МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
**Высшая школа электроники и компьютерных наук**
**Кафедра системного программирования**

РАБОТА ПРОВЕРЕНА

Рецензент
Инженер ВШЭУ

_____ В.В. Переведенцев

“__”_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

“__”_____ 2019 г.

# РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ВЫЧИСЛЕНИЯ ОПТИМАЛЬНОГО ПУТИ ДЛЯ СБОРЩИКА ТОВАРА НА СКЛАДЕ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2019.115-074.ВКР

Научный руководитель,
кандидат физ.-мат. наук
_____Р.Э. Шангин

Автор работы,
студент группы КЭ-401
_____И.С. Талько

Ученый секретарь
(нормоконтролер)
_____ О.Н. Иванова
“___”_____ 2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет**
**(национальный исследовательский университет)»**
**Высшая школа электроники и компьютерных наук**
**Кафедра системного программирования**

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
09.02.2019

**ЗАДАНИЕ**
**на выполнение выпускной квалификационной работы бакалавра**
студенту группы КЭ-401
Талько Игорь Сергеевич,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от «25» Апреля 2019 № 899)

Разработка приложения для вычисления оптимального маршрута сборщика товара на складе.

2. **Срок сдачи студентом законченной работы:** 04.06.2019.

3. **Исходные данные к работе**

3.1. Наука и образование МГТУ им. Н.Э. Баумана. Электрон. Журн. 2015 №4. С.270-310.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ аналогов;
4.2. Спроектировать приложение;
4.3. Реализовать приложение;
4.4. Протестировать приложение.

5. **Дата выдачи задания:** 08.02.2019.

**Научный руководитель**

Доцент кафедры СП,
кандидат физ.-мат. наук Р.Э. Шангин

**Задание принял к исполнению** И.С. Талько

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ ..... 5
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ ..... 7
1.1. Предметная область ..... 7
1.2. Постановка задачи ..... 8
1.3. Анализ аналогичных проектов ..... 10
2. ПРОЕКТИРОВАНИЕ ..... 13
2.1. Функциональные требования ..... 13
2.2. Нефункциональные требования ..... 13
2.3. Архитектура системы ..... 14
2.3.1. Общий вид системы ..... 14
2.3.2. Подключение ..... 14
2.3.3. Работа с приложением. Клиентская часть ..... 15
2.4. Используемые алгоритмы ..... 15
2.4.1. Построение графа ..... 15
2.4.2. Поиск оптимального пути ..... 16
3. РЕАЛИЗАЦИЯ ..... 18
3.1. Средства для реализации ..... 18
3.2. Реализация модулей системы ..... 19
3.2.1. Структура сервера ..... 24
3.2.2. Структура клиента ..... 23
4. ТЕСТИРОВАНИЕ ..... 28
4.1. Тестирование сервера ..... 28
4.2. Тестирование клиента ..... 28
4.3. Тестирование верстки на устройствах ..... 29
4.4. Тестирование затрат ресурсов на устройствах ..... 30
4.5. Тестирование времени выполнения алгоритма ..... 34

ЗАКЛЮЧЕНИЕ ..... 35
ЛИТЕРАТУРА ..... 36

# ВВЕДЕНИЕ

Задача поиска минимального пути очень востребована в современном мире, по причине того, что необходимо выполнить работу быстро, не снижая при этом качество. Любая задержка грозит лишними расходами на производстве. И при сборке товара на складе данная проблема поднимается наиболее часто.

Склады бывают разных размеров [9] отсюда создается проблема маршрутизации. Весьма сложно собрать товар в сроки на складах, размеры которых превышают 5 тыс. $м^2$. И человеку необходимо не только знать, где находится нужные ячейки, но и решить проблему задачи коммивояжера посетить все точки за минимальное количество времени.

По этой причине была поставлена цель разработки приложения для вычисления оптимального пути для сборщика товара на складе.

## Структура и объем работы

Работа состоит из введения, пяти глав, заключения и библиографии. Объем работы составляет 33 страниц, для написания работы использовалось 16 источников.

В первой главе «Анализ предметной области», была представлена актуальность работы, рассматривались параметры склада, а также характеристика работ, производимых на складе. Была поставлена задача, после которой были проведен анализ существующих на данный момент средств для вычисления оптимального пути на складе.

Вторая глава «Проектирование», содержит в себе функциональные и нефункциональные требования к приложению, спроектированные компоненты системы и их описание, так же были описаны алгоритмы, которые будут использоваться для решения поставленной задачи из прошлой главы.

В третьей главе «Реализация», рассматриваются технологии, которые были использованы для достижения цели. Описаны основы реализации для нахождения оптимального пути на складе.

Четвертая глава «Тестирование», содержит в себе протоколы функционального тестирования готовой системы. Наиболее важные тестирования заключался в наблюдении затрачиваемых ресурсов на поддержание рабочего состояние приложения, и как долго приложение будет обрабатывать данные и возвращать результат.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Предметная область

Причина актуальности этой темы состоит в том, что на огромных складах очень сложно ориентироваться и тем самым сложность приходится на задачу где необходимо построить такой маршрут, который займет меньше всего времени, и если количество необходимых товаров превышает десяток, то такая объемная задача займет много времени на этапе подготовки маршрута с точки зрения затрачиваемого времени.

Под складом [8] будем понимать помещение, в котором хранятся детали, которые необходимы для сборки определенного товара. Склады бывают разных размеров и имеет разные функциональности в нашей же проблематике достаточно рассмотреть лишь размерность склада. Существуют несколько видов классов [9]: Класс А (Малый склад), Класс В (Средний) и Класс С (Большой). Каждый имеет свою площадь, но основное их различие в высоте зданий и стеллажей, количество входов и в принципах доставки товара. Для проекта будут интересны только огромные склады, так как решение не имеет смысла на маленьких складах по причине того, что в тех отсутствует система WMS, так как с этой задачей может справиться и живой человек. Задача маршрутизации сборщиков заказов на складе является наиболее трудозатратной подзадачей процесса комплектования заказов. Перемещение сборщиков по складу во время ее выполнения достигает до 50% от всех временных затрат. Среди складских функций можно выделить комплектование заказов (order picking) как наиболее важную. Комплектование заказов - это процесс поиска и извлечения товаров из ячеек хранения с целью удовлетворения потребительского заказа. Подзадачами процесса комплектования заказов являются:

- проектирование планировки (layout design) и размеров (dimensioning) складской системы;

- распределение товаров по ячейкам хранения (storage assignment);
- распределение по партиям (batching) и зонирование (zoning);
- накопление заказов/сортировка (accumulation/sortation);
- маршрутизация (routing) сборщиков заказов.

На рис. 1 изображена логистика склада.

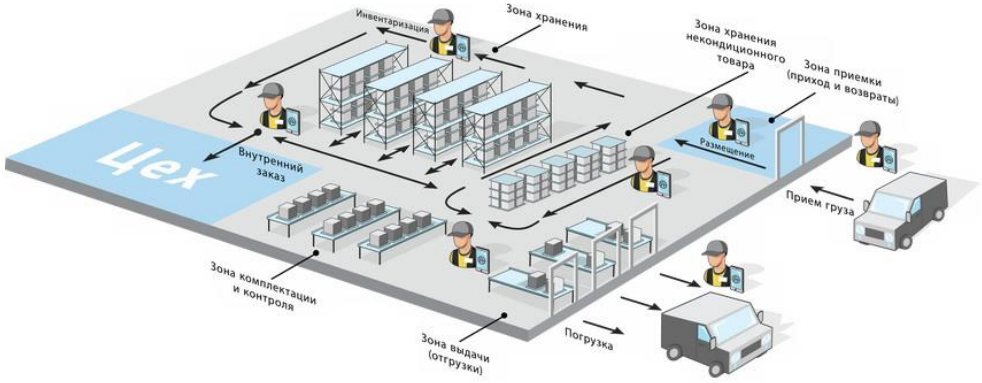


Рис. 1. Логистика склада

### 1.2. Постановка задачи

Из рис. 1 можно сказать, что задача маршрутизации сборщиков заказов (order-pickers routing problem) является частным случаем задачи коммивояжера (traveling salesman problem) [2]. Задача коммивояжера получила свое название благодаря следующей ситуации. Коммивояжеру требуется посетить определенное число городов только один раз и вернуться обратно в родной город на место старта. Он знает расстояния между каждой парой городов и хочет определить такой порядок их посещения, чтобы общее пройденное расстояние было минимальным.

На рис. 2 представлено типовое задание для сборщика товара, склад имеет 2 прохода: дальний и ближний. Имеет 6 пролетов среди стеллажей. На каждом стеллаже максимальное количество ячеек составляет 14 единиц. Название проходов зависит от расположения базы, в данном случае база находится в нижней части склада между 3 и 5 пролетом. На задании ячейки, закрашенные черным, являются

вершинами (точками назначения), которые сборщик обязан посетить для выполнения сборки товара. И перед сборщиком стоит цель посетить все помеченные вершины за минимальное количество времени. Перед проектом стоит задача, преобразовать данный рисунок в граф.

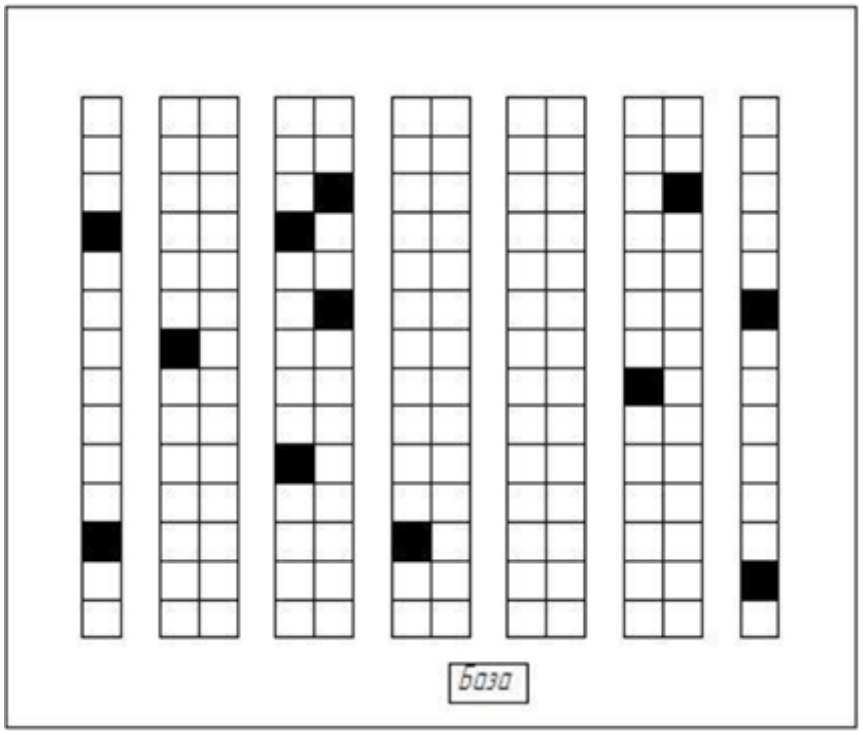


Рис. 2. Планировка склада с типовым заданием на сборку

Дано I={1,...,n} – множество пунктов назначения, матрица ($C_{ij}$) – попарные расстояния между точками, $1 \leq i, j \leq n$. Необходимо найти минимальный контур длины, то есть цикл, проходящий через каждую вершину ровно один раз и имеющий минимальный вес.

Определим граф G как совокупность вершин $v_i$, представляющих собой местоположение каждого из i=1,2,3..,m элементов заказа (т.е. расположение сборочных ячеек), и вершин $a_j$ и $b_j$ , являющих собой начало и конец каждого из j=1,2,3..,n проходов. Все смежные вершины графа соединяем дугами, длины которых соответствуют реальному расстоянию между точками склада, представленными смежными вершинами графа. Граф, описывающий конфигурацию склада, заданную на рис. 2, представлен на рис. 3. Для наглядности, между совпадающими

вершинами $v_0$ и $b_4$ добавлена дуга нулевой длины. Маршрут сборщика заказов - это цикл в G, включающий в себя все вершины $v_i$,i=1,2,3..,n хотя бы один раз. Задачей маршрутизации сборщика заказов является нахождение минимального такого цикла. На рис. 3 Представлен преобразованное в граф типовое задание.

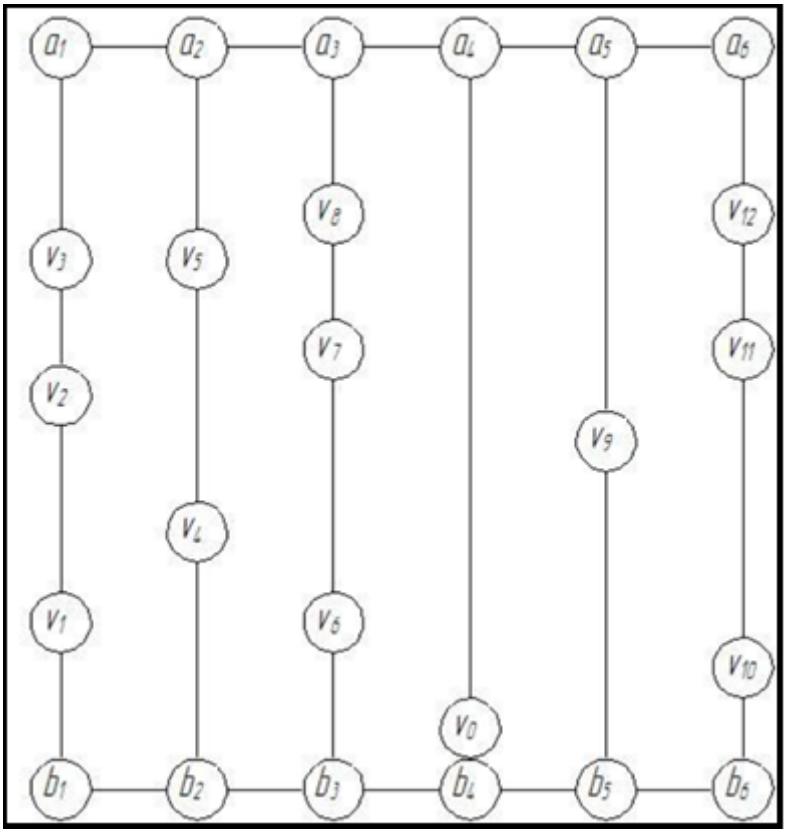


Рис. 3. Граф, описывающий конфигурацию склада, представленную на рис. 2

### 1.3. Анализ аналогичных проектов

Существуют способы оптимизации с помощью доступных инструментов WMS, ниже описаны некоторые из них по отбору заказа.

**Индивидуальный отбор [11]**

При индивидуальном отборе комплектовщик ответственен за сборку всех позиций заказа. С детальным списком необходимого для сборки товара работник перемещается по складу и полностью собирает заказ. Этот подход будет эффективен только для заказов, состоящих из небольшого для отбора списка товаров. Заказы обрабатываются по

очереди, точность сборки достаточно велика, необходимость обработки товаров минимальна, отобранный товар сразу поступает с места хранения в зону отгрузки. Недостатком этого метода является низкая эффективность и точность сборки для заказов, состоящих из большого количества позиций. При увеличении наименований в заказе количество ошибок возрастает.

**Комплексный отбор [11]**

Эффективность комплексного отбора выше, чем индивидуального, так как в этом случае одновременно группой комплектовщиков собирается несколько заказов. ИТ-система формирует для сборщика комплектовочный лист, где указывается наиболее оптимальный маршрут для одновременного отбора нескольких заказов. Если нужно какие-либо заказы собрать в первую очередь, то WMS-система, поддерживающая функцию комплексного отбора, укажет приоритет сборки определенных позиций.

**Зоновая комплектации [11]**

При зоновой комплектации склад делится на несколько зон, каждая из которых закреплена за отдельным сборщиком. Во время комплектации заказы могут передаваться из одной зоны в другую в коробках, тележках, паллетах, конвейерах и т.п. Если заказ формируется из некоторого количества товарных позиций разных зон, то для каждой зоны формируется лист комплектации, который назначается для отбора закрепленному за зоной сборщику. Разделение сборщиков по зонам и их знания о месторасположении товара в сочетании с возможностью одновременной сборки нескольких заказов позволяют значительно увеличить скорость и эффективность процесса комплектации. Зоновую комплектацию рекомендуется применять на складах и распределительных центрах, где обрабатывается большое количество товарных позиций и заказов. Минусы данной комплектации заключается

что если для определенной зоны нет задания, то персонал будет простаивать.

**Волновая комплектация [11]**

При волновом способе комплектации заказы группируются и передаются сборщикам в планируемое время, зависящее от длительности сборки заказов и специфики работы склада. Волновая комплектация наиболее эффективна при управлении складом WMS-системой. WMS-система рассчитывает и запускает последовательные волны, зависящие от специфических критериев их формирования. Одним из преимуществ волновой сборки является возможность сокращения рабочей нагрузки на каждого сборщика, что позволяет ему сосредоточится на специфических задачах (операциях) комплектации каждой волны.

Существуют и другие способы комплектации, но они не рассматриваются по причине недостаточной эффективности на больших складах.

**Выводы**

В этом разделе была поставлена задача, была описана актуальность и был анализ существующих методов. Подводя итоги можно сказать, что маршрутизация самый трудоемкий процесс и система WMS не может полноценно справляться с этой задачей, по этой причине необходимо дополнить функционал системы с помощью дополнительного модуля, а именно приложения с клиент-серверной архитектурой, серверная часть которого будет заниматься маршрутизацией, в то время как клиент будет отправлять задачи на сервер и выводить результат выполненных задач.

# 2. ПРОЕКТИРОВАНИЕ

## 2.1. Функциональные требования

Требования, относящиеся к тому, какие функции должны быть в приложении:

- приложение должно иметь клиент-серверную архитектуру;
- пользователь должен подключаться к серверу через IP-адрес и порт;
- пользователь должен создавать типовое задание через отдельное окно;
- пользователь должен отправлять задание на сервер в отдельном окне;
- приложение обрабатывать задание и строить оптимальный путь на сервере.

## 2.2. Нефункциональные требования

К таким требованиям относятся:

- пользователь должен вводить данные в форме Пролет, стеллаж, ячейка;
- пользователь должен видеть сообщение о том, что клиент подключился к серверу;
- когда сервер обработал данные он должен вернуть пользователю только последовательность посещения вершин;
- приложение должно поддерживать соединение с сервером до тех пора сам клиент не разорвет соединение либо не завершит выполнение приложения.

Требования, изложенные в пунктах 2.1 и 2.2, позволяют создать приложение, в котором необходимо лишь знать местоположения товара, что исключает возможность утечки информации об количестве и наименовании номенклатуры [18]. Для подключения к серверу

необходимо устройство с поддержкой возможности беспроводного интернета.

### 2.3. Архитектура системы

#### 2.3.1. Общий вид системы

На рис. 4 представлена диаграмма классов показывающее взаимодействие классов между собой.

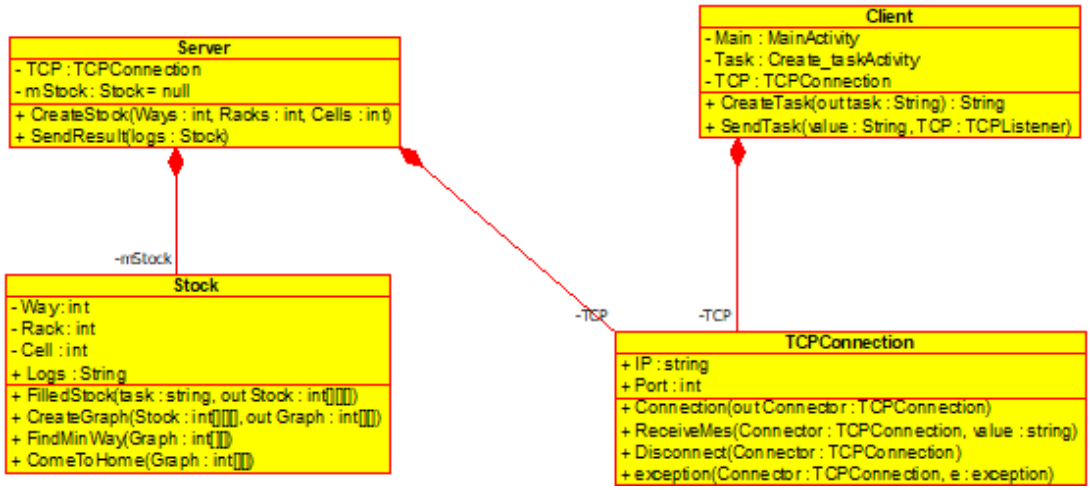


Рис. 4. Диаграмма классов

Система состоит из 3 компонентов:

- сервер, который располагается на персональном компьютере;
- клиент базируется на мобильных устройствах с операционной системой Android;
- класс **TCPConnection** - это класс, который соединяет клиента с сервером.

Но также на стороне сервера присутствует ещё один класс, который реализуют имитацию склада с некоторыми заполненными ячейками, полученными из задания, отправленного от клиента. Также на стороне этого класса реализуется создание графа и поиска минимального пути на графе.

#### 2.3.2. Подключение

Для подключения необходимо ввести IP-адрес сервера и его порт, используется версия IPv4, так как данный вид протокола, наиболее

используемый по сравнению с более современным IPv6. Для сервера используется проводной доступ к сети, клиент же имеет только беспроводной способ подключения к сети. Авторизация была исключена, так как в этом нет необходимости, пользователь не получает никакую информацию об номенклатуре и ее местоположению.

В случае успешного подключение пользователь получает информацию о том, что он может создавать запросы к серверу.

В случае возникновения ошибки при подключении, пользователю сообщают об какой ошибке идет речь, они классифицируются на:

- недоступность сервера;
- ошибка тайм-аута;
- неизвестная ошибка.

### 2.3.3. Работа с приложением. Клиентская часть.

После того как пользователь успешно подключился, с помощью отдельного окна создает типовой лист задания. В окне присутствуют следующие параметры:

- проход, в котором находится необходимый стеллаж;
- номер стеллажа в проходе;
- номер необходимой ячейки.

После создания листа, пользователь может отправить его на сервер. После обработки данных клиент получает последовательность посещения точек назначения.

## 2.4. Используемые алгоритмы

### 2.4.1. Построение графа

Рассматривается задача, существует склад, в котором находятся стеллажи. Внутри стеллажей находятся ячейки, некоторые из них, которые сборщик обязан посетить. После посещения всех точек назначения, сборщик товара должен вернуться на базу. Суть задачи

состоит в том, чтобы преобразовать топологию склада в G = (V, J) где G – полный граф.

Идея алгоритма **CreateGraph** состоит в том, что ответственный за сбор товара может попасть в любую вершину из текущей, это правило действует на все вершины. Для построения ребер к вершинам должно учитываться, количество проходов, стеллажей в ряду и номер ячейки, а также необходимо знать расстояние между стеллажами и ячейками. Way- содержит в себе количество проходов а Way.Distance – это расстояние между проходами, Rack — количество стеллажей между двумя проходами. Cell – количество ячеек в одном стеллаже. Для каждого шага Edge, определяется чему будет равен вес ребра для текущего значения Vertex. Все это нужно чтобы правильно определить вес ребра к нужной вершине по отношению к текущей. Ниже на рис. 5 представлен псевдокод алгоритма.

```
//Way, Rack, Cell - это классы которые имеют внутри себя два параметра:
Count - количество на складе, Distance - расстояние между объектами
CreateGraph(Input: int CountVertex, int CurrentVertex, int Graph[][][];
Output: G(V, E);)
int Edge = 0;// Это ребра вершин
int Dis=0; // расстояние
      for(int i=0; i<Rack.Count-1);i++)//
      Dis=0;
      Dis+=(i+1)*Rack.Distance;
      for(itn j=0;j<((Way.Count-1)*2);j++)
            Если остаток от деления j равен нулю
            Тогда Dis+=j*Way.Distance;
            иначе Dis+=(j-1)*Way.Distance;
            конецЕсли;
            for(h=0;h<Cell.Count;h++)
                  Если значение Graph[i][j][h] не пустое
                  Тогда G[CurrentVertex][Edge]=Dis+(h*Cell.Distance);
                  Edge++;
                  Если Edge>=CountVertex
                  тогда закончить Цикл
                  конецЕсли;
```

Рис. 5. Псевдокод алгоритма CreateGraph

**CreateGraph** вычисляет вес ребер к вершинам только для базы, но данный псевдокод является основой вычисления для всех вершин графа.

### 2.4.2. Поиск оптимального пути

Так как жадный алгоритм не оптимален для решения задач коммивояжера, а Алгоритм Краскала эффективный алгоритм построения минимального остовного дерева для взвешенного связного графа, то можно использовать функцию минимального пути и устранения цикличности для оптимизации жадного алгоритма.

На рис. 6 изображен псевдокод алгоритма поиска оптимального пути. Суть данного алгоритма заключается в том, что исключается возможность посещения помеченной вершины. На вход поступает граф с вершинами и ребрами. После того как вершина помечена ее нельзя будет посетить и тем самым исключается цикличность внутри графа. Сложность такого алгоритма O(n) и не меняется как в жадном алгоритме на $O(n^2)$. На выходе будет строка, которая содержит в себе последовательность посещения вершин.

```
Travel(Input: G(V,E); Output: String Log;)
for i=0 to E do
      Если вес ребра E наименьший из текущей вершины тогда
      Если вершина в которую планируется перейти не была посещенной
      тогда
                  Mark(V) // помечаем текущую вершину
                  Visited(E) // Переходим в новую
```

Рис. 6. Псевдокод алгоритма Travel

### Выводы

В данной главе была спроектирована структура проекта, было представлено взаимодействие с системой с точки зрения пользователя. Это взаимодействие отображено на Use-Case диаграмме. Отдельно были описаны структуры сервера и клиента с помощью диаграммы классов, после которых было описание методов и их назначение. Подводя итоги, можно сказать, что результатом данной главы является система, с помощью которой и будет реализоваться проект.

# 3. РЕАЛИЗАЦИЯ

## 3.1. Средства для реализации

Для реализации клиент-серверной архитектуры был использован язык Java. Java - сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретенной компанией Oracle). В настоящее время проект принадлежит OpenSource и распространяется по лицензии GPL. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины.

Для разработки сервера использовалась среда разработки от компании JetBrains IntelliJ IDEA. Данная IDE* является одной из мощнейших редакторов исходного кода. Здесь есть не только классические подсветки синтаксиса, табы, фолдинг. Переходы к определению, автокомплит с учетом нужного в контексте типа, автоматическое добавление нужных импортов, а также оптимизация импортов с удалением ненужных, встроенное отображение документации и типов, подсказка по аргументам, переименование аргументов, переменных и модулей буквально за секунду, быстрое указание возвращаемого типа у метода. Также код форматируется автоматически, что позволяет не следить за стилистикой все эту работу выполняет IDEA. Все эти полезные черты повышает производительность.

Для разработки клиента для мобильной платформы использовалась среда разработки программного обеспечения Android Studio. В 2017 году официальным языком Android Studio (в дополнение к первоначально заявленным Java и C++) был объявлен *Kotlin*, который позиционируется как более простой, благодаря чему ускоряется процесс компиляции приложений. В Android Studio создание приложения происходит достаточно просто и понятно, в том числе по той причине,

что эта среда разработки предлагает достаточное количество готовых шаблонов и макетов. С точки зрения функционала к преимуществам можно отнести:

- **Instant Run.** Это функция, которой на протяжении всего времени развития Android Studio было уделено довольно много внимания, благодаря чему к выходу версии 3.0 она уже работала в полноценном режиме. Instant Run включена для того, чтобы разработчик приложений для Android после изменения кода мог сразу оценить, как это изменение повлияет на результат - и без дополнительных временных затрат на перекомпиляцию;

- **Использование Cloud Test Lab**. Естественно, что любое приложение проходит этап тестирования, и Google Test Lab дает разработчику возможность проверить готовый продукт на самых разных устройствах, которые располагаются в дата-центре компании.

### 3.2. Реализация модулей Системы

#### 3.2.1. Структура сервера

На рис. 7 изображена диаграмма классов структуры сервера. На диаграмме показаны связанные последовательно между собой классы.

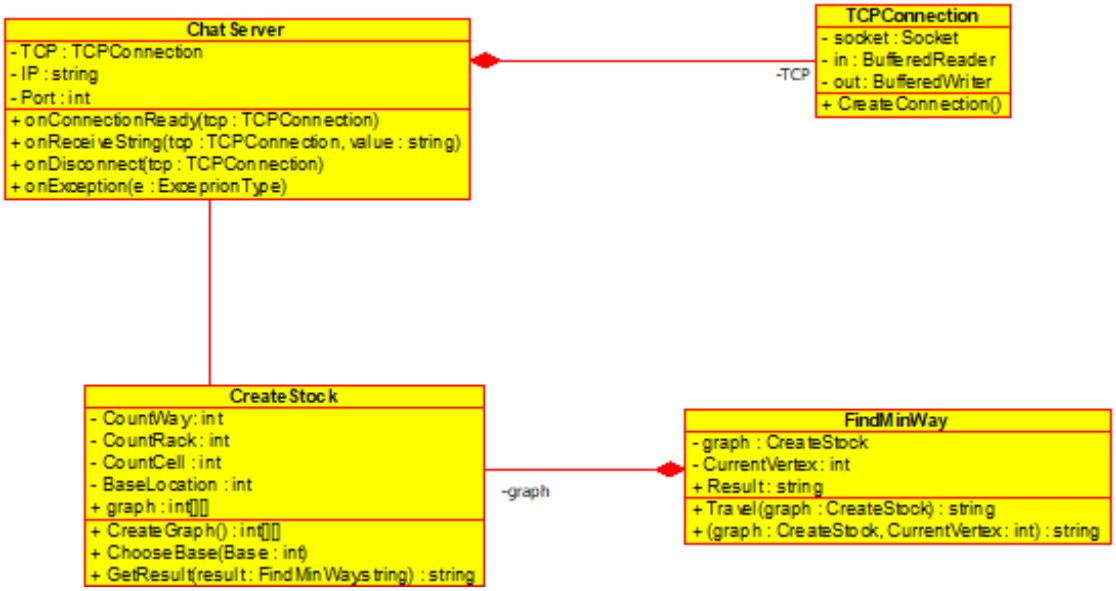

Рис. 7. Диаграмма классов для сервера

**TCPConnection** – этот класс создает соединение между клиентами и сервером, с помощью данного класса и реализуется получения задания от клиента и отсылка результата обратно клиенту.

**ChatServer** – класс хранит в себе IP-адрес и порт сервера, вызывает методы для взаимодействия с клиентами через класс **TCPConnection**. На один склад поступает множество заданий, и через классы **CreateStock** и **FindMinWay** создает результат для клиента.

**CreateStock** - этот класс создает граф для вычисления оптимального пути до точек назначения из типового задания от клиента. Переменные с приставкой Count, не изменяются пользователем и являются постоянным для него значением. Way, Rack, Cell - это переменные которые отвечают за количество пролётов, стеллажей в ряду и количество ячеек в стеллажах. BaseLocation, которая также не изменяется пользователем, она влияет на то с какой точки начнется вычисление веса, существует 4 варианта расположения баз, они отображены на рис. 8.

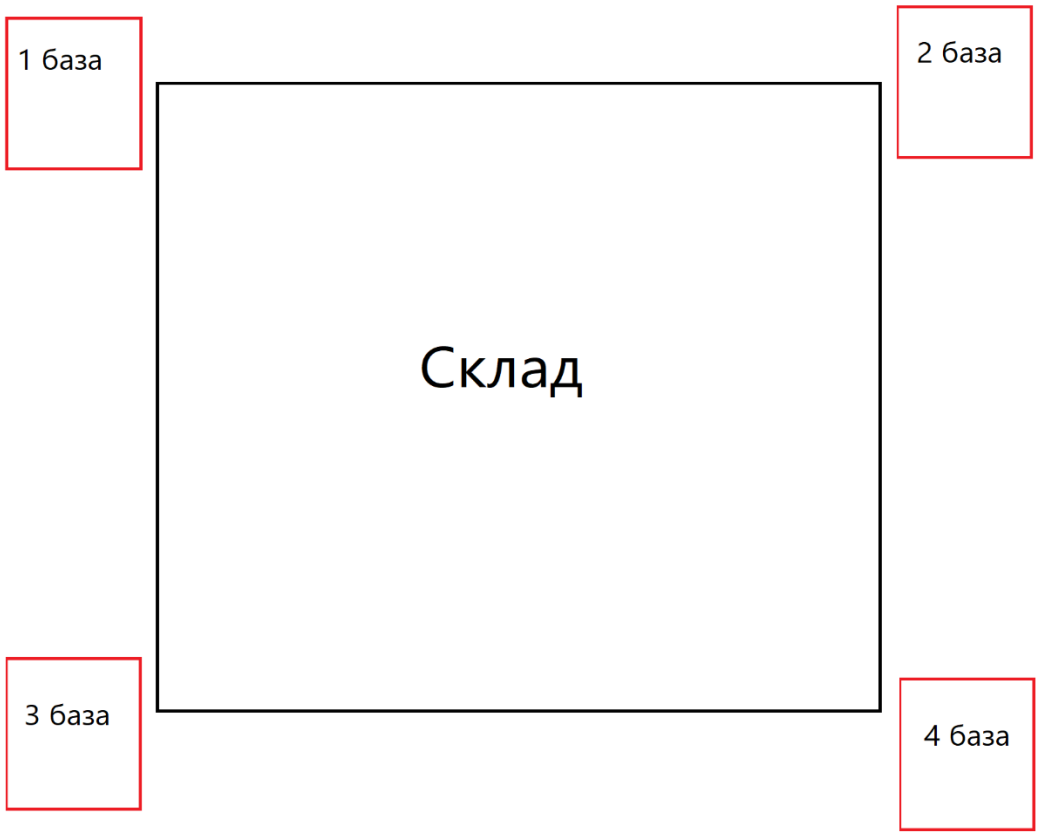


Рис. 8. Схема расположения баз на складе

Расчет веса ребер от любой базы до всех вершин выполняется в методе **ChooseBase**. Класс получает лист с заданием в формате строковой переменной, которая затем формируется в массив целых чисел, данные для каждой вершины хранятся в формате Пролет - Номер стеллажа - номер ячейки, после создается многомерный массив, который является проекцией склада. Слиянием формируем заполненные ячейки в многомерном массиве, как это выглядит можно наблюдать на листинге 1.

Листинг 1. Алгоритм проецирования листа на склад

```
//product это массив который будет хранить в себе координаты расположения
пунктов назначения
for (int i = 0; i < product.length; i = i + 3) {
            int a, b, c = 0;
            a = product[i];//Хранит номер пролета
            b = product[i + 1]; //Хранит номер стеллажа
            c = product[i + 2]; //Номер Ячейки
            Cell[LEN]=c+1; //Записывается номер ячейки для одной вершины
            LEN++; //Счетчик вершин
            Product[a][b][c] = 2;
}
```

Затем строятся вес ребер от базы до каждой вершины, на листинге 2 рассмотрен алгоритм построения для первой базы (Расположение баз изображено на рис. 8).

Листинг 2. Вычисление веса ребер от базы ко всем вершинам

```
int sum = 0; //Хранит вес
for (int i = 0; i < (CountRackWay - 1); i++) {
    for (int j = 0; j < ((CountWay - 1) * 2); j++) {
        sum = (i + 1) * RackDistance;//RackDistance
        if (j > 1) {
            int f = j % 2; //Строки находят расстояние до нужного
стеллажа если база находится не в превых двух проходах
            if (f == 0) sum += (CountCell * (j / 2));
            else sum += (CountCell * ((j / 2)));
        }
        for (int h = 0; h < CountCell; h++) {
            if (Product[i][j][h] != 0) {    //Если ячейка не пуста то
записываем вес
                sum += 1;
                Product[i][j][h] = sum;
                len += 1;
            }
            else {
                sum += 1;
            }
        }
    }
}
return Product;
```

Для оптимизации алгоритма, нужный стеллаж ищется вдоль пролетов. Затем проверяется насколько далеко находится стеллаж от начальных двух проходов, если проход находится не между ними, то к конечному результату добавляется расстояние между проходами.

После выполнения этого алгоритма многомерный массив преобразуется в двумерный, а номера проходов, стеллажей и ячеек заносятся в отдельный одномерный массив, сделано это с целью повышения читаемости кода.

Для дальнейшего создания полного графа, индуцированного заданными вершинами, требуется сравнение, которое выполняется в отдельной функции с наименованием **CreateGraph**:

- на вход поступает одномерные массивы со значением пролета, стеллажа и ячейки, и двумерный массив, в который будут записываться значения веса ребер;

- на выходе будет заполнена матрица, которая будет входными данными для класса **FindMinWay**.

Суть алгоритма заключается в следующем, он разделяется на две ветви. Стеллажи находятся на одном и том же пролете, и стеллажи находятся в разных пролетах. Выполнение первого варианта сводится к подсчету прямой от точки А к точке В.

Но для расчета расстояния при разных пролетах, требуется узнать следующие параметры, текущая ячейка находится выше необходимой относительно базы, сколько ячеек необходимо преодолеть для того, чтобы выйти к ближайшему проходу в сторону нужной ячейки, и сколько нужно пройти стеллажей для того, чтобы свести все вычисление к первому варианту.

Результатом является полный граф, в виде матрицы, граф отправляется на обработку в следующий класс **FindMinWay**.

**FindMinWay** – этот класс ищет оптимальный путь в графе полученный из класса **CreateGraph** и в своем распоряжении имеет две функции: **Travel** и **CometoHome**. Результатами который является строковая переменная хранящая в себе последовательность посещения вершин.

**Travel** – функция, в которой на вход подается двумерный массив с целочисленными значениями, и переменная с целым числом «n», она отвечает за количество вершин в графе.  На листинге 3 изображена инициализации всех переменных, значения которых сравнивается в каждой итерации цикла по поиску минимального пути.

Листинг 3. Переменные для поиска оптимального пути

```
int max = 0;//Максимальное значение веса
int min = 99; // Минимальное значение веса
boolean end = false;//Переменная отвечает за остановку цикла
boolean Traveled[];//Переменная которая сверяется с тем что была ли
посещена вершина или нет
```

Каждая итерация работает в формате, найден минимальный локальный элемент в строке матрицы, помечаем, и после проверки

остальных элементов строки, происходит следующие действия. Текущая вершина отмечается в переменной **Traveled**, и затем происходит переход в новую вершину. Внутри алгоритма проверяются **Traveled** и все истинные значения отбрасываются, хранящиеся внутри матрицы, тем самым исключается возможность зацикливание алгоритма.

**CometoHome** – самый короткий метод в функционале сервера, отвечает за возвращение на базе, для выполнение алгоритма требуется двумерный массив целых чисел, вершина на которой завершилось выполнение функции **Travel**. На листинге 4 изображено выполнение функции.

Листинг 4. Функция по возвращению на базу из текущей вершины

```
public int CometoHome(int graph[][], int sum) {
    if(currentCity!=0) {    //CurrentCity - содержит информацию о том с
какой вершины будет возращению на базу
        Logs += Integer.toString(currentCity + 1) + "->" + Inte-
ger.toString(1);
        sum += graph[currentCity][0];
    }
    return sum;
}
```

**CurrentCity** – переменная, которая хранит в себе номер вершины на которой остановился цикл. Существует ситуация, когда выполнение закончилось на базе, в таком случае, ничего изменяться не будет, и значение останется тем же самым.

### 3.2.2. Структура клиента

На рис. 9 изображена диаграмма классов относящиеся к клиенту.

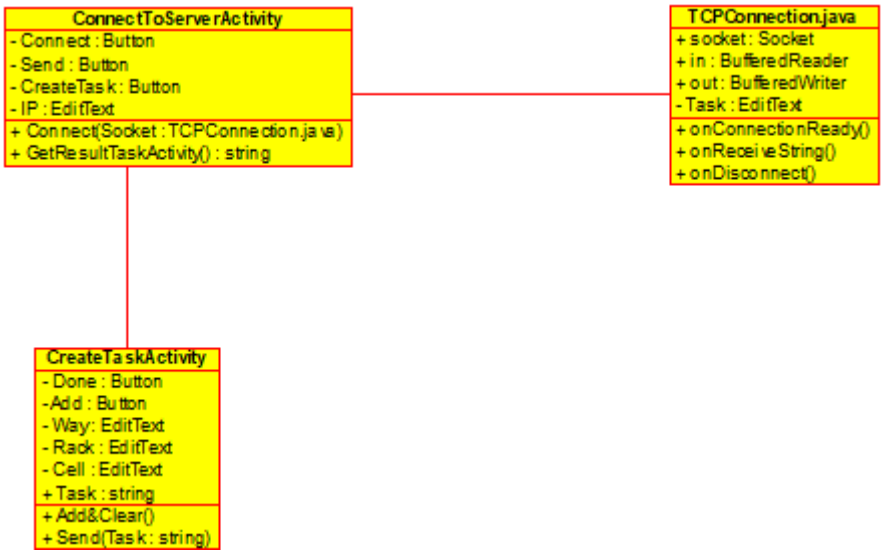

Рис. 9. Диаграмма классов клиента

Исключая обязательные файлы для Android – приложения, на рис. 9 изображены диаграмма классов клиента. Клиент состоит из трех компонентов, класс **TCPConnection**, который отвечает за создание соединения, отправку задач. **ConnectToServerActivity**, интерфейс для взаимодействия с сервером и с окном который создает лист с заданием в **CreateTaskActivity**.

**TCPConnection** - этот класс создает соединение между клиентом и сервером и с помощью данного класса реализуется отправка задания на сервер и получение результата из него.

**CreateTaskActivity** – Класс который реализует функцию создания листа задания. Внутри себя содержит несколько функций.

На листинге 5 изображена функция “AddOrder”, функция сохраняет внесенные данные в строку и очищает поле, где необходимо вводить координаты расположения ячейки на складе. Под координатами имеется ввиду пролёт, номер стеллажа и номер ячейки. Такие данные вводятся для каждой точки, которую сборщик должен посетить.

Листинг 5. Функция создания новой точки назначения

```
Private void addOrder(){
      strOrder+=Way.getText()+” “+Rack.getText()+” “+Cell.getText()+” “;
Way.setText(null); Rack.setText(null); Cell.setText(null);
}
```

Данная функция вызывается, когда мы хотим добавить ещё один пункт назначения и, если пользователь заполнил поля и нажал кнопку «Готово». На листинге 6 изображен алгоритм отправки готового листа в основную активность.

Листинг 6. Отправка значения в другую активность

```
Dones.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        addOrder();
        Intent intent = new Intent();
        RedacherStr(strOrder);
        intent.putExtra(ORDER,strOrder);
        setResult(RESULT_OK,intetn);
        finish();
    }
});
```

Мы записываем последние заказ в строку, затем инициализируем переход между активностями, в функции **RedacherStr**, мы изменяем содержание строки для будущей отправки на сервер. На рис. 10 представлен скриншот активности, где создается лист с заданием.


Рис. 10. Скриншот активности, где создается лист с заданием

**ConnectToServer** – как было сказано ранее, это активность является интерфейсом взаимодействия с сервером.

Внутри активности расположены элементы ввода и вывода в ввиде переменных. Их список вы можете наблюдать на листинге 7.

Листинг 7. Элементы вывода и ввода

```
private TextView mLog;
private EditText mInput
private EditText mIPChoose
private Button mSend
private Handler mHandler
private Button mConnect
```

**mLog –** Данная переменная хранит в себе результат всех операций текущего сеанса, то есть если пользователь освободит память приложения завершив его тем самым, то значения переменной будет равно Null.

**mInput –** Этот элемент является вводом листа заданий, через это поле заносится местоположение ячеек.

**mIpChoose** – Этот элемент является вводом для IP-адреса сервера.

**mSend** – Отвечает за отправку данных на сервер, а также реализует функцию перехода к **CreateTaskActivity**,

**mConnect-** элемент, который выполнчет алгоритм подключения к серверу для возможности дальнейшего взаимодействия с ним в виде отправка заданий и получения результат и вывод их **mLog.**

**mHandler** – Инструмент позволяющий изменять значение **mLog** в реальном времени.

На листинге 8 изображен алгоритм получения данных из активности **CreateTaskActivity**.

Листинг 8. Получение данных из активности создания заданий

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data){
    super.onActivityResult(requestCode,resultCode,data);
    if(requestCode == ORDER){
        if(resultCode == RESULT_OK){
            mInput.setText(data.getStringExtra(COActivity.ORDER));
        }
    }
}
```

Для получения данных из активности необходимо проверить тэг с помощью, которого данные получает только та активность, которая вызывает данный тэг. После идет проверка на успешное завершение активности, из которой мы хотим получить данные. И затем данные, которые мы получили из активности присваиваются элементу **mInput.**

Скриншот главного меню представлено на рис. 11.


Рис. 11. Скриншот главного меню

**Выводы**

С помощью спроектированной структуры из прошлой главы, была реализована система в составе которой находятся два приложения Сервер и клиент. Сервер был реализован для устройства под операционной системой Windows, клиент же был создан как мобильное приложение для устройств под ОС Android. В данной главе описывалась структура системы внутри прикладывались листинги кода для особо важных методов. Также внутри главы было описание используемых ПО и их преимущества использования в данном проекте. Результатом является полноценно реализованная система, но, чтобы узнать, насколько данное решение работоспособно требуется провести ряд тестирований.

# 4. ТЕСТИРОВАНИЕ

## 4.1. Тестирование сервера

Тестирование сервера содержит в себе тесты, которые направлены на поиск потери данных со стороны либо клиента, либо сервера, все проходит три теста, подключения, что будет если нет интернета и как сервер получает данные и возвращает обработанный результат. На табл. 1 представлен протокол тестирования сервера.

Табл. 1. Тестирование сервера

| № | Название | Описание | Ожидаемый результат | ОК? |
|---|---|---|---|---|
| 1 | Подключение | Клиент с помощью IP адреса подключается к серверу | У клиента в элементе для вывода появляется информация об успешном подключении | Да |
| 3 | Получение задания | (Дебаг Тестирование) после нажатие клавиши отправить со стороны клиента, сервер получает данные | На сервере отображаются данные которые отправлялись с клиента | Да |
| 4 | Отправка Результата | Сервер отправил данные, клиент получил и вывел их | На экране клиента появилась последовательность посещения точек в специализированном элементе вывода | Да |

## 4.2. Тестирование клиента

Тестирование клиента направлено на проверку функционала с точки зрения локального пользователя, пользователь взаимодействует с приложением, а оно должно обрабатывать эти действия. Внутри тестов ведется тестирования перехода между активностями, передачи данных между ними, на табл. 2 представлен протокол тестирования клиента.

Табл. 2. Тестирование Клиента

| № | Название | Описание | Ожидаемый Результат | Пройден? |
|---|---|---|---|---|
| 1 | Поле ввода IP-адреса и обработка нажатия клавиши "Connect" | Вручную вводим адрес сервера, и нажимаем кнопку для подключения | Произошло подключение и был вывод о том, что подключение успешно | Да |
| 2 | Обработка клавиши перехода на новую активность по созданию задания | Кнопка инициализирует переход с одной активности в другую | Изменилась Активность | Да |
| 3 | Заполнение задания и передача значений в главную активность, где отправляется задачи на сервер с помощью кнопки «Готово» | Пользователь заполнил данные после нажатия на клавишу активность передает значения в другую | После завершения активности на элементе текущей активности появились данные из прошлой | Да |

### 4.3. Тестирование верстки на устройствах

Тестирование проводилось на следующих устройствах: Dexp ES2'5, Fly Numbus, BQ Magic, Redmi 5 Plus, Samsung Galaxy A20, Samsung J2 Prime. Сильных изменений со стороны верстки не было, версии ОС граничили от 4.1 до 8.1, рис. 12 отвечает за главное меню устройства на базе ОС Android с версией 6.1, и экраном с разрешением 720x1080, а рис. 13 с версией 4.1. с разрешением 1080x1440.


Рис. 12. Главное меню на реальном устройстве с версией ОС 6.1

Для рис. 12 использовался экран типа «420dpi», для рис. 13 же использовался экран типа «xhdpi».


Рис. 13. Создание заявки на устройстве версией 7.1

### 4.4. Тестирование затрат ресурсов на устройствах

Данный этап тестирования направлен на выявление утечки памяти, а также наблюдения, как приложение будут влиять при длительной работе на устройство, тестирование проводилось на трех устройствах с версиями ОС Android с 4.1 до 6.0. На табл. 3 тестирование на Fly Numbus. Данное устройство имеет версию ОС Android 6.0. Данная система является весьма экономной по отношению к оперативной памяти, а это значит, что большинство ресурсов будет поступать в приложения, что исключает возможность зависание приложения.

Табл. 3. Тестирования клиента на устройстве Fly Numbus

| № теста | Модель | Описание теста | Результат |
|---|---|---|---|
| 1 | Fly FS454 | С помощью утилиты IDE мы можем посмотреть сколько тратится оперативной памяти на | От 13 MB до 15,8 MB |

| | | | |
|---|---|---|---|
| | | взаимодействие с приложением | |
| 2 | Fly FS454 | Ведется просмотр как сильно нагружается процессор при работе с приложением | Максимальная использования Процессора достигала до 15% |
| 3 | Fly FS454 | Наблюдения за затрачиваемым трафиком | На каждый запрос тратится 1 КБ трафика |

Тестирование на мобильном устройстве BQ Magic, представлено на табл. 4 данное устройство обладает большими техническими характеристиками, чем Fly Numbus, но операционная система имеет версию 5.1. Из минусов данной операционной системы является быстродействие, для комфортной работы с системой требуется 3 ГБ памяти, устройство BQ Magic, оборудовано лишь 2 ГБ, и данное значение является пороговым для работоспособности системы.

Табл. 4. Тестирование клиента на устройстве BQ Magic.

| Номер теста | Модель | Описание теста | Результат |
|---|---|---|---|
| 1 | BQ BQS-5070 | С помощью утилиты IDE мы можем посмотреть сколько тратится оперативной памяти на взаимодействие | От 27MB до 30,5 MB |

| | | | |
|---|---|---|---|
| | | с приложением | |
| 2 | BQ BQS-5070 | Ведется просмотр как сильно нагружается процессор при работе с приложением | Максимальное использования ЦП достигала до 6% |
| 3 | BQ BQS-5070 | Ведется наблюдение того как много используется интернет трафика | На каждый запрос тратится 1 КБ трафика |

На табл. 5 представлено тестирование на мобильном устройстве DEXP ES2 5, данное устройство базируется на ОС системе Android версией 4.1. И по сравнению с прошлыми устройствами, данная модель имеет наименьшие технические характеристики. По этой причине данное устройство показало наибольшую нагрузку на ЦП.

Табл. 5. Тестирование клиента на устройстве DEXP ES2 5

| Номер | Модель | Описание | Результат |
|---|---|---|---|
| 1 | DEXP DEXP Ix-ion ES 2 5 | С помощью утилиты IDE мы можем | От 10 MB до 20,3 MB |

| | | | |
|---|---|---|---|
| | | посмотреть сколько тратится оперативной памяти на взаимодействие с приложением | |
| 2 | DEXP DEXP Ix-ion ES 2 5 | Ведется просмотр как сильно нагружается процессор при работе с приложением | Максимальная нагрузка на ЦП достигала до 22% |
| 3 | DEXP DEXP Ix-ion ES 2 5 | Ведется наблюдение того, как много используется интернет трафика | На каждый запрос тратится 1 КБ трафика |

Подводя итоги данного этапа тестирований, можно сказать, что результат является приемлемым и не нагружает устройство до такой степени что во время эксплуатации устройство может прийти в негодность.

### 4.5. Тестирование времени выполнения алгоритма

Было тестирование времени выполнения алгоритма, которое было реализовано с помощью внутренних инструментов интегрированной среды разработки. На табл. 6 протокол этого тестирования.

Табл. 6. Протокол тестирования времени исполнения алгоритма.

| № Теста | Количество точек назначения | Время выполнения в мс |
|---|---|---|

| | | |
|---|---|---|
| 1 | 10 | 0-15 |
| 2 | 100 | 10-25 |
| 3 | 1000 | 30-100 |
| 4 | 10000 | 224-398 |
| 5 | 100000 | 594-1127 |

**Выводы**

В составе этой главы присутствует несколько видов функциональных тестирований, тестировалась верстка для разных версий операционной системы, для разных экранов. Проводилось тестирования функционала подключения на разных моделях устройств и качеством соединения с сервером. С помощь встроенных инструментов IDE, было зафиксировано сколько тратится оперативной памяти и как сильно нагружается процессор устройства. И для проверки как долго будет работать алгоритм с огромным количеством данных только при количестве более 100000 точек назначения алгоритм в ¼ случаях работал в течение секунды. Подводя итоги, можно сказать, что реализованная система с точки зрения функционала работает приемлемо и выполняет все поставленные задачи.

## ЗАКЛЮЧЕНИЕ

Целью данной работы заключалась в разработке приложения, которое вычисляла оптимальный путь на складе для сборщика товара. В ходе разбора аналогов и предметной области было принято решение что для реализации данной задачи требуется клиент-серверная архитектура так как обычное бюджетное устройство не сможет справиться с такой нагрузкой без чрезмерного нагревания устройства и траты заряда батареи. В главе проектирование было выполнено планирование о том, что должно на себя брать мобильное устройство и что требуется для аутентификации на сервере, и была спроектирована структура приложения. В части реализации было описаны методы, которые входят в состав клиента и сервера и было создана система из результатов главы проектирования. Чтобы убедиться, что данная система работает и выполняет корректно необходимый функционал было проведено тестирования, содержания этих тестов были описаны в главе тестирования, было протестирован дизайн, время работы, а также качество точности вычислений. Подводя итоги, можно сказать, что данный проект является прототипом направленный на уменьшении габаритов устройств, которыми пользуются сборщики товара для выполнения заказа. Ведь мобильное устройство есть даже у ребенка, а это значит, что организации не требуется производить дополнительную закупку оборудования для новых сотрудников. Подводя итоги, существующий продукт является прототипом, а это значит, что добавление функционала и доработка дизайна является возможными направлениями для улучшения приложения, а также дальнейших исследований по оптимизации поиска минимального пути для сборщика товара на складе.

**ЛИТЕРАТУРА**

1. Developer guide for Android. [Электронный ресурс] URL: https://developer.android.com/guide (дата обращения: 12.02.2019).

2. Goransson A. Efficient Android Threading: Asynchronous Processing Techniques for Android Applications. - O'Reilly Media. - 2014. - 280 p.

3. Lawler E.L., Lenstra J.K., Kan A.R., Shmoys D.B. The Travelling Salesman Problem: A Guided Tour of Combinational Optimization. - Autumn. - 1988. - 463 p.

4. Leiva A. Kotlin for Android Developers: Learn Kotlin the easy way while developing an Android App. - CreateSpace Independent Publishing Platform. - 2016. - 240 p.

5. Phillips B., Stewart C., Marsicano K. Android Programming: The Big Nerd Ranch Guide. - Big Nerd Ranch Guides. - 2017. - 998 p.

6. Nurkiewicz T., Christensen B. Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications. - O'Reilly Media. - 2016. - 372 p.

7. Алгоритм Краскала. [Электронный ресурс] URL: http://www.math.spbu.ru/user/jvr/DA_html/_lec_2_05.html (дата обращения: 19.02.2019).

8. Шилдт Г. «Java 8. Руководство для начинающих». - Вильямс. - 2015. – 720 с.

9. Дейтел П., Дейтел Х., Уолд А. Android для разработчиков. - Питер. - 2016. - 512 с.

10. Дискретная математика алгоритмы. [Электронный ресурс] URL: http://rain.ifmo.ru/cat/view.php/theory/algorithm-analysis/greedy-2004 (дата обращения: 12.02.2019).

11. Жадные Алгоритмы. [Электронный ресурс] URL: http://www.williamspublishing.com/PDF/5-8459-0857-4/part.pdf (дата обращения: 12.02.2019).

12. Задача Коммивояжера. [Электронный ресурс] URL: http://math.nsc.ru/~alekseeva/DP-MMF-2013/L4_mmf_2012_2013.pdf (дата обращения: 14.02.2019).

13. Как выбрать склад - виды складов и их классификация. [Электронный ресурс] URL: http://skladovoy.ru/kak-vybrat-sklad-vidy-skladov-i-ix-klassifikaciya.html (дата обращения: 08.10.2018).

14. Система управления складом WMS. [Электронный ресурс] URL: https://mossahar.ru/articles/sistema-upravleniya-skladom-wms (дата обращения: 10.10.2018).

15. Склады их определения, виды и функции. [Электронный ресурс] URL: https://studwood.ru/1778597/marketing/sklady_opredelenie_vidy_funktsii (дата обращения: 08.10.2018).

16. Складские операции: оптимизация процесса комплектации. [Электронный ресурс] URL: https://www.ant-tech.ru/fields/skladskie-operatsii-optimizatsiya-protsessa-komplektatsii (дата обращения: 18.10.2018).