Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Департамент анализа данных и машинного обучения**

**Пояснительная записка к курсовой работе по дисциплине «Современные технологии программирования»**

на тему:

**Разработка информационно-справочной системы автосервиса**

Выполнил:

студент группы ЗБ-ПИ19-2

Малкеров Геннадий Александрович

(Подпись)

Научный руководитель:

доцент, канд. физ.-мат. наук.

Корчагин Сергей Алексеевич

(Подпись)

Москва 2021 г.

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc90822221)

[ПОСТАНОВКА ЗАДАЧИ 4](#_Toc90822222)

[ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc90822223)

[АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ 8](#_Toc90822224)

[ОПИСАНИЕ ПРОГРАММЫ 9](#_Toc90822225)

[Алгоритмические решения и состав приложения 9](#_Toc90822226)

[Описание интерфейса программы 13](#_Toc90822227)

[НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ 22](#_Toc90822229)

[Клиент 22](#_Toc90822230)

[Сервер 24](#_Toc90822231)

[ЗАКЛЮЧЕНИЕ 26](#_Toc90822232)

[СПИСОК ЛИТЕРАТУРЫ 27](#_Toc90822233)

[ПРИЛОЖЕНИЕ 28](#_Toc90822234)

# ВВЕДЕНИЕ

В наше время достаточно важно иметь быстрые и эффективные бизнес-процессы внутри компании, которым будут не страшные резкие изменения по тем или иным причинам. Также на текущий момент человек работает с большим объемом не структурированной информации, которая в её структурированно виде могла бы быть в достаточной степени полезна и ускоряла бизнес-процессы.

Чтобы ускорить и улучшить процессы внутри компаний были придуманы так называемые «информационно автоматизированные системы», которые как раз таки призваны улучшить бизнес-процессы и скорость оказываемых услуг.

В рамках работы над данным курсовым проектом предстоит разработать информационно автоматизированную систему автосервиса, которая призваны улучшить процессы и качество оказываемых услуг

Разработка будет производиться в рамках клиент-серверной парадигмы, которая также может улучшить процессы по средствам разделения обязанностей на нескольких работников, которые смогут работать на разных операционных системах.

Для разработки будет использовать достаточно популярный язык программирования JAVA, который был создан в парадигме того, что код будет написан только один раз, но сможет работать на всех современных операционных системах.

При этом главной целью данного курсового проекта является создание клиент-серверного приложения на языке программирования Java, при этом клиент должен быть кросс платформенным.

# ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать программный продукт, состоящий из трех слоев:

* База данных — призвана хранить и минимально обрабатывать данные по необходимости на предназначенном для этого скриптовом языке (в большинстве случаев из семейства языков программирования SQL, но зависит от выбранного типа базы данных);
* Сервер — призван выполнять основную связывающую роль между клиентом и базой данных и при этом для клиента совершенно не важно откуда и как были вытащены данные, которые он запросил, потому что с сервером принято общаться структурированными форматами данных (например JSON, XML(SOAP — под вид XML, но с более декларированным стилем) и так далее);
* Клиент — призван выполнить все что касается работы с сервером и отображением данных для пользователя в удобном для пользователя виде. Клиент может быть реализован как web-сайт или приложение для компьютера.

В рамках же конкретно моей темы, необходимо разработать клиент-серверной приложение на языке программирования JAVA, которое в качестве БД будет использовать PostgreSQL, а для общения с между клиентом и сервером будет использован REST Full подход.

Среди выполненных условий можно выделить следующие:

* Отображать необходимые для конкретной предметной области данные в табличном виде;
* Данные в базе данных должны быть предварительно заполнены и быть релевантными;
* Реализовать возможность сортировки и поиска данных по некоторым параметрам;
* Добавить возможность обновлять данные в клиенте при необходимости пользователя;
* Данные после добавления должны быть актуальными для пользователя без дополнительных действий;
* Добавить отдельны пункт описания автора и возможность выйти из программы.

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметной областью данного курсового проекта является «условный» автосервис.

В предметной области можно выделить следующие основные сущности, с которыми придется работать пользователям программного комплекса:

* Клиент — краткое описание информации о пользователе и способе связи с ним при необходимости уточнения каких-то деталей или статуса заказа если к назначенному времени пользователь не явился;
* Услуга — тип услуг, которые могут быть оказаны в данном автосервисе. Помимо типа услуг можно также указать ее название, а также среднюю длительность, за которую в среднем она может быть выполнена.
* Заказ — основная сущность предметной области, суть которой является связь сущностей клиента и услуги, которую он хочет чтобы ему произвели.

Основной сущностью программы является заказ. Заказ должен быть назначен на время и определенного клиента, а также должна быть указана услуга, которую клиент хочет получить.

Заказ представляет из себя сущность, у которой есть ссылка на услугу, а также ссылка на клиента, которую планируется предоставить услугу. У заказа есть 2 статуса:

* Не выполнено — проставляется автоматически при создании заказа и означает, что заказ еще не выполнен и только ожидает выполнение или находиться в процессе выполнения;
* Выполнено — проставляется, когда заказ был выполнен физически. При этом при изменении статуса необходимо также указать конечную цену для пользователя.

Разработанный программы комплекс, для автоматизации автосервиса, поможет конечным пользователя проще работать с заказами и получать статистику о работе.

# АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ

В наше не простое время, когда количество контактов между людьми должно быть минимизировано, важно иметь возможность ремонтировать автомобиль, который используется для перемещения в рамках необходимости, с большим комфортом для пользователя и меньшими трудовыми затратами для коллективов автосервиса.

Также важно понимать примерный график выручки и количества оказанных услуг относительно некоторого промежутка времени.

А также в наше время все чаще на некоторых производствах и в некоторых компаниях начинаю использовать операционные системы отличные от windows, при этом разработка кросс платформенного клиента поможет сгладить переход и пользователям не придется привыкать к новому программному обеспечению.

# ОПИСАНИЕ ПРОГРАММЫ

## Алгоритмические решения и состав приложения

Суть работы заключается в автоматизации процессов автосервиса по средствам разработки программного комплекса из трех составляющих:

* База данных;
* Сервер;
* Клиент.

Для работы была выбрана база данных PostgreSQL. Данная база в некотором роде избыточна для данного не большого программного комплекса, однако у нее есть ряд плюсов:

* Она бесплатна и разрабатывается в режиме открытого исходного кода;
* Для популярной среды виртуализации — Docker— дистрибутив базы данных поставляется в удобном для разворота виде;
* Сертифицирована для работы в России;
* Производительна на больших объемах данных (на текущий момент это не важно, но фундамент программного комплекса положен и дальнейшая смена провайдера баз данных будет затратна).

База данных при этом имеет 3 таблицы. Схема базы данных при этом расположена на рисунке 1, более подробное описание таблиц приведем ниже.

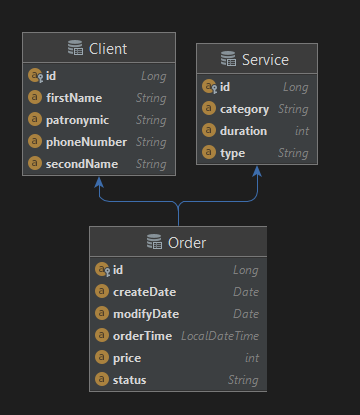


Рисунок 1 — Схема созданной базы данных

Таблица клиентов предназначена для хранения информации о клиентах, которых обслуживались в автосервисе и имеет название в базе данных «Client», при этом имеет следующие поля:

* id — с типом данных long — уникальный идентификатор записи в таблице;
* firstName — с типом данных String — имя определенного клиента;
* secondName— с типом данных String — фамилия определенного клиента;
* patronymic— с типом данных String — отчество определенного клиента;
* phoneNumber— с типом данных String — номер телефона для связи с клиентом.

Таблица услуг предназначена для хранения информации об услугах, которые могу быть оказаны в автосервисе и имеет название в базе данных «Service», при этом имеет следующие поля:

* id — с типом данных long — уникальный идентификатор записи в таблице;
* type— с типом данных String — тип определенной услуги;
* duration— с типом данных int — длительность оказания услуги;
* category— с типом данных String —категория услуг;

Таблица заказов является основной в данной структуре и предназначена для хранения информации о заказах, которых выполнялись в автосервисе и имеет название в базе данных «Service», при этом имеет следующие поля:

* id — с типом данных long — уникальный идентификатор записи в таблице;
* orderTime— с типом данных Date— дата на которую был создан заказ;
* price— с типом данных int — стоимость оказанных услуг;
* client— с типом данных long — идентификатор клиента из таблицы Client, с которой соединение происходит по принципу «многие к одному», то есть клиент может иметь несколько заказов;
* service— с типом данных long — идентификатор услуги из таблицы Service, с которой соединение происходит по принципу «многие к одному», то есть одна услуга может использоваться в нескольких заказах;
* status— с типом данных String — статус в которой находиться заказ;
* createDate— с типом данных Date—дата создания заказа;
* modifyDate— с типом данных Date—дата последнего редактирования заказа;

С базой данных работает по средства ORM Hibernate- SpringBootFramework.

Суть работы ORM состоит в том что программист описывает структуру таблицы в привычном для него месте, а также добавляет определенные атрибуты описывающие те или иные аспекты работы с базой данных, например связи путем добавления атрибута @ManyToOne или @OneToMany, или для указания того что поле является идентификатором в таблице @Id.

После описания моделей, необходимо создать так называемые репозитории, суть которых абстрагировать пользователя, а точнее программиста от написания sql запросов, потому что эту работу как раз и выполняет ORM. Мной для работы были выбраны JpaRepository, от которых и были от наследованы мои реализации репозиториев.

Для работы же с клиентом был использован SpringBootFramework, который опять же посредствам определенных атрибутов может в определенных классах, которые называются контроллерами, общаться с внешним миром по средствам Rest запросов.

Клиент в данной связке был реализован по средствам JAVAFx, который призван упростить работу с интерфейсом для пользователя и сделать конечный программный продукт кроссплатформенный, что и требовалось по требования к программному продукту.

## Описание интерфейса программы

В данном программном комплексе интерфейс для пользователя имеется только в клиентской части, серверная при этом интерфейса для пользователя не имеет.

При этом в клиентской части программного продукта представлено 8 окон, рассмотрим их по отдельности.

Начнем с так называемого главного окна, которое при старте клиентской части продукта будет видеть пользователь. Для этого запустим приложение и увидим пример окна с рисунка 2.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2 — Пример главного окна приложения

Сверху данного окна мы видим меню, рассмотрим его подробнее на рисунке 3.

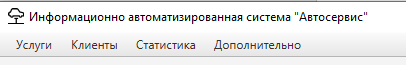


Рисунок 3 — Основные пункты меню

Часть меню «Услуги» содержит все возможные операция над услугами, которые может выполнить пользовать. Пункты из меню представлены на рисунке 4

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4 — Пункт меню «Услуги»

Подпункт меню «Просмотреть» из пункта «Услуги» открывает окно с возможностью просмотра, пример окна представлен на рисунке 5.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 5 — Просмотр всех возможных услуг

Также в данном окне работает поиск, для этого достаточно начать что-то писать в нижнее поле для поиска. Пример работы поиска представлен на рисунке 6. Поиск при этом работает без учета регистра.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 6 — Пример работы поиска в списке оказываемых услуг.

Если же закрыть окно выше и открыть подпункт «Создать» пункта меню «Услуги», то можно увидеть окно, представленное на рисунке 7. Данное окно предназначено для создания новой услуги. При это в данном окне есть параметры валидации для заполненных полей. Если в поле «Название» ничего не ввести, то будет показано предупреждение, пример которых для всей клиентской части представлен на рисунке 8.

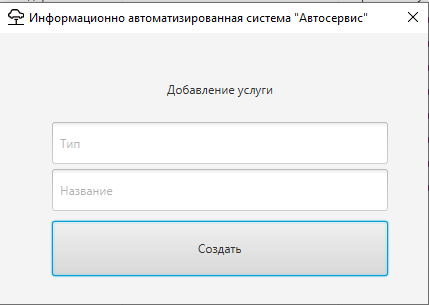


Рисунок 7 — Пример окна «Добавление услуг».

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 — Пример окна внимания.

Пункт меню «Клиенты» с рисунка 3, состоит из следующих вариантов, представленных на рисунке 9. При нажатии на данный пункт меню произойдет открытие окна аналогичное окну списка оказываемых услуг, только там будут отображены клиенты, пример отображения окна с клиентами представлен на рисунке 10.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 — Подпункт из меню «Клиенты».

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 10 — Окно просмотра информации о клиентах.

Пункт меню «Статистика» с рисунка 3 имеет подпункт, представленный на рисунке 11. При нажатии на данный подпункт откроется окно, представленное на рисунке 12. Суть данного окна в отображении статистики по востребованным услугам и стоимости ремонтов.

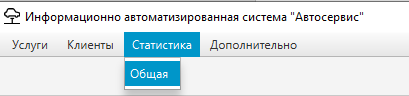


Рисунок 11 — Подпункт меню «Статистика»

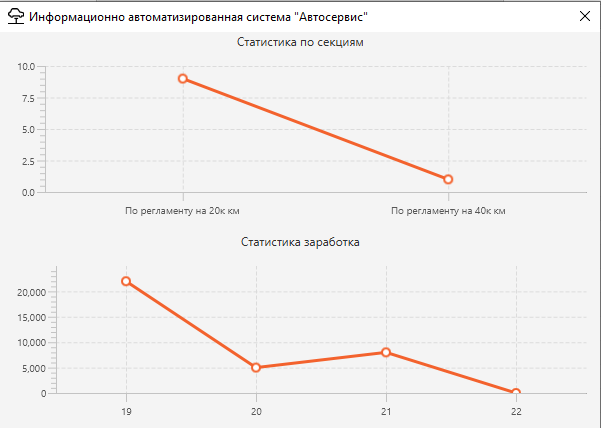


Рисунок 12 — Пример отображении окна общей статистки

Пункт меню «Дополнительно» с рисунка 3 имеет следующие подпункты представленные на рисунке 13. При нажатии на пункт «Об авторе», можно увидеть окно, представленное на рисунке 14, на котором представлена информация о создателе данного программного продукта. При нажатии на пункт меню «Выход» программа закроется.

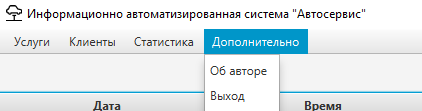


Рисунок 13 — Подпункты меню «Дополнительно»

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 — Окно «Об авторе»

Теперь разберет главную часть окна, представленного на рисунке 2. Сверху окна имеется кнопка обновления данных, при нажатии на которую данные о заказ заново получаются из базы данных по средствам сервера на случай, если над заказами работает несколько работников одновременно.

Посереди окна расположена таблица, на которое выведена основная информация о заказах, которые сейчас есть в системе. При это при желании данная таблица поддерживает сортировка по

При необходимости добавления нового заказа необходимо нажать на кнопку «Создать заказ», после чего откроется окно создания заказа, представленное на рисунке 15. Далее в данном окне необходимо выбрать тип услуг, которые необходимо оказать и выбрать услуг из типа для оказания. Далее в поле ввода номера телефона вводиться номер телефона клиента в стандартном режиме, при этом если такой номер уже есть у клиента, то данные о клиенте заполняться автоматически. Далее нужно выбрать дату, которая должна быть сегодня или позже иначе появиться текст предупреждения сто заказы на прошлое создать нельзя. В последнем пункте необходимо выбрать час, в который пользователь может приехать.

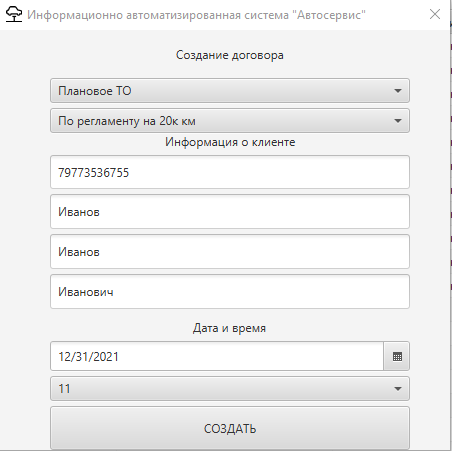


Рисунок 15 — Пример заполнения окна создания заказа.

После нажатия на кнопку создать появиться информация о том, что заказ успешно создан.

Для того чтобы описать сколько стоил заказ или завершить его необходимо выбрать заказ из списка и нажать на кнопку «Редактировать заказ», после чего описание заказа откроется в новом окне, пример которого представлен на рисунке 16.

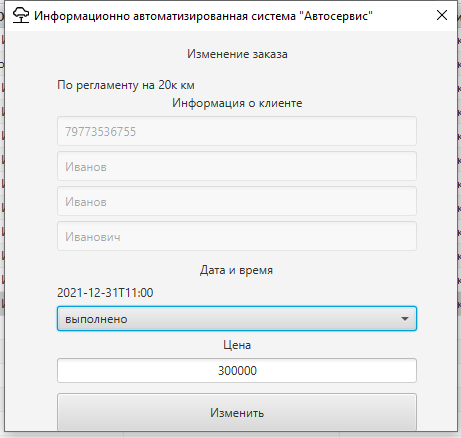


Рисунок 16 — Пример отображения окна редактирования заказа.

Если же заказ, по каким-то причинам отмене его можно удалить для этого его также надо выбрать из списка и нажать кнопку «Удалить заказ» после чего появиться предупреждение, пример которого представлен на рисунке 17

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 — Пример отображения окна «Согласие на действие»

Стоит отметить что основное окно программы возможно редактировать и возможно скрывать, однако все окна, которые открываются дополнительно заблокированы для изменения размера. Данное ограничение было сделано специально, чтобы у пользователя была возможность видеть главное окно программы не зависимо от других открытых окон.

# НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ

## Клиент

Проект клиента состоит из 3 основных групп файлов

1. Контроллеры обработки данных, расположенные в папки «Controllers» проекта клиента;
   * ClientListController — класс обработки данных о списке клиентов;
   * EditOrderController — класс обработки обновления данных о заказе ;
   * NewOrderController ؙ— класс обработки создания нового заказа;
   * NewServiceController — класс обработки создания новой услуги;
   * RootController — класс обработки основного окна приложения;
   * ServiceListController — класс обработки действия по отображения списка услуг;
   * StatInfoController — класс обработки окна информации о статистке;
2. Вспомогательный классы, которые расположены в папке «Support» проекта клиента;
   * ClientStr — класс представления данных о клиенте в формате приемлемом для JavaFx;
   * OrderStr — класс представления данных о заказах в формате приемлемом для JavaFx;
   * PhoneValidator — класс созданный для валидации данных о телефоне, работает валидация через регексы;
   * ServerRequestIOperator — сервисный класс созданный для обработки запросов к серверу и представления результатов в объектном виде
   * ServiceStr — класс представления данных об услугах в формате приемлемом для JavaFx;
   * Main ؅— класс основной для описания логики старта и создания новых окон;
3. Ресурсы
   * Author.fxml — описания структуры окна в формате JavaFX для окна описания создателя программного продукта;
   * ClientList.fxml — описания структуры окна в формате JavaFX для окна описания списка же имеющихся в системе клиентов;
   * EditOrder.fxml — описания структуры окна в формате JavaFX для окна редактирования информации о заказе;
   * NewOrder.fxml — описания структуры окна в формате JavaFX для окна создания нового заказа;
   * NewService.fxml — описания структуры окна в формате JavaFX для окна создания новой услуги;
   * Root.fxml — описания структуры окна в формате JavaFX для главного окна приложения;
   * ServiceList.fxml — описания структуры окна в формате JavaFX для окна списка оказываемых услуг;
   * Stat.fxml — описания структуры окна в формате JavaFX для описание окна статистки по заказам.

## Сервер

Проект сервера состоит из трех основных групп файлов:

1. Контроллеры обработки запросов со стороны клиента расположены в папке «Controller»;
   * ClientController — контроллер обработки Http запросов, связанных с работой с клиентами;
   * OrderController — контроллер обработки Http запросов, связанных с работой с заказами;
   * ServiceController — контроллер обработки Http запросов, связанных с работой с оказываемыми услугами;
2. Модели описания таблиц к базе данных по средствам определенных атрибутов, а также модели общения с клиентом описаны в папке «Model»;
   * Client — описание модели данных о клиенте;
   * Order — описание модели данных о заказе;
   * Service — описание модели данных об оказываемых услугах;
3. Репозитории для работы с базой данных посредствам Hibernate описаны в папке «Repositrory»;
   * ClientRepository — описание интерфейса репозитория через которых идет общение с базой данных по средствам наследования интерфейса JpaRepository для работы с информацией о клиентах;
   * OrderRepository — описание интерфейса репозитория через которых идет общение с базой данных по средствам наследования интерфейса JpaRepository для работы с информацией о заказах;
   * ServiceRepository — описание интерфейса репозитория через которых идет общение с базой данных по средствам наследования интерфейса JpaRepository для работы с информацией об оказываемых услугах;

# ЗАКЛЮЧЕНИЕ

В рамках работы над данным курсовом проектом было реализовано клиент серверное—приложения на языке программирования Java с кроссплатформенным клиентом на тему «Информационно справочная система автосервиса».

Со стороны клиента реализован достаточно удобный интерфейс с понятными для пользователя путями решения определенных задач в рамках выполнения своих профессиональных задачах.

Созданный программный комплекс решает все поставленные задачи по средствам выполнения RESTFull API с помощью протокола HTTP c последующим удобным выводом или вводом данных по средствам клиента основанном на графическом интерфейсе построенным по средствам библиотеки JavaFX.

Во время выполнения проекты были получены навыки работы со следующими технологиями и инструментами:

* Java и JavaFX в качестве основных языков и концепций;
* IntelliJ IDEA в качестве основного инструмента написания;

Решение по средствам использования достаточно чистой архитектуры может просто расширяться и обновляться при необходимости. Например, в программу можно внедрить систему распределения заказов на работников, а также вести полноценный учет сотрудников определенного автосервиса. Также можно добавить систему ролей с разными обязанностями пользователей, чтобы улучшить производительность в выполнение своих профессиональных задач еще большего количества рабочего коллектива

# СПИСОК ЛИТЕРАТУРЫ

Законодательные и нормативные акты

1. ГОСТ 7.1. – 2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
2. ГОСТ 7.32 – 2001. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.
3. ГОСТ 7.82 – 2001. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2001. –21 с.
4. Единая система программной документации. – М.: Стандартинформ, 2005. –128 с.
5. Козмина Ю., Харроп Р. Spring 5 для профессионалов. - Киев: Диалектика-Вильямс, 2019. - 1120 с.
6. Коузен К. Современный Java. Рецепты программирования . - М.: ДМК Пресс, 2018. - 274 с.
7. Прохоренок Н.А. JavaFX. - СПб: БХВ-Петербург, 2020. - 768 с.

# ПРИЛОЖЕНИЕ

Программы код основных модулей серверной части приложения приложен ниже

ClientController

package auto\_service.server.Controller;

import auto\_service.server.Model.Client;

import auto\_service.server.Repositrory.ClientRepository;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("auto\_service")

public class ClientController

{

private final ClientRepository clientRepository;

public ClientController(ClientRepository clientRepository)

{

this.clientRepository = clientRepository;

}

@PostMapping("/clients")

Client newClient(@RequestBody Client client)

{

return clientRepository.save(client);

}

@GetMapping("/clients")

public ResponseEntity<List<Client>> getAllClients(){

return new ResponseEntity<>(clientRepository.findAll(), HttpStatus.OK);

}

@GetMapping("/clients/{phoneNumber}")

@ResponseBody

public ResponseEntity<Client> getClientByPhone(@PathVariable(value = "phoneNumber") String phone)

{

return new ResponseEntity<>(clientRepository.findClientByPhoneNumber(phone), HttpStatus.OK);

}

}

OrderController

package auto\_service.server.Controller;  
  
import auto\_service.server.Model.Order;  
import auto\_service.server.Repositrory.OrderRepository;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("auto\_service")  
public class OrderController  
{  
 private final OrderRepository orderRepository;  
  
 public OrderController(OrderRepository orderRepository)  
 {  
 this.orderRepository = orderRepository;  
 }  
  
 @GetMapping("/orders")  
 public ResponseEntity<List<Order>> getAllOrders()  
 {  
 return new ResponseEntity<>(orderRepository.findAll(), HttpStatus.OK);  
 }  
  
 @PostMapping("/orders")  
 Order newOrder(@RequestBody Order order)  
 {  
 return orderRepository.save(order);  
 }  
  
 @PutMapping("/orders/{id}")  
 @ResponseBody  
 public ResponseEntity<Order> updateStoreProduct(@PathVariable(value = "id") Long id, @RequestBody Order order)  
 {  
  
 Order order1 =orderRepository.findStoreProductById(id);  
 if(order1 != null){  
 order1.setPrice(order.getPrice());  
 order1.setStatus(order.getStatus());  
 return new ResponseEntity<>(orderRepository.save(order1),HttpStatus.OK);  
 }  
 return new ResponseEntity<>(orderRepository.save(order),HttpStatus.OK);  
 }  
   
 @GetMapping("/time/{date}/{duration}")  
 @ResponseBody  
 public ResponseEntity<ArrayList<Integer>> getClientByPhone(@PathVariable(value = "date") String date,  
 @PathVariable(value="duration") Integer duration){  
 List<Order> orders = orderRepository.findAll();  
 ArrayList<Integer> times = new ArrayList<>();  
 for(int i=8;i<19;i++){  
 times.add(i);  
 }  
 for(Order o:orders){  
 if(o.getOrderTime().toLocalDate().toString().equals(LocalDate.parse(date).toString())){  
 times.removeIf(i -> i == o.getOrderTime().getHour());  
 int d = o.getService().getDuration()/60;  
 if(d!=0 ){  
 for(int j=0;j<d;j++){  
 int finalJ = j;  
 times.removeIf(i -> i ==o.getOrderTime().getHour()+ finalJ);  
 }  
 }  
 }  
 }  
 var d = 1/60;  
 for(int i=0;i<times.size();i++){  
 for(int dd=0;dd<d;dd++){  
 if(!times.contains(times.get(i)+dd)){  
 int finalI = i;  
 times.removeIf(z -> z.equals(times.get(finalI)));  
 }  
 }  
 }  
 for(int i=0;i<times.size();i++){  
 if(24-d<times.get(i)){  
 int finalI1 = times.get(i);  
 times.removeIf(z -> z.equals(finalI1));  
 }  
 }  
  
  
 return new ResponseEntity<>(times, HttpStatus.OK);  
 }  
  
 @DeleteMapping("/orders/{id}")  
 void deleteOrder(@PathVariable Long id)  
 {  
 orderRepository.deleteById(id);  
 }  
}

ServiceController

package auto\_service.server.Controller;  
  
import auto\_service.server.Model.Service;  
import auto\_service.server.Repositrory.ServiceRepository;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("auto\_service")  
public class ServiceController  
{  
 private final ServiceRepository serviceRepository;  
  
 public ServiceController(ServiceRepository serviceRepository)  
 {  
 this.serviceRepository = serviceRepository;  
 }  
   
 @GetMapping("/services")  
 public ResponseEntity<List<Service>> getAllServices(){  
 return new ResponseEntity<>(serviceRepository.findAll(), HttpStatus.OK);  
 }  
  
  
 @PostMapping("/serviceCreate")  
 Service newService(@RequestBody Service service)  
 {  
 return serviceRepository.save(service);  
 }  
}

Client

package auto\_service.server.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
import java.util.Objects;  
  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name="Clients")  
@Data  
public class Client {  
 ;  
 public Client(String firstName, String secondName,String patronymic,  
 String phoneNumber)  
 {  
 this.firstName = firstName;  
 this.secondName = secondName;  
 this.phoneNumber = phoneNumber;  
 this.patronymic = patronymic;  
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id")  
 private Long id;  
  
 @Column(name = "firstName", length = 128)  
 private String firstName;  
  
 @Column(name = "secondName", length = 128)  
 private String secondName;  
  
 @Column(name = "patronymic", length = 128)  
 private String patronymic;  
  
 @Column(name = "phoneNumber", length = 128)  
 private String phoneNumber;  
  
 @Override  
 public boolean equals(Object o)  
 {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Client client = (Client) o;  
 return Objects.equals(id, client.id) && Objects.equals(firstName, client.firstName) && Objects.equals(secondName, client.secondName) && Objects.equals(patronymic, client.patronymic) && Objects.equals(phoneNumber, client.phoneNumber);  
 }  
  
 @Override  
 public int hashCode()  
 {  
 return Objects.hash(id, firstName, secondName, patronymic, phoneNumber);  
 }  
}

Order

package auto\_service.server.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.hibernate.annotations.CreationTimestamp;  
import org.hibernate.annotations.UpdateTimestamp;  
  
import javax.persistence.\*;  
import java.time.LocalDateTime;  
import java.util.Date;  
  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name="Orders")  
@Data  
public class Order {  
   
 @CreationTimestamp  
 private Date createDate;  
  
 @UpdateTimestamp  
 private Date modifyDate;  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private LocalDateTime orderTime;  
  
 private int price;  
  
 @ManyToOne(fetch = FetchType.EAGER)  
 private Client client;  
  
 @ManyToOne  
 private Service service;  
  
 private String status;  
  
 ;  
 public Order(LocalDateTime orderTime, int price, String status){  
 this.price = price;  
 this.status = status;  
 this.orderTime = orderTime;  
  
 }  
}

Service

package auto\_service.server.Model;  
  
import lombok.\*;  
import org.hibernate.Hibernate;  
  
import javax.persistence.\*;  
import java.util.Objects;  
  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Getter  
@Setter  
@Table(name="Services")  
@ToString  
public class Service {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id")  
 private Long id;  
  
 @Column(name = "type")  
 private String type;  
  
 @Column(name = "duration", nullable = false)  
 private int duration;  
  
 @Column(name = "category")  
 private String category;  
  
 public Service(String type,int duration)  
 {  
 this.type = type;  
 this.duration = duration;  
 }  
   
 @Override  
 public boolean equals(Object o)  
 {  
 if (this == o) return true;  
 if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;  
 Service service = (Service) o;  
 return id != null && Objects.equals(id, service.id);  
 }  
   
 @Override  
 public int hashCode()  
 {  
 return getClass().hashCode();  
 }  
}

Код клиентской части продукта

ClientListController

package autoService.Controllers;  
  
import autoService.Main;  
import autoService.Support.ClientStr;  
import autoService.Support.ServerRequestIOperator;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.collections.transformation.FilteredList;  
import javafx.collections.transformation.SortedList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Alert;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.TextField;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
  
public class ClientListController  
{  
  
 @FXML private TableColumn<ClientStr, String> firstName;  
 @FXML private TableColumn<ClientStr, String> secondName;  
 @FXML private TextField textField;  
 @FXML private TableColumn<ClientStr, String> patronymic;  
 @FXML private TableColumn<ClientStr, String> phoneNumber;  
 @FXML private TableView<ClientStr> clientTable;  
 private ObservableList<ClientStr> clientData = FXCollections.observableArrayList();  
 private FilteredList<ClientStr> filteredData = new FilteredList<>(clientData );  
 private SortedList<ClientStr> sortableData = new SortedList<>(filteredData);  
 private Main main;  
 private Stage dialogStage;  
  
 public void setMain(Main main) {  
 this.main = main;  
 }  
 @FXML  
 void initialize(){  
 firstName.setCellValueFactory(cellData -> cellData.getValue().firstNameProperty());  
 secondName.setCellValueFactory(cellData -> cellData.getValue().secondNameProperty());  
 patronymic.setCellValueFactory(cellData -> cellData.getValue().patronymicProperty());  
 phoneNumber.setCellValueFactory(cellData -> cellData.getValue().phoneNumberProperty());  
 textField.textProperty().addListener((observable, oldValue, newValue) ->  
 filteredData.setPredicate(p -> p.getFirstName().toLowerCase().contains(textField.getText().toLowerCase()) |  
 p.getPhoneNumber().toLowerCase().contains(textField.getText().toLowerCase()) |  
 p.getSecondName().toLowerCase().contains(textField.getText().toLowerCase())|  
 (p.getFirstName().toLowerCase()+" "+p.getSecondName().toLowerCase()).contains(textField.getText().toLowerCase()))  
  
 );  
  
 }  
 public void load() {  
 clientData = ServerRequestIOperator.getClients();  
 if(clientData == null){  
 Alert alert = new Alert(Alert.AlertType.WARNING);  
 alert.initModality(Modality.APPLICATION\_MODAL);  
 alert.initOwner(main.getPrimaryStage());  
 alert.setHeaderText("Ошибка на сервере");  
 alert.setContentText("Не удалось загрузить данные");  
 alert.show();  
 }  
 else{  
 this.filteredData = new FilteredList<>(clientData);  
 this.sortableData = new SortedList<>(this.filteredData);  
 this.clientTable.setItems(this.sortableData);  
 this.sortableData.comparatorProperty().bind(this.clientTable.comparatorProperty());  
 this.dialogStage.show();  
 }  
  
 }  
  
 public void setDialogStage(Stage dialogStage) {  
 this.dialogStage = dialogStage;  
 }  
}

EditOrderController

package autoService.Controllers;  
  
import autoService.Main;  
import autoService.Support.ServerRequestIOperator;  
import autoService.Support.OrderStr;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Alert;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.io.IOException;  
  
public class EditOrderController {  
 private Main main;  
 private Stage stage;  
 private RootController root;  
 @FXML  
 Label service;  
 @FXML  
 TextField phoneNumber;  
 @FXML  
 TextField name;  
 @FXML  
 TextField secondName;  
 @FXML  
 TextField patr;  
 @FXML Label date;  
 @FXML TextField price;  
 @FXML  
 ComboBox<String> status;  
  
 public void setStage(Stage primaryStage) {  
 this.stage = primaryStage;  
 }  
 public void setMain(Main main){  
 this.main = main;  
 }  
 public void setRoot(RootController root){  
 this.root = root;  
 }  
 public void load() {  
  
 OrderStr orderStr = this.root.getOrderTable().getSelectionModel().getSelectedItem();  
 this.service.setText(orderStr.getServiceName().get());  
 this.phoneNumber.setText(orderStr.getClientPhone().get());  
 this.name.setText(orderStr.getClient().getString("firstName"));  
 this.secondName.setText(orderStr.getClient().getString("secondName"));  
 try{  
 this.patr.setText(orderStr.getClient().getString("patronymic"));  
 }  
 catch(JSONException j){  
 this.patr.setText(null);  
 }  
 this.date.setText(orderStr.getOrderTime().toString());  
 try{  
 this.price.setText(orderStr.getPrice().toString());  
 }  
 catch(JSONException j){  
 this.price.setText(null);  
 }  
 ObservableList<String> status = FXCollections.observableArrayList();  
 status.add("не выполнено");  
 status.add("выполнено");  
 this.status.setItems(status);  
 this.status.setValue(orderStr.getStatus());  
 this.stage.show();  
 }  
 @FXML  
 public void change(){  
 if(!this.price.getText().equals(String.valueOf(this.root.getOrderTable().getSelectionModel().getSelectedItem().getPrice()))|  
 !this.status.getValue().equals(this.root.getOrderTable().getSelectionModel().getSelectedItem().getStatus())){  
  
 JSONObject jsonObject = new JSONObject();  
 jsonObject.put("status",this.status.getValue());  
 try{  
 jsonObject.put("price",Integer.parseInt(this.price.getText()));  
 jsonObject.put("orderTime",this.root.getOrderTable().getSelectionModel().getSelectedItem().getOrderTime());  
 jsonObject.put("client",this.root.getOrderTable().getSelectionModel().getSelectedItem().getClient());  
 jsonObject.put("service",this.root.getOrderTable().getSelectionModel().getSelectedItem().getService());  
 try{  
 ServerRequestIOperator.changeOrder(this.root.getOrderTable().getSelectionModel().getSelectedItem().getId(),jsonObject);  
 this.stage.close();  
 this.root.load();  
 }  
 catch(IOException e){  
 Alert alert = new Alert(Alert.AlertType.WARNING);  
 alert.initModality(Modality.APPLICATION\_MODAL);  
 alert.initOwner(main.getPrimaryStage());  
 alert.setContentText("Сервер не отвечает");  
 alert.setHeaderText("Данные не были изменены");  
 alert.show();  
 }  
 }  
 catch(Exception e){  
 Alert alert = new Alert(Alert.AlertType.WARNING);  
 alert.initModality(Modality.APPLICATION\_MODAL);  
 alert.initOwner(main.getPrimaryStage());  
 alert.setContentText("Цена должна быть числом!");  
 alert.setHeaderText("Данные не были обновлены");  
 alert.show();  
 }  
  
  
 }  
 else{  
 Alert alert = new Alert(Alert.AlertType.WARNING);  
 alert.initModality(Modality.APPLICATION\_MODAL);  
 alert.initOwner(main.getPrimaryStage());  
 alert.setContentText("Нет изменений");  
 alert.setHeaderText("Данные не были обновлены");  
 alert.show();  
 }  
 }  
}

RootController

package autoService.Controllers;  
  
  
import autoService.Main;  
import autoService.Support.ServerRequestIOperator;  
import autoService.Support.OrderStr;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Alert;  
import javafx.scene.control.ButtonType;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.Optional;  
  
public class RootController {  
 private Main main;  
 @FXML private TableView<OrderStr> orderTable;  
 @FXML private TableColumn<OrderStr, String> orderTime;  
 @FXML private TableColumn<OrderStr, String> orderDate;  
 private ObservableList<OrderStr> orderData = FXCollections.observableArrayList();  
 @FXML private TableColumn<OrderStr, String> client;  
 @FXML private TableColumn<OrderStr, String> fio;  
 @FXML private TableColumn<OrderStr, String> service;  
 @FXML private TableColumn<OrderStr, String> status;  
 @FXML private TableColumn<OrderStr, Integer> price;  
 private Stage stage;  
  
 @FXML  
 void initialize(){  
 orderTime.setCellValueFactory(cellData -> cellData.getValue().getTime());  
 orderDate.setCellValueFactory(cellData -> cellData.getValue().getDate());  
 price.setCellValueFactory(cellData -> cellData.getValue().priceProperty().asObject());  
 status.setCellValueFactory(cellData -> cellData.getValue().statusProperty());  
 service.setCellValueFactory(cellData -> cellData.getValue().getServiceName());  
 client.setCellValueFactory(cellData -> cellData.getValue().getClientPhone());  
 fio.setCellValueFactory(cellData -> cellData.getValue().getClientFio());  
   
 }  
  
 public void setMain(Main main){  
 this.main = main;  
 }  
   
 @FXML  
 public void getAuthor(){  
 this.main.initAbout();  
 }  
 @FXML  
 public void getServices(){  
 this.main.initServices();  
 }  
 @FXML  
 public void getClients(){  
 this.main.initClients();  
 }  
  
 public void load() {  
 orderData = ServerRequestIOperator.getOrders();  
 stage.show();  
 if(orderData == null){  
 showAlert("Ошибка на сервере","Не удалось загрузить заказы");  
 }  
 else{  
 this.orderTable.setItems(orderData);  
 }  
 }  
 @FXML  
 public void exit(){  
 System.exit(0);  
 }  
 @FXML  
 public void create(){  
 this.main.initNewOrder(this);  
 }  
 @FXML  
 public void change(){  
 int selectedIndex = this.orderTable.getSelectionModel().getSelectedIndex();  
 if (selectedIndex >= 0) {  
 this.main.initEditOrder(this);  
 }  
 else{  
 showAlert("Не выбран заказ","Выберите заказ");  
 }  
 }  
  
 public TableView<OrderStr> getOrderTable() {  
 return orderTable;  
 }  
  
 @FXML  
 private void delete() {  
  
 int selectedIndex = this.orderTable.getSelectionModel().getSelectedIndex();  
 if (selectedIndex >= 0) {  
 Alert alert = new Alert(Alert.AlertType.CONFIRMATION);  
 alert.initOwner(main.getPrimaryStage());  
 alert.setHeaderText("Согласие на действие");  
 alert.setContentText("Вы уверены?");  
 alert.initModality(Modality.APPLICATION\_MODAL);  
 alert.titleProperty().setValue("Уверены?");  
 Optional<ButtonType> option = alert.showAndWait();  
 OrderStr order = this.orderTable.getSelectionModel().getSelectedItem();  
 if (option.get() == ButtonType.OK){  
 try{  
 ServerRequestIOperator.deleteOrder(order.getId());  
 Alert a = new Alert(Alert.AlertType.INFORMATION);  
 a.initOwner(main.getPrimaryStage());  
 a.setHeaderText("Успешно!");  
 a.setContentText("Успешное удаление заказа");  
 a.show();  
  
 }  
 catch(IOException e)  
 {  
 showAlert("Не удалось удалить договор","Сервер не отвечает");  
 }  
 this.load();  
 }  
 } else   
 {  
 showAlert("Не выбран заказ","Выберите заказ");  
 }  
 }  
  
 public void setStage(Stage primaryStage) {  
 this.stage = primaryStage;  
 }  
 @FXML public void getStat(){  
 this.main.initStat();  
 }  
  
 @FXML  
 public void newService()  
 {  
 this.main.initNewServiceCreation();  
 }  
  
 private void showAlert(String header, String message)  
 {  
 Alert alert = new Alert(Alert.AlertType.WARNING);  
 alert.initModality(Modality.APPLICATION\_MODAL);  
 alert.initOwner(main.getPrimaryStage());  
 alert.setHeaderText(header);  
 alert.setContentText(message);  
 alert.show();  
 }  
}

Main

package autoService;  
  
import autoService.Controllers.\*;  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.image.Image;  
import javafx.scene.layout.AnchorPane;  
import javafx.scene.layout.BorderPane;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
public class Main extends Application {  
 private Stage primaryStage;  
 private BorderPane rootLayout;  
 public static String title="Информационно автоматизированная система \"Автосервис\"";  
 public Image icon;  
 public Stage getPrimaryStage() {  
 return primaryStage;  
 }  
  
 public void start(Stage stage) {  
 primaryStage = stage;  
 icon=new Image(Main.class.getResource("/views/icon.png").toString());  
 primaryStage.getIcons().add(icon);  
 primaryStage.setTitle(title);  
 initRoot();  
 }  
  
 private void initRoot() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/Root.fxml"));  
 rootLayout = loader.load();  
 Scene scene = new Scene(rootLayout);  
 primaryStage.setScene(scene);  
 RootController controller = loader.getController();  
 controller.setMain(this);  
 controller.setStage(primaryStage);  
 controller.load();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 public void initAbout() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/Author.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 dialogStage.show();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void initServices() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/ServiceList.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 ServiceListController controller = loader.getController();  
 controller.setDialogStage(dialogStage);  
 controller.setMain(this);  
 controller.load();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 public void initClients() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/ClientList.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 ClientListController controller = loader.getController();  
 controller.setDialogStage(dialogStage);  
 controller.setMain(this);  
 controller.load();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private Stage getLoader(FXMLLoader loader) throws IOException {  
 AnchorPane rootLayout = loader.load();  
 Stage dialogStage = new Stage();  
 dialogStage.initOwner(this.primaryStage);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 dialogStage.setResizable(false);  
 dialogStage.initModality(Modality.APPLICATION\_MODAL);  
 Scene scene = new Scene(rootLayout);  
 dialogStage.setScene(scene);  
 return dialogStage;  
 }  
  
 public void initNewOrder(RootController root) {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/NewOrder.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 NewOrderController controller = loader.getController();  
 controller.setStage(dialogStage);  
 controller.setMain(this);  
 controller.setRoot(root);  
 controller.load();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void initEditOrder(RootController rootController) {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/EditOrder.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 EditOrderController controller = loader.getController();  
 controller.setStage(dialogStage);  
 controller.setMain(this);  
 controller.setRoot(rootController);  
 controller.load();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void initStat() {  
 try{  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/Stat.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
 StatInfoController controller = loader.getController();  
 controller.setStage(dialogStage);  
 controller.setMain(this);  
 controller.loadData();  
 }  
 catch(IOException e){  
 e.printStackTrace();  
 }  
  
 }  
  
 public void initNewServiceCreation() {  
 try{  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(Main.class.getResource("/views/NewService.fxml"));  
 Stage dialogStage = getLoader(loader);  
 dialogStage.setTitle(title);  
 dialogStage.getIcons().add(icon);  
   
 NewServiceController controller = loader.getController();  
 controller.setStage(dialogStage);  
 controller.setMain(this);  
 dialogStage.show();  
 }  
 catch(IOException e){  
 e.printStackTrace();  
 }  
  
 }  
  
}

PhoneValidator

package autoService.Support;  
  
public class PhoneValidator {  
 public static boolean isNormal(String number){  
 return number.matches("^(\\+7|7|8)?[\\s\\-]?\\(?[489][0**-**9]{2}\\)?[\\s\\-]?[0**-**9]{3}[\\s\\-]?[0**-**9]{2}[\\s\\-]?[0**-**9]{2}$");  
 }  
}