**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Разработка ПК для диспетчера склада»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1307 |  | Угрюмов М.М. |
| Преподаватель |  | Гречухин М.Н. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)**

Разработать ПК для диспетчера склада, используя знания и навыки, приобретенные в ходе курса по объектно-ориентированному программированию.

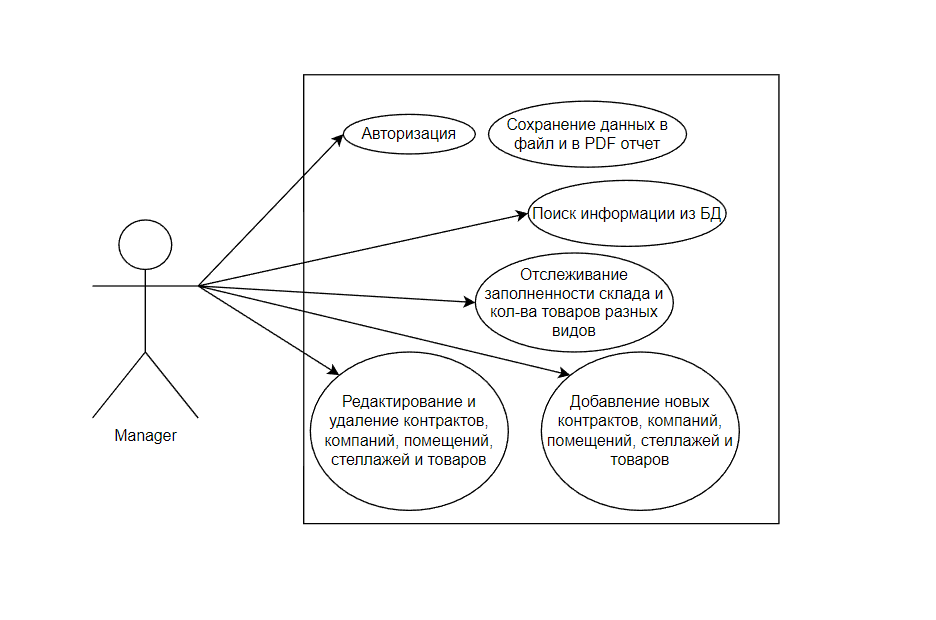
В ПК должны храниться сведения о компаниях, с которыми заключены договоры хранения, о помещениях и стеллажах для хранения груза, о размещенных грузах.

Диспетчер может добавлять, изменять и удалять эту информацию.

***Ему могут потребоваться следующие сведения: общий перечень договоров и дат их окончания общий процент заполненности склада товарами количество товаров определенного вида на складе.***

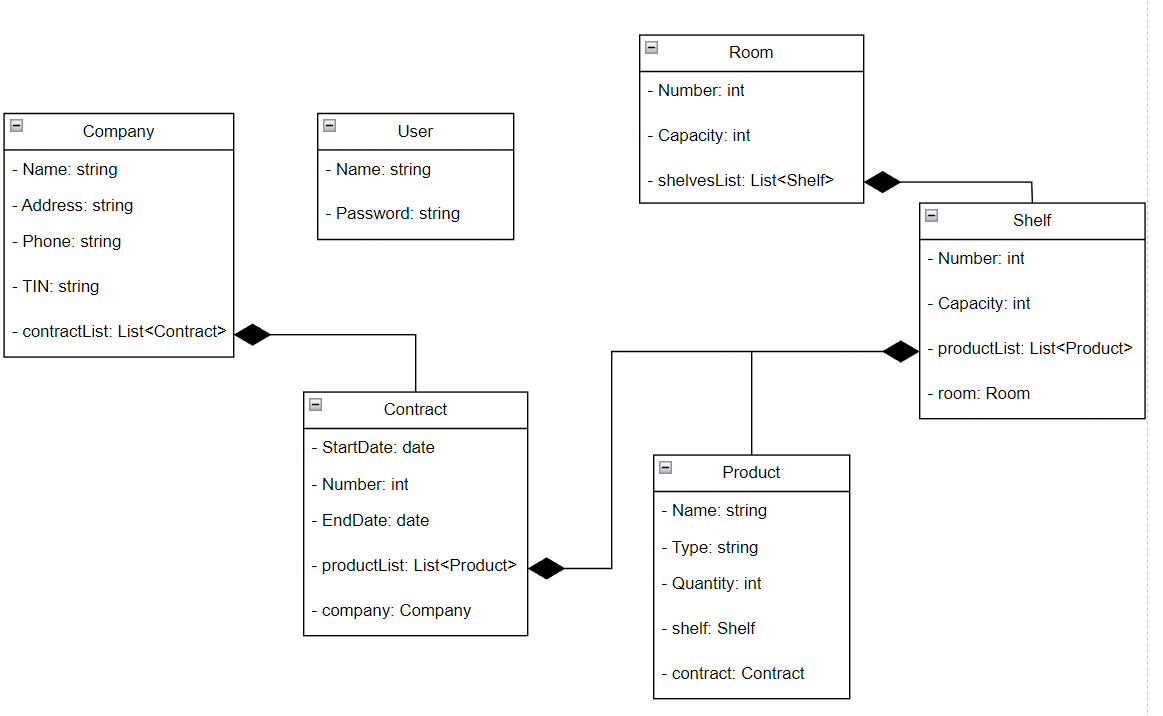
В конечном итоге был разработан ПК для удобной и корректной работы менеджера. ПК поддерживает авторизацию существующих пользователей, а также управление ключевыми сущностями — контрактами, компаниями, помещениями, стеллажами и товарами.

**ДИАГРАММА ПРЕЦЕДЕНТОВ**

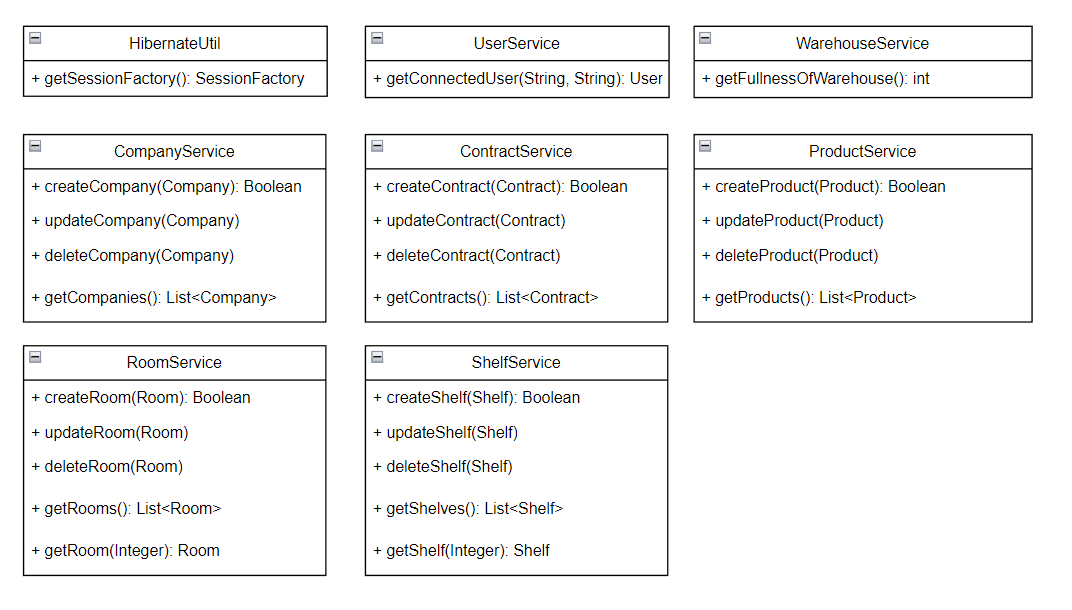
****

**UML ДИАГРАММА КЛАССОВ**

Классы данных



Классы для работы с базой данных

****

**ОПИСАНИЕ КЛАССОВ**

В работе используются 7 классов данных составляющих общую модель.

* Класс Contract представляет собой классификацию контракта. Имеет поля (свойства) id\_contract - уникальный идентификатор и startdate — дата подписания контракта, enddate — дата окончания контракта, company\_id — идентификатор компании с которой заключен контракт.
* Класс Company представляет собой классификацию компании. Имеет поля (свойства) id\_company - уникальный идентификатор и name - название компании, address – адрес компании, phone – телефонный номер компании, tin – ИНН компании.
* Класс Product представляет собой классификацию товара. Имеет поля (свойства) id\_product - уникальный идентификатор и name — название товара, type — тип товара, quantity — кол-во товара определенного вида, shelf\_id — идентификатор стеллажа на котором лежит товар, contract\_id — идентификатор контракта по которому товар имеет право находится на складе.
* Класс Room представляет собой классификацию помещения. Имеет поля (свойства) id\_room - уникальный идентификатор и number — номер помещения, capacity — вместимость помещения (кол-во стеллажей, которые можно поместить в данное помещение).
* Класс Shelf представляет собой классификацию стеллажа. Имеет поля (свойства) id\_shelf - уникальный идентификатор и number — номер стеллажа, capacity — вместимость полки (кол-во товаров, которые можно поместить на данный стеллаж, предполагается, что товары примерно одинакового размера и веса), room\_id - идентификатор помещения в котором находится стеллаж.
* Класс User представляет собой классификацию помещения. Имеет поля (свойства) id\_user - уникальный идентификатор и name — логин, password — пароль (предполагается что данные для входа в систему выдаются работнику отдельно без регистрации).

Также в работе использованы семь классов для работы с базой данных (далее - БД), один из которых HibernateUtil – используется во всех остальных для создания сессии подключения к БД, остальные же классы, функционирующие непосредственно с данными БД представлены ниже:

* Класс ContractService - класс сервиса для работы с таблицей контрактов.

Содержит методы для добавления, редактирования, удаления и получения данных из БД.

* Класс CompanyService - класс сервиса для работы с таблицей компаний.

Содержит методы для добавления, редактирования, удаления и получения данных из БД.

* Класс ProductService - класс сервиса для работы с таблицей товаров.

Содержит методы для добавления, редактирования, удаления и получения данных из БД.

* Класс RoomService - класс сервиса для работы с таблицей помещений.

Содержит методы для добавления, редактирования, удаления и получения данных из БД.

* Класс ShelfService - класс сервиса для работы с таблицей стеллажей.

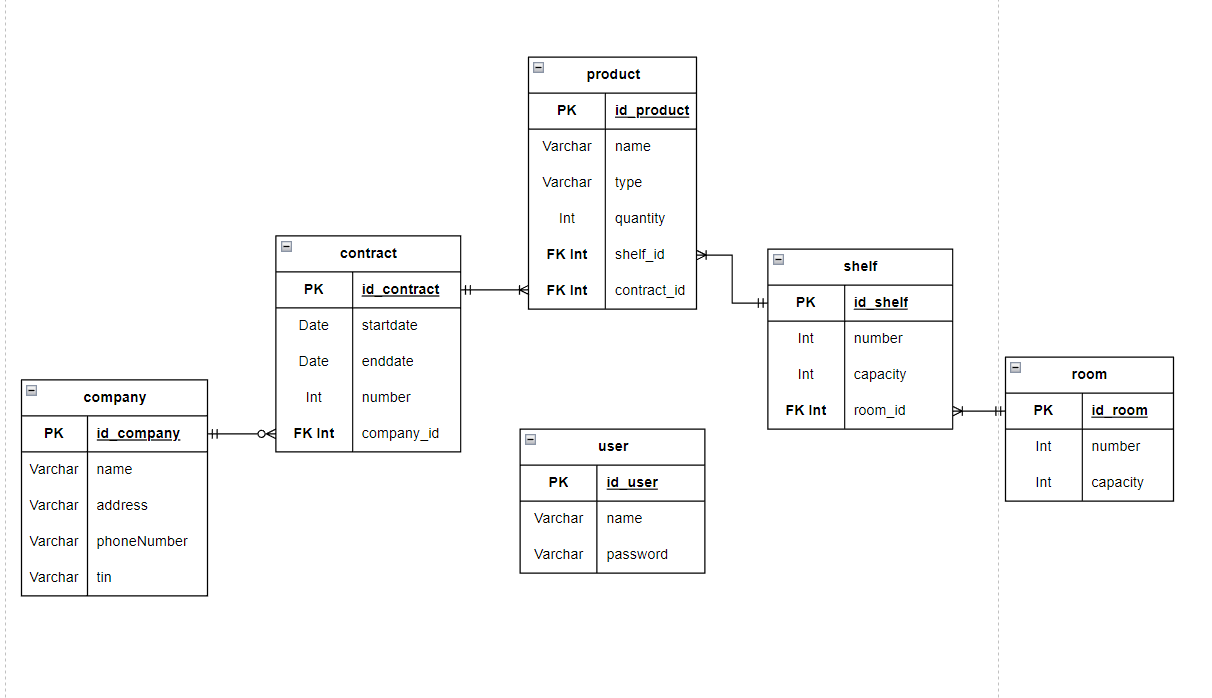
Содержит методы для добавления, редактирования, удаления и получения данных из БД.

* Класс UserService - класс сервиса для работы с таблицей пользователей.

Содержит метод получения данных из БД для входа в систему.

* Класс WarehouseService - класс сервиса для подсчета заполненности всего склада.

**СХЕМА ДАННЫХ**

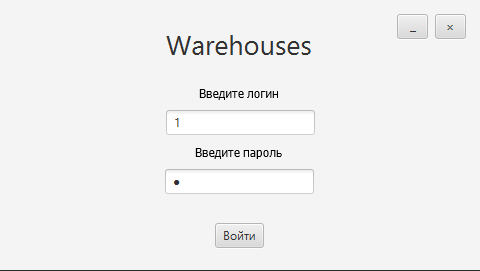


В работе представлены 6 таблиц БД:

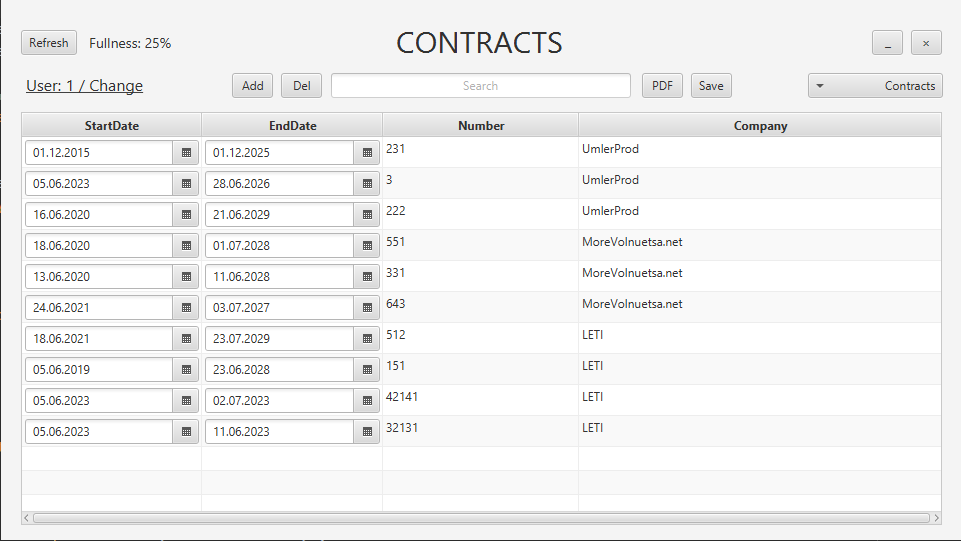
* Таблица Contract представляет собой классификацию контракта. Имеет поля (свойства) id\_contract - уникальный идентификатор и startdate — дата подписания контракта, enddate — дата окончания контракта, company\_id — идентификатор компании с которой заключен контракт.
* Таблица Company представляет собой классификацию компании. Имеет поля (свойства) id\_company - уникальный идентификатор и name - название компании, address – адрес компании, phone – телефонный номер компании, tin – ИНН компании.
* Таблица Product представляет собой классификацию товара. Имеет поля (свойства) id\_product - уникальный идентификатор и name — название товара, type — тип товара, quantity — кол-во товара определенного вида, shelf\_id — идентификатор стеллажа на котором лежит товар, contract\_id — идентификатор контракта по которому товар имеет право находится на складе.
* Таблица Room представляет собой классификацию помещения. Имеет поля (свойства) id\_room - уникальный идентификатор и number — номер помещения, capacity — вместимость помещения (кол-во стеллажей, которые можно поместить в данное помещение).
* Таблица Shelf представляет собой классификацию стеллажа. Имеет поля (свойства) id\_shelf - уникальный идентификатор и number — номер стеллажа, capacity — вместимость полки (кол-во товаров, которые можно поместить на данный стеллаж, предполагается, что товары примерно одинакового размера и веса), room\_id - идентификатор помещения в котором находится стеллаж.
* Таблица User представляет собой классификацию помещения. Имеет поля (свойства) id\_user - уникальный идентификатор и name — логин, password — пароль (предполагается что данные для входа в систему выдаются работнику отдельно без регистрации).

**ПРИМЕРЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА**

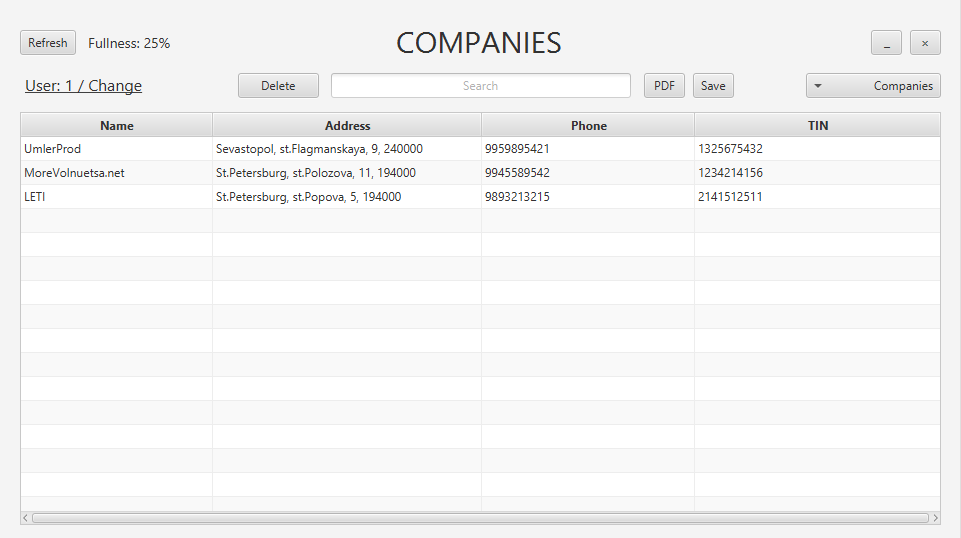
Вход

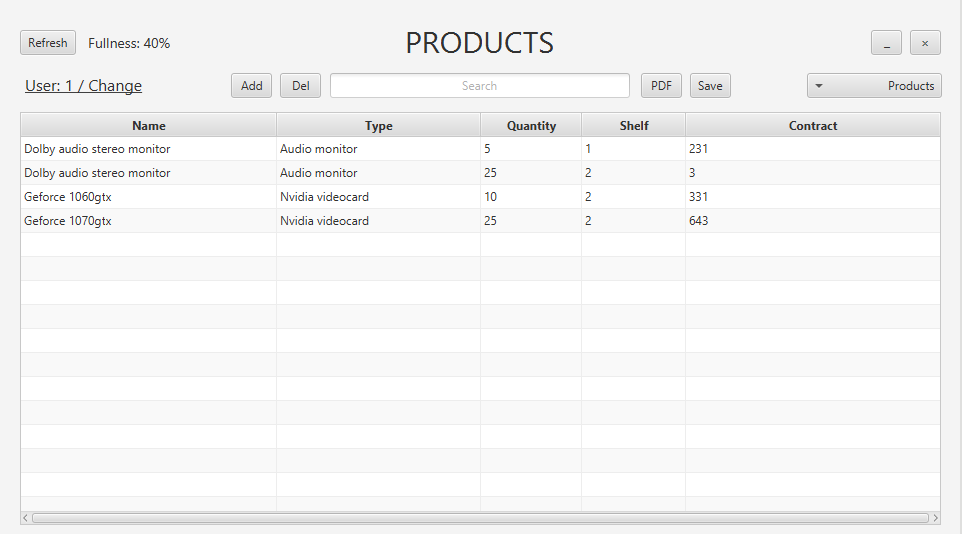
****

Список контрактов

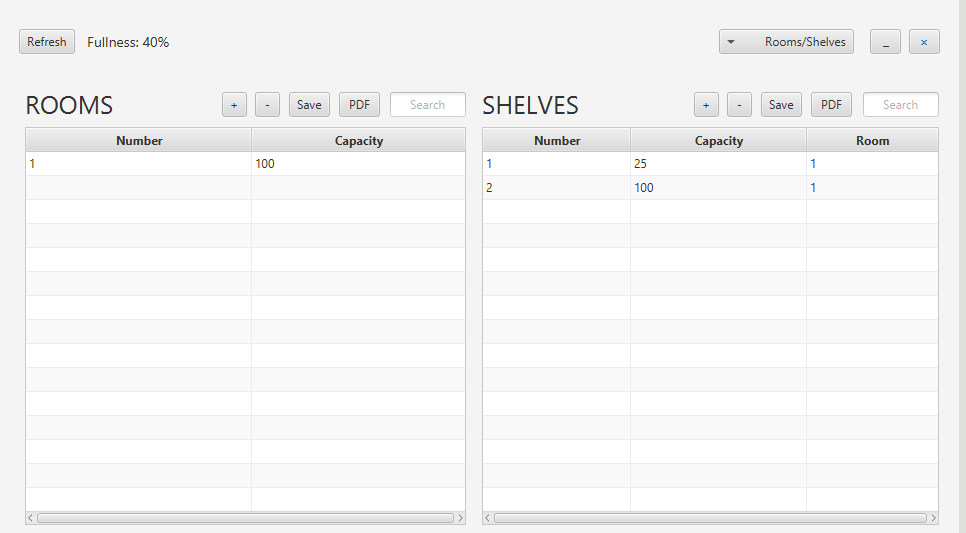


Список компаний

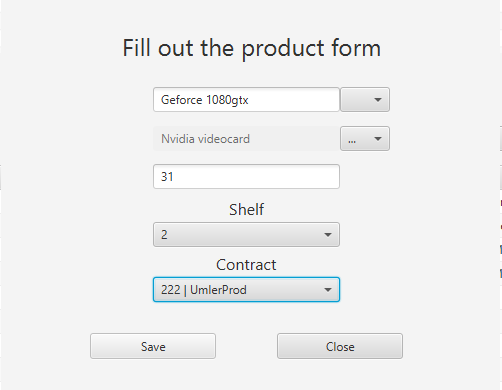
Список товаров



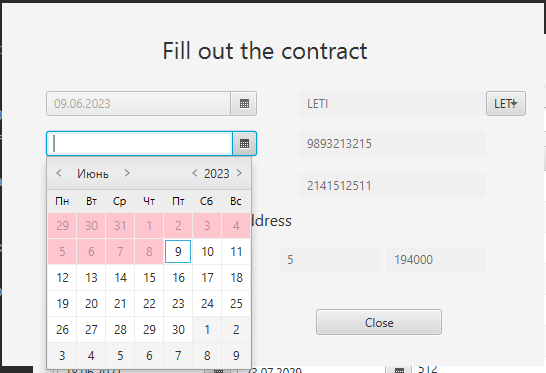
Список помещений и стеллажей

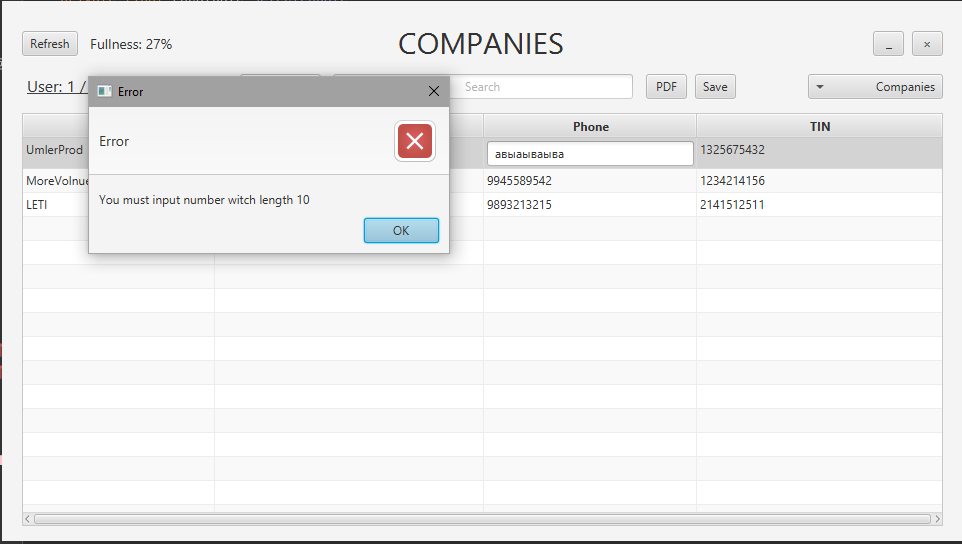


**Пример добавления записей**

****

**Пример добавления контракта** (дата окончания ограничена датой начала, выбрана уже существующая компания)

****

**Обработка неправильного ввода**

**КРАТКАЯ ИНСТРУКЦИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ**

После запуска программы откроется таблица Контрактов, чтобы переключатся между таблицами, нужно выбрать нужную Вам таблицу в choicebox находящийся в правой части экрана.

Доступные действия в таблице Контрактов:

• Поиск – в поисковую строку ввести запрос

• Добавить – нажать кнопку «+», откроется окно добавления – заполнить поля и сохранить.

• Удаление – выберите одну или несколько (зажать shift/ctrl) записей и нажмите кнопку «-»

• PDF отчет – нажмите кнопку «PDF»

• Сохранить в файл – нажмите кнопку «Save»

• Редактирование – два раза щелкните по нужной ячейке, поменяйте значение и нажмите Enter на клавиатуре

Доступные действия в таблице Company:

• Поиск – в поисковую строку ввести запрос

• Добавить – нажать кнопку «+», откроется окно добавления – заполнить поля и сохранить.

• Удаление – выберите одну или несколько (зажать shift/ctrl) записей и нажмите кнопку «-»

• Редактирование – два раза щелкните по нужной ячейке, поменяйте значение и нажмите Enter на клавиатуре

• PDF отчет – нажмите кнопку «PDF»

• Сохранить в файл – нажмите кнопку «Save»

Доступные действия в таблицах Помещения/Стеллажи:

• Поиск – в поисковую строку ввести запрос

• Добавить – нажать кнопку «+», откроется окно добавления – заполнить поля и сохранить.

• Удаление – выберите одну или несколько (зажать shift/ctrl) записей и нажмите кнопку «-»

• Редактирование – два раза щелкните по нужной ячейке, поменяйте значение и нажмите Enter на клавиатуре

• PDF отчет – нажмите кнопку «PDF»

• Сохранить в файл – нажмите кнопку «Save»

Доступные действия в таблице Товаров:

• Поиск – в поисковую строку ввести запрос

• Добавить – нажать кнопку «+», откроется окно добавления – заполнить поля и сохранить.

• Удаление – выберите одну или несколько (зажать shift/ctrl) записей и нажмите кнопку «-»

• Редактирование – два раза щелкните по нужной ячейке, поменяйте значение и нажмите Enter на клавиатуре

• PDF отчет – нажмите кнопку «PDF»

• Сохранить в файл – нажмите кнопку «Save»

**Необходимые пункты по ТЗ:**

Список компаний и информации о них - Выберите таблицу Companies

Список договоров и информации о них - Выберите таблицу Contracts

Список помещений и стеллажей и информация о них - Выберите таблицу Rooms/Shelves

Список товаров и информация о них - Выберите таблицу Products

Список свободных номеров с указанием его вместимости - Выберите таблицу

Общий процент заполненности склада показан в левом верхнем углу надписью:

Fullness: %

Кол-во товаров определенного вида — столбец Quantity в таблице Products.

**ЗАКЛЮЧЕНИЕ**

Было разработано приложение с использованием современных технологий и подходов к разработке: в качестве языка программирования был выбран Java. В ходе работы были применены знания и навыки, полученные в ходе курса объектно-ориентированного программирования, в частности, были применены основные концепции ООП. Также была проделана работа с освоением языка SQL и библиотеки JavaFX для написания фронтенда. В конечном итоге был разработан ПК, полностью соответствующий техническому заданию.

**ПРИЛОЖЕНИЕ 1**

**ИСХОДНЫЙ КОД**

**https://github.com/umlerr/Warehouses.git**

AddContractCompanyController

package com.umler.warehouses.AddControllers;

import com.umler.warehouses.Helpers.DatePickerCellFactory;

import com.umler.warehouses.Model.Company;

import com.umler.warehouses.Model.Contract;

import com.umler.warehouses.Services.ContractService;

import javafx.animation.PauseTransition;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.util.Duration;

import com.umler.warehouses.Controllers.SceneController;

import com.umler.warehouses.Helpers.UpdateStatus;

import com.umler.warehouses.Services.CompanyService;

import java.net.URL;

import java.time.LocalDate;

import java.util.List;

import java.util.Objects;

import java.util.ResourceBundle;

public class AddContractCompanyController implements Initializable {

@FXML

public TextField name\_field;

@FXML

public TextField phone\_field;

@FXML

public TextField tin\_field;

@FXML

public DatePicker enddatepicker;

@FXML

public DatePicker startdate\_field;

@FXML

public TextField number\_field;

@FXML

public TextField city\_field;

@FXML

public TextField street\_field;

@FXML

public TextField building\_field;

@FXML

public TextField index\_field;

@FXML

public ChoiceBox<Company> company\_choicebox;

CompanyService companyService = new CompanyService();

ContractService contractService = new ContractService();

/\*\*

\* Сохраняет изменения в базе данных после добавления компании и контракта.

\* Если данные введены корректно, то создается новый объект компании и контракта и происходит обновление базы данных.

\* Если компания уже была создана, и ее выбрали, новый контракт присоеденяется к этой компании.

\* Если данные введены некорректно, то выводится сообщение об ошибке.

\*/

@FXML

private void saveNewContractCompanyToDb(ActionEvent event){

if (validateInputs()) {

int flag = 0;

Contract contract = createContractFromInput();

Company company = createCompanyFromInput();

List<Company> companies = companyService.getCompanies();

for (Company company1 : companies)

{

if (company1.getTin().equals(company.getTin()) &&

company1.getName().equals(company.getName())

&& company1.getPhoneNumber().equals(company.getPhoneNumber())

&& company1.getAddress().equals(company.getAddress()))

{

flag = 1;

company1.addContract(contract);

boolean isSaved = new ContractService().createContract(contract);

new CompanyService().updateCompany(company1);

if (isSaved) {

UpdateStatus.setIsContractCompanyAdded(true);

delayWindowClose(event);

}

}

}

if (flag == 0)

{

company.addContract(contract);

boolean isSaved = new CompanyService().createCompany(company);

if (isSaved) {

UpdateStatus.setIsContractCompanyAdded(true);

delayWindowClose(event);

}

}

}

}

/\*\*

\* Проверяет корректность введенных данных.

\* Если все поля заполнены корректно, возвращает true.

\* Если есть незаполненные поля или данные введены некорректно, выводит сообщение об ошибке и возвращает false.

\*/

private boolean validateInputs() {

DatePickerCellFactory enddate\_field = new DatePickerCellFactory(enddatepicker);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Input Error", ButtonType.OK);

if (name\_field.getText().equals("")

|| phone\_field.getText().equals("")

|| tin\_field.getText().equals("")

|| enddate\_field.getValue() == null

|| startdate\_field.getValue() == null

|| number\_field.getText().equals("")

|| city\_field.getText().equals("")

|| street\_field.getText().equals("")

|| building\_field.getText().equals("")

|| index\_field.getText().equals("")) {

IOAlert.setContentText("You must fill out the contract to continue");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(number\_field.getText())){

IOAlert.setContentText("Incorrect input for contract number - you must put a positive number");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(index\_field.getText()) || isCorrectIndex(index\_field.getText())){

IOAlert.setContentText("Incorrect input for index number - you must put a number in format XXXYYY");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(phone\_field.getText()) || isCorrectTINorPhone(phone\_field.getText())){

IOAlert.setContentText("Incorrect input for phone number - you must put a number with 10 symbols");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (!isNumberExist(Integer.valueOf(number\_field.getText()))){

IOAlert.setContentText("Incorrect input for CONTRACT NUMBER - CONTRACT with this number already exists");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(tin\_field.getText()) || isCorrectTINorPhone(tin\_field.getText())){

IOAlert.setContentText("Incorrect input for TIN - you must put a number with 10 symbols");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

return true;

}

/\*\*

\* Создает новый объект контракта на основе введенных данных.

\* @return новый объект контракта

\*/

private Contract createContractFromInput() {

DatePickerCellFactory enddate\_field = new DatePickerCellFactory(enddatepicker);

Contract contract = new Contract();

contract.setStartdate(startdate\_field.getValue());

contract.setEnddate(enddate\_field.getValue());

contract.setNumber(Integer.valueOf(number\_field.getText()));

return contract;

}

/\*\*

\* Создает новый объект компании на основе введенных данных.

\* @return новый объект компании

\*/

private Company createCompanyFromInput() {

Company company = new Company();

company.setName(name\_field.getText());

company.setAddress(capitalizeAfterDot(capitalize(city\_field.getText().toLowerCase())) + ", st." +

capitalize(street\_field.getText().toLowerCase()) + ", " +

building\_field.getText() +

", " + index\_field.getText());

company.setPhoneNumber(phone\_field.getText());

company.setTin(tin\_field.getText());

return company;

}

/\*\*

\* Делает первую букву строки прописной.

\* @return строку с большой буквы

\*/

private static String capitalize(String str)

{

if (str == null || str.length() == 0) {

return str;

}

return str.substring(0, 1).toUpperCase() + str.substring(1);

}

/\*\*

\* Делает букву после точки прописной.

\* @return строку с большой буквы после точки для проверки городов по типу St.Petersburg

\*/

private static String capitalizeAfterDot(String str) {

StringBuilder result = new StringBuilder();

boolean capitalizeNext = false;

for (int i = 0; i < str.length(); i++) {

char c = str.charAt(i);

if (capitalizeNext && Character.isLetter(c)) {

result.append(Character.toUpperCase(c));

capitalizeNext = false;

} else {

result.append(c);

}

if (c == '.') {

capitalizeNext = true;

}

}

return result.toString();

}

/\*\*

\* Выбор уже существующей компании.

\* Блокировка полей если компания выбрана

\*/

@FXML

private void getChoices() {

Company company = company\_choicebox.getValue();

if (!Objects.equals(company, null))

{

String[] parts = company.getAddress().replaceAll("\\s+","").split(",");

name\_field.setEditable(false);

name\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

name\_field.setText(company.getName());

city\_field.setEditable(false);

city\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

city\_field.setText(parts[0]);

street\_field.setEditable(false);

street\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

street\_field.setText(parts[1].substring(3));

building\_field.setEditable(false);

building\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

building\_field.setText(parts[2]);

index\_field.setEditable(false);

index\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

index\_field.setText(parts[3]);

phone\_field.setEditable(false);

phone\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

phone\_field.setText(company.getPhoneNumber());

tin\_field.setEditable(false);

tin\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

tin\_field.setText(company.getTin());

}

else

{

name\_field.setEditable(true);

name\_field.setStyle(null);

city\_field.setEditable(true);

city\_field.setStyle(null);

street\_field.setEditable(true);

street\_field.setStyle(null);

building\_field.setEditable(true);

building\_field.setStyle(null);

index\_field.setEditable(true);

index\_field.setStyle(null);

phone\_field.setEditable(true);

phone\_field.setStyle(null);

tin\_field.setEditable(true);

tin\_field.setStyle(null);

}

}

/\*\*

\* Проверка введенной строки на число.

\*/

private static boolean isNumeric(String str) {

try {

return Double.parseDouble(str) <= 0;

} catch(NumberFormatException e){

return true;

}

}

/\*\*

\* Проверка введеного почтового индекса на кол-во символов.

\*/

private static boolean isCorrectIndex(String str) {

try {

return str.length()!=6;

} catch(NumberFormatException e){

return true;

}

}

/\*\*

\* Проверка введеного ИНН на кол-во символов.

\*/

private static boolean isCorrectTINorPhone(String str) {

try {

return str.length()!=10;

} catch(NumberFormatException e){

return true;

}

}

/\*\*

\* Проверка что введеный номер контракта уже существует.

\*/

private boolean isNumberExist(Integer number){

for (Contract contracts : contractService.getContracts()){

if (Objects.equals(contracts.getNumber(), number))

return false;

}

return true;

}

/\*\*

\* Задержка закрытия окна добавления до нажатия клавишы сохранения.

\*/

private void delayWindowClose(ActionEvent event) {

PauseTransition delay = new PauseTransition(Duration.seconds(1));

delay.setOnFinished(event2 -> closeWindow(event));

delay.play();

}

/\*\*

\* Закрытия окна по кнопке.

\*/

@FXML

private void closeWindow(ActionEvent event) {

SceneController.close(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

startdate\_field.setEditable(false);

startdate\_field.setShowWeekNumbers(false);

startdate\_field.setStyle("-fx-opacity: 0.9;");

startdate\_field.setValue(LocalDate.now());

ObservableList<Company> roomObservableList = FXCollections.observableArrayList(companyService.getCompanies());

roomObservableList.add(null);

company\_choicebox.getItems().addAll(roomObservableList);

}

}

AddProductController

package com.umler.warehouses.AddControllers;

import com.umler.warehouses.Controllers.SceneController;

import com.umler.warehouses.Helpers.DistinctObservableList;

import com.umler.warehouses.Helpers.UpdateStatus;

import com.umler.warehouses.Helpers.ComboBoxUtil;

import com.umler.warehouses.Model.Contract;

import com.umler.warehouses.Model.Product;

import com.umler.warehouses.Model.Shelf;

import com.umler.warehouses.Services.ContractService;

import com.umler.warehouses.Services.ProductService;

import com.umler.warehouses.Services.ShelfService;

import javafx.animation.PauseTransition;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.util.Duration;

import java.net.URL;

import java.util.List;

import java.util.Objects;

import java.util.ResourceBundle;

public class AddProductController implements Initializable {

@FXML

public ChoiceBox<Contract> contract\_choicebox;

@FXML

public ChoiceBox<Shelf> shelf\_choicebox;

@FXML

public TextField quantity\_field;

@FXML

public TextField type\_field;

@FXML

public TextField name\_field;

@FXML

public ComboBox<Product> type\_comboBox;

@FXML

public ComboBox<Product> name\_comboBox;

ContractService contractService = new ContractService();

ProductService productService = new ProductService();

ShelfService shelfService = new ShelfService();

/\*\*

\* Сохраняет изменения в базе данных после добавления продукта.

\* Если данные введены корректно, то создается новый объект компании и контракта и происходит обновление базы данных.

\* Если данные введены некорректно, то выводится сообщение об ошибке.

\*/

@FXML

private void saveNewProductToDb(ActionEvent event){

if (validateInputs()) {

Product product = createProductFromInput();

boolean isSaved = new ProductService().createProduct(product);

if (isSaved) {

UpdateStatus.setIsProductAdded(true);

delayWindowClose(event);

}

}

}

/\*\*

\* Выбор названия продукта и блокировка изменения поля если выбран уже существующий тип.

\*/

@FXML

private void getChoicesName() {

Product product = name\_comboBox.getValue();

if (!Objects.equals(product, null))

{

name\_field.setEditable(false);

name\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

name\_field.setText(product.getName());

}

else

{

name\_field.setEditable(true);

name\_field.setStyle(null);

}

}

/\*\*

\* Выбор типа продукта и блокировка изменения поля если выбран уже существующий тип.

\*/

@FXML

private void getChoicesType() {

Product product = type\_comboBox.getValue();

if (!Objects.equals(product, null))

{

type\_field.setEditable(false);

type\_field.setStyle("-fx-opacity: 0.7; -fx-background-color: #eee");

type\_field.setText(product.getType());

}

else

{

type\_field.setEditable(true);

type\_field.setStyle(null);

}

}

/\*\*

\* Проверяет корректность введенных данных.

\* Если все поля заполнены корректно, возвращает true.

\* Если есть незаполненные поля или данные введены некорректно, выводит сообщение об ошибке и возвращает false.

\*/

private boolean validateInputs() {

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Input Error", ButtonType.OK);

if (name\_field.getText().equals("")

|| type\_field.getText().equals("")

|| quantity\_field.getText().equals("")

|| shelf\_choicebox.getValue() == null) {

IOAlert.setContentText("You must fill out the product to continue");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (contract\_choicebox.getValue() == null){

IOAlert.setContentText("You must add contract first to add products");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(quantity\_field.getText())){

IOAlert.setContentText("Incorrect input for product number - you must put a positive number");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if(!isShelfFree(shelf\_choicebox.getValue().getNumber(), Integer.parseInt(quantity\_field.getText())))

{

IOAlert.setContentText("Not enough space on the shelf to add new product!");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

return true;

}

/\*\*

\* Создает новый объект товара на основе введенных данных.

\* @return новый объект товара

\*/

private Product createProductFromInput() {

Product product = new Product();

product.setName(capitalize(name\_field.getText().toLowerCase()));

product.setType(capitalize(type\_field.getText().toLowerCase()));

product.setQuantity(Integer.valueOf(quantity\_field.getText()));

product.setShelf(shelf\_choicebox.getValue());

product.setContract(contract\_choicebox.getValue());

return product;

}

/\*\*

\* Делает первую букву строки прописной.

\* @return Строка с большой буквы

\*/

private static String capitalize(String str)

{

if (str == null || str.length() == 0) {

return str;

}

return str.substring(0, 1).toUpperCase() + str.substring(1);

}

/\*\*

\* Проверка стеллажа на вместимость добавляемого товара.

\* @return Заполненность стеллажа

\*/

private boolean isShelfFree(Integer number, Integer quantity){

Shelf shelf = shelfService.getShelf(number);

List<Product> products = shelf.getProductList();

Integer fullness = 0;

for (Product product : products) {

fullness += product.getQuantity();

}

return fullness + quantity <= shelf.getCapacity();

}

/\*\*

\* Проверка введенной строки на число.

\* @return true - число больше 0/false

\*/

private static boolean isNumeric(String str) {

try {

return Double.parseDouble(str) <= 0;

} catch(NumberFormatException e){

return true;

}

}

/\*\*

\* Задержка закрытия окна добавления до нажатия клавишы сохранения.

\*/

private void delayWindowClose(ActionEvent event) {

PauseTransition delay = new PauseTransition(Duration.seconds(1));

delay.setOnFinished(event2 -> closeWindow(event));

delay.play();

}

/\*\*

\* Закрытия окна по кнопке.

\*/

@FXML

private void closeWindow(ActionEvent event) {

SceneController.close(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

ObservableList<Contract> contractObservableList = FXCollections.observableArrayList(contractService.getContracts());

ObservableList<Shelf> shelfObservableList = FXCollections.observableArrayList(shelfService.getShelves());

ObservableList<Product> productObservableList = FXCollections.observableArrayList(productService.getProducts());

DistinctObservableList<Product> distinctListNames = new DistinctObservableList<>(productObservableList);

distinctListNames.distinct(Product::getName);

DistinctObservableList<Product> distinctListTypes = new DistinctObservableList<>(productObservableList);

distinctListTypes.distinct(Product::getType);

name\_comboBox.getItems().addAll(distinctListNames.getFilteredList());

type\_comboBox.getItems().addAll(distinctListTypes.getFilteredList());

name\_comboBox.getItems().add(null);

type\_comboBox.getItems().add(null);

ComboBoxUtil.configureNameComboBox(name\_comboBox);

ComboBoxUtil.configureTypeComboBox(type\_comboBox);

contract\_choicebox.getItems().addAll(contractObservableList);

shelf\_choicebox.getItems().addAll(shelfObservableList);

}

}

AddRoomController

package com.umler.warehouses.AddControllers;

import com.umler.warehouses.Controllers.SceneController;

import com.umler.warehouses.Helpers.UpdateStatus;

import com.umler.warehouses.Model.Room;

import com.umler.warehouses.Services.RoomService;

import javafx.animation.PauseTransition;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

import javafx.scene.control.TextField;

import javafx.util.Duration;

import java.util.Objects;

public class AddRoomController {

@FXML

public TextField number\_field;

@FXML

public TextField capacity\_field;

RoomService roomService = new RoomService();

/\*\*

\* Сохраняет изменения в базе данных после добавления комнаты.

\* Если данные введены корректно, то создается новый объект компании и контракта и происходит обновление базы данных.

\* Если данные введены некорректно, то выводится сообщение об ошибке.

\*/

@FXML

private void saveNewRoomToDb(ActionEvent event){

if (validateInputs()) {

Room room = createRoomFromInput();

boolean isSaved = new RoomService().createRoom(room);

if (isSaved) {

UpdateStatus.setIsRoomAdded(true);

delayWindowClose(event);

}

}

}

/\*\*

\* Проверяет корректность введенных данных.

\* Если все поля заполнены корректно, возвращает true.

\* Если есть незаполненные поля или данные введены некорректно, выводит сообщение об ошибке и возвращает false.

\*/

private boolean validateInputs() {

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Input Error", ButtonType.OK);

if (number\_field.getText().equals("")

|| capacity\_field.getText().equals("")) {

IOAlert.setContentText("You must fill out the room to continue");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (!isNumberExist(Integer.valueOf(number\_field.getText()))){

IOAlert.setContentText("Incorrect input for ROOM NUMBER - A ROOM with this number already exists");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(number\_field.getText())){

IOAlert.setContentText("Incorrect input for room number - you must put a positive number");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(capacity\_field.getText())){

IOAlert.setContentText("Incorrect input for capacity - you must put a positive number");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

return true;

}

/\*\*

\* Проверка что введеный номер помещения уже существует.

\*/

private boolean isNumberExist(Integer number){

for (Room rooms : roomService.getRooms()){

if (Objects.equals(rooms.getNumber(), number))

return false;

}

return true;

}

/\*\*

\* Создает новый объект помещения на основе введенных данных.

\* @return новый объект помещения

\*/

private Room createRoomFromInput() {

Room room = new Room();

room.setNumber(Integer.valueOf(number\_field.getText()));

room.setCapacity(Integer.valueOf(capacity\_field.getText()));

return room;

}

/\*\*

\* Проверка введенной строки на число.

\* @return true - число больше 0/false

\*/

private boolean isNumeric(String str) {

try {

return Double.parseDouble(str) <= 0;

} catch(NumberFormatException e){

return true;

}

}

/\*\*

\* Задержка закрытия окна добавления до нажатия клавишы сохранения.

\*/

private void delayWindowClose(ActionEvent event) {

PauseTransition delay = new PauseTransition(Duration.seconds(1));

delay.setOnFinished(event2 -> closeWindow(event));

delay.play();

}

/\*\*

\* Закрытия окна по кнопке.

\*/

@FXML

private void closeWindow(ActionEvent event) {

SceneController.close(event);

}

}

AddShelfController

package com.umler.warehouses.AddControllers;

import com.umler.warehouses.Controllers.SceneController;

import com.umler.warehouses.Helpers.UpdateStatus;

import com.umler.warehouses.Model.Shelf;

import com.umler.warehouses.Model.Room;

import com.umler.warehouses.Services.ShelfService;

import com.umler.warehouses.Services.RoomService;

import javafx.animation.PauseTransition;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

import javafx.scene.control.ChoiceBox;

import javafx.scene.control.TextField;

import javafx.util.Duration;

import java.net.URL;

import java.util.List;

import java.util.Objects;

import java.util.ResourceBundle;

public class AddShelfController implements Initializable {

@FXML

public TextField number\_field;

@FXML

public TextField capacity\_field;

@FXML

public ChoiceBox<Room> room\_choicebox;

RoomService roomService = new RoomService();

ShelfService shelfService = new ShelfService();

/\*\*

\* Сохраняет изменения в базе данных после добавления стеллажа.

\* Если данные введены корректно, то создается новый объект компании и контракта и происходит обновление базы данных.

\* Если данные введены некорректно, то выводится сообщение об ошибке.

\*/

@FXML

private void saveNewShelfToDb(ActionEvent event){

if (validateInputs()) {

Shelf shelf = createShelfFromInput();

boolean isSaved = new ShelfService().createShelf(shelf);

if (isSaved) {

UpdateStatus.setIsShelfAdded(true);

delayWindowClose(event);

}

}

}

/\*\*

\* Проверяет корректность введенных данных.

\* Если все поля заполнены корректно, возвращает true.

\* Если есть незаполненные поля или данные введены некорректно, выводит сообщение об ошибке и возвращает false.

\*/

private boolean validateInputs() {

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Input Error", ButtonType.OK);

if (number\_field.getText().equals("")

|| capacity\_field.getText().equals("")

|| room\_choicebox.getValue() == null) {

IOAlert.setContentText("You must fill out the shelf to continue");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (!isNumberExist(Integer.valueOf(number\_field.getText()))){

IOAlert.setContentText("Incorrect input for SHELF NUMBER - SHELF with this number already exists");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if (isNumeric(number\_field.getText())){

IOAlert.setContentText("Incorrect input for shelf number - you must put a positive number");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

if(!isRoomFree(room\_choicebox.getValue().getNumber()))

{

IOAlert.setContentText("Not enough space in room to add new shelf!");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

return false;

}

return true;

}

/\*\*

\* Создает новый объект стеллажа на основе введенных данных.

\* @return новый объект стеллажа

\*/

private Shelf createShelfFromInput() {

Shelf shelf = new Shelf();

shelf.setNumber(Integer.valueOf(number\_field.getText().toLowerCase()));

shelf.setCapacity(Integer.valueOf(capacity\_field.getText().toLowerCase()));

shelf.setRoom(room\_choicebox.getValue());

return shelf;

}

/\*\*

\* Проверка что введеный номер стеллажа уже существует.

\*/

private boolean isNumberExist(Integer number){

for (Shelf shelf : shelfService.getShelves()){

if (Objects.equals(shelf.getNumber(), number))

return false;

}

return true;

}

/\*\*

\* Проверка помещения на возможность добавления нового стеллажа.

\* @return Заполненность помещения

\*/

private boolean isRoomFree(Integer number){

Room room = roomService.getRoom(number);

List<Shelf> shelves = room.getShelvesList();

return shelves.size() + 1 <= room.getCapacity();

}

/\*\*

\* Проверка введенной строки на число.

\* @return true - число больше 0/false

\*/

private boolean isNumeric(String str) {

try {

return Double.parseDouble(str) <= 0;

} catch(NumberFormatException e){

return true;

}

}

/\*\*

\* Задержка закрытия окна добавления до нажатия клавишы сохранения.

\*/

private void delayWindowClose(ActionEvent event) {

PauseTransition delay = new PauseTransition(Duration.seconds(1));

delay.setOnFinished(event2 -> closeWindow(event));

delay.play();

}

/\*\*

\* Закрытия окна по кнопке.

\*/

@FXML

private void closeWindow(ActionEvent event) {

SceneController.close(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

ObservableList<Room> roomObservableList = FXCollections.observableArrayList(roomService.getRooms());

room\_choicebox.getItems().addAll(roomObservableList);

}

}

AppClass

package com.umler.warehouses.AppClasses;  
  
import com.umler.warehouses.Helpers.HibernateUtil;  
import com.umler.warehouses.Model.\*;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import java.time.LocalDate;  
  
  
public class AppClass {  
 public static void main(String[] args){  
  
 Session session = HibernateUtil.getSessionFactory().openSession();  
 Transaction transaction = session.beginTransaction();  
  
 Contract contract = new Contract();  
 contract.setStartdate(LocalDate.of(2015,12,1));  
 contract.setEnddate(LocalDate.of(2025,12,1));  
 contract.setNumber(231);  
  
 Company company = new Company();  
  
 company.setAddress("St.Petersburg, " + "st.Popova, " + "5, " + "194000");  
 company.setName("LETI");  
 company.setPhoneNumber("9959895421");  
 company.setTin("1234567891");  
  
 Room room = new Room();  
 room.setNumber(1);  
 room.setCapacity(100);  
  
 Shelf shelf = new Shelf();  
 shelf.setNumber(1);  
 shelf.setCapacity(200);  
 shelf.setRoom(room);  
  
 Product product = new Product();  
 product.setName("DDDD");  
 product.setType("dsadasddas");  
  
 product.setShelf(shelf);  
 product.setContract(contract);  
  
 contract.setCompany(company);  
 shelf.setRoom(room);  
  
  
 session.save(company);  
 session.save(contract);  
 session.save(room);  
 session.save(shelf);  
 session.save(product);  
  
 transaction.commit();  
 System.exit(0);  
 }  
  
}

CompaniesController

package com.umler.warehouses.Controllers;

import java.awt.\*;

import java.io.\*;

import com.itextpdf.text.\*;

import com.itextpdf.text.Font;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import com.umler.warehouses.Helpers.CurrentUser;

import com.umler.warehouses.Model.\*;

import com.umler.warehouses.Services.CompanyService;

import com.umler.warehouses.Services.WarehouseService;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.collections.transformation.FilteredList;

import javafx.collections.transformation.SortedList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.control.cell.TextFieldTableCell;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.net.URL;

import java.util.Objects;

import java.util.ResourceBundle;

/\*\*

\* Контроллер для таблицы компаний.

\* @author Umler

\*/

public class CompaniesController implements Initializable

{

@FXML

public Button exit\_btn;

@FXML

public Button wrap\_btn;

@FXML

public Hyperlink current\_user;

@FXML

public Label search\_invalid\_label1;

@FXML

public Button refresh\_btn;

@FXML

private ChoiceBox<String> choice\_box;

@FXML

private TableColumn<Company, String> name\_column;

@FXML

private TableColumn<Company, String> address\_column;

@FXML

private TableColumn<Company, String> phone\_column;

@FXML

private TableColumn<Company, String> tin\_column;

@FXML

private TableView<Company> table;

@FXML

private TextField search;

@FXML

public Label fullness\_label;

WarehouseService warehouseService = new WarehouseService();

ObservableList<Company> CompanyList = FXCollections.observableArrayList();

CompanyService companyService = new CompanyService();

private static final Logger logger = LoggerFactory.getLogger("Warehouse Logger");

/\*\*

\* Мое исключение для записи в PDF файл

\*/

static class MyPDFException extends Exception

{

public MyPDFException()

{

super("There is nothing to save");

}

}

/\*\*

\* Мое исключение для записи в удаления строк таблицы

\*/

static class myDeleteException extends Exception

{

public myDeleteException()

{

super("Choose a row to delete");

}

}

private final String[] choices = {"Companies","Contracts","Products","Rooms/Shelves"};

/\*\*

\* Обработчик события выбора значения в выпадающем списке.

\* Получает выбранное значение и вызывает соответствующий метод в классе SceneController для отображения соответствующей сцены.

\* @param event событие выбора значения в выпадающем списке

\* @throws IOException если возникает ошибка ввода-вывода при отображении сцены

\*/

@FXML

private void getChoices(ActionEvent event) throws IOException {

logger.info("Choice box action");

String choice = choice\_box.getValue();

if (Objects.equals(choice, "Contracts"))

{

logger.info("Choice box Contracts selected");

SceneController.getContractsScene(event);

}

if (Objects.equals(choice, "Products"))

{

logger.info("Choice box Managers selected");

SceneController.getProductsScene(event);

}

if (Objects.equals(choice, "Rooms/Shelves"))

{

logger.info("Choice box Managers selected");

SceneController.getRoomsShelvesScene(event);

}

}

/\*\*

\* Устанавливает список компаний.

\* Добавляет в список компании из БД.

\*/

private void setCompanyList() {

CompanyList.clear();

CompanyList.addAll(companyService.getCompanies());

}

/\*\*

\* Метод для получения отфильтрованного и отсортированного списка отчетов.

\* Создает новый отфильтрованный список на основе исходного списка компании, используя фильтр из searchField.

\* Затем создает новый отсортированный список на основе отфильтрованного списка и связывает его с компаратором таблицы.

\* @return Отсортированный список компании.

\*/

private ObservableList<Company> getSortedList() {

SortedList<Company> sortedList = new SortedList<>(getFilteredList());

sortedList.comparatorProperty().bind(table.comparatorProperty());

return sortedList;

}

/\*\*

\* Метод для получения отфильтрованного списка компаний на основе заданного текстового фильтра.

\* Создает новый отфильтрованный список на основе исходного списка компаний, используя заданный текстовый фильтр.

\* Фильтр применяется к полям "Название компании", "Адресс компании", "Номер телефона компании", "ИНН" каждой компании.

\* @return Отфильтрованный список компаний.

\*/

private FilteredList<Company> getFilteredList() {

FilteredList<Company> filteredList = new FilteredList<>(CompanyList, b -> true);

search.textProperty().addListener((observable, oldValue, newValue) ->

filteredList.setPredicate(company -> {

if (newValue == null || newValue.isEmpty()) {

return true;

}

String lowerCaseFilter = newValue.toLowerCase();

if (company.getName().toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (company.getAddress().toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (company.getPhoneNumber().toLowerCase().contains(lowerCaseFilter)) {

return true;

} else return company.getTin().toLowerCase().contains(lowerCaseFilter);

}));

return filteredList;

}

/\*\*

\* Обработчик события удаления выбранных компаний из таблицы.

\* Удаляет выбранные компании из таблицы.

\* Если ни одна компания не выбрана, выбрасывает исключение myDeleteException.

\* После удаления компаний обновляет экран.

\* @param event Событие удаления компаний.

\*/

@FXML

private void delete(ActionEvent event)

{

try {

logger.debug("deleting a worker");

deleteRows(event);

logger.info("worker deleted");

}

catch (myDeleteException | IOException myEx){

logger.error("MyDeleteException " + myEx);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, myEx.getMessage(), ButtonType.OK);

IOAlert.setContentText(myEx.getMessage());

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события удаления выбранных компаний из таблицы.

\* Удаляет выбранные компании из таблицы.

\* Если ни одна компания не выбрана, выбрасывает исключение myDeleteException.

\* После удаления компаний обновляет экран.

\* @param event Событие удаления компаний.

\* @throws myDeleteException Если ни одна компания не выбрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

private void deleteRows(ActionEvent event) throws myDeleteException, IOException {

int selectedID = table.getSelectionModel().getSelectedIndex();

if (selectedID == -1) throw new myDeleteException();

else {

ObservableList<Company> selectedRows = table.getSelectionModel().getSelectedItems();

for (Company company : selectedRows) {

companyService.deleteCompany(company);

}

refreshScreen(event);

}

}

/\*\*

\* Обработчик события сохранения списка компаний в файл.

\* Сохраняет список компаний в файл "saves/save\_company.csv".

\* Если произошла ошибка ввода-вывода при сохранении, выбрасывает исключение IOException.

\* После сохранения открывает папку "saves".

\*/

@FXML

private void save()

{

try

{

logger.debug("saving to file");

BufferedWriter writer = new BufferedWriter(new FileWriter("saves/save\_company.csv"));

for(Company companies : CompanyList)

{

writer.write(companies.getName() + ";" + companies.getAddress() + ";"

+ companies.getPhoneNumber() + ";"

+ companies.getTin());

writer.newLine();

}

writer.close();

Desktop.getDesktop().open(new File("saves"));

logger.info("saved to file");

}

catch (IOException e)

{

logger.warn("Exception " + e);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Error!", ButtonType.OK);

IOAlert.setContentText("Error");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события изменения названия компании в таблице.

\* @param editEvent Событие изменения названия компании в таблице.

\*/

@FXML

private void editName(TableColumn.CellEditEvent<Company, String> editEvent)

{

Company selectedCompany = table.getSelectionModel().getSelectedItem();

selectedCompany.setName(editEvent.getNewValue());

companyService.updateCompany(selectedCompany);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения адресса компании в таблице.

\* @param editEvent Событие изменения адресса компании в таблице.

\*/

@FXML

private void editAddress(TableColumn.CellEditEvent<Company, String> editEvent)

{

Company selectedCompany = table.getSelectionModel().getSelectedItem();

selectedCompany.setAddress(editEvent.getNewValue());

companyService.updateCompany(selectedCompany);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения телефона компании в таблице.

\* @param editEvent Событие изменения телефона компании в таблице.

\*/

@FXML

private void editPhone(TableColumn.CellEditEvent<Company, String> editEvent)

{

Company selectedCompany = table.getSelectionModel().getSelectedItem();

if(editEvent.getNewValue().length() == 10 && editEvent.getNewValue().matches("\\d+"))

{

selectedCompany.setPhoneNumber(editEvent.getNewValue());

companyService.updateCompany(selectedCompany);

}

else

{

logger.warn("Editing failing");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "EditException", ButtonType.OK);

IOAlert.setContentText("You must input number witch length 10");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

table.refresh();

}

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения ИНН компании в таблице.

\* @param editEvent Событие изменения ИНН компании в таблице.

\*/

@FXML

private void editTIN(TableColumn.CellEditEvent<Company, String> editEvent)

{

Company selectedCompany = table.getSelectionModel().getSelectedItem();

if(editEvent.getNewValue().length() == 10 && editEvent.getNewValue().matches("\\d+"))

{

selectedCompany.setTin(editEvent.getNewValue());

companyService.updateCompany(selectedCompany);

}

else

{

logger.warn("Editing failing");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "EditException", ButtonType.OK);

IOAlert.setContentText("You must input number witch length 10");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

table.refresh();

}

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события нажатия на кнопку сохранения таблицы компаний в PDF файл.

\* Сохраняет данные из таблицы в файл "pdf/report\_companies.pdf".

\* Если список компаний пуст, выбрасывает исключение MyPDFException.

\* Если возникает ошибка ввода-вывода, выводит сообщение об ошибке.

\*/

@FXML

private void toPDF(ActionEvent event) throws IOException {

try {

// log.debug("Saving to PDF");

Document my\_pdf\_report = new Document();

PdfWriter.getInstance(my\_pdf\_report, new FileOutputStream("report\_companies.pdf"));

my\_pdf\_report.open();

PdfPTable my\_report\_table = new PdfPTable(4);

PdfPCell table\_cell;

my\_report\_table.setHeaderRows(1);

my\_report\_table.addCell(new PdfPCell(new Phrase("Name", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Address", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Phone", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("TIN", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

if (CompanyList.isEmpty()) throw new MyPDFException();

for(Company companies : CompanyList)

{

table\_cell=new PdfPCell(new Phrase(companies.getName()));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(companies.getAddress()));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(companies.getPhoneNumber()));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(companies.getTin()));

my\_report\_table.addCell(table\_cell);

}

my\_pdf\_report.add(my\_report\_table);

my\_pdf\_report.close();

// log.info("Saved to PDF");

}

catch (FileNotFoundException | DocumentException | MyPDFException e)

{

// log.warn("Exception " + e);

e.printStackTrace();

}

refreshScreen(event);

}

/\*\*

\* Выход из окна программы.

\*/

public void ExitMainWindow() {

logger.debug("Closing main window");

exit\_btn.setOnAction(SceneController::close);

}

/\*\*

\* Сворачивание окна программы.

\*/

public void WrapMainWindow() {

logger.debug("Wrapping main window");

wrap\_btn.setOnAction(SceneController::wrap);

}

/\*\*

\* Обработчик события обновления экрана.

\* Вызывает метод SceneController для отображения сцены с компаниями.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void refreshScreen(ActionEvent event) throws IOException {

SceneController.getCompaniesScene(event);

}

/\*\*

\* Обработчик события смены экрана.

\* Вызывает метод SceneController для отображения сцены с авторизацией.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void changeUser(ActionEvent event) throws IOException {

SceneController.getLoginScene(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

warehouseService.getFullnessOfWarehouse();

fullness\_label.setText("Fullness: " + warehouseService.getFullnessOfWarehouse() + "%");

current\_user.setVisited(true);

current\_user.setText("User: " + CurrentUser.getCurrentUser().getName() + " / Change");

table.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

choice\_box.setValue("Companies");

choice\_box.getItems().addAll(choices);

setCompanyList();

name\_column.setCellValueFactory(new PropertyValueFactory<>("name"));

address\_column.setCellValueFactory(new PropertyValueFactory<>("address"));

phone\_column.setCellValueFactory(new PropertyValueFactory<>("phoneNumber"));

tin\_column.setCellValueFactory(new PropertyValueFactory<>("tin"));

table.setEditable(true);

name\_column.setCellFactory(TextFieldTableCell.forTableColumn());

address\_column.setCellFactory(TextFieldTableCell.forTableColumn());

phone\_column.setCellFactory(TextFieldTableCell.forTableColumn());

tin\_column.setCellFactory(TextFieldTableCell.forTableColumn());

table.setItems(getSortedList());

}

}

ContractsController

package com.umler.warehouses.Controllers;

import com.itextpdf.text.Font;

import com.itextpdf.text.\*;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import com.umler.warehouses.Converters.CustomIntegerStringConverter;

import com.umler.warehouses.Helpers.\*;

import com.umler.warehouses.Model.Company;

import com.umler.warehouses.Model.Contract;

import com.umler.warehouses.Services.CompanyService;

import com.umler.warehouses.Services.ContractService;

import com.umler.warehouses.Services.WarehouseService;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.collections.transformation.FilteredList;

import javafx.collections.transformation.SortedList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.control.\*;

import javafx.scene.control.cell.ChoiceBoxTableCell;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.control.cell.TextFieldTableCell;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.awt.\*;

import java.io.\*;

import java.net.URL;

import java.time.LocalDate;

import java.util.Objects;

import java.util.ResourceBundle;

/\*\*

\* Контроллер для таблицы контрактов.

\* @author Umler

\*/

public class ContractsController implements Initializable

{

@FXML

public Button exit\_btn;

@FXML

public Button wrap\_btn;

@FXML

public Hyperlink current\_user;

@FXML

public Label search\_invalid\_label1;

@FXML

public Button refresh\_btn;

@FXML

public Label fullness\_label;

@FXML

private ChoiceBox<String> choice\_box;

@FXML

private TableColumn<Contract, LocalDate> startdate\_column;

@FXML

private TableColumn<Contract, LocalDate> enddate\_column;

@FXML

private TableColumn<Contract, Integer> number\_column;

@FXML

private TableColumn<Contract, Company> company\_column;

@FXML

private TableView<Contract> table;

@FXML

private TextField search;

ObservableList<Contract> ContractList = FXCollections.observableArrayList();

ContractService contractService = new ContractService();

CompanyService companyService = new CompanyService();

WarehouseService warehouseService = new WarehouseService();

private static final Logger logger = LoggerFactory.getLogger("Contracts Logger");

/\*\*

\* Мое исключение для записи в PDF файл

\*/

static class MyPDFException extends Exception

{

public MyPDFException()

{

super("There is nothing to save");

}

}

/\*\*

\* Мое исключение для записи в удаления строк таблицы

\*/

static class myDeleteException extends Exception

{

public myDeleteException()

{

super("Choose a row to delete");

}

}

private final String[] choices = {"Contracts","Companies", "Products", "Rooms/Shelves"};

/\*\*

\* Обработчик события выбора значения в выпадающем списке.

\* Получает выбранное значение и вызывает соответствующий метод в классе SceneController для отображения соответствующей сцены.

\* @param event событие выбора значения в выпадающем списке

\* @throws IOException если возникает ошибка ввода-вывода при отображении сцены

\*/

@FXML

private void getChoices(ActionEvent event) throws IOException {

logger.info("Choice box action");

String choice = choice\_box.getValue();

if (Objects.equals(choice, "Companies"))

{

logger.info("Choice box Companies selected");

SceneController.getCompaniesScene(event);

}

if (Objects.equals(choice, "Products"))

{

logger.info("Choice box Products selected");

SceneController.getProductsScene(event);

}

if (Objects.equals(choice, "Rooms/Shelves"))

{

logger.info("Choice box RoomsShelves selected");

SceneController.getRoomsShelvesScene(event);

}

}

/\*\*

\* Обработчик события добавления нового контракта.

\* Вызывает метод NewWindowController для отображения окна добавления нового контракта.

\* Если контракт был успешно добавлен, обновляет экран с контрактами.

\* @param event Событие добавления нового контракта.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void add(ActionEvent event) throws IOException {

logger.debug("adding a contract");

NewWindowController.getNewContractCompanyWindow();

if(UpdateStatus.isIsContractCompanyAdded()) {

refreshScreen(event);

UpdateStatus.setIsContractCompanyAdded(false);

}

logger.info("contract added");

}

/\*\*

\* Устанавливает список контрактов.

\* Добавляет список контрактов из БД.

\*/

private void setContractList() {

logger.info("Contract list get from DB");

ContractList.clear();

ContractList.addAll(contractService.getContracts());

}

/\*\*

\* Метод для получения отфильтрованного и отсортированного списка контрактов.

\* Создает новый отфильтрованный список на основе исходного списка контрактов, используя фильтр из searchField.

\* Затем создает новый отсортированный список на основе отфильтрованного списка и связывает его с компаратором таблицы.

\* @return Отсортированный список контрактов.

\*/

private ObservableList<Contract> getSortedList() {

logger.info("ObservableList get");

SortedList<Contract> sortedList = new SortedList<>(getFilteredList());

sortedList.comparatorProperty().bind(table.comparatorProperty());

return sortedList;

}

/\*\*

\* Метод для получения отфильтрованного списка контрактов на основе заданного текстового фильтра.

\* Создает новый отфильтрованный список на основе исходного списка контрактов, используя заданный текстовый фильтр.

\* Фильтр применяется к полям "Дата подписания", "Дата оканчания", "Номер контракта", "Компания, с которой заключен договор" каждого контракта.

\* @return Отфильтрованный список контрактов.

\*/

private FilteredList<Contract> getFilteredList() {

logger.info("Filtering list with search");

FilteredList<Contract> filteredList = new FilteredList<>(ContractList, b -> true);

search.textProperty().addListener((observable, oldValue, newValue) ->

filteredList.setPredicate(contract -> {

if (newValue == null || newValue.isEmpty()) {

return true;

}

String lowerCaseFilter = newValue.toLowerCase();

if (date\_converter(contract.getStartdate().toString()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (date\_converter(contract.getEnddate().toString()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (String.valueOf(contract.getNumber()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else return contract.getCompany().getName().toLowerCase().contains(lowerCaseFilter);

}));

return filteredList;

}

/\*\*

\* Конвертация даты для поиска в Российском формте dd.MM.yyyy.

\*/

private String date\_converter(String temp){

logger.info("Date converting for the search in Russian style");

String[] temp2 = temp.split("-");

return temp2[2] + '.' + temp2[1] + '.' + temp2[0];

}

/\*\*

\* Обработчик события удаления выбранных контрактов из таблицы.

\* Удаляет выбранные контракты из таблицы.

\* Если ни один контракт не выбран, выбрасывает исключение myDeleteException.

\* После удаления контрактов обновляет экран.

\* @param event Событие удаления контрактов.

\*/

@FXML

private void delete(ActionEvent event)

{

try {

logger.debug("deleting a contract");

deleteRows(event);

logger.info("contract deleted");

}

catch (myDeleteException | IOException myEx){

logger.error("MyDeleteException " + myEx);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, myEx.getMessage(), ButtonType.OK);

IOAlert.setContentText(myEx.getMessage());

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события удаления выбранных контрактов из таблицы.

\* Удаляет выбранные контракты из таблицы.

\* Если ни один контракт не выбран, выбрасывает исключение myDeleteException.

\* После удаления контрактов обновляет экран.

\* @param event Событие удаления контрактов.

\* @throws ContractsController.myDeleteException Если ни один контракт не выбрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

private void deleteRows(ActionEvent event) throws myDeleteException, IOException {

logger.info("Deleting rows");

int selectedID = table.getSelectionModel().getSelectedIndex();

if (selectedID == -1) throw new myDeleteException();

else {

ObservableList<Contract> selectedRows = table.getSelectionModel().getSelectedItems();

for (Contract contract : selectedRows) {

contractService.deleteContract(contract);

}

refreshScreen(event);

}

}

/\*\*

\* Обработчик события сохранения списка контрактов в файл.

\* Сохраняет список контрактов в файл "saves/save\_contract.csv".

\* Если произошла ошибка ввода-вывода при сохранении, выбрасывает исключение IOException.

\* После сохранения открывает папку "saves".

\*/

@FXML

private void save()

{

try

{

logger.debug("saving to file");

BufferedWriter writer = new BufferedWriter(new FileWriter("saves/save\_contract.csv"));

for(Contract companies : ContractList)

{

writer.write(companies.getStartdate() + ";" + companies.getEnddate() + ";"

+ companies.getNumber() + ";" + companies.getCompany().getName());

writer.newLine();

}

writer.close();

Desktop.getDesktop().open(new File("saves"));

logger.info("saved to file");

}

catch (IOException e)

{

logger.warn("Exception " + e);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Error!", ButtonType.OK);

IOAlert.setContentText("Error");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события изменения даты подписания контракта в таблице.

\* @param editEvent Событие изменения даты подписания компании в таблице.

\*/

@FXML

private void editStartDate(TableColumn.CellEditEvent<Contract, LocalDate> editEvent) {

Contract selectedContract = table.getSelectionModel().getSelectedItem();

System.out.println(editEvent.getNewValue());

selectedContract.setStartdate(editEvent.getNewValue());

contractService.updateContract(selectedContract);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения даты окончания контракта в таблице.

\* @param editEvent Событие изменения даты окончания компании в таблице.

\*/

@FXML

private void editEndDate(TableColumn.CellEditEvent<Contract, LocalDate> editEvent)

{

Contract selectedContract = table.getSelectionModel().getSelectedItem();

selectedContract.setEnddate(editEvent.getNewValue());

contractService.updateContract(selectedContract);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения номера контракта в таблице.

\* @param editEvent Событие изменения ИНН компании в таблице.

\*/

@FXML

private void editNumber(TableColumn.CellEditEvent<Contract, Integer> editEvent)

{

Contract selectedContract = table.getSelectionModel().getSelectedItem();

selectedContract.setNumber(editEvent.getNewValue());

contractService.updateContract(selectedContract);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения компании привязанной к контракту в таблице.

\* @param editEvent Событие изменения компании привязанной к контракту в таблице.

\*/

@FXML

private void editCompany(TableColumn.CellEditEvent<Contract, Company> editEvent)

{

Contract selectedContract = table.getSelectionModel().getSelectedItem();

Company company = editEvent.getNewValue();

selectedContract.setCompany(company);

contractService.updateContract(selectedContract);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события нажатия на кнопку сохранения таблицы контрактов в PDF файл.

\* Сохраняет данные из таблицы в файл "pdf/report\_contracts.pdf".

\* Если список контрактов пуст, выбрасывает исключение MyPDFException.

\* Если возникает ошибка ввода-вывода, выводит сообщение об ошибке.

\*/

@FXML

private void toPDF(ActionEvent event) throws IOException {

try {

logger.debug("Saving to PDF");

Document my\_pdf\_report = new Document();

PdfWriter.getInstance(my\_pdf\_report, new FileOutputStream("report\_contracts.pdf"));

my\_pdf\_report.open();

PdfPTable my\_report\_table = new PdfPTable(4);

PdfPCell table\_cell;

my\_report\_table.setHeaderRows(1);

my\_report\_table.addCell(new PdfPCell(new Phrase("StartDate", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("EndDate", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Number", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Company", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

if (ContractList.isEmpty()) throw new MyPDFException();

for(Contract contracts : ContractList)

{

table\_cell=new PdfPCell(new Phrase(String.valueOf(contracts.getStartdate())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(String.valueOf(contracts.getEnddate())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(String.valueOf(contracts.getNumber())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(contracts.getCompany().getName()));

my\_report\_table.addCell(table\_cell);

}

my\_pdf\_report.add(my\_report\_table);

my\_pdf\_report.close();

logger.info("Saved to PDF");

}

catch (FileNotFoundException | DocumentException | MyPDFException e)

{

logger.warn("Exception " + e);

e.printStackTrace();

}

refreshScreen(event);

}

/\*\*

\* Выход из окна программы.

\*/

public void ExitMainWindow() {

logger.debug("Closing main window");

exit\_btn.setOnAction(SceneController::close);

}

/\*\*

\* Сворачивание окна программы.

\*/

public void WrapMainWindow() {

logger.debug("Wrapping main window");

wrap\_btn.setOnAction(SceneController::wrap);

}

/\*\*

\* Обработчик события обновления экрана.

\* Вызывает метод SceneController для отображения сцены с контрактами.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void refreshScreen(ActionEvent event) throws IOException {

logger.info("Refreshing screen");

SceneController.getContractsScene(event);

}

/\*\*

\* Обработчик события смены экрана.

\* Вызывает метод SceneController для отображения сцены с авторизацией.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void changeUser(ActionEvent event) throws IOException {

logger.info("Changing user");

SceneController.getLoginScene(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

logger.info("Contract initializing");

logger.debug("Fullness getting");

warehouseService.getFullnessOfWarehouse();

fullness\_label.setText("Fullness: " + warehouseService.getFullnessOfWarehouse() + "%");

logger.debug("Current user getting");

current\_user.setVisited(true);

current\_user.setText("User: " + CurrentUser.getCurrentUser().getName() + " / Change");

table.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

logger.debug("Contract getting");

choice\_box.setValue("Contracts");

choice\_box.getItems().addAll(choices);

setContractList();

logger.debug("CellValueFactory setting");

startdate\_column.setCellValueFactory(new PropertyValueFactory<>("startdate"));

enddate\_column.setCellValueFactory(new PropertyValueFactory<>("enddate"));

number\_column.setCellValueFactory(new PropertyValueFactory<>("number"));

company\_column.setCellValueFactory(new PropertyValueFactory<>("company"));

logger.debug("Company list setting");

ObservableList<Company> companiesObservableList = FXCollections.observableArrayList(companyService.getCompanies());

logger.debug("Editable table setting");

table.setEditable(true);

logger.debug("CellFactory setting");

startdate\_column.setCellFactory(new LocalStartDateCellFactory());

enddate\_column.setCellFactory(new LocalEndDateCellFactory());

number\_column.setCellFactory(TextFieldTableCell.forTableColumn(new CustomIntegerStringConverter()));

company\_column.setCellFactory(ChoiceBoxTableCell.forTableColumn(companiesObservableList));

logger.info("Table setting items");

table.setItems(getSortedList());

}

}

LoginController

package com.umler.warehouses.Controllers;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

import com.umler.warehouses.Helpers.CurrentUser;

import com.umler.warehouses.Model.User;

import com.umler.warehouses.Services.UserService;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*\*

\* Контроллер для таблицы пользователей.

\* @author Umler

\*/

public class LoginController implements Initializable {

public Button LoginButton;

public Button exit\_btn;

public Button wrap\_btn;

@FXML

private Label invalid\_label;

@FXML

private TextField username\_box;

@FXML

private TextField password\_box;

UserService userService = new UserService();

private static final Logger logger = LoggerFactory.getLogger("LoginScene Logger");

/\*\*

\* Обработчик события открытия главного интерфейса программы.

\* Получает выбранное значение и вызывает соответствующий метод в классе SceneController для отображения соответствующей сцены.

\* @param event событие открытия главного интерфейса программы.

\* @throws IOException если возникает ошибка ввода-вывода при отображении сцены.

\*/

@FXML

void showVisitScreen(ActionEvent event) throws IOException {

if (validateLogin())

{

SceneController.getContractsScene(event);

}

else

{

logger.warn("Invalid credentials");

username\_box.clear();

password\_box.clear();

invalid\_label.setText("Invalid credentials");

}

}

/\*\*

\* Проверяет корректность введенных данных.

\* Если введеные данные для входа есть в базе, возвращает true.

\* Если есть незаполненные поля или данные введенные не соответсвуют БД, выводит сообщение об ошибке и возвращает false.

\*/

private boolean validateLogin() {

User user = userService.getConnectedUser(username\_box.getText(), password\_box.getText());

if (user == null) {

return false;

}

CurrentUser.setCurrentUser(user);

return true;

}

/\*\*

\* Выход из окна программы.

\*/

public void ExitLoginWindow() {

logger.debug("Closing login window");

exit\_btn.setOnAction(SceneController::close);

}

/\*\*

\* Сворачивание окна программы.

\*/

public void WrapLoginWindow() {

logger.debug("Wrapping login window");

wrap\_btn.setOnAction(SceneController::wrap);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb) {

// TODO

}

}

NewWindowController

package com.umler.warehouses.Controllers;  
  
import com.umler.warehouses.Enums.ScenePath;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import javafx.stage.StageStyle;  
  
import java.io.IOException;  
import java.util.Objects;  
  
/\*\*  
 \* Контроллер переключения.  
 \* @author Umler  
 \*/  
public class NewWindowController {  
  
 static double x;  
 static double y;  
  
 /\*\*  
 \* Открывает новое окно для добавления нового стеллажа.  
 \* @throws IOException Если произошла ошибка ввода-вывода.  
 \*/  
 public static void getNewShelfWindow() throws IOException {  
 getPopUpWindow(ScenePath.ADDSHELF.getPath());  
 }  
  
 /\*\*  
 \* Открывает новое окно для добавления нового помещения.  
 \* @throws IOException Если произошла ошибка ввода-вывода.  
 \*/  
 public static void getNewRoomWindow() throws IOException {  
 getPopUpWindow(ScenePath.ADDROOM.getPath());  
 }  
  
 /\*\*  
 \* Открывает новое окно для добавления нового продукта.  
 \* @throws IOException Если произошла ошибка ввода-вывода.  
 \*/  
 public static void getNewProductWindow() throws IOException {  
 getPopUpWindow(ScenePath.ADDPRODUCT.getPath());  
 }  
  
 /\*\*  
 \* Открывает новое окно для добавления нового контракта и компании.  
 \* @throws IOException Если произошла ошибка ввода-вывода.  
 \*/  
 public static void getNewContractCompanyWindow() throws IOException {  
 getPopUpWindow(ScenePath.ADDCONTRACTCOMPANY.getPath());  
 }  
  
 /\*\*  
 \* Открывает новое всплывающее окно с заданным FXML-файлом.  
 \* @param path Путь к FXML-файлу в классе Enum ScenePath.  
 \* @throws IOException Если произошла ошибка ввода-вывода.  
 \*/  
 public static void getPopUpWindow(String path) throws IOException {  
 Stage stage = new Stage();  
 Pane main = FXMLLoader.load(Objects.requireNonNull(NewWindowController.class.getResource(path)));  
 controlDrag(main, stage);  
 stage.setScene(new Scene(main));  
 stage.initModality(Modality.APPLICATION\_MODAL);  
 stage.initStyle(StageStyle.UNDECORATED);  
 stage.setTitle("Popup window");  
 stage.getScene();  
 stage.showAndWait();  
 }  
  
 /\*\*  
 \* Окно можно перетаскивать по экрану при зажатии ЛКМ по окну.  
 \*/  
 public static void controlDrag(Pane main, Stage stage) {  
 main.setOnMousePressed(event -> {  
 x = stage.getX() - event.getScreenX();  
 y = stage.getY() - event.getScreenY();  
 });  
 main.setOnMouseDragged(event -> {  
 stage.setX(event.getScreenX() + x);  
 stage.setY(event.getScreenY() + y);  
 });  
 }  
  
}

ProductsController

package com.umler.warehouses.Controllers;

import com.itextpdf.text.Font;

import com.itextpdf.text.\*;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import com.umler.warehouses.Converters.CustomIntegerStringConverter;

import com.umler.warehouses.Helpers.CurrentUser;

import com.umler.warehouses.Helpers.UpdateStatus;

import com.umler.warehouses.Model.Contract;

import com.umler.warehouses.Model.Product;

import com.umler.warehouses.Model.Shelf;

import com.umler.warehouses.Services.ContractService;

import com.umler.warehouses.Services.ProductService;

import com.umler.warehouses.Services.ShelfService;

import com.umler.warehouses.Services.WarehouseService;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.collections.transformation.FilteredList;

import javafx.collections.transformation.SortedList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.control.\*;

import javafx.scene.control.cell.ChoiceBoxTableCell;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.control.cell.TextFieldTableCell;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.awt.\*;

import java.io.\*;

import java.net.URL;

import java.util.List;

import java.util.Objects;

import java.util.ResourceBundle;

/\*\*

\* Контроллер для таблицы товаров.

\* @author Umler

\*/

public class ProductsController implements Initializable

{

@FXML

public Button exit\_btn;

@FXML

public Button wrap\_btn;

@FXML

public Hyperlink current\_user;

@FXML

public Label search\_invalid\_label1;

@FXML

public TableColumn<Product, String> name\_column;

@FXML

public TableColumn<Product, String> type\_column;

@FXML

public TableColumn<Product, Integer> quantity\_column;

@FXML

public TableColumn<Product, Shelf> shelf\_column;

@FXML

public Button refresh\_btn;

@FXML

private TableColumn<Product, Contract> contract\_column;

@FXML

private ChoiceBox<String> choice\_box;

@FXML

private TableView<Product> table;

@FXML

private TextField search;

@FXML

public Label fullness\_label;

WarehouseService warehouseService = new WarehouseService();

ObservableList<Product> ProductList = FXCollections.observableArrayList();

ProductService productService = new ProductService();

ContractService contractService = new ContractService();

ShelfService shelfService = new ShelfService();

private static final Logger logger = LoggerFactory.getLogger("Warehouse Logger");

/\*\*

\* Мое исключение для записи в PDF файл

\*/

static class MyPDFException extends Exception

{

public MyPDFException()

{

super("There is nothing to save");

}

}

/\*\*

\* Мое исключение для записи в удаления строк таблицы

\*/

static class myDeleteException extends Exception

{

public myDeleteException()

{

super("Choose a row to delete");

}

}

private final String[] choices = {"Products","Companies", "Contracts", "Rooms/Shelves"};

/\*\*

\* Обработчик события выбора значения в выпадающем списке.

\* Получает выбранное значение и вызывает соответствующий метод в классе SceneController для отображения соответствующей сцены.

\* @param event событие выбора значения в выпадающем списке

\* @throws IOException если возникает ошибка ввода-вывода при отображении сцены

\*/

@FXML

private void getChoices(ActionEvent event) throws IOException {

logger.info("Choice box action");

String choice = choice\_box.getValue();

if (Objects.equals(choice, "Companies"))

{

logger.info("Choice box Managers selected");

SceneController.getCompaniesScene(event);

}

if (Objects.equals(choice, "Contracts"))

{

logger.info("Choice box Managers selected");

SceneController.getContractsScene(event);

}

if (Objects.equals(choice, "Rooms/Shelves"))

{

logger.info("Choice box Managers selected");

SceneController.getRoomsShelvesScene(event);

}

}

/\*\*

\* Обработчик события добавления нового товара.

\* Вызывает метод NewWindowController для отображения окна добавления нового товара.

\* Если товар был успешно добавлен, обновляет экран с товарами.

\* @param event Событие добавления нового товара.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void add(ActionEvent event) throws IOException {

logger.debug("adding a room");

NewWindowController.getNewProductWindow();

if(UpdateStatus.isIsProductAdded()) {

refreshScreen(event);

UpdateStatus.setIsProductAdded(false);

}

logger.info("room added");

}

/\*\*

\* Устанавливает список товаров.

\* Добавляет список товаров из БД.

\*/

private void setProductList() {

ProductList.clear();

ProductList.addAll(productService.getProducts());

}

/\*\*

\* Метод для получения отфильтрованного и отсортированного списка товаров.

\* Создает новый отфильтрованный список на основе исходного списка товаров, используя фильтр из searchField.

\* Затем создает новый отсортированный список на основе отфильтрованного списка и связывает его с компаратором таблицы.

\* @return Отсортированный список товаров.

\*/

private ObservableList<Product> getSortedList() {

SortedList<Product> sortedList = new SortedList<>(getFilteredList());

sortedList.comparatorProperty().bind(table.comparatorProperty());

return sortedList;

}

/\*\*

\* Метод для получения отфильтрованного списка товаров на основе заданного текстового фильтра.

\* Создает новый отфильтрованный список на основе исходного списка товаров, используя заданный текстовый фильтр.

\* Фильтр применяется к полям "Название товара", "Тип товара", "Кол-во товара определенного вида", "Контракт, по которому товары добавлены на склад" каждого товара.

\* @return Отфильтрованный список товаров.

\*/

private FilteredList<Product> getFilteredList() {

FilteredList<Product> filteredList = new FilteredList<>(ProductList, b -> true);

search.textProperty().addListener((observable, oldValue, newValue) ->

filteredList.setPredicate(product -> {

if (newValue == null || newValue.isEmpty()) {

return true;

}

String lowerCaseFilter = newValue.toLowerCase();

if (product.getName().toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (product.getType().toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (String.valueOf(product.getQuantity()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else return String.valueOf(product.getContract().getNumber()).toLowerCase().contains(lowerCaseFilter);

}));

return filteredList;

}

/\*\*

\* Обработчик события удаления выбранных товаров из таблицы.

\* Удаляет выбранные товары из таблицы.

\* Если ни один товар не выбран, выбрасывает исключение myDeleteException.

\* После удаления товаров обновляет экран.

\* @param event Событие удаления товаров.

\*/

@FXML

private void delete(ActionEvent event)

{

try {

logger.debug("deleting a product");

deleteRows(event);

logger.info("product deleted");

}

catch (myDeleteException | IOException myEx){

logger.error("MyDeleteException " + myEx);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, myEx.getMessage(), ButtonType.OK);

IOAlert.setContentText(myEx.getMessage());

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события удаления выбранных товаров из таблицы.

\* Удаляет выбранные товары из таблицы.

\* Если ни один товар не выбран, выбрасывает исключение myDeleteException.

\* После удаления товаров обновляет экран.

\* @param event Событие удаления товаров.

\* @throws ProductsController.myDeleteException Если ни один товар не выбран.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

private void deleteRows(ActionEvent event) throws myDeleteException, IOException {

int selectedID = table.getSelectionModel().getSelectedIndex();

if (selectedID == -1) throw new myDeleteException();

else {

ObservableList<Product> selectedRows = table.getSelectionModel().getSelectedItems();

for (Product product : selectedRows) {

productService.deleteProduct(product);

}

refreshScreen(event);

}

}

/\*\*

\* Обработчик события сохранения списка товаров в файл.

\* Сохраняет список товаров в файл "saves/save\_contract.csv".

\* Если произошла ошибка ввода-вывода при сохранении, выбрасывает исключение IOException.

\* После сохранения открывает папку "saves".

\*/

@FXML

private void save()

{

try

{

// logger.debug("saving to file");

BufferedWriter writer = new BufferedWriter(new FileWriter("saves/save\_product.csv"));

for(Product companies : ProductList)

{

writer.write(companies.getName() + ";" + companies.getType() + ";"

+ companies.getQuantity() + ";" + companies.getContract().getNumber());

writer.newLine();

}

writer.close();

Desktop.getDesktop().open(new File("saves"));

// logger.info("saved to file");

}

catch (IOException e)

{

// logger.warn("Exception " + e);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Error!", ButtonType.OK);

IOAlert.setContentText("Error");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события изменения названия товара в таблице.

\* @param editEvent Событие изменения названия товара в таблице.

\*/

@FXML

private void editName(TableColumn.CellEditEvent<Product, String> editEvent)

{

Product selectedProduct = table.getSelectionModel().getSelectedItem();

selectedProduct.setName(editEvent.getNewValue());

productService.updateProduct(selectedProduct);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения типа товара в таблице.

\* @param editEvent Событие изменения типа товара в таблице.

\*/

@FXML

private void editType(TableColumn.CellEditEvent<Product, String> editEvent)

{

Product selectedProduct = table.getSelectionModel().getSelectedItem();

selectedProduct.setType(editEvent.getNewValue());

productService.updateProduct(selectedProduct);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения кол-ва товара в таблице.

\* @param editEvent Событие изменения кол-ва товара в таблице.

\*/

@FXML

private void editQuantity(TableColumn.CellEditEvent<Product, Integer> editEvent)

{

Product selectedProduct = table.getSelectionModel().getSelectedItem();

Shelf shelf = selectedProduct.getShelf();

List<Product> products = shelf.getProductList();

Integer fullness = 0;

for (Product product : products) {

fullness += product.getQuantity();

}

if (fullness - selectedProduct.getQuantity() + editEvent.getNewValue() <= shelf.getCapacity())

{

selectedProduct.setQuantity(editEvent.getNewValue());

productService.updateProduct(selectedProduct);

}

else

{

logger.error("MyAddException");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "MyAddException", ButtonType.OK);

IOAlert.setContentText("MyAddException");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

table.refresh();

}

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения стеллажа на котором находится товар.

\* @param editEvent Событие изменения стеллажа на котором находится товар.

\*/

@FXML

private void editShelf(TableColumn.CellEditEvent<Product, Shelf> editEvent)

{

Product selectedProduct = table.getSelectionModel().getSelectedItem();

Shelf shelf = editEvent.getNewValue();

List<Product> products = shelf.getProductList();

Integer fullness = 0;

for (Product product : products) {

fullness += product.getQuantity();

}

if (shelf.getCapacity() - fullness - selectedProduct.getQuantity() >= 0)

{

selectedProduct.setShelf(shelf);

productService.updateProduct(selectedProduct);

}

else

{

logger.error("MyAddException");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "MyAddException", ButtonType.OK);

IOAlert.setContentText("No space in this shelf try another one / Or click on refresh");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

table.refresh();

}

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения контракта по которому товар находится на складе.

\* @param editEvent Событие изменения контракта по которому товар находится на складе.

\*/

@FXML

private void editContract(TableColumn.CellEditEvent<Product, Contract> editEvent)

{

Product selectedProduct = table.getSelectionModel().getSelectedItem();

Contract contract = editEvent.getNewValue();

selectedProduct.setContract(contract);

productService.updateProduct(selectedProduct);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события нажатия на кнопку сохранения таблицы товаров в PDF файл.

\* Сохраняет данные из таблицы в файл "pdf/report\_products.pdf".

\* Если список товаров пуст, выбрасывает исключение MyPDFException.

\* Если возникает ошибка ввода-вывода, выводит сообщение об ошибке.

\*/

@FXML

private void toPDF(ActionEvent event) throws IOException {

try {

logger.debug("Saving to PDF");

Document my\_pdf\_report = new Document();

PdfWriter.getInstance(my\_pdf\_report, new FileOutputStream("report\_products.pdf"));

my\_pdf\_report.open();

PdfPTable my\_report\_table = new PdfPTable(5);

PdfPCell table\_cell;

my\_report\_table.setHeaderRows(1);

my\_report\_table.addCell(new PdfPCell(new Phrase("Name", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Type", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Quantity", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Shelf", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table.addCell(new PdfPCell(new Phrase("Contract", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

if (ProductList.isEmpty()) throw new MyPDFException();

for(Product products : ProductList)

{

table\_cell=new PdfPCell(new Phrase(String.valueOf(products.getName())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(String.valueOf(products.getType())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(String.valueOf(products.getQuantity())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(String.valueOf(products.getShelf().getNumber())));

my\_report\_table.addCell(table\_cell);

table\_cell=new PdfPCell(new Phrase(String.valueOf(products.getContract().getNumber())));

my\_report\_table.addCell(table\_cell);

}

my\_pdf\_report.add(my\_report\_table);

my\_pdf\_report.close();

logger.info("Saved to PDF");

}

catch (FileNotFoundException | DocumentException | MyPDFException e)

{

logger.warn("Exception " + e);

e.printStackTrace();

}

refreshScreen(event);

}

/\*\*

\* Выход из окна программы.

\*/

public void ExitMainWindow() {

logger.debug("Closing main window");

exit\_btn.setOnAction(SceneController::close);

}

/\*\*

\* Сворачивание окна программы.

\*/

public void WrapMainWindow() {

logger.debug("Wrapping main window");

wrap\_btn.setOnAction(SceneController::wrap);

}

/\*\*

\* Обработчик события обновления экрана.

\* Вызывает метод SceneController для отображения сцены с товарами.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void refreshScreen(ActionEvent event) throws IOException {

SceneController.getProductsScene(event);

}

/\*\*

\* Обработчик события смены экрана.

\* Вызывает метод SceneController для отображения сцены с авторизацией.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void changeUser(ActionEvent event) throws IOException {

SceneController.getLoginScene(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

warehouseService.getFullnessOfWarehouse();

fullness\_label.setText("Fullness: " + warehouseService.getFullnessOfWarehouse() + "%");

current\_user.setVisited(true);

current\_user.setText("User: " + CurrentUser.getCurrentUser().getName() + " / Change");

table.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

choice\_box.setValue("Products");

choice\_box.getItems().addAll(choices);

setProductList();

name\_column.setCellValueFactory(new PropertyValueFactory<>("name"));

type\_column.setCellValueFactory(new PropertyValueFactory<>("type"));

quantity\_column.setCellValueFactory(new PropertyValueFactory<>("quantity"));

shelf\_column.setCellValueFactory(new PropertyValueFactory<>("shelf"));

contract\_column.setCellValueFactory(new PropertyValueFactory<>("contract"));

ObservableList<Contract> contractsObservableList = FXCollections.observableArrayList(contractService.getContracts());

ObservableList<Shelf> shelfsObservableList = FXCollections.observableArrayList(shelfService.getShelves());

table.setEditable(true);

name\_column.setCellFactory(TextFieldTableCell.forTableColumn());

type\_column.setCellFactory(TextFieldTableCell.forTableColumn());

quantity\_column.setCellFactory(TextFieldTableCell.forTableColumn(new CustomIntegerStringConverter()));

shelf\_column.setCellFactory(ChoiceBoxTableCell.forTableColumn(shelfsObservableList));

contract\_column.setCellFactory(ChoiceBoxTableCell.forTableColumn(contractsObservableList));

table.setItems(getSortedList());

}

}

RoomsShelvesController

package com.umler.warehouses.Controllers;

import com.itextpdf.text.Font;

import com.itextpdf.text.\*;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import com.umler.warehouses.Helpers.UpdateStatus;

import com.umler.warehouses.Model.Product;

import com.umler.warehouses.Model.Room;

import com.umler.warehouses.Model.Shelf;

import com.umler.warehouses.Services.RoomService;

import com.umler.warehouses.Services.ShelfService;

import com.umler.warehouses.Services.WarehouseService;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.collections.transformation.FilteredList;

import javafx.collections.transformation.SortedList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.control.\*;

import javafx.scene.control.cell.ChoiceBoxTableCell;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.control.cell.TextFieldTableCell;

import javafx.util.converter.IntegerStringConverter;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.awt.\*;

import java.io.\*;

import java.net.URL;

import java.util.List;

import java.util.Objects;

import java.util.ResourceBundle;

/\*\*

\* Контроллер для таблицы помещений и стеллажей.

\* @author Umler

\*/

public class RoomsShelvesController implements Initializable

{

@FXML

public Button exit\_btn;

@FXML

public Button wrap\_btn;

@FXML

public TableColumn<Shelf, Integer> capacity\_column;

@FXML

public TableColumn<Shelf, Integer> number\_column;

@FXML

public Label search\_invalid\_label11;

@FXML

public Label search\_invalid\_label111;

@FXML

public Button refresh\_btn;

@FXML

private TableColumn<Shelf, Room> room\_column;

@FXML

private ChoiceBox<String> choice\_box;

@FXML

private TableView<Shelf> table\_shelves;

@FXML

public TableView<Room> table\_rooms;

@FXML

public TableColumn<Room, Integer> roomnumber\_column;

@FXML

public TableColumn<Room, Integer> roomcapacity\_column;

@FXML

private TextField searchRoom;

@FXML

private TextField searchShelves;

@FXML

public Label fullness\_label;

WarehouseService warehouseService = new WarehouseService();

ObservableList<Shelf> ShelfList = FXCollections.observableArrayList();

ObservableList<Room> RoomList = FXCollections.observableArrayList();

ShelfService shelfService = new ShelfService();

RoomService roomService = new RoomService();

private static final Logger logger = LoggerFactory.getLogger("Warehouse Logger");

/\*\*

\* Мое исключение для записи в PDF файл

\*/

static class MyPDFException extends Exception

{

public MyPDFException()

{

super("There is nothing to save");

}

}

/\*\*

\* Мое исключение для записи в удаления строк таблицы

\*/

static class myDeleteException extends Exception

{

public myDeleteException()

{

super("Choose a row to delete");

}

}

private final String[] choices = {"Rooms/Shelves","Products","Companies", "Contracts"};

/\*\*

\* Обработчик события выбора значения в выпадающем списке.

\* Получает выбранное значение и вызывает соответствующий метод в классе SceneController для отображения соответствующей сцены.

\* @param event событие выбора значения в выпадающем списке

\* @throws IOException если возникает ошибка ввода-вывода при отображении сцены

\*/

@FXML

private void getChoices(ActionEvent event) throws IOException {

logger.info("Choice box action");

String choice = choice\_box.getValue();

if (Objects.equals(choice, "Companies"))

{

logger.info("Choice box Companies selected");

SceneController.getCompaniesScene(event);

}

if (Objects.equals(choice, "Contracts"))

{

logger.info("Choice box Contracts selected");

SceneController.getContractsScene(event);

}

if (Objects.equals(choice, "Products"))

{

logger.info("Choice box Products selected");

SceneController.getProductsScene(event);

}

}

/\*\*

\* Устанавливает список помещений.

\* Добавляет список помещений из БД.

\*/

private void setRoomList() {

RoomList.clear();

RoomList.addAll(roomService.getRooms());

}

/\*\*

\* Метод для получения отфильтрованного и отсортированного списка помещений.

\* Создает новый отфильтрованный список на основе исходного списка помещений, используя фильтр из searchField.

\* Затем создает новый отсортированный список на основе отфильтрованного списка и связывает его с компаратором таблицы.

\* @return Отсортированный список помещений.

\*/

private ObservableList<Room> getRoomsSortedList() {

SortedList<Room> sortedList = new SortedList<>(getRoomsFilteredList());

sortedList.comparatorProperty().bind(table\_rooms.comparatorProperty());

return sortedList;

}

/\*\*

\* Метод для получения отфильтрованного списка помещений на основе заданного текстового фильтра.

\* Создает новый отфильтрованный список на основе исходного списка помещений, используя заданный текстовый фильтр.

\* Фильтр применяется к полям "Номер", "Вместимость" каждого помещения.

\* @return Отфильтрованный список помещений.

\*/

private FilteredList<Room> getRoomsFilteredList() {

FilteredList<Room> filteredList = new FilteredList<>(RoomList, b -> true);

searchRoom.textProperty().addListener((observable, oldValue, newValue) ->

filteredList.setPredicate(room -> {

if (newValue == null || newValue.isEmpty()) {

return true;

}

String lowerCaseFilter = newValue.toLowerCase();

if (String.valueOf(room.getNumber()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else return (String.valueOf(room.getCapacity()).toLowerCase().contains(lowerCaseFilter));

}));

return filteredList;

}

/\*\*

\* Обработчик события добавления нового помещения.

\* Вызывает метод NewWindowController для отображения окна добавления нового помещения.

\* Если помещение был успешно добавлено, обновляет экран с помещениями.

\* @param event Событие добавления нового помещения.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void addRoom(ActionEvent event) throws IOException {

logger.debug("adding a room");

NewWindowController.getNewRoomWindow();

if(UpdateStatus.isIsRoomAdded()) {

refreshScreen(event);

UpdateStatus.setIsRoomAdded(false);

}

logger.info("room added");

}

/\*\*

\* Обработчик события удаления выбранных помещений из таблицы.

\* Удаляет выбранные помещения из таблицы.

\* Если ни одно помещение не выбрано, выбрасывает исключение myDeleteException.

\* После удаления помещений обновляет экран.

\* @param event Событие удаления помещений.

\*/

@FXML

private void deleteRooms(ActionEvent event)

{

try {

logger.debug("deleting a shelf");

deleteRoomRows(event);

logger.info("shelf deleted");

}

catch (myDeleteException | IOException myEx){

logger.error("MyDeleteException " + myEx);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, myEx.getMessage(), ButtonType.OK);

IOAlert.setContentText(myEx.getMessage());

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события удаления выбранных помещений из таблицы.

\* Удаляет выбранные помещения из таблицы.

\* Если ни одно помещение не выбрано, выбрасывает исключение myDeleteException.

\* @param event После удаления помещений обновляет экран.

\* @throws RoomsShelvesController.myDeleteException Если ни одно помещение не выбрано.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

private void deleteRoomRows(ActionEvent event) throws myDeleteException, IOException {

int selectedID = table\_rooms.getSelectionModel().getSelectedIndex();

if (selectedID == -1) throw new myDeleteException();

else {

ObservableList<Room> selectedRows = table\_rooms.getSelectionModel().getSelectedItems();

for (Room room : selectedRows) {

roomService.deleteRoom(room);

}

refreshScreen(event);

}

}

/\*\*

\* Обработчик события сохранения списка помещений в файл.

\* Сохраняет список помещений в файл "saves/save\_room.csv".

\* Если произошла ошибка ввода-вывода при сохранении, выбрасывает исключение IOException.

\* После сохранения открывает папку "saves".

\*/

@FXML

private void saveRooms()

{

try

{

// logger.debug("saving to file");

BufferedWriter writer = new BufferedWriter(new FileWriter("saves/save\_rooms.csv"));

for(Room room : RoomList)

{

writer.write(room.getNumber() + ";" + room.getCapacity());

writer.newLine();

}

writer.close();

Desktop.getDesktop().open(new File("saves"));

// logger.info("saved to file");

}

catch (IOException e)

{

// logger.warn("Exception " + e);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Error!", ButtonType.OK);

IOAlert.setContentText("Error");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события изменения номера помещения в таблице.

\* Проверка на существования помещения с введеным для изменения номером.

\* Вывод ошибки если помещение с таким номером уже существует.

\* @param editEvent Событие изменения названия помещения в таблице.

\*/

@FXML

private void editRoomNumber(TableColumn.CellEditEvent<Room, Integer> editEvent)

{

Room selectedRoom = table\_rooms.getSelectionModel().getSelectedItem();

for (Room rooms : roomService.getRooms()){

if (!Objects.equals(editEvent.getNewValue(), rooms.getNumber()))

{

selectedRoom.setNumber(editEvent.getNewValue());

roomService.updateRoom(selectedRoom);

}

else

{

logger.warn("Room with this number already exist");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Room existence", ButtonType.OK);

IOAlert.setContentText("Room with this number already exist / Click the refresh button and try again");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

break;

}

}

}

/\*\*

\* Обработчик события изменения вместительности помещения в таблице.

\* @param editEvent Событие изменения вместительности помещения в таблице.

\*/

@FXML

private void editRoomCapacity(TableColumn.CellEditEvent<Room, Integer> editEvent)

{

Room selectedRoom = table\_rooms.getSelectionModel().getSelectedItem();

selectedRoom.setCapacity(editEvent.getNewValue());

roomService.updateRoom(selectedRoom);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события нажатия на кнопку сохранения таблицы помещений в PDF файл.

\* Сохраняет данные из таблицы в файл "pdf/report\_rooms.pdf".

\* Если список помещений пуст, выбрасывает исключение MyPDFException.

\* Если возникает ошибка ввода-вывода, выводит сообщение об ошибке.

\*/

@FXML

private void toPDFRooms(ActionEvent event) throws IOException {

try {

Document my\_pdf\_report1 = new Document();

PdfWriter.getInstance(my\_pdf\_report1, new FileOutputStream("report\_rooms.pdf"));

my\_pdf\_report1.open();

PdfPTable my\_report\_table\_rooms = new PdfPTable(2);

PdfPCell table\_rooms\_cell;

my\_report\_table\_rooms.setHeaderRows(2);

my\_report\_table\_rooms.addCell(new PdfPCell(new Phrase("Number", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table\_rooms.addCell(new PdfPCell(new Phrase("Capacity", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

if (RoomList.isEmpty()) throw new MyPDFException();

for(Room rooms : RoomList)

{

table\_rooms\_cell=new PdfPCell(new Phrase(String.valueOf(rooms.getNumber())));

my\_report\_table\_rooms.addCell(table\_rooms\_cell);

table\_rooms\_cell=new PdfPCell(new Phrase(String.valueOf(rooms.getCapacity())));

my\_report\_table\_rooms.addCell(table\_rooms\_cell);

}

my\_pdf\_report1.add(my\_report\_table\_rooms);

my\_pdf\_report1.close();

}

catch (FileNotFoundException | DocumentException | MyPDFException e)

{

logger.warn("Exception " + e);

e.printStackTrace();

}

refreshScreen(event);

}

/\*\*

\* Устанавливает список стеллажей.

\* Добавляет список товаров из БД.

\*/

private void setShelfList() {

ShelfList.clear();

ShelfList.addAll(shelfService.getShelves());

}

/\*\*

\* Метод для получения отфильтрованного и отсортированного списка стеллажей.

\* Создает новый отфильтрованный список на основе исходного списка стеллажей, используя фильтр из searchField.

\* Затем создает новый отсортированный список на основе отфильтрованного списка и связывает его с компаратором таблицы.

\* @return Отсортированный список стеллажей.

\*/

private ObservableList<Shelf> getShelvesSortedList() {

SortedList<Shelf> sortedList = new SortedList<>(getShelvesFilteredList());

sortedList.comparatorProperty().bind(table\_shelves.comparatorProperty());

return sortedList;

}

/\*\*

\* Метод для получения отфильтрованного списка стеллажей на основе заданного текстового фильтра.

\* Создает новый отфильтрованный список на основе исходного списка стеллажей, используя заданный текстовый фильтр.

\* Фильтр применяется к полям "Номер", "Вместительность", "Помещение, в котором стоит стеллаж" каждого стеллажа.

\* @return Отфильтрованный список стеллажей.

\*/

private FilteredList<Shelf> getShelvesFilteredList() {

FilteredList<Shelf> filteredList = new FilteredList<>(ShelfList, b -> true);

searchShelves.textProperty().addListener((observable, oldValue, newValue) ->

filteredList.setPredicate(shelf -> {

if (newValue == null || newValue.isEmpty()) {

return true;

}

String lowerCaseFilter = newValue.toLowerCase();

if (String.valueOf(shelf.getNumber()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else if (String.valueOf(shelf.getCapacity()).toLowerCase().contains(lowerCaseFilter)) {

return true;

} else return String.valueOf(shelf.getRoom().getNumber()).toLowerCase().contains(lowerCaseFilter);

}));

return filteredList;

}

/\*\*

\* Обработчик события добавления нового стеллажа.

\* Вызывает метод NewWindowController для отображения окна добавления нового стеллажа.

\* Если стеллаж был успешно добавлен, обновляет экран с стеллажами.

\* @param event Событие добавления нового стеллажа.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void addShelves(ActionEvent event) throws IOException {

logger.debug("adding a shelf");

NewWindowController.getNewShelfWindow();

if(UpdateStatus.isIsShelfAdded()) {

refreshScreen(event);

UpdateStatus.setIsShelfAdded(false);

}

logger.info("room shelf");

}

/\*\*

\* Обработчик события удаления выбранных стеллажей из таблицы.

\* Удаляет выбранные стеллажи из таблицы.

\* Если ни один стеллаж не выбран, выбрасывает исключение myDeleteException.

\* После удаления стеллажей обновляет экран.

\* @param event Событие удаления стеллажей.

\*/

@FXML

private void deleteShelves(ActionEvent event)

{

try {

logger.debug("deleting a shelf");

deleteShelveRows(event);

logger.info("shelf deleted");

}

catch (myDeleteException | IOException myEx){

logger.error("MyDeleteException " + myEx);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, myEx.getMessage(), ButtonType.OK);

IOAlert.setContentText(myEx.getMessage());

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события удаления выбранных стеллажей из таблицы.

\* Удаляет выбранные стеллажи из таблицы.

\* Если ни один стеллаж не выбран, выбрасывает исключение myDeleteException.

\* После удаления стеллажей обновляет экран.

\* @param event Событие удаления стеллажей.

\* @throws RoomsShelvesController.myDeleteException Если ни один стеллаж не выбран.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

private void deleteShelveRows(ActionEvent event) throws myDeleteException, IOException {

int selectedID = table\_shelves.getSelectionModel().getSelectedIndex();

if (selectedID == -1) throw new myDeleteException();

else {

ObservableList<Shelf> selectedRows = table\_shelves.getSelectionModel().getSelectedItems();

for (Shelf shelf : selectedRows) {

shelfService.deleteShelf(shelf);

}

refreshScreen(event);

}

}

/\*\*

\* Обработчик события сохранения списка стеллажей в файл.

\* Сохраняет список стеллажей в файл "saves/save\_shelf.csv".

\* Если произошла ошибка ввода-вывода при сохранении, выбрасывает исключение IOException.

\* После сохранения открывает папку "saves".

\*/

@FXML

private void saveShelves()

{

try

{

logger.debug("saving to file");

BufferedWriter writer = new BufferedWriter(new FileWriter("saves/save\_shelf.csv"));

for(Shelf shelves : ShelfList)

{

writer.write(shelves.getNumber() + ";" + shelves.getCapacity() + ";"

+ shelves.getRoom().getNumber());

writer.newLine();

}

writer.close();

Desktop.getDesktop().open(new File("saves"));

logger.info("saved to file");

}

catch (IOException e)

{

logger.warn("Exception " + e);

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Error!", ButtonType.OK);

IOAlert.setContentText("Error");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

}

}

/\*\*

\* Обработчик события изменения номера стеллажа в таблице.

\* Проверка на существования стеллажа с введеным для изменения номером.

\* Вывод ошибки если стеллаж с таким номером уже существует.

\* @param editEvent Событие изменения названия стеллажа в таблице.

\*/

@FXML

private void editShelfNumber(TableColumn.CellEditEvent<Shelf, Integer> editEvent)

{

Shelf selectedShelf = table\_shelves.getSelectionModel().getSelectedItem();

for (Shelf shelves : shelfService.getShelves()){

if (!Objects.equals(editEvent.getNewValue(), shelves.getNumber()))

{

selectedShelf.setNumber(editEvent.getNewValue());

shelfService.updateShelf(selectedShelf);

}

else

{

logger.warn("Shelf with this number already exist");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Shelf existence", ButtonType.OK);

IOAlert.setContentText("Shelf with this number already exist / Click the refresh button and try again");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

break;

}

}

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения вместительности стеллажа в таблице.

\* Проверка на кол-во товара уже на стеллаже.

\* Вывод ошибки если стеллаж заполнен больше чем пользователь пытается ввести.

\* @param editEvent Событие изменения вместительности стеллажа в таблице.

\*/

@FXML

private void editShelfCapacity(TableColumn.CellEditEvent<Shelf, Integer> editEvent)

{

Shelf selectedShelf = table\_shelves.getSelectionModel().getSelectedItem();

Integer product\_space = 0;

List<Product> products = selectedShelf.getProductList();

for(Product product : products)

{

product\_space += product.getQuantity();

}

if (editEvent.getNewValue() >= product\_space)

{

selectedShelf.setCapacity(editEvent.getNewValue());

shelfService.updateShelf(selectedShelf);

}

else

{

logger.error("MyAddException");

Alert IOAlert = new Alert(Alert.AlertType.ERROR, "MyAddException", ButtonType.OK);

IOAlert.setContentText("You can't put this capacity products occupy much more space / Retry!");

IOAlert.showAndWait();

if(IOAlert.getResult() == ButtonType.OK)

{

IOAlert.close();

}

table\_shelves.refresh();

}

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события изменения помещения в котором находится стеллаж.

\* @param editEvent Событие изменения помещения в котором находится стеллаж.

\*/

@FXML

private void editRoom(TableColumn.CellEditEvent<Shelf, Room> editEvent)

{

Shelf selectedShelf = table\_shelves.getSelectionModel().getSelectedItem();

Room room = editEvent.getNewValue();

selectedShelf.setRoom(room);

shelfService.updateShelf(selectedShelf);

logger.debug("Editing cell");

}

/\*\*

\* Обработчик события нажатия на кнопку сохранения таблицы товаров в PDF файл.

\* Сохраняет данные из таблицы в файл "pdf/report\_shelves.pdf".

\* Если список товаров пуст, выбрасывает исключение MyPDFException.

\* Если возникает ошибка ввода-вывода, выводит сообщение об ошибке.

\*/

@FXML

private void toPDFShelves(ActionEvent event) throws IOException {

try {

// logger.debug("Saving to PDF");

Document my\_pdf\_report = new Document();

PdfWriter.getInstance(my\_pdf\_report, new FileOutputStream("report\_shelves.pdf"));

my\_pdf\_report.open();

PdfPTable my\_report\_table\_shelves = new PdfPTable(3);

PdfPCell table\_shelves\_cell;

my\_report\_table\_shelves.setHeaderRows(1);

my\_report\_table\_shelves.addCell(new PdfPCell(new Phrase("Number", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table\_shelves.addCell(new PdfPCell(new Phrase("Capacity", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

my\_report\_table\_shelves.addCell(new PdfPCell(new Phrase("Room", FontFactory.getFont(FontFactory.COURIER, 16, Font.BOLD))));

if (ShelfList.isEmpty()) throw new MyPDFException();

for(Shelf shelfs : ShelfList)

{

table\_shelves\_cell=new PdfPCell(new Phrase(String.valueOf(shelfs.getNumber())));

my\_report\_table\_shelves.addCell(table\_shelves\_cell);

table\_shelves\_cell=new PdfPCell(new Phrase(String.valueOf(shelfs.getCapacity())));

my\_report\_table\_shelves.addCell(table\_shelves\_cell);

table\_shelves\_cell=new PdfPCell(new Phrase(String.valueOf(shelfs.getRoom().getNumber())));

my\_report\_table\_shelves.addCell(table\_shelves\_cell);

}

my\_pdf\_report.add(my\_report\_table\_shelves);

my\_pdf\_report.close();

logger.info("Saved to PDF");

}

catch (FileNotFoundException | DocumentException | MyPDFException e)

{

logger.warn("Exception " + e);

e.printStackTrace();

}

refreshScreen(event);

}

/\*\*

\* Выход из окна программы.

\*/

public void ExitMainWindow() {

logger.debug("Closing main window");

exit\_btn.setOnAction(SceneController::close);

}

/\*\*

\* Сворачивание окна программы.

\*/

public void WrapMainWindow() {

logger.debug("Wrapping main window");

wrap\_btn.setOnAction(SceneController::wrap);

}

/\*\*

\* Обработчик события обновления экрана.

\* Вызывает метод SceneController для отображения сцены с помещениями и стеллажами.

\* @param event Событие обновления экрана.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

@FXML

private void refreshScreen(ActionEvent event) throws IOException {

SceneController.getRoomsShelvesScene(event);

}

/\*\*

\* Инициализация.

\*/

@Override

public void initialize(URL url, ResourceBundle rb)

{

warehouseService.getFullnessOfWarehouse();

fullness\_label.setText("Fullness: " + warehouseService.getFullnessOfWarehouse() + "%");

table\_shelves.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

choice\_box.setValue("Rooms/Shelves");

choice\_box.getItems().addAll(choices);

setShelfList();

setRoomList();

number\_column.setCellValueFactory(new PropertyValueFactory<>("number"));

capacity\_column.setCellValueFactory(new PropertyValueFactory<>("capacity"));

room\_column.setCellValueFactory(new PropertyValueFactory<>("room"));

roomnumber\_column.setCellValueFactory(new PropertyValueFactory<>("number"));

roomcapacity\_column.setCellValueFactory(new PropertyValueFactory<>("capacity"));

ObservableList<Room> roomsObservableList = FXCollections.observableArrayList(roomService.getRooms());

table\_shelves.setEditable(true);

number\_column.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));

capacity\_column.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));

room\_column.setCellFactory(ChoiceBoxTableCell.forTableColumn(roomsObservableList));

roomnumber\_column.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));

roomcapacity\_column.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));

table\_shelves.setItems(getShelvesSortedList());

table\_rooms.setItems(getRoomsSortedList());

}

}

SceneController

package com.umler.warehouses.Controllers;

import com.umler.warehouses.Enums.ScenePath;

import javafx.event.ActionEvent;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

import javafx.stage.StageStyle;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.IOException;

import java.util.Objects;

/\*\*

\* Контроллер для переключения сцен интерфейса.

\* @author Umler

\*/

public class SceneController {

private static double x;

private static double y;

private static Parent main;

private static final Logger logger = LoggerFactory.getLogger("Scene Logger");

/\*\*

\* Метод для перехода на сцену с авторизицией / Создание главной сцены.

\* @param stage Главная сцена.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

public static void getInitialScene(Stage stage) throws IOException {

logger.debug("InitialScene");

main = FXMLLoader.load((Objects.requireNonNull(SceneController.class.getResource(ScenePath.LOGIN.getPath()))));

Scene scene = new Scene(main);

controlDrag(stage);

logger.debug("Draggable scene made");

stage.initStyle(StageStyle.UNDECORATED);

stage.setTitle("WAREHOUSES");

stage.setScene(scene);

stage.show();

}

/\*\*

\* Метод для перехода на сцену со списком компаний.

\* @param event Событие, вызвавшее переход.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

public static void getCompaniesScene(ActionEvent event) throws IOException {

logger.debug("Transition to Companies scene");

changeScreen(event, ScenePath.COMPANIES.getPath());

}

/\*\*

\* Метод для перехода на сцену со списком товаров.

\* @param event Событие, вызвавшее переход.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

public static void getProductsScene(ActionEvent event) throws IOException {

logger.debug("Transition to Products scene");

changeScreen(event, ScenePath.PRODUCTS.getPath());

}

/\*\*

\* Метод для перехода на сцену со списком помещений и стеллажей.

\* @param event Событие, вызвавшее переход.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

public static void getRoomsShelvesScene(ActionEvent event) throws IOException {

logger.debug("Transition to RoomsShelves scene");

changeScreen(event, ScenePath.ROOMSSHELVES.getPath());

}

/\*\*

\* Метод для перехода на сцену со списком контрактов.

\* @param event Событие, вызвавшее переход.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

public static void getContractsScene(ActionEvent event) throws IOException {

logger.debug("Transition to Contracts scene");

changeScreen(event, ScenePath.CONTRACTS.getPath());

}

/\*\*

\* Метод для перехода на сцену с авторизацией.

\* @param event Событие, вызвавшее переход.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

public static void getLoginScene(ActionEvent event) throws IOException {

logger.debug("Transition to Login scene");

changeScreen(event, ScenePath.LOGIN.getPath());

}

/\*\*

\* Метод для смены сцены на заданную.

\* @param event Событие, вызвавшее смену сцены.

\* @param path Путь к файлу FXML новой сцены из класса Enum ScenePath.

\* @throws IOException Если произошла ошибка ввода-вывода при загрузке сцены.

\*/

private static void changeScreen(ActionEvent event, String path) throws IOException

{

logger.debug("Changing scene");

FXMLLoader loader = new FXMLLoader();

loader.setLocation(Objects.requireNonNull(SceneController.class.getResource(path)));

main = loader.load();

Scene visitScene = new Scene(main);

Stage window = (Stage) ((Node) event.getSource()).getScene().getWindow();

controlDrag(window);

window.setScene(visitScene);

window.show();

}

/\*\*

\* Окно можно перетаскивать по экрану при зажатии ЛКМ по окну.

\*/

public static void controlDrag(Stage stage) {

main.setOnMousePressed(event -> {

x = stage.getX() - event.getScreenX();

y = stage.getY() - event.getScreenY();

});

main.setOnMouseDragged(event -> {

stage.setX(event.getScreenX() + x);

stage.setY(event.getScreenY() + y);

});

}

/\*\*

\* Метод для закрытия окон по нажатию кнопки.

\*/

public static void close(ActionEvent actionEvent) {

logger.debug("Closing window");

Node source = (Node) actionEvent.getSource();

Stage stage = (Stage) source.getScene().getWindow();

stage.close();

}

/\*\*

\* Метод для сворачивания окон по нажатию кнопки.

\*/

public static void wrap(ActionEvent actionEvent) {

logger.debug("Wrapping window");

Node source = (Node) actionEvent.getSource();

Stage stage = (Stage) source.getScene().getWindow();

stage.setIconified(true);

}

}

CustomIntegerStringConverter

package com.umler.warehouses.Converters;  
  
import javafx.scene.control.Alert;  
import javafx.scene.control.ButtonType;  
import javafx.util.converter.IntegerStringConverter;  
  
  
/\*\*  
 \* Конвертер Integer to/from String.  
 \* @author Umler  
 \*/  
public class CustomIntegerStringConverter extends IntegerStringConverter {  
 private final IntegerStringConverter converter = new IntegerStringConverter();  
  
 /\*\*  
 \* Конвертация Integer to String.  
 \*/  
 @Override  
 public String toString(Integer object) {  
 try {  
 return converter.toString(object);  
 } catch (NumberFormatException e) {  
 showAlert();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* Конвертация Integer from String.  
 \*/  
 @Override  
 public Integer fromString(String string) {  
 try {  
 return converter.fromString(string);  
 } catch (NumberFormatException e) {  
 showAlert();  
 }  
 return -1;  
 }  
  
 /\*\*  
 \* Вывод ошибок.  
 \*/  
 private void showAlert(){  
 Alert IOAlert = new Alert(Alert.AlertType.ERROR, "Input Error", ButtonType.OK);  
 IOAlert.setContentText("Incorrect input, you need to input a number");  
 IOAlert.showAndWait();  
 if(IOAlert.getResult() == ButtonType.OK)  
 {  
 IOAlert.close();  
 }  
 }  
}  
ScenePath  
  
package com.umler.warehouses.Enums;

/\*\*

\* Enum для хранения путей к частям интерфейса.

\* @author Umler

\*/

public enum ScenePath {

COMPANIES("/Companies.fxml"),

CONTRACTS("/Contracts.fxml"),

PRODUCTS("/Products.fxml"),

ROOMSSHELVES("/RoomsShelves.fxml"),

ADDROOM("/addRoom.fxml"),

ADDSHELF("/addShelf.fxml"),

ADDCONTRACTCOMPANY("/addContractCompany.fxml"),

ADDPRODUCT("/addProduct.fxml"),

LOGININFO("/loginInfo.fxml"),

LOGIN("/login.fxml");

private final String path;

ScenePath(String path) {

this.path = path;

}

public String getPath() {

return path;

}

}

ComboBoxUtil

package com.umler.warehouses.Helpers;

import com.umler.warehouses.Model.Product;

import javafx.scene.control.ComboBox;

import javafx.scene.control.ListCell;

import javafx.util.StringConverter;

/\*\*

\* Класс для переопределения вида ComboBox,

\* для вывода разных параметров одного класса в разные ComboBox.

\* @author Umler

\*/

public class ComboBoxUtil {

public static void configureNameComboBox(ComboBox<Product> nameComboBox) {

nameComboBox.setCellFactory(param -> new ListCell<>() {

@Override

protected void updateItem(Product item, boolean empty) {

super.updateItem(item, empty);

if (empty || item == null) {

setText("");

} else {

setText(item.getName());

}

}

});

nameComboBox.setConverter(new StringConverter<>() {

@Override

public String toString(Product product) {

return product == null ? "" : product.getName();

}

@Override

public Product fromString(String string) {

return null;

}

});

}

public static void configureTypeComboBox(ComboBox<Product> typeComboBox) {

typeComboBox.setCellFactory(param -> new ListCell<>() {

@Override

protected void updateItem(Product item, boolean empty) {

super.updateItem(item, empty);

if (empty || item == null) {

setText("");

} else {

setText(item.getType());

}

}

});

typeComboBox.setConverter(new StringConverter<>() {

@Override

public String toString(Product product) {

return product == null ? "" : product.getType();

}

@Override

public Product fromString(String string) {

return null;

}

});

}

}

CurrentUser

package com.umler.warehouses.Helpers;

import com.umler.warehouses.Model.User;

/\*\*

\* Класс для сохранения данных о текущем авторизированном пользователе.

\* @author Umler

\*/

public class CurrentUser {

private static User user;

private CurrentUser() {

}

public static User getCurrentUser() {

return user;

}

public static void setCurrentUser(User currentUser) {

user = currentUser;

}

}

DatePickerCellFactory

package com.umler.warehouses.Helpers;

import javafx.scene.control.DateCell;

import javafx.scene.control.DatePicker;

import javafx.util.Callback;

import java.time.LocalDate;

/\*\*

\* Класс для переопределения вида ячеек таблицы под DatePicker календарь.

\* @author Umler

\*/

public class DatePickerCellFactory implements Callback<DatePicker, DateCell> {

private final LocalDate selectedDate;

public DatePickerCellFactory(DatePicker datePicker) {

if (datePicker.getValue()==null)

{

selectedDate = LocalDate.now();

}

else {

selectedDate = datePicker.getValue();

}

datePicker.setDayCellFactory(this);

}

@Override

public DateCell call(DatePicker param) {

return new DateCell() {

@Override

public void updateItem(LocalDate item, boolean empty) {

super.updateItem(item, empty);

if (item.isBefore(selectedDate)) {

setDisable(true);

this.setStyle("-fx-background-color: #ffc5cf;");

}

}

};

}

public LocalDate getValue() {

return selectedDate;

}

}

DistinctObservableList

package com.umler.warehouses.Helpers;

import javafx.collections.ObservableList;

import javafx.collections.transformation.FilteredList;

import java.util.HashSet;

import java.util.Set;

import java.util.function.Function;

import java.util.function.Predicate;

/\*\*

\* Класс для отбрасывания повторов с списке с типом объекта.

\* Используется для корректного отображения ComboBox, для вывода разных параметров одного класса в разные ComboBox.

\* @author Umler

\*/

public class DistinctObservableList<T> {

private final FilteredList<T> filteredList;

public DistinctObservableList(ObservableList<T> list) {

filteredList = new FilteredList<>(list);

}

public void distinct(Function<? super T, ?> keyExtractor) {

filteredList.setPredicate(distinctByKey(keyExtractor));

}

private <U> Predicate<T> distinctByKey(Function<? super T, ? extends U> keyExtractor) {

Set<U> seen = new HashSet<>();

return t -> seen.add(keyExtractor.apply(t));

}

public ObservableList<T> getFilteredList() {

return filteredList;

}

}

HibernateUtil

package com.umler.warehouses.Helpers;

import com.umler.warehouses.Model.\*;

import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.hibernate.cfg.Environment;

import org.hibernate.service.ServiceRegistry;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.Properties;

/\*\*

\* Класс HibernateUtil предоставляет методы для работы с Hibernate.

\* Он содержит методы для создания объекта SessionFactory и закрытия его.

\* @author Umler

\*/

public class HibernateUtil {

private static SessionFactory sessionFactory;

private static final Logger logger = LoggerFactory.getLogger("Hibernate Logger");

/\*\*

\* Возвращает объект SessionFactory, который используется для создания сессий Hibernate.

\* Если объект SessionFactory еще не создан, то создает его с помощью настроек, указанных в методе.

\* @return объект SessionFactory

\*/

public static SessionFactory getSessionFactory() {

if (sessionFactory == null) {

try {

logger.debug("SessionFactory opened");

Configuration configuration = new Configuration();

Properties settings = new Properties();

settings.put(Environment.URL, "jdbc:mysql://localhost:3306/Warehouses");

settings.put(Environment.USER, "Umler");

settings.put(Environment.PASS, "Umler1337228");

settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQLDialect");

settings.put(Environment.SHOW\_SQL, "false");

settings.put(Environment.CURRENT\_SESSION\_CONTEXT\_CLASS, "thread");

settings.put(Environment.HBM2DDL\_AUTO, "update");

logger.debug("Settings put");

configuration.setProperties(settings);

configuration.addAnnotatedClass(User.class);

configuration.addAnnotatedClass(Contract.class);

configuration.addAnnotatedClass(Company.class);

configuration.addAnnotatedClass(Room.class);

configuration.addAnnotatedClass(Shelf.class);

configuration.addAnnotatedClass(Product.class);

logger.debug("Configuration added");

logger.debug("ServiceRegistry");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()

.applySettings(configuration.getProperties()).build();

sessionFactory = configuration.buildSessionFactory(serviceRegistry);

} catch (Exception e) {

e.printStackTrace();

}

}

return sessionFactory;

}

/\*\*

\* Закрывает объект SessionFactory и освобождает все связанные с ним ресурсы.

\*/

public static void shutdown() {

getSessionFactory().close();

}

}

LocalEndDateCellFactory

package com.umler.warehouses.Helpers;  
  
import javafx.scene.control.TableCell;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.DatePicker;  
import javafx.scene.control.DateCell;  
import javafx.util.Callback;  
import com.umler.warehouses.Model.Contract;  
import com.umler.warehouses.Services.ContractService;  
  
import java.time.LocalDate;  
import java.sql.Date;  
  
/\*\*  
 \* Класс для переопределения вида ячеек даты окончания договора под тип LocalDate.  
 \* @author Umler  
 \*/  
public class LocalEndDateCellFactory implements Callback<TableColumn<Contract, LocalDate>, TableCell<Contract, LocalDate>> {  
 ContractService contractService = new ContractService();  
 @Override  
 public TableCell<Contract, LocalDate> call(TableColumn<Contract, LocalDate> col) {  
 return new TableCell<>() {  
 private final DatePicker datePicker = new DatePicker();  
 private LocalDate minDate = LocalDate.now();  
  
 {  
 datePicker.editableProperty().set(false);  
 datePicker.setOnAction((e) -> commitEdit(datePicker.getValue()));  
 this.setGraphic(datePicker);  
  
 // задаем диапазон дат для выбора значения EndDate  
 Callback<DatePicker, DateCell> dayCellFactory = new Callback<>() {  
 @Override  
 public DateCell call(final DatePicker datePicker) {  
 return new DateCell() {  
 @Override  
 public void updateItem(LocalDate item, boolean empty) {  
 super.updateItem(item, empty);  
  
 // если дата раньше minDate, то недоступна для выбора  
 if (item != null && item.isBefore(minDate)) {  
 this.setDisable(true);  
 this.setStyle("-fx-background-color: #ffc5cf;"); // также помечаем цветом  
 }  
 }  
 };  
 }  
 };  
 datePicker.setDayCellFactory(dayCellFactory);  
 }  
  
 @Override  
 public void updateItem(LocalDate item, boolean empty) {  
 super.updateItem(item, empty);  
 Contract contract = getTableRow().getItem();  
 if (item == null || empty || contract == null) {  
 setText(null);  
 setGraphic(null);  
 } else {  
 datePicker.setValue(item);  
 setGraphic(datePicker);  
 minDate = contract.getStartdate().plusDays(1); // Устанавливаем minDate для этой ячейки  
 }  
 }  
  
 @Override  
 public void commitEdit(LocalDate newValue) {  
 super.commitEdit(newValue);  
 Contract contract = getTableView().getColumns().get(1).getTableView().getItems().get(getIndex());  
 contract.setEnddate(Date.valueOf(newValue).toLocalDate());  
 contractService.updateContract(contract);  
 }  
  
 @Override  
 public void startEdit() {  
 super.startEdit();  
 LocalDate value = getItem();  
 if (value != null) {  
 datePicker.setValue(value);  
 }  
 // задаем минимальную дату выбора EndDate  
 Contract contract = getTableView().getColumns().get(1).getTableView().getItems().get(getIndex());  
 LocalDate startDate = contract.getStartdate();  
 minDate = startDate.plusDays(1);  
 Callback<DatePicker, DateCell> dayCellFactory = datePicker.getDayCellFactory();  
 datePicker.setDayCellFactory(param -> {  
 DateCell cell = dayCellFactory.call(param);  
 if (cell.getItem().isBefore(minDate)) {  
 cell.setDisable(true);  
 }  
 return cell;  
 });  
 }  
  
 @Override  
 public void cancelEdit() {  
 super.cancelEdit();  
 }  
 };  
 }  
}

LocalStartDateCellFactory

package com.umler.warehouses.Helpers;  
  
import com.umler.warehouses.Model.Contract;  
import com.umler.warehouses.Services.ContractService;  
import javafx.scene.control.\*;  
import javafx.util.Callback;  
  
import java.sql.Date;  
import java.time.LocalDate;  
  
/\*\*  
 \* Класс для переопределения вида ячеек даты подписания договора под тип LocalDate.  
 \* @author Umler  
 \*/  
  
public class LocalStartDateCellFactory implements Callback<TableColumn<Contract, LocalDate>, TableCell<Contract, LocalDate>> {  
 ContractService contractService = new ContractService();  
 @Override  
 public TableCell<Contract, LocalDate> call(TableColumn<Contract, LocalDate> col) {  
 return new TableCell<>() {  
 private final DatePicker datePicker = new DatePicker();  
 private LocalDate maxDate = LocalDate.of(9999,12,1);  
  
 {  
 datePicker.editableProperty().set(false);  
 datePicker.setOnAction((e) -> commitEdit(datePicker.getValue()));  
 this.setGraphic(datePicker);  
  
 // задаем диапазон дат для выбора значения EndDate  
 Callback<DatePicker, DateCell> dayCellFactory = new Callback<>() {  
 @Override  
 public DateCell call(final DatePicker datePicker) {  
 return new DateCell() {  
 @Override  
 public void updateItem(LocalDate item, boolean empty) {  
 super.updateItem(item, empty);  
  
 // если дата раньше minDate, то недоступна для выбора  
 if (item != null && item.isAfter(maxDate)) {  
 this.setDisable(true);  
 this.setStyle("-fx-background-color: #ffc5cf;"); // также помечаем цветом  
 }  
 }  
 };  
 }  
 };  
 datePicker.setDayCellFactory(dayCellFactory);  
 }  
  
 @Override  
 public void updateItem(LocalDate item, boolean empty) {  
 super.updateItem(item, empty);  
 Contract contract = getTableRow().getItem();  
 if (item == null || empty || contract == null) {  
 setText(null);  
 setGraphic(null);  
 } else {  
 datePicker.setValue(item);  
 setGraphic(datePicker);  
 maxDate = contract.getEnddate().minusDays(1); // Устанавливаем minDate для этой ячейки  
 }  
 }  
  
 @Override  
 public void commitEdit(LocalDate newValue) {  
 super.commitEdit(newValue);  
 Contract contract = getTableView().getColumns().get(0).getTableView().getItems().get(getIndex());  
 contract.setStartdate(Date.valueOf(newValue).toLocalDate());  
 contractService.updateContract(contract);  
 }  
  
 @Override  
 public void startEdit() {  
 super.startEdit();  
 LocalDate value = getItem();  
 if (value != null) {  
 datePicker.setValue(value);  
 }  
 // задаем максимальную дату выбора EndDate  
 Contract contract = getTableView().getColumns().get(0).getTableView().getItems().get(getIndex());  
 LocalDate endDate = contract.getStartdate();  
 maxDate = endDate.minusDays(1);  
 Callback<DatePicker, DateCell> dayCellFactory = datePicker.getDayCellFactory();  
 datePicker.setDayCellFactory(param -> {  
 DateCell cell = dayCellFactory.call(param);  
 if (cell.getItem().isAfter(maxDate)) {  
 cell.setDisable(true);  
 }  
 return cell;  
 });  
 }  
  
 @Override  
 public void cancelEdit() {  
 super.cancelEdit();  
 }  
 };  
 }  
}

UpdateStatus

package com.umler.warehouses.Helpers;  
  
  
/\*\*  
 \* Класс для проверки на добавление элемента в базу данных  
 \* @author Umler  
 \*/  
  
public final class UpdateStatus {  
  
 private UpdateStatus() {  
 }  
 private static boolean isProductAdded;  
 private static boolean isContractCompanyAdded;  
 private static boolean isShelfAdded;  
 private static boolean isRoomAdded;  
  
 public static boolean isIsContractCompanyAdded(){  
 return isContractCompanyAdded;  
 }  
  
 public static void setIsContractCompanyAdded(boolean isContractCompanyAdded) {  
 UpdateStatus.isContractCompanyAdded = isContractCompanyAdded;  
 }  
  
 public static boolean isIsProductAdded(){  
 return isProductAdded;  
 }  
  
 public static void setIsProductAdded(boolean isProductAdded) {  
 UpdateStatus.isProductAdded = isProductAdded;  
 }  
  
 public static boolean isIsShelfAdded(){  
 return isShelfAdded;  
 }  
  
 public static void setIsShelfAdded(boolean isShelfAdded) {  
 UpdateStatus.isShelfAdded = isShelfAdded;  
 }  
  
 public static boolean isIsRoomAdded(){  
 return isRoomAdded;  
 }  
  
 public static void setIsRoomAdded(boolean isRoomAdded) {  
 UpdateStatus.isRoomAdded = isRoomAdded;  
 }  
}

Company

package com.umler.warehouses.Model;

import org.hibernate.annotations.Cascade;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

/\*\*

\* Класс компании

\* @author Umler

\*/

@Entity

@Table(name="Company")

public class Company {

@Id

@Column(name = "id\_company")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id\_company;

@Column(name = "name")

private String name;

@Column(name = "address")

private String address;

@Column(name = "phone")

private String phoneNumber;

@Column(name = "tin")

private String tin; // ИНН

@OneToMany(mappedBy = "company", cascade=CascadeType.ALL, orphanRemoval=true, fetch = FetchType.EAGER)

@Cascade(org.hibernate.annotations.CascadeType.DELETE)

private List<Contract> contractList = new ArrayList<>();

/\*\*

\* Добавляет контракт в список контрактов, привязывает контракт к компании.

\*/

public void addContract(Contract contract) {

contractList.add(contract);

contract.setCompany(this);

}

public Integer getId\_company() {

return id\_company;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public String getTin() {

return tin;

}

public void setTin(String tin) {

this.tin = tin;

}

public List<Contract> getContractList() {

return contractList;

}

/\*\*

\* Переопределение метода toString для корректного вывода объектов класса.

\*/

@Override

public String toString() {

return String.format("%s", this.name);

}

/\*\*

\* Переопределение метода equals для корректного сравнения объектов класса.

\*/

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Company company = (Company) o;

return

Objects.equals(name, company.name) &&

Objects.equals(address, company.address) &&

Objects.equals(phoneNumber, company.phoneNumber)&&

Objects.equals(tin, company.tin);

}

/\*\*

\* Переопределение метода hashCode для корректного сравнения объектов класса.

\*/

@Override

public int hashCode() {

return Objects.hash(name, address, phoneNumber,tin);

}

}

Contract

package com.umler.warehouses.Model;

import org.hibernate.annotations.Cascade;

import javax.persistence.\*;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Класс контракта

\* @author Umler

\*/

@Entity

@Table(name="Contract")

public class Contract {

@Id

@Column(name = "id\_contract")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id\_contract;

@Column(name = "startdate")

private LocalDate startdate;

@Column(name = "enddate")

private LocalDate enddate;

@Column(name = "number")

private Integer number;

@ManyToOne

@JoinColumn(name = "company\_id")

private Company company;

@OneToMany(mappedBy = "contract", cascade=CascadeType.ALL, orphanRemoval=true, fetch = FetchType.EAGER)

@Cascade(org.hibernate.annotations.CascadeType.DELETE)

private List<Product> productList = new ArrayList<>();

public List<Product> getProductList() {

return productList;

}

public Integer getId\_contract() {

return id\_contract;

}

public LocalDate getStartdate() {

return startdate;

}

public void setStartdate(LocalDate startdate) {

this.startdate = startdate;

}

public LocalDate getEnddate() {

return enddate;

}

public void setEnddate(LocalDate enddate) {

this.enddate = enddate;

}

public Integer getNumber() {

return number;

}

public void setNumber(Integer number) {

this.number = number;

}

public Company getCompany() {

return company;

}

public void setCompany(Company company) {

this.company = company;

}

/\*\*

\* Переопределение метода toString для корректного вывода объектов класса.

\*/

@Override

public String toString() {

return String.format("%s | %s", this.number, this.company.getName());

}

}

Product

package com.umler.warehouses.Model;

import javax.persistence.\*;

/\*\*

\* Класс товара

\* @author Umler

\*/

@Entity

@Table(name="Product")

public class Product {

@Id

@Column(name = "id\_product")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id\_product;

@Column(name = "name")

private String name;

@Column(name = "type")

private String type;

@Column(name = "quantity")

private Integer quantity;

@ManyToOne

@JoinColumn(name = "shelf\_id")

private Shelf shelf;

@ManyToOne

@JoinColumn(name = "contract\_id")

private Contract contract;

public Integer getQuantity() {

return quantity;

}

public void setQuantity(Integer quantity) {

this.quantity = quantity;

}

public Contract getContract() {

return contract;

}

public void setContract(Contract contract) {

this.contract = contract;

}

public Integer getId\_product() {

return id\_product;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public Shelf getShelf() {

return shelf;

}

public void setShelf(Shelf shelf) {

this.shelf = shelf;

}

}

Room

package com.umler.warehouses.Model;

import org.hibernate.annotations.Cascade;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Класс помещения

\* @author Umler

\*/

@Entity

@Table(name="Room")

public class Room {

@Id

@Column(name = "id\_room")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id\_room;

@Column(name = "number")

private Integer number;

@Column(name = "capacity")

private Integer capacity;

@OneToMany(mappedBy = "room", cascade=CascadeType.ALL, orphanRemoval=true, fetch = FetchType.EAGER)

@Cascade(org.hibernate.annotations.CascadeType.DELETE)

private List<Shelf> shelvesList = new ArrayList<>();

public List<Shelf> getShelvesList() {

return shelvesList;

}

public Integer getId\_room() {

return id\_room;

}

public Integer getNumber() {

return number;

}

public void setNumber(Integer number) {

this.number = number;

}

public Integer getCapacity() {

return capacity;

}

public void setCapacity(Integer capacity) {

this.capacity = capacity;

}

/\*\*

\* Переопределение метода toString для корректного вывода объектов класса.

\*/

@Override

public String toString() {

return String.format("%s", this.number);

}

}

Shelf

package com.umler.warehouses.Model;

import org.hibernate.annotations.Cascade;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Класс стеллажа

\* @author Umler

\*/

@Entity

@Table(name="Shelf")

public class Shelf {

@Id

@Column(name = "id\_shelf")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id\_shelf;

@Column(name = "number")

private Integer number;

@Column(name = "capacity")

private Integer capacity;

@ManyToOne

@JoinColumn(name = "room\_id")

private Room room;

@OneToMany(mappedBy = "shelf", cascade=CascadeType.ALL, orphanRemoval=true, fetch = FetchType.EAGER)

@Cascade(org.hibernate.annotations.CascadeType.DELETE)

private List<Product> productList = new ArrayList<>();

public List<Product> getProductList() {

return productList;

}

public Room getRoom() {

return room;

}

public void setRoom(Room room) {

this.room = room;

}

public Integer getId\_shelf() {

return id\_shelf;

}

public Integer getNumber() {

return number;

}

public void setNumber(Integer number) {

this.number = number;

}

public Integer getCapacity() {

return capacity;

}

public void setCapacity(Integer capacity) {

this.capacity = capacity;

}

/\*\*

\* Переопределение метода toString для корректного вывода объектов класса.

\*/

@Override

public String toString() {

return String.format("%s", this.number);

}

}

User

package com.umler.warehouses.Model;

import javax.persistence.\*;

/\*\*

\* Класс пользователя

\* @author Umler

\*/

@Entity

@Table(name="User")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_user")

private Integer id;

@Column(name = "name")

private String name;

@Column(name = "password")

private String password;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

CompanyService

package com.umler.warehouses.Services;  
  
  
import com.umler.warehouses.Helpers.HibernateUtil;  
import com.umler.warehouses.Model.Company;  
import org.hibernate.Hibernate;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
import org.hibernate.resource.transaction.spi.TransactionStatus;  
  
import java.util.ArrayList;  
import java.util.List;  
  
  
/\*\*  
 \* Класс для работы с таблицей Company в базе данных  
 \* @author Umler  
 \*/  
  
public class CompanyService {  
  
 /\*\*  
 \* Создает новой компании или обновляет существующую в базе данных.  
 \* @param company объект, представляющий компанию  
 \* @return true, если операция выполнена успешно, false в противном случае  
 \*/  
 public Boolean createCompany(Company company) {  
 Transaction transaction = null;  
 try (Session session = HibernateUtil.getSessionFactory().openSession()) {  
 transaction = session.beginTransaction();  
 session.saveOrUpdate(company);  
 transaction.commit();  
 return transaction.getStatus() == TransactionStatus.COMMITTED;  
 } catch (Exception ex) {  
 if (transaction != null) {  
 transaction.rollback();  
 }  
 ex.printStackTrace();  
 }  
 return false;  
 }  
  
 /\*\*  
 \* Обновляет информацию о компании в базе данных.  
 \* @param company объект, представляющий компанию с обновленными данными  
 \*/  
 public void updateCompany(Company company) {  
 Transaction transaction = null;  
 try (Session session = HibernateUtil.getSessionFactory().openSession()) {  
 transaction = session.beginTransaction();  
 session.update(company);  
 transaction.commit();  
 } catch (Exception ex) {  
 if (transaction != null) {  
 transaction.rollback();  
 }  
 ex.printStackTrace();  
 }  
 }  
  
 /\*\*  
 \* Удаляет информацию о компании из базы данных.  
 \* @param company объект, представляющий компанию, которую нужно удалить  
 \*/  
 public void deleteCompany(Company company) {  
 Transaction transaction = null;  
 try (Session session = HibernateUtil.getSessionFactory().openSession()) {  
 transaction = session.beginTransaction();  
 session.delete(company);  
 transaction.commit();  
 } catch (Exception ex) {  
 if (transaction != null) {  
 transaction.rollback();  
 }  
 ex.printStackTrace();  
 }  
 }  
  
 /\*\*  
 \* Возвращает список всех компаний из базы данных.  
 \* @return список объектов, представляющих компаний  
 \*/  
 public List<Company> getCompanies() {  
 try (Session session = HibernateUtil.getSessionFactory().openSession()) {  
 List<Company> companies = session.createQuery("from Company", Company.class).list();  
 for(Company company : companies){  
 Hibernate.initialize(company.getContractList());  
 }  
 return companies;  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 return new ArrayList<>();  
 }  
 }  
  
}

ContractService

package com.umler.warehouses.Services;

import com.umler.warehouses.Helpers.HibernateUtil;

import com.umler.warehouses.Model.Contract;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.resource.transaction.spi.TransactionStatus;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Класс для работы с таблицей Contract в базе данных

\* @author Umler

\*/

public class ContractService {

/\*\*

\* Создает новый контракт или обновляет существующую в базе данных.

\* @param contract объект, представляющий контракт

\* @return true, если операция выполнена успешно, false в противном случае

\*/

public Boolean createContract(Contract contract) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.save(contract);

transaction.commit();

return transaction.getStatus() == TransactionStatus.COMMITTED;

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

return false;

}

/\*\*

\* Обновляет информацию о контракте в базе данных.

\* @param contract объект, представляющий контракт с обновленными данными

\*/

public void updateContract(Contract contract) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.update(contract);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Удаляет информацию о контракте из базы данных.

\* @param contract объект, представляющий контракт, который нужно удалить

\*/

public void deleteContract(Contract contract) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.delete(contract);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Возвращает список всех контрактов из базы данных.

\* @return список объектов, представляющих контракты

\*/

public List<Contract> getContracts() {

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

return session.createQuery("from Contract ", Contract.class).list();

} catch (Exception ex) {

ex.printStackTrace();

return new ArrayList<>();

}

}

}

ProductService

package com.umler.warehouses.Services;

import com.umler.warehouses.Helpers.HibernateUtil;

import com.umler.warehouses.Model.Product;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.resource.transaction.spi.TransactionStatus;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Класс для работы с таблицей Product в базе данных

\* @author Umler

\*/

public class ProductService {

/\*\*

\* Создает новый товар или обновляет существующую в базе данных.

\* @param product объект, представляющий товар

\* @return true, если операция выполнена успешно, false в противном случае

\*/

public Boolean createProduct(Product product) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.saveOrUpdate(product);

transaction.commit();

return transaction.getStatus() == TransactionStatus.COMMITTED;

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

return false;

}

/\*\*

\* Обновляет информацию о товаре в базе данных.

\* @param product объект, представляющий товар с обновленными данными

\*/

public void updateProduct(Product product) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.update(product);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Удаляет информацию о товаре из базы данных.

\* @param product объект, представляющий товаре, который нужно удалить

\*/

public void deleteProduct(Product product) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.delete(product);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Возвращает список всех товаров из базы данных.

\* @return список объектов, представляющих товары

\*/

public List<Product> getProducts() {

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

return session.createQuery("from Product ", Product.class).list();

} catch (Exception ex) {

ex.printStackTrace();

return new ArrayList<>();

}

}

}

RoomService

package com.umler.warehouses.Services;

import com.umler.warehouses.Helpers.HibernateUtil;

import com.umler.warehouses.Model.Room;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.resource.transaction.spi.TransactionStatus;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

/\*\*

\* Класс для работы с таблицей Room в базе данных

\* @author Umler

\*/

public class RoomService {

/\*\*

\* Создает новое помещение или обновляет существующую в базе данных.

\* @param room объект, представляющий помещение

\* @return true, если операция выполнена успешно, false в противном случае

\*/

public Boolean createRoom(Room room) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.saveOrUpdate(room);

transaction.commit();

return transaction.getStatus() == TransactionStatus.COMMITTED;

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

return false;

}

/\*\*

\* Обновляет информацию о помещении в базе данных.

\* @param room объект, представляющий помещение с обновленными данными

\*/

public void updateRoom(Room room) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.update(room);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Удаляет информацию о помещении из базы данных.

\* @param room объект, представляющий помещении, который нужно удалить

\*/

public void deleteRoom(Room room) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.delete(room);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Возвращает помещение по номеру.

\* @param number номер помещения для поиска в БД.

\* @return помещение по номеру

\*/

public Room getRoom(Integer number) {

List<Room> t = getRooms();

for (Room room : t){

if (Objects.equals(room.getNumber(), number))

return room;

}

return null;

}

/\*\*

\* Возвращает список всех помещений из базы данных.

\* @return список объектов, представляющих помещения

\*/

public List<Room> getRooms() {

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

return session.createQuery("from Room ", Room.class).list();

} catch (Exception ex) {

ex.printStackTrace();

return new ArrayList<>();

}

}

}

ShelfService

package com.umler.warehouses.Services;

import com.umler.warehouses.Helpers.HibernateUtil;

import com.umler.warehouses.Model.Shelf;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.resource.transaction.spi.TransactionStatus;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

/\*\*

\* Класс для работы с таблицей Shelf в базе данных

\* @author Umler

\*/

public class ShelfService {

/\*\*

\* Создает новый стеллаж или обновляет существующую в базе данных.

\* @param shelf объект, представляющий стеллаж

\* @return true, если операция выполнена успешно, false в противном случае

\*/

public Boolean createShelf(Shelf shelf) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.saveOrUpdate(shelf);

transaction.commit();

return transaction.getStatus() == TransactionStatus.COMMITTED;

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

return false;

}

/\*\*

\* Обновляет информацию о стеллаже в базе данных.

\* @param shelf объект, представляющий стеллаж с обновленными данными

\*/

public void updateShelf(Shelf shelf) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.update(shelf);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Удаляет информацию о стеллаже из базы данных.

\* @param shelf объект, представляющий стеллаж, который нужно удалить

\*/

public void deleteShelf(Shelf shelf) {

Transaction transaction = null;

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

transaction = session.beginTransaction();

session.delete(shelf);

transaction.commit();

} catch (Exception ex) {

if (transaction != null) {

transaction.rollback();

}

ex.printStackTrace();

}

}

/\*\*

\* Возвращает стеллаж по номеру.

\* @param number номер стеллажа для поиска в БД.

\* @return стеллаж по номеру

\*/

public Shelf getShelf(Integer number) {

List<Shelf> shelves = getShelves();

for (Shelf shelf : shelves){

if (Objects.equals(shelf.getNumber(), number))

return shelf;

}

return null;

}

/\*\*

\* Возвращает список всех стеллажей из базы данных.

\* @return список объектов, представляющих стеллажи

\*/

public List<Shelf> getShelves() {

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

return session.createQuery("from Shelf ", Shelf.class).list();

} catch (Exception ex) {

ex.printStackTrace();

return new ArrayList<>();

}

}

}

UserService

package com.umler.warehouses.Services;

import com.umler.warehouses.Model.User;

import com.umler.warehouses.Helpers.HibernateUtil;

import org.hibernate.Session;

import javax.persistence.NoResultException;

import javax.persistence.TypedQuery;

/\*\*

\* Класс для работы с таблицей User в базе данных

\* @author Umler

\*/

public class UserService {

/\*\*

\* Возвращает данные пользователя по введным логину и паролю.

\* @return данные о пользователе

\*/

public User getConnectedUser(String name, String password) {

try (Session session = HibernateUtil.getSessionFactory().openSession()) {

TypedQuery<User> query = session.createQuery("SELECT u FROM User u " +

"WHERE name = :name AND password = :pass", User.class);

query.setParameter("name", name);

query.setParameter("pass", password);

return query.getSingleResult();

} catch (NoResultException ex) {

System.err.println("User not found");

return null;

}

}

}

WarehouseService

package com.umler.warehouses.Services;  
  
import com.umler.warehouses.Model.Product;  
import com.umler.warehouses.Model.Room;  
import com.umler.warehouses.Model.Shelf;  
  
import java.util.List;  
  
/\*\*  
 \* Класс для подсчета обзей заполненности склада  
 \* @author Umler  
 \*/  
  
public class WarehouseService {  
  
 RoomService roomService = new RoomService();  
  
 /\*\*  
 \* Возвращает заполенность склада.  
 \* @return заполенность склада  
 \*/  
 public Integer getFullnessOfWarehouse()  
 {  
 try  
 {  
 int roomcapacity = 0;  
 List<Room> rooms = roomService.getRooms();  
 if(rooms.size() != 0)  
 {  
 for (Room room : rooms){  
 List<Shelf> shelves = room.getShelvesList();  
 int shelfcapacity = 0;  
 if(shelves.size()!=0)  
 {  
 for (Shelf shelf : shelves) {  
 List<Product> products = shelf.getProductList();  
 Integer product\_space = 0;  
 for (Product product : products) {  
 product\_space += product.getQuantity();  
 }  
 shelfcapacity += product\_space \* 100 / shelf.getCapacity();  
 }  
 roomcapacity += shelfcapacity / shelves.size();  
 }  
 }  
 return roomcapacity / rooms.size();  
 }  
 }  
 catch (NullPointerException ex)  
 {  
 System.out.println("NULL" + ex);  
 }  
 return null;  
 }  
}

App

package com.umler.warehouses;

import com.umler.warehouses.Controllers.SceneController;

import com.umler.warehouses.Helpers.HibernateUtil;

import javafx.application.Application;

import javafx.stage.Stage;

import java.io.IOException;

import org.slf4j.\*;

/\*\*

\* Стартовый класс для запуска приложения

\* @author Umler

\*/

public class App extends Application {

private static final Logger logger = LoggerFactory.getLogger("Main Logger");

/\*\*

\* Начало программы

\*/

@Override

public void start(Stage stage) throws IOException {

SceneController.getInitialScene(stage);

logger.debug("Start of a program");

}

/\*\*

\* Конец программы

\*/

@Override

public void stop() throws Exception {

logger.debug("End of a program");

super.stop();

HibernateUtil.shutdown();

}

public static void main(String[] args) {

launch(args);

}

}