МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.Ф. УТКИНА»

Факультет вычислительной те­хники

Кафедра электронных вычислительных машин

К защите

Руководитель проекта

|  |
| --- |
|  |

дата, подпись

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

по дисциплине

**«Промышленное программирование»**

**по теме**

**«Разработка web-приложения на языке Java»**

Выполнил:

студент группы 845

|  |  |
| --- | --- |
| Муханов В.С. |  |
|  | дата сдачи на проверку, подпись |

Руководитель проекта:

ассистент кафедры ЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Бастрычкин А.С. |  |  |  |  |
|  |  | оценка |  | дата защиты, подпись |

Рязань, 2021 г.

Требования к содержанию пояснительной записки:

ВВЕДЕНИЕ

1. Теоретическая часть
   1. Описание предметной области
   2. Технологии, с помощью которых была разработана КР
2. Характеристики ИС
3. Разработка серверной части информационной системы

3.1 Инфологическое проектирование базы данных

3.1.1 Выделение сущностей

3.1.2 Построение ER диаграмм

3.2 Даталогическое проектирование БД

3.2.1 Переход от ER диаграмм к предварительным отношениям

3.2.2 Заполнение предварительных отношений атрибутами

3.2.3 Формирование предварительной базы данных

1. Разработка web-приложения

4.1 Выбор программных компонентов web-приложения

4.2 Разработка интерфейса пользователя

4.2.1 Разработка форм

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЕ А: Код рабочего web-приложения

Дата выдачи задания: \_\_\_ сентября 2021 г.

Срок сдачи проекта на проверку: \_\_\_ декабря 2021 г.

Подпись исполнителя:

Подпись руководителя:

**Оглавление**

[Введение 4](#_Toc88662478)

[1 Теоретическая часть 5](#_Toc88662479)

[1.1 Описание предметной области 5](#_Toc88662480)

[1.2 Технологии, с помощью которых была разработана КР 6](#_Toc88662481)

[2 Характеристики ИС 10](#_Toc88662482)

[3 Разработка серверной части информационной системы 11](#_Toc88662483)

[3.1 Инфологическое проектирование БД 11](#_Toc88662484)

[3.1.1 Выделение сущностей 11](#_Toc88662489)

[3.1.2 Построение ER диаграмм 11](#_Toc88662490)

[3.2 Даталогическое проектирование БД 12](#_Toc88662491)

[3.2.1 Переход от ER диаграмм к предварительным отношениям 12](#_Toc88662492)

[3.2.2 Заполнение предварительных отношений атрибутами 12](#_Toc88662493)

[3.2.3 Формирование предварительной базы данных 13](#_Toc88662494)

[4 Разработка web-приложения 14](#_Toc88662495)

[4.1 Выбор программных компонентов web-приложения 14](#_Toc88662496)

[4.2 Разработка интерфейса пользователя 14](#_Toc88662497)

[4.2.1 Разработка окон 14](#_Toc88662498)

[Заключение 21](#_Toc88662499)

[Список литературы 22](#_Toc88662500)

[ПРИЛОЖЕНИЕ А: рабочий код web-приложения. 23](#_Toc88662501)

Введение

В данной курсовой работе рассматривается проектирование web-приложения средствами языка высокого уровня Java. За основу работы была взята структура справочника управляющей компании.

**Веб-приложение** — клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами.

Веб-приложения стали широко использоваться в конце 1990-х — начале 2000-х годов.

1 Теоретическая часть

* 1. Описание предметной области

Управляющая компания — коммерческая организация, юридическое лицо, осуществляющее доверительное управление имуществом других физических и юридических лиц, переданным в соответствии с договором доверительного управления.

Целью деятельности управляющих организаций в сфере ЖКХ является содержание ОИ (общего имущества) МКД (в том числе придомовой территории) в надлежащем состоянии, а также обеспечение возможности использования ОИ по его назначению.

С точки зрения организации деятельности, управляющая организация является посредником между собственниками многоквартирного жилого дома и лицами, оказывающими услуги по его содержанию и ремонту общего имущества многоквартирного дома, РСО (ресурсоснабжающими организациями) и другими организациями, деятельность которых связана с ОИ МКД дома (арендаторы общедомового имущества, организации по охране ОИ и пр.).

Для учета обращений собственников и жильцов необходимо разработать программное обеспечение, предусматривающее контроль исполнения заявок и содержащие следующие сущности:

* Физ. лица
* Дома
* Квартиры

1.2 Технологии, с помощью которых была разработана КР

Основная часть web-приложения, которая видна пользователю на web-странице, описана с помощью объектно-ориентированного языка высокого уровня Java. База данных, к которой обращается данное приложение, спроектирована в СУБД PostgresSQL.

**Java** — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией.

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

* применение технологии трансляции байт-кода в машинный код

непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,

* обширное использование платформенно-ориентированного кода

(native-код) в стандартных библиотеках,

* аппаратные средства, обеспечивающие ускоренную обработку

байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами архитектуры ARM).

По данным сайта shootout.alioth.debian.org, для семи разных задач время выполнения на Java составляет в среднем в полтора-два раза больше, чем для C/C++, в некоторых случаях Java быстрее, а в отдельных случаях в 7 раз медленнее. С другой стороны, для большинства из них потребление памяти Java-машиной было в 10—30 раз больше, чем программой на C/C++. Также примечательно исследование, проведённое компанией Google, согласно которому отмечается существенно более низкая производительность и бо́льшее потребление памяти в тестовых примерах на Java в сравнении с аналогичными программами на C++.

Идеи, заложенные в концепцию и различные реализации среды виртуальной машины Java, вдохновили множество энтузиастов на расширение перечня языков, которые могли бы быть использованы для создания программ, исполняемых на виртуальной машине. Эти идеи нашли также выражение в спецификации общеязыковой инфраструктуры CLI, заложенной в основу платформы .NET компанией Microsoft.

**Spring Framework** (или коротко **Spring**) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET. Особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

**Spring Boot** — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода. Spring Boot обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов.

Spring Boot позволяет вам легко создавать полноценные, производственного класса Spring-приложения, про которые можно сказать - "просто запусти". Большинству Spring Boot приложениям требуется совсем маленькая Spring-конфигурация.

Возможности Spring Boot Java:

* создание полноценных Spring приложений
* встроенный Tomcat или Jetty (не требуется установки WAR

файлов)

* обеспечивает 'начальные' POMs для упрощения вашей Maven

конфигурации

* автоматическая конфигурирация Spring когда это возможно
* обеспечивает такими возможностями, как метрики, мониторинг

состояниями и расширенная конфигурация

* абсолютно без генерации кода и без написания XML

конфигурации

**Mustache** — шаблонизатор, который содержит минимум управляющей логики и доступен для разных языков программирования. Шаблон пишется на простом языке, состоящем из нескольких типов тегов. Теги обрамляются двумя или тремя фигурными скобками с каждой стороны. Можно использовать вложенные шаблоны.

**PostgreSQL** — это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности.

Преимущества PostgreSQL:

* поддержка БД неограниченного размера;
* мощные и надёжные механизмы транзакций и репликации;
* расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;
* наследование;
* легкая расширяемость.

Текущие ограничения PostgreSQL:

* нет ограничений на максимальный размер базы данных
* нет ограничений на количество записей в таблице
* нет ограничений на количество индексов в таблице
* максимальный размер таблицы — 32 Тбайт
* максимальный размер записи — 1,6 Тбайт
* максимальный размер поля — 1 Гбайт
* максимум полей в записи250—1600 (в зависимости от типов полей)

Особенности PostgreSQL:

Функции в PostgreSQL являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом SQL, реализация дополнительной логики, например, условных переходов и циклов, выходит за рамки, собственно, SQL и требует использования некоторых языковых расширений. Функции могут писаться с использованием различных языков программирования. PostgreSQL допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса. Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя. Иногда функции отождествляются с хранимыми процедурами, однако между этими понятиями есть различие.

Наследование в PostgreSQL реализовано на уровне таблиц. Таблицы могут наследовать характеристики и наборы полей от других таблиц (родительских). При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице.

1. Характеристики ИС

Необходимо разработать базу данных управляющей компании и хранить в ней следующую информацию:

Обслуживаемые дома (Адрес, Кол-во Квартир, Кол-во Подъездов, Кол-во Этажей)

Находящиеся в домах квартиры (Код Квартиры, Адрес, Подъезд, Этаж, Номер квартиры)

Список физических лиц, проживающих в квартирах или являющихся сотрудниками (ФИО, Паспорт, Телефон)

1. Разработка серверной части информационной системы
   1. Инфологическое проектирование БД

4. 1. 1. Выделение сущностей

* Дом (Адрес)
* Квартира (Код Квартиры)
* Физ. лицо (ФИО)

В предметной области можно выбелить следующую связь между сущностями:

* Дом содержит Квартиры
* Квартирой владеет Физ. лицо
  + 1. Построение ER диаграмм

Диаграммы должны включать все выделенные сущности и связи между ними. На данном этапе должны быть определены характеристики связи, а именно степень связи и класс принадлежности сущности к связи.

1. Связь Дом содержит Квартиры

Для степени связи:

* В Доме может быть много Квартир
* Квартира может быть только в одном Доме

Для класса принадлежности сущности к связи:

* Дом обязательно содержит Квартиры
* Квартира обязательно есть в Доме

ER-диаграмма по данной связи представлена на Рисунке 3.1.



Рисунок 3.1 - Связь Дом содержит Квартиры

2. Связь Квартирой владеет/проживает Физ. лицо

Для степени связи:

* Квартирой может владеть/проживать несколько Персон
* Физ. лицо может владеть несколькими Квартирами

ER-диаграмма по данной связи представлена на Рисунке 3.2.

Для класса принадлежности сущности к связи:

* Квартирой не обязательно владеет/проживает Физ. лицо
* Физ. лицо не обязательно владеет/проживает в Квартире

ER-диаграмма по данной связи представлена на Рисунке 3.3.

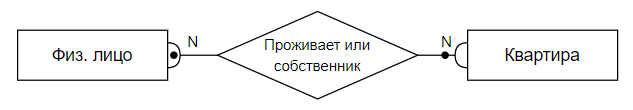


Рисунок 3.2 - Связь Квартирой владеет/проживает Физ. лицо



Рисунок 3.3 — Общая ER диаграмма

* 1. Даталогическое проектирование БД
     1. Переход от ER диаграмм к предварительным отношениям

Имеем следующие отношения:

* Дом (Адрес)
* Квартира (Код Квартиры, Адрес, Владелец)
* Физ. лицо (ФИО)
  + 1. Заполнение предварительных отношений атрибутами
* Дом (Адрес, Кол-во Квартир, Кол-во Подъездов, Кол-во Этажей)
* Квартира (Код Квартиры, Адрес, Подъезд, Этаж, Номер квартиры)
* Физ. Лицо (ФИО, Паспорт, Телефон)
  + 1. Формирование предварительной базы данных

Предварительная схема базы данных представлена на рисунке 3.4.

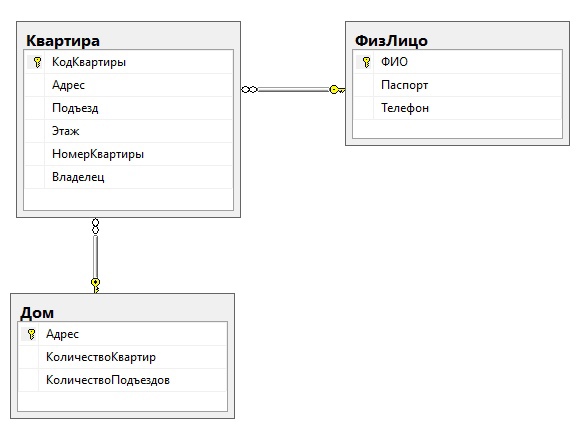


Рисунок 3.4 — Общая схема БД

1. Разработка web-приложения
   1. Выбор программных компонентов web-приложения

Web-приложение разрабатывалось с помощью программы IntelliJ IDEA Community Edition 2021.1 и написано на языке программирования Java. Код программы представлен в Приложении A. Объекты базы данных разрабатывались с помощью СУБД PostgreSQL.

* 1. Разработка интерфейса пользователя

При переходе на сайт по адресу <http://localhost:8080/>, пользователь попадает на главную страницу (Рисунок 4.1) с инструкцией: «Выберите справочник» и тремя кнопками выбора: «Физ. лица», «Дома», «Квартиры».

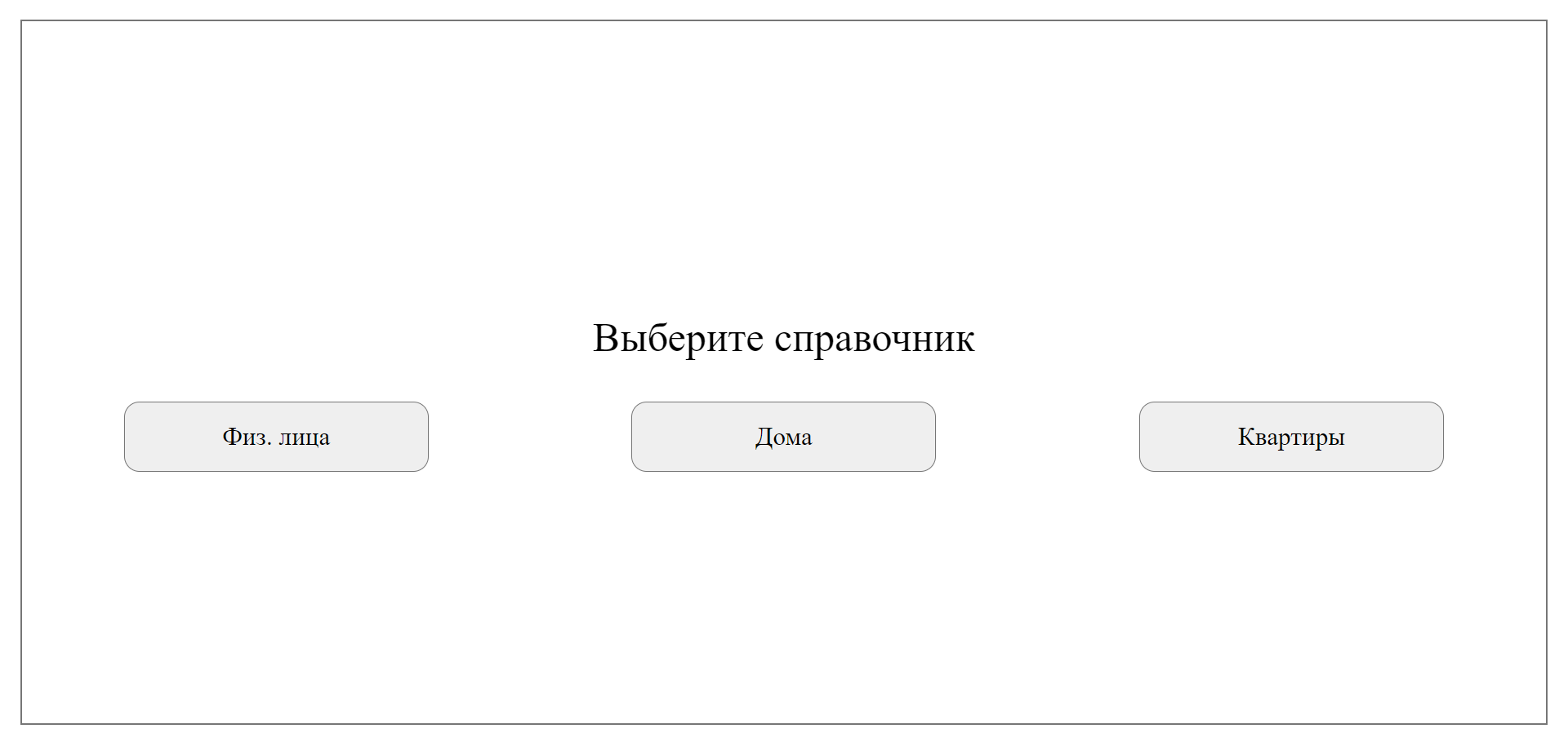


Рисунок 4.1 — Главное окно приложения

* + 1. Разработка окон

Разработаем страницу справочника физ. лиц. Данное окно позволяет добавлять физ. лицо, редактировать его, удалять, а также выводит список всех физ. лиц. (Рисунок 4.2).

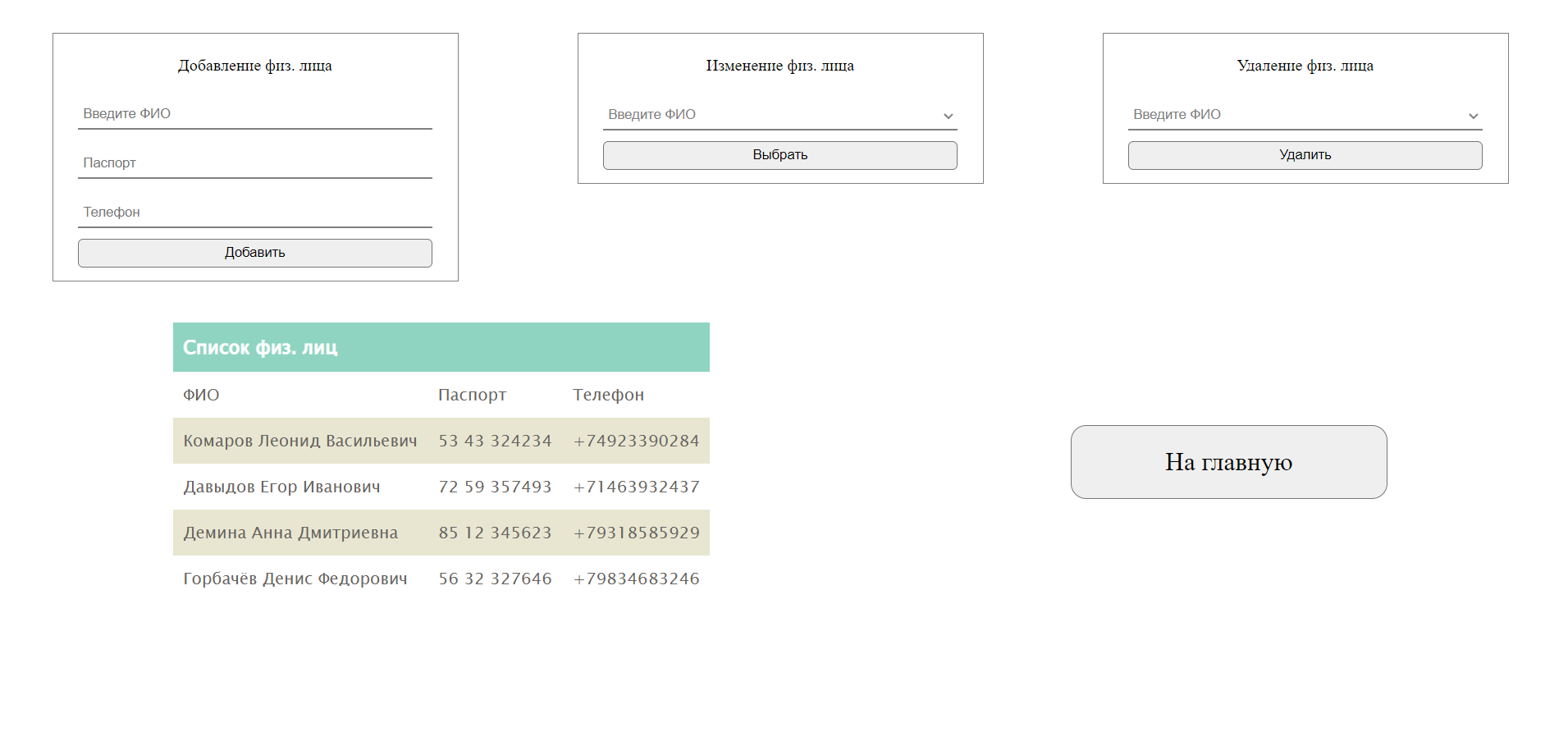


Рисунок 4.2 — Справочник физ. лиц

Для добавления физ. лица необходимо на форме «Добавление физ. лица» заполнить поля:

* ФИО
* Паспорт
* Телефон

Для изменения данных о физ. лице необходимо на форме «Изменение физ. лица» выбрать ФИО из выпадающего списка в поле «Введите ФИО» и нажать кнопку «Выбрать», перейдя на страницу редактирования физ. лица, а затем изменить значения в полях «Паспорт», «Телефон», «Email» и нажать на кнопку «Изменить».

Для удаления данных о конкретном физ. лице достаточно ввести его ФИО в поле «Введите ФИО» на форме «Удаление физ. лица» и выбрать необходимого человека с помощью выпадающего списка.

Далее пользователь может вернуться на главную страницу с помощью

соответствующей кнопки и перейти на страницу «Дома» (Рисунок 4.3).

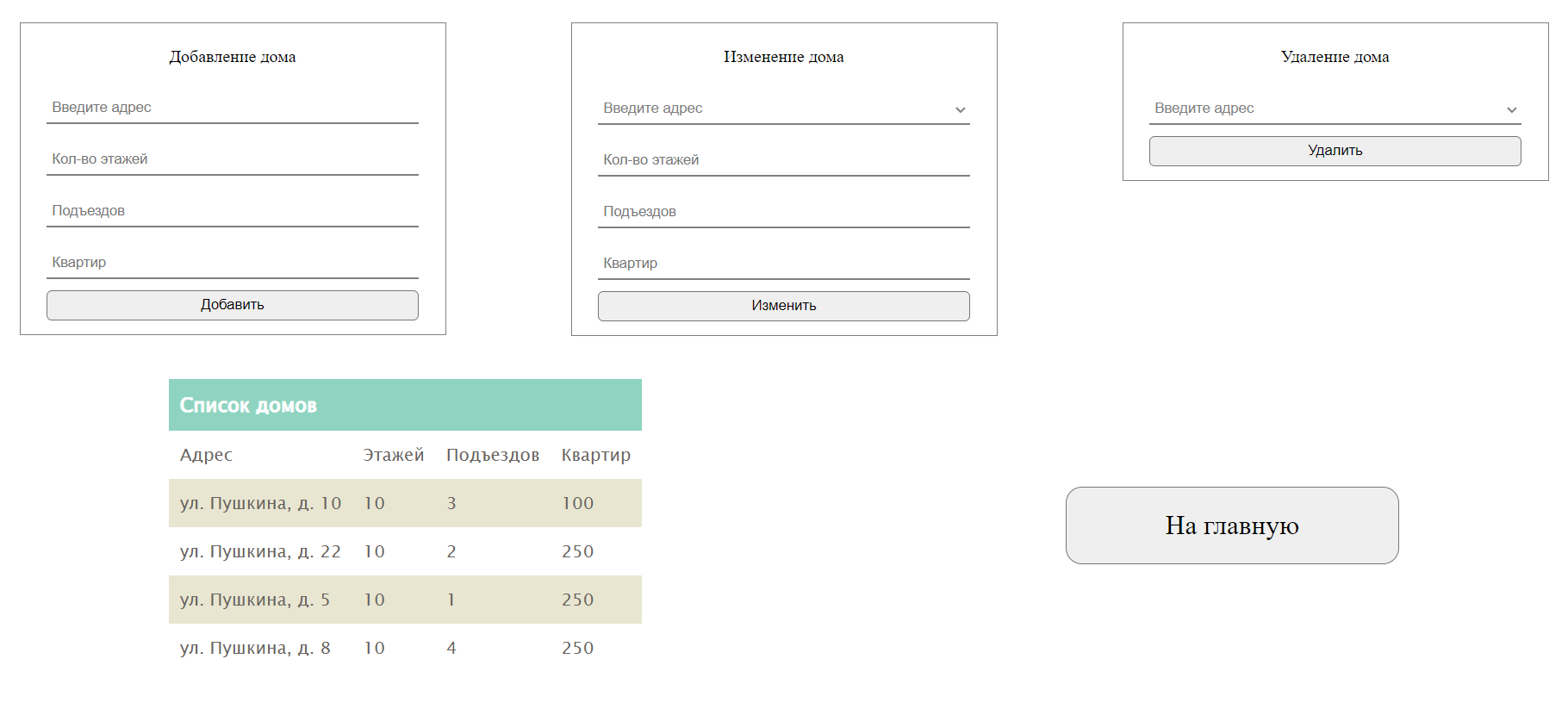


Рисунок 4.3 — Справочник домов

Для добавления дома на форме «Добавление дома» необходимо заполнить поля:

* Адрес
* Кол-во этажей
* Подъездов
* Квартир

Для изменения данных о доме необходимо на форме «Изменение дома» выбрать Адрес дома из выпадающего списка в поле «Адрес», а затем изменить значения в доступных полях.

Для удаления данных о конкретном доме достаточно ввести его Адрес в поле «Введите адрес» на форме «Удаление дома» и выбрать необходимый дом с помощью выпадающего списка.

Далее пользователь может вернуться на главную страницу с помощью

соответствующей кнопки и перейти на страницу «Квартиры» (Рисунок 4.4).

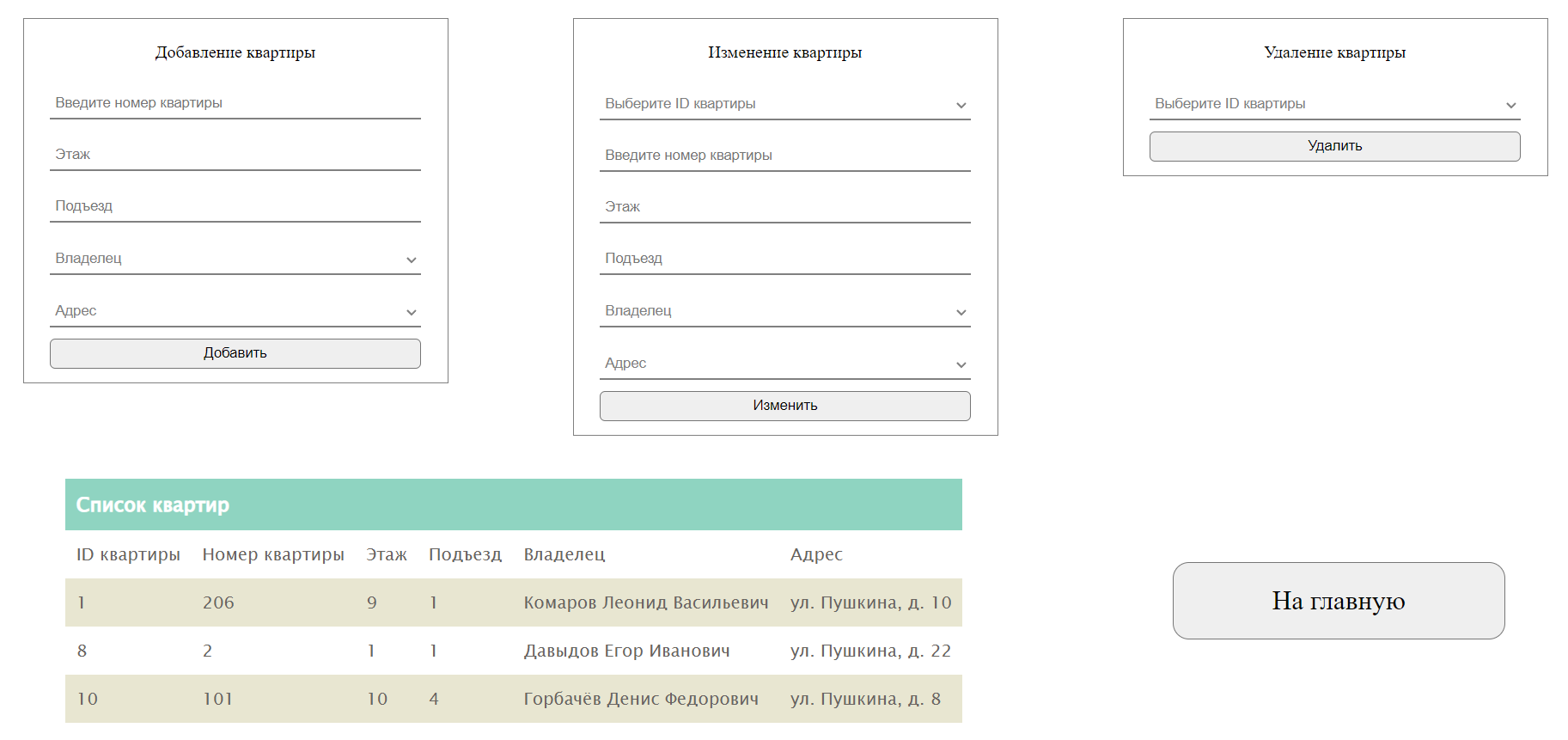


Рисунок 4.4 — Справочник квартир

Для добавления квартиры на форме «Добавление квартиры» необходимо заполнить поля:

* Номер квартиры
* Этаж
* Подъезд
* Владелец
* Адрес

Для изменения данных о квартире необходимо на форме «Изменение квартире» выбрать ID квартиры из выпадающего списка в поле «ID квартиры», а затем изменить значения в доступных полях.

Для удаления данных о квартире достаточно ввести ее ID в поле «Введите ID» на форме «Удаление квартиры» и выбрать необходимую квартиру с помощью выпадающего списка.

Далее пользователь может вернуться на главную страницу с помощью

соответствующей кнопки.

Заполнение всех полей всех окон и результирующая строка в таблице «Список физ. лиц» представлены на Рисунках 4.5 - 4.7.

Добавление физ. лица:

1. Добавим физ. лицо (Рисунок 4.5).



Рисунок 4.5 — Ввод нового физ. лица

Изменим данные физ. лица (Рисунок 4.6).



Рисунок 4.6 — Изменение данных

Удалим физ. лицо (Рисунок 4.7).



Рисунок 4.7 — Удаление физ. лица из списка

1. Перейдем на страницу «Дома» - создадим новый дом (Рисунок 4.8).



Рисунок 4.8 — Создание нового дома

Изменим данный дом (Рисунок 4.9).



Рисунок 4.9 — Изменение дома

Удалим дом (Рисунок 4.10).

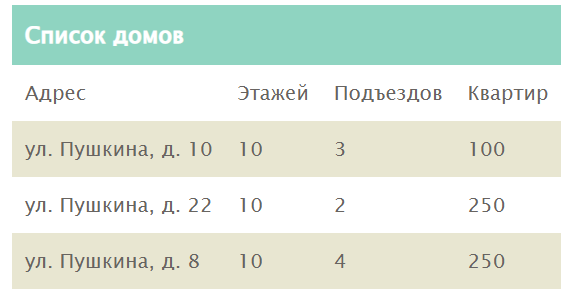


Рисунок 4.10 — Удаление дома

1. Перейдем на страницу справочника квартир и добавим новую квартиру (Рисунок 4.11).

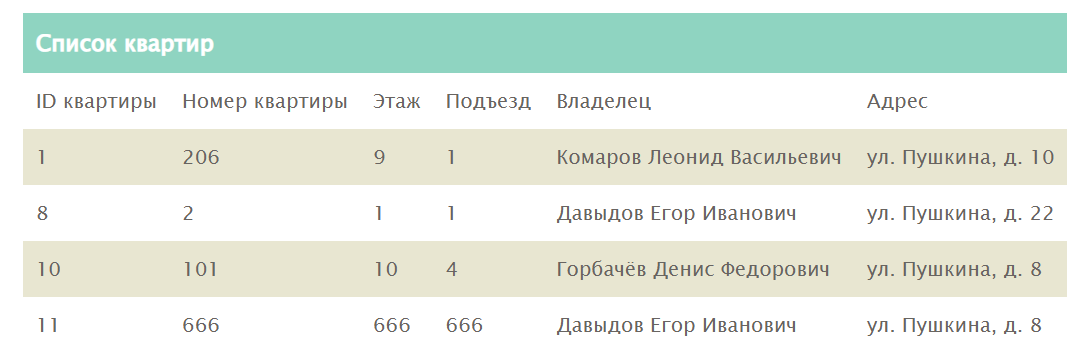


Рисунок 4.11 — Добавление квартиры

Изменим данные о квартире (Рисунок 4.12).



Рисунок 4.12 — Изменение данных

Удалим квартиру (Рисунок 4.13).



Рисунок 4.13 — Удаление квартиры

Были проделаны все операции, доступные в web-приложении. Поскольку полученные в ходе работы результаты совпадают с ожидаемыми, то можно сделать вывод, что данное приложение работает корректно.

Заключение

В результате выполнения данного курсового проекта было разработано web-приложение-справочник для управляющей компании. Была разработана база данных для взаимодействия с приложением. Была проделана работа и получен опыт разработки приложений на языке Java с использованием среды разработки IntelliJ IDEA Community Edition 2021.1.

Список литературы

1. Лекции по предмету «Промышленное программирование», преподаватель Бастрычкин А.С.
2. Курс по Java, компания Epam, ссылка: <https://autocode.lab.epam.com/student/>
3. Справочник по работе с PostgreSQL, ссылка: <https://postgrespro.ru/docs/postgresql/9.6/sql-commands>
4. Обучающие видеоролики по Java Spring Boot, сервис YouTube, ссылка: <https://www.youtube.com/playlist?list=PLU2ftbIeotGpAYRP9Iv2KLIwK36-o_qYk>

ПРИЛОЖЕНИЕ А: рабочий код web-приложения.

* Текст класса **Person**:

package com.example.servingwebcontent.domain;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

@Entity

public class Person {

@Id

private String fio;

private String passport;

private String phone;

@OneToMany(mappedBy = "owner", fetch = FetchType.EAGER, orphanRemoval = false)

private List<Apart> listApart = new ArrayList<>();

public Person() {

}

public Person(String fio, String passport, String phone) {

this.fio = fio;

this.passport = passport;

this.phone = phone;

}

public String getFio() {

return fio;

}

public void setFio(String fio) {

this.fio = fio;

}

public String getPassport() {

return passport;

}

public void setPassport(String passport) {

this.passport = passport;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

}

* Текст класса **House**:

package com.example.servingwebcontent.domain;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

@Entity

public class House {

@Id

private String address;

private Integer floorsNumber;

private Integer entrancesNumber;

private Integer apartsNumber;

@OneToMany(mappedBy = "address", fetch = FetchType.EAGER, orphanRemoval = false)

private List<Apart> listApart = new ArrayList<>();

public House() {

}

public House(String address, Integer floorsNumber, Integer entrancesNumber, Integer apartsNumber) {

this.address = address;

this.floorsNumber = floorsNumber;

this.entrancesNumber = entrancesNumber;

this.apartsNumber = apartsNumber;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Integer getFloorsNumber() {

return floorsNumber;

}

public void setFloorsNumber(Integer floorsNumber) {

this.floorsNumber = floorsNumber;

}

public Integer getEntrancesNumber() {

return entrancesNumber;

}

public void setEntrancesNumber(Integer entrancesNumber) {

this.entrancesNumber = entrancesNumber;

}

public Integer getApartsNumber() {

return apartsNumber;

}

public void setApartsNumber(Integer apartsNumber) {

this.apartsNumber = apartsNumber;

}

}

* Текст класса **Apart**:

package com.example.servingwebcontent.domain;

import javax.persistence.\*;

@Entity

public class Apart {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private Integer apartId;

private Integer apartNumber;

private Integer floor;

private Integer entrance;

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "person.fio")

private Person owner;

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "house.address")

private House address;

public Apart() {

}

public Apart(Integer apartNumber, Integer floor, Integer entrance, Person owner, House address) {

this.apartNumber = apartNumber;

this.floor = floor;

this.entrance = entrance;

this.owner = owner;

this.address = address;

}

public Integer getApartId() {

return apartId;

}

public void setApartId(Integer apartId) {

this.apartId = apartId;

}

public Integer getApartNumber() {

return apartNumber;

}

public void setApartNumber(Integer apartNumber) {

this.apartNumber = apartNumber;

}

public Integer getFloor() {

return floor;

}

public void setFloor(Integer floor) {

this.floor = floor;

}

public Integer getEntrance() {

return entrance;

}

public void setEntrance(Integer entrance) {

this.entrance = entrance;

}

public Person getOwner() {

return owner;

}

public void setOwner(Person owner) {

this.owner = owner;

}

public House getAddress() {

return address;

}

public void setAddress(House address) {

this.address = address;

}

}

* Текст интерфейса **ApartRepo**:

package com.example.servingwebcontent.repos;

import com.example.servingwebcontent.domain.Apart;

import org.springframework.data.repository.CrudRepository;

public interface ApartRepo extends CrudRepository<Apart, Integer> {

}

* Текст интерфейса **HouseRepo**:

package com.example.servingwebcontent.repos;

import com.example.servingwebcontent.domain.House;

import org.springframework.data.repository.CrudRepository;

public interface HouseRepo extends CrudRepository<House, String>{

}

* Текст интерфейса **PersonRepo**:

package com.example.servingwebcontent.repos;

import com.example.servingwebcontent.domain.Person;

import org.springframework.data.repository.CrudRepository;

public interface PersonRepo extends CrudRepository<Person, String> {

}

* Текст класса **ServingWebContentApplication**:

package com.example.servingwebcontent;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ServingWebContentApplication {

public static void main(String[] args) {

SpringApplication.run(ServingWebContentApplication.class, args);

}

}

* Текст класса **HandbookController**:

package com.example.servingwebcontent;

import com.example.servingwebcontent.domain.Apart;

import com.example.servingwebcontent.domain.House;

import com.example.servingwebcontent.domain.Person;

import com.example.servingwebcontent.repos.HouseRepo;

import com.example.servingwebcontent.repos.PersonRepo;

import com.example.servingwebcontent.repos.ApartRepo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import java.util.Map;

@Controller

public class HandbookController {

@Autowired

private PersonRepo personRepo;

@Autowired

private ApartRepo apartRepo;

@Autowired

private HouseRepo houseRepo;

@GetMapping

public String main(Map<String, Object> model) {

return "main";

}

@GetMapping("/persons")

public String readPersons(Map<String, Object> model) {

Iterable<Person> persons = personRepo.findAll();

model.put("persons", persons);

String error = "Введите ФИО";

model.put("error", error);

return "persons";

}

@PostMapping("addPerson")

public String addPerson(@RequestParam String fio, @RequestParam String passport,

@RequestParam String phone, Map<String, Object> model) {

Person person = new Person(fio, passport, phone);

personRepo.save(person);

Iterable<Person> persons = personRepo.findAll();

model.put("persons", persons);

String error = "Введите ФИО";

model.put("error", error);

return "persons";

}

@PostMapping("editPerson")

public String editPerson(@RequestParam String fio, @RequestParam String passport,

@RequestParam String phone, Map<String, Object> model) {

Person person = personRepo.findById(fio).orElseThrow();

person.setPassport(passport);

person.setPhone(phone);

personRepo.save(person);

Iterable<Person> persons = personRepo.findAll();

model.put("persons", persons);

String error = "Введите ФИО";

model.put("error", error);

return "persons";

}

@PostMapping("editPersonSelect")

public String editPersonSelect(@RequestParam String fio, Map<String, Object> model) {

Person person = personRepo.findById(fio).orElseThrow();

model.put("persons", person);

return "personsSelect";

}

@PostMapping("deletePerson")

public String deletePerson(@RequestParam String fio, Map<String, Object> model) {

Person person = personRepo.findById(fio).orElseThrow();

Iterable<Apart> aparts = apartRepo.findByOwner(person);

String error = "Введите ФИО";

int counter = 0;

for (Object i : aparts) {

counter++;

}

if (counter == 0) {

personRepo.delete(person);

}

else {

error = "Нарушение внешнего ключа!";

}

model.put("error", error);

Iterable<Person> persons = personRepo.findAll();

model.put("persons", persons);

return "persons";

}

@GetMapping("/houses")

public String readHouses(Map<String, Object> model) {

Iterable<House> houses = houseRepo.findAll();

model.put("houses", houses);

return "houses";

}

@PostMapping("addHouse")

public String addHouse(@RequestParam String address, @RequestParam Integer floors,

@RequestParam Integer entrances, @RequestParam Integer aparts,

Map<String, Object> model) {

House house = new House(address, floors, entrances, aparts);

houseRepo.save(house);

Iterable<House> houses = houseRepo.findAll();

model.put("houses", houses);

return "houses";

}

@PostMapping("editHouse")

public String editHouse(@RequestParam String address, @RequestParam Integer floors,

@RequestParam Integer entrances, @RequestParam Integer aparts,

Map<String, Object> model) {

House house = houseRepo.findById(address).orElseThrow();

house.setFloorsNumber(floors);

house.setEntrancesNumber(entrances);

house.setApartsNumber(aparts);

houseRepo.save(house);

Iterable<House> houses = houseRepo.findAll();

model.put("houses", houses);

return "houses";

}

@PostMapping("deleteHouse")

public String deleteHouse(@RequestParam String address, Map<String, Object> model) {

House house = houseRepo.findById(address).orElseThrow();

houseRepo.delete(house);

Iterable<House> houses = houseRepo.findAll();

model.put("houses", houses);

return "houses";

}

@GetMapping("/aparts")

public String readApart(Map<String, Object> model) {

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "aparts";

}

@PostMapping("addApart")

public String addApart(@RequestParam Integer apartNumber, @RequestParam Integer floor,

@RequestParam Integer entrance, @RequestParam String owner,

@RequestParam String address, Map<String, Object> model) {

Person person = personRepo.findById(owner).orElseThrow();

House house = houseRepo.findById(address).orElseThrow();

Apart apart = new Apart(apartNumber, floor, entrance, person, house);

apartRepo.save(apart);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "aparts";

}

@PostMapping("editApart")

public String editApart(@RequestParam Integer apartNumber, @RequestParam Integer floor,

@RequestParam Integer entrance, @RequestParam String owner,

@RequestParam String address, @RequestParam Integer apartId,

Map<String, Object> model) {

Apart apart = apartRepo.findById(apartId).orElseThrow();

Person person = personRepo.findById(owner).orElseThrow();

House house = houseRepo.findById(address).orElseThrow();

apart.setApartNumber(apartNumber);

apart.setFloor(floor);

apart.setEntrance(entrance);

apart.setOwner(person);

apart.setAddress(house);

apartRepo.save(apart);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "aparts";

}

@PostMapping("deleteApart")

public String deleteApart(@RequestParam Integer apartId, Map<String, Object> model) {

Apart apart = apartRepo.findById(apartId).orElseThrow();

apartRepo.delete(apart);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "aparts";

}

@GetMapping("/handbook")

public String mainH(Map<String, Object> model) {

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "handbook";

}

@PostMapping("/addPersonH")

public String addPersonH(@RequestParam String fio, @RequestParam String passport,

@RequestParam String phone, Map<String, Object> model) {

Person person = new Person(fio, passport, phone);

personRepo.save(person);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "handbook";

}

@PostMapping("editPersonH")

public String editPersonH(@RequestParam String fio, @RequestParam String passport,

@RequestParam String phone, Map<String, Object> model) {

Person person = personRepo.findById(fio).orElseThrow();

person.setPassport(passport);

person.setPhone(phone);

personRepo.save(person);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "handbook";

}

@PostMapping("deletePersonH")

public String deletePersonH(@RequestParam String fio, Map<String, Object> model) {

Person person = personRepo.findById(fio).orElseThrow();

personRepo.delete(person);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "handbook";

}

@PostMapping("apart")

public String addApartH(@RequestParam Integer apartNumber, @RequestParam Integer floor,

@RequestParam Integer entrance, @RequestParam String owner,

@RequestParam String address, Map<String, Object> model) {

Person person = personRepo.findById(owner).orElseThrow();

House house = houseRepo.findById(address).orElseThrow();

Apart apart = new Apart(apartNumber, floor, entrance, person, house);

apartRepo.save(apart);

Iterable<Person> persons = personRepo.findAll();

Iterable<Apart> aparts = apartRepo.findAll();

Iterable<House> houses = houseRepo.findAll();

model.put("persons", persons);

model.put("aparts", aparts);

model.put("houses", houses);

return "handbook";

}

}

* Текст файла **main.mustache**:

<html>

<head>

</head>

<body>

<div class="nav\_label">Выберите справочник</div>

<div class="nav\_panel">

<a class="nav\_button" href="persons">Физ. лица</a>

<a class="nav\_button" href="houses">Дома</a>

<a class="nav\_button" href="aparts">Квартиры</a>

</div>

</body>

</html>

* Текст файла **houses.mustache**:

<html>

<head>

</head>

<body>

<div class="form\_container">

<form method ="post" action="addHouse">

<p>Добавление дома</p>

<input type="text" name="address" placeholder="Введите адрес " />

<input type="text" name="floors" placeholder="Кол-во этажей " />

<input type="text" name="entrances" placeholder="Подъездов " />

<input type="text" name="aparts" placeholder="Квартир " />

<button type="submit">Добавить</button>

</form>

<form method ="post" action="editHouse">

<p>Изменение дома</p>

<select name="address" onchange="this.style.color='black'">

<option value="" disabled selected hidden>Введите адрес</option>

{{#houses}}

<option value="{{address}}">{{address}}</option>

{{/houses}}

</select>

<input type="text" name="floors" placeholder="Кол-во этажей " />

<input type="text" name="entrances" placeholder="Подъездов " />

<input type="text" name="aparts" placeholder="Квартир " />

<button type="submit">Изменить</button>

</form>

<form method ="post" action="deleteHouse">

<p>Удаление дома</p>

<select name="address" onchange="this.style.color='black'">

<option value="" disabled selected hidden>Введите адрес</option>

{{#houses}}

<option value="{{address}}">{{address}}</option>

{{/houses}}

</select>

<button type="submit">Удалить</button>

</form>

</div>

<div class=table\_container>

<table>

<caption>

Список домов

</caption>

<tr>

<td> Адрес</td>

<td> Этажей</td>

<td> Подъездов</td>

<td> Квартир</td>

</tr>

{{#houses}}

<tr>

<td> {{address}}</td>

<td> {{floorsNumber}}</td>

<td> {{entrancesNumber}}</td>

<td> {{apartsNumber}}</td>

</tr>

{{/houses}}

</table>

<a href="../" class="nav\_button"> На главную</a>

</div>

</body>

</html>

* Текст файла **persons.mustache**:

<html>

<head>

</head>

<body>

<div class="form\_container">

<form method ="post" action="addPerson">

<p>Добавление физ. лица</p>

<input type="text" name="fio" placeholder="Введите ФИО " />

<input type="text" name="passport" placeholder="Паспорт " />

<input type="text" name="phone" placeholder="Телефон " />

<button type="submit">Добавить</button>

</form>

<form method ="post" action="editPerson">

<p>Изменение физ. лица</p>

<select name="fio" onchange="this.style.color='black'">

<option value="" disabled selected hidden>Введите ФИО</option>

{{#persons}}

<option value="{{fio}}">{{fio}}</option>

{{/persons}}

</select>

<input type="text" name="passport" placeholder="Паспорт " />

<input type="text" name="phone" placeholder="Телефон " />

<button type="submit">Изменить</button>

</form>

<form method ="post" action="deletePerson">

<p>Удаление физ. лица</p>

<select name="fio" onchange="this.style.color='black'">

<option value="" disabled selected hidden>Введите ФИО</option>

{{#persons}}

<option value="{{fio}}">{{fio}}</option>

{{/persons}}

</select>

<button type="submit">Удалить</button>

</form>

</div>

<div class=table\_container>

<table>

<caption>

Список физ. лиц

</caption>

<tr>

<td> ФИО</td>

<td> Паспорт</td>

<td> Телефон</td>

</tr>

{{#persons}}

<tr>

<td> {{fio}}</td>

<td> {{passport}}</td>

<td> {{phone}}</td>

</tr>

{{/persons}}

</table>

<a href="../" class="nav\_button"> На главную</a>

</div>

</body>

</html>

* Текст файла **application.properties**:

spring.datasource.url=jdbc:postgresql://localhost/Curs

spring.datasource.username=postgres

spring.datasource.password=123

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto=update