Федеральное государственное образовательное бюджетное

учреждение высшего образования

**«Финансовый университет при Правительстве Российской Федерации»**

**(Финансовый университет)**

Факультет информационных технологий и анализа больших данных

Департамент

анализа данных, принятия решений и финансовых технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ**

**на тему: «Информационно-справочная система магазина одежды»**

Направление подготовки 09.03.03 «Прикладная информатика»

Профиль: «ИТ-сервисы и технологии обработки данных в экономике и финансах»

Выполнил:

студент группы ПИ19-3

Александрова Дарья Игоревна

Научный руководитель:

кандидат технических наук, доцент

Петросов Давид Арегович

Работа защищена

с оценкой \_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись научного руководителя)

\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ 2021г.

**СОДЕРЖАНИЕ**

[**1.** **ВВЕДЕНИЕ** 2](#_Toc70685186)

[**2.** **ПОСТАНОВКА ЗАДАЧИ** 4](#_Toc70685187)

[**3.** **ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ** 5](#_Toc70685188)

[**4.** **АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ** 6](#_Toc70685189)

[**5.** **ОПИСАНИЕ ПРОГРАММЫ** 7](#_Toc70685190)

[**1.** **Алгоритмические решения** 7](#_Toc70685191)

[**2.** **Описание интерфейса программы** 9](#_Toc70685192)

[**3.** **СОСТАВ ПРИЛОЖЕНИЯ** 21](#_Toc70685193)

[**1.** **Cервер** 21](#_Toc70685194)

[**2.** **Клиент** 21](#_Toc70685195)

[**3.** **База данных** 22](#_Toc70685196)

[**4.** **НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ** 23](#_Toc70685197)

[**1.** **Сервер** 23](#_Toc70685198)

[**2.** **Клиент** 24](#_Toc70685199)

[**5.** **ЗАКЛЮЧЕНИЕ** 27](#_Toc70685200)

[**6.** **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ** 28](#_Toc70685201)

[**7.** **ПРИЛОЖЕНИЕ** 29](#_Toc70685202)

[**1.** **Реляционная модель** 29](#_Toc70685203)

[**2.** **Исходный код** 30](#_Toc70685204)

[**Сервер** 30](#_Toc70685205)

[**Клиент** 32](#_Toc70685206)

# **ВВЕДЕНИЕ**

Целью курсовой работы является приобретение навыков в анализе предметной области и в разработке кроссплатформенного приложения трехуровневой архитектуры на объектно-ориентированном языке программирования Java.

Анализ предметной области необходим при разработке. Он является не только первым этапом, но и главным звеном для автоматизации подходов в программировании к решению тех или иных задач. Важно создать хорошую информационную модель, которая станет крепкой основой для данного приложения.

Для трехуровневой структуры необходимо наличие трех компонентов: клиента, сервера приложений и баз данных. Клиент занимается хранением и обработкой информацией, а также соединением с сервером при помощи API запросов. Задача же сервера – получать данные и уметь обрабатывать их.

Разумеется, техническая документация является тоже важной частью программы, так как она определяет ее качество, поэтому приобретение навыков разработки документации текста имеет роль.

# **ПОСТАНОВКА ЗАДАЧИ**

В соответствии с выбранной темой нужно разработать информационную модель предметной области, представленной в виде таблиц базы данных и пользовательских классов. Кроме того, обязательно должен быть пользовательский интерфейс для удобной работы с клиентом. В ходе выполнения данной работы были поставлены следующие задачи:

* изучить литературу, заложив теоретическую основу темы исследования;
* изучить существующие программные продукты, относящихся к данной тематике;
* создать, соединить и протестировать три компонента информационно-учётной системы: базу данных, сервер и клиент;
* отображать в таблице данные предметной области;
* реализовать добавление, удаление и редактирование объектов в БД, а также фильтрацию данных по сложному критерию и последующую сортировку;
* создать пункт меню «об авторе»;
* разработать несколько полезных пользователю функций для отображения статистических данных;
* программа не должна завершаться «аварийно»;
* задокументировать код по стандарту JavaDoc.

# **ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

Одежда является важным изделием, выполняющим не только эстетическую, но и защитную роль от различных факторов внешней среды. На сегодняшний день она несет еще одну культурную ценность – моду. Именно благодаря большому выбору в магазинах различных моделей человек может самовыражаться. Массовые покупки составляют значительную долю оборота успешного бизнеса.

Разумеется, в любой компании необходимо контролировать убытки и доходы, чтобы выстраивать актуальные бизнес-стратегии. На сегодняшний день существует огромное количество различных приложений для магазинов розничной торговли.

Конечной целью данного проекта является разработка информационно-справочной системы, которая поможет не только работать с поставками и контролировать статус каждого товара, вести его учет, но и предоставлять аналитику, работать с клиентской базой, продавать товары и возвращать их. Большая часть задач сотрудников может обеспечиваться ИТ-технологиями.

Это приложение является удобным инструментом для работников организации, так как оно имеет возможность быстро изменять данные о товарах, создавать промокоды, присваивать клиентам покупки и генерировать чеки – главные исторические данные, с которыми впоследствии удобно вести аналитику.

# **АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ**

Для магазина важна не только посещаемость отделений, но и их прибыльность: для того, чтобы отследить это, нужно нажать на специальную кнопку «Статистика», которая высветит доход отделения по годам, месяцам и дням. С помощью этой функции менеджеры могут контролировать уровень поставок, чтобы окупать их при продаже. Почему данная система удобна? Не нужно вручную проводить расчеты и инвентаризацию, потому что у каждого товара есть свой статус и текущая информация в базе данных. Кроме того, генерация автоматических отчетов помогает сэкономить время.

В приложении существуют права доступа, так как оно рассчитано не только для менеджеров, но и для кассиров. Таким образом, автоматизация данного процесса помогает избежать случайных изменений данных неуполномоченными лицами.

При добавлении новой поставки требуется ввести артикул – это делается для автоматизации процесса, чтобы не вбивать одинаковую информацию. Если же товара с таким артикулом нет, то менеджеру предоставляется незаполненная форма для оформления информации. Благодаря этому алгоритму ускоряется процесс оформления новой поставки.

Поиск по складу осуществляется очень просто: достаточно ввести название или артикул. Кроме того, возможна сортировка по статусам товара. Автоматическая окраска рабочих промокодов в зеленый цвет помогает находить их намного быстрее.

Программа позволяет сократить непроизводительные, ручные затраты работников по учету товаров, поднять скорость обслуживания магазина. Таким образом, автоматизация этой предметной области является актуальной задачей.

# **ОПИСАНИЕ ПРОГРАММЫ**

## **Алгоритмические решения**

Сервер имеет реализацию RESTful API для взаимодействия с клиентом (используется Spring Boot). Клиент обращается к API при помощи запросов и имеет графический интерфейс, написанный с помощью Java FX. Хранение данных происходит в базе данных MySQL.

Стилизация окон выполнена с помощью связки fxml + css файлов. Каждому окну соответствует не только свой fxml-файл, но и java class controller, который реагирует на действия пользователя.

При запуске клиента отображается окно авторизации сотрудника – это обязательное требование для входа в систему, сверка паролей происходит с помощью хеширования. Если авторизация успешна, то происходит развертывание основной сцены.

При входе в учетную запись отображаются информационные элементы, хранящиеся в Label: название офиса, имя сотрудника и его должность, полученные во время авторизации GET-запросом. В центре окна расположен TableView – это чековая область при продаже товаров, отображающая текущую информацию покупки с учетом скидок. Сортировка данных доступна по каждому столбцу. К покупке можно прикреплять карту лояльности или промокод: поиск происходит по необходимым критериям с валидацией данных, после которой отправляется GET-запрос на сервер.

При добавлении товара к покупке используется Combobox для выбора размера товара. Нажатие на Label «New check» сбрасывает всю информацию о текущей покупке (удаляются объекты сессии чека). Сверху расположен Menu bar, состоящий из активных Label.

Программа сохраняет все измененные данные и отправляет их в базу данных при подтверждении. При выходе из учетной записи сессия завершается. При нажатии некоторых кнопок могут возникать alert, несущие различный характер: подтверждение, осведомление или предупреждение.

При занесении новых объектов в базу данных происходит первичная валидация на стороне клиента. Чаще всего используется ввод с клавиатуры в TextField при заполнении новой информации. Если же серверу не удается завершить запрос, то возникает alert-предупреждение.

Некоторые окна не будут доступны при отсутствии соединения с сервером. Приложение не будет завершаться в таком случае, отправление и получение новых данных будет недоступно, пока соединение не восстановится.

Кнопки menu bar, в основном, относятся к сущностям базы данных, такие как клиенты, промокоды, офисы или товары. Для обзора информации при наличии права доступа используется TableView, который удобно сортировать по многокритериальному параметру. В некоторых случаях фильтрация данных происходит с помощью ChoiceBox.

При создании нового промокода дата выбирается с помощью DatePicker – это удобное средство для упрощенной валидации. Если дата окончания раннее даты начала или менеджер не выбрал даты, то возникнет предупреждение.

Статистика реализована с помощью LineChart, стилизацию и типы данных которого настраивать очень удобно. Оси можно представлять не только в виде числовых значений, но и категориальных, например, название месяцев.

Все кнопки реализованы с помощью onAction() Label.

При добавлении новых экземпляров (например, товаров или клиентов) на сервер посылается POST-запрос с JSON-объектом. Важно отметить, что хранение исторических данных присутствует: при создании нового клиента также происходит POST-запрос к контроллеру ClientHistory, которому передаются данные не только о клиенте, но и о времени создания, работнике организации.

Все первичные ключи базы данных создаются автоматически, за исключением сущности промокодов (в данной таблице первичным ключем выступает название промокода).

Если же данные существующего объекта изменяются (например, статус товара переходит с «поставки» на «продажу»), то выполняется PUT-запрос на сервер.

Важно отметить, что как на стороне клиента, так и на стороне сервера, существует логирование действий.

При получении запроса от клиента, сервер сопоставляет переданный пусть URL с каким-либо из существующих контроллеров. Если такой ассоциации не существует, то будет возвращена ошибка 404 not found (так как логика клиента реализована, опираясь на существующие страницы, то таких проблем не возникнет), в ином случае Spring Boot сопоставляет переданные данные с аргументами. Если в ходе их обработки возникает проблема, то сервер вернёт ошибку 400 Bad Request. Если же все переданные данные корректны, то контроллер выполняет определенные действия с ними благодаря методам JPARepository и возвращает ответ клиенту.

## **Описание интерфейса программы**

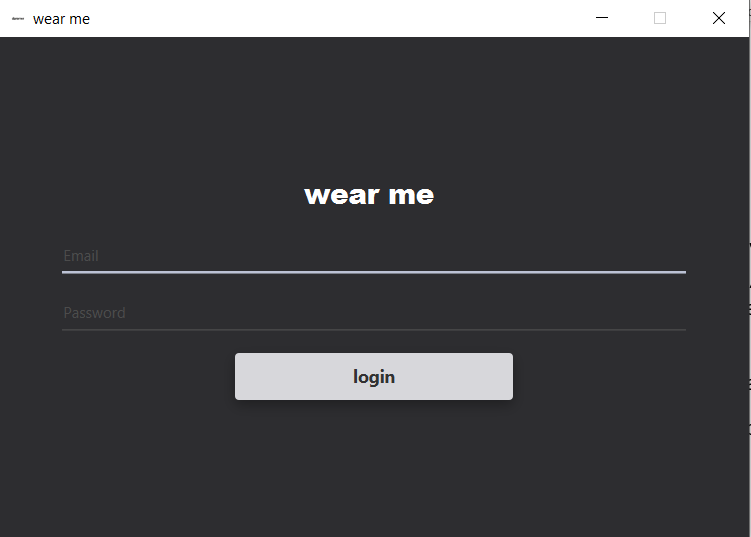
Так как у сервера нет интерфейса, то речь в данном разделе пойдет о клиенте. Элементы управления и компоненты, которые используются в приложении JavaFX: Label, AnchorPane, TableView, TextField, ChoiceBox, LineChart, DatePicker, JFXTextField, JFXPasswordField, JFXButton, Button, HBox, VBox. Для проектирования интерфейса использовался не только Java FX, но и специальная библиотека JFoenix для минималистичного дизайна, которая расширяет возможности по кастомизации стандартных элементов JavaFX разрабатываемой системы. Средства JFoenix были использованы для окна авторизации приложения. Всего в клиенте представлено 21 окно.

**Окно авторизации сотрудника**

Запрашивает логин и пароль работника. Если логина не существует или же хеш-код пароля не совпадает с хеш-кодом в базе данных, то вход не будет выполнен. При корректной авторизации происходит переход на главную страницу программы.

На форме существует следующие валидаторы входных данных:

* фильтрация на пустые поля e-mail и пароля (на стороне клиента);
* фильтрация на некорректную авторизацию по логину (на стороне сервера);
* фильтрация по некорректной паре логин/пароль (на стороне сервера).



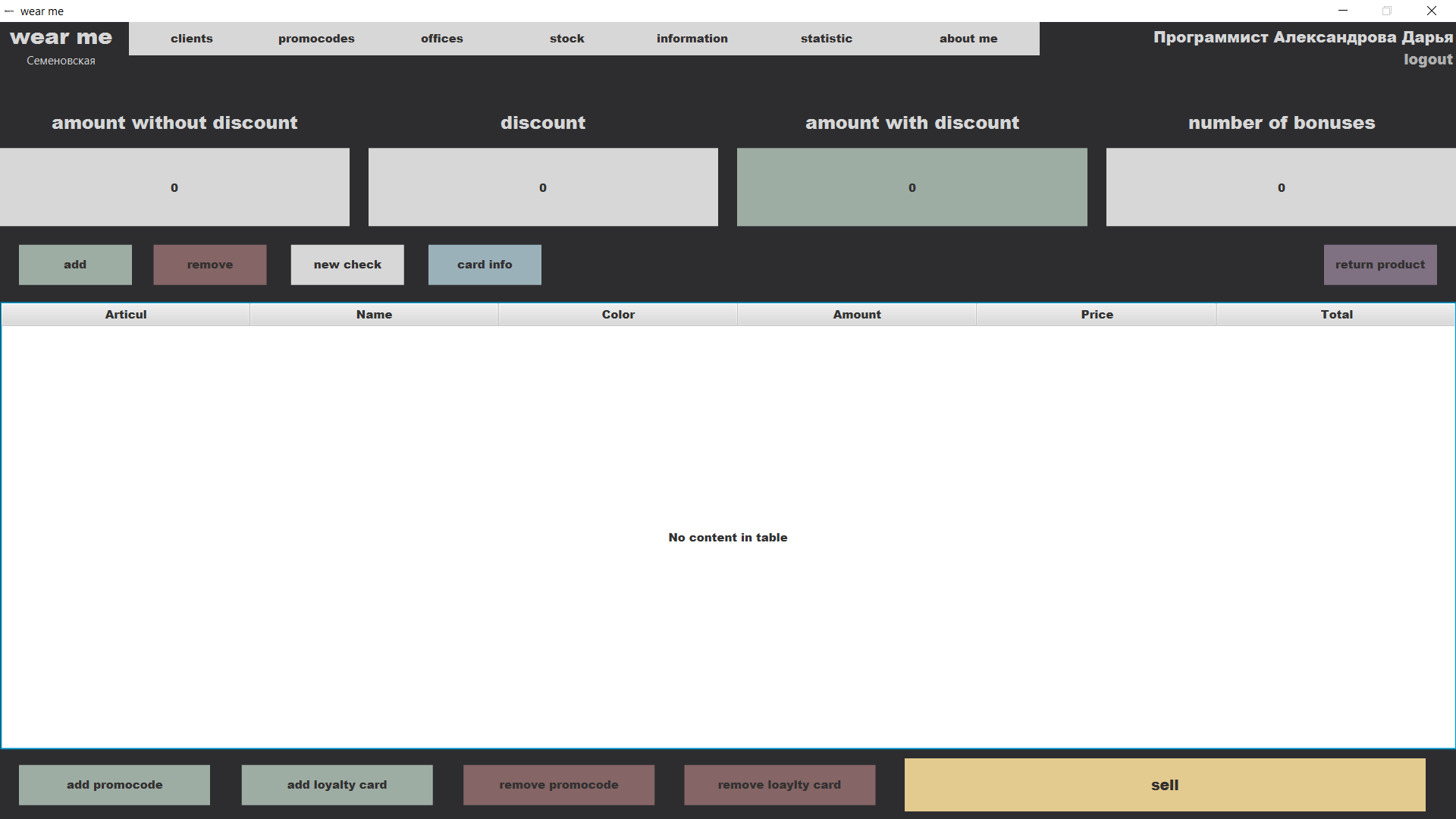
*Рисунок 1 - авторизация сотрудника*

**Рабочая область**

Основное окно, являющееся связующим звеном для остальных. Это главная информационная единица, на странице которой представлены:

* чековая область;
* кнопка добавления в чек;
* кнопка удаления из чека;
* создание нового чека;
* возврат товара;
* информация о карте клиента;
* добавление карты клиента;
* добавление промокода;
* удаление карты клиента;
* удаление промокода;
* продажа товара;
* кнопка «клиенты»;
* кнопка «офисы»;
* кнопка «склад»;
* кнопка «промокоды»;
* кнопка «информация»;
* кнопка «об авторе»;
* кнопка «статистика».

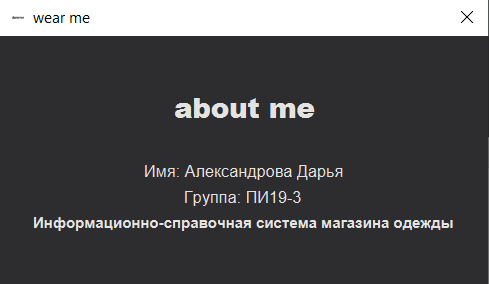
В левом углу расположено название магазина. Во время авторизации подтягиваются данные работника и офиса для детальной информации сессии. Сверху расположен меню-бар с информационными кнопками, посередине – чековая область для расчетов. Дизайн выполнен в минималистичном формате с опорой на несколько основных цветов.



*Рисунок 2 - Рабочая область приложения*

**Об авторе**

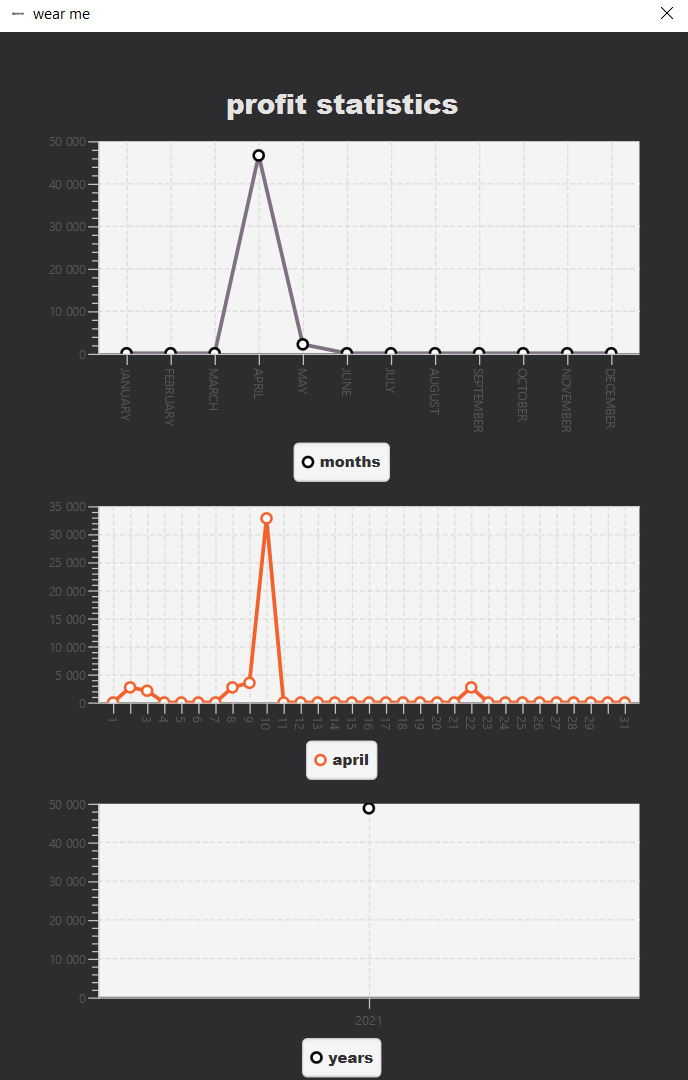
При открытии данного окна показывается основная информация о программе и авторе*:*

**

*Рисунок 3 - «Об авторе»*

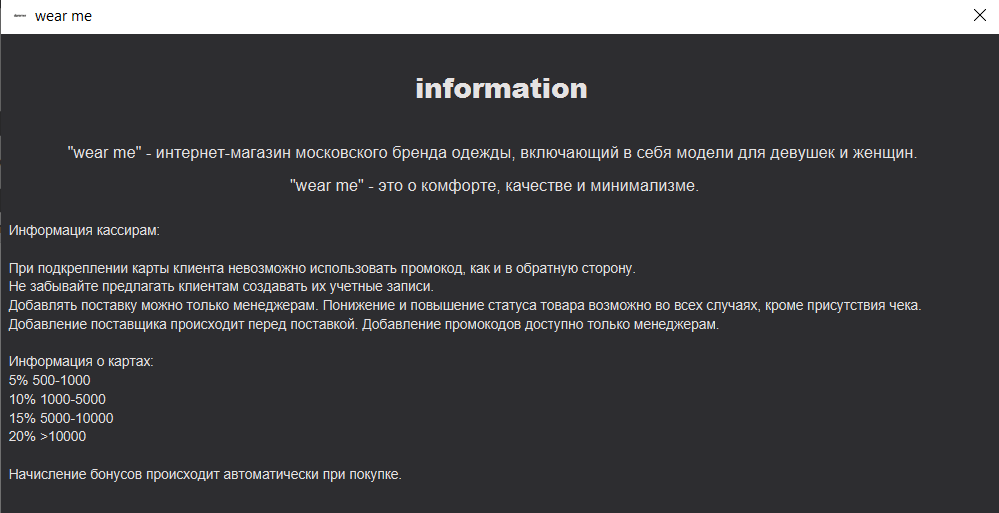
**Статистика**

О пользе данного окна было сказано уже неоднократно: знать статистику прибыльности отделения не только полезно, но и необходимо. Всего представлено 3 графика: по годам, по месяцам и по дням текущего месяца.

**

*Рисунок 4 - Статистика*

**Информация сотрудникам**

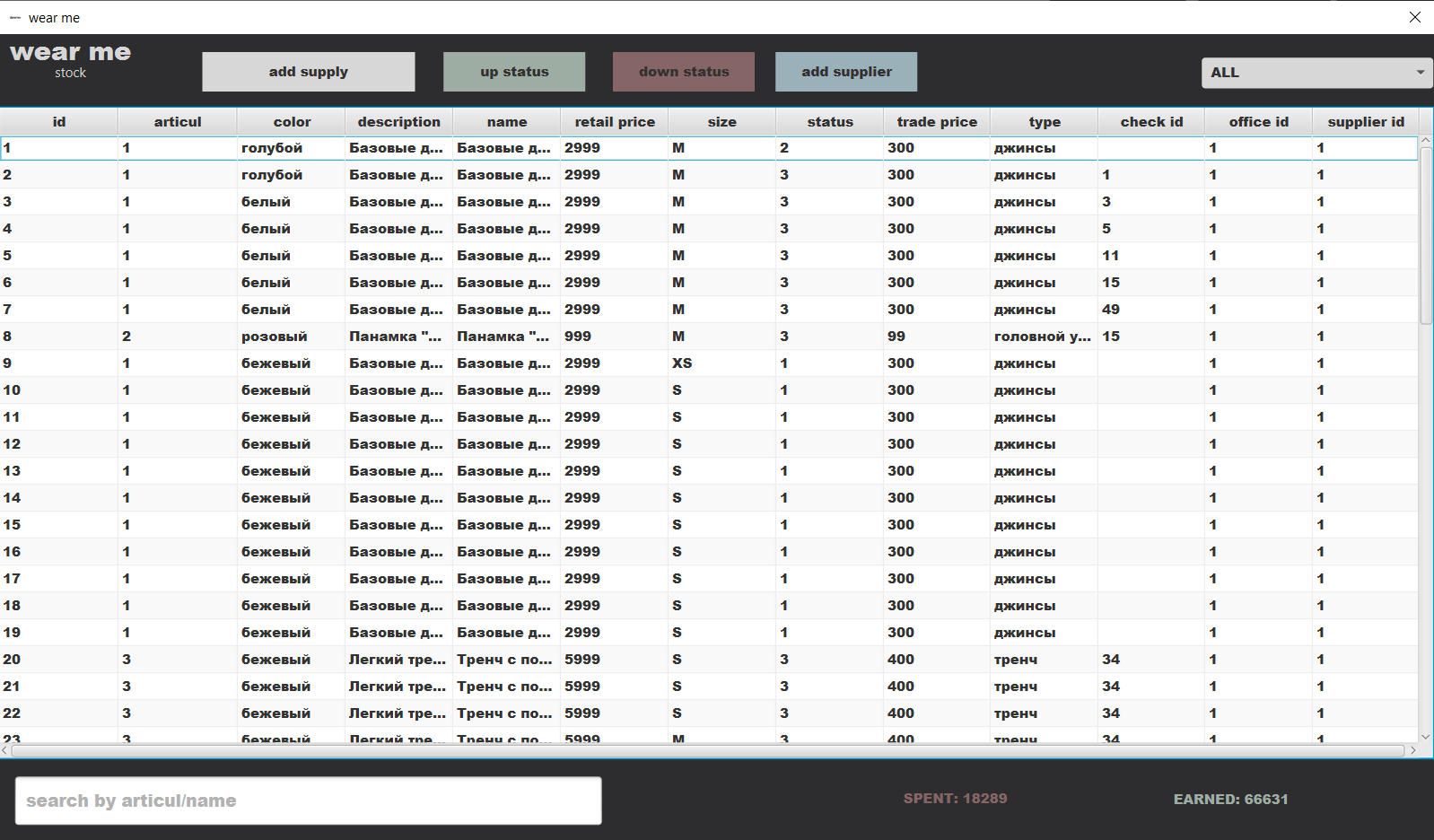
Данное окно предназначено для сотрудников организации, на нем представлены основные бизнес-правила:**

*Рисунок 5 - Информация сотрудникам*

**Информация о складе**

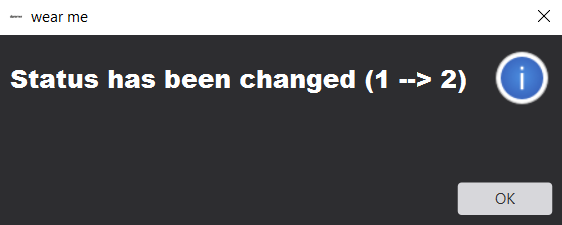
При открытии этого окна появляется таблица с текущей информацией по всем отделениям и товарам. Таким образом, можно просматривать наличие интересующего товара. С помощью поиска по артикулу или названию процесс займет меньше времени. Товары распределены по 3 группам: списаны, на складе и поставляются.

Менеджер имеет право повышать или понижать статус таким товарам, за исключением тех, которые уже были проданы. Товар, получающий статус 2(склад), автоматически попадает в оборот. В нижнем углу для удобства представлены суммы затрат и прибыли. Также на странице имеются вспомогательные кнопки для оформления новых поставок.



*Рисунок 6 - Окно «Склад»*

При изменении статуса товару сперва высветится предупреждение, после соглашения с которым операция успешно завершится:



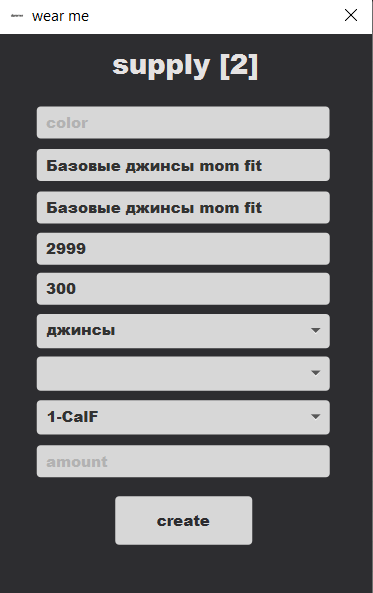
*Рисунок 7 - Изменение статуса товару*

**Добавление поставщика**

Данная функция доступна в окне «Информация о складе». При добавлении нового поставщика появляется возможность выбирать его при создании поставки.

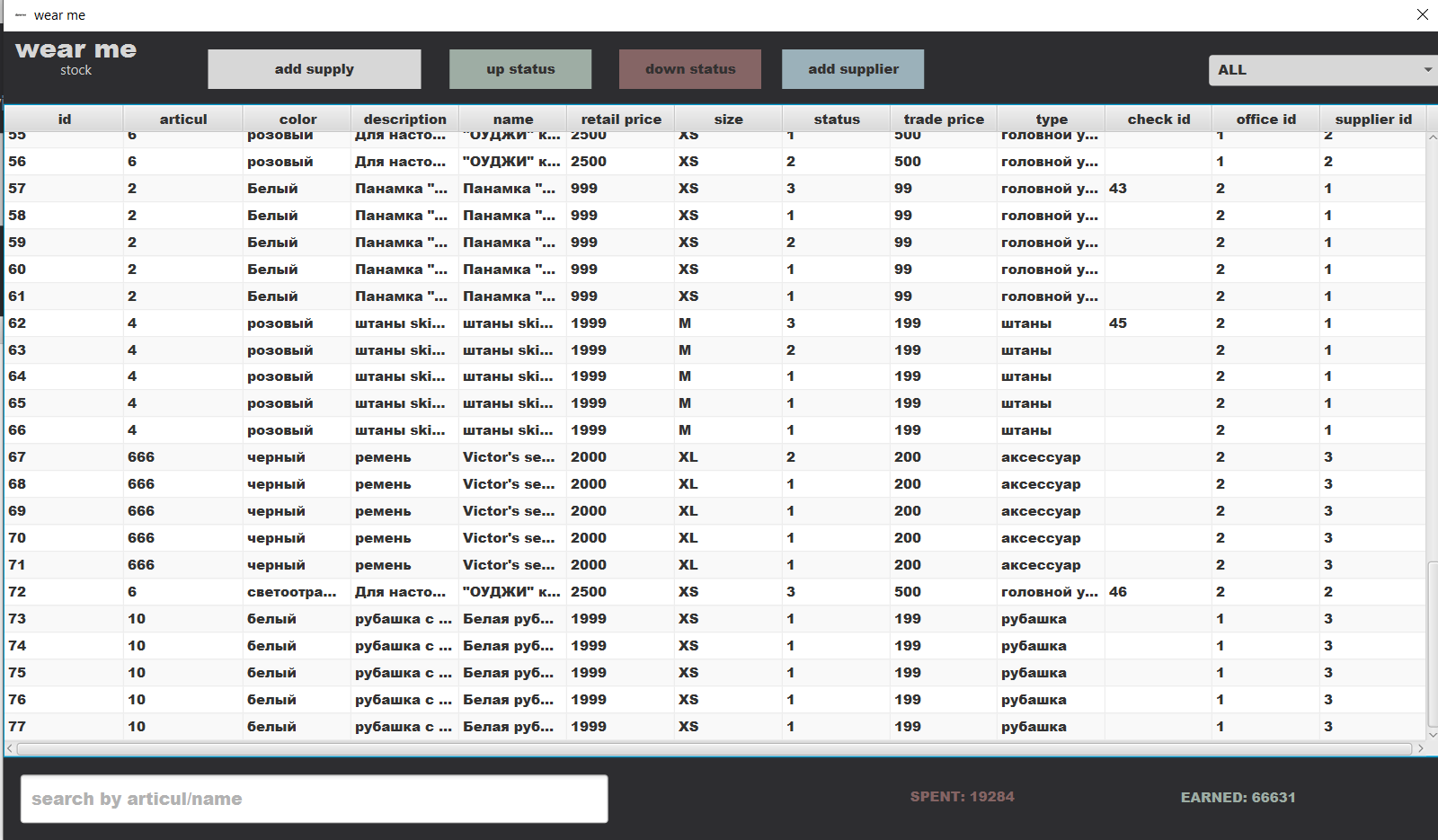
**Добавление поставки**

Данный процесс происходит в 2 этапа: сперва необходимо ввести артикул товара. Если текущий артикул уже есть в базе данных, то форма заполнится автоматически:

*Рисунки 9, 10- Автоматическое заполнение поставки*

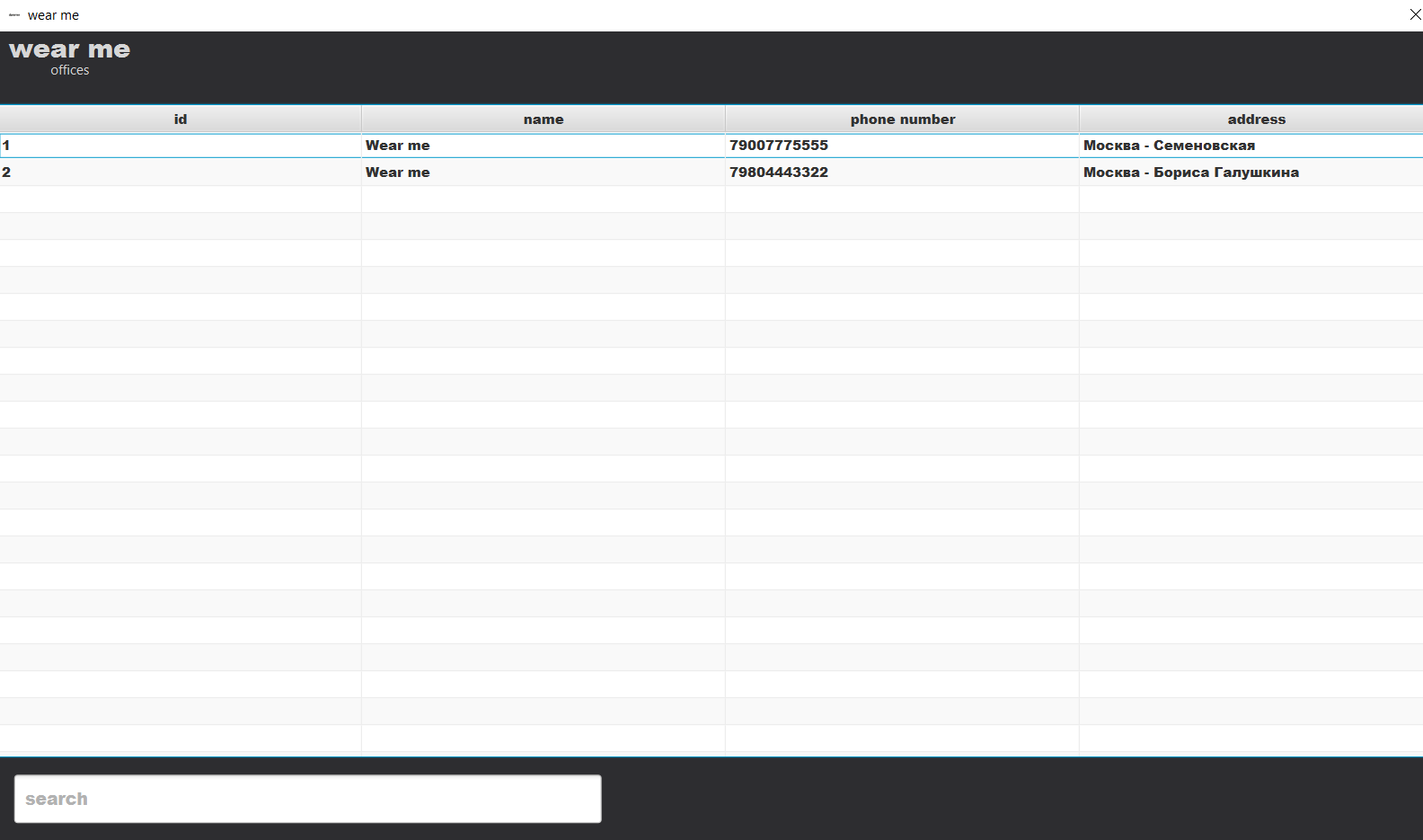
Попробуем добавить поставку с несуществующим артикулом и убедимся в том, что форма будет незаполнена. При успешном добавлении поставки товары появятся в таблице на складе:



*Рисунок 11 - Добавление новой поставки*

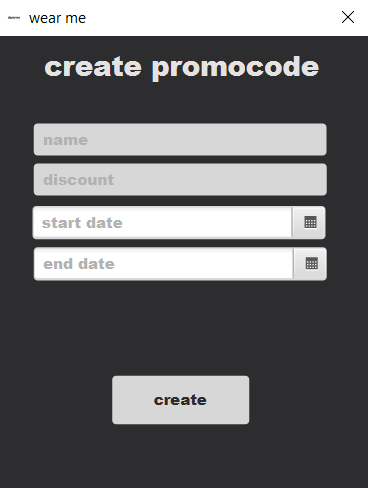
**Окно «Офисы»**

Данное окно является исключительно информационным для просмотра существующих отделений:



*Рисунок 12 - Окно «Офисы»***Окно «Создания промокода»**

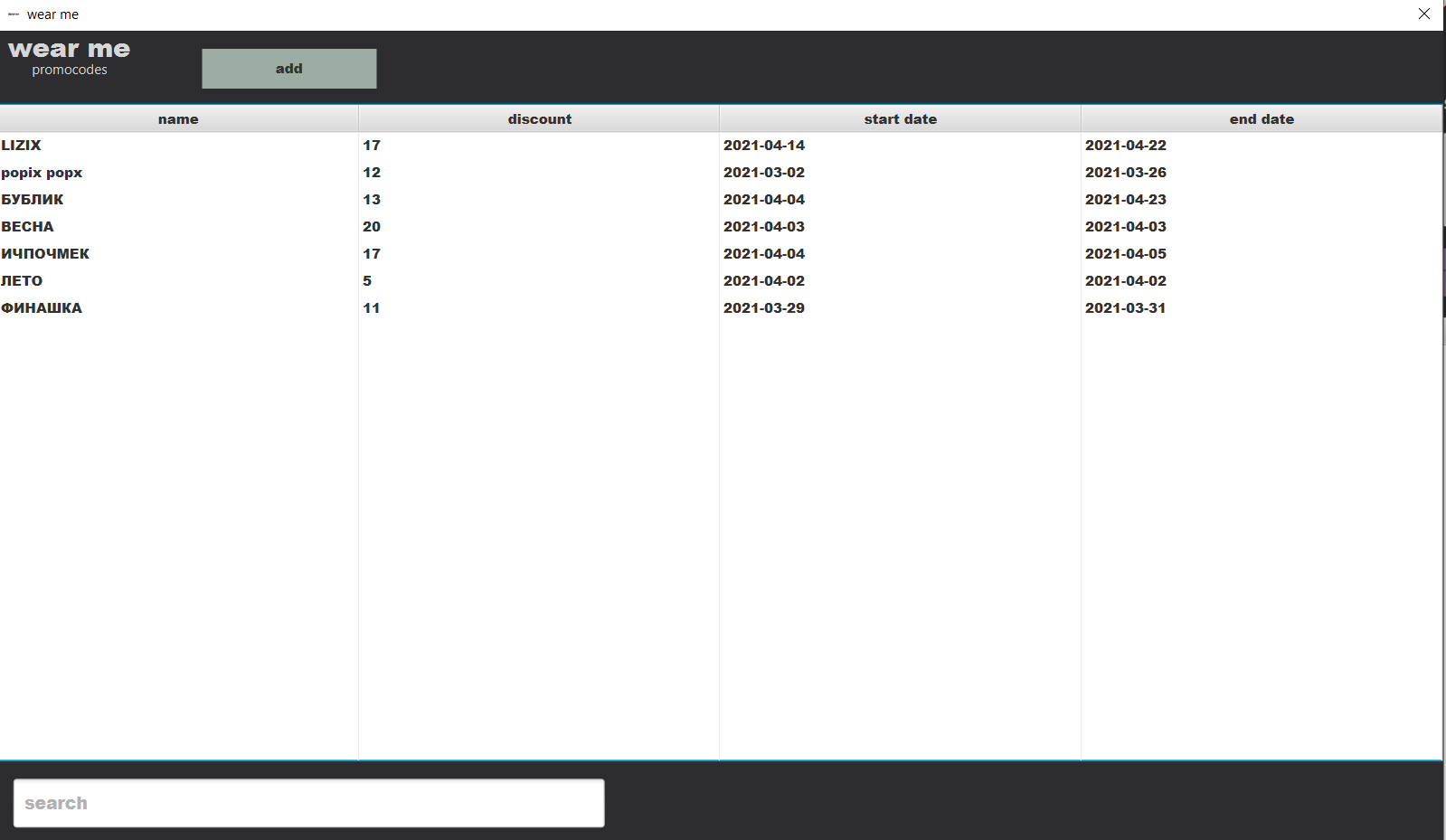
Для создания промокода нужно обладать специальными правами. Менеджеру необходимо ввести название, размер скидки и даты начала и окончания действия промокода. После успешного заполнения формы промокод будет добавлен в таблицу.



*Рисунок 13 – создание промокода*

**Окно «Промокоды»**

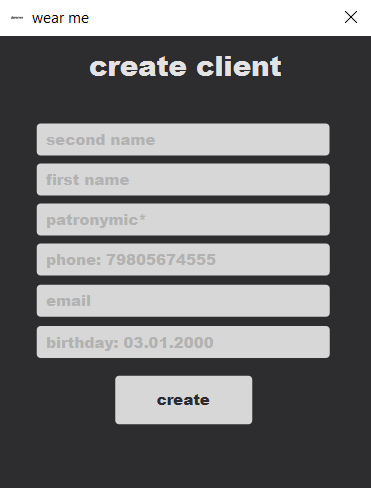
Предназначено для просмотра истории всех промокодов организации с доступом по праву менеджера. Можно заметить, что на данный момент таблица не имеет зеленых записей – это означает, что действующих промокодов не существует. На данной странице также существует умный поиск.



*Рисунок 14 - Окно «Промокоды»*

**Окно «Добавление клиента»**

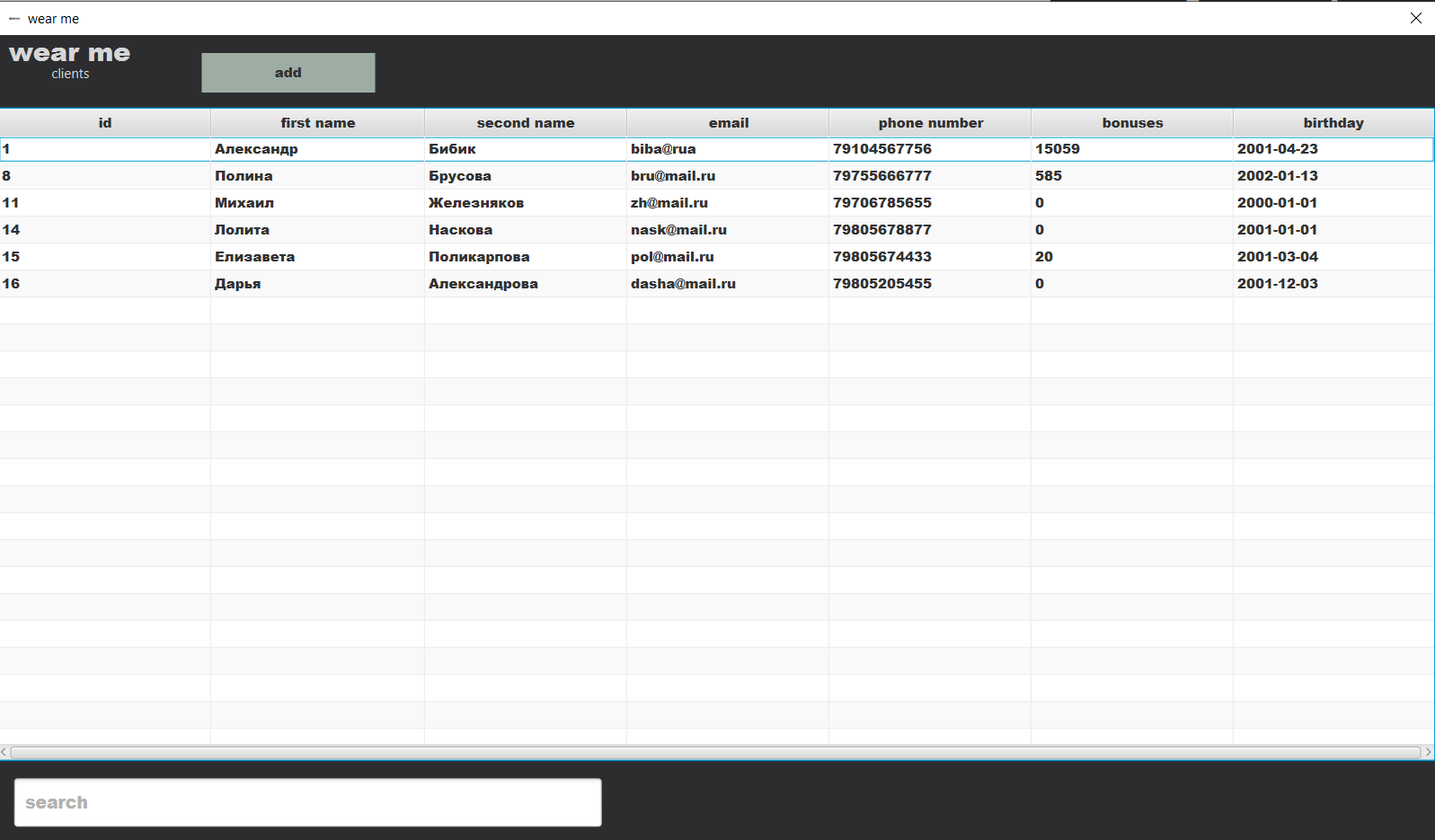
При успешном заполнении формы клиента и отсутствии его в базе данных появится новая запись в таблице. Эту карту лояльности можно использовать для начисления баллов при покупке.



*Рисунок 15 - Создание клиента*

**Окно «Клиенты»**

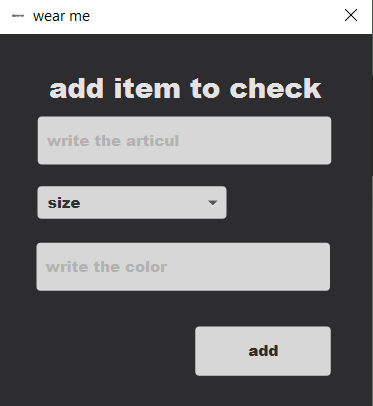
На данной странице предоставляется информация о клиентах организации с возможностью поиска и фильтрации по таблице. Нажатие на кнопку «создать» открывает новое окно оформления клиента.



*Рисунок 16 - Окно «Клиенты»*

**Окно «Добавление товара в чек»**

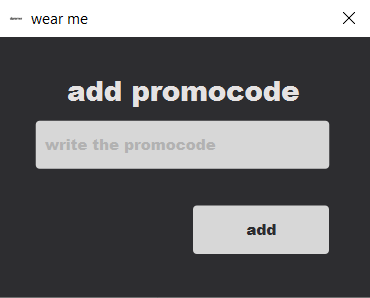
Данная функция используется для кассовых операций с клиентами. Таким образом, можно с легкостью прикрепить необходимый товар к чеку. Если же товара нет на складе текущего офиса с данными параметрами, то он не будет добавлен.



*Рисунок 17 – добавление товара в чек*

**Окно «Добавление промокода»**

Если покупатель знает рабочий промокод и хочет воспользоваться им, то можно ввести название.



*Рисунок 18 – добавление промокода*

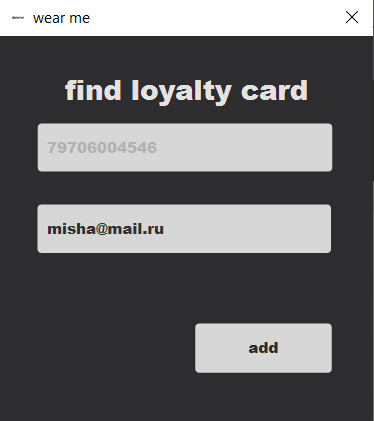
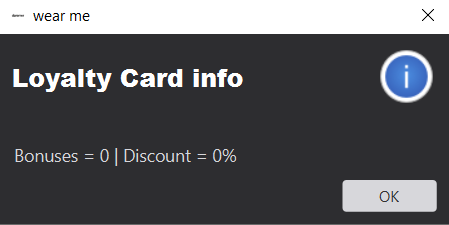
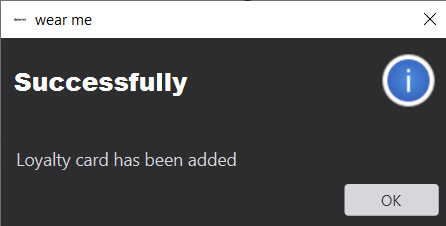
Если данные верны, то действие скидки вступит в силу. Использование одновременно карты лояльности и промокода недопустимо.



*Рисунок 19 - результат добавления промокода*

**Окно «Добавление карты лояльности»**

Если клиент зарегистрирован в базе, то к каждой покупке можно прикреплять его карту лояльности – начисление баллов происходит автоматически и гарантирует рост скидки с каждым разом с учетом бонусной системы. Поиск по базе происходит либо с помощью номера телефона, либо с помощью электронной почты.

*Рисунок 20 -добавление карты лояльности*

**Окно «Структура чека»**

Когда все товары были добавлены в итоговую таблицу расчетов, можно нажать на итоговую кнопку «продать». При удачном оформлении покупки на экране будет выведена основная информация. Возврат товаров по чеку доступен в течение месяца.



*Рисунок 21 -* Структура чека

**Окна «Оформление возврата»**

Оформление возврата происходит в несколько этапов. Сперва необходимо ввести идентификатор чека. Если такой будет найден в базе данных, то в новом окне высветится информация о текущих товарах, возврат которых возможен. Необходимо выбрать из таблицы интересующий товар и подтвердить операцию. В конце на экране появится окно об успешном возврате с подсчетом итоговой суммы.



*Рисунок 22 – успешный возврат*

# **СОСТАВ ПРИЛОЖЕНИЯ**

## **Cервер**

Сервер построен с помощью конфигурации Spring Boot, неявно упаковывающий необходимые зависимости и предоставляющий starter-пакеты. В состав сервера входят следующие компоненты:

* apache maven – основной сборщик проекта;
* spring-boot-maven-plugin – плагин для сборки решения;
* spring-data-jpa – решение для улучшения реализации слоя доступа к данным;
* mysql-connector-java - предоставляет возможность соединения с базой данных при помощи драйвера MySQL Connector, реализующий интерфейс API JDBC;
* spring Web для создания RESTful, с использованием Spring MVC.

## **Клиент**

В состав клиента входят следующие компоненты:

* apache maven – основной сборщик проекта для работы с библиотеками и плагинами;
* maven-shade-plugin - предоставляет возможность упаковать артефакт в uber-jar, включая его зависимости;
* maven-compiler-plugin – плагин для компиляции исходных файлов;
* javaFX – библиотека для отображения GUI и её компонентов. Выбор в ее пользу был сделан из-за возможности разделять дизайн и код, в отличие от Swing;
* jfoenix – библиотека для использования «кастомных» элементов интерфейса;
* org.json – библиотека для работы с запаковкой/распавкой данных в формате JSON;
* org.apache.logging.log4j – решение для логирования действий;
* commons-validator, com.vaadin.external.google – вспомогательные библиотеки для валидации данных.

## **База данных**

В качестве СУБД используется свободная реляционная система MySQL. Всего в таблице представлено 13 сущностей, атрибуты и связи которых отображены на реляционной модели (Приложение 1).

* addresses – таблица для хранения адресов;
* offices – таблица для хранения офисов;
* positions – таблица для хранения ролей работников;
* employees – таблица для хранения работников;
* clients – таблица для хранения клиентов;
* promocodes – таблица для хранения промокодов;
* client\_history – таблица для хранения истории создания клиентов;
* checks – таблица для хранения чеков;
* suppliers – таблица для хранения поставщиков;
* store\_products – таблица для хранения товаров;
* returns\_history – таблица для хранения истории возвратов;
* supply\_history – таблица для хранения истории поставок;
* clients\_checks – решение связи M:M между клиентами и чеками.

# **НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ**

## **Сервер**

* + **Пакет model:**

Каждый класс представляет собой сущность объекта в базе данных с реализацией связей и назначением атрибутов.

* + **Пакет controller:**

В данных классах реализуются Rest-запросы, такие как Post, Get, Put и Delete. Каждый контроллер назначается определенной сущности пакета model путем взаимодействия с методами JPARepository.

* + **Пакет repository:**

В состав входят репозитории каждой сущности, представляющие собой интерфейсы, которые обеспечивают основные операции по поиску, сохранению, удалению данных (CRUD операции).

* + **Пакет resources:**

Содержит ресурсы, необходимые для функционирования приложения. Например, application.properties помогает установить необходимые установки для базы данных.

**Описание классов**

* ShopApplication – точка входа в программу;
* Address, Check, Client, ClientHistory, Employee, Office, Position, Promocode, ReturnsHistory, StoreProduct, Supplier, SupplierHistory – сущности для представления в СУБД;
* AddressRepository, CheckRepository, ClientRepository, ClientHistoryRepository, EmployeeRepository, OfficeRepository, PositionRepository, PromocodeRepository, ReturnsHistoryRepository, StoreProductRepository, SupplieRepository, SupplierHistoryRepository – интерфейсы JPARepository для работы с коллекциями сущностей;
* AddressController, CheckController, ClientController, ClientHistoryController, EmployeeController, OfficeController, PositionController, PromocodeController, ReturnsHistoryController, StoreProductController, SupplierController, SupplierHistoryController – контроллеры для обработки http-запросов, связанных с конкретными сущностями;

## **Клиент**

* + **Пакет exception:**

Специальные классы, наследованные от Exception. Вызываются при особых ситуациях, возникающих в программе и описывающих их.

* + **Пакет controller:**

Содержит контроллеры, закрепляющиеся за определенными fxml-сценами. Реализуют основную логику взаимодействия с пользователем и последующей обработкой данных.

* + **Пакет api:**

Содержит единственный класс MyAPI, реализующий основную логику по взаимодействию с сервером и хранению данных сессии.

* + **Пакет utils:**

Содержит классы разного назначения для обособленных и повторяющихся задач.

* + **Пакет validator:**

Вспомогательный пакет для первичной валидации данных со стороны клиента при создании и обработки новых запросов.

* + **Пакет resources:**

Содержит ресурсы, использующиеся в приложении, такие как изображения, fxml-файлы, xml-файлы, css-стили, шрифты и т.д.

**Описание классов**

* JavaFXApplication – класс приложения Java FX;
* Main – точка входа в программу;
* CheckStructure,SupplierStructure,ProductStructure,OfficeStructure, ClientStructure,PromocodeStructure – решение по распаковке JSON-объектов сущностей и хранения с удобным назначением и доступом;
* MyLogger – класс, предназначенный для логирования действий;
* HTTPRequest – класс со статическими методами для http-запросов;
* HashUtil помогает хешировать данные в SHA-256;
* AlertInfo полезен для быстрого создания alert-предупреждений;
* NoAppDataException,NoClientException,NoColoException,NoOfficeException,NoPromocodeException,NoStoreProductException,NoSupplierException,ResponseStatusException,SellProductException,TimeOutPromocodeException – решения по подхвату исключительных ситуаций;
* MyAPI – реализует взаимодействие с сервером и хранение временных данных сессии;
* AddingClientController – контроллер-обработчик окна создания клиента;
* AddingLoyaltyCardController - контроллер-обработчик окна добавления карты лояльности;
* AddingPromocodeController - контроллер-обработчик окна добавления промокода;
* AddingPromocodeNewController - контроллер-обработчик окна создания промокода;
* AddingSupplierController - контроллер-обработчик окна создания поставщика;
* AddingSupply1Controller, AddingSupply2Controller- контроллеры-обработчики окна добавления поставки;
* AddingToCheckController – контроллер-обработчик добавления в чек;
* ClientController – контроллер-обработчик окна клиентов;
* CheckController - контроллер-обработчик окна чека;
* InfoController - контроллер-обработчик окна информации;
* OfficeController - контроллер-обработчик окна офисов;
* PromocodeController - контроллер-обработчик окна промокодов;
* ReturnController - контроллер-обработчик окна возврата;
* ReturnInfoController – контроллер-обработчик результата возврата;
* RootController – контроллер-обработчик главного окна;
* SignInController – контроллер-обработчик окна авторизации;
* StatController – контроллер-обработчик окна статистики;
* StockController – контроллер-обработчик склада;

# **ЗАКЛЮЧЕНИЕ**

В ходе написания данной работы было разработано клиент-серверное приложение для магазина одежды, позволяющее вести учет предприятия и упрощающее его функционирование.

Приложение полностью готово: клиент запакован в exe и jar форматы, что позволяет использовать его кроссплатформенно. Все методы решения могут генерировать автоматически составляемую документацию при помощи JavaDoc.

Созданное решение удовлетворяет всем поставленным задачам и требованиям. Программа не завершается аварийно, алгоритмы обработки данных эффективны и рациональны, соответствуют требованиям объектно-ориентированного и структурного программирования.

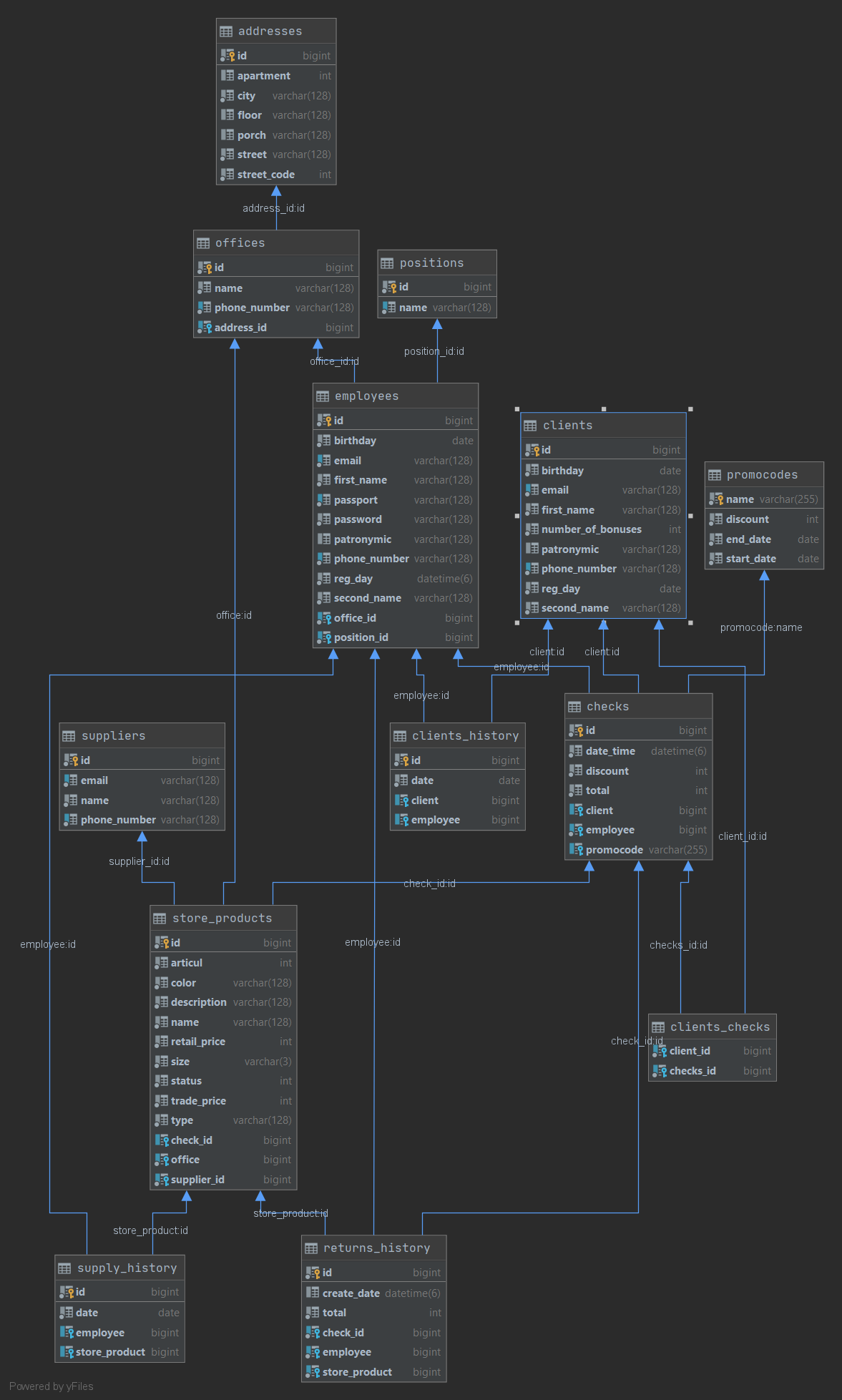
Программа может модернизироваться и обновляться. К примеру, возможность расширения списка клиентов, путем добавления новых объектов или окно регистрации сотрудников с помощью специального сайта. Возможно добавление новых видов аналитики и статистики, например, работоспособности каждого сотрудника. Наличие главного окна позволяет свободно подключать новые функции путем размещения кнопок.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Mapping в Spring Boot. 04.03.2021. URL: https://howtodoinjava.com/spring5/webmvc/controller-getmapping-postmapping/
2. Валидация данных Spring Boot. 04.04.2021. URL: https://www.baeldung.com/spring-boot-bean-validation
3. Request mapping в Spring Boot. 14.03.2021. URL: https://www.baeldung.com/spring-new-requestmapping-shortcuts
4. Building REST services with Spring // spring.io. 04.03.2021. URL: https://spring.io/guides/tutorials/rest/
5. Cascade Types примеры. 24.03.2021 URL: https://sysout.ru/tipy-cascade-primer-na-hibernate-i-spring-boot/
6. Вспомогательная информация по учету товаров на складе. 05.03.2021. URL: <https://klientiks.zendesk.com/hc/ru/articles/212809425-%D0%A3%D1%87%D0%B5%D1%82-%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D0%BE%D0%B2-%D0%BD%D0%B0-%D0%A1%D0%BA%D0%BB%D0%B0%D0%B4%D0%B5>
7. Конфигурация application.properties. 20.04.2021. URL: <https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html>

# **ПРИЛОЖЕНИЕ**

## **Реляционная модель**



## **Исходный код**

### **Сервер**

**ShopApplication:**

package wearme.server;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.transaction.annotation.EnableTransactionManagement;  
  
*/\*\*  
 \* Главный класс приложения  
 \*/*@SpringBootApplication  
@EnableTransactionManagement  
public class ShopApplication {  
 */\*\*  
 \* Точка входа в приложение  
 \** ***@param*** *args - аргументы  
 \*/* public static void main(String[] args) {  
 SpringApplication.*run*(ShopApplication.class, args);  
 }  
  
}

**Пример из пакета repository(остальные имеют схожую структуру). AddressRepository:**

package wearme.server.repository;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import wearme.server.model.Address;  
*/\*\*  
 \* Интерфейс JpaRepository для работы с адресами  
 \*/*@Repository  
public interface AddressRepository extends JpaRepository<Address,Long> {  
 */\*\*  
 \* Получает адрес по идентификатору  
 \** ***@param*** *id - идентификатор  
 \** ***@return*** *address  
 \*/* Address findAddressById(Long id);  
}

**Пример из пакета model(остальные имеют схожую структуру). Supplier:**

package wearme.server.model;  
import javax.persistence.\*;  
*/\*\*  
 \* Сущность поставщиков  
 \*/*@Entity  
@Table(name = "suppliers")  
public class Supplier {  
 */\*\*  
 \* Получает идентификатор  
 \* @return id  
 \*/* public Long getId() {  
 return id;  
 }  
  
 */\*\*  
 \* Получает имя  
 \* @return name  
 \*/* public String getName() {  
 return name;  
 }  
  
 */\*\*  
 \* Получает номер телефона  
 \* @return phoneNumber  
 \*/* public String getPhoneNumber() {  
 return phoneNumber;  
 }  
  
 */\*\*  
 \* Получает электронную почту  
 \* @return email  
 \*/* public String getEmail() {  
 return email;  
 }  
  
 */\*\*  
 \* Конструктор для поставщиков  
 \* @param name - имя поставщика  
 \* @param phoneNumber - телефонный номер  
 \* @param email - электронная почта  
 \*/* public Supplier(String name, String phoneNumber, String email){  
 this.name = name;  
 this.phoneNumber = phoneNumber;  
 this.email = email;  
 };  
 */\*\*  
 \* Инициализация класса  
 \*/* public Supplier(){};  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id", nullable = false)  
 private Long id;  
  
 @Column(name = "name", length = 128, nullable = false)  
 private String name;  
  
 @Column(name = "phoneNumber", length = 128, nullable = false,unique = true)  
 private String phoneNumber;  
 @Column(name = "email", length = 128, nullable = false,unique = true)  
 private String email;  
}

**Пример из пакета controller(остальные имеют схожую структуру). SupplyHistoryController:**

package wearme.server.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import wearme.server.model.SupplyHistory;  
import wearme.server.repository.SupplyHistoryRepository;  
  
*/\*\*  
 \* Рест-контроллер для SupplyHistoryRepository  
 \*/*@RestController  
@RequestMapping("shop")  
public class SupplyHistoryController {  
 private final SupplyHistoryRepository supplyHistoryRepository;  
  
 */\*\*  
 \* Инициализирует контроллер истории поставок  
 \** ***@param*** *supplyHistoryRepository - контроллер истории поставок  
 \*/* @Autowired public SupplyHistoryController(SupplyHistoryRepository supplyHistoryRepository){  
 this.supplyHistoryRepository = supplyHistoryRepository;  
 }  
  
 */\*\*  
 \* Создает новую историю поставки  
 \** ***@param*** *supplyHistory - сущность истории поставки  
 \** ***@return*** *SupplyHistory  
 \*/* @PostMapping("/supplyHistory")  
 SupplyHistory SupplyHistory(@RequestBody SupplyHistory supplyHistory){  
 return supplyHistoryRepository.save(supplyHistory);  
 }  
}

### **Клиент**

**MyAPI:**

package client.api;  
  
import client.JavaFXApplication;  
import client.exception.\*;  
import client.utils.\*;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
import java.io.IOException;  
import java.net.URLEncoder;  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.util.\*;  
import static java.time.temporal.ChronoUnit.*DAYS*;  
  
*/\*\*  
 \* АПИ приложения  
 \*/*public class MyAPI {  
 private final JavaFXApplication mainApp;  
 private final ObservableList<CheckStructure> checkData = FXCollections.*observableArrayList*();  
 private final HashMap<Long,ArrayList<String>> ListOfProducts = new HashMap<>();  
 private Long employeeId;  
 private Long officeId;  
 private Long addressId;  
 private Long clientId;  
 private Long checkId;  
 private Integer currentBonuses = 0;  
 private Integer bonuses = 0;  
 private final List<String> listOfColors = new ArrayList<>();  
 */\*\*  
 \* json адреса  
 \*/* public JSONObject jsonAddress= new JSONObject();  
 */\*\*  
 \* json работника  
 \*/* public JSONObject jsonEmployee= new JSONObject();  
 */\*\*  
 \* json офиса  
 \*/* public JSONObject jsonOffice= new JSONObject();  
 private String position;  
 private JSONObject jsonClient= new JSONObject();  
 private JSONObject jsonPromocode = new JSONObject();  
 private JSONArray jsonStoreProducts = new JSONArray();  
 private JSONObject updatedCheck = new JSONObject();  
 private ObservableList<ProductStructure> productData = FXCollections.*observableArrayList*();  
 private JSONArray jsonProductData = new JSONArray();  
 private String returnResult;  
 private String totalResult;  
 private JSONArray jsonClients = new JSONArray();  
 private JSONArray jsonPromocodes = new JSONArray();  
 private JSONArray jsonOffices = new JSONArray();  
 private JSONArray jsonProducts = new JSONArray();  
 private JSONArray articulProducts = new JSONArray();  
 private JSONArray jsonSuppliers = new JSONArray();  
  
 */\*\*  
 \* Очищает информацию о всех сохраненных товарах во время сессии  
 \*/* public void deleteAllProducts(){  
 this.checkData.clear();  
 this.ListOfProducts.clear();  
 this.listOfColors.clear();  
 this.jsonClient = new JSONObject();  
 this.jsonPromocode = new JSONObject();  
 this.jsonStoreProducts = new JSONArray();  
 this.updatedCheck = new JSONObject();  
 this.bonuses = 0;  
 this.currentBonuses = 0;  
 this.clientId = null;  
 }  
  
 */\*\*  
 \* Получает JSON-объект офиса  
 \** ***@return*** *jsonOffice  
 \*/* public JSONObject getJsonOffice() {  
 return jsonOffice;  
 }  
  
 */\*\*  
 \* Получает json-список поставщиков  
 \** ***@return*** *jsonSuppliers  
 \*/* public JSONArray getJsonSuppliers() {  
 return jsonSuppliers;  
 }  
  
 */\*\*  
 \* Получает количество текущих бонусов  
 \** ***@return*** *currentBonuses  
 \*/* public Integer getCurrentBonuses() {  
 return currentBonuses;  
 }  
  
 */\*\*  
 \* Устанавливает текущее количество бонусов  
 \** ***@param*** *currentBonuses - текущие бонусы  
 \*/* public void setCurrentBonuses(Integer currentBonuses) {  
 this.currentBonuses = currentBonuses;  
 }  
  
 */\*\*  
 \* Получает бонусы  
 \** ***@return*** *bonuses  
 \*/* public Integer getBonuses() {  
 return bonuses;  
 }  
  
 */\*\*  
 \* Получает идентификатор клиента  
 \** ***@return*** *clientId  
 \*/* public Long getClientId() {  
 return clientId;  
 }  
  
 */\*\*  
 \* Получает информацию чека  
 \** ***@return*** *checkData  
 \*/* public ObservableList<CheckStructure> getCheckData() {  
 return checkData;  
 }  
  
 */\*\*  
 \* Устанавливает идентификатор работнику  
 \** ***@param*** *employeeId - идентификатор работника  
 \*/* public void setEmployeeId(Long employeeId) {  
 this.employeeId = employeeId;  
 }  
  
 */\*\*  
 \* Получает идентификатор работника  
 \** ***@return*** *employeeId  
 \*/* public Long getEmployeeId() {  
 return employeeId;  
 }  
  
 */\*\*  
 \* Получает информацию о сотруднике  
 \** ***@return*** *position+secondName+firstName  
 \** ***@throws*** *JSONException - при работе с JSON-объектом  
 \*/* public String getEmployeeInfo() throws JSONException {  
 return position+" "+jsonEmployee.get("secondName").toString()+" "+jsonEmployee.get("firstName").toString();  
 }  
  
 */\*\*  
 \* Получает сумму чека  
 \** ***@return*** *int - сумма чека  
 \*/* public int getTotalPrice(){  
 int z = 0;  
 for(CheckStructure i: checkData){  
 z+=i.getTotal();  
 }  
 return z;  
 }  
  
 */\*\*  
 \* Получает сумму чека с учетом скидки  
 \** ***@return*** *int - сумма чека с учетом скидки  
 \** ***@throws*** *JSONException - при работе с JSON-объектом  
 \*/* public int getPriceWithDiscount() throws JSONException {  
 int disc = this.getPromocodeDiscount();  
 int total = 0;  
 if (disc==0){  
 disc = this.getLoyaltyDiscount();  
 if(disc!=0){  
 for(CheckStructure i: checkData){  
 total+=i.getAmount()\*i.getPrice()\*disc/100;  
 }  
 }  
 }  
 else{  
 for(CheckStructure i: checkData){  
 total+=i.getAmount()\*i.getPrice()\*disc/100;  
 }  
 }  
 return total;  
 }  
  
 */\*\*  
 \* Удаляет продукт из сессии  
 \** ***@param*** *articul - артикул  
 \** ***@param*** *color - цвет  
 \** ***@throws*** *JSONException - при работе с JSON-объектом  
 \*/* public void deleteProduct(Integer articul,String color) throws JSONException {  
 for(CheckStructure i: checkData){  
 if(i.getArticul() == articul & i.getColor().equals(color)){  
 if(i.getAmount() == 1){  
 checkData.remove(i);  
 }  
 else{  
 i.setAmount(i.getAmount()-1);  
 i.setTotal(i.getAmount()\*i.getPrice());  
 }  
 for(Map.Entry<Long, ArrayList<String>> a: ListOfProducts.entrySet()){  
 if(a.getValue().get(0).equals(articul.toString()) & a.getValue().get(1).equals(color)){  
 this.ListOfProducts.remove(a.getKey(),a.getValue());  
 break;  
 }  
 }  
 break;  
 }  
 }  
 for(int j=0;j<this.jsonStoreProducts.length();j++){  
 if(this.jsonStoreProducts.getJSONObject(j).getInt("articul") == articul  
 & this.jsonStoreProducts.getJSONObject(j).getString("color").equals(color)){  
 this.jsonStoreProducts.remove(j);  
 break;  
 }  
 }  
 System.*out*.println(this.jsonStoreProducts.toString());  
 }  
  
*...*  
}

**Пример из пакета controller(остальные имеют схожую структуру). OfficeController:**

package client.controller;  
  
import client.JavaFXApplication;  
import client.api.MyAPI;  
import client.exception.NoOfficeException;  
import client.utils.OfficeStructure;  
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.\*;  
import javafx.stage.Stage;  
import org.json.JSONException;  
  
/\*\*  
 \* Контроллер офиса  
 \*/  
public class OfficeController {  
 private Stage stage;  
 @FXML private TableView<OfficeStructure> officeTable;  
 @FXML private TextField textField;  
 @FXML private TableColumn<OfficeStructure, Long> idColumn;  
 @FXML private TableColumn<OfficeStructure, String> nameColumn;  
 @FXML private TableColumn<OfficeStructure,String> phoneColumn;  
 @FXML private TableColumn<OfficeStructure,String> addressColumn;  
  
 private JavaFXApplication mainApp;  
  
 private ObservableList<OfficeStructure> officeData = FXCollections.observableArrayList();  
 private FilteredList<OfficeStructure> filteredData = new FilteredList<>(officeData);  
 private SortedList<OfficeStructure> sortableData = new SortedList<>(filteredData);  
 private MyAPI API;  
 /\*\*  
 \* Устанавливает значение сцены  
 \* @param dialogStage - сцена  
 \*/  
 public void setStage(Stage dialogStage) {  
 this.stage = dialogStage;  
 }  
 /\*\*  
 \* Устанавливает значение API  
 \* @param API - апи  
 \*/  
 public void setAPI(MyAPI API) {  
 this.API = API;  
 }  
 /\*\*  
 \* Устанавливает значение главному приложению  
 \* @param mainApp - главное приложение  
 \*/  
 public void setMainApp(JavaFXApplication mainApp) {  
 this.mainApp = mainApp;  
 }  
  
 /\*\*  
 \* Инициализация первичных объектов контроллера  
 \*/  
 @FXML  
 public void initialize() {  
 idColumn.setCellValueFactory(cellData -> cellData.getValue().idProperty().asObject());  
 nameColumn.setCellValueFactory(cellData -> cellData.getValue().nameProperty());  
 phoneColumn.setCellValueFactory(cellData -> cellData.getValue().phoneProperty());  
 addressColumn.setCellValueFactory(cellData -> cellData.getValue().addressProperty());  
 textField.textProperty().addListener((observable, oldValue, newValue) ->  
 filteredData.setPredicate(p -> p.getName().toLowerCase().contains(textField.getText().toLowerCase()) |  
 p.getAddress().toLowerCase().contains(textField.getText().toLowerCase()) |  
 p.getPhone().toLowerCase().contains(textField.getText().toLowerCase())  
 )  
 );  
 }  
  
 /\*\*  
 \* Загружает данные об офисах  
 \* @throws JSONException - при работе с JSON-объектом  
 \* @throws NoOfficeException - при работе с информацией об офисах  
 \*/  
 public void loadData() throws JSONException, NoOfficeException {  
 this.officeData = this.API.getOffices();  
 this.filteredData = new FilteredList<>(officeData);  
 this.sortableData = new SortedList<>(this.filteredData);  
 this.officeTable.setItems(this.sortableData);  
 this.sortableData.comparatorProperty().bind(this.officeTable.comparatorProperty());  
 }  
}

**Пример из пакета exception(остальные имеют схожую сотруктуру). NoAppDataException:**

package client.exception;  
  
import client.JavaFXApplication;  
import client.utils.MyLogger;  
  
*/\*\*  
 \* Исключение при отсутствии данных  
 \*/*public class NoAppDataException extends Exception{  
 */\*\*  
 \* Инициализирует исключение  
 \* @param mainApp - главное приложение  
 \* @param errorMessage - сообщение  
 \*/* public NoAppDataException(JavaFXApplication mainApp, String errorMessage){  
 super(errorMessage);  
 MyLogger.*logger*.error(errorMessage);  
 }  
}

**Main:**

package client;  
  
*/\*\*  
 \* Главный класс  
 \*/*public class Main {  
 */\*\*  
 \* Точка входа в приложение  
 \* @param args - аргументы  
 \*/* public static void main(String[] args) {  
 JavaFXApplication.*main*(args);  
 }  
}

**NumberValidator:**

package client.validator;  
  
*/\*\*  
 \* Валидация телефонных номеров  
 \*/*public class NumberValidator {  
 */\*\*  
 \* Проверяет, является ли строка номером телефона  
 \** ***@param*** *number - номер телефона  
 \** ***@return*** *true/false  
 \*/* public static boolean isNumber(String number){  
 String regex = "^((8|\\+7)[\\- ]?)?(\\(?\\d{3,4}\\)?[\\- ]?)?[\\d\\- ]{5,10}$";  
 return number.matches(regex);  
 }  
  
 */\*\*  
 \* Проверяет, является ли строка русским номером телефона  
 \** ***@param*** *number - номер телефона  
 \** ***@return*** *true/false  
 \*/* public static boolean isRussianNumber(String number){  
 //длина 11  
 if (number.length() ==11 & number.charAt(0) == '7'  
 ){  
 try{  
 Long.*parseLong*(number.substring(2));  
 return true;  
 }  
 catch(NumberFormatException e){  
 return false;  
 }  
 }  
 return false;  
 }  
}

**SupplierStructure:**

package client.utils;  
  
import javafx.beans.property.LongProperty;  
import javafx.beans.property.SimpleLongProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.property.StringProperty;  
  
*/\*\*  
 \* Класс для структуризации поставщиков  
 \*/*public class SupplierStructure {  
 */\*\*  
 \* StringProperty название  
 \*/* private final StringProperty name;  
 */\*\*  
 \* LongProperty идентификатор  
 \*/* private final LongProperty id;  
  
 */\*\*  
 \* Инициализирует данные поставщика  
 \** ***@param*** *id - идентификатор  
 \** ***@param*** *name - название  
 \*/* public SupplierStructure(Long id,String name){  
 this.id = new SimpleLongProperty(id);  
 this.name = new SimpleStringProperty(name);  
 }  
  
 */\*\*  
 \* Получает название  
 \** ***@return*** *name  
 \*/* public String getName() {  
 return name.get();  
 }  
  
 */\*\*  
 \* Получает nameProperty  
 \** ***@return*** *nameProperty  
 \*/* public StringProperty nameProperty() {  
 return name;  
 }  
  
 */\*\*  
 \* Устанавливает название  
 \** ***@param*** *name - название  
 \*/* public void setName(String name) {  
 this.name.set(name);  
 }  
  
 */\*\*  
 \* Получает id  
 \** ***@return*** *id  
 \*/* public long getId() {  
 return id.get();  
 }  
  
 */\*\*  
 \* Устанавливает id  
 \** ***@param*** *id - идентификатор  
 \*/* public void setId(long id) {  
 this.id.set(id);  
 }  
}

**HTTPRequest:**

package client.utils;  
import org.json.JSONObject;  
  
import java.io.\*;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.nio.charset.StandardCharsets;  
  
*/\*\*  
 \* Класс для работы с http-запросами  
 \*/*public class HTTPRequest {  
 */\*\*  
 \* Реализует GET-обращение к серверу  
 \** ***@param*** *urlString - адрес  
 \** ***@return*** *sb - полученные данные  
 \*/* public static String Get(String urlString) {  
 try {  
 URL url = new URL(urlString);  
 HttpURLConnection conn = (HttpURLConnection) url.openConnection();  
 conn.setRequestMethod("GET");  
 {  
 StringBuilder sb = new StringBuilder();  
 InputStream is = new BufferedInputStream(conn.getInputStream());  
 BufferedReader br = new BufferedReader(new InputStreamReader(is, StandardCharsets.*UTF\_8*));  
 String inputLine;  
 while ((inputLine = br.readLine()) != null) {  
 sb.append(inputLine);  
 }  
 return sb.toString();  
 }  
  
 } catch (IOException e) {  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Реализует POST-обращение к серверу  
 \** ***@param*** *link - адрес  
 \** ***@param*** *jsonObject - объект  
 \** ***@return*** *String - полученные данные  
 \** ***@throws*** *IOException - может возникнуть при отсутствии подключения  
 \*/* public static String Post(String link, JSONObject jsonObject) throws IOException {  
 URL url = new URL(link);  
 HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();  
 httpURLConnection.setRequestMethod("POST");  
 httpURLConnection.setDoOutput(true);  
 httpURLConnection.setRequestProperty("Accept", "application/json");  
 httpURLConnection.setRequestProperty("Content-Type", "application/json; utf-8");  
 try(OutputStream os = httpURLConnection.getOutputStream()) {  
 byte[] input = jsonObject.toString().getBytes(StandardCharsets.*UTF\_8*);  
 os.write(input, 0, input.length);  
 }  
 return *execResponse*(httpURLConnection);  
 }  
 */\*\*  
 \* Реализует PUT-обращение к серверу  
 \** ***@param*** *link - адрес  
 \** ***@param*** *jsonObject - объект  
 \** ***@return*** *String - полученные данные  
 \** ***@throws*** *IOException - может возникнуть при отсутствии подключения/успешного завершения  
 \*/* public static String Put(String link, JSONObject jsonObject) throws IOException {  
 URL url = new URL(link);  
 HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();  
 httpURLConnection.setRequestMethod("PUT");  
 httpURLConnection.setRequestProperty("Content-Type", "application/json; utf-8");  
 httpURLConnection.setRequestProperty("Accept", "application/json");  
 httpURLConnection.setDoOutput(true);  
 try(OutputStream os = httpURLConnection.getOutputStream()) {  
 byte[] input = jsonObject.toString().getBytes(StandardCharsets.*UTF\_8*);  
 os.write(input, 0, input.length);  
 }  
 return *execResponse*(httpURLConnection);  
 }  
  
 */\*\*  
 \* Реализует DELETE-обращение к серверу  
 \** ***@param*** *link - адрес  
 \** ***@throws*** *IOException - может возникнуть при отсутствии подключения/успешного завершения  
 \** ***@return*** *String - полученные данные  
 \*/* public static String Delete(String link) throws IOException {  
 URL url = new URL(link);  
 HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();  
 httpURLConnection.setRequestMethod("DELETE");  
 httpURLConnection.setRequestProperty("Content-Type", "application/json; utf-8");  
 httpURLConnection.setRequestProperty("Accept", "application/json");  
 httpURLConnection.setDoOutput(true);  
 return *execResponse*(httpURLConnection);  
 }  
  
 */\*\*  
 \* Считывает полученные данные  
 \** ***@param*** *httpURLConnection - http-соединение  
 \** ***@return*** *String - полученные данные  
 \** ***@throws*** *IOException - может возникнуть при отсутствии подключения/успешного завершения  
 \*/* private static String execResponse(HttpURLConnection httpURLConnection) throws IOException {  
 try(BufferedReader br = new BufferedReader(  
 new InputStreamReader(httpURLConnection.getInputStream(), StandardCharsets.*UTF\_8*))) {  
 StringBuilder response = new StringBuilder();  
 String responseLine;  
 while ((responseLine = br.readLine()) != null) {  
 response.append(responseLine.trim());  
 }  
 return response.toString();  
 }  
 }  
  
}

**HashUtil:**

package client.utils;  
  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
  
*/\*\*  
 \* Класс для работы с хешированием  
 \*/*public class HashUtil {  
 */\*\*  
 \* Преобразование байтов в HEX  
 \*  
 \** ***@param*** *hash - байтовые данные  
 \** ***@return*** *string - преобразованные байты  
 \*/* private static String bytesToHex(byte[] hash) {  
 StringBuilder hexString = new StringBuilder(2 \* hash.length);  
 for (byte b : hash) {  
 String hex = Integer.*toHexString*(0xff & b);  
 if (hex.length() == 1) {  
 hexString.append('0');  
 }  
 hexString.append(hex);  
 }  
 return hexString.toString();  
 }  
  
 */\*\*  
 \* Конвертация хеширование данных в SHA-256  
 \*  
 \** ***@param*** *input - строка для хеширования  
 \** ***@return*** *String - преобразованная строка  
 \*/* public static String convert(String input) {  
  
 try {  
 MessageDigest digest = MessageDigest.*getInstance*("SHA-256");  
 return *bytesToHex*(digest.digest(input.getBytes(StandardCharsets.*UTF\_8*)));  
 } catch (NoSuchAlgorithmException e) {  
 return null;  
 }  
 }  
}

**AlertInfo:**

package client.utils;  
  
import client.JavaFXApplication;  
import javafx.scene.control.Alert;  
import javafx.scene.control.DialogPane;  
import javafx.stage.Modality;  
  
*/\*\*  
 \* Вспомогательный класс для отображения алертов  
 \*/*public class AlertInfo {  
 */\*\*  
 \* Создает алерт-предупреждение  
 \** ***@param*** *mainApp - экземпляр главного приложения  
 \** ***@return*** *alert  
 \*/* public static Alert getWarningAlert(JavaFXApplication mainApp){  
 Alert alert = new Alert(Alert.AlertType.*WARNING*);  
 alert.initModality(Modality.*APPLICATION\_MODAL*);  
 alert.initOwner(mainApp.getPrimaryStage());  
 alert.setTitle("wear me");  
 DialogPane dialogPane = alert.getDialogPane();  
 dialogPane.getStylesheets().add(  
 String.*valueOf*(JavaFXApplication.class.getResource("styles/ConnectionError.css").toExternalForm()));  
 dialogPane.getStyleClass().add("myDialog");  
 return alert;  
 }  
 */\*\*  
 \* Создает успешный алерт  
 \** ***@param*** *mainApp - экземпляр главного приложения  
 \** ***@return*** *alert  
 \*/* public static Alert getOkAlert(JavaFXApplication mainApp) {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.initModality(Modality.*APPLICATION\_MODAL*);  
 alert.setTitle("wear me");  
 alert.initOwner(mainApp.getPrimaryStage());  
 DialogPane dialogPane = alert.getDialogPane();  
 dialogPane.getStylesheets().add(  
 String.*valueOf*(JavaFXApplication.class.getResource("styles/ConnectionError.css").toExternalForm()));  
 dialogPane.getStyleClass().add("myDialog");  
 return alert;  
 }  
 */\*\*  
 \* Создает алерт-подтверждение  
 \** ***@param*** *mainApp - экземпляр главного приложения  
 \** ***@return*** *alert  
 \*/* public static Alert getConfirmationAlert(JavaFXApplication mainApp) {  
 Alert alert = new Alert(Alert.AlertType.*CONFIRMATION*);  
 alert.initOwner(mainApp.getPrimaryStage());  
 alert.initModality(Modality.*APPLICATION\_MODAL*);  
 alert.setTitle("wear me");  
 DialogPane dialogPane = alert.getDialogPane();  
 dialogPane.getStylesheets().add(  
 String.*valueOf*(JavaFXApplication.class.getResource("styles/ConnectionError.css").toExternalForm()));  
 dialogPane.getStyleClass().add("myDialog");  
 return alert;  
 }  
  
}

**JavaFXApplication:**

package client;  
  
import client.api.MyAPI;  
import client.controller.\*;  
import client.exception.NoStoreProductException;  
import client.utils.AlertInfo;  
import client.utils.MyLogger;  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.Alert;  
import javafx.scene.control.DialogPane;  
import javafx.scene.image.Image;  
import javafx.scene.layout.\*;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import org.json.JSONException;  
  
import java.io.IOException;  
  
*/\*\*  
 \* Java FX Application  
 \*/*public class JavaFXApplication extends Application {  
 private Stage primaryStage;  
 private RootController rootController;  
 private MyAPI API;  
  
 */\*\*  
 \* Устанавливает значение API  
 \** ***@param*** *API - экземпляр класса MyAPI  
 \*/* public void setAPI(MyAPI API) {  
 this.API = API;  
 }  
  
 */\*\*  
 \* Получает значение API  
 \** ***@return*** *MyAPI - значение API  
 \*/* public MyAPI getAPI() {  
 return API;  
 }  
  
 */\*\*  
 \* Получает главную сцену  
 \** ***@return*** *Stage primaryStage  
 \*/* public Stage getPrimaryStage() {  
 return primaryStage;  
 }  
  
 */\*\*  
 \* Запускает приложение и основную сцену,запускает окно авторизации  
 \** ***@param*** *primaryStage - основная сцена  
 \*/* @Override  
 public void start(Stage primaryStage) {  
 this.primaryStage = primaryStage;  
 this.primaryStage.setTitle("wear me");  
 this.primaryStage.getIcons().add(new Image("client/images/icon.png"));  
 initSignIn();  
 }  
  
 */\*\*  
 \* Инициализирует сцену авторизации сотрудников  
 \*/* public void initSignIn() {  
 try {  
 this.primaryStage.setMaximized(false);  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/signIn.fxml"));  
 VBox rootLayout = loader.load();  
 Scene scene = new Scene(rootLayout);  
 this.primaryStage.setScene(scene);  
 this.primaryStage.show();  
 SignInController controller = loader.getController();  
 controller.setMainApp(this);  
  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены авторизации");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Возвращает значение rootController  
 \** ***@return*** *RootController rootController  
 \*/* public RootController getRootController() {  
 return rootController;  
 }  
  
 */\*\*  
 \* Инициализирует корневую часть приложения  
 \*/* public void initRoot() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/root.fxml"));  
 AnchorPane rootLayout = loader.load();  
 Scene scene = new Scene(rootLayout);  
 this.primaryStage.close();  
 this.primaryStage.setScene(scene);  
 this.primaryStage.setMaximized(true);  
 this.primaryStage.setResizable(false);  
 MyLogger.*logger*.info("Запущено главное окно");  
 this.primaryStage.show();  
 RootController controller = loader.getController();  
 controller.setMainApp(this);  
 controller.setAPI(this.API);  
 controller.setView();  
 this.rootController = controller;  
  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке основной сцены");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену при отсутствии соединения с сервером  
 \*/* public void connectionError() {  
 try {  
 Alert a = new Alert(Alert.AlertType.*WARNING*);  
 a.initOwner(this.getPrimaryStage());  
 a.setTitle("wear me");  
 DialogPane dialogPane = a.getDialogPane();  
 dialogPane.getStylesheets().add(  
 getClass().getResource("styles/ConnectionError.css").toExternalForm());  
 dialogPane.getStyleClass().add("myDialog");  
 a.setHeaderText("Server not responding");  
 a.setContentText("Login failed");  
 a.initModality(Modality.*APPLICATION\_MODAL*);  
 a.showAndWait();  
 } catch (NullPointerException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены ошибки подключения");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену при отсутствии данных при подключении к серверу  
 \*/* public void noDataError() {  
 try {  
 Alert a = new Alert(Alert.AlertType.*WARNING*);  
 a.initOwner(this.getPrimaryStage());  
 a.setTitle("store app");  
 DialogPane dialogPane = a.getDialogPane();  
 dialogPane.getStylesheets().add(  
 getClass().getResource("styles/ConnectionError.css").toExternalForm());  
 dialogPane.getStyleClass().add("myDialog");  
 a.setHeaderText("Server not responding");  
 a.setContentText("No data for app");  
 a.initModality(Modality.*APPLICATION\_MODAL*);  
 a.showAndWait();  
 this.primaryStage.close();  
 } catch (NullPointerException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены ошибки подключения");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену добавления товаров в чек  
 \*/* public void initAddingToCheck() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingToCheck.fxml"));  
 Stage dialogStage = getLoader(loader);  
 AddingToCheckController controller = loader.getController();  
 controller.setAPI(this.API);  
 controller.setMainApp(this);  
 controller.setCombobox();  
 controller.setStage(dialogStage);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 MyLogger.*logger*.info("Запущено окно добавления товара в чек");  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены добавления товара в чек");  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Точка входа в программу  
 \** ***@param*** *args - аргументы  
 \*/* public static void main(String[] args) {  
 MyLogger log = new MyLogger();  
 if (log.deploy())  
 *launch*(args);  
 }  
  
 */\*\*  
 \* Инициализирует сцену добавления промокода  
 \*/* public void initAddingPromocode() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingPromocode.fxml"));  
 Stage dialogStage = getLoader(loader);  
 AddingPromocodeController controller = loader.getController();  
 controller.setAPI(this.API);  
 controller.setMainApp(this);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setStage(dialogStage);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно промокода");  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены добавления промокода");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену добавления карты лояльности  
 \*/* public void initAddingLoyaltyCard() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingLoyaltyCard.fxml"));  
 Stage dialogStage = this.getLoader(loader);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 AddingLoyaltyCardController controller = loader.getController();  
 controller.setAPI(this.API);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно добавления карты клиента");  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены добавления карты клиента");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену чека  
 \*/* public void initCheck() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/check.fxml"));  
 Stage dialogStage = getLoader(loader);  
 CheckController controller = loader.getController();  
 controller.setAPI(this.API);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 this.getRootController().createCheck();  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно запуска чека");  
 controller.setItems();  
 dialogStage.showAndWait();  
 } catch (IOException | JSONException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены чека");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену возврата  
 \*/* public void initReturn() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/returnProduct.fxml"));  
 Stage dialogStage = getLoader(loader);  
 ReturnController controller = loader.getController();  
 controller.setAPI(this.API);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно запуска возврата");  
 dialogStage.show();  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены возврата");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену информации о возврате  
 \*/* public void initReturnInfo() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/returnCheckInfo.fxml"));  
 Stage dialogStage = getLoader(loader);  
 ReturnInfoController controller = loader.getController();  
 controller.setAPI(this.API);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 this.API.setReturnResult("");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно запуска возврата таблицы");  
 controller.setTable(this.API.getProductData());  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены возврата таблицы");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену успешного возврата  
 \*/* public void initSucReturn() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/successReturn.fxml"));  
 Stage dialogStage = getLoader(loader);  
 ReturnInfoController controller = loader.getController();  
 controller.setAPI(this.API);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 controller.getSuccessfullReturn();  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 this.API.setReturnResult("");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно успешного возврата");  
 dialogStage.showAndWait();  
 } catch (IOException e) {  
 MyLogger.*logger*.error("Ошибка при загрузке сцены успешного возврата");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Создает новую диалоговую сцену с предзагрузками  
 \** ***@param*** *loader - лоадер  
 \** ***@return*** *dialogStage - диалоговая сцена с предзагрузками  
 \** ***@throws*** *IOException - может возникнуть исключение  
 \*/* public Stage getLoader(FXMLLoader loader) throws IOException {  
 AnchorPane rootLayout = loader.load();  
 Stage dialogStage = new Stage();  
 dialogStage.getIcons().add(new Image("client/images/icon.png"));  
 dialogStage.setTitle("wear me");  
 dialogStage.initOwner(this.primaryStage);  
 dialogStage.setResizable(false);  
 Scene scene = new Scene(rootLayout);  
 dialogStage.setScene(scene);  
 return dialogStage;  
 }  
  
 */\*\*  
 \* Инициализирует сцену клиентов  
 \*/* public void initClients() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/client.fxml"));  
 Stage dialogStage = getLoader(loader);  
 ClientController controller = loader.getController();  
 controller.setAPI(this.API);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 controller.loadData();  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно клиентов");  
 dialogStage.show();  
 }  
 catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No client info");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену добавления клиента  
 \** ***@param*** *clientController - контроллер клиента  
 \*/* public void initAddingClient(ClientController clientController) {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingClient.fxml"));  
 Stage dialogStage = getLoader(loader);  
 AddingClientController controller = loader.getController();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setClientController(clientController);  
 controller.setStage(dialogStage);  
 controller.setAPI(this.API);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно клиентов");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("Can't load adding client");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену промокодов  
 \*/* public void initPromocodes() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
  
 loader.setLocation(JavaFXApplication.class.getResource("views/promocode.fxml"));  
 Stage dialogStage = getLoader(loader);  
 PromocodeController controller = loader.getController();  
 controller.setAPI(this.API);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 controller.loadData();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно промокодов");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No promocodes info");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену добавления промокода  
 \** ***@param*** *promocodeController - контроллер промокодов  
 \*/* public void initAddingPromocodeNew(PromocodeController promocodeController) {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingPromocodeNew.fxml"));  
 Stage dialogStage = getLoader(loader);  
 AddingPromocodeNewController controller = loader.getController();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setPromocodeController(promocodeController);  
 controller.setStage(dialogStage);  
 controller.setAPI(this.API);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно добавления промокодов");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("Can't load adding promocodes");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену офисов  
 \*/* public void initOffices() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/office.fxml"));  
 Stage dialogStage = getLoader(loader);  
 OfficeController controller = loader.getController();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setAPI(this.API);  
 controller.setMainApp(this);  
 controller.setStage(dialogStage);  
 controller.loadData();  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно офисов");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No offices info");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену складов  
 \*/* public void initStock() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/stock.fxml"));  
 Stage dialogStage = getLoader(loader);  
 StockController controller = loader.getController();  
 controller.setAPI(this.API);  
 controller.setMainApp(this);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setStage(dialogStage);  
 controller.loadData();  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно склада");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No stock info");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует сцену добавления поставщика  
 \** ***@param*** *stockController - контроллер склада  
 \*/* public void initAddingSupplier(StockController stockController) {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingSupplier.fxml"));  
 Stage dialogStage = getLoader(loader);  
 AddingSupplierController controller = loader.getController();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setStockController(stockController);  
 controller.setStage(dialogStage);  
 controller.setAPI(this.API);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно поставщиков");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("Can't load adding supplier");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует первую сцену поставки  
 \** ***@param*** *stockController - контроллер склада  
 \*/* public void initSupply1(StockController stockController){  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingSupply1.fxml"));  
 Stage dialogStage = this.getLoader(loader);  
 AddingSupply1Controller controller = loader.getController();  
 controller.setMainApp(this);  
 controller.setStockController(stockController);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setStage(dialogStage);  
 controller.setAPI(this.API);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно 1 стадии поставки");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No info about supply-1");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует вотрую сцену поставки  
 \** ***@param*** *addingSupplyController - контроллер первой стадии поставки  
 \*/* public void initSupply2(AddingSupply1Controller addingSupplyController) {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/addingSupply2.fxml"));  
 Stage dialogStage = this.getLoader(loader);  
 AddingSupply2Controller controller = loader.getController();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 controller.setMainApp(this);  
 controller.setSupply1Controller(addingSupplyController);  
 controller.setStage(dialogStage);  
 controller.setAPI(this.API);  
 controller.loadData();  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно 2 стадии поставки");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No info about supply-2");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует информацию для сотрудников  
 \*/* public void initInfo() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/info.fxml"));  
 Stage dialogStage = this.getLoader(loader);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно информации");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No info about information");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует окно статистики  
 \*/* public void initStat() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/stats.fxml"));  
 Stage dialogStage = this.getLoader(loader);  
 StatController controller = loader.getController();  
  
 controller.setAPI(this.API);  
 controller.setStage(dialogStage);  
 controller.setMainApp(this);  
 controller.loadData();  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно статистики");  
 dialogStage.show();  
 }  
 catch(NoStoreProductException e){  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No info about stats");  
 a.setContentText("Server not responding");  
 e.printStackTrace();  
 a.show();  
 }  
 catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("Client Error");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
  
 */\*\*  
 \* Инициализирует окно информации об авторе  
 \*/* public void initAboutMe() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(JavaFXApplication.class.getResource("views/aboutMe.fxml"));  
 Stage dialogStage = this.getLoader(loader);  
 dialogStage.initModality(Modality.*APPLICATION\_MODAL*);  
 dialogStage.setOnCloseRequest(we -> {  
 MyLogger.*logger*.info("Окно закрыто");  
 dialogStage.close();  
 });  
 MyLogger.*logger*.info("Открыто окно об авторе");  
 dialogStage.show();  
 } catch (Exception e) {  
 MyLogger.*logger*.error(e.getMessage());  
 Alert a = AlertInfo.*getWarningAlert*(this);  
 a.setHeaderText("No info about author");  
 a.setContentText(e.getMessage());  
 e.printStackTrace();  
 a.show();  
 }  
 }  
}