Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«ПРОГРАММНОЕ СРЕДСТВО ДЛЯ СЛУЖБЫ ТЕХНИЧЕСКОЙ ПОДДЕРЖКИ АБОНЕНТОВ СЕТИ»

БГУИР КП 1-40 01 01 009  ПЗ

Студент: гр. 951005 Доведько Д.Ю.

Руководитель: асс. Болтак С.В.

Минск 2020

**СОДЕРЖАНИЕ**

Введение ........................................................................................................ 3

1 Анализ прототипов, литературных источников и формирование

требований к проектируемому программному средству........................ 5

1.1 Понятие технической поддержки....................................................... 5

1.2 Анализ существующих программных продуктов............................. 6

1.3 Формирование требований к разрабатываемому ПС....................... 11

2 Разработка структуры данных.................................................................... 12

2.1 Описание используемых структур данных........................................ 12

3 Разработка программного средства........................................................... 18

3.1 Схема программы.................................................................................. 18

3.2 Проектирование главного окна системы ........................................... 22

3.3 Проектирование окна списка клиентов............................................... 23

3.4 Проектирование окна списка поступивших заявок........................... 25

3.5 Проектирование окна статистики поступивших заявок.................... 25

4 Обоснование технических приемов программирования ......................... 27

5 Тестирование................................................................................................. 29

5.1 Результаты тестирования программы................................................. 29

5.2 Подтверждение корректности тестирования..................................... 33

6 Руководство пользователя программы...................................................... 42

Заключение ..................................................................................................... 44

Список использованных источников ........................................................... 45

Приложение. Исходный код программы ..................................................... 46

Ведомость......................................................................................................... 99

**ВВЕДЕНИЕ**

По мере появления современных инновационных технологий и ИТ сервисов уровень обслуживания клиентов постепенно повышается. Компании стараются максимально быстро и эффективно удовлетворять запросы потребителей. Когда у пользователей возникает какая-либо проблема, они могут отправить запрос, содержащий описание вопроса и просьбу о помощи в его решении. Благодаря этому служба технической поддержки пользователей (HelpDesk, Service Desk) сможет рассмотреть поступившую заявку и дать свой ответ.

В целом, техническая поддержка служит попыткам помочь конкретным пользователям решать возникающие конкретные проблемы с продуктом и его использованием.

Цель курсового проектирования - создание программного средства, позволяющее персоналу службы технической поддержки абонентов сети регистрировать и обрабатывать заявки клиентов с использованием принципов объектно-ориентированного программирования, принципов объективности, целенаправленности, системности научного исследования и динамических структур данных.

Пояснительная записка содержит несколько разделов:

* Первый раздел пояснительной записки содержит понятие технической поддержки, сравнительный анализ существующих программных систем с примерами и постановку общей задачи на курсовую работу.
* Второй раздел пояснительной записки посвящён алгоритму разработки структуры данных.
* Третий раздел посвящён непосредственно разработке программного средства. Он включает разработку схемы и структуры программы, алгоритмическое конструирование и конструирование пользовательского интерфейса.
* В четвёртом разделе пояснительной записки обосновываются технические приёмы программирования, то есть обоснование архитектурного построения программы и использования конкретных возможностей языка программирования.
* Пятый раздел посвящён тестированию отдельных функций и всей системы в целом. Проверяется корректность выполнения программы и её устойчивость к возможным ошибкам, оговаривается анализ результатов.
* Шестой раздел содержит руководство пользователя, в котором отображена последовательность действий, выполняемых пользователем для успешной работы системы.
* В заключении перечислены итоги выполнения курсовой работы.

После всех вышеперечисленных разделов далее размещены:

* список использованной в ходе работы литературы;
* обязательные приложения с текстом программы.

**1 АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ И СУЩЕСТВУЮЩИХ**

**АНАЛОГОВ**

**1.1** **Понятие технической поддержки**

Техническая поддержка или техподдержка — понятие, обобщающее собой и охватывающее множество услуг, посредством которых предприятия и организации обеспечивают помощь пользователям технологичных продуктов и услуг, например, таких, как мобильные и стационарные телефоны, телевизоры, компьютеры, программное обеспечение, а также других электронных и механических товаров и услуг.

В целом, техническая поддержка служит попыткам помочь конкретным пользователям решать возникающие конкретные проблемы с продуктом и его использованием, нежели задачи, связанные с обучением, индивидуальной настройкой или другими услугами поддержки.

Говоря о технической поддержке, могут иметь в виду и т. н. helpdesk — дословно с англ. «стол помощи», и service desk — досл. с англ. «стол услуги» и поддержку продукта или услуги, и поддержку клиента, и систему работы с инцидентами, заявками и проблемами.

Система учета заявок и их последующей обработки обычно называется [HelpDesk](https://okdesk.ru/). В общем случае, классическая программа или сервис хелпдеск предоставляют возможности по регистрации запросов от клиентов и контроля за их исполнением.

По мере появления современных инновационных технологий и ИТ сервисов уровень обслуживания клиентов постепенно повышается. Компании стараются максимально быстро и эффективно удовлетворять запросы потребителей. Когда у пользователей возникает какая-либо проблема, они могут отправить запрос, содержащий описание вопроса и просьбу о помощи в его решении. Благодаря этому служба технической поддержки пользователей (HelpDesk, Service Desk) сможет рассмотреть поступившую заявку и дать свой ответ.

Однако сам процесс отправки в HelpDesk не столь уж прост, как кажется.

Он состоит из нескольких этапов, включая:

* заполнение формы;
* отправку письма с подтверждением запроса (не всегда);
* закрепление заявки за конкретным специалистом Service Desk;
* анализ и решение вопроса;
* отправление уведомительных сообщений пользователям (не всегда) и т.п.

Компании, имеющие большую клиентскую базу, должны пользоваться системой запросов, необходимой для оперативного, организованного и качественного решения проблем потребителей.

Требования к системе технической поддержки пользователей:

* Бесплатная или не слишком недорогая.
* Активность разработчиков ПО и сообщества его пользователей - программный продукт должен обновляться и поддерживаться.
* Понятность и структурированность исходного кода.
* Корректная обработка UTF-8 и русских кодировок и в веб-интерфейсе, и при работе с электронной почтой.
* Перевод иноязычного интерфейса на русский (т.е., наличие локализации либо способа перевода).
* Интерфейс для работы с запросами пользователей.
* Поддержка разных отделов, при которой запросы автоматически поступают в нужный отдел.
* Возможность перенаправления запроса в другой отдел.
* Разграничение доступа персонала Service Desk к разным отделам.
* Механизм эскалации тикетов (когда на запрос не отвечают вовремя, об этом оперативно узнает менеджер HelpDesk).
* Функция персональной подписи в письмах каждого из отделов.
* Функция выделения какого-либо сообщения из переписки в отдельный запрос (split ticket).
* Наличие шаблонов ответов.
* Поддержка в письмах вложенных файлов.
* Поиск по тикетам: по отделу, дате, получателю, сотруднику, содержанию, теме.
* Опция внутренних комментариев в тикете.

Дополнительные критерии при выборе системы технической поддержки пользователей:

* Ведение статистики: число обработанных каждым специалистом заявок, среднее время, затраченное на ответ.
  1. **Анализ существующих программных продуктов**

Существует довольно много программных средств для служб технмической поддерки. Среди них выделим "Обработка заявок Service Desk" и "Help Desk", разработанный на базе Microsoft SharePoint.

"Обработка заявок" Service Desk - система, предоставляющая возможности для регистрации, обработки, контроля исполнения и поиска заявок пользователей, а также для создания формализованных автоматизированных бизнес-процессов и контроля над ходом их выполнения. Работая в системе, каждый сотрудник вовремя получает задание на выполнение своего этапа работ и оперативный доступ ко всей необходимой информации.

Из особенностей данного программного продукта можно выделить следующее:

* возможность автоматической маршрутизации заявок согласно задаваемым правилам бизнес-процесса;
* обеспечение совместной работы над документами процесса поддержки пользователей;
* обеспечение обратной связи посредством оповещений по e-mail и "прозрачности" прохождения заявки для ее заказчика;
* модульность системы - позволяет подключать дополнительный функционал к системе;
* реализация алгоритмов, оказания услуг, разной сложности;
* повышение качества анализа работы корпоративной службы поддержки.

К минусам данного программного продукта можно отнести следующее:

* Для работы необходимо интернет-подключение;
* пользователь имеет возможность оставлять заявку только по телефону. Это в значительной мере нагружает телефонную линию предприятия, увеличивает время поступления и обработки заявки;
* пользователь не имеет возможности сам формировать свою заявку, за него это должен делать диспетчер. Это также увеличивает время принятия и обработки заявки, а также подразумевает наличия дополнительного штата работников, что увеличивает затраты предприятия;
* данное программное средство не имеет возможности автоматического распределения заявок между специалистами в зависимости от их загруженности.

Еще одним примером является "Help Desk" - разработанное на базе SharePoint решение для службы технической поддержки, позволяющее внедрить в организациях процессы управления инцидентами, изменениями и уровнем сервиса.

Help Desk позволяет организовать учет заявок клиентов и пользователей, и наладить схему управления. Решение предназначено для служб технической, клиентской и сервисной поддержки 24/7. Модуль Help Desk позволяет создать единую точку сбора обращений и заявок пользователей, организовать цепочку управления инцидентами.

Помимо обеспечения жизненного цикла заявок от инициализации до закрытия, модуль дает возможность существенно повысить эффективность непосредственной работы над ними за счет инструментов совместной работы, привлечения к работе внешних пользователей, многоуровневой системы контроля, автоматических уведомлений и широкого спектра отчетов.

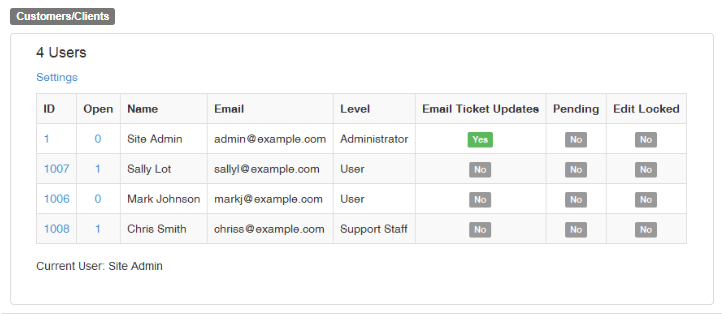


Рисунок 1 – пример интерфейса (список заявок)

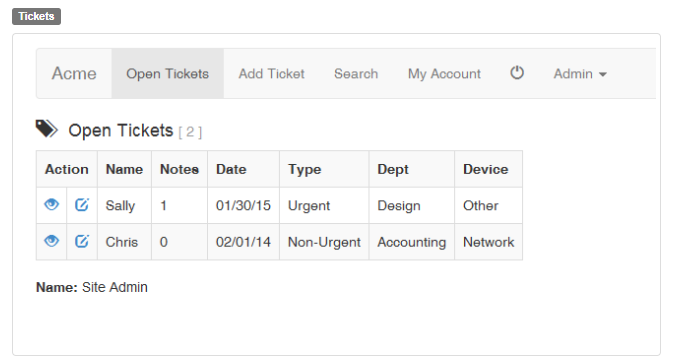


Рисунок 2– пример интерфейса (список заявок по выбранной категории)

Выделяются следующие положительные аспекты:

* классификация и диспетчеризация приходящих заявок, в том числе для назначения исполнителей, категории, приоритета и т.д.;
* отслеживание текущего статуса заявки;
* протоколирование работ, выполняемых по заявке, а также всех вносимых в нее изменений;
* построение отчетов;
* возможность создания любых представлений заявок (открытие, закрытые, на контроле), а также отслеживать и группировать их по клиентам или контрагентам. Также есть возможность настройки любой схемы и цепочки согласования и назначения инцидентов (дополнительная функциональность по запросу).

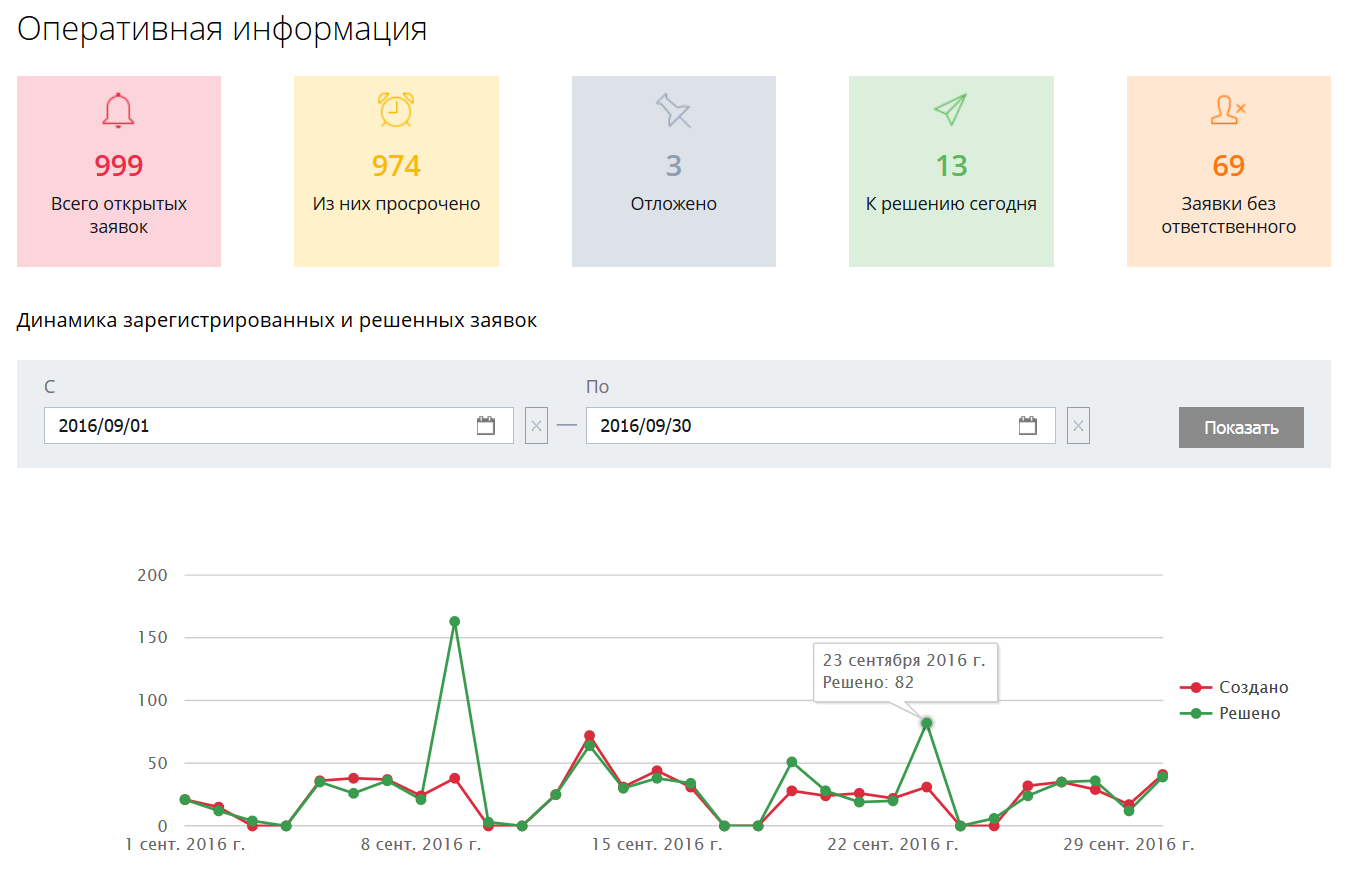


Рисунок 3– пример интерфейса (оперативная информация по заявкам)

Недостатки следующие:

* Ограниченная функциональность программного обеспечения;
* громоздкость приложения, что в значительной мере ухудшает его быстродействие;
* большое количество окон, вкладок и кнопок. В связи с этим пользователь вынужден тратить больше времени на изучение интерфейса программы, а также это ведет к снижению концентрации внимания на каком-либо одном объекте;
* также как и в случае с Service Desk "Обработка заявок", предприятию необходим дополнительный штат работников, что увеличит расходы.

Данные программные продукты способны работать как на локальном компьютере, так и в составе корпоративной сети предприятия.

Сравнительная характеристика разрабатываемого программного средства для службы технической поддержки абонентов сети и его аналогов рассмотрена в таблице 1.1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | "Help Desk" | "Обработка заявок" | Программное средство для службы технической поддержки абонентов сети |
| Язык разработки | C++ | C# | Java |
|  | "Help Desk" | "Обработка заявок" | Программное средство для службы технической поддержки абонентов сети |
| Проверка корректности ввода данных | - | + | + |
| Создание отчета в MS Exce | - | - | + |
| Протоколирование выполнения заявки | + | + | + |
| Сортировка, поиск по выбранному полю | + | + | + |
| Добавление, удаление, редактирование заявок | + | + | + |
| Необходимость сервисного обслуживания | + | + | + |

Как видно из таблицы, разрабатываемое программное средство имеет ряд преимуществ перед своими аналогами.

Программные продукты "Help Desk" и "Обработка заявок" обладают несколькими общими недостатками: для их полного функционирования необходим дополнительный штаб работников для обслуживания данных систем сбора и обработки заявок, что влечет за собой дополнительные расходы предприятия; они не способны создавать отчеты в формате MS Excel. Программное средство "Обработка заявок" не является серверным, что не позволяет работать с ним одновременно нескольким пользователям. Кроме того, пользователь оставляет свою заявку по телефону. Программное средство "Help Desk" не имеет возможности следить за корректностью вводимых данных пользователем, что может приводить к различным конфликтам.

Рассмотрев аналоги разрабатываемого средства, можно сделать вывод, что разрабатываемое программное средство для службы технической поддержки абонентов сети в полной мере позволит автоматизировать процессы подачи заявки, контроля ее исполнения, увеличить оперативность работников, вести учет выполненных заявок.

## Формирование требований к разрабатываемому ПС

Создать в среде **Java** приложение, позволяющее персоналу регистрировать и обрабатывать заявки клиентов. Реализовать автоматическое заполнение даты и времени поступления заявки. Программа должна хранить следующую информацию в списке клиентов: ФИО клиента, номер договора, домашний адрес, IP адрес (сетевой адрес компьютера). Список заявок должен содержать следующие поля: ФИО клиента, проблема, дата и время поступления заявки, состояние готовности. Заявка заполняется в том случае, если такой клиент существует. Поле ФИО клиента в заявке должно быть привязано ссылкой к списку клиентов (поле ФИО).

Входные данные:

* типизированный файл, хранящий информацию о клиентах;
* типизированный файл, хранящий уже имеющиеся заявки клиентов.

Выходные данные:

* типизированный файл/ы с информацией;
* вывести на экран, а также в текстовый файл все заявки, принятые в выбранный диапазон времени;
* вывести на экран, а также в текстовый файл все заявки, выполненные за текущие сутки;
* изобразить графически статистику поступления заявок за выбранный диапазон времени (день, неделя, месяц и т.д.).

Условия написания:

В приложении должны быть реализованы следующие функции:

* добавление, удаление, редактирование заявок;
* сортировка, поиск по выбранному полю;
* сохранение в файл, чтение из файла.

Для реализации поставленной задачи использовать динамические структуры данных.

1. **РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ**

## 2.1 Описание используемых структур данных

Для реализации поставленной задачи, программа должна выполнять несколько основных функций, реализуемых с помощью следующих структур данных.

**2.1.1 Связные списки**

Хранение данных в массивах имеет ряд недостатков. В неупорядоченном массиве поиск выполняется относительно медленно, тогда как в упорядоченном массиве медленно выполняется вставка. Удаление выполняется медленно в обеих разновидностях массивов. Кроме того, размер массива невозможно изменить после его создания.

Связанные списки — структура данных, которая решает некоторые из этих проблем. Гибкость связанных списков хорошо подходит для многих общих задач хранения данных. Кроме того, связанный список может заменить массив в качестве базы для других структур хранения данных (таких, как стеки и очереди). Более того, связанные списки часто могут использоваться вместо массивов (исключение составляют ситуации с частым произвольным доступом к отдельным элементам по индексу). Связанные списки не решают всех проблем хранения данных, но они на редкость универсальны, а на концептуальном уровне более просты, чем другие структуры данных (например, деревья).

**Строение связного списка**

В связанном списке каждый элемент данных встраивается в специальный объект, называемый элементом списка. Так как список содержит много однотипных элементов, для них удобно создать отдельный класс, отличный от класса самого связанного списка. Каждый элемент содержит ссылку на следующий элемент списка; поле, в котором эта ссылка хранится, обычно называется next. Объект списка содержит ссылку на первый элемент first. Отношения между объектами в этой архитектуре представлены на рис. 2.

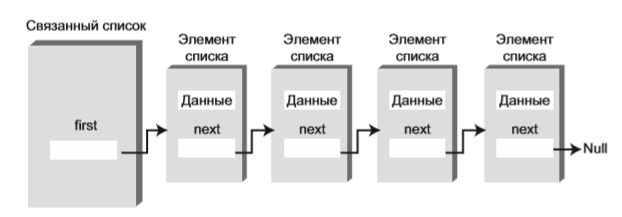


Рис. 2 Связный список

Подобные определения классов иногда называются самоотносимыми (self-referential), потому что объект класса содержит поле (next в данном случае) со ссылкой на объект того же типа.

Метод insertFirst() класса LinkList вставляет новый элемент в начало списка.

В этой позиции вставка выполняется проще всего, потому что first уже указывает на первый элемент. Чтобы вставить в список новый элемент, достаточно присвоить полю next созданного объекта ссылку на предыдущий первый элемент, а затем из-менить поле first так, чтобы оно указывало на только что вставленный элемент. Ситуация показана на рис. 2.1.

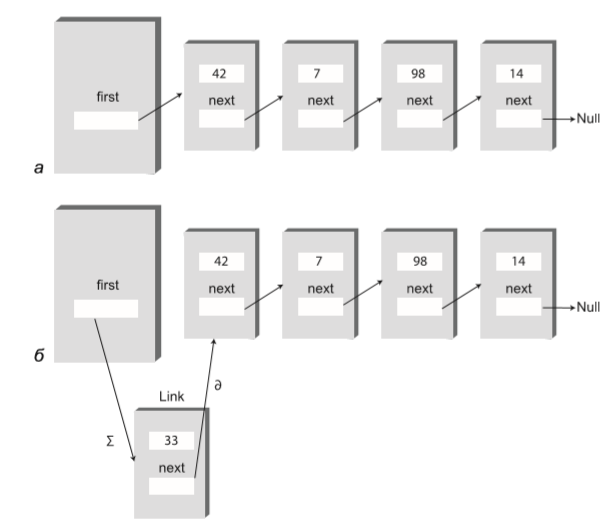


Рис. 2.1. Вставка элемента: а- до вставки; б-после вставки

Метод deleteFirst() является противоположностью insertFirst(). Он отсоединяет первый элемент, для чего в поле first заносится ссылка на второй элемент (который находится по значению поля next в первом элементе). На рис. 2.2 показано, как изменяется значение first при удалении объекта.

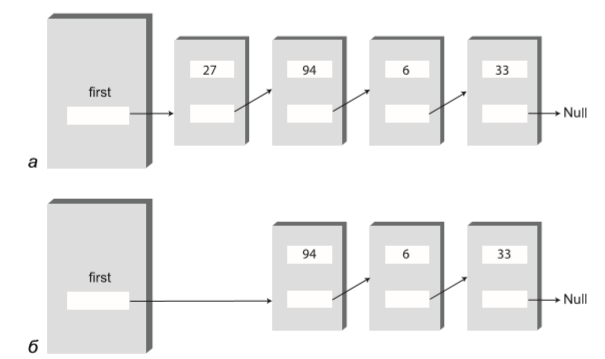


Рис. 2.2 Удаление элемента: а- до удаления; б- после удаления

**2.1.2 Двусвязные списки**

Рассмотрим следующую разновидность связанных списков: **двусвязный список.**

Потенциальным недостатком обычных связанных списков является сложность перемещения по списку в обратном направлении. Команда вида «current=current.next» позволяет легко перейти к следующему элементу списка, но соответствующего способа перехода к предыдущему элементу не существует. В некоторых ситуациях это ограничение создает проблемы. Допустим, в текстовом редакторе связанный список используется для хранения текста. Каждая строка на экране хранится в виде объекта String, встроенного в элемент списка. Когда пользователь перемещает курсор вниз по экрану, программа переходит к следующему элементу списка для обработки или вывода следующей строки. Но что произойдет, если пользователь переместит курсор вверх? При использовании обычного связанного списка придется вернуть переменную current (или ее аналог) к началу списка, а затем перебрать все элементы до предыдущего элемента. Конечно, такой способ неэффективен — перемещение к предыдущей строке должно осуществляться за один шаг.

Двусвязный список предоставляет такую возможность. Он позволяет перемещаться по списку как в прямом, так и в обратном направлении. Дело в том, что каждый элемент хранит ссылки на два других элемента вместо одного. Первая ссылка указывает на следующий элемент, как и в обычных списках. Вторая ссылка указывает на предыдущий элемент. Структура такого списка изображена на рис. 2.1

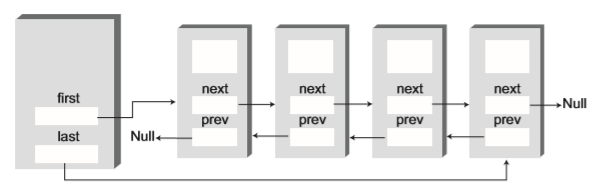


Рис. 2.3. Двусвязный список

К недостаткам двусвязных списков следует отнести то, что при каждой вставке или удалении ссылки вам приходится изменять четыре ссылки вместо двух: две связи с предыдущим элементом и две связи со следующим элементом. И конечно, каждый элемент списка занимает чуть больше места из-за дополнительной ссылки.

## Вставка элемента в двусвязный список

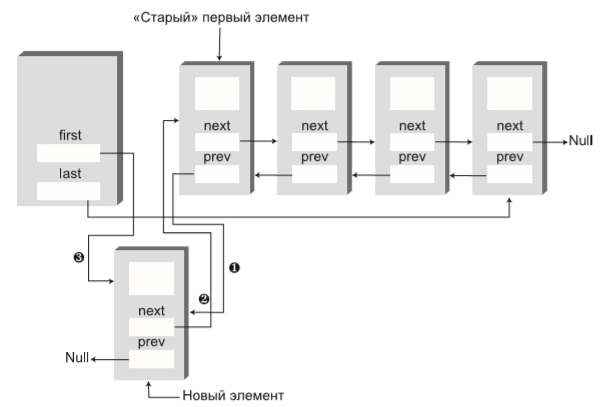


Рис. 2.4. Вставка в начале

## Класс двусвязного списка содержит несколько методов вставки. Метод insertFirst() вставляет новый элемент в начале списка, метод insertLast() — в конце, а метод insert() — по заданному ключу. Убедившись в том, что список не пуст, метод insertFirst() записывает ссылку на новый элемент в поле previous «старого» первого элемента, а ссылку на «старый» первый элемент — в поле next нового элемента. Наконец, в поле first заносится ссылка на новый элемент.

Метод insertLast() выполняет те же действия в конце списка; он является своего рода «зеркальным отражением» insertFirst(). Метод insertAfter() вставляет новый элемент после элемента с заданным ключом. Операция несколько усложняется, потому что в этой ситуации необходимо изменить четыре ссылки. Прежде всего следует найти элемент с заданным ключом, затем, если позиция вставки находится не в конце списка, необходимо создать две связи между новым и следующим элементом, и еще две — между current и новым элементом.

Метод insertAfter() вставляет новый элемент после элемента с заданным клю-чом. Операция несколько усложняется, потому что в этой ситуации необходимо изменить четыре ссылки. Прежде всего следует найти элемент с заданным ключом. Затем, если позиция вставки находится не в конце списка, необходимо создать две связи между новым и следующим элементом, и еще две — между current и новым элементом. Процесс показан на рис. 2.3

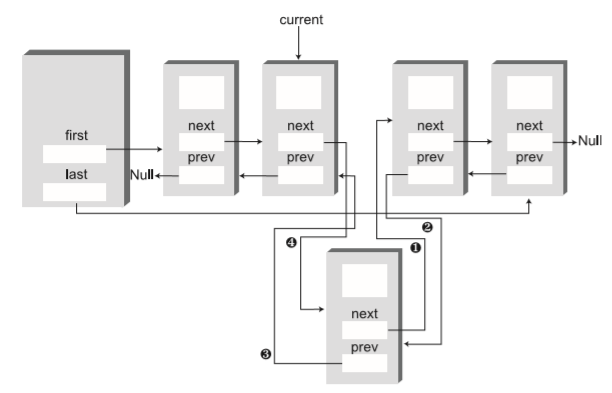


Рис. 2.5. Вставка в произвольной позиции

Если новый элемент должен вставляться в конце списка, то его поле next должно содержать null, а поле last — ссылку на новый элемент.

**Удаление**

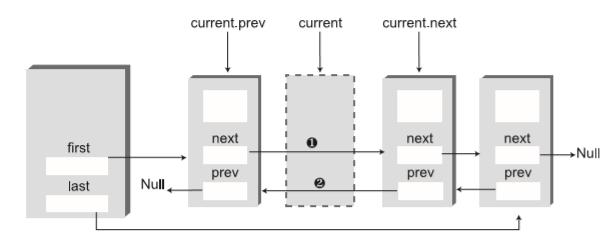
****

Рис. 2.6. Удаление произвольного элемента

Класс также содержит три метода удаления: deleteFirst(), deleteLast() и delete(). Первые два относительно тривиальны; в методе delete() удаляется элемент current. Если удаляемый элемент не является ни первым, ни последним в списке, то в поле next элемента current.getPrevious() (элемент, предшествующий удаляемому) заносится ссылка на current.getNext() (элемент, следующий после удаляемого), а в поле previous элемента current.getNext() заносится ссылка на current.getPrevious(). В результате элемент current исключается из списка.

Удаление реализуется следующими двумя командами: current.getPrevious().getNext()=current.getNext(); current.getNext().getPrevious() = current.getPrevious().

Если удаляемый элемент находится в первой или последней позиции списка, это особый случай, потому что ссылка на следующий или предыдущий элемент должна быть сохранена в поле first или last.

**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Схема программы**

Схема работы программы изображена на рис. 3.1(а-ж)

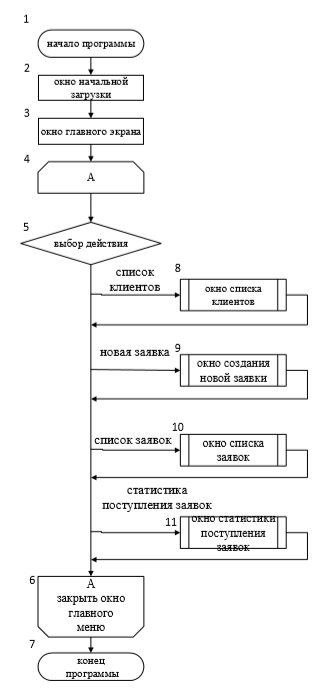
****

Рис. 3.1(а) Схема работы программы

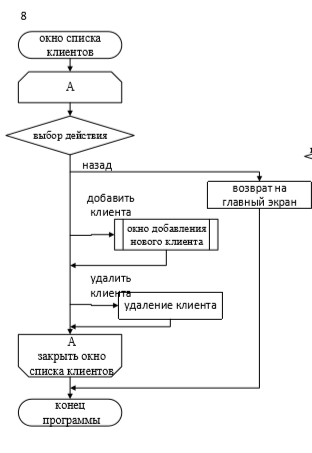


Рис.3.1(б) Схема окна списка клиентов

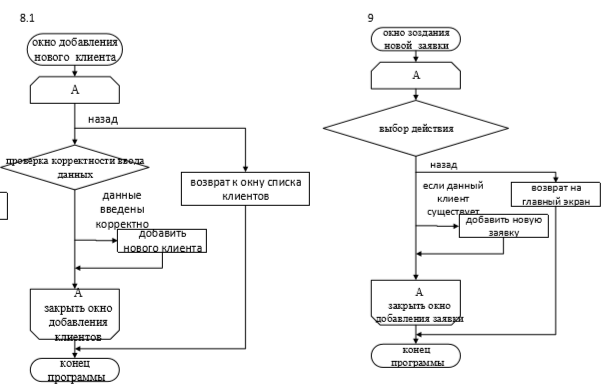


Рис. 3.1(в) Схемы окон добавления нового клиента и создания новой заявки

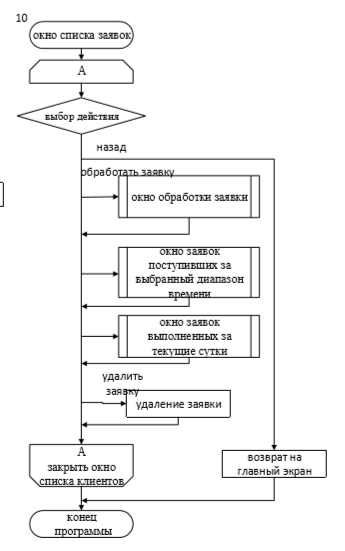


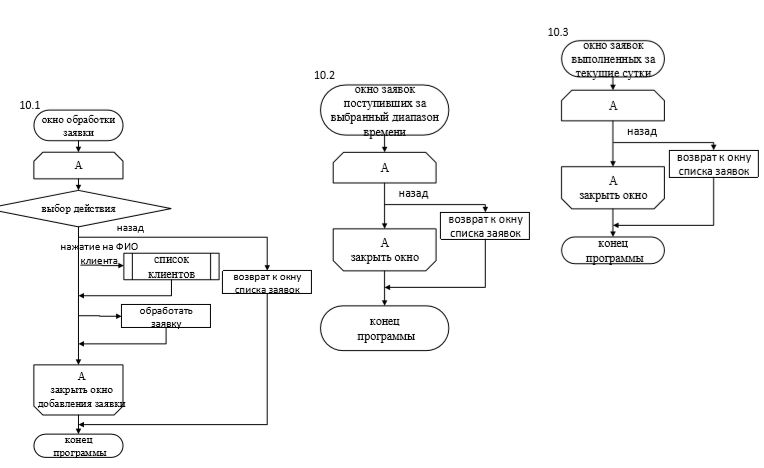
Рис. 3.1(г)Схема окна списка заявок

Рис. 3.1(д) Схемы окон действий над списком заявок

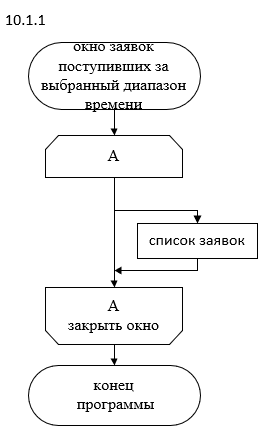


Рис. 3.1(е) Схема окна списка заявок, поступивших за выбранный диапазон

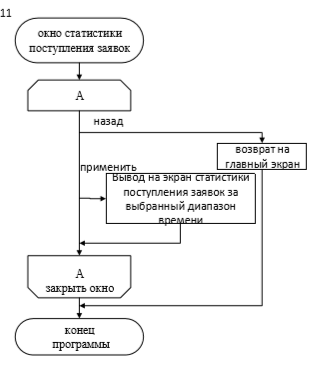


Рис. 3.1(ж) Схема окна статистики поступления заявок

## Алгоритм добавления новой заявки

Одним из основных алгоритмов программы был алгоритм добавления новой заявки (нового клиента). Когда пользователь вводит в поля данные нового клиента, каждое поле проверяется на корректность:

* Поле ФИО должно содержать только символы латиницы, пробелы, а также допустимо использование знака «-»;
* Поле ip-адреса (так как было решено использовать ipv4) должно представлять собой 4 числа в диапазоне от 0 до 255, разделённых между собой точками;
* Поле номера договора должно содержать только целые неотрицательные числа;
* Поле домашнего адреса допускает использование символов кириллицы, цифр, пробела, точки, запятой и дефиса.

Соответственно и при добавлении новой заявки осуществляется проверка на ввод, а также проверяется, существует ли в списке клиент, ФИО которого мы указываем при добавлении заявки. Схема этого алгоритма приведена на рис. 3.1.1

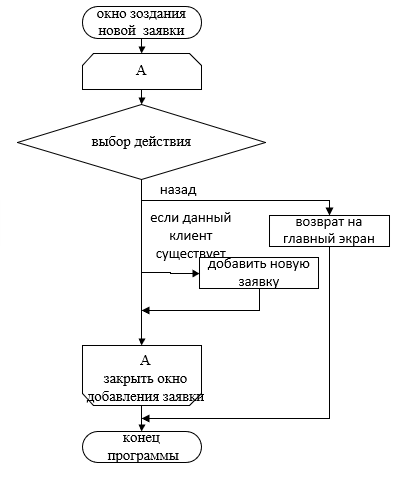


Рис. 3.1.1 Схема алгоритма создания новой заявки

* 1. **Проектирование главного меню системы**

На главном меню расположены 4 кнопки для перехода на другие окна программы рис. 4.1

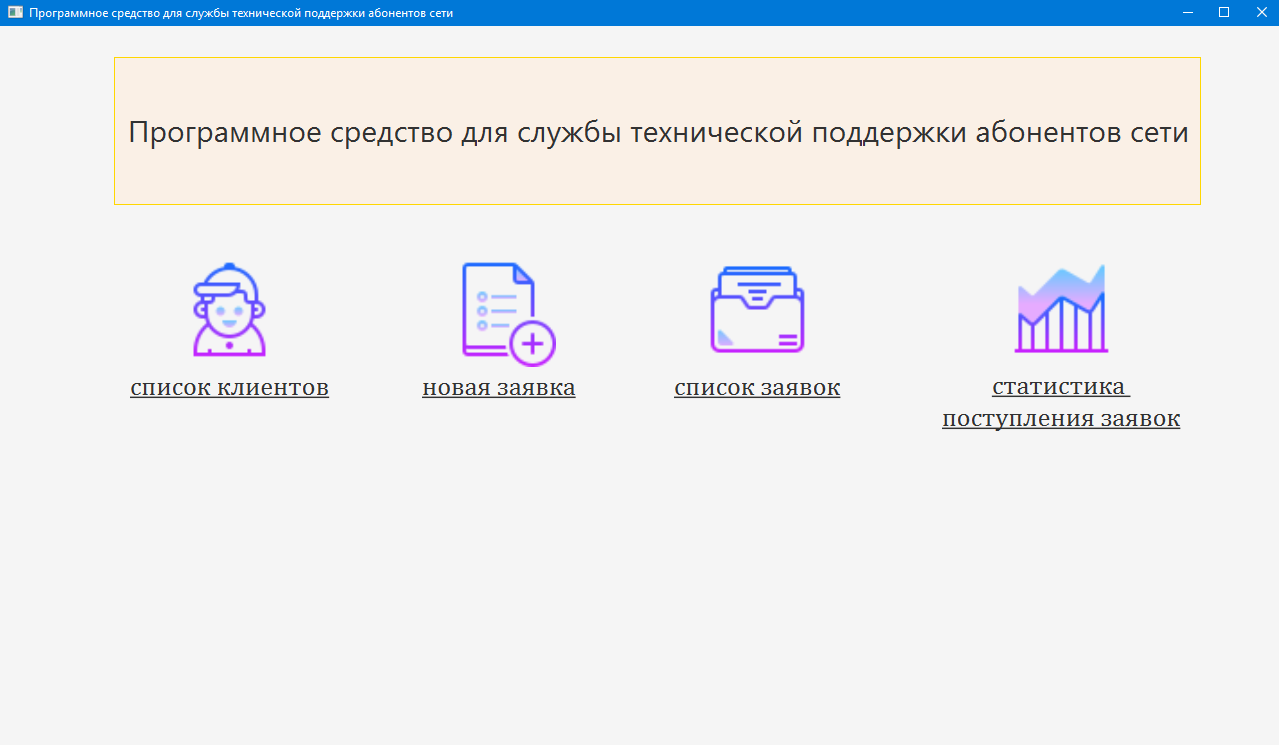


Рис. 4.1 Главное меню программы

**3.3 Проектирование окна списка клиентов сети**

Окно списка клиентов сети должно в первую очередь отображать список клиентов, что очень удобно сделать с помощью соответствующей таблицы. Также у пользователя есть возможность добавить клиента и удалить. Когда пользователь выполняет некоторые манипуляции со списком клиентов, соответственно изменённый список будет отображаться на экране. Также, с помощью текстовых полей таблицы у пользователя есть возможность редактировать информацию о клиентах рис. 3.3, рис. 3.3.1

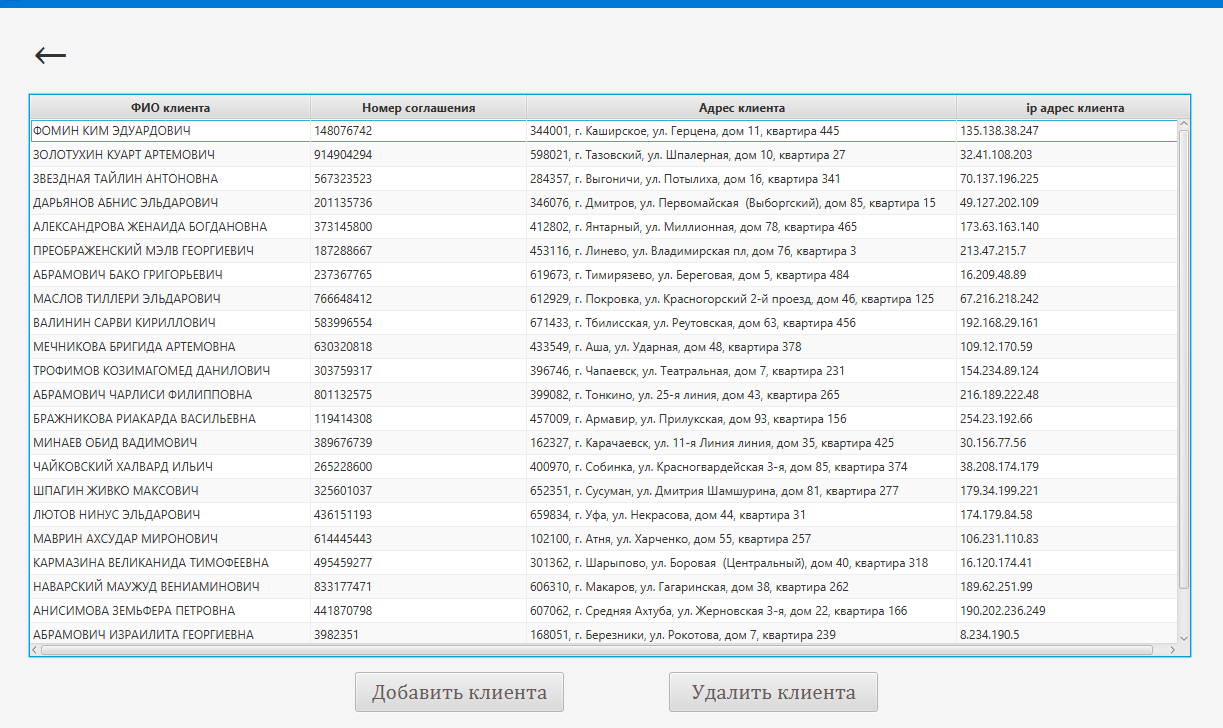


Рис. 3.3 Окно списка клиентов сети

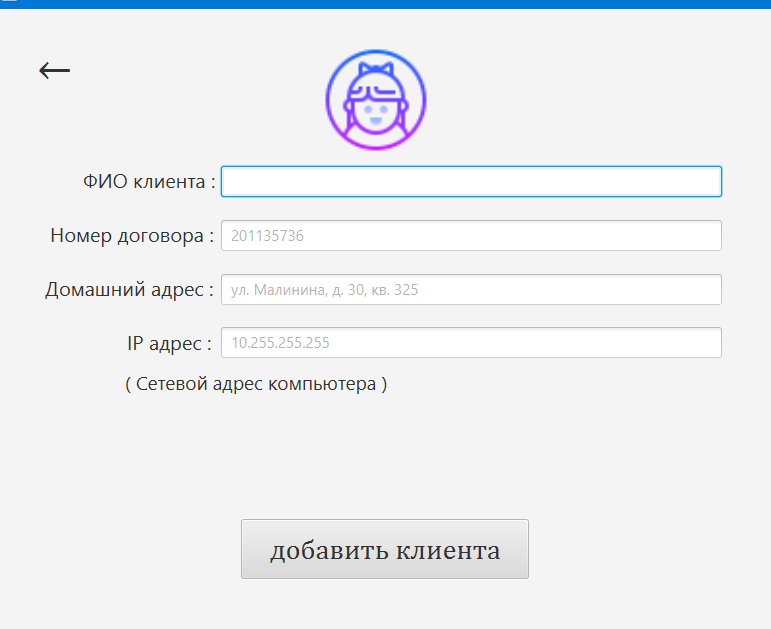
****

Рис. 3.3.1 Окно добавления клиентов сети

Если пользователь ввёл некорректные данные, ему будет выведено сообщение об ошибке рис. 3.3.2

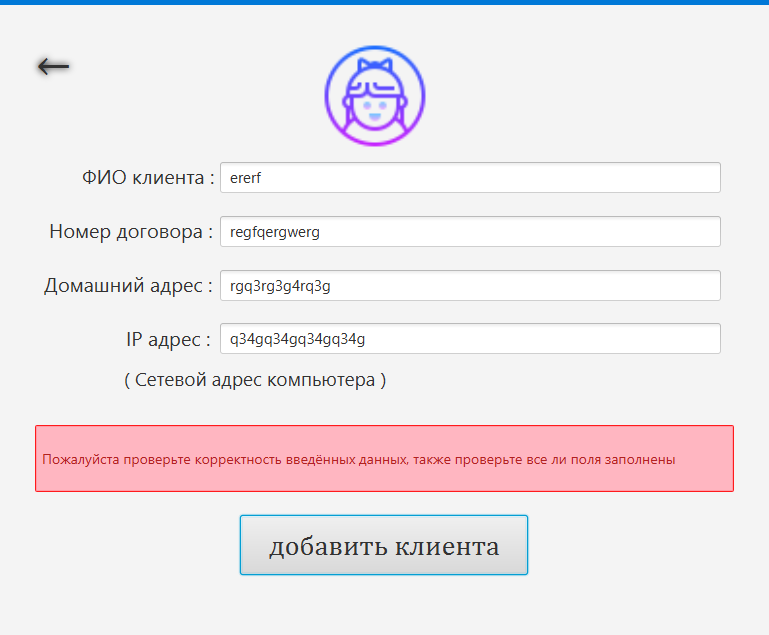


Рис. 3.3.2 Сообщение об ошибке

**3.4 Проектирование окна списка поступивших заявок**

В данном окне пользователь может увидеть полный список поступивших заявок, как всех, так и поступивших в выбранный диапазон времени рис. 3.4.1, рис. 3.4.2 или заявок, выполненных за текущие сутки

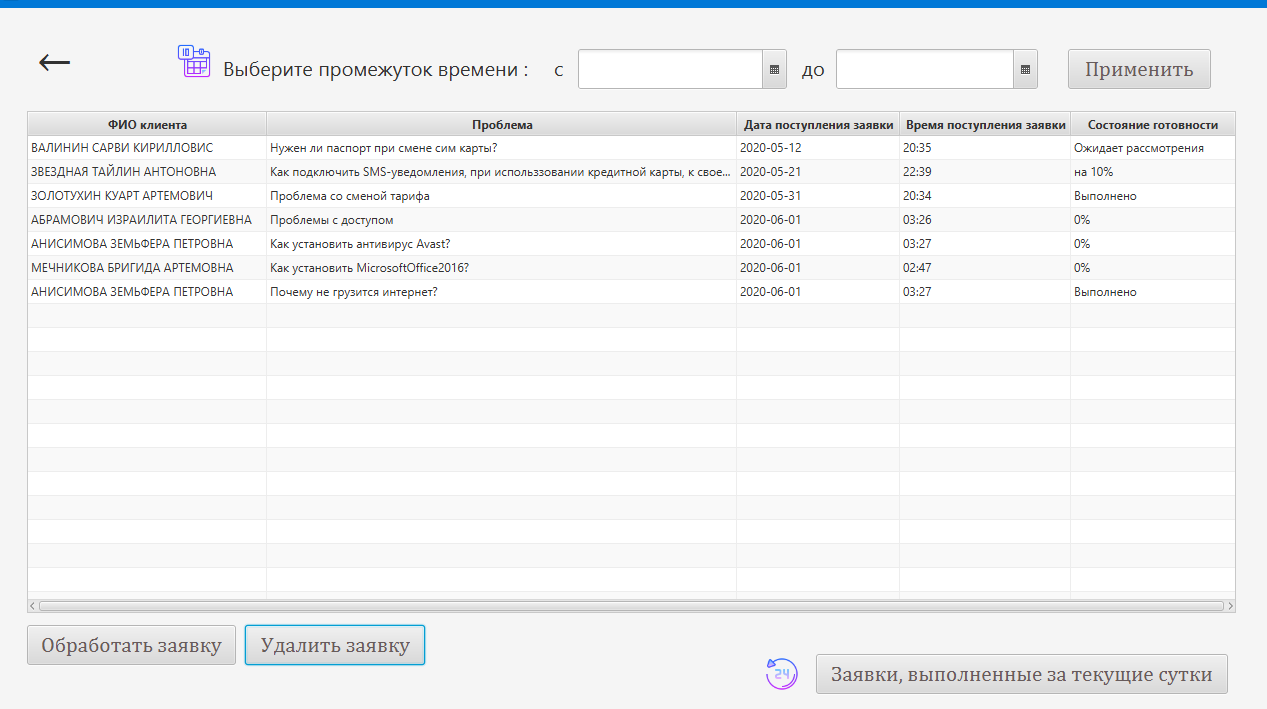


Рис. 3.4.1 Сообщение об ошибке

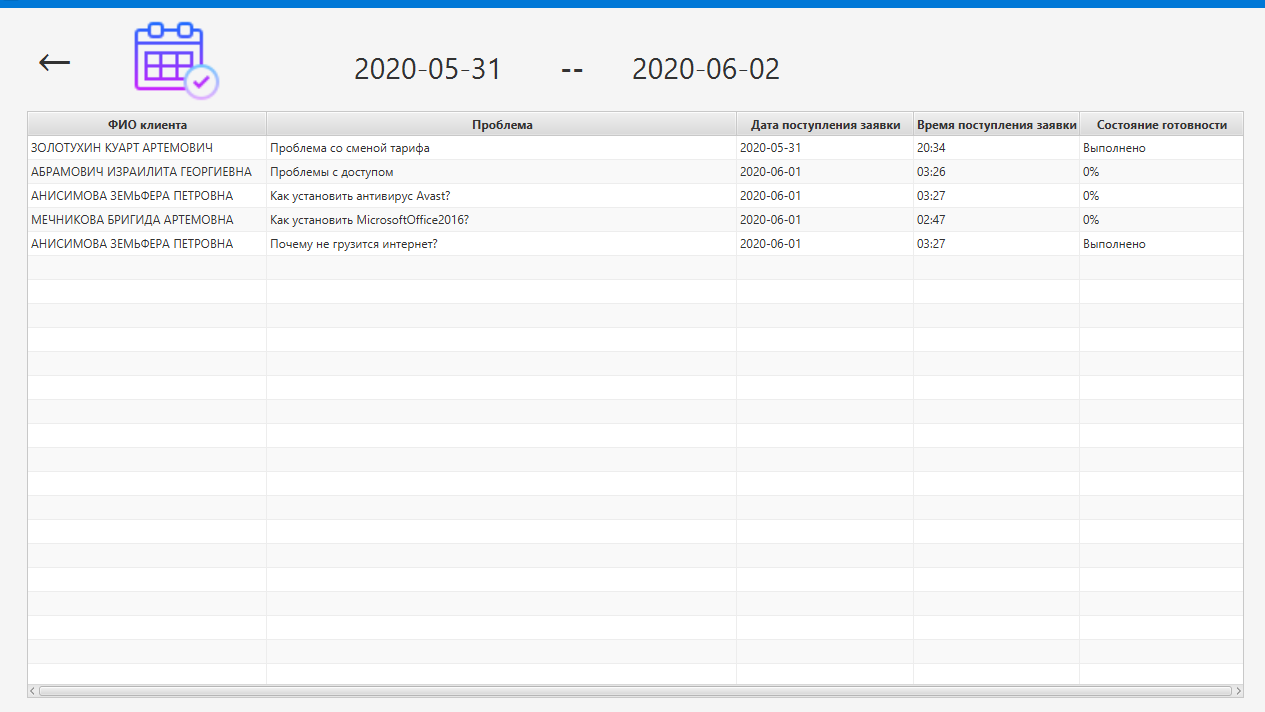


Рис. 3.4.2 Заявки, поступившие в выбранный диапазон времени

**3.5 Проектирование окна статистики поступления заявок**

Программное средство также может отображать на экран статистику количества поступивших заявок за некоторый (любой, выбранный пользователем) диапазон времени рис. 3.5

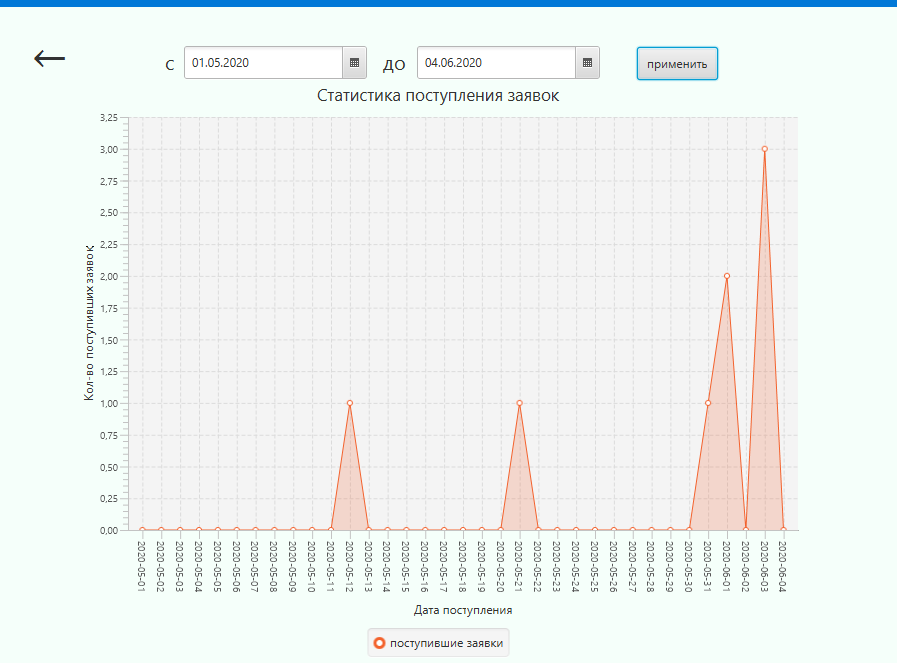


Рис. 3.5 Статистика поступивших заявок

1. **ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ**

**ПРОГРАММИРОВАНИЯ**

Данная программа, как и большинство программ с графическим интерфейсом строятся по принципу: Модель-Вид-Контроллер, где контроллер работает с действиями пользователя, вид отображает данные, модель формирует работу с данными.

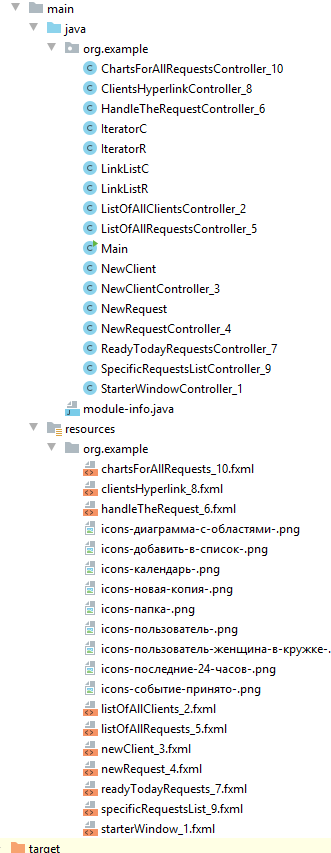


Рис. 4 Контроллеры и классы программы

Созданная программа включает в себя 10 различных модулей:

* SterterWindowController\_1

Эта форма отвечает за показ главного меню программы. Является главным окном программы. На нём находится 4 кнопки (Button), которые отвечают за переход на другие окна программы. На форме также находятся еще 4 иконки (ImageView).

* ListOfAllClientsController\_2

Модуль, отвечающий за вывод на экран списка всех клиентов. На фоне находится таблица, отвечающая за вывод списка и 2 кнопки, для удаления клиента из списка и для создания нового клиента. При нажатии на кнопку «добавить клиента» программа переходит на окно создания нового клиента. На форме находится кнопка (Button), при нажатии на которую пользователь возвращается на главный экран.

* NewClientController\_3

Эта форма отвечает за создание нового клиента. На форме находятся 4 поля для вывода текста (Text). Еще одно фигура (rectangle) отвечает за вывод сообщения об ошибке или об успешном добавлении нового. На форме находится кнопка (Button), при нажатии на которую пользователь возвращается на главный экран.

* NewRequestController\_4

Эта форма отвечает за регистрацию новой заявки. На ней находятся 2 поля ФИО (TextField) и поле для ввода проблемы (TextField). Также при нажатии на кнопку добавить заявку, автоматически добавляется время и дата поступления заявки. На форме находится кнопка (Button), при нажатии на которую пользователь возвращается на главный экран.

* ListOfAllRequestsController\_5

Модуль, отвечающий за вывод на экран списка всех заявок. На фоне находится таблица (TableView), отвечающая за вывод списка и 2 кнопки, для удаления заявки из списка и для обработки заявки клиента. Также на форме имеются 3 кнопки, одна из которых отвечает за возврат на главное меню, другая за вывод заявок, выполненных за текущие сутки, а кнопка «применить» считывает диапазон с двух полей (DatePicker) и после открывает окно, ос списком заявок, поступивших в выбранный промежуток времени.

* HandleTheRequestController\_6

Эта форма предоставляет возможность обработать выбранную заявку. На этой форме находится поле с заявкой (TextField). А вверху окна есть поле ФИО (HyperLink), при нажатии на которое открывается окно со списком клиентов сети. На форме находится кнопка (Button), при нажатии на которую пользователь возвращается на главный экран.

* ClientsHyperLinkController\_8

Эта форма отвечает за показ списка клиентов. На фоне находится таблица, отвечающая за вывод списка.

* SpecificRequestsListController\_9

Модуль, отвечающий за вывод на экран списка заявок, принятых в выбранный пользователем диапазон времени. На форме находится кнопка (Button), при нажатии на которую пользователь возвращается на главный экран.

* ChartsForAllRequestsController\_10

Эта форма представляет возможность просмотреть статистику за указанный промежуток времени. На форме расположены два поля (DatePicker) с помощью которых можно выбрать нужный промежуток времени для просмотра статистики. На данном окне есть линейный график (AreaChart), который позволяет пользователю увидеть количество поступивших заявок. Также на форме находится кнопка (Button), при нажатии на которую пользователь возвращается на главный экран.

**5** **ТЕСТИРОВАНИЕ**

**5.1 Результаты тестирования программы**

Таблица 1 – Тест-кейс наличия главного меню

|  |  |  |
| --- | --- | --- |
| Тест[1] | Ожидаемый результат | Результат |
| **Наличие главного меню и возможность работы с ним**   1. Запустить программу 2. Проверить элементы главного меню | 1. При запуске программы попадаем на главное меню. 2. Отображаются следующие элементы:   - кнопка «Список клиентов», кнопка для перехода на окно «Новая заявка», кнопка для перехода на окно «Список заявок», кнопка для перехода на окно «Статистика поступления заявок», 4 иконки. | Успех |

Таблица 2 – Тест-кейс работы кнопок главного меню

|  |  |  |
| --- | --- | --- |
| Тест[2] | Ожидаемый результат | Результат |
| **Работа с кнопками главного меню**   1. Запуск программы 2. Нажать на любую кнопку главного меню | 1. Открывается окно главного меню. 2. Кнопки активны и кликабельны во всей области. При нажатии происходит переход на соответствующее окно приложения. При наведении мыши на кнопки, у них появляется эффект «тени». | Успех |

Таблица 3 – Тест-кейс отображения окна «Список клиентов»

|  |  |  |
| --- | --- | --- |
| Тест[3] | Ожидаемый результат | Результат |
| **Отображение окна «Список клиентов»**   1. Открыть окно «Список клиентов» 2. Проверить элементы окна | 1. Перейти на окно «Список клиентов». 2. Отображаются следующие элементы:  * кнопка «Добавить клиента»; * кнопка «Назад»; * кнопка «Удалить клиента»; * таблица со списком всех клиентов. | Успех |

Таблица 4 – Тест-кейс отображения окна «Добавить клиентов»

|  |  |  |
| --- | --- | --- |
| Тест[4] | Ожидаемый результат | Результат |
| **Отображение окна «Добавить клиента»**   1. Открыть окно «добавить клиента» 2. Проверить элементы окна 3. Проверить сообщение о корректности   /некорректности заполнения полей | 1. Перейти на окно «Добавить клиента». 2. Отображаются следующие элементы:  * текстовое поле «ФИО клиента», текстовое поле «Номер договора», текстовое поле «Домашний адрес», текстовое поле «IP-адрес»; * кнопка «Добавить клиента»; * иконка (рисунок).   3.Отображаются следующие сообщения:   * если при добавлении клиента не все поля заполнены верно, следовательно выводится сообщение об ошибке; * если при добавлении клиента все поля заполнены верно, соответственно выводится сообщение об успешном добавлении. | Успех |

Таблица 5 – Тест-кейс контента окна «Новая заявка»

|  |  |  |
| --- | --- | --- |
| Тест[5] | Ожидаемый результат | Результат |
| **Отображение контента окна «Новая заявка»**   1. Перейти на окно «Новая заявка» 2. Проверить элементы окна 3. Проверка корректности ввода | 1. Открывается окно «Оценить услугу». 2. Отображаются следующие элементы:  * рисунок; * заглавие, содержащее время, надпись «Банк» и кнопку возврата; * поле, для ввода номера; * 5 кнопок, представляющих собой смайлы.  1. Отображаются следующие элементы:  * если все поля заполнены верно, то выводится сообщение об успешной добавлении заявки; * если были введены некорректные данные, то выводится сообщение об ошибке; * если клиента, отправляющего заявку, нет в списке клиентов сети. | Успех |

Таблица 6 – Тест-кейс контента окна «Статистика поступления заявок»

|  |  |  |
| --- | --- | --- |
| Тест[6] | Ожидаемый результат | Результат |
| **Отображение контента окна «Статистика поступления заявок»**   1. Перейти на окно «Статистика» 2. Проверить элементы окна | 1. Открывается окно «Статистика поступления заявок». 2. Отображаются следующие элементы:  * 2 поля выбора даты (DatePicker); * линейный график(AreaChart); * кнопка «применить»; * кнопка «назад». | Успех |

Таблица 7 – Тест-кейс сохранения данных о клиенте в файл

|  |  |  |
| --- | --- | --- |
| Тест[7] | Ожидаемый результат | Результат |
| **Сохранение клиента в файл**   1. Открыть окно «Добавить клиента» 2. Ввести данные 3. Нажать кнопку «добавить клиента» 4. Проверить, сохранился ли клиент в файл 5. Проверить, отображается ли он в списке клиентов | 1. Открывается окно «Добавить клиента». 2. Элементы окна кликабельны и работают корректно. 3. Клиент записывается в список клиентов. 4. В файле клиент находится последним в списке. 5. В окне «список клиентов» последним отображается добавленный клиент. | Успех |

Таблица 8 – Тест-кейс сохранения добавленной заявки в файл

|  |  |  |
| --- | --- | --- |
| Тест[8] | Ожидаемый результат | Результат |
| **Сохранение добавленной заявки в файл**   1. Открыть окно «Добавить заявку» 2. Заполнить поля заявки 3. Нажать кнопку «добавить заявку» 4. Проверить, сохранилась ли заявка | 1. Открывается окно «Добавить заявку». 2. В полях отображается введенная информация. 3. Заявка добавлена в случае корректного заполнения полей. 4. В файле с заявками сохранились данные о добавленной заявке. | Успех |

Таблица 9 – Тест-кейс открытие окна заявок, выполненных за текущие сутки

|  |  |  |
| --- | --- | --- |
| Тест[9] | Ожидаемый результат | Результат |
| **Открытие окна заявок, выполненных за текущие сутки**   1. На окне списка всех поступивших заявок нажать на кнопку «список заявок, выполненных за текущие сутки» | 1. Открывается окно с таблице, в которой находятся все заявки, выполненные за текущие сутки. | Успех |

Таблица 10 – Тест-кейс корректной работы графиков

|  |  |  |
| --- | --- | --- |
| Тест[10] | Ожидаемый результат | Результат |
| **Корректная работа графиков**   1. Перейти на окно «статистика поступления заявок» 2. Поменять промежуток времени для выборки статистики 3. Сверить данные с хранящимися в файлах | 1. Открывается окно «Статистика поступления заявок». 2. Данные на графиках изменяются согласно промежутку времени. 3. Данные, отображенные на графиках, совпадают с расчетными по всем файлам. | Успех |

* 1. **Подтверждение корректности тестирования**

**Тест [1]- Рис. 5.1**

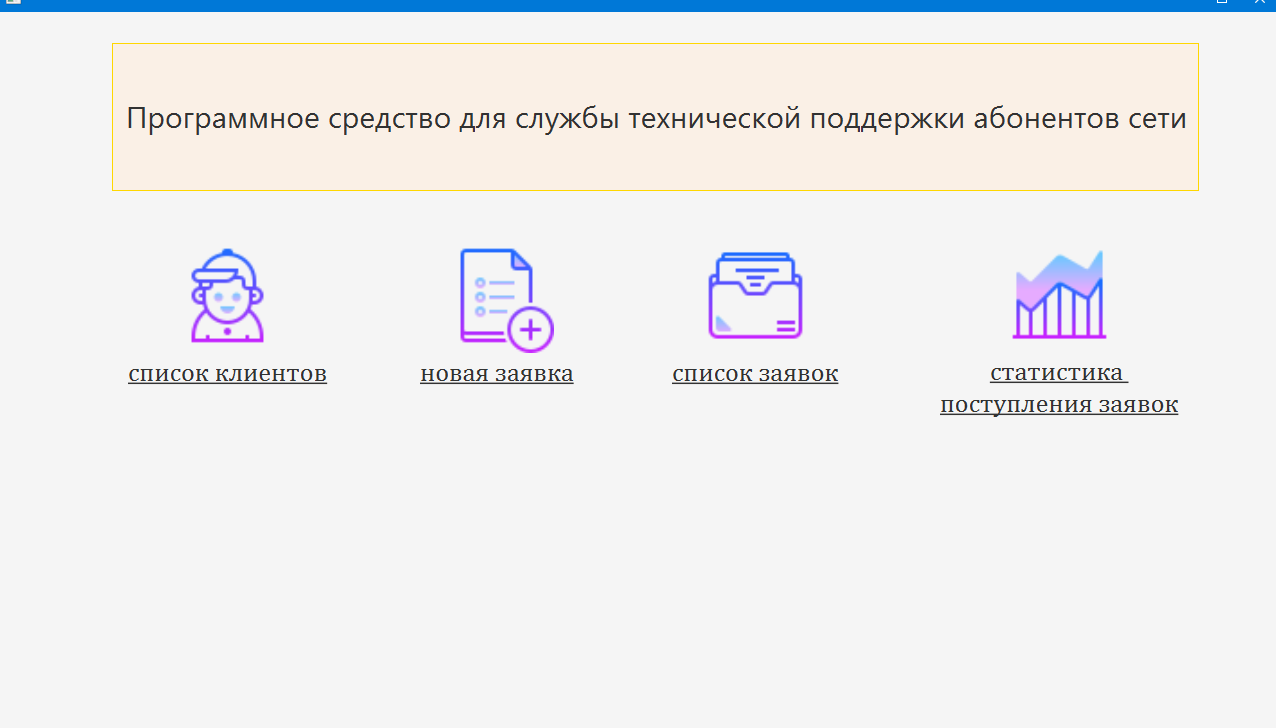
****

Рис. 5.1 Наличие главного меню и возможность работы с ним

**Тест [2]- Рис. 5.2**

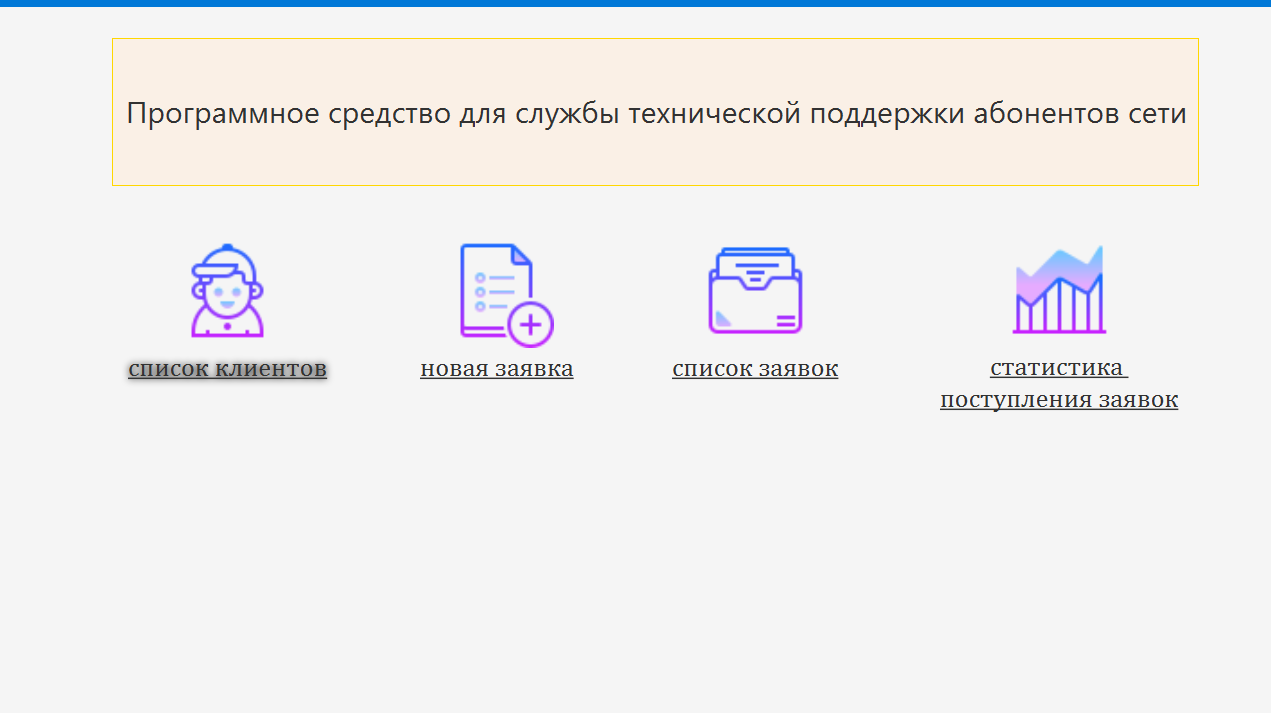
****

Рис. 5.2 Работа с кнопками главного меню (нажатие например на кнопку «список клиентов»)

**Тест [3]- Рис. 5.3**

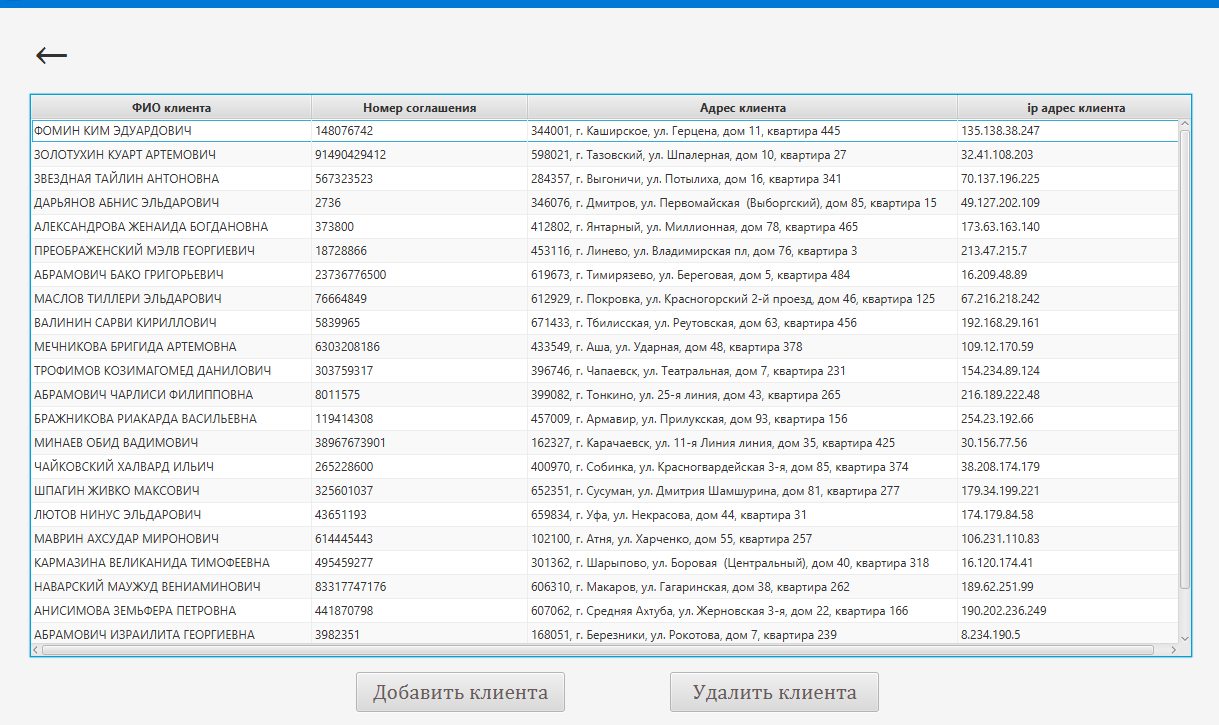
****

Рис. 5.3 Отображение окна «Список клиентов»

**Тест [4]- Рис. 5.4 , Рис. 5.4(3-ий пункт)(а), Рис. 5.4(3-ий пункт)(б)**

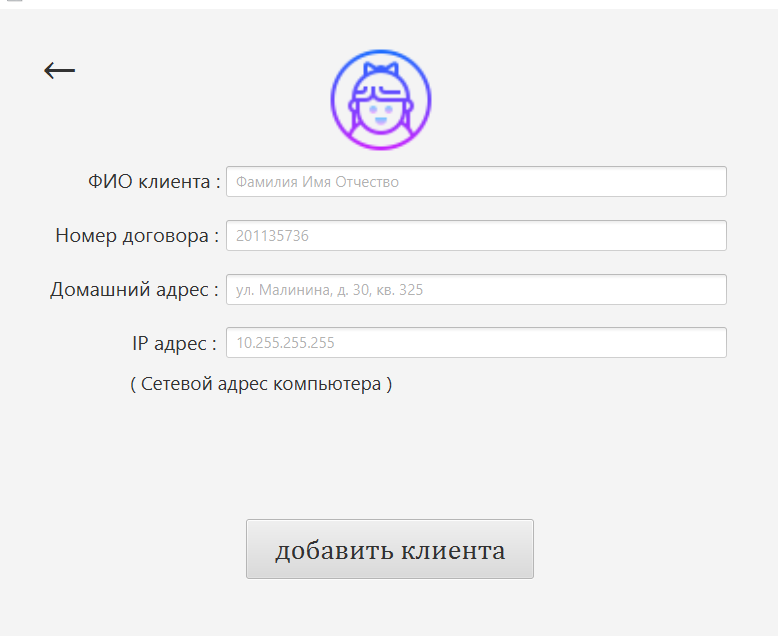
****

Рис. 5.4 Отображение окна «Добавить клиента»

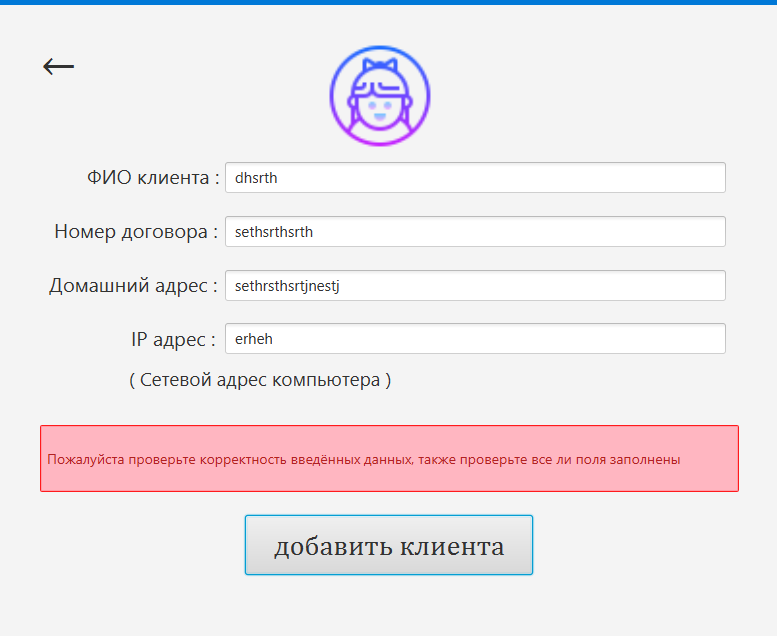


Рис. 5.4 (3-ий пункт) (а) Отображение окна «Добавить клиента»

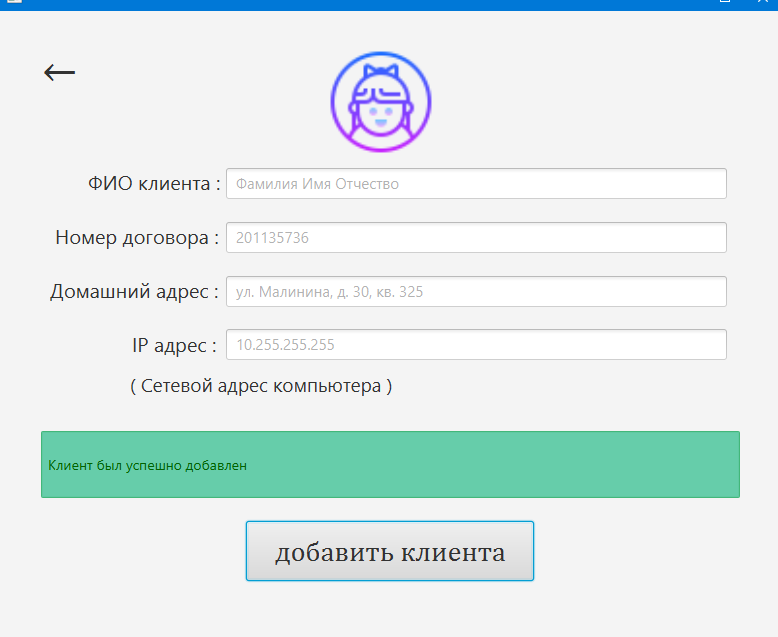


Рис. 5.4 (3-ий пункт) (б) Отображение окна «Добавить клиента»

**Тест [5]- Рис. 5.5(а-в)**

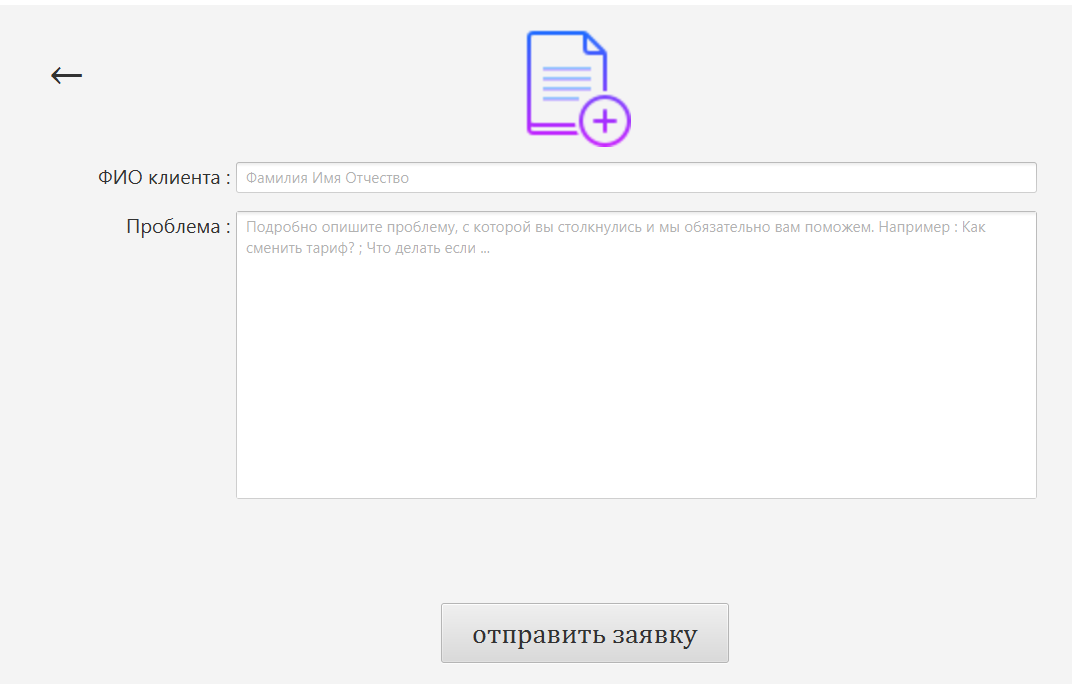


Рис. 5.5(а) Отображение контента окна «Новая заявка»

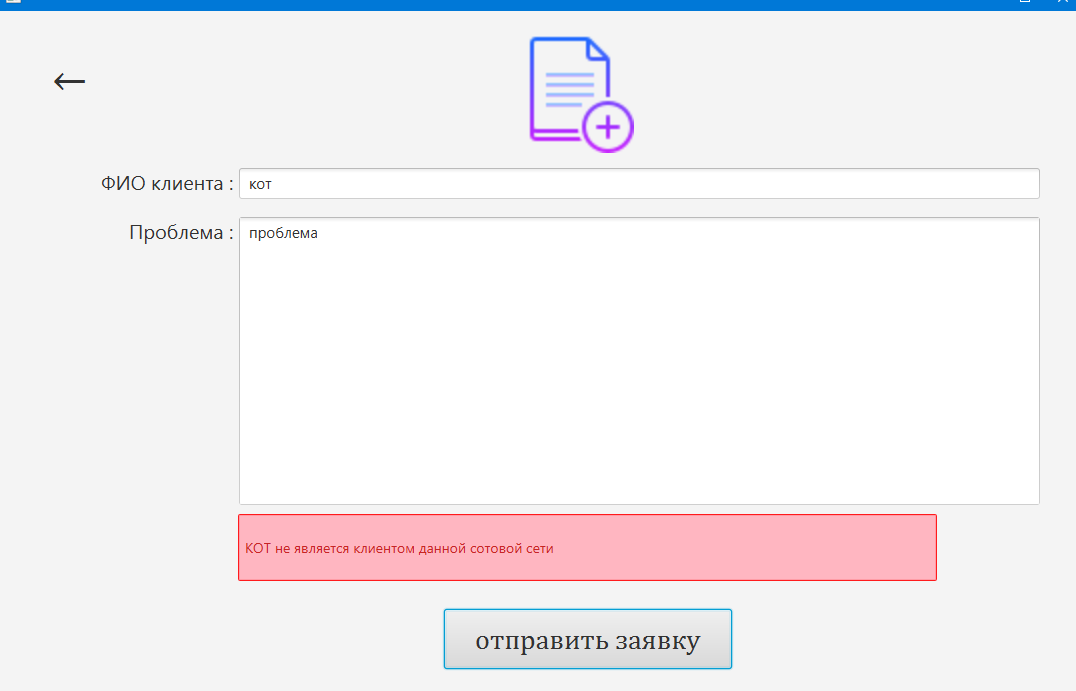
****

Рис. 5.5(б) (3-ий пункт) Отображение контента окна «Новая заявка»

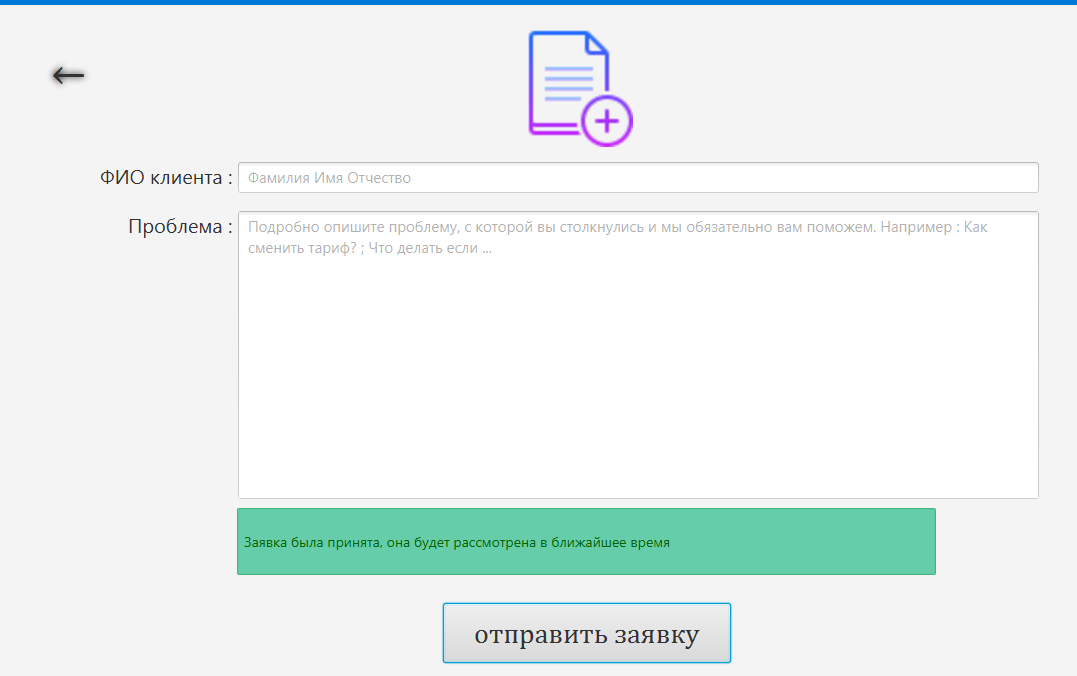


Рис. 5.5(в) (3-ий пункт) Отображение контента окна «Новая заявка»

**Тест [6]- Рис. 5.6**

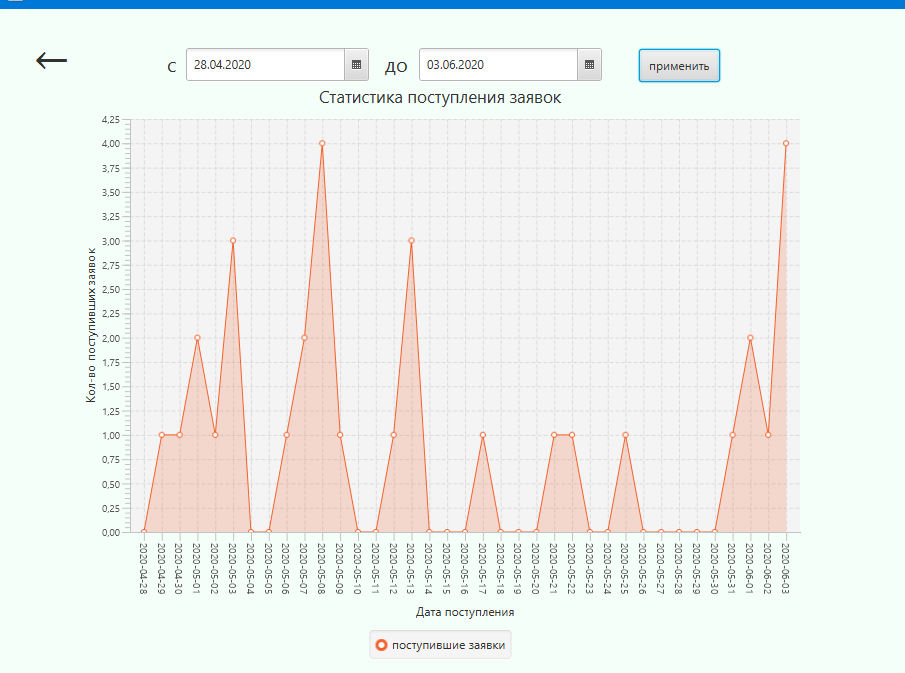


Рис. 5.6 Отображение контента окна «Статистика поступления заявок»

**Тест [7]- Рис. 5.7.(1-4)**

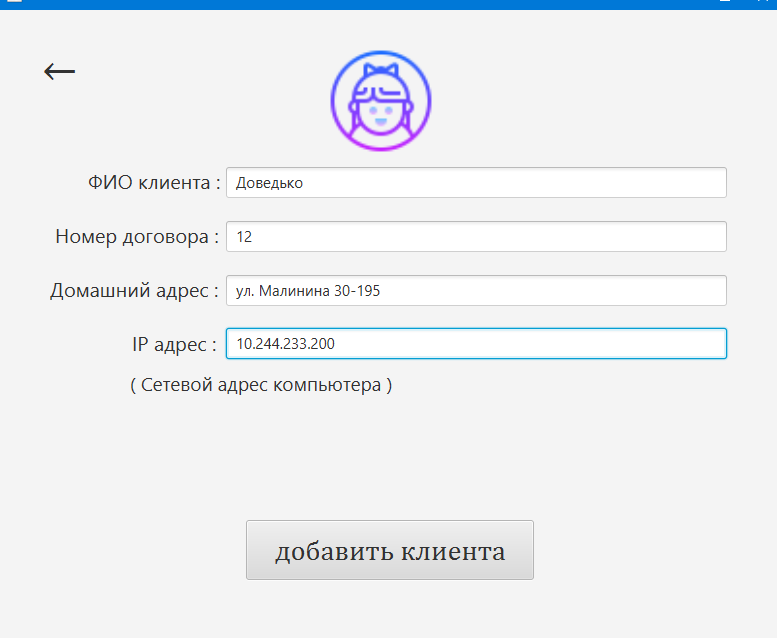


Рис. 5.7.1 Сохранение данных о клиенте в файл

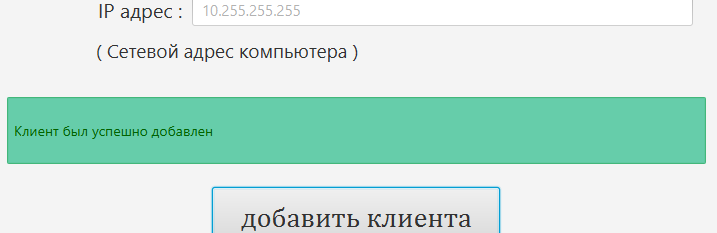


Рис. 5.7.2 Клиент добавлен

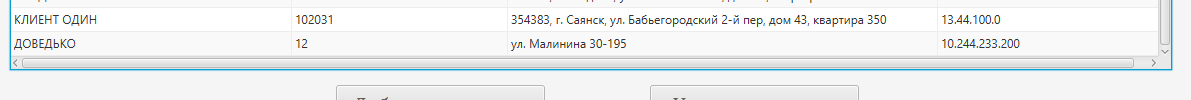
****

Рис. 5.7.3 Данные отображаются в таблице «Списка клиентов»

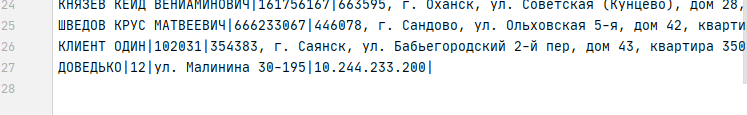
****

Рис. 5.7.4 Данные отображаются в файле

**Тест [8]- Рис. 5.8.(1-3)**

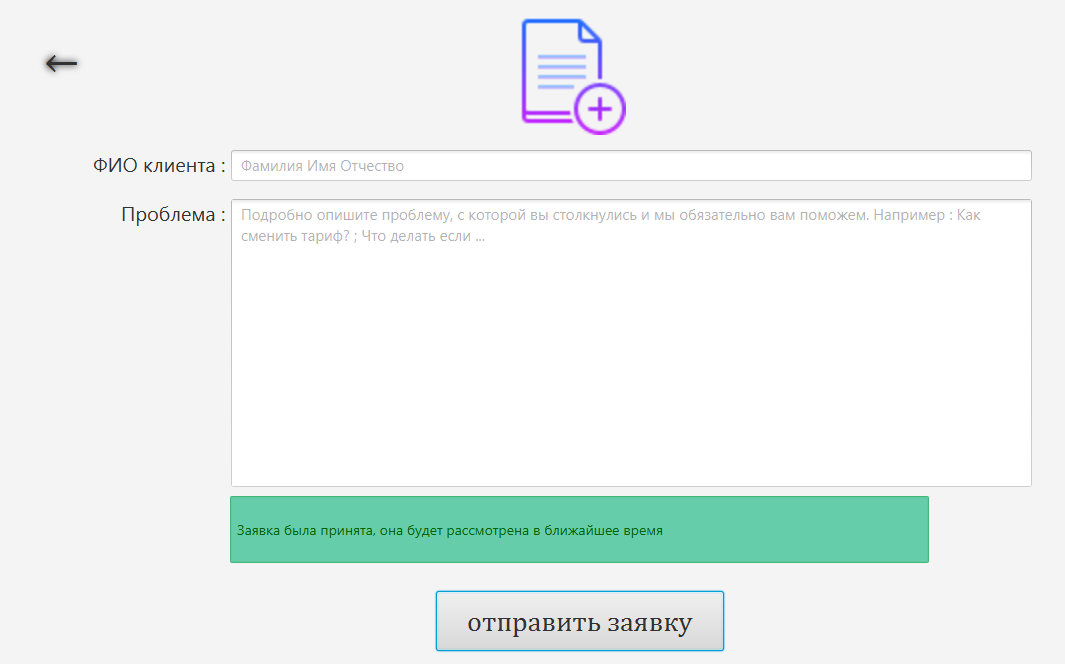


Рис. 5.8.1 Заявка добавлена

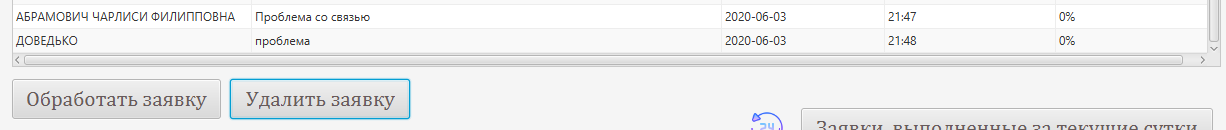
****

Рис. 5.8.2 Заявка добавлена в таблицу со всеми заявками

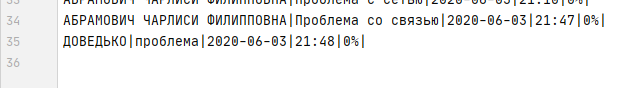
****

Рис. 5.8.3 Заявка добавлена в файл со всеми заявками

**Тест [9]- Рис. 5.9**

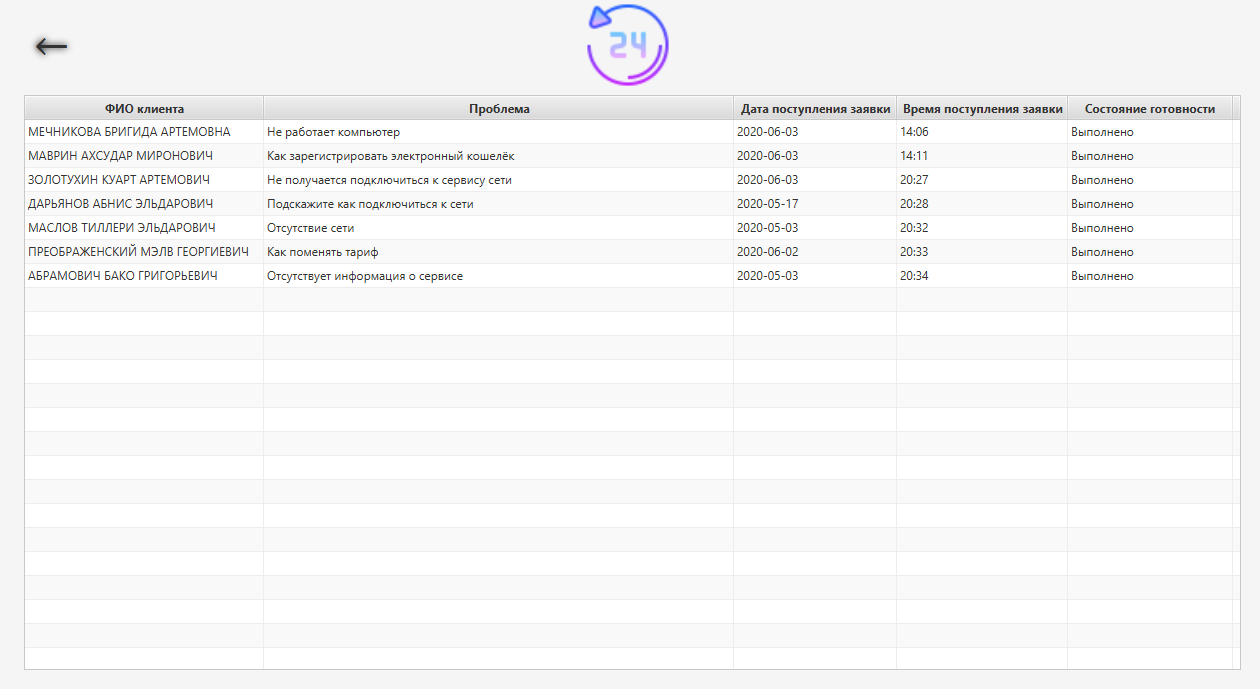
****

Рис. 5.9 Открытие окна заявок, выполненных за текущие сутки

**Тест [10]- Рис. 5.10**

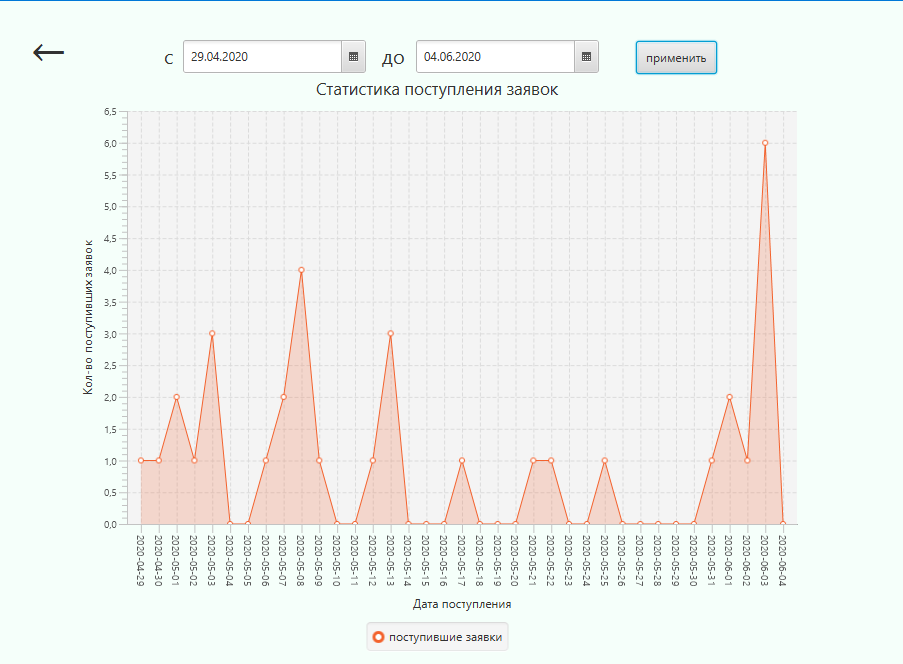
****

Рис. 5.10 Корректность отображаемой статистики поступления заявок

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ**
2. Порядок добавления нового абонента сети:

* Пользователю необходимо перейти на окно «Список клиентов»;
* Необходимо кликнуть кнопкой мыши на кнопку «Добавить клиента»;
* Необходимо заполнить все пустые поля корректными данными;
* После этого пользователю необходимо кликнуть кнопкой мыши на кнопку «Добавить клиента»;
* Нажав сначала на строку с клиентом, а затем кнопку «Удалить клиента», можно удалить клиента из таблицы;
* Если все данные заполнены корректно, то система выдаст оповещение об успешном добавлении клиента;
* Если в заполнении полей была допущена ошибка, на экран будет выведено сообщение об ошибке.

1. Порядок добавления новой заявки:

* Нужно перейти на окно «Новая заявка» из главного меню;
* Корректно заполнить поля заявки, а в особенности поле ФИО клиента, т.к. заявка будет добавлена только в том случае, если данный человек является клиентом сети;
* После этого пользователю необходимо кликнуть кнопкой мыши на кнопку «Добавить заявку»;
* Если все данные заполнены корректно, то система выдаст оповещение об успешном добавлении заявки;
* Если в заполнении полей была допущена ошибка, на экран будет выведено сообщение об ошибке.

1. Порядок просмотра списка заявок:

* Перейти на окно «Список заявок» и в данном окне отображается список всех заявок;
* Также в этом окне нажав на кнопку «Обработать заявку» пользователь может обработать заявку;
* Также, нажав сначала на строку с клиентом, а затем кнопку «Удалить клиента», можно удалить клиента из таблицы;
* Выбрать промежуток времени, за который необходимо вывести список поступивших заявок, можно с помощью полей DatePicker вверу окна и, после, нажав на кнопку «Применить». После этого откроется окно заявок, поступивших за выбранный диапазон времени;
* Нажав на кнопку «Заявки, выполненные за текущие сутки», можно просмотреть список заявок, выполненных соответственно за текущие сутки.

1. Порядок просмотра статистики поступивших заявок за выбранный диапазон времени:

* Статистика будет отображаться в виде 3 графиков (круговая диаграмма «Количество оценок», линейный график «Посещаемость» и гистограмма «Средний рейтинг»);
* Выбрать промежуток времени, за который необходимо вывести статистику поступивших заявок, можно с помощью полей DatePicker вверу окна и, после, нажав на кнопку «Применить»;
* После этого на экран будет выведена статистика заявок, поступивших за выбранный диапазон времени. Статистика будет отображаться в виде графика AreaChart.

1. Возможные ошибки:

* Если на вкладке «Добавить клиента» неверно заполнить любое из полей, то система сообщит об ошибке и попросит исправить данные;
* Если на вкладке «Новая заявка» неверно заполнить любое из полей, или ввести ФИО клиента, которого нет в базе, то система выведет сообщение об ошибке и попросит ввести исправить ошибку;
* Если на вкладке «Статистика поступивших заявок» в один из дней, за который вы хотите увидеть статистику, не будет оценок, то статистика за этот день будет отображаться на графике в виде точки напротив количества заявок равных нулю.
* Если на вкладке «Статистика поступления заявок» ввести диапазон времени в обратном порядке (первая дата в календаре находится после второй даты), то после нажатия на кнопку «Применить» на графике не будет отображаться статистика.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта **проведен анализ предметной области, исходя из которого разработано программное обеспечение для создания программного средство для службы технической поддержки абонентов сети.** Для этого был использован язык объектно-ориентированного программирования Java.

В начале работы были изучены прототипы данного программного обеспечения, проведен анализ поставленной задачи. В соответствии с ними сформулированы требования к программному обеспечению.

На этапе разработки программного средства были изучены варианты использования программного обеспечения и в соответствии с этим реализован основной функционал системы.

На следующем этапе было дано обоснование приемов программирования, используемых в построении системы, а также был до конца реализован функционал программного обеспечения, включающий интерфейс системы и его дизайн.

На этапе тестирования была испытана работа системы в различных условиях и на различных наборах данных.

Далее было составлено руководство пользователя, в котором было перечислено: способы использования системы для того, чтобы система работала правильно и корректно.

В результате была построена система с помощью языка Java согласно сформулированным на начальном этапе требованиям. В ней были использованы различные программные решения, что позволило нам детально спроектировать программу.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Лафоре Р. Структуры данных и алгоритмы Java. 2-е изд. — СПб.: Питер, 2013. —704 c.

[2] Шилдт Г. Java. Полное руководство – М.: Диалектика; СПб.: Альфа-книга,2018. — 1488 с.

[3] Официальная документация “Oracle” [Электронный ресурс]**.** – Электронные данные. – Режим доступа: https://docs.oracle.com/javase/7/docs/api/overview-summary.html

[4] ГОСТ 19.701–90 (ИСО 5807–85) [Текст]. – Единая система программной документации: Сб. ГОСТов. - М.: Стандартинформ, 2005 с.

[5] Шилдт Г. Swing. Руководство для начинающих. – ИД Вильямс, 2007. —

704 с.

[6] Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. — СПб.: Питер, 2017. —288 c.

[7] Блох Дж. Java. Эффективное программирование Москва: ДМК, 2013. —294 c.

[8] Хаггарти Р. Дискретная математика для программистов. Москва: Техносфера, 2003. —320 c.

[9] METANIT.COM[Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com

**ПРИЛОЖЕНИЕ**

**(обязательное)**

**Листинг кода**

**Класс Main**

package sample;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
*// extends это ключевое слово, предназначенное для расширения реализации какого-то  
// существующего класса. Создается новый класс на основе существующего, и этот новый  
// класс расширяет (extends) возможности старого.*public class Main extends Application {  
  
 *// необходимо переопределить абстрактный метод из класса Application* @Override  
 *// Нужно окно, которое мы хотим показать* public void start(Stage primaryStage) throws Exception{  
 *// мы можем изменить наш метод запуска для работы с FXML* Parent root = FXMLLoader.*load*(getClass().getResource("starterWindow\_1.fxml"));  
 primaryStage.setTitle("Программное средство для службы технической поддержки абонентов сети");  
 *//нельзя добавлять компоненты непосредственно в Stage (окно).  
 //нужна сцена (scene).  
 // создаём новую сцену* primaryStage.setScene(new Scene(root, 1280, 720));  
 *//Мы можем использовать эту компоненту stage.  
 //Единственная проблема в том, что она скрыта по умолчанию.  
 //К счастью, мы можем легко показать ее, используя метод primaryStage.show()* primaryStage.show();  
 }  
  
  
 public static void main(String[] args) {  
 *// Внутри метода main() приложение можно запустить используя метод:  
 launch*(args);  
 }  
}

**Класс NewClient**

package sample;  
  
import javafx.beans.property.SimpleStringProperty;  
  
*// класс клиентов*public class NewClient {  
  
 *// Property позволяет нам получать автоматические уведомления при любых изменениях переменных* private final SimpleStringProperty fio;  
 private final SimpleStringProperty agreementNumber;  
 private final SimpleStringProperty homeAddress;  
 private final SimpleStringProperty ipAddress;  
 *// ссылка на следующий элемент списка  
 // поле next содержит всего лишь ссылку на другой элемент списка,  
 // а не сам объект NewClient* public NewClient next;  
 *// ссылка представляет собой число,ассоциированное с объектом  
 // это число соответствует АДРЕСУ объекта в памяти компьютера  
  
 // Конструктор с четырьмя аргументами (по характеристикам объекта)* public NewClient(String fio, String agreementNumber, String homeAddress, String ipAddress) {  
 */\*  
 с помощью ключевого слова this мы обращаемся к полям объекта.  
 (требуется для того, чтобы метод мог сослаться на вызвавший его объект)  
 Если в методе нет своей переменной с таким же именем, this можно опустить.  
 \*/* this.fio = new SimpleStringProperty(fio);  
 this.agreementNumber = new SimpleStringProperty(agreementNumber);  
 this.homeAddress = new SimpleStringProperty(homeAddress);  
 this.ipAddress = new SimpleStringProperty(ipAddress);  
 }  
  
  
 *// конструктор по готовому элементу* public NewClient(NewClient students) {  
 this.fio = students.fio;  
 this.agreementNumber = students.agreementNumber;  
 this.homeAddress = students.homeAddress;  
 this.ipAddress = students.ipAddress;  
 *// значение null указывает на то, что ссылка не указывает ни на что  
 // до того, как элемент будет связан с другими элементами* next = null;  
 }  
  
 *// метод для получения следующего элемента* public NewClient getNext(){  
 return next;  
 }  
  
 *// метод для установки следующего элемента* public void setNext(NewClient students){  
 next = students;  
 }  
  
 public String getFio() {  
 return fio.get();  
 }  
  
 public SimpleStringProperty fioProperty() {  
 return fio;  
 }  
  
 public void setFio(String fio) {  
 this.fio.set(fio);  
 }  
  
 public String getAgreementNumber() {  
 return agreementNumber.get();  
 }  
  
 public SimpleStringProperty agreementNumberProperty() {  
 return agreementNumber;  
 }  
  
 public void setAgreementNumber(String agreementNumber) {  
 this.agreementNumber.set(agreementNumber);  
 }  
  
 public String getHomeAddress() {  
 return homeAddress.get();  
 }  
  
 public SimpleStringProperty homeAddressProperty() {  
 return homeAddress;  
 }  
  
 public void setHomeAddress(String homeAddress) {  
 this.homeAddress.set(homeAddress);  
 }  
  
 public String getIpAddress() {  
 return ipAddress.get();  
 }  
  
 public SimpleStringProperty ipAddressProperty() {  
 return ipAddress;  
 }  
  
 public void setIpAddress(String ipAddress) {  
 this.ipAddress.set(ipAddress);  
 }  
}

**Класс LinkListC**

package sample;  
  
*// связный список для объектов класса NewClient*public class LinkListC {  
  
 *//счётчик для количества элементов в нашем списке* public static int *size* = 0;  
 *// класс LinkedList содержит всего один элемент данных  
 // ссылку на первый элемент списка  
 // это единственная информации о местанахождении элементов, которая хранится в списке* private NewClient first;  
 *// остальные элементы отслеживаются по цепочке ссылок next, начиная с элемента first  
  
 //конструктор для создания списка* public LinkListC() {  
 *// конструктор инициализирует first значением null* first = null;  
 *// список пока не содержит элементов  
 // Если бы списке был хоть один элемент, в first хранилась бы ссылка на него.  
 // Метод isEmpty() использует этот факт для проверки отсутствия элементов в списке.* }  
  
 *//метод для получения первого элемента списка* public NewClient getFirst() {  
 return first;  
 }  
  
 *//метод для установки первого значения списка* public void setFirst(NewClient students) {  
 first = students;  
 }  
  
 *// Возвращает количество элементов в этом списке.* public int size() {  
 return *size*;  
 }  
  
 *//возвращает true, если список пуст* public boolean isEmpty() {  
 return first == null;  
 }  
  
  
 *// Вставляет указанный элемент в указанную позицию в этом списке.* public void add(int number, NewClient client) {  
  
 IteratorC iterator = NewClientController\_3.*iterator*;  
 boolean flag = true;  
 *// Сброс итератора и возврат к первому значению* iterator.reset();  
  
 *//если вставляемый элемент является первым* if (number == 0) {  
 *// то вставляем первым без проверки на следующий элемент* iterator.insertBefore(client);  
 } else {  
 *// обнуляем индекс* int i = 0;  
 *// проходим по всем элементам списка в поисках нужного по счёту элемента  
 // если флаг==false => мы дошли до конца списка* while ((flag) && (i != number - 1)) {  
 *// инкриментируем индекс вставки* i++;  
 try {  
 *// Перемещение итератора к следующему элементу* iterator.nextLink();  
 } catch (NullPointerException npe) {  
 *// если следующего элемента нет, то меняем значение флага* flag = false;  
 }  
 }  
 try {  
 *// если индекс равен последнеднему элементу, то мы вставляем элемент в конец* iterator.insertAfter(client);  
 } catch (NullPointerException npe) {  
 System.*out*.println("Не удалось добавить элемент");  
 }  
 }  
 }  
  
 *// добавление элемента в начало списка* public void addFirst(NewClient students) {  
  
 NewClientController\_3.*list*.add(0, students);  
 }  
  
 *// добавление элемента в конец списка* public void addLast(NewClient students) {  
 IteratorC iterator = NewClientController\_3.*iterator*;  
 boolean flag = true;  
 *//Сброс итератора и возврат к первому значению* iterator.reset();  
  
 *//если текущий элемент является первым* if (NewClientController\_3.*list*.isEmpty()) {  
 iterator.insertBefore(students);  
 } else {  
 int i = 0;  
 while (flag) {  
 i++;  
 try {  
 *// Перемещение итератора к следующему элементу* iterator.nextLink();  
 } catch (NullPointerException npe) {  
 flag = false;  
 }  
 }  
 NewClientController\_3.*list*.add(i - 1, students);  
 }  
 }  
  
 *// удаление элемента по индексу* public void remove(int number) {  
 IteratorC iterator = NewClientController\_3.*iterator*;  
 boolean flag = true;  
 iterator.reset();  
  
 if (number == 0) {  
 iterator.deleteCurrent();  
 } else {  
 int i = 0;  
 while ((flag) && (i != number)) {  
 i++;  
 try {  
 iterator.nextLink();  
 } catch (NullPointerException npe) {  
 flag = false;  
 }  
 }  
 try {  
 iterator.deleteCurrent();  
 } catch (NullPointerException npe) {  
 System.*out*.println("Не удалось удалить элемент");  
 }  
 }  
 }  
  
 *// получение элемента по индексу* public NewClient get(int number) {  
 IteratorC iterator = NewClientController\_3.*iterator*;  
 boolean flag = true;  
 iterator.reset();  
  
 int i = 0;  
 while ((flag) && (i != number)) {  
 i++;  
 try {  
 iterator.nextLink();  
 } catch (NullPointerException npe) {  
 flag = false;  
 }  
 }  
 try {  
 return (NewClient) iterator.getCurrent();  
 } catch (NullPointerException npe) {  
 System.*out*.println("Не удалось получить элемент");  
 return null;  
 }  
  
 }  
  
 *// установить значение по индексу* public void set(int number, NewClient students) {  
 NewClientController\_3.*list*.remove(number);  
 NewClientController\_3.*list*.add(number, students);  
 }  
  
}

**Класс IteratorC**

package sample;  
  
*// Объекты, содержащие ссылки на элементы структур данных и используемые  
// для перебора элементов этих структур, обычно называются итераторами*public class IteratorC {  
 *// current ссылается на текущий элемент списка  
 // сначала он содержит first (тоесть ссылку на первый элемент списка)* private NewClient current;  
 *// Предыдущий элемент списка* private NewClient previous;  
 *// Связный список* private LinkListC ourList;  
  
 *//Конструктор для создания итератора* public IteratorC(LinkListC list) {  
 ourList = list;  
 reset();  
 }  
  
 *//Сброс итератора и возврат к первому значению* public void reset() {  
 *// current --> first* current = ourList.getFirst();  
 *// previous --> null* previous = null;  
 }  
  
 *// Переход к следующему элементу* public void nextLink() {  
 *// Присваивание текущего элемента previous* previous = current;  
 *// Присваивание текущему элементу ссылку на следующий элемент (next)* current = current.getNext();  
 }  
  
 *//Получение текущего элемента* public NewClient getCurrent() {  
 return current;  
 }  
  
 *//Вставка перед текущим элементом* public void insertBefore(NewClient client) {  
 *//для методов вставки в список* NewClient newElement = new NewClient(client);  
 *//если текущий элемент является первым* if (previous == null) {  
 *// тогда новому элементу переопределяем указатели  
 // его поле next указывает на первый элемент списка* newElement.setNext(ourList.getFirst());  
 *// а в списке поле first указывает на текущий элемент* ourList.setFirst(newElement);  
 reset();  
 *//если текущий элемент не первый* } else {  
 *// поле next вставялемого элемента указывает на текущий элемент* newElement.setNext(previous.getNext());  
 *// поле next элемента перед текущим указывает на вставляемый элемент* previous.setNext(newElement);  
 *// текущим становится вставляемый элемент* current = newElement;  
 }  
 *//увеличение размера списка* LinkListC.*size*++;  
 }  
  
 *//Вставка после текущего элемента* public void insertAfter(NewClient clients) {  
 *//для методов вставки в LinkList* NewClient newElement = new NewClient(clients);  
 *//Если список пуст* if (ourList.isEmpty()) {  
 *// то вставляемый элемент устанавливаем первым* ourList.setFirst(newElement);  
 *// а текущим ставим вставляемый элемент* this.current = newElement;  
 *//Если список не пуст* } else {  
 *// поле next вставялемого элемента указывает на следующий после текущего элемент* newElement.setNext(this.current.getNext());  
 *// поле next текущего элемента указывает на вставляемый элемент* this.current.setNext(newElement);  
 nextLink();  
 }  
 *//увеличение размера списка* LinkListC.*size*++;  
 }  
  
 *//определяет, является ли текущий элемент последним в списке* public boolean atEnd() {  
 *//проверка, является ли элемент последним в списке* return (current.getNext() == null);  
 }  
  
 *//Удаление текущего элемента* public void deleteCurrent() {  
 *//Если текущий элемент является первым* if (previous == null) {  
 ourList.setFirst(current.getNext());  
 reset();  
 *//Если текущий элемент не первый* } else {  
 previous.setNext(current.getNext());  
 if (atEnd())  
 *//Сброс, если удалили последний* reset();  
 else  
 current = current.getNext();  
 }  
 *//уменьшение размера* LinkListC.*size*--;  
 }  
  
}

**Класс StarterWindowController\_1**

package sample;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.effect.DropShadow;  
import javafx.scene.input.MouseEvent;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.net.URL;  
import java.util.ResourceBundle;  
  
public class StarterWindowController\_1 {  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Button list\_of\_clients;  
  
 @FXML  
 private Button new\_request;  
  
 @FXML  
 private Button list\_of\_requests;  
  
 @FXML  
 private Button charts;  
  
 @FXML  
 void initialize() {  
 *// тени, появляющиеся при наведении курсора на кнопки  
 shadow*(list\_of\_clients);  
 *shadow*(new\_request);  
 *shadow*(list\_of\_requests);  
 *shadow*(charts);  
  
 *// создаём отслеживание событий,для каждого пункта меню  
  
 // создаём stage для новых Scene* Stage stage = new Stage();  
 *// выделяем память и в качестве параметра ничего не передаём* FXMLLoader loader = new FXMLLoader();  
  
 *// кнопка "список клиентов"* list\_of\_clients.setOnAction((ActionEvent event) -> {  
 *// закроем окно* list\_of\_clients.getScene().getWindow().hide();  
 *// теперь отображаем нужное нам окно  
 // указываем место расположения нужного нам файла для дальнейшей его загрузки* loader.setLocation(getClass().getResource("listOfAllClients\_2.fxml"));  
 *// загружаем файл для дальнейшего отображения* try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 *// путь к файлу который необхоодимо загрузить* Parent root = loader.getRoot();  
 *// указываем то окошко которое нам необходимо загрузить* stage.setScene(new Scene(root));  
 *// показать и подождать того момента когда само наше окошко отобразиться* stage.show();  
 });  
  
 *// аналогично с остальными кнопками  
  
 // кнопка "новая заявка"* new\_request.setOnAction((ActionEvent event) -> {  
 new\_request.getScene().getWindow().hide();  
 loader.setLocation(getClass().getResource("newRequest\_4.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
 *// кнопка "список заявок"* list\_of\_requests.setOnAction((ActionEvent event) -> {  
 list\_of\_requests.getScene().getWindow().hide();  
 loader.setLocation(getClass().getResource("listOfAllRequests\_5.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
 *// кнопка "статистика"* charts.setOnAction((ActionEvent event) -> {  
 *// закроем окно* charts.getScene().getWindow().hide();  
 *// теперь отображаем нужное нам окно  
 // указываем место расположения нужного нам файла для дальнейшей его загрузки* loader.setLocation(getClass().getResource("chartsForAllRequests\_10.fxml"));  
 *// загружаем файл для дальнейшего отображения* try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 *// путь к файлу который необхоодимо загрузить* Parent root = loader.getRoot();  
 *// указываем то окошко которое нам необходимо загрузить* stage.setScene(new Scene(root));  
 *// показать и подождать того момента когда само наше окошко отобразиться* stage.show();  
 });  
 }  
 *//метод добавляет тени кнопкам* public static void shadow(Button shadoww) {  
 *// Adding the shadow when the mouse cursor is on* DropShadow shadow = new DropShadow();  
 shadoww.addEventHandler(MouseEvent.*MOUSE\_ENTERED*, e -> shadoww.setEffect(shadow));  
 *// Removing the shadow when the mouse cursor is off* shadoww.addEventHandler(MouseEvent.*MOUSE\_EXITED*, e -> shadoww.setEffect(null));  
 }  
}

**Класс ListOfAllClientsController\_2**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.ResourceBundle;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
*// отобразим в нашей таблице некоторые данные.  
// Для этого необходимо создать класс-контроллер для представления listOfAllClients\_2.fxml*public class ListOfAllClientsController\_2 {  
  
 private static int *counter* = 0;  
  
 */\*  
 для записи данных из списка в таблицу используем ObservableList  
 Мы работаем с классами-представлениями JavaFX, которые необходимо информировать при любых изменениях  
 в списке адресатов. Это важно, потому что, не будь этого, мы бы не смогли синхронизировать  
 представление данных с самими данными  
 \*/* private static ObservableList<NewClient> *tableData* = FXCollections.*observableArrayList*();  
  
 public static ObservableList<NewClient> getTableData() {  
 return *tableData*;  
 }  
  
 public static void setTableData(ObservableList<NewClient> tableData) {  
 ListOfAllClientsController\_2.*tableData* = tableData;  
 }  
  
 */\*  
 Для того, чтобы получить доступ к таблице и меткам представления, мы определим некоторые переменные.  
 Эти переменные и некоторые методы имеют специальную аннотацию @FXML.  
 Она необходима для того, чтобы fxml-файл имел доступ к приватным полям и методам.  
 \*/* @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private TableView<NewClient> table;  
  
 @FXML  
 private TableColumn<NewClient, String> fioOfTheClient;  
  
 @FXML  
 private TableColumn<NewClient, String> agreementNumber;  
  
 @FXML  
 private TableColumn<NewClient, String> address;  
  
 @FXML  
 private TableColumn<NewClient, String> ipAddress;  
  
 @FXML  
 private Button deleteClient;  
  
 @FXML  
 private Button addClient;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 void initialize() {  
 *// тени, появляющиеся при наведении курсора на кнопки* StarterWindowController\_1.*shadow*(addClient);  
 StarterWindowController\_1.*shadow*(deleteClient);  
 StarterWindowController\_1.*shadow*(back);  
  
 *// кнопка"назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 *// создаём обЪект класса FXMLLoader* FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("starterWindow\_1.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
  
 *// сохраняем наши значения в файл ("перезаписываем" файл)  
 saveToFile*(*tableData*);  
 });  
  
  
 *//таблица  
 // если не первый с запуска программы раз заходим в окно* if (*counter* != 0) {  
 *// то очищаем ObservableList  
 tableData*.clear();  
 }  
 *counter*++;  
 *// делаем нашу табличку редактируемой* table.setEditable(true);  
 */\* специфицируем, так сказть, колонки таблицы  
 Use the setCellFactory method to reimplement the table cell as a text field with  
 the help of the TextFieldTableCell class.  
 \*/* fioOfTheClient.setCellFactory(TextFieldTableCell.*forTableColumn*());*//как текстовое поле* fioOfTheClient.setCellValueFactory(cellData -> cellData.getValue().fioProperty());  
  
 agreementNumber.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 agreementNumber.setCellValueFactory(cellData -> cellData.getValue().agreementNumberProperty());  
  
 address.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 address.setCellValueFactory(cellData -> cellData.getValue().homeAddressProperty());  
  
 ipAddress.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 ipAddress.setCellValueFactory(cellData -> cellData.getValue().ipAddressProperty());  
  
  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> clientList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("ClientsList.txt"))) {  
 *// convert it into a List* clientList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < clientList.size(); i++) {  
 String[] separated = clientList.get(i).split("\\|");  
 String fioOfTheClient1 = separated[0].trim();  
 String agreementNumber1 = separated[1].trim();  
 String address1 = separated[2].trim();  
 String ipAddress1 = separated[3].trim();  
 *tableData*.add(new NewClient(fioOfTheClient1, agreementNumber1, address1, ipAddress1));  
 }  
  
  
 */\* The setOnEditCommit method processes editing and assigns the updated value to the corresponding  
 table cell.(обрабатываем изменение полей таблички)  
 \*/* fioOfTheClient.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setFio(event.getNewValue()));  
 agreementNumber.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setAgreementNumber(event.getNewValue()));  
 address.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setHomeAddress(event.getNewValue()));  
 ipAddress.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setIpAddress(event.getNewValue()));  
  
  
 *// кнопка"удалить клиента"* deleteClient.setOnAction((ActionEvent) -> {  
 *// удаляем выбранного клиента* NewClient p = table.getSelectionModel().getSelectedItem();  
 *tableData*.remove(p);  
 });  
  
 *// загружаем данные из ObservableList в таблицу* table.setItems(*tableData*);  
  
  
 *// кнопка "добавить клиента"* addClient.setOnAction((ActionEvent event) -> {  
 *// создаём stage для новых Scene* Stage stage = new Stage();  
 *// выделяем память и в качестве параметра ничего не передаём* FXMLLoader loader = new FXMLLoader();  
 *// закроем окно* addClient.getScene().getWindow().hide();  
 *// теперь отображаем нужное нам окно  
 // указываем место расположения нужного нам файла для дальнейшей его загрузки* loader.setLocation(getClass().getResource("newClient\_3.fxml"));  
 *// загружаем файл для дальнейшего отображения* try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 *// путь к файлу который необхоодимо загрузить* Parent root = loader.getRoot();  
 *// указываем то окошко которое нам необходимо загрузить* stage.setScene(new Scene(root));  
 *// показать и подождать того момента когда само наше окошко отобразиться* stage.show();  
  
 *// сохраняем наши значения в файл ("перезаписываем" файл)  
 saveToFile*(*tableData*);  
 });  
  
 }  
  
  
 *// записываем введённые значения в файл ClientsList.txt* public static void saveToFile(ObservableList<NewClient> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("ClientsList.txt");  
 StringBuilder sb = new StringBuilder();  
 sb.append("ФИО|номер договора|домашний адрес|ip адрес|\n");  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String number0fTheAgreement = list.get(i + 1).getAgreementNumber();  
 String homeAddress = list.get(i + 1).getHomeAddress();  
 String IPAddress = list.get(i + 1).getIpAddress();  
 sb.append(Fio);  
 sb.append("|");  
 sb.append(number0fTheAgreement);  
 sb.append("|");  
 sb.append(homeAddress);  
 sb.append("|");  
 sb.append(IPAddress);  
 sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
}

**Класс NewClientController\_3**

package sample;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.ResourceBundle;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class NewClientController\_3 {  
  
 *// создание двусвязного списка (объекта класса LinkListC)  
 // однако поле list содержит не объект,а только ссылку на него.  
 // Объект находится где-то в другом месте памяти* public static LinkListC *list* = new LinkListC();  
  
 *// объект-итератор, ассоциированный с этим списком* public static IteratorC *iterator* = new IteratorC(*list*);  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private TextField fio;  
 public String fio1;  
  
 @FXML  
 private TextField IPAddress;  
 public String IPAddress1;  
  
 @FXML  
 private TextField homeAddress;  
 public String homeAddress1;  
  
 @FXML  
 private TextField number0fTheAgreement;  
 public String number0fTheAgreement1;  
  
 @FXML  
 private Button addNewClient;  
  
 @FXML  
 private Rectangle rectangle;  
  
 @FXML  
 private Label label;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 void initialize() {  
 StarterWindowController\_1.*shadow*(addNewClient);  
 StarterWindowController\_1.*shadow*(back);  
  
  
 *// создаём stage для новых Scene* Stage stage = new Stage();  
 *// создаём лбЪект класса FXMLLoader  
 // выделяем память и в качестве параметра ничего не передаём* FXMLLoader loader = new FXMLLoader();  
  
  
 *// кнопка "назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 loader.setLocation(getClass().getResource("listOfAllClients\_2.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
  
 *// очищаем поле лейбла который выводит ошибки ввода* label.setText("");  
 *// задаём рамке для ошибок "прозрачный" цвет* rectangle.setStroke(Color.*TRANSPARENT*);  
 *// задаём прямоугольнику для ошибок "прозрачный" цвет* rectangle.setFill(Color.*TRANSPARENT*);  
  
  
 *// обрабатываем нажатие кнопки "добавить клиента"* addNewClient.setOnAction((ActionEvent event) -> {  
boolean check = false;  
  
 *// вводим переммные correct, для выявления ввода неверных символов* boolean correctIP = true;  
 boolean correctNumber0fTheAgreement = true;  
 boolean correctFio = true;  
 boolean correctAddress = true;  
 *// создаём переммную notEmpty, для выявления наличия незаполненных полей* boolean empty = false;  
 *// переменная для проверки уже существующего клиента* boolean repeat = false;  
  
  
 *// выполняем проверку на корректность вводимых данных  
 // считываем данные с каждого поля* number0fTheAgreement1 = number0fTheAgreement.getText().trim();  
 IPAddress1 = IPAddress.getText().trim();  
 fio1 = fio.getText().trim().toUpperCase();  
 homeAddress1 = homeAddress.getText().trim();  
  
 *// проверяем пустое ли какое-нибудь поле* if (fio1.isEmpty() || IPAddress1.isEmpty()  
 || homeAddress1.isEmpty() || number0fTheAgreement1.isEmpty()) {  
 *// если да, то изменяем значение переменной Empty* empty = true;  
 }  
  
 *// IPAddress должен быть только целыми неотрицательными числами, допускается точка в записи* for (int i = 0; i < IPAddress1.length(); i++) {  
 *// числа в диапазоне 48-57, код точки 46* if (!((int) IPAddress1.charAt(i) >= 48 && (int) IPAddress1.charAt(i) <= 57)  
 && ((int) IPAddress1.charAt(i) != 46)) {  
 *// изменяем значение флаговой переменной если в поле кол-ва товара были ввведены символы кроме цифр* correctIP = false;  
 break;  
 }  
 }  
  
 *// number0fTheAgreement должен быть только целыми неотрицательными числами* for (int i = 0; i < number0fTheAgreement1.length(); i++) {  
 *// числа в диапазоне 48-57* if (!((int) number0fTheAgreement1.charAt(i) >= 48 && (int) number0fTheAgreement1.charAt(i) <= 57)) {  
 *// изменяем значение флаговой переменной если в поле кол-ва товара были ввведены символы кроме цифр* correctNumber0fTheAgreement = false;  
 break;  
 }  
 }  
  
  
 *// делаем проверку на корректность фио* for (int i = 0; i < fio1.length(); i++) {  
 *// кириллица-диапазон : 1040-1103  
 // пробел с кодом 32  
 // "-" с кодом 45* if (!((int) fio1.charAt(i) >= 1040 && (int) fio1.charAt(i) <= 1103) && ((int) fio1.charAt(i) != 32)  
 && ((int) fio1.charAt(i) != 45)) {  
 *// изменяем значение флаговой переменной если введён подходящий символ* correctFio = false;  
 break;  
 }  
 }  
  
  
  
 *// проверяем есть ли уже такой клиент в списке сот сети  
 // считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> listString = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("ClientsList.txt"))) {  
 *// convert it into a List* listString = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < listString.size(); i++) {  
 String[] separated = listString.get(i).split("\\|");  
 String agreementNumber1 = separated[1].trim();  
  
 if (number0fTheAgreement1.equals(agreementNumber1)) {  
 *// если такая завка уже есть то мы не добавялем эту заявку в список готовых  
 // а просто обновляем* repeat = true;  
 break;  
 }  
 }  
  
 *// проверяем правильность введённого ip адреса (ipv4)  
 // записывается в виде 4-ёх десятичных чисел (от 0 до 255), разделённых точками  
 validIP*(IPAddress1);  
  
 if (repeat) {  
 *// задаём рамке для ошибок "красный" цвет* rectangle.setStroke(Color.*RED*);  
 *// задаём прямоугольнику для ошибок "красный" цвет* rectangle.setFill(Color.*LIGHTPINK*);  
  
 *// иначе, если поля неверно были заполнены, очищаем поле лейбла* label.setText("");  
 *// выводим в пустой лейбел текст об ошибке* label.setText("Договор с данным номером уже был заключён ");  
  
 } else {  
  
 *// записываем нового клиента только если все поля заполнены и притом правильно  
 // а также, если такого клиента ещё нет в списке* if (!empty && correctIP && correctFio && correctAddress && correctNumber0fTheAgreement && *validIP*(IPAddress1)) {  
 *// задаём рамке для ошибок "прозрачный" цвет* rectangle.setStroke(Color.*MEDIUMSEAGREEN*);  
 *// задаём прямоугольнику для ошибок "прозрачный" цвет* rectangle.setFill(Color.*MEDIUMAQUAMARINE*);  
  
 *// записываем новый элемент в список для записи в файл  
 list*.addLast(new NewClient(fio1, number0fTheAgreement1, homeAddress1, IPAddress1));  
 *// записываем введённые значения в файл (добавляем новый эл-т)  
 saveToFile*(*list*);  
  
 *// очищаем поля* number0fTheAgreement.clear();  
 fio.clear();  
 homeAddress.clear();  
 IPAddress.clear();  
  
 *// если поля верно были заполнены, очищаем поле лейбла* label.setText("");  
 label.setTextFill(Color.*DARKGREEN*);  
 *// выводим в пустой лейбел текст* label.setText("Клиент был успешно добавлен ");  
 check = true;  
 } else {  
 *// задаём рамке для ошибок "красный" цвет* rectangle.setStroke(Color.*RED*);  
 *// задаём прямоугольнику для ошибок "красный" цвет* rectangle.setFill(Color.*LIGHTPINK*);  
  
 *// иначе, если поля неверно были заполнены, очищаем поле лейбла* label.setText("");  
 label.setTextFill(Color.*FIREBRICK*);  
 *// выводим в пустой лейбел текст об ошибке* label.setText("Пожалуйста проверьте корректность введённых данных, также проверьте все ли поля заполнены ");  
 }  
 }  
 if(check){  
 *// теперь очистим список чтобы при записи в файл нового клиента,не записывались сразу все имеющиеся в списке  
 list*.remove(0);  
 }  
 });  
  
  
 }  
  
 *// записываем введённые значения в файл ClientsList.txt* public static void saveToFile(LinkListC list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("ClientsList.txt", true);  
 StringBuilder sb = new StringBuilder();  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String number0fTheAgreement = list.get(i + 1).getAgreementNumber();  
 String homeAddress = list.get(i + 1).getHomeAddress();  
 String IPAddress = list.get(i + 1).getIpAddress();  
 sb.append(Fio);  
 sb.append("|");  
 sb.append(number0fTheAgreement);  
 sb.append("|");  
 sb.append(homeAddress);  
 sb.append("|");  
 sb.append(IPAddress);  
 sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
  
 *// проверка валидности введённого ip* public static boolean validIP(String ip) {  
 try {  
 if (ip == null || ip.isEmpty()) {  
 return false;  
 }  
  
 String[] parts = ip.split("\\.");  
 if (parts.length != 4) {  
 return false;  
 }  
  
 for (String s : parts) {  
 int i = Integer.*parseInt*(s);  
 if ((i < 0) || (i > 255)) {  
 return false;  
 }  
 }  
 if (ip.endsWith(".")) {  
 return false;  
 }  
  
 return true;  
 } catch (NumberFormatException nfe) {  
 return false;  
 }  
 }  
  
}

**Класс NewRequestController\_4**

package sample;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextArea;  
import javafx.scene.control.TextField;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class NewRequestController\_4 {  
  
 *// создание двусвязного списка* public static LinkListR *list* = new LinkListR();  
 *// объект-итератор, ассоциированный с этим списком* public static IteratorR *iterator* = new IteratorR(*list*);  
  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private TextField fio;  
 public String fio1;  
  
 @FXML  
 private TextArea problem;  
 public String problem1;  
  
 @FXML  
 private Button addNewRequest;  
  
 @FXML  
 private Rectangle rectangle;  
  
 @FXML  
 private Label label;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 void initialize() {  
 StarterWindowController\_1.*shadow*(addNewRequest);  
 StarterWindowController\_1.*shadow*(back);  
  
 *// создаём stage для новых Scene* Stage stage = new Stage();  
 *// создаём лбЪект класса FXMLLoader  
 // выделяем память и в качестве параметра ничего не передаём* FXMLLoader loader = new FXMLLoader();  
  
  
 *// кнопка "назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 loader.setLocation(getClass().getResource("starterWindow\_1.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
 *// очищаем поле лейбла который выводит ошибки ввода* label.setText("");  
 *// задаём рамке для ошибок "прозрачный" цвет* rectangle.setStroke(Color.*TRANSPARENT*);  
 *// задаём прямоугольнику для ошибок "прозрачный" цвет* rectangle.setFill(Color.*TRANSPARENT*);  
  
  
 *// обрабатываем нажатие кнопки "добавить заявку"* addNewRequest.setOnAction((ActionEvent event) -> {  
 boolean check = false;  
 *// вводим переменую correct, для выявления ввода неверных символов* boolean correctFio = true;  
 boolean correctRequest = true;  
 *// создаём переммную notEmpty, для выявления наличия незаполненных полей* boolean empty = false;  
 *// вводим переменую existingFio, для выявления ввода неверных символов* boolean existingFio = false;  
  
  
 *// выполняем проверку на корректность вводимых данных  
 // считываем данные с каждого поля* fio1 = fio.getText().trim().toUpperCase();  
 problem1 = problem.getText().trim();  
  
 *// проверяем пустое ли какое-нибудь поле* if (fio1.isEmpty() || problem1.isEmpty()) {  
 *// если да, то изменяем значение переменной Empty* empty = true;  
 }  
  
 *// делаем проверку на корректность фио* for (int i = 0; i < fio1.length(); i++) {  
 *// кириллица-диапазон : 1040-1103  
 // пробел с кодом 32* if (!((int) fio1.charAt(i) >= 1040 && (int) fio1.charAt(i) <= 1103) && ((int) fio1.charAt(i) != 32) ) {  
 *// изменяем значение флаговой переменной если введён подходящий символ* correctFio = false;  
 break;  
 }  
 }  
  
 *// делаем проверку на отсутствие символа "|"* for (int i = 0; i < problem1.length(); i++) {  
 *// "|" с кодом 124* if ((int) problem1.charAt(i) == 124) {  
 *// изменяем значение флаговой переменной если введён неподходящий символ* correctRequest = false;  
 break;  
 }  
 }  
  
 *// автоматически вводится дата в поле date при добавлении заявки* Date now = new Date();  
 DateFormat df = new SimpleDateFormat("yyyy-MM-dd");  
 DateFormat tf = new SimpleDateFormat("HH:mm");  
 String dateString = df.format(now);  
 String timeString = tf.format(now);  
  
 *// делаем проверку имеется ли фио клиента в списке клиентов  
 // считываем каждую новую строку в массив, а потом проверяем эл-ты* List<String> clientList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("ClientsList.txt"))) {  
 *// convert it into a List* clientList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < clientList.size(); i++) {  
 String[] separated = clientList.get(i).split("\\|");  
 String fioOfTheClient1 = separated[0].trim();  
 if (fioOfTheClient1.equals(fio1)) {  
 existingFio = true;  
 break;  
 }  
 }  
  
  
 *// записываем нового клиента только если все поля заполнены и притом правильно* if (!empty && correctFio && existingFio && correctRequest) {  
 *// задаём рамке для ошибок цвет* rectangle.setStroke(Color.*MEDIUMSEAGREEN*);  
 *// задаём прямоугольнику для ошибок цвет* rectangle.setFill(Color.*MEDIUMAQUAMARINE*);  
  
 *// записываем новый элемент в список  
 list*.addLast(new NewRequest( fio1, problem1, dateString, timeString, "0%"));  
 *// записываем введённые значения в файл (добавляем новый эл-т)  
 saveToFile*(*list*);  
 *// очищаем поля* fio.clear();  
 problem.clear();  
  
 *// если поля верно были заполнены, очищаем поле лейбла* label.setText("");  
 label.setTextFill(Color.*DARKGREEN*);  
 *// выводим в пустой лейбел текст* label.setText("Заявка была принята, она будет рассмотрена в ближайшее время ");  
 check = true;  
 }else {  
 *// задаём рамке для ошибок "красный" цвет* rectangle.setStroke(Color.*RED*);  
 *// задаём прямоугольнику для ошибок "красный" цвет* rectangle.setFill(Color.*LIGHTPINK*);  
  
 *// иначе, если поля неверно были заполнены, очищаем поле лейбла* label.setText("");  
 *// выводим в пустой лейбел текст об ошибке* label.setText("Пожалуйста проверьте корректность введённых данных, также проверьте все ли поля заполнены ");  
 if (!existingFio){  
 *// иначе, если поля fio нет в списке, очищаем поле лейбла* label.setText("");  
 *// выводим в пустой лейбел текст об ошибке* label.setText( fio1 + " не является клиентом данной сотовой сети ");  
 }  
 }  
 if(check){  
 *// теперь очистим список чтобы при записи в файл нового клиента,не записывались сразу все имеющиеся в списке  
 list*.remove(0);  
 }  
 });  
  
  
 }  
  
 *// записываем введённые значения в файл2* public static void saveToFile(LinkListR list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("RequestsList.txt", true);  
 StringBuilder sb = new StringBuilder();  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String problem = list.get(i + 1).getProblem();  
 String date = list.get(i + 1).getDate();  
 String time = list.get(i + 1).getTime();  
 sb.append(Fio);sb.append("|");  
 sb.append(problem);sb.append("|");  
 sb.append(date);sb.append("|");  
 sb.append(time);sb.append("|");  
 sb.append("0%");sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
}

**Класс ListOfAllRequestsController\_5**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.DatePicker;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.ResourceBundle;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class ListOfAllRequestsController\_5 {  
  
 private static int *counter* = 0;  
  
 *// поля для обработки заявки* public static String *ourFio* = "";  
 public static String *ourComplain* = "";  
  
 *// поля для выбора даты* public static LocalDate *ourPeriod1* ;  
 public static LocalDate *ourPeriod2* ;  
  
 private static ObservableList<NewRequest> *dataFromTheTable* = FXCollections.*observableArrayList*();  
  
 public static ObservableList<NewRequest> getTableData() {  
 return *dataFromTheTable*;  
 }  
  
 public static void setTableData(ObservableList<NewRequest> tableData) {  
 ListOfAllRequestsController\_5.*dataFromTheTable* = tableData;  
 }  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Button deleteTheRequest;  
  
 @FXML  
 private Button handleTheRequest;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 private TableView<NewRequest> table1;  
  
 @FXML  
 private TableColumn<NewRequest, String> fioOfTheClient1;  
  
 @FXML  
 private TableColumn<NewRequest, String> problem1;  
  
 @FXML  
 private TableColumn<NewRequest, String> date1;  
  
 @FXML  
 private TableColumn<NewRequest, String> time1;  
  
 @FXML  
 private TableColumn<NewRequest, String> ready1;  
  
 @FXML  
 public DatePicker period1;  
  
 @FXML  
 public DatePicker period2;  
  
 @FXML  
 private Button readyTodayRequests;  
  
 @FXML  
 private Button apply;  
  
  
 @FXML  
 void initialize() {  
 *// тени, появляющиеся при наведении курсора на кнопки* StarterWindowController\_1.*shadow*(deleteTheRequest);  
 StarterWindowController\_1.*shadow*(handleTheRequest);  
 StarterWindowController\_1.*shadow*(apply);  
 StarterWindowController\_1.*shadow*(readyTodayRequests);  
 StarterWindowController\_1.*shadow*(back);  
  
  
 *// кнопка"назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("starterWindow\_1.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
  
 *// "перезаписываем" файл  
 saveToFile*(*dataFromTheTable*);  
 });  
  
  
  
  
 *// кнопка "readyTodayRequests"* readyTodayRequests.setOnAction((ActionEvent event) -> {  
 *// создаём stage для новых Scene* Stage stage = new Stage();  
 *// выделяем память и в качестве параметра ничего не передаём* FXMLLoader loader = new FXMLLoader();  
 *// закроем окно* readyTodayRequests.getScene().getWindow().hide();  
 *// теперь отображаем нужное нам окно  
 // указываем место расположения нужного нам файла для дальнейшей его загрузки* loader.setLocation(getClass().getResource("readyTodayRequests\_7.fxml"));  
 *// загружаем файл для дальнейшего отображения* try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 *// путь к файлу который необхоодимо загрузить* Parent root = loader.getRoot();  
 *// указываем то окошко которое нам необходимо загрузить* stage.setScene(new Scene(root));  
 *// показать и подождать того момента когда само наше окошко отобразиться* stage.show();  
  
 *// "перезаписываем" файл  
 saveToFile*(*dataFromTheTable*);  
 });  
  
  
 *//таблица 1 (изначально выводим список заявок за всё время)  
  
 // если не первый с запуска программы раз заходим в окно* if (*counter* != 0) {  
 *// то очищаем ObservableList  
 dataFromTheTable*.clear();  
 }  
 *counter*++;  
 *// делаем нашу табличку редактируемой* table1.setEditable(true);  
 *//специфицируем колонки таблицы* fioOfTheClient1.setCellFactory(TextFieldTableCell.*forTableColumn*());*//как текстовое поле* fioOfTheClient1.setCellValueFactory(cellData -> cellData.getValue().fioProperty());  
  
 problem1.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 problem1.setCellValueFactory(cellData -> cellData.getValue().problemProperty());  
  
 date1.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 date1.setCellValueFactory(cellData -> cellData.getValue().dateProperty());  
  
 time1.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 time1.setCellValueFactory(cellData -> cellData.getValue().timeProperty());  
  
 ready1.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 ready1.setCellValueFactory(cellData -> cellData.getValue().readyProperty());  
  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> clientList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("RequestsList.txt"))) {  
 *// convert it into a List* clientList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < clientList.size(); i++) {  
 String[] separated = clientList.get(i).split("\\|");  
 String fio1 = separated[0].trim();  
 String problem1 = separated[1].trim();  
 String dateString = separated[2].trim();  
 String timeString = separated[3].trim();  
 String ready1 = separated[4].trim();  
 *dataFromTheTable*.add(new NewRequest(fio1, problem1, dateString, timeString, ready1));  
 }  
 *// загружаем данные из ObservableList в таблицу* table1.setItems(*dataFromTheTable*);  
  
 *//(обрабатываем изменение полей таблички, можем редактировать не все поля в таблице)* date1.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setDate(event.getNewValue()));  
 time1.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setTime(event.getNewValue()));  
  
 *// кнопка"удалить заявку"* deleteTheRequest.setOnAction((ActionEvent) -> {  
 *// удаляем выбранную заявку из таблицы 1* NewRequest t = table1.getSelectionModel().getSelectedItem();  
 *dataFromTheTable*.remove(t);  
 });  
  
  
 *// кнопка "обработать заявку"* handleTheRequest.setOnAction((ActionEvent event) -> {  
 NewRequest p = table1.getSelectionModel().getSelectedItem();  
 *// если заявка уже выполнена,то изменить её не можем* if (!p.getReady().equals("Выполнено")) {  
 *// охраняем фио и проблему в отдельную переменную, чтобы знать атрибуты какой заявки нужно будет обрабатываать  
 ourFio* = p.getFio();  
 *ourComplain* = p.getProblem();  
 *// и удаляем выбранный элемент из ObservableList и соответственно из таблицы  
 dataFromTheTable*.remove(p);  
 handleTheRequest.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("handleTheRequest\_6.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
  
  
 *// "перезаписываем" файл  
 saveToFile*(*dataFromTheTable*);  
 }  
 });  
  
  
 period1.setOnAction((ActionEvent) -> {  
 *ourPeriod1* = period1.getValue();  
 });  
 period2.setOnAction((ActionEvent) -> {  
 *ourPeriod2* = period2.getValue();  
 });  
  
  
 *// обрабатываем нажатие клавиши "apply" (применить)  
 // используя выбранный пользователем промежуток времени, выводим только соответствующие ему заявки* apply.setOnAction((ActionEvent) -> {  
 *// если поля диапазона дат заполнены,* if (period1.getValue() != null && period2.getValue() != null) {  
 *// то открываем таблицу с соответствующими диапазону заявками* handleTheRequest.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("specificRequestsList\_9.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
  
  
 *// "перезаписываем" файл  
 saveToFile*(*dataFromTheTable*);  
 }  
 });  
  
  
  
 }  
  
  
 *// записываем введённые значения в файл RequestsList.txt* public static void saveToFile(ObservableList<NewRequest> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("RequestsList.txt");  
 StringBuilder sb = new StringBuilder();  
 sb.append("ФИО|проблема|дата поступления заявки|время поступления заявки|состояние готовности|\n");  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String problem = list.get(i + 1).getProblem();  
 String date = list.get(i + 1).getDate();  
 String time = list.get(i + 1).getTime();  
 String ready = list.get(i + 1).getReady();  
 sb.append(Fio);  
 sb.append("|");  
 sb.append(problem);  
 sb.append("|");  
 sb.append(date);  
 sb.append("|");  
 sb.append(time);  
 sb.append("|");  
 sb.append(ready);  
 sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
}

**Класс HandleTheRequestController\_6**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.ChoiceBox;  
import javafx.scene.control.Hyperlink;  
import javafx.scene.control.TextArea;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class HandleTheRequestController\_6 extends ListOfAllRequestsController\_5 {  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Hyperlink hyperlink;  
  
 @FXML  
 private TextArea problem;  
  
 @FXML  
 private ChoiceBox<String> choiceBox;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 void initialize() {  
 StarterWindowController\_1.*shadow*(back);  
  
 *// работаем с полем готовности (choiceBox)* ObservableList<String> args = FXCollections.*observableArrayList*( "0%" ,"Выполнено",  
 "Ожидает рассмотрения", " на 10%", " на 30%", " на 50%", " на 80%");  
 choiceBox.setItems(args);  
  
 *// ссылка на список слиентов* hyperlink.setText("");  
 hyperlink.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow();  
 *// создаём обЪект класса FXMLLoader* FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("clientsHyperlink\_8.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
  
 *// список, чтобы после записать этот элемент в файл с готовыми элементами* LinkedList<NewRequest> newRequest = new LinkedList<>();  
 *// список для пеерзаписи элементов* LinkedList<NewRequest> allRequests = new LinkedList<>();  
  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> StudentList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("RequestsList.txt"))) {  
 *// convert it into a List* StudentList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < StudentList.size(); i++) {  
 String[] separated = StudentList.get(i).split("\\|");  
 String fio1 = separated[0].trim();  
 String problem1 = separated[1].trim();  
 String date = separated[2].trim();  
 String time = separated[3].trim();  
 String ready = separated[4].trim();  
  
 *// находим в текстовом файле нужного нам клиента и его заявку и соответсвенно заполняем поля на экране* if (*ourFio*.equals(fio1) && *ourComplain*.equals(problem1)) {  
 choiceBox.setValue(ready);  
 *// работаем с остальными полями* problem.setText(problem1);  
 hyperlink.setText(*ourFio*);  
  
 *// добавляем элемент в список, чтобы после записать этот элемент в файл с готовыми элементами* newRequest.add(new NewRequest(fio1, problem1, date, time, ready));  
 *// все остальные заявки, которые мы не обрабатываем в данный момент* }else allRequests.add(new NewRequest(fio1, problem1, date, time, ready));  
 }  
  
 *// перезаписали список2 без наших элементов  
 saveToFile1*(allRequests);  
 *// а затем очистили* allRequests.clear();  
  
 *// проверка, имеются ли повторяющиеся уже выполненные заявки  
 // переменная для проверки повторяющихся заявок* boolean repeat = false;  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> list = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("ReadyRequests.txt"))) {  
 *// convert it into a List* list = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < list.size(); i++) {  
 String[] separated = list.get(i).split("\\|");  
 String fio1 = separated[0].trim();  
 String problem1 = separated[1].trim();  
 String date = separated[2].trim();  
 String time = separated[3].trim();  
 String ready = separated[4].trim();  
  
 *// находим в текстовом файле нужного нам клиента и его заявку и соответсвенно заполняем поля на экране* if (*ourFio*.equals(fio1) && *ourComplain*.equals(problem1)) {  
 *// если такая завка уже есть то мы не добавялем эту заявку в список готовых  
 // а просто обновляем* repeat = true;  
 break;  
 }  
 }  
  
  
 *// кнопка"назад"* boolean finalRepeat = repeat;  
 back.setOnAction((ActionEvent event) -> {  
  
 *// .getSelectionModel().getSelectedItem() - возвращает текущий выбранный элемент* if(choiceBox.getSelectionModel().getSelectedItem().equals("Выполнено") && !finalRepeat){  
 *// если заявка выполнена, то добавляем её в файл к другим готовым  
 saveInTheReadyFile*(newRequest);  
 }  
 *// и добавляем изменённую заявку тоже и в список к невыполненным пока заявкам* FileWriter fw;  
 try {  
 fw = new FileWriter("RequestsList.txt", true);  
 String sb = newRequest.get(0).getFio() + "|" +  
 newRequest.get(0).getProblem() + "|" +  
 newRequest.get(0).getDate() + "|" +  
 newRequest.get(0).getTime() + "|" +  
 *//choiceBox.getValue() + "|" + choiceBox.getSelectionModel().getSelectedItem()* choiceBox.getSelectionModel().getSelectedItem() + "|" +  
 "\n";  
 fw.write(sb);  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
  
  
 back.getScene().getWindow().hide();  
 *// создаём обЪект класса FXMLLoader* FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("listOfAllRequests\_5.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
  
 }  
  
 *// записываем введённые значения в файл2* public static void saveInTheReadyFile(LinkedList<NewRequest> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("ReadyRequests.txt", true);  
 StringBuilder sb = new StringBuilder();  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String problem = list.get(i + 1).getProblem();  
 String date = list.get(i + 1).getDate();  
 String time = list.get(i + 1).getTime();  
 String ready = list.get(i + 1).getTime();  
 sb.append(Fio);sb.append("|");  
 sb.append(problem);sb.append("|");  
 sb.append(date);sb.append("|");  
 sb.append(time);sb.append("|");  
 sb.append("Выполнено");sb.append("|");  
 *// автоматически вводится текущая дата* Date now = new Date();  
 DateFormat df = new SimpleDateFormat("dd/MM/yyyy");  
 String dateString = df.format(now);  
 sb.append(dateString);sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
 *// записываем введённые значения в файл2* public static void saveToFile1(LinkedList<NewRequest> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("RequestsList.txt");  
 StringBuilder sb = new StringBuilder();  
 sb.append("ФИО|проблема|дата поступления заявки|время поступления заявки|состояние готовности|\n");  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String problem = list.get(i + 1).getProblem();  
 String date = list.get(i + 1).getDate();  
 String time = list.get(i + 1).getTime();  
 String ready = list.get(i + 1).getReady();  
 sb.append(Fio);sb.append("|");  
 sb.append(problem);sb.append("|");  
 sb.append(date);sb.append("|");  
 sb.append(time);sb.append("|");  
 sb.append(ready);sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
  
}

**Класс ReadyTodayRequestsController\_7**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class ReadyTodayRequestsController\_7 {  
  
 private static int *counter2* = 0;  
  
 private static ObservableList<NewRequest> *dataFromTheTable2* = FXCollections.*observableArrayList*();  
  
 public static ObservableList<NewRequest> getTableData2() {  
 return *dataFromTheTable2*;  
 }  
  
 public static void setTableData2(ObservableList<NewRequest> tableData) {  
 ReadyTodayRequestsController\_7.*dataFromTheTable2* = tableData;  
 }  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 private TableView<NewRequest> table2;  
  
 @FXML  
 private TableColumn<NewRequest, String> fioOfTheClient2;  
  
 @FXML  
 private TableColumn<NewRequest, String> problem2;  
  
 @FXML  
 private TableColumn<NewRequest, String> date2;  
  
 @FXML  
 private TableColumn<NewRequest, String> time2;  
  
 @FXML  
 private TableColumn<NewRequest, String> ready2;  
  
 @FXML  
 void initialize() {  
 StarterWindowController\_1.*shadow*(back);  
  
 *// кнопка"назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("listOfAllRequests\_5.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
  
  
 *// записываем в текстовый файл все заявки, выполненные сегодня  
 // записываем ObservableList в LinkedList, а затем сохраняем наши значения в файл* LinkedList<NewRequest> ordersFirstPart = new LinkedList<>(*dataFromTheTable2*);  
 *// "перезаписываем" файл  
 saveToReadyTodayFile*(ordersFirstPart);  
 *// а затем очищаем* ordersFirstPart.clear();  
 });  
  
  
 *//таблица 2 (выводим список заявок, выполненных за текущие сутки)  
  
 // если не первый с запуска программы раз заходим в окно* if (*counter2* != 0) {  
 *// то очищаем ObservableList  
 dataFromTheTable2*.clear();  
 }  
 *counter2*++;  
  
 *// делаем нашу табличку редактируемой* table2.setEditable(true);  
 *// специфицируем, так сказть, колонки таблицы* fioOfTheClient2.setCellFactory(TextFieldTableCell.*forTableColumn*());*//как текстовое поле* fioOfTheClient2.setCellValueFactory(cellData -> cellData.getValue().fioProperty());  
  
 problem2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 problem2.setCellValueFactory(cellData -> cellData.getValue().problemProperty());  
  
 date2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 date2.setCellValueFactory(cellData -> cellData.getValue().dateProperty());  
  
 time2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 time2.setCellValueFactory(cellData -> cellData.getValue().timeProperty());  
  
 ready2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 ready2.setCellValueFactory(cellData -> cellData.getValue().readyProperty());  
  
  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> clientList2 = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("ReadyRequests.txt"))) {  
 *// convert it into a List* clientList2 = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < clientList2.size(); i++) {  
 String[] separated = clientList2.get(i).split("\\|");  
 String fio1 = separated[0].trim();  
 String problem1 = separated[1].trim();  
 String date1 = separated[2].trim();  
 String time1 = separated[3].trim();  
 String ready1 = separated[4].trim();  
 *//LocalDate readyTime = LocalDate.parse(separated[5].trim());* String readyTime = separated[5].trim();  
  
 *// проверяем время, когда заявку выполнили  
 // и если сегодня, то выводим на экран и добавляем в файл  
  
 // текущая дата* Date now = new Date();  
 DateFormat df = new SimpleDateFormat("dd/MM/yyyy");  
 *//LocalDate dateString = LocalDate.parse(df.format(now));* String dateString = df.format(now);  
  
 if (readyTime.equals(dateString)) {  
 *// добавляем элемент в список  
 dataFromTheTable2*.add(new NewRequest( fio1, problem1, date1, time1, ready1));  
 }  
 }  
  
 *// (обрабатываем изменение полей таблички)* date2.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setDate(event.getNewValue()));  
 time2.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setTime(event.getNewValue()));  
  
  
 *// загружаем данные из ObservableList в таблицу 2* table2.setItems(*dataFromTheTable2*);  
  
  
  
 }  
  
 *// записываем введённые значения в файл2* public static void saveToReadyTodayFile(LinkedList<NewRequest> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("ReadyTodayRequests.txt");  
 StringBuilder sb = new StringBuilder();  
 sb.append("ФИО|проблема|дата поступления заявки|время поступления заявки|состояние готовности|\n");  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String problem = list.get(i + 1).getProblem();  
 String date = list.get(i + 1).getDate();  
 String time = list.get(i + 1).getTime();  
 String ready = list.get(i + 1).getReady();  
 sb.append(Fio);sb.append("|");  
 sb.append(problem);sb.append("|");  
 sb.append(date);sb.append("|");  
 sb.append(time);sb.append("|");  
 sb.append(ready);sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
}

**Класс ClientsHyperLinkController\_8**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.TextFieldTableCell;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.ResourceBundle;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
  
public class ClientsHyperlinkController\_8 {  
  
 private static int *counter* = 0;  
  
 private static ObservableList<NewClient> *tableData* = FXCollections.*observableArrayList*();  
  
 public static ObservableList<NewClient> getTableData() {  
 return *tableData*;  
 }  
  
 public static void setTableData(ObservableList<NewClient> tableData) {  
 ClientsHyperlinkController\_8.*tableData* = tableData;  
 }  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private TableView<NewClient> table;  
  
 @FXML  
 private TableColumn<NewClient, String> fioOfTheClient;  
  
 @FXML  
 private TableColumn<NewClient, String> agreementNumber;  
  
 @FXML  
 private TableColumn<NewClient, String> address;  
  
 @FXML  
 private TableColumn<NewClient, String> ipAddress;  
  
  
 @FXML  
 void initialize() {  
  
 *//таблица  
 // если не первый с запуска программы раз заходим в окно* if (*counter* != 0) {  
 *// то очищаем ObservableList  
 tableData*.clear();  
 }  
 *counter*++;  
 *// делаем нашу табличку редактируемой* table.setEditable(true);  
 */\* специфицируем, так сказть, колонки таблицы  
 Use the setCellFactory method to reimplement the table cell as a text field with  
 the help of the TextFieldTableCell class.  
 \*/* fioOfTheClient.setCellFactory(TextFieldTableCell.*forTableColumn*());*//как текстовое поле* fioOfTheClient.setCellValueFactory(cellData -> cellData.getValue().fioProperty());  
  
 agreementNumber.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 agreementNumber.setCellValueFactory(cellData -> cellData.getValue().agreementNumberProperty());  
  
 address.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 address.setCellValueFactory(cellData -> cellData.getValue().homeAddressProperty());  
  
 ipAddress.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 ipAddress.setCellValueFactory(cellData -> cellData.getValue().ipAddressProperty());  
  
  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> clientList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("ClientsList.txt"))) {  
 *// convert it into a List* clientList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < clientList.size(); i++) {  
 String[] separated = clientList.get(i).split("\\|");  
 String fioOfTheClient1 = separated[0].trim();  
 String agreementNumber1 = separated[1].trim();  
 String address1 = separated[2].trim();  
 String ipAddress1 = separated[3].trim();  
 *tableData*.add(new NewClient(fioOfTheClient1, agreementNumber1, address1, ipAddress1));  
 }  
  
  
 *// загружаем данные из ObservableList в таблицу* table.setItems(*tableData*);  
  
  
 *// записываем ObservableList в LinkedList, а затем сохраняем наши значения в файл* LinkedList<NewClient> clientsFirstPart = new LinkedList<>(*tableData*);  
 *// "перезаписываем" файл  
 saveToFile*(clientsFirstPart);  
  
  
 }  
  
 *// записываем введённые значения в файл1* public static void saveToFile(LinkedList<NewClient> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("ClientsList.txt");  
 StringBuilder sb = new StringBuilder();  
 sb.append("ФИО|номер договора|домашний адрес|ip адрес|\n");  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String number0fTheAgreement = list.get(i + 1).getAgreementNumber();  
 String homeAddress = list.get(i + 1).getHomeAddress();  
 String IPAddress = list.get(i + 1).getIpAddress();  
 sb.append(Fio);  
 sb.append("|");  
 sb.append(number0fTheAgreement);  
 sb.append("|");  
 sb.append(homeAddress);  
 sb.append("|");  
 sb.append(IPAddress);  
 sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
  
}

**Класс SpecificRequestsListController\_9**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.time.ZoneId;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import java.util.ResourceBundle;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class SpecificRequestsListController\_9 {  
  
 private static int *counter2* = 0;  
  
 private static ObservableList<NewRequest> *specificTableData* = FXCollections.*observableArrayList*();  
  
 public static ObservableList<NewRequest> getTableData2() {  
 return *specificTableData*;  
 }  
  
 public static void setTableData2(ObservableList<NewRequest> tableData) {  
 SpecificRequestsListController\_9.*specificTableData* = tableData;  
 }  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 private TableView<NewRequest> table2;  
  
 @FXML  
 private TableColumn<NewRequest, String> fioOfTheClient2;  
  
 @FXML  
 private TableColumn<NewRequest, String> problem2;  
  
 @FXML  
 private TableColumn<NewRequest, String> date2;  
  
 @FXML  
 private TableColumn<NewRequest, String> time2;  
  
 @FXML  
 private TableColumn<NewRequest, String> ready2;  
  
 @FXML  
 private Label lb1;  
  
 @FXML  
 private Label lb2;  
  
 @FXML  
 void initialize() throws ParseException {  
 StarterWindowController\_1.*shadow*(back);  
  
 *// кнопка"назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("listOfAllRequests\_5.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
  
 *// "перезаписываем" файл  
 saveToSpecificFile*(*specificTableData*);  
 });  
  
 *//таблица 2 (выводим список заявок, выполненных за текущие сутки)  
  
 // если не первый с запуска программы раз заходим в окно* if (*counter2* != 0) {  
 *// то очищаем ObservableList  
 specificTableData*.clear();  
 }  
 *counter2*++;  
  
 *// делаем нашу табличку редактируемой* table2.setEditable(true);  
 *// специфицируем, так сказть, колонки таблицы* fioOfTheClient2.setCellFactory(TextFieldTableCell.*forTableColumn*());*//как текстовое поле* fioOfTheClient2.setCellValueFactory(cellData -> cellData.getValue().fioProperty());  
  
 problem2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 problem2.setCellValueFactory(cellData -> cellData.getValue().problemProperty());  
  
 date2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 date2.setCellValueFactory(cellData -> cellData.getValue().dateProperty());  
  
 time2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 time2.setCellValueFactory(cellData -> cellData.getValue().timeProperty());  
  
 ready2.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 ready2.setCellValueFactory(cellData -> cellData.getValue().readyProperty());  
  
 *// устанавливаем в label диапазон* lb1.setText(ListOfAllRequestsController\_5.*ourPeriod1*.toString());  
 lb2.setText(ListOfAllRequestsController\_5.*ourPeriod2*.toString());  
 *// затем преобразуем в String крайние даты диапазона, взятые из datePicker-ов* SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  
 ZoneId defaultZoneId = ZoneId.*systemDefault*();  
 *// преобразуем LocalDate в Date* Date sDate=Date.*from*(ListOfAllRequestsController\_5.*ourPeriod1*.atStartOfDay(defaultZoneId).toInstant());  
 Date eDate=Date.*from*(ListOfAllRequestsController\_5.*ourPeriod2*.atStartOfDay(defaultZoneId).toInstant());  
  
  
  
 *// считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> clientList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("RequestsList.txt"))) {  
 *// convert it into a List* clientList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 for (int i = 1; i < clientList.size(); i++) {  
  
 String[] separated = clientList.get(i).split("\\|");  
 String fio1 = separated[0].trim();  
 String problem1 = separated[1].trim();  
 String dateString = separated[2].trim();  
 *// преобразуем в Date* Date date = sdf.parse(dateString);  
 String timeString = separated[3].trim();  
 String ready1 = separated[4].trim();  
 *// проверяем подходит ли дата в промежуток времени сравниваем тип Date* if (date.compareTo(sDate) >= 0 && date.compareTo(eDate) <= 0) {  
 *// то добавляем эту заявку в список  
 specificTableData*.add(new NewRequest(fio1, problem1, dateString, timeString, ready1));  
 }  
 }  
 *// загружаем данные из ObservableList в таблицу* table2.setItems(*specificTableData*);  
  
 *// (обрабатываем изменение полей таблички)* time2.setOnEditCommit(event ->  
 event.getTableView().getItems().get(event.getTablePosition().getRow()).setTime(event.getNewValue()));  
  
  
 *// "перезаписываем" файл SpecificRequests.txt  
 saveToSpecificFile*(*specificTableData*);  
 }  
  
  
 *// записываем введённые значения в файл SpecificRequests.txt* public static void saveToSpecificFile(ObservableList<NewRequest> list) {  
 FileWriter fw;  
 try {  
 fw = new FileWriter("SpecificRequests.txt");  
 StringBuilder sb = new StringBuilder();  
 sb.append("ФИО|проблема|дата поступления заявки|время поступления заявки|состояние готовности|\n");  
 int N = list.size();  
 for (int i = -1; i < N - 1; i++) {  
 String Fio = list.get(i + 1).getFio();  
 String problem = list.get(i + 1).getProblem();  
 String date = list.get(i + 1).getDate();  
 String time = list.get(i + 1).getTime();  
 String ready = list.get(i + 1).getReady();  
 sb.append(Fio);  
 sb.append("|");  
 sb.append(problem);  
 sb.append("|");  
 sb.append(date);  
 sb.append("|");  
 sb.append(time);  
 sb.append("|");  
 sb.append(ready);  
 sb.append("|");  
 sb.append("\n");  
 }  
 fw.write(sb.toString());  
 fw.close();  
 } catch (IOException e) {  
 System.*out*.println("Файл не найден");  
 }  
 }  
}

**Класс ChartsForAllRequestsController\_10**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.chart.AreaChart;  
import javafx.scene.chart.CategoryAxis;  
import javafx.scene.chart.NumberAxis;  
import javafx.scene.chart.XYChart;  
import javafx.scene.control.Button;  
import javafx.scene.control.DatePicker;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.SimpleDateFormat;  
import java.time.LocalDate;  
import java.time.ZoneId;  
import java.util.\*;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class ChartsForAllRequestsController\_10 {  
  
 private static int *counter* = 0;  
  
 *// поля для выбора даты* public static LocalDate *ourPeriod1*;  
 public static LocalDate *ourPeriod2*;  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private AreaChart<?, ?> areaChart;  
 *// "ось категорий"* @FXML  
 private CategoryAxis x;  
 *// ось NumberAxis вдоль которой каждое значение представляет числовое значение* @FXML  
 private NumberAxis y;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 private DatePicker dp1;  
  
 @FXML  
 private DatePicker dp2;  
  
 @FXML  
 private Button apply;  
  
  
 @FXML  
 void initialize() {  
 StarterWindowController\_1.*shadow*(back);  
 StarterWindowController\_1.*shadow*(apply);  
 *// кнопка"назад"* back.setOnAction((ActionEvent event) -> {  
 back.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("starterWindow\_1.fxml"));  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.show();  
 });  
  
 *// обрабатываем выбор даты* dp1.setOnAction((ActionEvent) -> {  
 *ourPeriod1* = dp1.getValue();  
 });  
 dp2.setOnAction((ActionEvent) -> {  
 *ourPeriod2* = dp2.getValue();  
 });  
  
  
  
  
 XYChart.Series series = new XYChart.Series<>();  
 *// кнопка "применить"* apply.setOnAction((ActionEvent event) -> {  
 *// переменная для проверки, что первая дата раньше последней* boolean correctDates = false;  
 *// каждый раз когда применяем(нажимаем клавишу"применить") увеличиваем счётчик  
 // чтобы потом очистить график для нвого диапазона  
 counter*++;  
 if(*counter*>=2){  
 areaChart.getData().remove(series);  
 series.getData().clear();  
 }  
 *// преобразуем в String крайние даты диапазона, взятые из datePicker-ов* ZoneId defaultZoneId = ZoneId.*systemDefault*();  
 *// преобразуем в Date* Date sDate = Date.*from*(ChartsForAllRequestsController\_10.*ourPeriod1*.atStartOfDay(defaultZoneId).toInstant());  
 Date eDate = Date.*from*(ChartsForAllRequestsController\_10.*ourPeriod2*.atStartOfDay(defaultZoneId).toInstant());  
  
  
 *// проверяем, что первая дата раньше последней* if(sDate.compareTo(eDate) <= 0){  
 correctDates = true;  
 }  
  
 *// если поля диапазона дат заполнены, только тогда можем отображать данные  
 // и если первая дата раньше последней* if (dp1.getValue() != null && dp2.getValue() != null && correctDates ) {  
  
 *// проходим по списку дат в заданном промежутке,  
 // запоминаем каждую дату промежутка и приравниваем её к каждой дате списка заявок  
 // и если в данный день есть заявки добавляем их на график, если нет, то дань добавляем с пустым значением* ObservableList<String> getDatesBetween = *getDatesBetweenString*(sDate, eDate);  
  
 *// цикл для добавления дат* for (int i = 0; i < getDatesBetween.size(); i++) {  
 *// берём дату из промежутка и ищем такую же дату из списка* String current = getDatesBetween.get(i);  
 *// для счётчика кол-ва заявок в определённый день* int counter = 0;  
 boolean found = false;  
  
 *// цикл для добавления элементов  
  
 // считываем каждую новую строку в массив, а потом добавляем эл-ты в список* List<String> clientList = new ArrayList<>();  
 try (Stream<String> stream = Files.*lines*(Paths.*get*("RequestsList.txt"))) {  
 *// convert it into a List* clientList = stream.collect(Collectors.*toList*());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 *// цикл для добавления элементов* for (int j = 1; j < clientList.size(); j++) {  
  
 String[] separated = clientList.get(j).split("\\|");  
  
 String dateString = separated[2].trim();  
  
 if (current.equals(dateString)) {  
 counter++;  
 *// если дата находится в нужном нам промежутке то добавляем эл-т  
 // и соответственно значение к этой дате уже добавлено* found = true;  
 }  
 }  
  
 *// если значение текущей дате current не присвоено,то просто доавляе её со значением "0"* if (!found) {  
 series.getData().add(new XYChart.Data<>(current, 0));  
 }else series.getData().add(new XYChart.Data<>(current, counter));  
 }  
  
 series.setName("поступившие заявки");  
  
 }  
 *// проверяем, что первая дата раньше последней* if(!correctDates){  
 areaChart.getData().remove(series);  
 series.getData().clear();  
 }else {  
 *// добавляем на график* areaChart.getData().addAll(series);  
 }  
 });  
  
 }  
  
 */\*  
 Получение всех дат между двумя датами  
 Одним из способов его вычисления является использование экземпляра Calendar ,  
 выполнение цикла и добавление 1 дня в каждой итерации с использованием метода add и единицы  
 поля Calendar.Date, когда достигается конечная дата.  
 \*/* public static ObservableList<String> getDatesBetweenString(  
 Date startDate, Date endDate) {  
 ObservableList<String> datesInRange = FXCollections.*observableArrayList*();  
 Calendar calendar = new GregorianCalendar();  
 calendar.setTime(startDate);  
  
 Calendar endCalendar = new GregorianCalendar();  
 endCalendar.setTime(endDate);  
 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  
  
 while (calendar.before(endCalendar)) {  
 Date result = calendar.getTime();  
 datesInRange.add(sdf.format(result));  
 calendar.add(Calendar.*DATE*, 1);  
 }  
 *// и ещё одну конечную добавим* Date result = calendar.getTime();  
 datesInRange.add(sdf.format(result));  
 calendar.add(Calendar.*DATE*, 1);  
 return datesInRange;  
 }  
  
  
}

**Класс NewRequest**

package sample;  
  
import javafx.beans.property.SimpleStringProperty;  
  
*// класс заявок*public class NewRequest {  
 private final SimpleStringProperty fio;  
 private final SimpleStringProperty problem;  
 private final SimpleStringProperty date;  
 private final SimpleStringProperty time;  
 private final SimpleStringProperty ready;  
 *// ссылка на следующий элемент списка* public NewRequest next;  
  
 *// Конструктор с пятью аргументами (по характеристикам объекта)* public NewRequest(String fio, String problem, String date,  
 String time, String ready) {  
 this.fio = new SimpleStringProperty(fio);  
 this.problem = new SimpleStringProperty(problem);  
 this.date = new SimpleStringProperty(date);  
 this.time = new SimpleStringProperty(time);  
 this.ready = new SimpleStringProperty(ready);  
 }  
  
 *// конструктор по готовому элементу* public NewRequest(NewRequest students) {  
 this.fio = students.fio;  
 this.problem = students.problem;  
 this.date = students.date;  
 this.time = students.time;  
 this.ready = students.ready;  
 next = null;  
 }  
  
 *// метод для получения следующего элемента* public NewRequest getNext(){  
 return next;  
 }  
  
 *// метод для установки следующего элемента* public void setNext(NewRequest students){  
 next = students;  
 }  
  
 public String getFio() {  
 return fio.get();  
 }  
  
 public SimpleStringProperty fioProperty() {  
 return fio;  
 }  
  
 public void setFio(String fio) {  
 this.fio.set(fio);  
 }  
  
 public String getProblem() {  
 return problem.get();  
 }  
  
 public SimpleStringProperty problemProperty() {  
 return problem;  
 }  
  
 public void setProblem(String problem) {  
 this.problem.set(problem);  
 }  
  
 public String getDate() {  
 return date.get();  
 }  
  
 public SimpleStringProperty dateProperty() {  
 return date;  
 }  
  
 public void setDate(String date) {  
 this.date.set(date);  
 }  
  
 public String getTime() {  
 return time.get();  
 }  
  
 public SimpleStringProperty timeProperty() {  
 return time;  
 }  
  
 public void setTime(String time) {  
 this.time.set(time);  
 }  
  
 public String getReady() {  
 return ready.get();  
 }  
  
 public SimpleStringProperty readyProperty() {  
 return ready;  
 }  
  
 public void setReady(String ready) {  
 this.ready.set(ready);  
 }  
}

**Класс IteratorR**

package sample;  
  
*// Объекты, содержащие ссылки на элементы структур данных и используемые  
// для перебора элементов этих структур, обычно называются итераторами*public class IteratorR {  
 *// Текущий элемент списка* private NewRequest current;  
 *// Предыдущий элемент списка* private NewRequest previous;  
 *// Связанный список* private LinkListR ourList;  
  
 *//Конструктор для создания итератора* public IteratorR(LinkListR list) {  
 ourList = list;  
 reset();  
 }  
  
 *//Сброс итератора и возврат к первому значению* public void reset() {  
 *// current --> first* current = ourList.getFirst();  
 *// previous --> null* previous = null;  
 }  
  
 *//Переход к следующему элементу* public void nextLink() {  
 *// Присваивание текущего элемента previous* previous = current;  
 *// Присваивание текущему элементу next* current = current.getNext();  
 }  
  
 *//Получение текущего элемента* public NewRequest getCurrent() {  
 return current;  
 }  
  
 *//Вставка перед текущим элементом* public void insertBefore(NewRequest students) {  
 *//для методов вставки в LinkList* NewRequest newStudents = new NewRequest(students);  
 *//если текущий элемент является первым* if (previous == null) {  
 newStudents.setNext(ourList.getFirst());  
 ourList.setFirst(newStudents);  
 reset();  
 *//если текущий элемент не первый* } else {  
 newStudents.setNext(previous.getNext());  
 previous.setNext(newStudents);  
 current = newStudents;  
 }  
 *//увеличение размера списка* LinkListR.*size*++;  
 }  
  
 *//Вставка после текущего элемента* public void insertAfter(NewRequest students) {  
 *//для методов вставки в LinkList* NewRequest newStudents = new NewRequest(students);  
 *//Если список пуст* if (ourList.isEmpty()) {  
 ourList.setFirst(newStudents);  
 current = newStudents;  
 *//Если список не пуст* } else {  
 newStudents.setNext(current.getNext());  
 current.setNext(newStudents);  
 nextLink();  
 }  
 *//увеличение размера списка* LinkListR.*size*++;  
 }  
  
 *//определяет, является ли текущий элемент последним в списке* public boolean atEnd() {  
 *//проверка, является ли элемент последним в списке* return (current.getNext() == null);  
 }  
  
 *//Удаление текущего элемента* public void deleteCurrent() {  
 *//Если текущий элемент является первым* if (previous == null) {  
 ourList.setFirst(current.getNext());  
 reset();  
 *//Если текущий элемент не первый* } else {  
 previous.setNext(current.getNext());  
 if (atEnd())  
 *//Сброс, если удалили последний* reset();  
 else  
 current = current.getNext();  
 }  
 *//уменьшение размера* LinkListR.*size*--;  
 }  
  
}

**Класс LinkListR**

package sample;  
  
*// двунаправленный список, где каждый элемент структуры содержит указатели на предыдущий и следующий элементы  
// список для объектов класса NewRequest*public class LinkListR {  
  
 *//счётчик для количества элементов в нашем списке* public static int *size* = 0;  
 *// ссылка на первый элемент списка* private NewRequest first;  
  
 *//конструктор для создания списка* public LinkListR() {  
 first = null;  
 }  
  
 *//метод для получения первого элемента списка* public NewRequest getFirst() {  
 return first;  
 }  
  
 *//метод для установки первого значения списка* public void setFirst(NewRequest students) {  
 first = students;  
 }  
  
 *// Возвращает количество элементов в этом списке.* public int size() {  
 return *size*;  
 }  
  
 *//возвращает true, если список пуст* public boolean isEmpty() {  
 return first == null;  
 }  
  
  
 *// Вставляет указанный элемент в указанную позицию в этом списке.* public void add(int number, NewRequest students) {  
  
 IteratorR iterator = NewRequestController\_4.*iterator*;  
 boolean flag = true;  
 *// Сброс итератора и возврат к первому значению* iterator.reset();  
  
 *//если текущий элемент является первым* if (number == 0) {  
 *// то вставляем первым без проверки на следующий элемент* iterator.insertBefore(students);  
 } else {  
 *// обнуляем индекс* int i = 0;  
 *// если флаг==false => мы дошли до конца списка* while ((flag) && (i != number - 1)) {  
 *// инкриментируем индекс вставки* i++;  
 try {  
 *// Перемещение итератора к следующему элементу* iterator.nextLink();  
 } catch (NullPointerException npe) {  
 *// если следующего элемента нет, то меняем значение флага* flag = false;  
 }  
 }  
 try {  
 *// если индекс равен последнеднему элементу, то мы вставляем элемент в конец* iterator.insertAfter(students);  
 } catch (NullPointerException npe) {  
 System.*out*.println("Не удалось добавить элемент");  
 }  
 }  
 }  
  
 *// добавление элемента в начало списка* public void addFirst(NewRequest students) {  
 NewRequestController\_4.*list*.add(0, students);  
 }  
  
 *// добавление элемента в конец списка* public void addLast(NewRequest students) {  
 IteratorR iterator = NewRequestController\_4.*iterator*;  
 boolean flag = true;  
 *//Сброс итератора и возврат к первому значению* iterator.reset();  
  
 *//если текущий элемент является первым* if (NewRequestController\_4.*list*.isEmpty()) {  
 iterator.insertBefore(students);  
 } else {  
 int i = 0;  
 while (flag) {  
 i++;  
 try {  
 *// Перемещение итератора к следующему элементу* iterator.nextLink();  
 } catch (NullPointerException npe) {  
 flag = false;  
 }  
 }  
 NewRequestController\_4.*list*.add(i - 1, students);  
 }  
 }  
  
 *// удаление элемента по индексу* public void remove(int number) {  
 IteratorR iterator = NewRequestController\_4.*iterator*;  
 boolean flag = true;  
 iterator.reset();  
  
 if (number == 0) {  
 iterator.deleteCurrent();  
 } else {  
 int i = 0;  
 while ((flag) && (i != number)) {  
 i++;  
 try {  
 iterator.nextLink( );  
 } catch (NullPointerException npe) {  
 flag = false;  
 }  
 }  
 try {  
 iterator.deleteCurrent( );  
 } catch (NullPointerException npe) {  
 System.*out*.println("Не удалось удалить элемент");  
 }  
 }  
 }  
  
 *// получение элемента по индексу* public NewRequest get(int number) {  
 IteratorR iterator = NewRequestController\_4.*iterator*;  
 boolean flag = true;  
 iterator.reset( );  
  
 int i = 0;  
 while ((flag) && (i != number)) {  
 i++;  
 try {  
 iterator.nextLink();  
 } catch (NullPointerException npe) {  
 flag = false;  
 }  
 }  
 try {  
 return (NewRequest) iterator.getCurrent();  
 } catch (NullPointerException npe) {  
 System.*out*.println("Не удалось получить элемент");  
 return null;  
 }  
  
 }  
  
 *// установить значение по индексу* public void set(int number, NewRequest students) {  
 NewRequestController\_4.*list*.remove(number);  
 NewRequestController\_4.*list*.add(number, students);  
 }  
  
}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 009 ПЗ | | | | Пояснительная записка | | | | 99 c. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951005 009 ПД | | | | Схема программы | | | | Формат А2 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 009 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | ПС для службы технической поддержки абонентов сети  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Доведько Д.Ю. |  | 25.05.20 | Т |  | |  | 99 | 99 |
| Пров. | | Болтак С.В. |  | 08.06.20 | Кафедра ПОИТ  гр. 951005 | | | | | |
|  | |  |  |  |