МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Тверской государственный технический университет»

( ТвГТУ)

Кафедра информационных систем

Проектирование ИС

Курсовая работа

«Разработка ИС «Бюро технической инвентаризации»»

Выполнил:

Писарь И.В.,   
студент 4 курса.

группы ИСТ-436-16

Проверил:

Полтавцев А.А.

Тверь, 2020

Оглавление

[Введение 2](#_Toc30953470)

[Постановка задачи 3](#_Toc30953471)

[Проектирование автоматизированной системы 5](#_Toc30953472)

[Прядок разработки АС 9](#_Toc30953473)

[Разработка платформы проектирования и её обоснование 9](#_Toc30953474)

[Выбор средства программирования 18](#_Toc30953475)

[Реализация Back-End части. 27](#_Toc30953476)

[Реализация Front-End части. 35](#_Toc30953477)

[Приложение А 43](#_Toc30953478)

# Введение

В данном курсовом проекте представлена программа, которая поможет организовать автоматизированный и централизованный учет квартир для **Бюро технической инвентаризации**. И хотя уже разработано много аналогов данной программы, она всегда будет в вашем личном распоряжении.

# Постановка задачи

Вариант 33. Разработать прикладное программное обеспечение деятельности отдела учета квартир «Бюро технической инвентаризации».

В нашем городе имеется 6000 зданий, в которых расположено 199000 квартир. Помещений в этих квартирах – 1 500 000 шт.

Кадастровый номер здания является уникальным. Используйте его в качестве простого первичного ключа таблицы зданий. Можете работать и с составным первичным ключом (адресом здания), но в данном случае – это не лучший вариант.

Набор данных к варианту 33

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Поле | Тип | Размер | Описание |
| 1 | Kadastr | Текстовый | 20 | Кадастровый номер здания |
| 2 | Address | Текстовый | 60 | Адрес здания |
| 3 | District | Текстовый | 15 | Район города |
| 4 | Land | Числовой | 10 | Площадь земельного участка |
| 5 | Year | Числовой | 4 | Год постройки здания |
| 6 | Material | Текстовый | 15 | Материал стен здания |
| 7 | Base | Текстовый | 15 | Материал фундамента |
| 8 | Comment | Текстовый | Авто | Примечания |
| 9 | Wear | Числовой | 2 | Износ в процентах |
| 10 | Flow | Числовой | 2 | Количество этажей в здании |
| 11 | Line | Числовой | 5 | Расстояние от центра города |
| 12 | Square | Числовой | 10 | Площадь квартир |
| 13 | Picture |  | Авто | Фото здания |
| 14 | Flats | Числовой |  | Количество квартир в здании |
| 15 | Elevator | Логический | 1 | Наличие лифта |
| 16 | Flat | Числовой | 4 | Номер квартиры |
| 17 | Storey | Числовой | 2 | Номер этажа |
| 18 | Rooms | Числовой | 1 | Количество комнат |
| 19 | Level | Логический | 1 | Квартира в двух уровнях |
| 20 | SquareFlat | Числовой | Авто | Общая площадь квартиры |
| 21 | Dwell | Числовой | Авто | Жилая площадь квартиры |
| 22 | Branch | Числовой | Авто | Вспомогательная площадь квартиры |
| 23 | Balcony | Числовой | Авто | Площадь балкона |
| 23 | Height | Числовой | Авто | Высота квартиры |
| 25 | Record | Числовой | 2 | Номер помещения в квартире |
| 26 | SquareRoom | Числовой | Авто | Площадь помещения |
| 27 | Size | Текстовый | 40 | Размеры помещения в плане |
| 28 | Name | Текстовый | 30 | Назначение (кухня, ниша … |
| 29 | Decoration | Текстовый | 60 | Отделка (паркет, обои …) |
| 30 | HeightRoom | Числовой | Авто | Высота помещения |
| 31 | Socket | Числовой | 2 | Число розеток в помещении |
| 32 | Sections | Числовой | 2 | Число элементов в батарее отопления |

# Проектирование автоматизированной системы

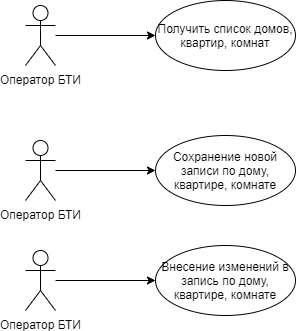


Рисунок 1.Диаграмма вариантов использования

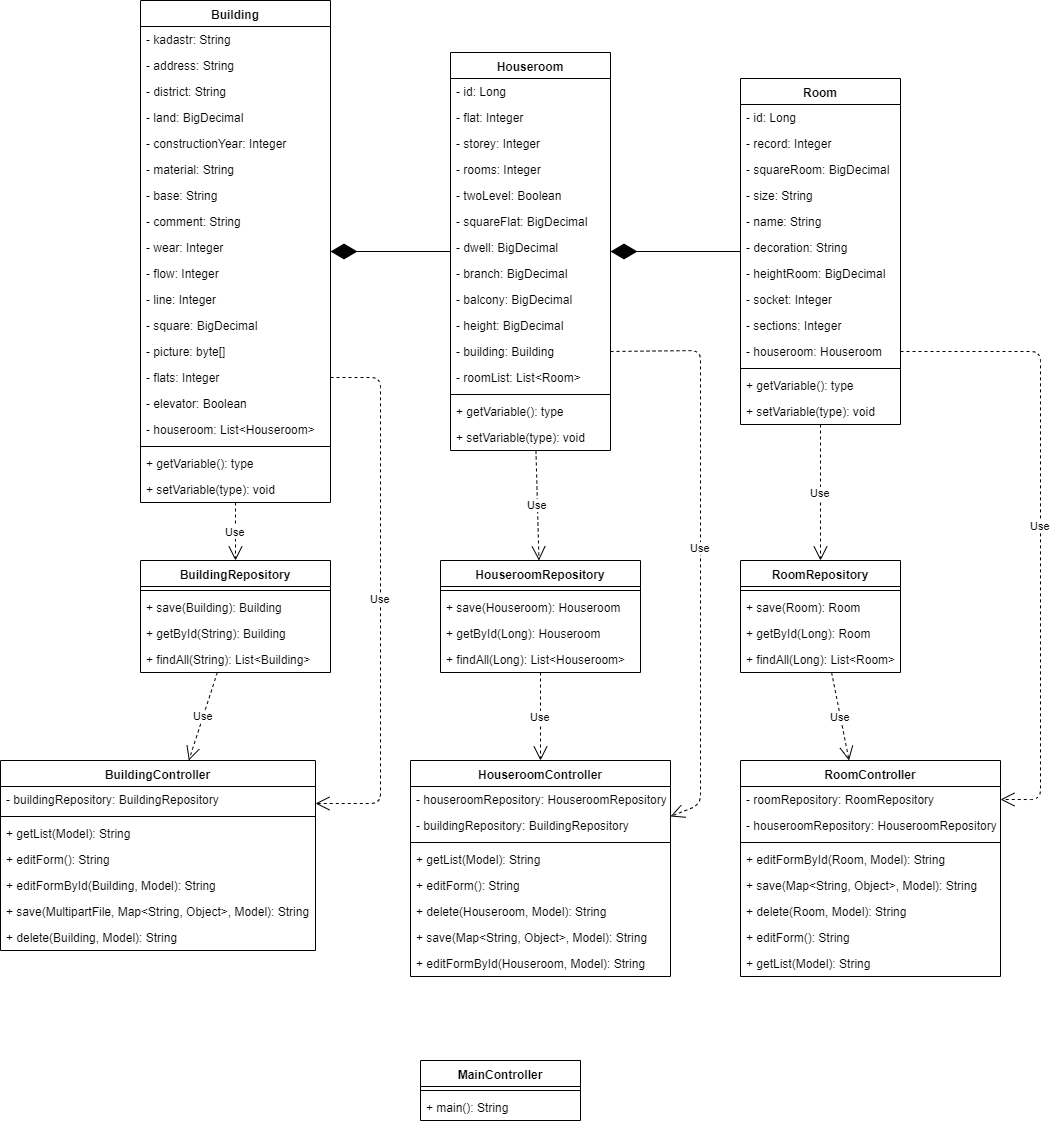


Рисунок 2. Диаграмма построения классов

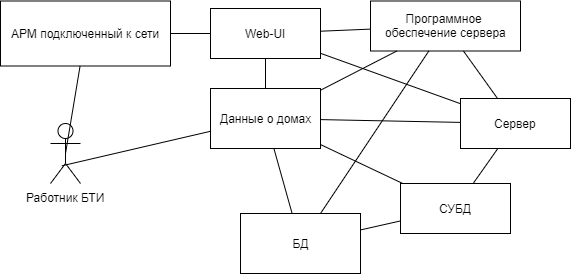


Рисунок 3. Диаграмма кооперации



Рисунок 4. Диаграмма состояний создания новой записи



Рисунок 5. Диаграмма состояний редактирования записи

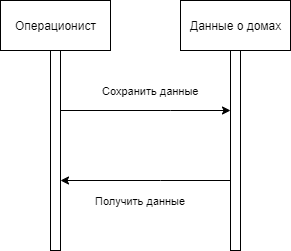


Рисунок 6. Диаграмма последовательностей для сохранения и редактирования данных о Домах



Рисунок 7. Общая диаграмма деятельности

# Прядок разработки АС

## Разработка платформы проектирования и её обоснование

Прежде, чем приступить к разработке предложенной информационной системы, была изучена информация о существующих средствах для разработки ИС. После долгих поисков в интернете и сравнивания различных вариантов, для разработки данного курсового проекта была выбрана клиент-серверная архитектура с сервером приложений.

В рамках многоуровневого представления вычислительных систем можно выделить три группы функций, ориентированных на решение различных подзадач:

функции ввода и отображения данных (обеспечивают взаимодействие с пользователем);

прикладные функции, характерные для данной предметной области;

функции управления ресурсами (файловой системой, базой даных и т.д.)

*[](http://www.4stud.info/networking/img/application-layers.png)*

Рисунок 8. Компоненты сетевого приложения

Выполнение этих функций в основном обеспечивается программными средствами, которые можно представить в виде взаимосвязанных компонентов (рис. 8), где:

*компонент представления* отвечает за пользовательский интерфейс;

*прикладной компонент* реализует алгоритм решения конкретной задачи;

*компонент управления* ресурсом обеспечивает доступ к необходимым ресурсам.

Автономная система (компьютер, не подключенный к сети) представляет все эти компоненты как на различных уровнях (ОС, служебное ПО и утилиты, прикладное ПО), так и на уровне приложений (не характерно для современных программ). Так же и сеть — она представляет все эти компоненты, но, в общем случае, распределенные между узлами. Задача сводится к обеспечению сетевого взаимодействия между этими компонентами.

**Архитектура «клиент-сервер»** определяет общие принципы организации взаимодействия в сети, где имеются *серверы*, узлы-поставщики некоторых специфичных функций (сервисов) и *клиенты*, потребители этих функций.

Практические реализации такой архитектуры называются **клиент-серверными технологиями**. Каждая технология определяет собственные или использует имеющиеся правила взаимодейстия между клиентом и сервером, которые называются *протоколом обмена (протоколом взаимодействия)*.

**Двухзвенная архитектура**

В любой сети (даже одноранговой), построенной на современных сетевых технологиях, присутствуют элементы клиент-серверного взаимодействия, чаще всего на основе **двухзвенной архитектуры**. Двухзвенной (two-tier, 2-tier) она называется из-за необходимости распределения *трех* базовых *компонентов* между *двумя узлами* (клиентом и сервером).

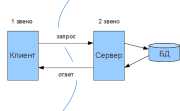
*[](http://www.4stud.info/networking/img/2-tier.png)*

Рисунок 9. Двухзвенная клиент-серверная архитектура

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса (рис. 9)

Расположение компонентов на стороне клиента или сервера определяет следующие основные модели их взаимодействия в рамках двухзвенной архитектуры:

**сервер терминалов** — распределенное представление данных;

**файл-сервер** — доступ к удаленной базе данных и файловым ресурсам;

**сервер БД** — удаленное представление данных;

**сервер приложений** — удаленное приложение.

Перечисленные модели с вариациями представлены на рис. 10.

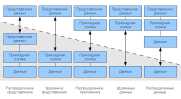
*[](http://www.4stud.info/networking/img/cs-models.png)*

Рисунок 10. Модели клиент-серверного взаимодействия

Исторически первой появилась модель распределенного представления данных (модель сервер терминалов). Она реализовывалась на универсальной ЭВМ (мэйнфрейме), выступавшей в роли сервера, с подключенными к ней алфавитно-цифровыми терминалами. Пользователи выполняли ввод данных с клавиатуры терминала, которые затем передавались на мэйнфрейм и там выполнялась их обработка, включая формирование «картинки» с результатами. Эта «картинка» и возвращалась пользователю на экран терминала.

С появлением персональных компьютеров и локальных сетей, была реализована модель файлового сервера, представлявшего доступ файловым ресурсам, в т.ч и к удаленной базе данных. В этом случае выделенный узел сети является файловым сервером, на котором размещены файлы базы данных. На клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программма), использующие подключенную удаленную базу как локальный файл. Протоколы обмена при этом представляют набор низкоуровневых вызовов операций файловой системы.

Такая модель показала свою неэффективность ввиду того, что при активной работе с таблицами БД возникает большая нагрузка на сеть. Частичным решением является поддержка тиражирования (репликации) таблиц и запросов. В этом случае, например при изменении данных, обновляется не вся таблица, а только модифицированная ее часть.

С появлением специализированных СУБД появилась возможность реализации другой модели доступа к удаленной базе данных — модели сервера баз данных. В этом случае ядро СУБД функционирует на сервере, прикладная программа на клиенте, а протокол обмена обеспечивается с помощью языка SQL. Такой подход по сравнению с файловым сервером ведет к уменьшению загрузки сети и унификации интерфейса «клиент-сервер». Однако, сетевой трафик остается достаточно высоким, кроме того, по прежнему невозможно удовлетворительное администрирование приложений, поскольку в одной программе совмещаются различные функции.

С разработкой и внедрением на уровне серверов баз данных механизма хранимых процедур появилась концепция *активного сервера БД*. В этом случае часть функций прикладного компонента реализованы в виде хранимых процедур, выполняемых на стороне сервера. Остальная прикладная логика выполняется на клиентской стороне. Протокол взаимодействия — соответствующий диалект языка SQL.

Преимущества такого подхода очевидны:

возможно централизованное администрирование прикладных функций;

снижение стоимости владения системой (TOC, total cost of ownership) за счет [аренды сервера](http://www.di-net.ru/collocation/dedicated/), а не его покупки;

значительное снижение сетевого трафика (т.к. передаются не SQL-запросы, а вызовы хранимых процедур).

Основной недостаток — ограниченность средств разработки хранимых процедур по сравнению с языками высокого уровня.

Реализация прикладного компонента на стороне сервера представляет следующую модель — сервер приложений. Перенос функций прикладного компонента на сервер снижает требования к конфигурации клиентов и упрощает администрирование, но представляет повышенные требования к производительности, безопасности и надежности сервера.

В настоящее время намечается тенденция возврата к тому, с чего начиналась клиент-серверная архитектура — к централизации вычислений на основе модели терминал-сервера. В современной реинкарнации терминалы отличаются от своих алфавитно-цифровых предков тем, что имея минимум программных и аппаратных средств, представляют мультимедийные возможности (в т.ч. *графический пользовательский интерфейс*). Работу терминалов обеспечивает высокопроизводительный сервер, куда вынесено все, вплоть до виртуальных драйверов устройств, включая драйверы видеоподсистемы.

**Трехзвенная архитектура**

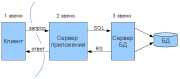
*[](http://www.4stud.info/networking/img/3-tier.png)*

Рисунок 11. Трехзвенная клиент-серверная архитектура

Еще одна тенденция в клиент-серверных технологиях связана со все большим использованием распределенных вычислений. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится **трехзвенной (three-tier, 3-tier)**.

Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом (рис. 11):

Представление данных — на стороне клиента.

Прикладной компонент — на выделенном сервере приложений (как вариант, выполняющем функции [промежуточного ПО](http://www.4stud.info/networking/lecture6.html)).

Управление ресурсами — на сервере БД, который и представляет запрашиваемые данные.

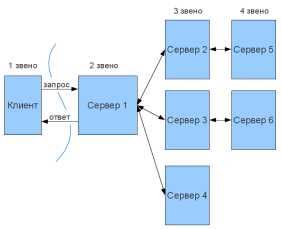
[](http://www.4stud.info/networking/img/N-tier.png)

Рисунок 12. Многозвенная (N-tier) клиент-серверная архитектура

Трехзвенная архитектура может быть расширена до **многозвенной (N-tier, Multi-tier)** путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня. Абстрактный пример многозвенной модели приведен на рис. 12.

**Сравнение архитектур**

Двухзвенная архитектура проще, так как все запросы обслуживаются одним сервером, но именно из-за этого она менее надежна и предъявляет повышенные требования к производительности сервера.

Трехзвенная архитектура сложнее, но благодаря тому, что функции распределены между серверами второго и третьего уровня, эта архитектура представляет:

Высокую степень гибкости и масштабируемости.

Высокую безопасность (т.к. защиту можно определить для каждого сервиса или уровня).

Высокую производительность (т.к. задачи распределены между серверами).

**Клиент-серверные технологии**

Архитектура клиент-сервер применяется в большом числе сетевых технологий, используемых для доступа к различным сетевым сервисам. Кратко рассмотрим некоторые типы таких сервисов (и серверов).

**Web-серверы**

Изначально представляли доступ к гипертекстовым документам по протоколу HTTP (Huper Text Transfer Protocol). Сейчас поддерживают расширенные возможности, в частности работу с бинарными файлами (изображения, мультимедиа и т.п.).

**Серверы приложений**

Предназначены для централизованного решения прикладных задач в некоторой предметной области. Для этого пользователи имеют право *запускать серверные программы* на исполнение. Использование серверов приложений позволяет снизить требования к конфигурации клиентов и упрощает общее управление сетью.

**Серверы баз данных**

Серверы баз данных используются для обработки пользовательских запросов на языке SQL. При этом СУБД находится на сервере, к которому и подключаются клиентские приложения.

**Файл-серверы**

Файл-сервер *хранит* информацию в виде файлов и представляет пользователям доступ к ней. Как правило файл-сервер обеспечивает и определенный уровень защиты от несакционированного доступа.

**Прокси-сервер**

Во-первых, действует как посредник, помогая пользователям получить информацию из Интернета и при этом обеспечивая защиту сети.

Во-вторых, сохраняет часто запрашиваемую информацию в кэш-памяти на локальном диске, быстро доставляя ее пользователям без повторного обращения к Интернету.

**Файрволы** (брандмауэры)

Межсетевые экраны, анализирующие и фильтрующие проходящий сетевой трафик, с целью обеспечения безопасности сети.

**Почтовые серверы**

Представляют услуги по отправке и получению электронных почтовых сообщений.

Серверы удаленного доступа (RAS)

Эти системы обеспечивают связь с сетью по коммутируемым линиям. Удаленный сотрудник может использовать ресурсы корпоративной ЛВС, подключившись к ней с помощью обычного модема.

Это лишь несколько типов из всего многообразия клиент-серверных технологий, используемых как в локальных, так и в глобальных сетях.

Для доступа к тем или иным сетевам сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента. Рассмотрим возможные граничные значения:

«Тонкий» клиент

Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами *статического* HTML (выполнение JavaScript не предусматривается), вся прикладная логика выполняется на сервере.  
Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

«Толстый» клиент

Таковым является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются в основном за дополнительными услугами (например, доступ к web-серверу или корпоративной базе данных).  
Так же под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

В последнее время все чаще используется еще один термин: «rich»-client. «Rich«-клиент своего рода компромисс между «толстым» и «тонким» клиентом. Как и «тонкий» клиент, «rich»-клиент также представляет графический интерфейс, описываемый уже средствами XML и включающий некоторую функциональность толстых клиентов (например интерфейс drag-and-drop, вкладки, множественные окна, выпадающие меню и т.п.)

Прикладная логика «rich»-клиента также реализована на сервере. Данные отправляются в стандартном формате обмена, на основе того же XML (протоколы SOAP, XML-RPC) и интерпретируются клиентом.

Некоторые основные протоколы «rich»-клиентов на базе XML приведены ниже:

XAML (eXtensible Application Markup Language) — разработан Microsoft, используется в приложениях на платформе .NET;

XUL (XML User Interface Language) — стандарт, разработанный в рамках проекта Mozilla, используется, например, в почтовом клиенте Mozilla Thunderbird или браузере Mozilla Firefox;

Flex — мультимедийная технология на основе XML, разработанная Macromedia/Adobe.

Для решения данной задачи была выбрана трехзвенная архитектура на тонком клиенте. Сервер приложений так же будет совмещать в себе web-сервер, так как разделение этих модулей на 2 части не целесообразно.

## Выбор средства программирования

**Back-end часть**

**Среда разработки**

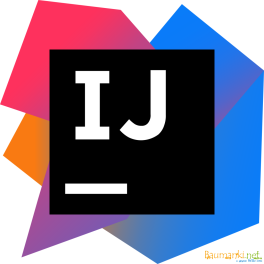


Рисунок 13. Логотип JetBrains IntelliJ IDEA

**JetBrains IntelliJ IDEA** – это ведущая среда быстрой разработки на языке Java. IntelliJ IDEA представляет собой высокотехнологичный комплекс тесно интегрированных инструментов программирования, включающий интеллектуальный редактор исходных текстов с развитыми средствами автоматизации, мощные инструменты рефакторинга кода, встроенную поддержку технологий J2EE, механизмы интеграции со средой тестирования Ant/JUnit и системами управления версиями, уникальный инструмент оптимизации и проверки кода Code Inspection, а также инновационный визуальный конструктор графических интерфейсов.

[**Cистема управления версиями**](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8)



Рисунок 14. Логотип Git

**Git** (произносится «гит») — распределённая [система управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8). Проект был создан [Линусом Торвальдсом](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D1%80%D0%B2%D0%B0%D0%BB%D1%8C%D0%B4%D1%81,_%D0%9B%D0%B8%D0%BD%D1%83%D1%81) для управления разработкой [ядра Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), первая версия выпущена [7 апреля](https://ru.wikipedia.org/wiki/7_%D0%B0%D0%BF%D1%80%D0%B5%D0%BB%D1%8F) [2005 года](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4). На сегодняшний день его поддерживает [Джунио Хамано](https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BC%D0%B0%D0%BD%D0%BE,_%D0%94%D0%B6%D1%83%D0%BD%D0%B8%D0%BE).

Среди проектов, использующих Git — [ядро Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), [Swift](https://ru.wikipedia.org/wiki/Swift_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Android](https://ru.wikipedia.org/wiki/Android), [Drupal](https://ru.wikipedia.org/wiki/Drupal), [Cairo](https://ru.wikipedia.org/wiki/Cairo), [GNU Core Utilities](https://ru.wikipedia.org/wiki/GNU_Coreutils), [Mesa](https://ru.wikipedia.org/wiki/Mesa_3D), [Wine](https://ru.wikipedia.org/wiki/Wine), [Chromium](https://ru.wikipedia.org/wiki/Chromium), [Compiz Fusion](https://ru.wikipedia.org/wiki/Compiz_Fusion), [FlightGear](https://ru.wikipedia.org/wiki/FlightGear), [jQuery](https://ru.wikipedia.org/wiki/JQuery), [PHP](https://ru.wikipedia.org/wiki/PHP), [NASM](https://ru.wikipedia.org/wiki/NASM), [MediaWiki](https://ru.wikipedia.org/wiki/MediaWiki), [DokuWiki](https://ru.wikipedia.org/wiki/DokuWiki), [Qt](https://ru.wikipedia.org/wiki/Qt), ряд дистрибутивов [Linux](https://ru.wikipedia.org/wiki/Linux).

Программа является свободной и выпущена под лицензией [GNU GPL](https://ru.wikipedia.org/wiki/GNU_General_Public_License) версии По умолчанию используется TCP порт 9418.

**Пакетный менеджер**

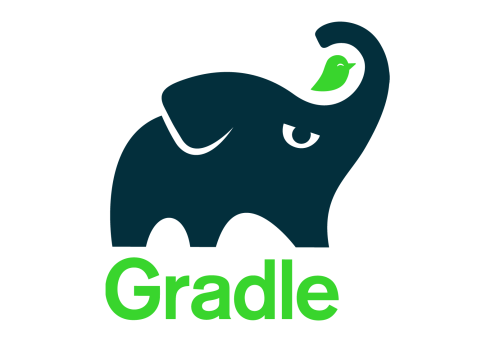


Рисунок 15. Логотип Gradle

**Gradle** — [система автоматической сборки](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%81%D0%B1%D0%BE%D1%80%D0%BA%D0%B8), построенная на принципах [Apache Ant](https://ru.wikipedia.org/wiki/Apache_Ant) и [Apache Maven](https://ru.wikipedia.org/wiki/Apache_Maven), но предоставляющая [DSL](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) на языках [Groovy](https://ru.wikipedia.org/wiki/Groovy) и [Kotlin](https://ru.wikipedia.org/wiki/Kotlin) вместо традиционной [XML](https://ru.wikipedia.org/wiki/XML)-образной формы представления конфигурации проекта.

В отличие от [Apache Maven](https://ru.wikipedia.org/wiki/Apache_Maven), основанного на концепции жизненного цикла проекта, и [Apache Ant](https://ru.wikipedia.org/wiki/Apache_Ant), в котором порядок выполнения задач (targets) определяется отношениями зависимости (depends-on), Gradle использует [направленный ациклический граф](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D1%86%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) для определения порядка выполнения задач.

Gradle был разработан для расширяемых многопроектных сборок, и поддерживает [инкрементальные сборки](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0), определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска.

Основные [плагины](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD) предназначены для разработки и развертывания [Java](https://ru.wikipedia.org/wiki/Java), [Groovy](https://ru.wikipedia.org/wiki/Groovy) и [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) приложений, но готовятся плагины и для других [языков программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

**СУБД**

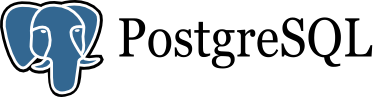


Рисунок 16. Логотип PostgreSQL

PostgreSQL - это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

**Система управления версии БД**

Файл:Liquibase Datical.png

Рисунок 17. Логотип Liquibase

Liquibase - независимая от базы данных библиотека для отслеживания, управления и применения изменений схемы базы данных. Библиотека была запущена в 2006 году, чтобы упростить отслеживание изменений в базе данных, особенно в гибкой среде разработки программного обеспечения.

**Язык программирования**



Рисунок 18. Логотип JDK 8

JDK 8(Java Development Kit 1.8) - объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения.

Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальной java-машине (JVM) — программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байт-код, в отличие от текста, обрабатывается значительно быстрее.

Язык Java зародился как часть проекта создания передового программного обеспечения для различных бытовых приборов. Реализация проекта была начата на языке [C++](http://progopedia.ru/language/c-plus-plus/), но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента — языка программирования. Стало очевидным, что необходим платформо-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами.

Язык Java потребовался для создания интерактивных продуктов для сети Internet. Фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с [C](http://progopedia.ru/language/c/) и [C++](http://progopedia.ru/language/c-plus-plus/). В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты [C++](http://progopedia.ru/language/c-plus-plus/).

Три ключевых элемента объединились в технологии языка Java

Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML

Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты

Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов

**Пакеты для разработки**

**Spring Boot**



Рисунок 19. Логотип Spring Boot

Авторы Spring решили предоставить разработчикам некоторые утилиты, которые автоматизируют процедуру настройки и ускоряют процесс создания и развертывания Spring-приложений, под общим названием **Spring Boot**

**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

**Особенности Spring Boot**

**Spring Boot** обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов

**Простота управления зависимостями**

Чтобы ускорить процесс управления зависимостями, Spring Boot неявно упаковывает необходимые сторонние зависимости для каждого типа приложения на основе Spring и предоставляет их разработчику посредством так называемых **starter**-пакетов (spring-boot-starter-web, spring-boot-starter-data-jpa и т.д.

**Starter**-пакеты представляют собой набор удобных дескрипторов зависимостей, которые можно включить в свое приложение. Это позволит получить универсальное решение для всех, связанных со Spring технологий, избавляя программиста от лишнего поиска примеров кода и загрузки из них требуемых дескрипторов зависимостей (пример таких дескрипторов и стартовых пакетов будет показан ниже)

Например, если вы хотите начать использовать Spring Data JPA для доступа к базе данных, просто включите в свой проект зависимость **spring-boot-starter-data-jpa** и все будет готово (вам не придется искать совместимые драйверы баз данных и библиотеки Hibernate)

Если вы хотите создать Spring web-приложение, просто добавьте зависимость **spring-boot-starter-web**, которая подтянет в проект все библиотеки, необходимые для разработки Spring MVC-приложений, таких как **spring-webmvc**, **jackson-json**, **validation-api** и **Tomcat**

Другими словами, **Spring Boot** собирает все общие зависимости и определяет их в одном месте, что позволяет разработчикам просто использовать их, вместо того, чтобы изобретать колесо каждый раз, когда они создают новое приложение

Следовательно, при использовании **Spring Boot**, файл **pom.xml** содержит намного меньше строк, чем при использовании его в Spring-приложениях

Обратитесь к [документации](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter), чтобы ознакомиться со всеми **Spring Boot starter-пакетами**

**Автоматическая конфигурация**

Второй превосходной возможностью **Spring Boot** является автоматическая конфигурация приложения

После выбора подходящего **starter**-пакета, **Spring Boot** попытается автоматически настроить Spring-приложение на основе добавленных вами **jar**-зависимостей

Например, если вы добавите **Spring-boot-starter-web**, Spring Boot автоматически сконфигурирует такие зарегистрированные бины, как **DispatcherServlet**, **ResourceHandlers**, **MessageSource**

Если вы используете **spring-boot-starter-jdbc**, **Spring Boot** автоматически регистрирует бины **DataSource**, **EntityManagerFactory**, **TransactionManager** и считывает информацию для подключения к базе данных из файла **application.properties**

Если вы не собираетесь использовать базу данных, и не предоставляете никаких подробных сведений о подключении в ручном режиме, Spring Boot автоматически настроит базу в памяти, без какой-либо дополнительной конфигурации с вашей стороны (при наличии H2 или HSQL библиотек)

Автоматическая конфигурация может быть полностью переопределена в любой момент с помощью пользовательских настроек

**Встроенная поддержка сервера приложений — контейнера сервлетов**  
Каждое Spring Boot web-приложение включает встроенный web-сервер. Посмотрите на [список](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#getting-started-system-requirements-servlet-containers) контейнеров сервлетов, которые поддерживаются "из коробки"

Разработчикам теперь не надо беспокоиться о настройке контейнера сервлетов и развертывании приложения на нем. Теперь приложение может запускаться само, как исполняемый jar-файл с использованием встроенного сервера

Если вам нужно использовать отдельный HTTP-сервер, для этого достаточно исключить зависимости по умолчанию. Spring Boot предоставляет отдельные starter-пакеты для разных HTTP-серверов

Создание автономных web-приложений со встроенными серверами не только удобно для разработки, но и является допустимым решением для приложений корпоративного уровня и становится все более полезно в мире микросервисов. Возможность быстро упаковать весь сервис (например, аутентификацию пользователя) в автономном и полностью развертываемом артефакте, который также предоставляет API — делает установку и развертывание приложения значительно проще

**Hibernate**



Рисунок 20. Логотип Hibernate

ORM-решением для языка [Java](http://ru.wikipedia.org/wiki/Java), является технология [Hibernate](http://ru.wikipedia.org/wiki/Hibernate_(%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0)), которая не только заботится о связи Java классов с таблицами базы данных (и типов данных Java в типы данных [SQL](http://ru.wikipedia.org/wiki/Sql)), но также предоставляет средства для автоматического построения запросов и извлечения данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL и [JDBC](http://ru.wikipedia.org/wiki/Jdbc) кода. Hibernate генерирует SQL вызовы и освобождает разработчика от ручной обработки результирующего набора данных и конвертации объектов, сохраняя приложение портируемым во все SQL базы данных.

**Front-end часть**

**HTML**



Рисунок 21. Логотип HTML

**HTML** (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *HyperText Markup Language* — «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки») — стандартизированный язык разметки документов во [Всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0). Большинство [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) содержат описание разметки на языке HTML (или [XHTML](https://ru.wikipedia.org/wiki/XHTML)). Язык HTML интерпретируется [браузерами](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80); полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Язык HTML до 5-й версии определялся как приложение [SGML](https://ru.wikipedia.org/wiki/SGML) (стандартного обобщённого языка разметки по стандарту [ISO](https://ru.wikipedia.org/wiki/ISO) 8879). Спецификации HTML5 формулируются в терминах [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model) (объектной модели документа).

Язык [XHTML](https://ru.wikipedia.org/wiki/XHTML) является более строгим вариантом HTML, он следует синтаксису [XML](https://ru.wikipedia.org/wiki/XML) и является приложением языка XML в области разметки гипертекста.

Во всемирной паутине HTML-страницы, как правило, передаются браузерам от сервера по протоколам [HTTP](https://ru.wikipedia.org/wiki/HTTP) или [HTTPS](https://ru.wikipedia.org/wiki/HTTPS), в виде простого текста или с использованием [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

**Bootstrap**



Рисунок 22. Логотип Bootstrap

**Bootstrap** (также известен как **Twitter Bootstrap**[[3]](https://ru.wikipedia.org/wiki/Bootstrap_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)#cite_note-_e02fd5ade744736e-3)[[4]](https://ru.wikipedia.org/wiki/Bootstrap_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)#cite_note-_8b1a178567366f5a-4)[[5]](https://ru.wikipedia.org/wiki/Bootstrap_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)#cite_note-_9ea25a04bd658533-5)) — свободный набор инструментов для создания [сайтов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82) и [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Включает в себя [HTML](https://ru.wikipedia.org/wiki/HTML)- и [CSS](https://ru.wikipedia.org/wiki/CSS)-шаблоны оформления для [типографики](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0), веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-расширения.

**Apache FreeMarker**



Рисунок 23. Логотип Apache FreeMarker

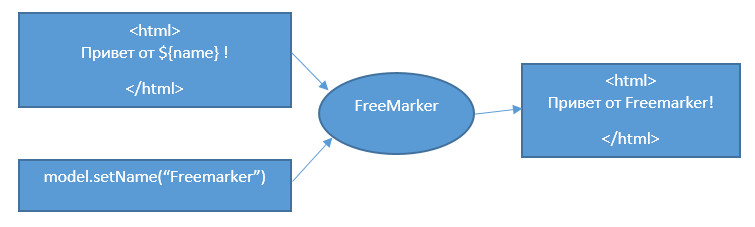
Apache FreeMarker — это механизм шаблонов: библиотека Java для генерации текстового вывода (HTML-страницы, xml, файлы конфигурации, исходный код и.т.д. На вход подается шаблон, например html в котором есть специальные выражения, подготавливаются данные соответствующие этим выражением, а Freemarker динамически вставляет эти данные и получается динамически заполненный документ.  
  


Рисунок 24. Схема взаимодействия Apache FreeMarker

## Реализация Back-End части.

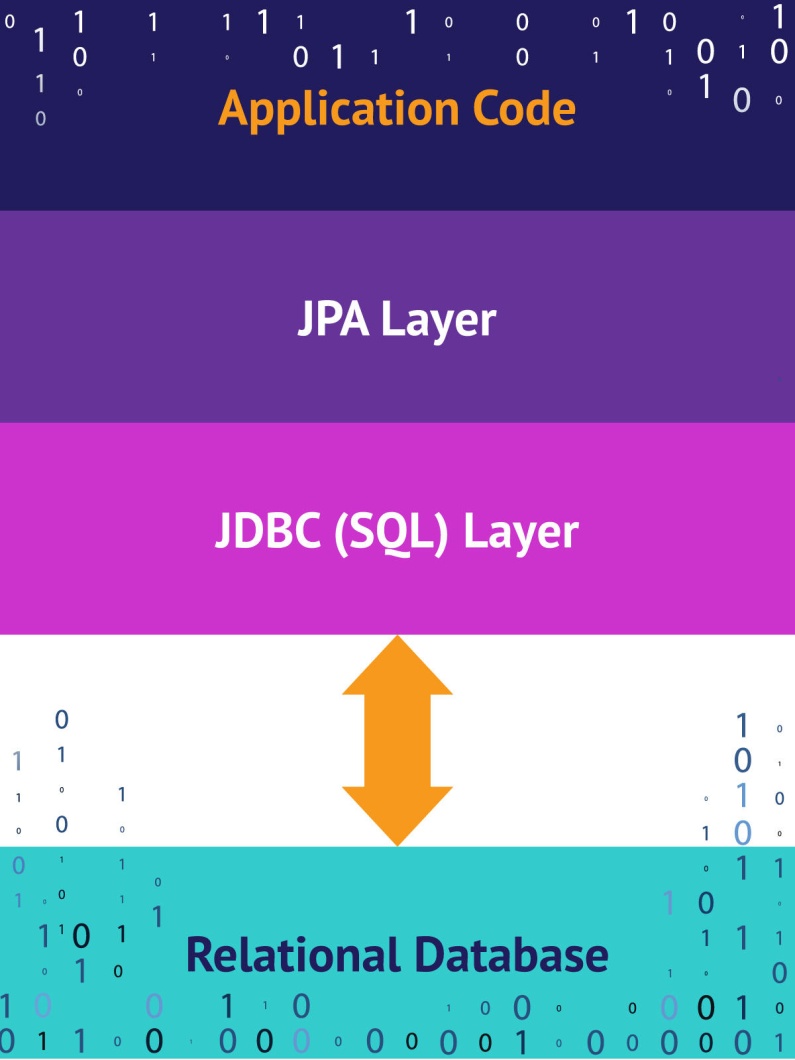


Рисунок 25. Упрощенная схема проекта

Данный проект реализован как Spring Boot проект, с использованием JPA технологии. В данном проекте используется реализация JPA – Hibernate.

**JPA** – это технология, обеспечивающая объектно-реляционное отображение простых JAVA объектов и предоставляющая API для сохранения, получения и управления такими объектами.

Серверную часть сервиса можно разделить на следующие модули:

Sql-скрипты для инициализации базы данных, а так же система контроля версии БД.

Доменные сущности для реализации представления таблиц по технологии JPA(hibertane).

DAO -репозитории для реализации API управления сущностями.

Rest-контроллеры, отвечаю за реализацию rest-запросов к сервису, а так же хранят в себе бизнес-логику.

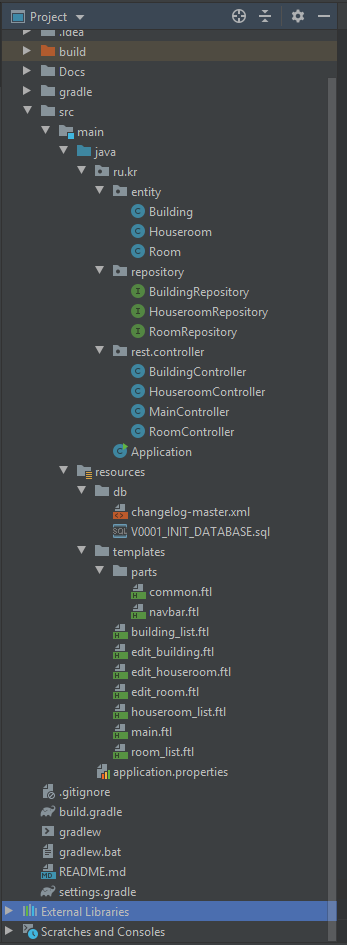


Рисунок 26. Структура проекта

**Sql-скрипты для инициализации базы данных, а так же система контроля версии БД.**

В папке resources/db лежит все необходимое для правильной работоспособности Liquibase, а так же корректного наката скриптов.

V0001\_INIT\_DATABASE.sql – файл инициализации БД.

changelog-master.xml – файл Liquibase с перечнем sql-скриптов для наката.

В реализация данного модуля в большей степени задействован пакет Spring Data JPA.

Spring Data — дополнительный удобный механизм для взаимодействия с сущностями базы данных, организации их в репозитории, извлечение данных, изменение, в каких то случаях для этого будет достаточно объявить интерфейс и метод в нем, без имплементации.

**Доменные сущности для реализации представления таблиц по технологии JPA и DAO -репозитории для реализации API управления сущностями.**

Все сущности находятся в пакете ru.kr.entity. Репозитории же лежат в ru.kr.repository.

При проектировании информационной системы выявляются некоторые слои, которые отвечают за взаимодействие различных модулей системы. Соединение с базой данных является одной из важнейшей составляющей приложения. Всегда выделяется часть кода, модуль, отвечающающий за передачу запросов в БД и обработку полученных от неё ответов. В общем случае, определение [Data Access Object](https://ru.wikipedia.org/wiki/Data_Access_Object)описывает его как прослойку между БД и системой. **DAO** абстрагирует сущности системы и делает их отображение на БД, определяет общие методы использования соединения, его получение, закрытие и (или) возвращение в **Connection Pool**.

Для создания доменных сущностей используется hibernate annotations.

Аннотации являются мощным иструментом  для предоставления метаданных, а также намного нагляднее при чтении нашего кода другим разработчиком

.

Обязательными аннотациями являются следующие:

**@Entity**

Эта аннотация указывает Hibernate, что данный класс является сущностью (entity bean). Такой класс должен иметь конструктор по-умолчанию (пустой конструктор).

**@Table**

С помощью этой аннотации мы говорим Hibernate,  с какой именно таблицей необходимо связать (map) данный класс. Аннотация **@Table**имеет различные аттрибуты, с помощью которых мы можем указать *имя таблицы, каталог, БД и уникальность столбцов в таблец БД*.

**@Id**

С помощью аннотации **@Id**мы указываем *первичный ключ (Primary Key)* данного класса.

**@GeneratedValue**

Эта аннотация используется вместе с аннотацией **@Id** и определяет такие паметры, как **strategy** и **generator**.

**@Column**

Аннотация **@Column**определяет к какому столбцу в таблице БД относится конкретное поле класса (аттрибут класса).

Наиболее часто используемые аттрибуты аннотации **@Column**такие:

* **name**Указывает имя столбца в таблице
* **unique**Определяет, должно ли быть данноезначение уникальным
* **nullable**Определяет, может ли данное поле быть NULL, или нет.
* **length**Указывает, какой размер столбца (например колчиство символов, при использовании String).

DAO -репозитории для реализации API управления сущностями реализованы через интерфейс Spring Repository.

Основное понятие в Spring Data — это репозиторий. Это несколько интерфейсов которые используют JPA Entity для взаимодействия с ней. Так например интерфейс  
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID>  
обеспечивает основные операции по поиску, сохранения, удалению данных (CRUD операции)

T save(T entity);

Optional findById(ID primaryKey);

void delete(T entity);

и др. операции.  
  
Есть и другие абстракции, например PagingAndSortingRepository.  
Т.е. если того перечня что предоставляет интерфейс достаточно для взаимодействия с сущностью, то можно прямо расширить базовый интерфейс для своей сущности, дополнить его своими методами запросов и выполнять операции. Сейчас я покажу коротко те шаги что нужны для самого простого случая (не отвлекаясь пока на конфигурации, ORM, базу данных).

Запросы к сущности можно строить прямо из имени метода. Для этого используется механизм префиксов find…By, read…By, query…By, count…By, и get…By, далее от префикса метода начинает разбор остальной части. Вводное предложение может содержать дополнительные выражения, например, Distinct. Далее первый By действует как разделитель, чтобы указать начало фактических критериев. Можно определить условия для свойств сущностей и объединить их с помощью And и Or.

В документации определен весь перечень, и правила написания метода. В качестве результата могут быть сущность T, Optional, List, Stream. В среде разработки, например в Idea, есть подсказка для написания методов запросов.

**Rest-контроллеры.**

Все котроллеры расположены в пакете ru.kr.rest.controllers.

Разделены они по принципу - для обработки каждой сущности свой контроллер.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.  
В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

REST быстро стал де-факто стандартом для создания web-сервисов, поскольку таким они легки для создания и обработки.

Существует большое обсуждение о том, как REST вписывается в мир микросервисов, но в этом уроке просто описывается создание RESTful сервисов.

Почему REST? Из [REST на практике](https://www.amazon.com/gp/product/0596805829?ie=UTF8&tag=martinfowlerc-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0596805829), заимствуется фраза [Martin Flower](https://martinfowler.com/): "Понятие того, что web является доказательством очень масштабируемой распределенной системы, которая работает действительно хорошо и мы можем брать идеи, исходя из того, чтобы создавать системы более легко." Я думаю, что это довольно веская причина: REST охватывает указания самого web, его архитектуры и его приемуществ.

Какие преимущества? Главным образом все те, какие несет в себе HTTP как платформа сама по себе. Безопасность приложений(шифрование и аутентификация), для которой известно некоторое количество решений. Кеширование встроено в протокол. Маршрутизация, через DNS, гибкая и хорошо известная система уже поддерживается повсеместно.

REST, несмотря на повсеместность использования, не является стандартом, как таковой, а подходом, стилем, ограниечением HTTP протокола. Его реализация может различаться в стиле, подходе. Для клиента такого API это может быть разочарованием. Качество REST сервисов дико варьируется.

Dr. Leonard Richardson собрал воедино модель, которая объясняет различные уровни соответствующих понятий REST и сортирует их. Она описывает 4 уровня, начиная с **level 0**. Martin Fowler [очень хорошо написал про модель](https://martinfowler.com/articles/richardsonMaturityModel.html)

**Level 0**: Swamp of POX - это уровень, где мы просто используем HTTP как транспорт. Вы можете вызвать SOAP технологию. Она использует HTTP, но как транспорт. Стоит также отметить, что вы можете также использовать SOAP [поверх чего-то подобного JMS](https://www.w3.org/TR/soapjms/) без HTTP. SOAP, соответветственно не является RESTful. Он всего лишь HTTP-aware.

**Level 1**: Resources - на этом уровне сервисы могут использовать HTTP URI для отличия между сущностями в системе. К примеру, вы можете направить запросы на /customers, /users и т.д. XML-RPC является примером **Level 1** технологии: он использует HTTP и он может использовать URI для различия точек выхода. В конечном счете, несмотря на то, что XML-RPC не является RESTful, он использует HTTP как транспорт для чего-нибудь ещё(удаленный вызов процедур).

**Level 2**: HTTP Verbs - это уровень, на котором вы хотите быть. Если вы делаете все неправильно с Spring MVC, то вы скорее всего все ещё остановитесь тут. На этом уровне сервисы используют преимущества нативных HTTP возможностей, таким как заголовки, коды статуса, определенные URI и другие. Отсюда мы начнем наше путешествие.

**Level 3**: Hypermedia Controls - это заключительный уровень, к которому мы стремимся. Гипермедиа как практическое применение [HATEOAS](https://en.wikipedia.org/wiki/HATEOAS) ("HATEOAS" является сокращением от "Hypermedia as the Engine of Application State") шаблона проектирования. Гипермедиа продлевает жизнь сервису, отделяя клиента сервиса от необходимости глубокого знания платформы и топологии сервиса. Она описывает REST сервисы. Сервис может ответить на вопросо о том, какой выл вызов и когда. Мы рассмотрим более глубже в дальнейшем.

Для маршрутизации запросов в Spring Framework используется аннотация **@RequestMapping** с указанием HTTP-метода при помощи свойства method или более простые аннотации вроде  
**@GetMapping**, **@PostMapping**, **@DeleteMapping**и т.д.

## Реализация Front-End части.

В реализации ui учавствуют следующие элементы:

* Freemarker – шаблонизатор, для наполнения html-заготовок данными.
* Bootstrap – подключаемая библиотека с готовым набором решений для реализации пользовательского интерфейса.
* HTML-файлы, заготовки с разметкой freemarker и подключенной библиотекой Bootstrap .

Все файлы шаблонов лежат в папке resources/templates.

**Верхняя панель навигации**

Файл navbar.ftl

Реализован переход на главную страницу, просмотра списка домов, просмотра списка квартир, просмотра списка комнат.

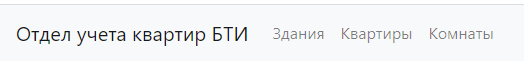


Рисунок 27. Панель навигации

**Страница главного экрана сервиса**

Файл main.ftl

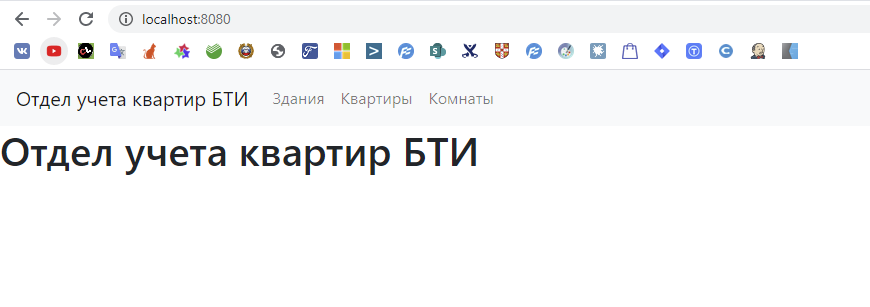


Рисунок 28. Главный экран

**Страница просмотра всех зданий зарегистрированных в системе.**

Файл building\_list.ftl

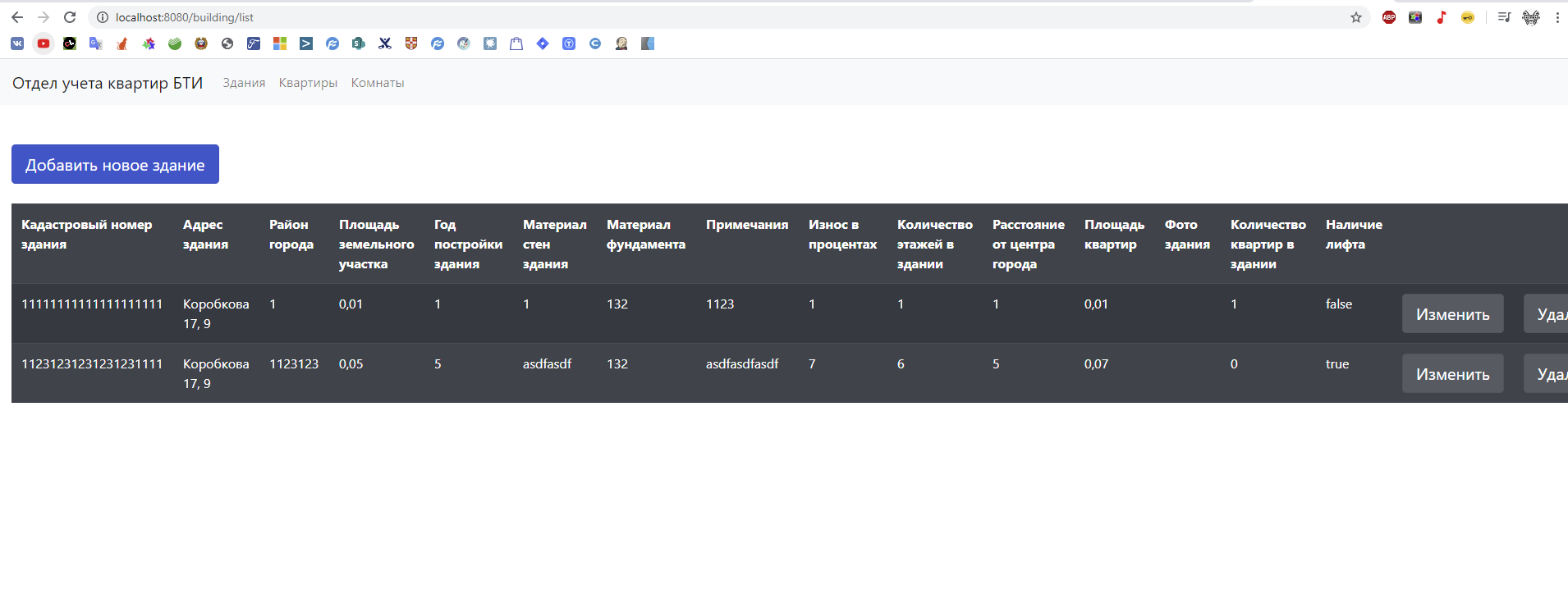


Рисунок 29. Страница просмотра всех зданий зарегистрированных в системе

**Страница детального просмотр и редактирования зданий.**

Файл edit\_building.ftl

Имеет два режима: заведения нового здания или редактирование существующего.

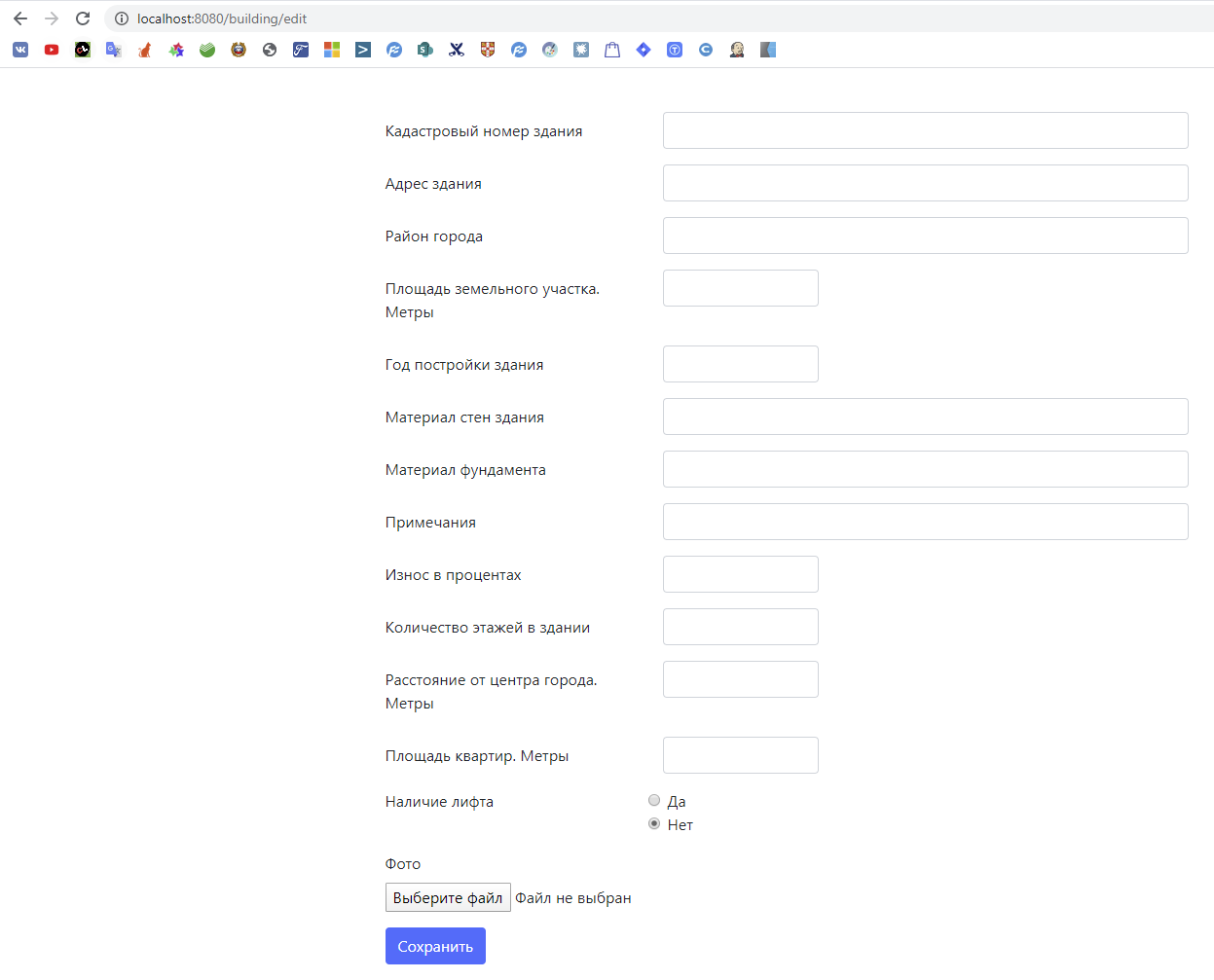


Рисунок 30. Страница добавления нового здания

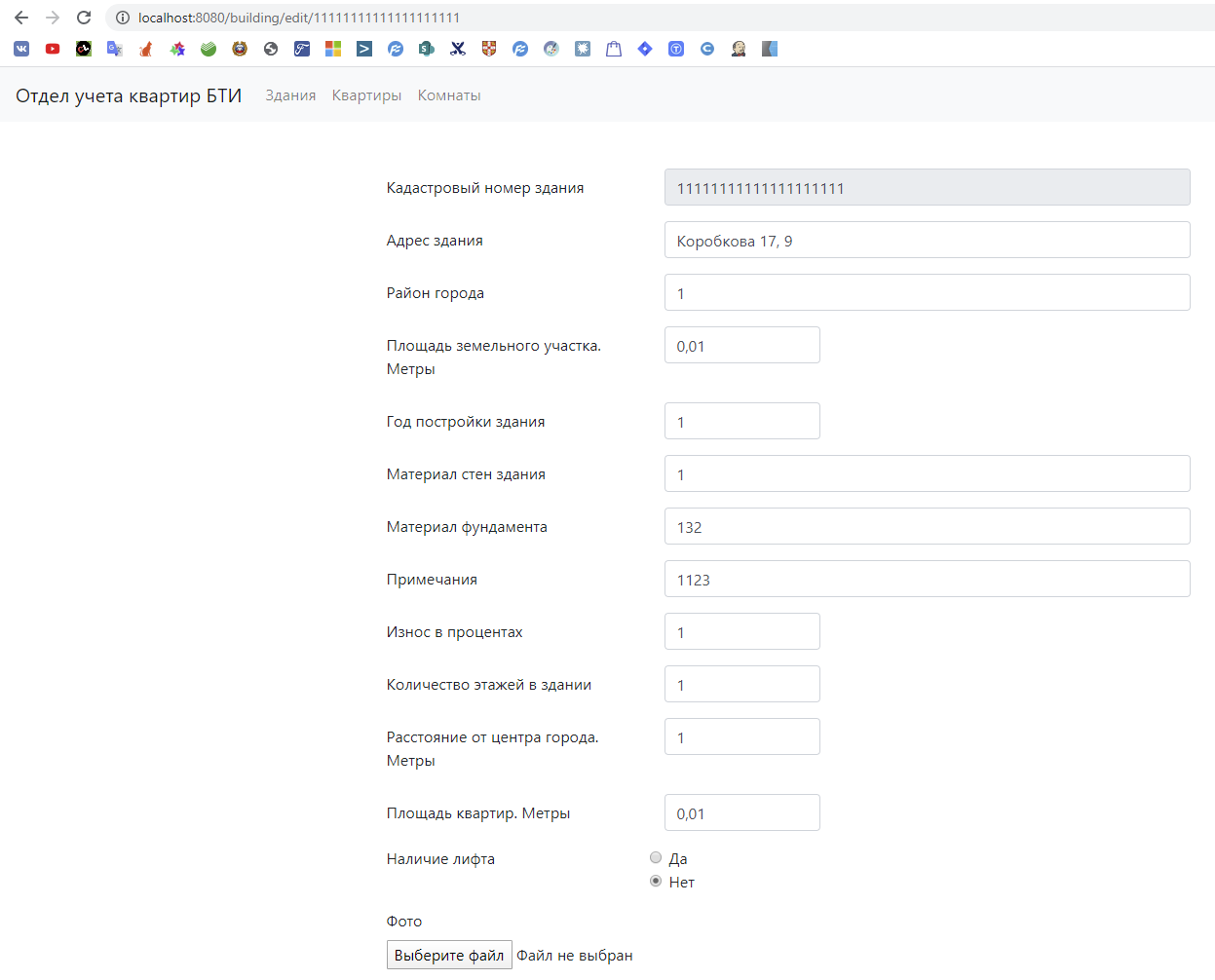


Рисунок 31. Страница редактирования существующего здания

**Страница просмотра всех квартир зарегистрированных в системе.**

Файл houseroom\_list.ftl

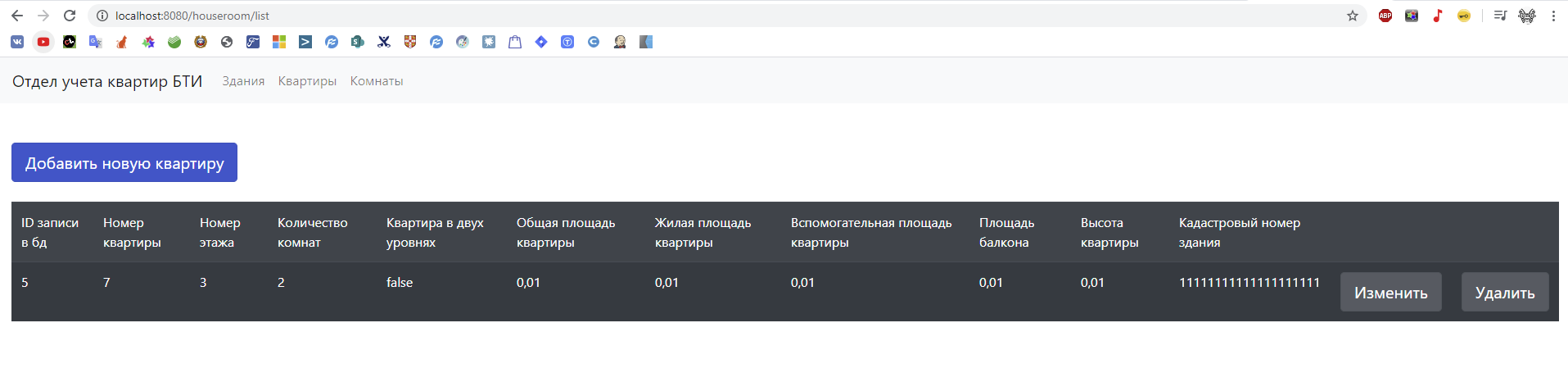


Рисунок 32. Страница просмотра всех квартир зарегистрированных в системе

**Страница детального просмотр и редактирования квартир.**

Файл edit\_houseroom.ftl

Имеет два режима: заведения новой квартиры или редактирование существующей.

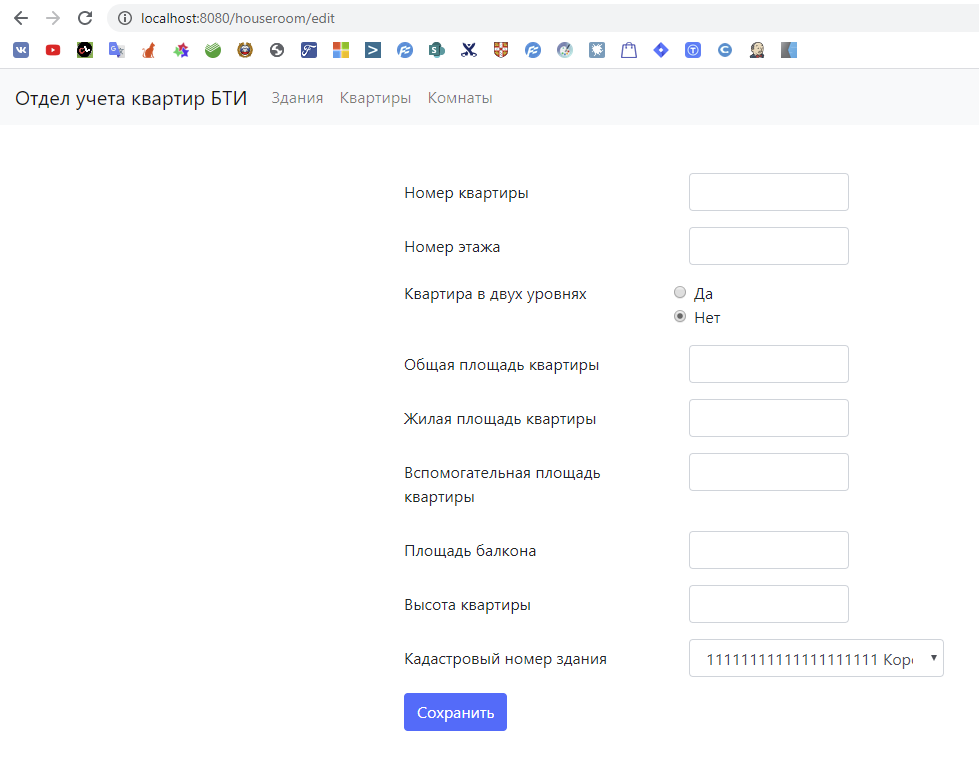


Рисунок 33. Страница добавления новой квартиры

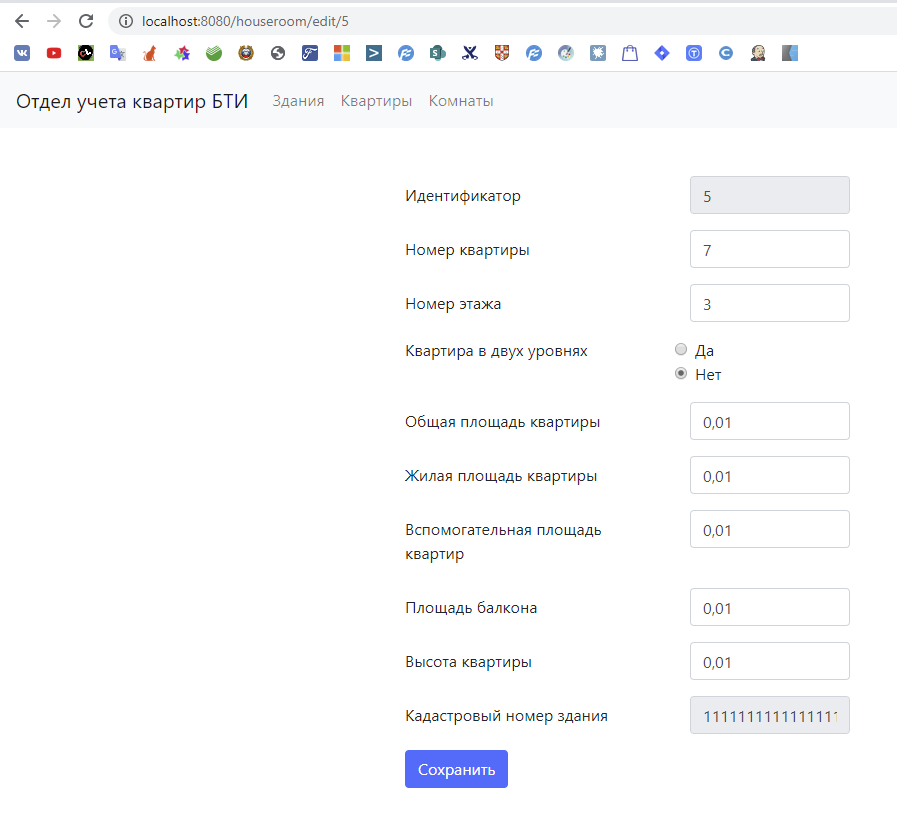


Рисунок 34. Страница редактирования существующей квартиры

**Страница просмотра всех комнат зарегистрированных в системе.**

Файл room\_list.ftl

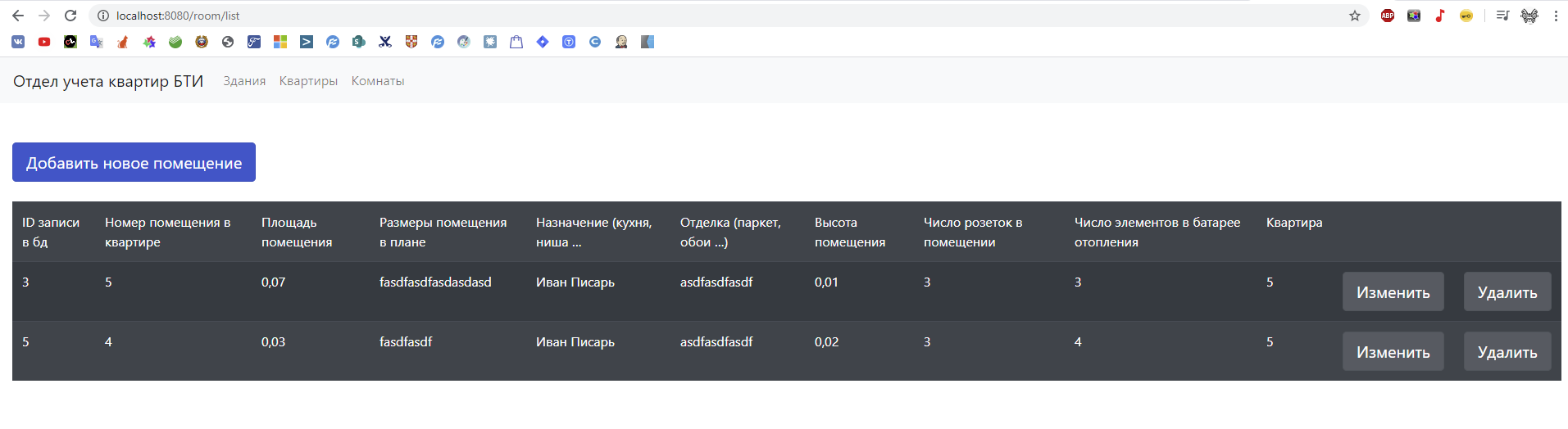


Рисунок 35. Страница просмотра всех комнат зарегистрированных в системе

**Страница детального просмотр и редактирования комнат.**

Файл edit\_houseroom.ftl

Имеет два режима: заведения новой комнаты или редактирование существующей.

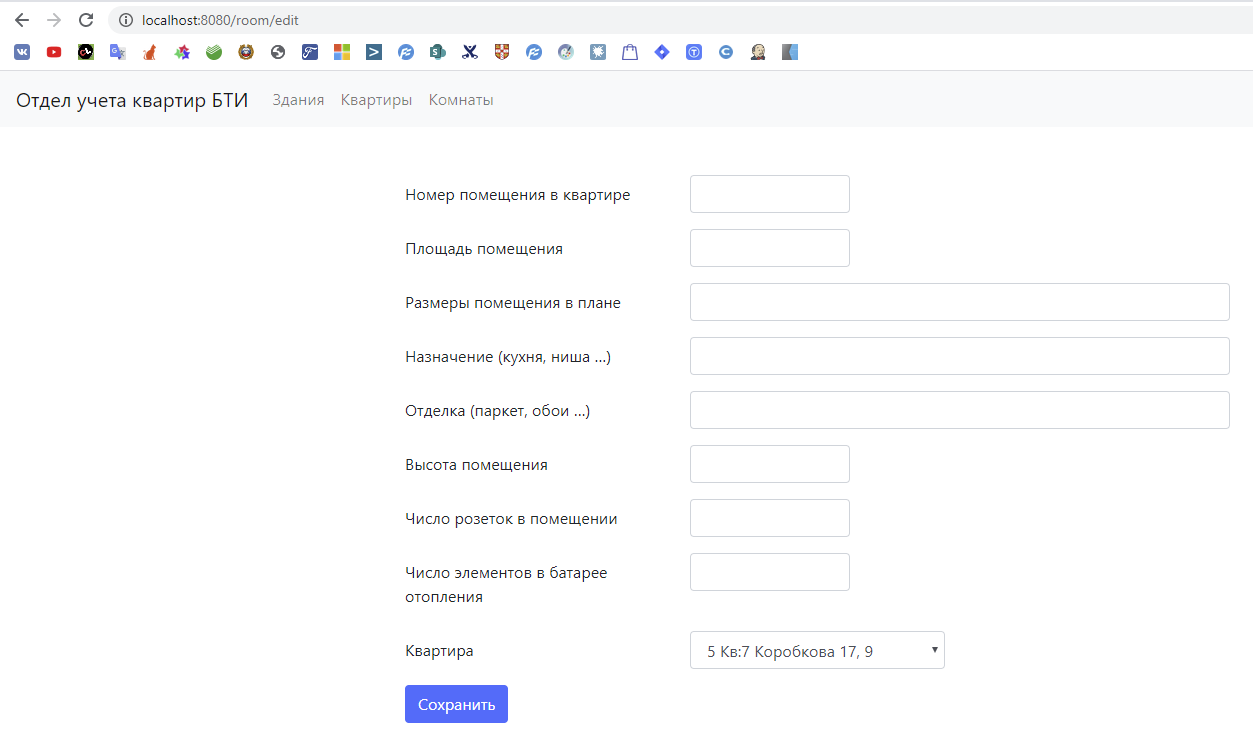


Рисунок 36. Страница добавления новой комнаты

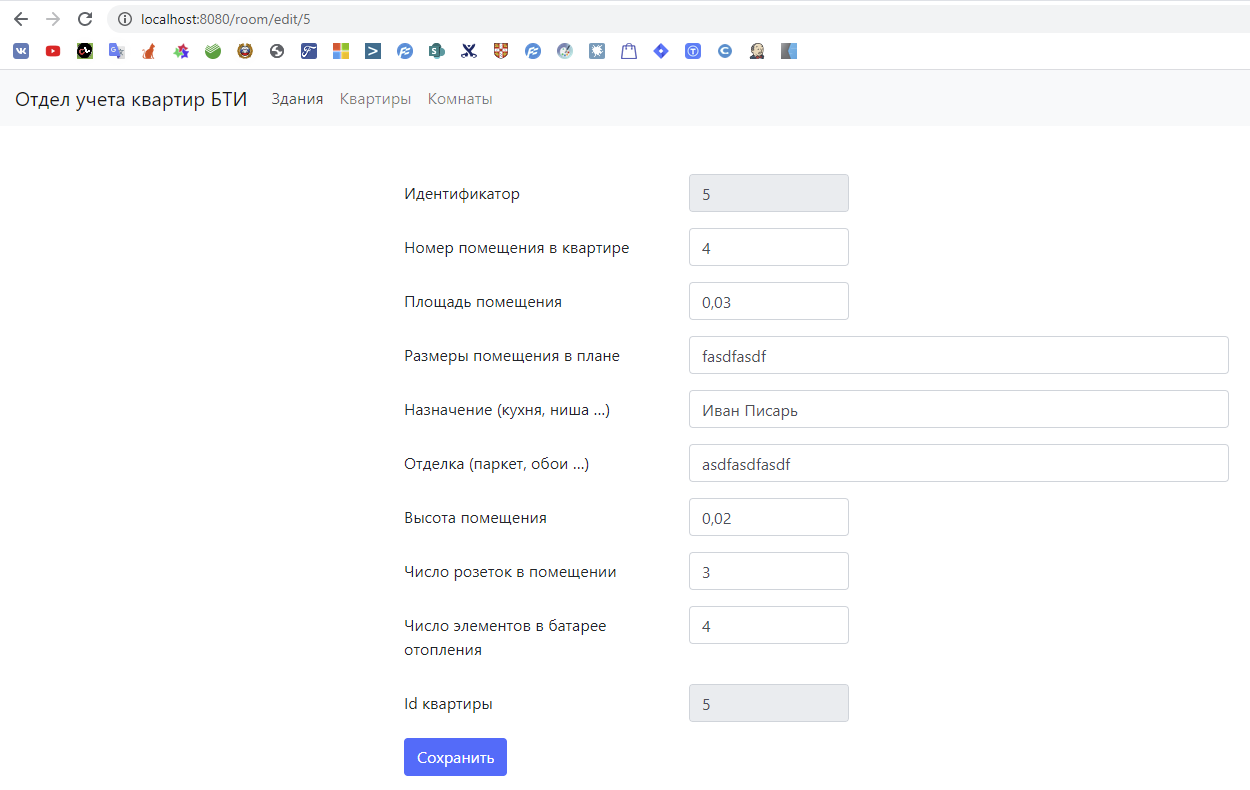


Рисунок 37. Страница редактирования существующей комнаты

# 

# Приложение А

Техническое Задание