations
    return texts


def detect_document(client, img):
    response = client.document_text_detection(image=img)
    document = response.full_text_annotation
    return document


def detect_lables(client, img):
    response = client.label_detection(image=img)
    labels = response.label_annotations
    return labels
```

Рисунок 4.3.17 – Функции для упрощения работы с библиотекой Google Vision API и для улучшения читабельности кода

Для корректной работы контактов ввода/вывода Raspberry Pi нужно сперва инициировать нужные контакты.

Нужно определить какие контакты будут использоваться для ультразвукового дальномера и для шлагбаума. Ультразвуковой дальномер имеет четыре пина: питание, земля, триггер и эхо. Поэтому нужно выделить два контакта для ультразвукового дальномера и один для шлагбаума.

Код инициации пинов приведен на рисунке 4.3.18.

```
GPIO.setmode(GPIO.BCM)

trigger = 20
echo = 16
gate_signal = 23

GPIO.setup(gate_signal, GPIO.OUT)
GPIO.setup(trigger, GPIO.OUT)
GPIO.setup(echo, GPIO.IN)

GPIO.output(gate_signal, False)
GPIO.output(trigger, False)
```

Рисунок 4.3.18 – Инициация пинов Raspberry Pi

Для триггера и эхо дальномера применены пин 20 и 16, а для управления шлагбаума 23. Важно указать в каком направлении будет работать пин, т. к. он не может принимать и отправлять данные одновременно. Также нужно подать на пины вывода логический 0. Это делается для того, чтобы на контакты не подавался ток.

Управление шлагбаумом осуществляется очень просто. У шлагбаума имеется встроенное реле управления. Для того, чтобы поднять его необходимо на вывод шлагбаума подать логическую 1. В противном случае подать логический 0.

Измерение расстояния с помощью ультразвукового дальномера требует его активации. У дальномера есть интегрированная схема активации, которая при определенных действиях активирует динамик дальномера. Для активации дальномера необходимо на триггер подать логическую 1, длительностью 10 микросекунд, после чего нужно опустить сигнал на 0. При выполнении данного действия ультразвуковой дальномер будет производить ультразвуковые волны, которые можно будет замерить.

После активации дальномера нужно считать с него данные. Считывание происходит с контакта эхо. Как только этот контакт поднимет свое значение на логическую единицу, то можно считать, что звуковые волны вернулись к ультразвуковому дальномеру. Разница во времени между активацией и получением сигнала можно использовать для вычисления расстояния до объекта, от которого произошло отражение ультразвуковых волн. Если расстояние менее двух метров, то можно начинать процесс обработки государственного регистрационного номера и проверки наличия номера в базе данных.

```
sensor_entrance = 0
sensor_exit = 0
gate = 0
GPIO.output(trigger, True)
time.sleep(0.00001)
GPIO.output(trigger, False)

pulse_start = time.time()
while GPIO.input(echo) == 0:
    pulse_start = time.time()

pulse_end = time.time()
while GPIO.input(echo) == 1:
    pulse_end = time.time()
    time.sleep(2)

pulse_duration = pulse_end - pulse_start
distance = pulse_duration * 17150
distance = round(distance, 2)
if distance < 200:
    sensor_entrance = 1
else:
    sensor_entrance = 0
    close_gate()
```

Рисунок 4.3.19 – Логика работы ультразвукового дальномера

С помощью функции time.time() можно записать самую последнюю временную метку времени компьютера. Например, если пин переходит от низкого к высокому уровню (от логического 0 к 1), и записывается состояние низкого уровня с помощью функции time.time(), то записанная метка времени будет самым последним временем, когда этот вывод был на низком уровне. На рисунке 4.3.19 записывается временная метка при отправке звукового сигнала и при получении сигнала.

Количество проверок на наличие автомобиля составляет пять попыток. Если произошло обнаружение объекта более пяти раз и при этом камера не зафиксировала никакой государственный регистрационный номер, то объект будет игнорироваться до тех пор, пока он не выйдет из диапазона дальномера.

После того как автомобиль проехал шлагбаум дальномер заново проверяет наличие автомобиля. Т. к. автомобиля уже нет, то запускается функция снижения шлагбаума с задержкой двух секунд.

Для того чтобы вычислить расстояние, которое «замерил» дальномер можно применить следующую формулу (4.1):

$$\text{Скорость} = \frac{\text{Расстояние}}{\text{Время}} \tag{4.1}$$

Нас самом деле все что делает дальномер – это отправлять и принимать ультразвуковые волны. Он ничего не замеряет. Все вычисления производятся в программном коде, который приведен на рисунке 4.3.19.

Звук – это вибрация, которая обычно распространяется как звуковая волна давления через среду передачи, такую как газ, жидкость или твердое вещество.

Ультразвук – это звуковые волны с частотами выше верхнего предела слышимости человеческого слуха.

Скорость звука зависит от среды, через которую распространяются звуковые волны. Приблизительная скорость звуковых волн через воздух равна 343 м/с.

Время, которое получается при измерении длительности между активацией дальномера и получением сигнала, нужно поделить на два, т. к. это время отвечает за прохождение сигнала в обе стороны, т. е. от дальномера к объекту и обратно (формула 4.2).

$$34300 = \frac{\text{Расстояние}}{\text{Время}/2} \tag{4.2}$$

Следовательно (4.3):

$$17150 = \frac{\text{Расстояние}}{\text{Время}} \tag{4.3}$$

Итоговую формулу (4.4) можно применить для вычисления расстояния от дальномера до «объекта».

$$17150 * \text{Время} = \text{Расстояние} \quad (4.4)$$

Формула 4.4 применяется при вычислении расстояние в программе функционирования дальномера, которая приведена на рисунке 4.3.18.

5 Экономическая часть

5.1 Расчет трудоемкости разработки

В данной дипломной работе описывается разработка системы, которая должна обеспечить автоматизированный контроль и вести учет автомобилей на паркингах. Система разрабатывается специально для ТОО «Almaz-Group» в целях упрощения регистрации машин, прибывшие на зону паркинга.

Таблица 5.1.1 - Распределение работ по этапам и оценка их трудоемкости

Этап разработки ПП	Вид работы на данном этапе	Трудоемкость разработки ПП, чел.× ч.
Формулировка/Постановка задания	Определение цели, которой нужно достичь	8
	Планирование и распределение работ	16
	Определение, анализ и выбор инструментов для разработки ПП	16
Анализ	Изучение и анализ рынка и необходимого материала	40
Разработка алгоритма и блок-схемы	Составление блок-схемы и разработка алгоритма	28
Составление программы по готовой блок - схеме	Создание скрипта для получения государственного регистрационного номера от изображений	24
	Создание Базы данных MySQL	48
	Сбор аппаратной части	13
Тестирование системы	Проверка работоспособности системы	16
Отладка программы на ПК	Отладка ошибок и оптимизирование	36
Подготовка документации для разработки	Подготовка документации	12
ИТОГО		257

5.2 Расчет затрат на разработку

Расчет затрат на разработку ПП производится путем составления соответствующей сметы, которая включает следующие статьи:

- материальные затраты;
- затраты на оплату труда;
- социальный налог;
- амортизация основных фондов;
- прочие затраты.

В статью «Материальные затраты» включаются затраты на основные и вспомогательные материалы (бумага, картриджи и др.), энергию, необходимые для разработки ПП.

Расчет затрат на материальные ресурсы производится по форме, приведенной в таблице 5.2.1.

Общая сумма затрат на материальные ресурсы ($З_М$) определяется по формуле (5.1):

$$З_М = \sum_{i=1}^{n} P_i * Ц \quad (5.1)$$

Расчет суммы материальных затрат приведен в формуле 5.2.

$$З_М = 2000 + 6000 + 512000 + 35000 + 33900 + 600 + 30000 = 619\,500 \quad (5.2)$$

где P_i - расход i-го вида материального ресурса, натуральные единицы;
$Ц_i$ - цена за единицу i-го вида материального ресурса, тг;
i - вид материального ресурса;
n - количество видов материальных ресурсов.

Стоимость упаковки бумаги А4 (500 листов) составляет 2000 тг. Количество листов данного дипломного проекта выходит на 80 листов А4. Картридж для принтера Samsung M2070W стоит 6000 тг.

Таблица 5.2.1 - Затраты на материальные ресурсы

Наименование материального ресурса	Единица измерения	Количество израсходованного материала	Цена за единицу, тг	Сумма, тг
Бумага	шт.	1	2 000	2 000
Картридж для принтера	шт.	1	6 000	6 000
Ноутбук HP	шт.	1	512 000	512 000

Продолжение таблицы 5.2.1

Наименование материального ресурса	Единица измерения	Количество израсходованного материала	Цена за единицу, тг	Сумма, тг
Одноплатный компьютер Raspberry Pi 3B	шт.	1	35 000	35 000
IP камера Ubiquiti Unifi	шт.	1	33 900	33 900
Ультразвуковой дальномер HC-SR04	шт.	1	600	600
4G+ WiFi Модем Huawei	шт.	1	30 000	30 000
ИТОГО				619 500

Если для разработки ПП используется электрооборудование, то необходимо рассчитать затраты на электроэнергию по форме, приведенной в таблице 5.2.2.

Стоимость 1 кв./ч для юридических лиц, по данным АО «Самрук-Энерго» - «АлматыЭнергоСбыт», составляет 18.31 тг.

Общая сумма затрат на электроэнергию ($З_Э$) рассчитывается по формуле:

$$З_э = \sum_{i=1}^{n} M_i * K_i * T_i * Ц \quad (5.3)$$

Расчет суммы затрат на электроэнергию приведен в формуле 5.4.

$$З_э = (0.2 * 0.7 * 257 * 18.31) + (0.304 * 0.7 * 6 * 18.31) + (0.065 * 0.7 * 200 * 18.31) + (0.024 * 0.7 * 257 * 18.31) + (0.0075 * 0.07 * 150 * 18.31) + (0.024 * 0.7 * 150 * 18.31) = 988.4 \text{ тг} \quad (5.4)$$

где M_i - паспортная мощность i-го электрооборудования, кВт;

K_i - коэффициент использования мощности i-го электрооборудования (принимается K_i=0.7¸0.9);

T_i - время работы i-го оборудования за весь период разработки ПП ч;

Ц - цена электроэнергии, тг/кВт×ч;

i - вид электрооборудования;

n - количество электрооборудования.

Таблица 5.2.2 - Затраты на электроэнергию

Наименование оборудования	Паспортная мощность, кВт	Коэффициент использования мощности	Время работы оборудования для разработки ПП, ч	Цена электро-энергии, $\frac{тг}{кВ*ч}$	Сумма, тг
Ноутбук HP	0.2	0.7	257	18.31	658.79
Принтер Samsung	0.304	0.7	6	18.31	23.37
Монитор Phillips	0.065	0.7	200	18.31	166.62
4G+ WiFi Модем	0.024	0.7	257	18.31	79.06
Компьютер Raspberry Pi 3B	0.0075	0.7	150	18.31	14.42
IP камера Ubiquiti Unifi	0.024	0.7	150	18.31	46.14
ИТОГО					988.4

В статью «Затраты на оплату труда» включаются расходы по оплате труда всех работников, занятых разработкой ПП (дипломника, руководителей и консультантов дипломной работы, привлеченных лиц).

Затраты на оплату труда рассчитываются по форме, приведенной в таблице 5.2.3.

Средняя заработная плата программистов, по данным, предоставленным ТОО «Trudbox.kz», составляет 210000 тг.

Таблица 5.2.3 - Затраты на оплату труда

Категория работника	Квалификация	Трудоемкость разработки ПП, чел.×ч	Часовая ставка, тг/ч	Сумма, тг
Служащий	Программист	257	1 000	257 000
Служащий	Директор департамента IT ТОО «Almaz-Group»	50	4 000	200 000
ИТОГО				457 000

В статью «Социальный налог» включается сумма, которая рассчитывается как 11% (9.5% социальный налог + 1.5% мед. страхование) от затрат на оплату труда всех работников ($З_{тр}$), занятых разработкой ПП. При

расчете необходимо учесть, что пенсионные отчисления (10% от $З_{тр}$) не облагаются социальным налогом (ставки указаны на 2019 год).

Социальный налог программиста = $(257000 - 25700) * 11\% = 25443$ тг

Социальный налог директора департамента IT ТОО «Almaz-Group» = $(200000 - 20000) * 11\% = 19800$ тг

Общая сумма = $25443 + 19800 = 45243$ тг

В статью «Амортизация основных фондов» включается сумма амортизационных отчислений от стоимости оборудования и программного обеспечения (ПО), используемых при разработке. Амортизационные отчисления рассчитываются по форме, приведенной в таблице 5.2.4.

Общая сумма амортизационных отчислений определяется по формуле:

$$З_{АМ} = \sum_{i=1}^{n} \frac{Ф_i * Н_{Ai} * T_{НИРi}}{100 * Т_{ЭФi}}, \quad (5.5)$$

Расчет суммы амортизации приведен в формуле 5.6.

$$З_{АМ} = \left(\frac{512000*25*257}{100*1970}\right) + \left(\frac{174665*25*200}{100*1970}\right) + \left(\frac{35000*25*150}{100*1970}\right) + \left(\frac{12000*25*257}{100*1970}\right) + \left(\frac{15000*14.29*6}{100*1970}\right) + \left(\frac{33900*20*150}{100*1970}\right) + \left(\frac{30000*33.33*257}{100*1970}\right) + \left(\frac{25800*20*257}{100*1970}\right) = 28211{,}9 \text{ тг} \quad (5.6)$$

где $Ф_i$ - стоимость i-го ОФ, тг;

$Н_{Ai}$ - годовая норма амортизации i-го ОФ, %;

$Т_{НИРi}$ - время работы i-го ОФ за весь период разработки ПП, ч;

$Т_{Эфi}$ - эффективный фонд времени работы i-го ОФ за год, ч/год;

i - вид ОФ;

n - количество ОФ.

Таблица 5.2.4 - Амортизация основных фондов (ОФ)

Наименование оборудования и ПО	Стоимость оборудования и ПО, тг	Годовая норма амортизации, %	Эффективный фонд времени работы оборудования и ПО, ч/год	Время работы оборудования и ПО для разработки ПП, ч	Сумма, тг
Ноутбук HP	512 000	25	1 970	257	16 698.5
Монитор Philips	174 665	25	1 970	200	4 433.12
Одноплатный компьютер Raspberry Pi 3B	35 000	25	1970	150	666.24

Продолжение таблицы 5.2.4

Наименование оборудования и ПО	Стоимость оборудования и ПО, тг	Годовая норма амортизации, %	Эффективный фонд времени работы оборудования и ПО, ч/год	Время работы оборудования и ПО для разработки ПП, ч	Сумма, тг
ОС Microsoft Windows 10 Pro	120 000	25	1 970	257	3 913.7
Принтер Samsung	15 000	14.29	1 970	6	6.53
IP камера Ubiquiti Unifi	33 900	20	1 970	150	516.24
4G+ WiFi Модем	30 000	33.33	1 970	257	1 304.44
Компьютерная мышь Logitech	25 800	20	1 970	257	673.16
ИТОГО					28 211.9

В статью «Прочие затраты» включаются расходы на арендную плату и услуги подключения к интернету.

Затраты на арендную плату определяются в зависимости от стоимости аренды 1 кв. м занимаемой площади.

Таблица 5.2.5 – Арендная плата за офисное помещение

Занимаемая площадь, кв. м	Цена за кв. м, тг	Стоимость за месяц, тг	Длительность аренды, месяцев	Сумма, тг
32	3 800	121 600	1	121 600

Для разработки был необходим интернет, поэтому была подключена услуга интернета «Безлимит Турбо» от Altel. Стоимось данной услуги приведена в таблице 5.2.6.

Таблица 5.2.6 – Стоимость услуги интернет «Безлимит Турбо»

Цена за месяц, тг	Срок, месяцев	Сумма, тг
10 000	1	10 000

Общая сумма «прочих затрат» составляет 136600 тг.

На основании полученных данных по отдельным статьям составляется смета затрат на разработку по форме, приведенной в таблице 5.2.7.

Т а б л и ц а 5.2.7 - Смета затрат на разработку

Статьи затрат	Сумма, тг
1. Материальные затраты, в том числе:	620 488.4
- материалы	619 500
- электроэнергия	988.4
2. Затраты на оплату труда.	457 000
3. Отчисления социальный налог.	45 243
4. Амортизация основных фондов.	28 211.9
5. Прочие затраты.	136 600
ИТОГО	1 287 543.3

На диаграмме 5.2.1 представлена смета затрат на разработку.

Диаграмма 1 – Смета затрат на разработку проекта

5.3 Определение возможной (договорной) цены

Величина возможной (договорной) цены ПП должна устанавливаться с учетом эффективности, качества и сроков ее выполнения на уровне, отвечающем экономическим интересам заказчика (потребителя) и исполнителя.

Договорная цена ($Ц_д$) для прикладных ПП рассчитывается по формуле (5.7):

$$Ц_д = З_{НИР}\left(1 + \frac{Р}{100}\right), \quad (5.7)$$

где $З_{НИР}$ - затраты на разработку ПП (из таблицы Х.8), тг;

Р - средний уровень рентабельности ПП. % (принимается в размере 20-30% по согласованию с консультантом по экономической части).

Расчет договорной цены приведен в формуле 5.8.

$$Ц_Д = 1\ 287\ 543.3 * \left(1 + \frac{25}{100}\right) = \ 1\ 609\ 429.125 \text{ тг} \quad (5.8)$$

Далее определяется цена реализации с учетом налога на добавленную стоимость (НДС), ставка НДС устанавливается законодательно Налоговым Кодексом РК. На 2019 год ставка НДС установлена в размере 12%.

Цена реализации с учетом НДС рассчитывается по формуле:

$$Цр = Цд + Цд * НДС \quad (5.9)$$

Расчет цены реализации приведен в формуле 5.10.

$$Ц_Р = 1\ 609\ 429.125 + \ 1\ 609\ 429.125 * 0{,}12 = \ 1\ 802\ 560.62 \text{ тг} \quad (5.10)$$

Таким образом себестоимость разрабатываемого продукта составляет 1287542.3 тг, прибыль получается 321885.825 тг и цена реализации, с учётом НДС составляет 1802560.62 тг.

5.4 Оценка социально - экономических результатов функционирования разработанного проекта

В результате проведения оценки социально-экономических результатов разработки автоматической системы регистрации парковочных мест были получены следующие результаты: цена реализации окупает все затраты, потрачены на разработку. Прибыль от реализации программного продукта равна (5.11):

$$П = \ Ц_р - ΣЗ \quad (5.11)$$

Расчет прибыли приведен в формуле 5.12.

$$П = 1\ 802\ 560.62 - 1\ 287\ 543.3 = 515\ 017.32 \text{тг} \quad (5.12)$$

где: П – прибыль;

$Ц_р$ – цена реализации;

ΣЗ – сумма всех затрат на разработку.

6 Охрана труда и безопасность жизнедеятельности

В данной дипломной работе ведется проектировка автоматической системы регистрации парковочных мест для ТОО «Almaz-Group». Данная система будет применяться при въезде на парковочные зоны, крытых автостоянок и подземного паркинга.

ТОО «Almaz-Group» находится по адресу Ауэзова, 60. В данном разделе рассматривается определенный офис, где работают 3 служащих. Для работы применяются ноутбуки. Данный вид оборудования не производит лишнего шума. В офисе есть два окна, с общей площадью 6м^2, которые выходят на юг, а также в близи нет зданий, прикрывающих естественное освещение для офиса. Также имеются хорошие лампы для освещения. Исходя из этих данных можно сказать, что проблем с шумом и освещением в данном помещении нет.

В данном офисе отсутствует кондиционер, из-за чего есть проблему с воздухообменом и температурой в летнее время года. Поэтому в данном разделе будет рассмотрен расчет вентиляционной системы с выбором подходящей системы кондиционирования для помещения, где ведется разработка проекта.

6.1 Расчет аспирационной системы для производственного помещения

Таблица 6.1 – Исходные данные

Город	Алматы
Параметры помещения (Д*Ш*В), м	8 * 4 * 3.2
Данные по оборудованию	
Количество компьютеров, шт.	3
Мощность $P_{об}$, кВт/ч	1.1
КПД η	0.7
Данные по источнику света	
Мощность $N_{осв}$, Вт/м^2	43
Вид источника освещения	люминесцентные лампы
Количество сотрудников	3 (мужчины)
Окна	
Количество, шт.	2
Площадь 1 окна, м^2	3
Расположение	ЮВ
Вид жалюзи	вертикальные металлические
Расчетное время суток, ч.	11-12
Температура в помещении	
Летом, ^{o}C	26
Зимой, ^{o}C	20
Вид положения работы	сидячее

6.1.1 Расчет тепловых нагрузок в помещении

В помещениях различного назначения действуют в основном тепловые нагрузки, возникающие снаружи помещения (наружные); тепловые нагрузки, возникающие внутри здания (внутренние).

6.1.2 Расчет тепловых нагрузок снаружи помещения

Наружные тепловые нагрузки подставлены следящими составляющими:

– теплопоступления или теплопотери в результате разности температур снаружи и внутри здания через стены, потолки, полы, окна и двери.

– разность температур снаружи здания и внутри него летом является положительной, в результате чего имеет место приток тепла снаружи во внутрь помещения; и наоборот – зимой эта разность является отрицательной и направления потока тепла меняется;

– теплопоступления от солнечного излучения через застекленные площади; данная нагрузка проявляется в форме ощущаемого тепла.

– теплопоступления от инфильтрации.

Следует отметить, что наружные тепловые нагрузки могут обладать различными свойствами, т. е. могут быть положительными в зависимости от времени года и времени суток.

В зависимости от времени года и времени суток наружные тепловые нагрузки могут быть положительными.

Теплопоступления и теплопотери в результате разности температур определяется по формуле (6.1):

$$Q_{\text{огр}} = V_{\text{пом}} * X_o * \left(t_{H\text{расч}} - t_{\text{Врасч}}\right) \quad (6.1)$$

где $V_{\text{пом}}$ – объем помещения (6.2), м^3

$$V_{\text{пом}} = 8 * 4 * 3.2 = 102.4 \text{ м}^3 ; \quad (6.2)$$

X_o – удельная тепловая характеристика, Вт/м^3 ^{o}С, X_o = 0.42;

$t_{\text{Нрасч}}$ – наружная температура. Для холодного периода - средняя температура самого холодного месяца в 13 часов, для теплого периода - средняя температура самого жаркого месяца в 13 часов.

$t_{\text{Врасч}}$ – внутренняя температура, выбирается с учетом комфортных условий или технологических требований, предъявляемых к производственным процессам.

Для теплого времени года(6.3):

$t_{\text{Нрасч}}$ = 28 ^{o}С

$t_{\text{Врасч}}$ = 26 ^{o}С

$$Q_{\text{огр}} = 102.4 * 0.42 * 2 = 86.016 \approx 86.02 \text{ Вт} \quad (6.3)$$

Для холодного времени года (6.4):

$t_{\text{Нрасч}}$ = -17 °С

$t_{\text{Врасч}}$ = 20 °С

$$Q_{\text{огр}} = 102.4 * 0.42 * 37 = 1591.296 \approx 1591.3 \text{ Вт} \quad (6.4)$$

Избыточная теплота солнечного излучения в зависимости от типа стекла почти до 90% поглощается средой помещения, остальная часть отражается. максимальная тепловая нагрузка достигается при максимальном уровне излучения, которое имеет прямую и рассеянную составляющие. Интенсивность излучения зависит от ширины местности, времени года и времени суток.

Теплопоступление от солнечного излучения через остекление определяется по формуле (6.5):

$$Q_P = (q^I F_O^I + q^{II} F_O^{II}) * \beta_{\text{с.з}} \quad (6.5)$$

где q^I, q^{II} - тепловые потоки от прямой и рассеянной солнечной радиации, Вт/м^2;

F_O^I, F_O^{II}- площади светового проема, облучаемые и необлучаемые прямой солнечной радиации, м^3;

$\beta_{\text{с.з}}$ - коэффициент теплопропускания. $\beta_{\text{с.з}} = 0.20$;

При отсутствие наружных затеняющих козырьков, ребер и т. д. для периода облучения остекления солнцем, когда его лучи проникают через окно в помещение $F_O^I = F_O$; $F_O^{II} = 0$, (6.6):

$$Q_P = q^I F_O * \beta_{\text{с.з}} = \left(q_{\text{вп}} + q_{\text{вр}}\right) * K_1^C * K_2 * n * S_O \quad (6.6)$$

$q_{\text{вп}}$, $q_{\text{вр}}$- тепловые потоки от прямой рассеянной радиации, Вт/м^2. Для широты в 44°СШ до полудня в 11-12 ч. При расположении Ю: $q_{\text{вп}}$, $q_{\text{вр}}$

$q_{\text{вп}} = 288$ Вт/м^2

$q_{\text{вр}} = 85$ Вт/м^2

F_O – площадь светового проема (6.7) (n – число окон, S_O – площадь одного окна);

$$F_O = n * S_o = 2 * 3 = 6 \text{ м}^2 \quad (6.7)$$

K_1 – коэффициент затемнения остекления переплетами (K_1^C – для облученных проемов).

$K_1^C = 0.72$

K_2 – коэфициент загрязнения остекления.

$K_2 = 0.9$
Тогда (6.8):

$$Q_P = (288 + 85) * 0.72 * 0.9 * 0.2 * 6 = 290 \text{ Вт} \quad (6.8)$$

Поступление солнечного излучения равно (6.9):

$$Q_P = 290 \text{ Вт} \quad (6.9)$$

6.1.3 Расчет тепловых нагрузок внутри помещения

Внутренние тепловые нагрузки в жилых, офисных или относящийся к сфере обслуживание помещения слагаются в основном из тепла:

– выделяемого людьми;

– выделяемого лампами и осветительными, электробытовыми приборами;

– выделяемого компьютерами, печатающими устройствами и фотокопировальными машинами и пр. (в офисных и других помещениях);

В производственных и технологических помещениях различного назначения дополнительными источниками тепловыделение могут быть: нагретое производственное оборудование; горячие материалы, в том числе жидкости различного рода полуфабрикаты; продукты сгорания и химических реакций.

Теплопоступления от людей зависит от интенсивности выполняемой работы и параметров окружающего воздуха. Тепло, выделяемое человеком, складывается из ощутимого (явного), то есть передаваемого в воздух помещения путём конвекции и лучеиспусканий, и скрытого тепла, затрачиваемого на испарение влаги с поверхности кожи и из легких.

Летом при $27\ ^oC \approx 28\ ^oC$ один мужчина выделяет явного тепла 51 Вт, а общего – 102 Вт (при работе стоя, легком движении). Выделение явного тепла составит (6.10):

$$Q_{\text{л}}^{\text{я}} = 51 * 3 = 153 \text{ Вт} \quad (6.10)$$

Выделение общего тепла (6.11):

$$Q_{\text{л}}^{\text{о}} = 102 * 3 = 306 \text{ Вт} \quad (6.11)$$

Зимой при $19\ ^oC \approx 20\ ^oC$ один мужчина выделяет явного тепла 82 Вт, а общего – 103 Вт. Тогда выделение явного тепла в помещении составит (6.12):

$$Q_{\text{л}}^{\text{я}} = 82 * 3 = 246 \text{ Вт} \quad (6.12)$$

Выделение общего тепла (6.13):

$$Q_{\text{л}}^{\text{о}} = 103 * 3 = 309 \text{ Вт} \quad (6.13)$$

Теплопоступление от осветительных приборов, оргтехники и оборудования рассчитывается следующим образом. Теплопоступление от ламп определяется по формуле (6.15):

$$Q_{\text{осв}} = \eta * N_{\text{осв}} * F_{\text{пол}} \quad (6.15)$$

где η - коэффициент перехода электрической энергии в тепловую (для люминесцентных ламп $\eta = 0.45 – 0.5$);

$N_{\text{осв}}$ – установленная мощность ламп ($N_{\text{осв}} = 43 \text{ Вт/м}^2$);

$F_{\text{пол}}$ – площадь пола ($F_{\text{пол}} = 8 * 4 = 32 \text{ м}^2$);

Тогда (6.16):

$$Q_{\text{осв}} = 0.45 * 43 * 32 = 619.2 \text{ Вт} \quad (6.16)$$

Тепло, выделяемое производственным оборудованием, определяется по формуле (6.17):

$$Q_{\text{об}} = N_{\text{уст}} * K \quad (6.17)$$

где $N_{\text{уст}}$ – паспортная мощность оборудования ($N_{\text{уст}} = 1.1 \text{ кВт/ч}$);

K - единиц оборудования ($K = 3$);

Тогда (6.18):

$$Q_{\text{об}} = 1.1 * 3 = 3.3 \text{ кВт} \quad (6.18)$$

Теплопритоки, возникающие за счет находящейся оргтехники – это 20% мощности оборудования (6.19):

$$Q_{\text{об}} = 1.1 * 3 * 0.2 = 0.66 \text{ кВт} \quad (6.19)$$

6.2 Расчет теплового баланса помещения

На основании выполненных расчетов поставим баланс теплопоступлений в помещении:

Летом (6.20):

$$Q_{\text{изб}} = 86.02 + 290 + 306 + 619.2 + 3300 + 660 = 5261.22 \text{ Дж} \quad (6.20)$$

Зимой (6.21):

$$Q_{\text{изб}} = 1592.3 + 290 + 309 + 619.2 + 3300 + 660 = 6770.5 \text{ Дж} \quad (6.21)$$

Т.к. тепловой баланс для зимы больше летнего теплового баланса, то рассчитаем тепло-напряжённость воздуха по формуле (6.22, 6.23):

$$Q_H = \frac{Q_{\text{изб}} * 860}{V_{\text{пом}}} \quad (6.22)$$

$$Q_H = \frac{6770.5 * 860}{102.4} = 56861{,}62 \text{ кал/м}^3 \approx 56{,}86 \text{ ккал/м}^3 \quad (6.23)$$

При $Q_H > 20$ ккал/м^3, $\Delta t = 8$ °C

Определение количества воздуха, необходимое для поступления в помещение (6.24) и расчет (6.25):

$$L = \frac{Q_{\text{изб}} * 860}{C * \Delta t * \gamma} \quad (6.24)$$

где $C = 0.24$ ккал/(кг°C) – теплоемкость воздуха,

$\gamma = 1.206$кг/м^3 – удельная масса приточного воздуха.

$$L = \frac{6770.5 * 860}{0.24 * 10^4 * 8 * 1.206} = 251.46 \frac{\text{м}^3}{\text{час}} \quad (6.25)$$

Определение кратности воздухообмена (6.26, 6.27):

$$n = \frac{L}{V_{\text{пом}}} \quad (6.26)$$

$$n = \frac{251{,}46}{102.4} = 2.46 \text{ час}^{-1} \quad (6.27)$$

6.3 Выбор кондиционера

Исходя из полученных данных, берется кондиционер типа FTYN35L/RYN35L в количестве 1 штук. Характеристики приведены в таблице 6.3.1 и 6.3.2.

Управление работой кондиционера производится с помощью фпульта, который позволяет задать режим работы кондиционера: обогрев, охлаждение, осушку, вентиляцию, ночной режим, энергосберегающий режим; задать требуемую температуру, которую должен поддерживать автоматический; выбрать режим работы вентилятора: настроить таймер, который включить или выключить кондиционер в заданное время; автоматически регулировать положение направляющих штор и изменить таким образом направление воздушного потока.

Таблица 6.3.1 – Характеристики выбранного кондиционера.

Электропитание	Расход воздуха внутреннего блока	Расход воздуха внешнего блока	Производительность по теплу	Производительность по холоду	Мощность компрессора	Электронагреватель
В	$м^3/ч$	$м^3/ч$	кВт	кВт	кВт	кВт
220	606	786	3.47	3.3	2.1	3

Таблица 6.3.2 – Размеры выбранного кондиционера.

Размеры внутреннего блока (В*Ш*Г)	Вес внутреннего блока	Хладагент	Уровень шума внутреннего блока
мм	кг	-	ДБ
288*800*206	9	R410A	27/29/41

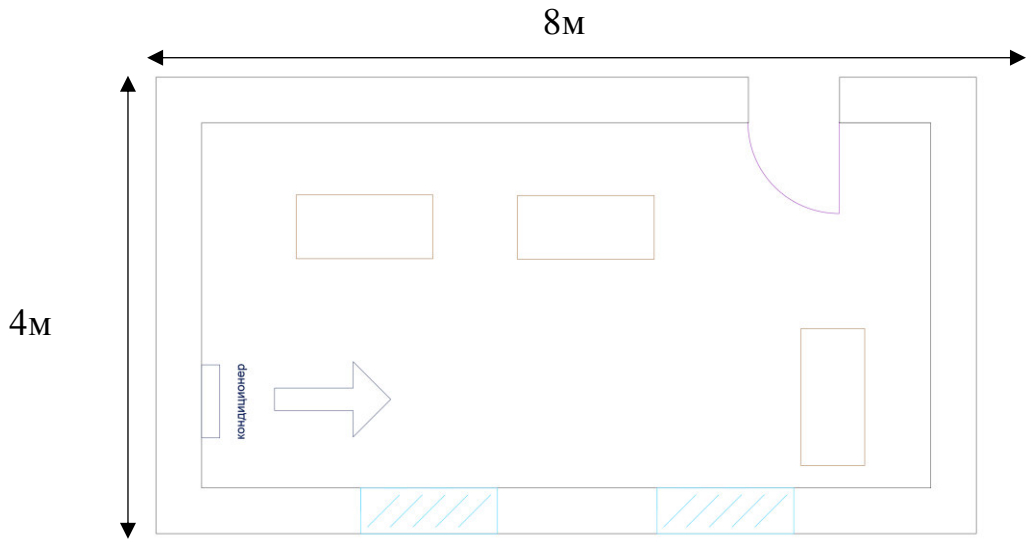

Рисунок 6.3.1 – Схема расположение кондиционера в производственном помещении

6.4 Вывод

В разделе «безопасность жизнедеятельности» был проведен расчет вентиляционной системы для обеспечения оптимальных условии жизнедеятельности человека для офиса ТОО «Almaz-Group». В данном разделе был проведен расчет параметров микроклимата и воздушной среды, количество теплоты, выделяемое при работе людей и оборудования, необходимый воздухообмен и подходящую систему кондиционирования воздуха. Всем необходимым параметрам (указанным выше) удовлетворяет кондиционер FTYN35L/RYN35L.

Заключение

В ходе выполнения дипломного проекта была разработана система, которая контролирует и ведет учет автомобилей на парковочной стоянке, а также обеспечивает определенный уровень безопасности.

При выполнении дипломного проекта были выполнены следующие задачи:

– был проведен анализ существующих аналогов систем управления парковками;

– определены основные требований к системе учета и регистрации;

– спроектирована базы данных, программная часть и аппаратная часть;

– проведен анализ существующих технологий для решения поставленных задач;

– проведен анализ и выбор компонентов для сборки разработанного проекта;

– разработка системы контроля и учета;

– настройка сервера и облачных услуг;

– произведено экономическое обоснование целесообразности разрабатываемого продукта, где были сделано следующее заключение:
цена реализации окупает все затраты, потрачены на разработку. Прибыл от реализации проекта равна 515 017 тг;

– также были исследованы условия труда в помещении, где велась разработка, и были предложены мероприятия по улучшению качества воздуха в помещении.

Список литературы

1 Jatuporn C., Udomporn S., Satien T. Smart Parking: An Application of Optical Wireless Sensor Network / URL: https://ieeexplore.ieee.org/document/4090136 (дата обращения: 02.02.2019)

2 Susan Shaheen, Caroline Rodier, Amanda M. Eaken. Smart Pariking Management: A Bay Area Rapid Transit (BART) District Parking Demonstration / URL: https://merritt.cdlib.org/d/ark:%252F13030%252Fm5m90bc4/2/producer%252FPRR-2005-05.pdf (дата обращения: 02.02.2019)

3 City of Edmonton: Evolution of Parking [Электронный ресурс] / Youtube / URL: https://www.youtube.com/watch?v=84YaqJMgXJM (дата обращения: 05.02.2019)

4 Wikipedia: Парковочные часы / URL: https://en.wikipedia.org/wiki/Parking_meter (дата обращения: 05.02.2019)

5 Wikipedia: Парковки жилых комплексов / URL: https://no.wikipedia.org/wiki/Bolig (дата обращения: 05.02.2019)

6 АО «Thon Eiendom»: ParkLink, Автоматичесая система управления паркингом / URL: https://strommenstorsenter.no/aktiviteter-og-nyheter/nyheter/parklink/ (дата обращения: 08.02.2019)

7 ТОО «Hi-Tec Security Systems», Автоматизированные платные парковки / URL: http://www.htss.kz/ (дата обращения: 08.02.2019)

8 Nordlys, Интернет доска новостей Норвегии: Системы парковок аэропортов / URL: https://www.nordlys.no/samferdsel/avinor/tromso/denne-uken-endres-parkeringssystemet-pa-tromso-lufthavn/s/5-34-959775 (дата обращения: 08.02.2019)

9 Faheem, S.A. Mahmud, G.M. Khan, M. Rahman and H. Zafar. Intelligent Car Parking Systems / URL: http://www.scielo.org.mx/pdf/jart/v11n5/v11n5a11.pdf (дата обращения: 08.02.2019)

10 «Умная парковка: адаптивное управление навигацией, освещением и зарядными станциями для электромобилей» / URL: https://habr.com/ru/company/intems/blog/443146/ (дата обращения: 08.02.2019)

11 «EasyPark – Система умного паркинга» / URL: https://easypark.no/ (дата обращения: 08.02.2019)

12 «Введение в модель данных SQL» / С.Д. Кузнецов – М.: Национальный Открытый Университет «ИНТУИТ», 2016.

13 Бен Лин. / «Волшебство Git» – 2016. / URL: http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/ch01.html (дата обращения: 08.02.2019)

14 «SQL или NoSQL — вот в чём вопрос» / URL: https://m.habr.com/ru/company/ruvds/blog/324936/ (дата обращения: 08.02.2019)

15 Google Cloud Platform (статья Википедии) / URL: https://ru.wikipedia.org/wiki/Google_Cloud_Platform (дата обращения: 12.02.2019)

16 «BitBucket – ведро битов» (статья Википедии) / URL: https://ru.wikipedia.org/wiki/Bitbucket (дата обращения: 12.02.2019)

17 Python (статья Википедии)/ URL: https://ru.wikipedia.org/wiki/Python (дата обращения: 12.02.2019)

18 «OpenCV – Компьютерное зрение» / URL: https://dic.academic.ru/dic.nsf/ruwiki/450029 (дата обращения: 12.02.2019)

19 MySQL (статья Википедии) / URL: https://ru.wikipedia.org/wiki/MySQL (дата обращения: 12.02.2019)

20 «Raspberry Pi для начинающих» / URL: http://edurobots.ru/raspberry-pi-dlya-nachinayushhix/ (дата обращения: 12.02.2019)

21 «Raspberry Pi 3B – характеристики» / URL: https://www.raspberrypi.org/magpi/raspberry-pi-3bplus-specs-benchmarks/ (дата обращения: 12.02.2019)

22 «MySQL: особенности и сферы применения» / URL: https://www.bytemag.ru/articles/detail.php?ID=6547 (дата обращения: 12.02.2019)

23 Mark Lutz / «Programming Python» - четвертый выпуск серии «Powerful Object-Oriented Programming» [Электронный ресурс] / URL: https://doc.lagout.org/programmation/python/Programming%20Python%2C%204th%20Edition%20%282010%29.pdf (дата обращения: 12.02.2019)

Приложение А
(обязательное)

Техническое задание

Техническое задание для разработки системы автоматической регистрации парковочных мест

1. Общие требования:

- Наименование разрабатываемой системы:
  - Система автоматической регистрации парковочных мест (САРП).
- Цель разработки:
  - Обеспечить контроль автомобилей, желающих попасть на паркинг;
  - Контролировать пропуск автомобилей по государственному регистрационному номеру.
  - Пропускать автомобили только с определенными регистрационными номерами;
  - Не пропускать автомобили, внесенные в черный список;
  - Вести учет прибывших и выбывших автомобилей с паркинга;
  - Возможность резервации парковочных мест.
- Предлагаемые технологии для разработки системы (на выбор разработчика):
  - Google Cloud Platform;
  - Amazon Web Services;
  - OpenCV;
  - SimpleCV;
  - Google Tesseract;
  - Luminoth;
  - Одноплатный компьютер (для стойки управления).
- Выбор архитектуры построения:
  - Клиент-Сервер.
- Предлагаемые языки и технологии программирования:
  - Python;
  - Ruby;
  - Kotlin;
  - Java;
  - C#;
  - Rust;
  - C++.
- Общий объем программной части системы, Мб
  - Не более 200 Мб.

2. Технические требования:

– Требования к программному обеспечению:
  – Умеренная скорость обработки входящей информации;
  – Кроссплатформенность;
  – Возможность работы с облачными решениям;
  – Код программы должен быть минималистичным и понятным.
– Требования к аппаратному обеспечению:
  – Наличие минимального количества оборудования;
  – Работа с IP-камерой;
  – Стойка управления должна обладать достаточной вычислительной мощностью для обработки графических изображений;
  – Возможность подключаться к сети Интернет.
– Тестирование и отладка системы:
  – Проверка работоспособности системы на тестовом объекте;
  – Тестирование системы на вероятность ложного позитива;
  – Тестирование системы на сбой.

3. Специфические требования:

– Адаптивность системы:
  – Должна присутствовать возможность адаптировать систему под различные условия, в зависимости оборудования, присутствующее на парковочной стоянке;
  – Должна быть возможность добавления функционала к системе без каких-либо серьезных изменений программного и аппаратного обеспечения.

4. Экономические требования:

– Расчет стоимости системы и стоимости разработки программного обеспечения (подлежит обсуждению):
  – Стоимость готового продукта 2 000 000тг;
  – Стоимость разработки 1 000 000тг.
– Потенциальные клиенты и области применения:
  – Жилые комплексы;
  – Бизнес центры;
  – Частные владельцы авто-паркингов.

Приложение Б
(обязательное)

Листинг программы

Proj.py

```
from Procrssor import Procrssor
import cv2
import MySQLdb
import DbQueries
import RPi.GPIO as GPIO
import time
from google.cloud import vision


def init_RASPBERRY():
    global trigger, echo, gate_signal
    GPIO.setmode(GPIO.BCM)

    trigger = 20
    echo = 16
    gate_signal = 23

    GPIO.setup(gate_signal, GPIO.OUT)
    GPIO.setup(trigger, GPIO.OUT)
    GPIO.setup(echo, GPIO.IN)

    GPIO.output(gate_signal, False)
    GPIO.output(trigger, False)


def open_gate():
    global gate_signal
    GPIO.output(gate_signal, True)

def close_gate():
    global gate_signal
    GPIO.output(gate_signal, False)
    time.sleep(2)


def init_googleapi():
    global VISION_API_CLIENT
    # Instantiates a client
    VISION_API_CLIENT = vision.ImageAnnotatorClient()

def detect_text(client, img):
    response = client.text_detection(image=img)
    texts = response.text_annotations
```

```
    return texts

def detect_document(client, img):


response = client.document_text_detection(image=img)
    document = response.full_text_annotation
    return document


def detect_lables(client, img):
    response = client.label_detection(image=img)
    labels = response.label_annotations
    return labels

def init_db():
    global dbconnect, cursor
    dbconnect = MySQLdb.connect(HOST, USER, PASSWORD, DB, charset='utf8')
    cursor = dbconnect.cursor()


def get_number_plate():
    global has_error, VISION_API_CLIENT
    error_counter = 0
    number_plate = None
    while 1:
        if error_counter == 5:
            has_error = True
            break
        cap = cv2.VideoCapture(0)
        ret, frame = cap.read()
        cap.release()

        imagecv = frame

        orig = imagecv.copy()
        binary_img = Procrssor.get_binary_image(imagecv)
        content = Procrssor.get_bytestring(binary_img)

        image = vision.types.Image(content=content)
        texts = detect_document(VISION_API_CLIENT, image)
        text = texts.text.replace('\n', '')
        text = text.replace(' ', '')
        number_plate = Procrssor.get_plate_text(text.upper())
        if number_plate != -1:
            break
        error_counter += 1
    return number_plate
```

```
def cleanup():
    global dbconnect

    dbconnect.close()
    GPIO.cleanup()


if __name__ == '__main__':

    trigger = 0
    echo = 0

    gate_signal = 0

    init_RASPBERRY()

    VISION_API_CLIENT = None
    dbconnect = None
    cursor = None

    init_googleapi()
    init_db()

    is_reservation_expired = False
    has_records = False
    has_error = False
    is_blacklisted = False
    is_occupied = False

    while True:
        sensor_entrance = 0

        sensor_exit = 0
        gate = 0
        GPIO.output(trigger, True)
        time.sleep(0.00001)
        GPIO.output(trigger, False)

        while GPIO.input(echo) == 0:
            pulse_start = time.time()

        while GPIO.input(echo) == 1:
            pulse_end = time.time()
            time.sleep(2)
        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17150
        distance = round(distance, 2)
```

```
if distance < 2:
            sensor_entrance = 1
        else:
            sensor_entrance = 0
            close_gate()
        # ON ENTER
        if sensor_entrance:
            plate = get_number_plate()
            if not has_error:
                # Blacklisted vehicle check
                try:
                    query_blacklisted =
DbQueries.QUERY_SELECT_BLACKLISTED_VEHICLE.format(plate)
                    cursor.execute(query_blacklisted)
                    if cursor.rowcount != 0:
                        is_blacklisted = True
                        result = cursor.fetchone()
                        ban_reason = result[2]
                        ban_date = result[3]
                        print('Registration number: {}\n'
                              'Ban reason: {}\n'
                              'Ban date: {}'.format(plate, ban_reason, ban_date))
                    else:
                        is_blacklisted = False
                except MySQLdb.Error as error:
                    print('ERROR GETTING BLACKLISTED VEHICLE INFORMATION:
{}'.format(error))

                if not is_blacklisted:
                    query_regnum = DbQueries.QUERY_SELECT_BY_REG_NUM.format(plate)
                    try:
                        cursor.execute(query_regnum)
                        parking_lot_id = None
                        parking_slot_id = None
                        if cursor.rowcount == 0:
                            query_guest =
DbQueries.QUERY_SELECT_GUEST_BY_REG_NUM.format(plate)
                            try:
                                # print(query_guest)
                                cursor.execute(query_guest)
                                if cursor.rowcount != 0:
                                    result = cursor.fetchone()
                                    guest_id = result[0]
                                    guest_name = result[1] + result[2]
                                    reg_num = result[4]

                                    query_reservation =
DbQueries.QUERY_SELECT_RESERVATION_BY_GUEST.format(guest_id)
```

```
                                try:                    cursor.execute(query_reservation)
                            result = cursor.fetchone()
                            reservation_date = result[3]
                            parking_slot_id = result[1]
                            is_occupied = DbQueries.select_parking_slot_occupied(cursor,
parking_slot_id)
                            print('Guest: {}\n'
                                'Registration_number: {}\n'
                                'Reservation_date: {}'.format(guest_name, reg_num,
reservation_date))
                            is_reservation_expired =
Procrssor.has_reservations_expired(reservation_date)

                            if is_reservation_expired:
                               print("Reservation Expired")
                            else:
                               query_parking_lot =
DbQueries.QUERY_SELECT_PARKING_LOT_ID_BY_SLOT.format(
                                  parking_slot_id)
                               try:
                                  # print(query_parking_lot)
                                  cursor.execute(query_parking_lot)
                                  result = cursor.fetchone()
                                  parking_lot_id = result[0]
                               except MySQLdb.Error as error:
                                  print("ERROR GETTING PARKING LOT GUEST:
{}".format(error))
                                  exit(-7)
                         except MySQLdb.Error as error:
                            print("ERROR GETTING RESERVATION GUEST: {}".format(error))
                            exit(0)
                         has_records = True
                      else:
                         print('Not in system')
                         has_records = False
                   except MySQLdb.Error as error:
                      print("ERROR GETTING GUEST: {}".format(error))
                      exit(-1)
                else:
                   result = cursor.fetchone()
                   user_id = result[1]
                   reg_num = result[2]
                   vehicle_type = result[3]
                   vehicle_color = result[4]
                   vehicle_brand = result[5]
                   # Get Resident information
                   user_name, parking_lot_id, parking_slot_id, is_occupied =
DbQueries.select_user_by_user_id(
                      cursor, user_id)
```

```
                    print('Registration Number: {}\n'
                          'Resident: {}\n'
                          'Car brand: {}\n'
                          'Car type: {}\n'
                          'Color: {}'.format(reg_num, user_name, vehicle_brand, vehicle_type,
                                             vehicle_color))
                    has_records = True
                if has_records:
                    if is_occupied:
                        print('Parking slot is occupied')
                    else:
                        if not is_reservation_expired:
                            DbQueries.update_parking_slot_occupied(dbconnect, cursor, 1,
parking_slot_id)
                            query_vehicle_event =
DbQueries.QUERY_INSERT_VEHICLE_EVENT_IN.format(
                                parking_lot_id, Procrssor.get_formatted_datetime())
                            query_last_ve_id = DbQueries.QUERY_SELECT_LAST_INSERTED_ID
                            ve_id = None
                            try:
                                cursor.execute(query_vehicle_event)
                                dbconnect.commit()
                                cursor.execute(query_last_ve_id)
                                if cursor.rowcount != 0:
                                    result = cursor.fetchone()
                                    ve_id = result[0]
                            except MySQLdb.Error as error:
                                dbconnect.rollback()
                                print("ERROR ON INSERT VEHICLE EVENT {}".format(error))
                                exit(-3)
                            query_parked_vehicle =
DbQueries.QUERY_INSERT_PARKED_VEHICLE.format(ve_id,
                                                                         parking_slot_id)
                            try:
                                cursor.execute(query_parked_vehicle)
                                dbconnect.commit()
                                open_gate()
                            except MySQLdb.Error as error:
                                dbconnect.rollback()
                                print('ERROR INSERT PARKED VEHICLE: {}'.format(error))
                        else:
                            print('Reservation has expired')
            except MySQLdb.Error as error:
                print("ERROR GETTING DATA BY REGISTRATION NUMBER:
{}".format(error))
        else:
            print('Vehicle is blacklisted')
    else:
```

```
            print('ERROR WHILE READING NUMBER')
```

Продолжение приложения Б

```
        # ON EXIT
        if sensor_exit:
            plate = get_number_plate()
            if not has_error:
                query_regnum = DbQueries.QUERY_SELECT_BY_REG_NUM.format(plate)
                try:
                    cursor.execute(query_regnum)
                    parking_lot_id = None
                    parking_slot_id = None
                    if cursor.rowcount == 0:
                        query_guest =
DbQueries.QUERY_SELECT_GUEST_BY_REG_NUM.format(plate)
                        try:
                            cursor.execute(query_guest)
                            if cursor.rowcount != 0:
                                result = cursor.fetchone()
                                guest_id = result[0]
                                query_reservation =
DbQueries.QUERY_SELECT_RESERVATION_BY_GUEST.format(guest_id)
                                try:
                                    cursor.execute(query_reservation)
                                    result = cursor.fetchone()
                                    parking_slot_id = result[1]
                                    DbQueries.update_parking_slot_occupied(dbconnect, cursor, 0,
parking_slot_id)
                                except MySQLdb.Error as error:
                                    print("ERROR GETTING RESERVATION GUEST:
{}".format(error))
                        except MySQLdb.Error as error:

                            print("ERROR GETTING GUEST: {}".format(error))
                    else:
                        result = cursor.fetchone()
                        user_id = result[1]

                        # Get Resident information
                        _, _, parking_slot_id, _ = DbQueries.select_user_by_user_id(
                            cursor, user_id)
                        DbQueries.update_parking_slot_occupied(dbconnect, cursor, 0, parking_slot_id)
                    DbQueries.update_vehicle_event(dbconnect, cursor, plate)
                    DbQueries.delete_parked_vehicle(dbconnect, cursor, plate)
                    open_gate()
                except MySQLdb.Error as error:
                    print("ERROR GETTING DATA BY REGISTRATION NUMBER:
{}".format(error))
            else:
                print('ERROR WHILE READING NUMBER')
```

DbQuery.py

```
import MySQLdb
from Procrssor import Procrssor
# SELECT QUERIES
QUERY_SELECT_PARKING_SLOT_BY_ID = 'SELECT * FROM parking_slot WHERE
parking_slot_id = {};'
QUERY_SELECT_BY_REG_NUM = 'SELECT * FROM registered_vehicle WHERE
registration_number="{}";'
QUERY_SELECT_USER_BY_REG_NUMBER = 'SELECT u.first_name, u.last_name,
u.parking_lot_id, ps.parking_slot_id, ' \
                          'ps.is_occupied ' \
                          'FROM user u ' \
                          'INNER JOIN parking_slot ps ON ps.owner_id = u.user_id ' \
                          'WHERE user_id={};'
QUERY_SELECT_GUEST_BY_REG_NUM = 'SELECT * FROM guest WHERE
vehicle_registration_number="{}";'

QUERY_SELECT_RESERVATION_BY_GUEST = 'SELECT * FROM reservation WHERE
guest_id = {};'

QUERY_SELECT_PARKING_LOT_ID_BY_SLOT = 'SELECT pl.parking_lot_id FROM
parking_lot pl ' \
                              'INNER JOIN zone z ' \
                              'ON z.parking_lot_id = pl.parking_lot_id ' \
                              'INNER JOIN floor f ' \
                              'ON f.zone_id = z.zone_id ' \
                              'INNER JOIN parking_slot ps ' \
                              'ON ps.floor_id = f.floor_id ' \
                              'WHERE ps.parking_slot_id = {};'
QUERY_SELECT_BLACKLISTED_VEHICLE = 'SELECT * FROM blacklisted_vehicle ' \
                           'WHERE registration_number = "{}";'

QUERY_SELECT_LAST_INSERTED_ID = 'SELECT LAST_INSERT_ID();'
# SELECT QUERIES END

# INSERT QUERIES
QUERY_INSERT_VEHICLE_EVENT_IN = 'INSERT INTO vehicle_event(parking_lot_id,
time_in) ' \
                        'VALUES({}, "{}");'

QUERY_INSERT_PARKED_VEHICLE = 'INSERT INTO parked_vehicle(vehicle_event_id,
parking_slot_id)' \
                       'VALUES({}, {});'
# INSERT QUERIES END
# UPDATE QUERIES

QUERY_UPDATE_PARKING_SLOT = 'UPDATE parking_slot SET is_occupied = {} WHERE
parking_slot_id = {};'
QUERY_UPDATE_VEHICLE_EVENT_OUT_BY_REG_NUM = 'UPDATE vehicle_event ' \
```

```
                'SET time_out = "{0}" ' \

                                'WHERE vehicle_event_id = ( ' \
                                'SELECT pv.vehicle_event_id FROM parked_vehicle pv ' \
                                'WHERE pv.parking_slot_id = ( ' \
                                'SELECT ps.parking_slot_id FROM parking_slot ps ' \
                                'INNER JOIN reservation r ' \
                                'ON r.parking_slot_id = ps.parking_slot_id ' \
                                'INNER JOIN guest g ' \
                                'ON g.guest_id = r.guest_id ' \
                                'WHERE g.vehicle_registration_number = "{1}") ' \
                                'OR pv.parking_slot_id = ( ' \
                                'SELECT ps.parking_slot_id FROM parking_slot ps ' \
                                'INNER JOIN user u ' \
                                'ON u.user_id = ps.owner_id ' \
                                'INNER JOIN registered_vehicle rv ' \
                                'ON u.user_id = rv.user_id ' \
                                'WHERE rv.registration_number = "{1}"));'
# UPDATE QUERIES END

# DELETE QUERIES
QUERY_DELETE_PARKED_VEHICLE_BY_REGNUM = 'DELETE FROM parked_vehicle ' \
                                'WHERE vehicle_event_id = ( ' \
                                'SELECT ve.vehicle_event_id from vehicle_event ve  ' \
                                'WHERE vehicle_event_id = ( ' \
                                'SELECT ve.vehicle_event_id FROM vehicle_event ve ' \
                                'INNER JOIN (SELECT * FROM parked_vehicle) as pv ' \
                                'ON ve.vehicle_event_id = pv.vehicle_event_id ' \
                                'WHERE pv.parking_slot_id = ( ' \
                                'SELECT ps.parking_slot_id FROM parking_slot ps ' \
                                'INNER JOIN reservation r ' \
                                'ON r.parking_slot_id = ps.parking_slot_id ' \
                                'INNER JOIN guest g ' \
                                'ON g.guest_id = r.guest_id ' \
                                'WHERE g.vehicle_registration_number = "{0}") ' \
                                'OR pv.parking_slot_id = ( ' \
                                'SELECT ps.parking_slot_id FROM parking_slot ps ' \
                                'INNER JOIN user u ' \
                                'ON u.user_id = ps.owner_id ' \
                                'INNER JOIN registered_vehicle rv ' \
                                'ON u.user_id = rv.user_id ' \
                                'WHERE rv.registration_number = "{0}")));'

# DEKETE QUERIES END

def update_parking_slot_occupied(dbconnect, cursor, is_occupied, parking_slot_id):
    try:
        query_update_parking_slot = QUERY_UPDATE_PARKING_SLOT.format(is_occupied,
parking_slot_id)
```

```
        cursor.execute(query_update_parking_slot)
        dbconnect.commit()
    except MySQLdb.Error as error:
        dbconnect.rollback()
        print('ERROR UPDATING PARKING SLOT OCCUPANCY: {}'.format(error))


def select_parking_slot_occupied(cursor, parking_slot_id):
    is_occupied = False
    try:
        query_parking_slot =
QUERY_SELECT_PARKING_SLOT_BY_ID.format(parking_slot_id)
        cursor.execute(query_parking_slot)
        result = cursor.fetchone()
        is_occupied = result[3]
    except MySQLdb.Error as error:
        print('ERROR GETTING PARKING SLOT: {}'.format(error))
    finally:
        return is_occupied


def select_user_by_user_id(cursor, user_id):
    user_name = ''
    parking_lot_id = 0
    parking_slot_id = 0
    is_occupied = False
    try:
        query_user = QUERY_SELECT_USER_BY_REG_NUMBER.format(user_id)
        cursor.execute(query_user)
        result = cursor.fetchone()
        user_name = result[0] + ' ' + result[1]
        parking_lot_id = result[2]
        parking_slot_id = result[3]
        is_occupied = result[4]
    except MySQLdb.Error as error:
        print("ERROR ON GETTING USER INFORMATION: {}".format(error))
    finally:
        return user_name, parking_lot_id, parking_slot_id, is_occupied


def update_vehicle_event(dbconnect, cursor, registration_number):
    try:
        query =
QUERY_UPDATE_VEHICLE_EVENT_OUT_BY_REG_NUM.format(Procrssor.get_formatted
_datetime(), registration_number)
        cursor.execute(query)
        dbconnect.commit()
    except MySQLdb.Error as error:
        dbconnect.rollback()
        print('ERROR UPDATE VEHICLE EVENT: {}'.format(error))
```

```
def delete_parked_vehicle(dbconnect, cursor, registration_number):
    try:
        query =
QUERY_DELETE_PARKED_VEHICLE_BY_REGNUM.format(registration_number)
        cursor.execute(query)
        dbconnect.commit()
    except MySQLdb.Error as error:
        dbconnect.rollback()
        print('ERROR ON DELETE PARKED VEHICLE: {}'.format(error))

            Parocessor.py
import re
import cv2
import datetime
import time

class Procrssor:

def __init__(self):
        pass

    @staticmethod
    def get_bytestring(image):
        return cv2.imencode('.jpg', image)[1].tostring()

    @staticmethod
    def get_binary_image(image):
        gray = cv2.cvtColor(image.copy(), cv2.COLOR_RGB2GRAY)
        # cv2.imshow('111', gray)
        smooth = cv2.bilateralFilter(gray, 11, 21, 21)
        # cv2.imshow('1111', smooth)
        binary = cv2.adaptiveThreshold(smooth, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 55, 5)
        return binary

    @staticmethod
    def canny(image):
        gray = cv2.cvtColor(image.copy(), cv2.COLOR_RGB2GRAY)
        # blur = cv2.bilateralFilter(gray, 9, 17, 17)
        # eq_histogram = cv2.equalizeHist(gray)
        canny_img = cv2.Canny(gray, 79, 100)
        return canny_img
    @staticmethod
    def get_plate_text(text):
        result = re.findall(r'\d{3}\s?[A-Z]+\s?\d{2}|'
                            r'[A-Z]\s\d{1,3}\s[A-Z]{3}|'
                            r'[A-Z]\s?\d{1,4}\s\d{1,2}', text)
        if len(result) < 1:
```

```
        print('No plates found')
    if len(result) > 1:
        return -1
    else:
        result = result[0].replace(' ', '')
        return result
@staticmethod
def contrast(image):
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    cv2.imshow("lab", lab)

    l, a, b = cv2.split(lab)
    cv2.imshow('l_channel', l)
    cv2.imshow('a_channel', a)
    cv2.imshow('b_channel', b)

    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)
    cv2.imshow('CLAHE output', cl)

    limg = cv2.merge((cl, a, b))
    cv2.imshow('limg', limg)

    final = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    return final
@staticmethod
def increase_brightness(img, value=30):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value
    final_hsv = cv2.merge((h, s, v))
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img

@staticmethod
def get_formatted_datetime():
    return datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

@staticmethod
def get_mktime(string):
    return time.mktime(time.strptime(string, '%Y-%m-%d %H:%M:%S'))

@staticmethod
def has_reservations_expired(reservation_date):
    reservation_mktime = Procrssor.get_mktime(reservation_date)
    now = time.time()
    return True if reservation_mktime < now else False
```

Приложение В
(обязательное)

Акты внедрения

«Almaz-Group» ЖШС | 100940000885 БСН
Қазақстан Республикасы, 050066, Алматы қ., Тоқтоғул к., 75
«ForteBank» АҚ, Алматы қ. | SWIFT IRTYKZKA | IBAN KZ5496502F0008920644
Тел. +7 (727) 354 53 98, +7 (701) 266 7499 | info@almaz-group.kz| www.almaz-group.kz

ТОО «Almaz-Group» | БИН 100940000885
Республика Казахстан, 050066, г. Алматы, ул. Токтогула, 75
АО «ForteBank», г. Алматы | SWIFT IRTYKZKA | IBAN KZ5496502F0008920644
Тел. +7 (727) 354 53 98, +7 701 266 7499 | info@almaz-group.kz | www.almaz-group.kz

Исх. №0515-1 от «15» мая 2019 г.

Акт внедрения автоматической системы регистрации парковочных мест «САРП»

Настоящий Акт свидетельствует о том, что система контроля и учета автотранспортных средств «САРП», разработанная Бакиевым Мухаммедом, внедрена в ТОО «Almaz-Group».

Процесс внедрения проходил с 22 по 26 апреля 2019 г.

Заявленные характеристики системы предполагали наличие следующих основных возможностей:

- Распознавание государственного регистрационного номера;
- Пропуск автомобилей строго по регистрационным номерам, занесенным в базу данных;
- Возможность резервации парковочных мест;
- Возможность добавления автомобилей в черный список;
- Для обеспечения надежности и безопасности база дынных должна храниться на виртуальном накопителе, например, Amazon Web Services или Google Cloud Platform.

В ходе опытной эксплуатации системы подтверждено, что она обладает всеми заявленными возможностями.

На момент подписания настоящего Акта система установлена на объекте, принадлежащем ТОО «Almaz-Group».

Директор ТОО "Almaz-Group" **Джалилов А.У.**