



Geekbrains

Факультет Разработчик — Программист

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА

Разработчик — Веб-разработка на Java

Наименование специальности

на тему:

Разработка веб-приложения для удобства наладки

вентиляционных систем по воздуху.

Обучающийся

Пуговкин Владислав Андреевич

Фамилия Имя Отчество

Москва 2025 г.

СОДЕРЖАНИЕ

| | |
|---|----|
| АННОТАЦИЯ | 4 |
| ВВЕДЕНИЕ..... | 5 |
| 1. Общие сведения о наладке вентиляционных систем | 6 |
| 1.1 История развития систем вентиляции | 6 |
| 1.2 Основные сведения о вентиляционных системах | 8 |
| 1.3 Теоретические данные для наладки вентиляционных систем | 19 |
| 1.4 Выводы по разделу | 26 |
| 2. Выбор основных технологий и сущностей для веб-приложения | 28 |
| 2.1 Что такое веб-приложение и его особенности..... | 28 |
| 2.2 Архитектура веб-приложения | 29 |
| 2.3 Выбор технологий для веб-приложения | 35 |
| 2.4 Выделение основных сущностей и их атрибутов..... | 41 |
| 2.5 Выводы по разделу | 43 |
| 3. Разработка приложения для наладки вентиляционных систем | 46 |
| 3.1 Инициализация и создание структуры приложения | 46 |
| 3.2 Написание классов приложения..... | 48 |
| 3.3 Составление и написание шаблонов HTML страниц..... | 56 |
| 3.4 Подготовка и вывод приложения для работы в контейнере | 63 |
| ЗАКЛЮЧЕНИЕ | 67 |
| БИБЛИОГРАФИЧЕСКИЙ СПИСОК | 69 |
| ПРИЛОЖЕНИЕ А | 70 |
| ПРИЛОЖЕНИЕ Б..... | 71 |
| ПРИЛОЖЕНИЕ В | 72 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В | 73 |

| | |
|--------------------------------|----|
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В | 74 |
| ПРИЛОЖЕНИЕ Г | 75 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г | 76 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г | 77 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г | 78 |
| ПРИЛОЖЕНИЕ Д | 79 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Д | 80 |
| ПРИЛОЖЕНИЕ Е | 81 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е | 82 |
| ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е | 83 |

АННОТАЦИЯ

Рассмотрены основные сведения о вентиляционных системах и способах их наладки. Выбраны технологии, которые используются для создания веб-приложения по наладке вентиляционных систем. Подобраны основные сущности, используемые для веб-приложения. Написан код основных и вспомогательных классов для работы приложения. Приложение выведено в автономную полноценную работу в контейнере. Объем выпускной квалификационной работы составляет 83 листов, из них список использованных источников – 1 листа. ВКР включает в себя 45 рисунков, 33 листинга и 6 приложений.

Ключевые слова:

КПД – коэффициент полезного действия

ПО – программное обеспечение

ОЗУ – оперативное запоминающее устройство

ООП – объектно-ориентированное программирование

СУБД – система управления базы данных

БД – база данных

ВВЕДЕНИЕ

В данный момент в сфере вентиляционного оборудования, а именно в пуско-наладочных работах, большинство инженеров используют большое количество бумажной продукции для записи, расчетов и фиксирования результатов работы. Когда время работы в данной сфере достаточно велико, то бумажной продукции в виде тетрадей, листов и похожей канцелярии становится настолько много, что физически не удобно хранить, искать и брать с собой в работу все эти материалы.

Целью данной итоговой аттестационной работы является разработка веб-приложения для удобства наладки вентиляционного оборудования. Соответственно данное приложение будет решать проблему складирования, поиска и переноски необходимых для работы материалов.

Для достижения цели в ходе работы предстоит решить следующие задачи:

- Рассмотреть основные сведения о вентиляционных системах и способах их наладки.
- Выбор технологий для создания веб-приложения по наладке вентиляционных систем.
- Выбор и описание основных сущностей и их атрибутов для дальнейшей разработки приложения.
- Написание кода для основных и вспомогательных классов программы.
- Написание шаблонов HTML страниц.
- Вывод приложения для полноценной самостоятельной работы в контейнере.

По итогам работы будет написано веб-приложение, которое можно будет развернуть в контейнере и соответственно закрыть потребность в бумажных носителях для наладки вентиляционных систем.

1. Общие сведения о наладке вентиляционных систем

1.1 История развития систем вентиляции

Для обеспечения комфортной жизни в помещении, помимо соблюдения температурных режимов, также необходим обмен воздуха. Для решения данной задачи есть два подходящих варианта.

Первый вариант – это естественная вентиляция, которая была открыта еще в древности. Она представляет собой специально оборудованные отверстия, или же не специально оставленные щели в окнах, дверях, за счет которых воздух циркулирует в помещении. Загрязненный воздух за счет физики поднимается вверх, так как более теплый воздух является более легким, где его ожидает отверстие под вытяжку. Так же это может происходить за счет ветра. В нижней части комнаты делается отверстие под приток свежего воздуха и, соответственно новый свежий воздух замещает собой загрязненный.

Таким образом естественной вентиляцией были оснащены старинные постройки, одним из таких строений являются пирамиды Хеопса. Во время постройки были учтены вентиляционные каналы размерами около 20 × 20 см и около 60м протяженностью. Также, например, в средние века вентиляционным отверстием служил воздуховод для камина, который выводил загрязненный воздух. Приточным отверстием являлось то, что двери и окна были сделаны не герметично и через щели поступал свежий воздух.

Вторым вариантом поддержать эффективный и постоянный обмен воздуха в различных помещениях – это принудительная вентиляция. Такой способ был использован уже в более близкие к нам века, а именно в 18 веке появились первые вентиляционные установки, также называемые машинами принудительного перемещения воздуха. Теоретические основы вентилятора, которые впоследствии стали базой для методик расчета современных систем механической вентиляции, были разработаны в 1754 году членом Петербургской академии наук Леонардом Эйлером.

Следующим этапом развития принудительной вентиляции является изобретение центробежного вентилятора. А. А. Саблуков – генерал-лейтенант горных инженеров, который является изобретателем этого вентилятора, так же предложил передовую на тот момент времени методику расчета таких вентиляторов. Данный вентилятор был впервые установлен и протестирован на сахарном и кожевенном заводах и в последующем нашли широкое распространение по России и даже за границей.



Рисунок 1.1 – Центробежный вентилятор

Далее, в 1892 году, появляется новый тип вентилятора – диаметральный. Данный тип вентилятора на некоторое время получил применение в шахтах, но позже его заменили центробежными вентиляторами за счет более высокого КПД.

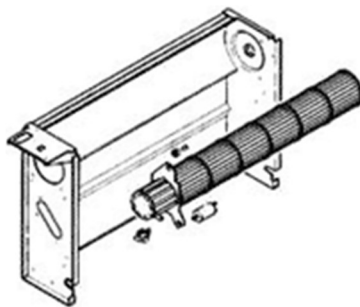


Рисунок 1.2 – Диаметральный вентилятор

Интерес к данному типу вентиляторов был на некоторое время утерян, но позже в Центральном аэрогидродинамическом институте (ЦАГИ) им. Н. Е. Жуковского начали проводить большую работу по улучшению и созданию совершенных конструкций вентилятора. В ходе данного исследования были выведены физически обоснованные теории о работе осевых и центробежных вентиляторов, что позволило сконструировать наиболее превосходящие машины, чем достигнутые в то время за рубежом.

Так же одним из подвидов центробежного вентилятора является тип с загнутыми назад лопатками, что позволило добиться более высокого КПД около 85%.

В настоящее время достаточно широко используется еще один вид вентиляторов, называемый канальными. Это подразумевает, что центробежный вентилятор встраивается в вентиляционный канал системы, не имея внешнего кожуха. Такие вентиляторы получили высокое применение за счет своей высокой эффективности и удобства применения.

1.2 Основные сведения о вентиляционных системах

Как уже было выяснено, основной функцией вентиляционных систем является обмен загрязненного или отработанного воздуха на свежий. В настоящее время это достигается путем комплекса технических устройств и мероприятий. Кроме основной функции существует еще ряд полезных свойств.

Одним из таких свойств является поддержание комфортных условий в помещениях. За счет вентиляционного оборудования системы вентиляции способны влиять на температуру и влажность воздуха в помещениях, что позволяет установить и поддерживать определенный комфортные микроклимат для любого вида задач.

Также определенно полезной функцией вентиляционных систем можно считать сохранение помещений от повреждений. При недостаточной вентиляции уровень влажности повышается, что способствует повреждению различных материалов и, несомненно, приводит к развитию плесени и грибковой инфекции. Все эти проблемы можно устранить путем подбора, монтажа и грамотной настройки систем вентиляции для сохранения постоянного обмена воздуха в помещениях и контроля уровня влажности.

На данный момент, различают несколько типов вентиляционных систем. Соответственно, естественные и принудительные типы вентиляции были рассмотрены ранее, поэтому начнем с приточно-вытяжной вентиляции. Такая

вентиляционная система может совмещать в себе принципы естественной и принудительной вентиляции, но при этом может быть и полностью принудительной. В первом варианте это означает, что для подачи свежего воздуха в помещение (приток воздуха), используются специальные воздухозаборники, а для выброса отработанного или загрязненного воздуха используют принудительную вентиляцию, а именно вытяжные вентиляторы. Во втором варианте подача и выброс воздуха осуществляется системами вентиляции, которые должны быть правильно подобраны для качественного использования.

Так же одним из типов вентиляции можно отметить рекуперационный вариант. В такой системе вентиляции отработанный воздух, который выбрасывается из помещения, не перестает быть полезным, так как во время выброса передает свое тепло входящему воздуху. За счет этого вентиляционная система становится более энергоэффективной, тем самым уменьшаются энергозатраты на отопление и кондиционирование воздуха.

Все эти типы вентиляции различаются за счет использования различного вентиляционного оборудования. Вентиляционное оборудование представляет собой набор из нескольких компонентов специализированных устройств и систем.

Одним из важных основных компонентов являются вентиляторы различных типов, которые соответственно создают поток воздуха. Так же важным компонентом являются воздуховоды, вентиляционные каналы и решетки. Они образуют собой сеть, которая протекает по необходимым помещениям, иногда даже по разным этажам, и осуществляет равномерное распределение воздуха. Без данных компонентов невозможно представить полноценную принудительную систему вентиляции.

Кроме важных составляющих вентиляционных систем, существуют компоненты, которые влияют на качество воздуха, но, соответственно, не являются обязательными. К таким компонентам можно отнести фильтры. Они бывают различной степени фильтрации и способствуют очищению

подаваемого воздуха от пыли, бактерий, пылицы и других вредных частиц. С их помощью в помещении достигается необходимого качества воздух. В некоторых частных случаях фильтры устанавливаются и на удаляемый воздух, если того требует проектное решение в связи, например, с запретом на загрязнение воздуха вредными веществами.

В зависимости от варианта исполнения вентиляционные установки делятся на две основные категории: наборные и моноблочные. Наборные установки представляют собой систему отдельных компонентов, которые необходимо собрать вместе. В зависимости от требований и характеристик помещений необходимо подбирать различные конфигурации компонентов, при чем для оптимальной работы данных систем необходимо правильно выстраивать последовательность отдельных компонентов. Для таких систем требуется тщательное проектирование, иначе не получится добиться желаемой эффективности системы.

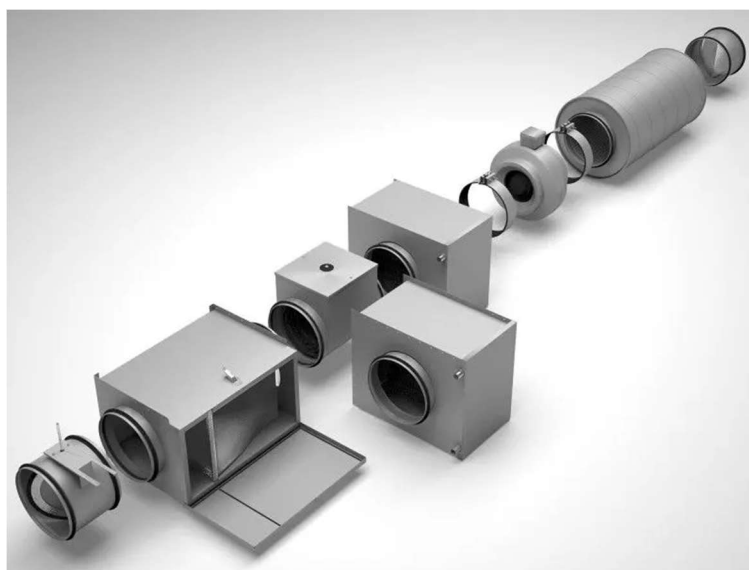


Рисунок 1.3 – Наборная установка

Моноблочные установки на данный момент являются наиболее распространенным выбором для промышленных объектов. Такие системы являются более удобными и простыми в обслуживании, за счет чего снижаются заботы и расходы предприятия.



Рисунок 1.4 – Моноблочная установка

В зависимости от конкретных требований и характеристик помещений выбор может пасть как на моноблочные, так и на наборные установки. Если необходимо иметь более чувствительную возможность настройки, то в таком формате поможет наборная установка, которую могут разработать под любые необходимые условия. Если необходима простота и надежность, то вариант с моноблочной установкой самый подходящий.

Во всех вариантах вентиляционных систем, неотъемлемой частью являются вентиляторы. Выделим некоторые виды вентиляторов, в зависимости от их модификаций. Выше уже были рассмотрены такие типы вентиляторов как центробежные, которые являются достаточно распространенными и обладают высокой эффективностью.

Так же существует такой тип вентилятора как осевой. В данном типе вентилятора воздух перемещается вдоль оси вращения лопастей. Осевые вентиляторы в основном применяются, когда необходимо перемещать большие объемы воздуха при низком давлении. Чаще всего данный вариант применяется в промышленности.



Рисунок 1.5 – Осевой вентилятор

В комбинации двух типов, представленных выше, появляются диагональные вентиляторы. Такой тип вентиляторов позволяет совместить в себе высокую производительность и отличную эффективность, что позволяет обеспечивать постоянный и качественный поток воздуха для больших помещений.



Рисунок 1.6 – Диагональный вентилятор

Следующим компонентом, который позволяет вентиляционной системе оптимизировать процесс обмена влаги и тепла в помещении, является рекуператор. Он представляет собой изолированную друг от друга систему, двустенный теплообменник, в котором два потока воздуха различной температуры могут передавать тепло, а также удалить избыточную влагу.

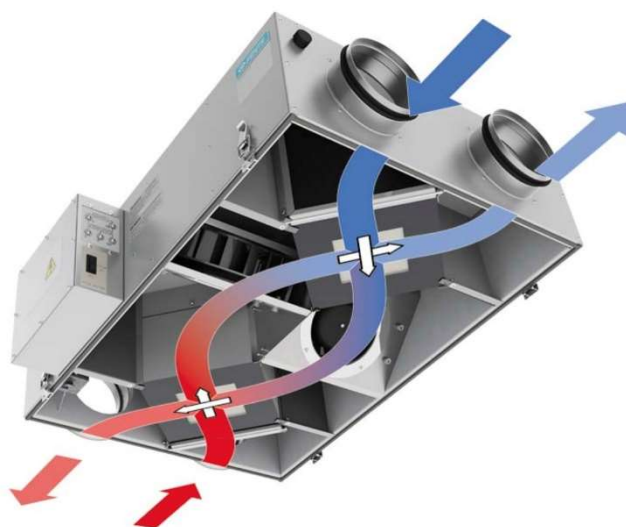


Рисунок 1.7 – Рекуператор

Рекуператор работает по принципу использования теплоотдачи вытяжного воздуха и передачи его свежему воздуху. Это позволяет системе быть более энергоэффективной и минимизировать потери полезного тепла. Также в рекуператоре может быть оборудована система рекуперации влаги, что позволяет поддерживать определенный микроклимат в помещении за счет удаления избыточной влаги из воздуха. Помимо этого, рекуператор может быть оснащен фильтрами, что позволяет довести до помещения более чистый и полезный воздух.

Еще одним полезным и часто устанавливаемым компонентом является фильтр. Этот компонент является достаточно важным, так как от него зависит какой воздух будет поступать в помещение. Фильтр выполняет задачу улавливания и задержания вредных веществ, излишков тепла, влаги и различных запахов, которые могут исходить от оборудования. В зависимости от места установки, фильтр может выполнять различные задачи. Место установки зависит соответственно от потребностей и условий.



Рисунок 1.8 – Фильтр карманный

Часто в систему устанавливают так называемый предварительный фильтр. Его задача предотвратить попадания в систему нежелательных предметов, таких как грязь, листья, крупные частицы пыли и других, за счет чего увеличивается срок работы оборудования.

На более поздних этапах также возможна установка фильтров, в таком случае его задачей будет уже задерживание более мелких частиц пыли,

аллергенов и других различных вредных веществ. В целом, большинство систем не обходится без фильтров, так как они способствуют улучшению качества воздуха и так же продлевают сроки службы самой системы, что является достаточно значимым преимуществом.

Для того чтобы вентиляционная система могла поддерживать постоянную определенную температуру в помещении при любых внешних погодных условиях, есть еще один компонент вентиляционного оборудования, называемый нагревателем. Данный компонент предназначен для нагрева входящего воздуха до определенной требуемой температуры, позволяя достичь комфортного климата в помещении. В основном, канальные нагреватели делятся на два основных типа: водяные и электрические.

В основе работы водяного нагревателя лежит принцип теплообмена с помощью теплоносителя в виде воды. Внутри устройства располагаются некие каналы или же трубки, по которым циркулирует горячая вода. Воздух, который проходит сквозь теплоноситель, подогревается до определенной температуры и попадает в помещение, создавая комфортный климат. Для такого типа нагревателя к вентиляционному оборудованию необходимо будет спроектировать и произвести монтаж труб с горячей водой.



Рисунок 1.9 – Водяной нагреватель

В свою очередь электрический канальный нагреватель работает за счет электрической энергии. Внутри устройства такого типа располагаются нагревательные элементы, называемые тэнами, которые производят тепло за

счет электрической энергии. Такой тип нагревателя требует для своей работы только необходимое электрическое питание.

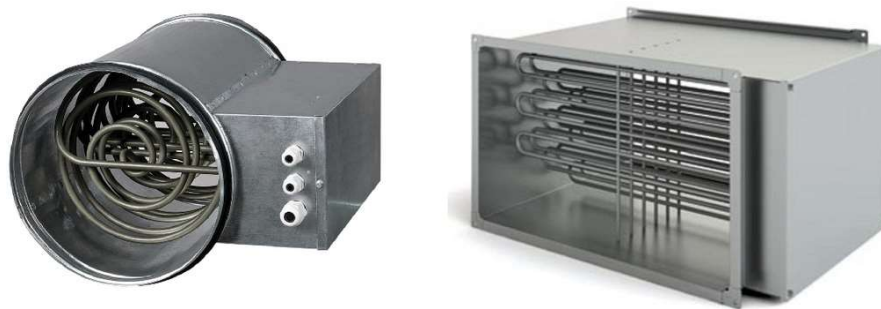


Рисунок 1.10 – Электрический нагреватель

Выбор подходящего типа нагревателя зависит от конкретных требований и условий эксплуатации на объекте. Возможна такая ситуация, что в помещении, где будет располагаться вентиляционная система, может отсутствовать доступ к горячей воде и, соответственно, в таком случае выбор не зависит от эффективности устройств, а от конкретных условий, и в таком варианте необходимо будет выбрать электрический нагреватель.

Также одним из компонентов, позволяющим поддерживать комфортный микроклимат, является водяной охладитель. Данное устройство предназначено для охлаждения входящего воздуха до определенной температуры. Принцип работы заключается в том, что входящий воздух, проходя через устройство, соприкасается с поверхностью испарителя, за счет чего поток воздуха охлаждается.



Рисунок 1.11 – Водяной охладитель

Водяной охладитель имеет несколько преимуществ относительно других систем охлаждения. Одним из преимуществ является то, что этот компонент способен быстро охладить большое пространство, так как имеет высокую производительность. При том, что этот компонент достаточно эффективно справляется со своей работой, он также является достаточно энергоэффективным. По сравнению, например, с кондиционерами, охладители потребляют намного меньше электроэнергии. Кроме того, охладители являются экологически безопасными за счет того, что не имеют в своей работе хладагентов.

Для того чтобы в вентиляционной системе можно было регулировать и контролировать поток воздуха, был изобретен еще один компонент вентиляционного оборудования, называемый клапаном. Помимо того, что клапан регулирует количество воздуха, которое подается в систему, еще клапан может перекрывать канал целиком в необходимый момент, например при пожаре.

Есть несколько различных видов воздушных клапанов. Одним из наиболее распространённых типов является обратный клапан. Его задача заключается в том, что, при остановке принудительной подачи воздуха, клапан автоматически блокируется для предотвращения обратного потока воздуха.



Рисунок 1.12 – Обратный клапан

Также достаточно распространенным типом клапана является дроссель-клапаны. Их основная задача контролировать и регулировать поток воздуха путем закрытия или открытия на определенный процент перегородки, которая

находится внутри. Это позволяет более точно настроить объем воздуха, который будет подаваться в систему.



Рисунок 1.13 – Дроссель-клапан

Есть также клапан, который выполняет функцию перенаправления потока воздуха между различными воздуховодами. Такой тип клапана называют перекидным. Эта функция достаточно полезна в системах, где необходимо поменять направление потока воздуха.

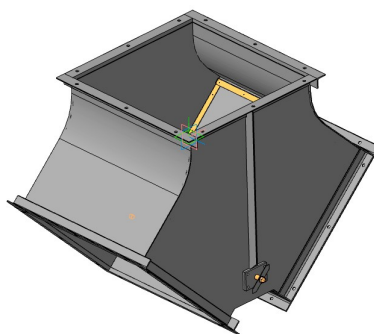


Рисунок 1.14 – Перекидной клапан

На подобии обратного клапана существует еще один компонент вентиляционных машин, называемый заслонками. Как и у обратного клапана главной задачей является блокировка обратного движения воздуха по системе после ее выключения. Заслонки могут иметь как регулируемый механизм, который можно открывать или закрывать по желанию эксплуатации, а могут иметь автоматизированные регуляторы, которые за счет специальных датчиков понимают в какое положение необходимо поставить заслонку. Такая функция позволяет более эффективно использовать систему вентиляции.



Рисунок 1.15 – Заслонка

Помимо всех технических компонентов самой вентиляционной системы, которые обеспечивают различные температурные и климатические условия, есть также отдельная часть системы, которая выполняет функцию управления вентиляцией. Такой компонент называется щит вентиляции и, соответственно, от него запитываются электроэнергией все остальные компоненты вентиляции.



Рисунок 1.16 – Щит вентиляции

Основная функция щита вентиляции заключается в том, чтобы полностью иметь возможность управлять системой. Например, как минимум с его помощью можно легко регулировать параметры объема, температуры, влажности и качества поступающего воздуха.

В зависимости от требований и необходимых условий на объекте, такая регулировка может происходить как автоматически, так и вручную. Автоматическая регулировка происходит за счет специальных датчиков, которые различаются между собой и отвечают за какой-либо определенный

аспект. Относительно требований на объекте и показаний датчиков, которые постоянно следят за качеством и показателями воздуха, щит подстраивает работу вентиляционной системы для комфортной работы.

Щит вентиляции служит как элементом управления, так и элементом защиты системы от негативных факторов. К нему подключаются различные датчики, которые помогают понять и предотвратить попадание влаги, пыли или других инородных предметов в систему, что позволяет системе работать эффективно.

Кроме всех основных компонентов есть еще некоторое количество инженерных решений, без которых не будет возможна работа системы вентиляции, но они являются более механическими, поэтому в отдельный блок они не будут выведены.

1.3 Теоретические данные для наладки вентиляционных систем

По завершении строительно-монтажных работ на объекте, в соответствии с СНиП 12-03-2001 должны производиться пусконаладочные работы. Такие работы производятся для того, чтобы вентиляционные системы были готовы для передачи в эксплуатацию. Пусконаладочные работы представляют собой этап, при котором можно убедиться в полной работоспособности всей системы, корректности монтажа установки и, что не мало важно, отрегулировать параметры системы для ее работы в условиях соответствия параметров проектной документации.

Пусконаладочные работы включают в себя достаточно большое количество мероприятий, по завершении которых в помещениях должны быть обеспечены комфортные условия в соответствии с требованиями и проектными решениями.

Рассмотрим основные этапы проведения пусконаладочных работ:

1. Подготовка к пусконаладке
2. Измерение и настройка основных параметров системы
3. Настройка работы автоматических систем
4. Проверка системы на герметичность
5. Пусконаладочные испытания
6. Окончательная регулировка и вывод на рабочий режим
7. Составление актов

Для того, чтобы появилось более ясное представление о каждом этапе, приведем основные примеры работ по каждому из них.

Первый этап является достаточно важным, так как от него зависят все дальнейшие работы. В него входят такие мероприятия как: проверка на соответствие физического исполнения системы вентиляции и кондиционирования относительно рабочей документации, проверка корректности монтажа системы, проверка всех механических соединений, проверка корректности исполнения электрических соединений и, не мало важно, убедиться в наличии необходимых инструментов, имеющих свидетельства об утверждении типа средств измерений и документы, подтверждающие о прохождении прибора поверки. Все эти мероприятия способствуют тому, что работа будет выполнена качественно и, соответственно, будет принята заказчиком своевременно и без лишних проблем.



Рисунок 1.17 – Анемометры

Далее идет этап измерения основных рабочих параметров вентиляционной системы, таких как: скорость потока воздуха, которая измеряется специальным прибором – анемометром, по специальным методикам; температура и влажность воздуха, для того чтобы в помещении был обеспечен требуемый комфортный микроклимат; давление в воздуховодах, которое может заранее просигнализировать о неверной работе вентилятора, и можно будет избежать ошибки на начальном этапе.

Следующий этап называется настройка работы автоматических систем и, исходя из названия, представляет собой наладку работы автоматизированными системами управления, которыми чаще всего оснащаются вентиляционные системы для более долгого срока службы и более удобной эксплуатации. В этом этапе инженер по пусконаладке должен убедиться в корректности работы всех датчиков, подключенных в систему, например датчики температуры уличные/комнатные, влажности, давления и другие. Кроме этого, производится задача всех необходимых параметров частотного преобразователя, которые позволяют в зависимости от каких-либо условий автоматически регулировать скорость вентилятора, температуру воздуха, время работы вентиляционной машины и других параметров для полноценной работы системы.



Рисунок 1.18 – Датчик температуры



Рисунок 1.19 – Реле давления

Четвертым этапом является проверка системы на герметичность. Этот этап определяет эффективность работы системы вентиляции: если воздух будет уходить (например, через неплотные соединения воздухопроводов), в некоторые помещения может поступать меньше воздуха, что приведёт к некомфортным условиям. Для проведения данного этапа необходимо проверить все соединения на герметичность, например, с помощью специального устройства ультразвукового течеискателя, который способен обнаружить даже самые маленькие утечки на расстоянии нескольких метров.



Рисунок 1.20 – Ультразвуковой течеискатель

Следующим этапом являются пусконаладочные испытания, которые представляют собой проверку ранее настроенных параметров и регулировок. Во время испытаний проверяют как вентиляторы ведут себя на заданных параметрах и, насколько эффективно они справляются с выполнением первоначальных требований.

Кроме этого, проверяются и отдельные компоненты системы, например, такие как фильтры, так как при неверной установке они могут как просто не выполнять свою функцию по очистке воздуха, так и в целом служить проблемой для вентилятора. Если с фильтра не снять транспортировочную целлофановую упаковку, то она полностью перекроет подачу воздуха и может вывести вентилятор из строя.

Когда система успешно проходит пусконаладочные испытания, наступает этап окончательной регулировка и вывода на рабочий режим системы. Для завершения этого этапа необходимо чтобы все элементы системы работали исправно и совместно без нареканий и создавали необходимые условия и микроклимат в нужном помещении. Соответственно, производится окончательная регулировка расходов воздуха на каждой точке системы согласно проектным данным и, если все производительности находятся в нормальном диапазоне, то этап можно считать завершенным.

Последним этапом называют составление актов, в которых фиксируют все работы, которые были проведены на объекте, настройки систем и окончательные результаты замеров. Когда данный документ утвержден и подписан, работу можно считать завершенной.

В окончательную регулировку и вывод на рабочий режим системы вентиляции входит одно из самых объемных действий, а именно определение скорости движения и расхода воздуха. Этот момент является самым длительным и сложным, так как в многоэтажных системах для настройки проектных значений расхода воздуха понадобится пройти вдоль всей системы большое количество раз. Все эти проходы и замеры расходов воздуха в основном записываются в бумажном виде и, когда количество проходов достигает хотя бы пяти, то возникает сложность еще и в прочтении верных замеров.

Приведем пример замеров скорости потока воздуха в круглом и прямоугольном воздуховодах. Для измерения скорости воздуха рекомендуется использовать механические анемометры или же электронные

термоанемометры с диапазоном скорости воздуха от 0 до 10 м/с и погрешностями $\pm(0,1 - 0,3)$ м/с.

Согласно ГОСТ 12.3.018, количество и расположение точек при измерении необходимо располагать как на рисунках 1.21 и 1.22 соответственно для круглых и прямоугольных воздуховодов.

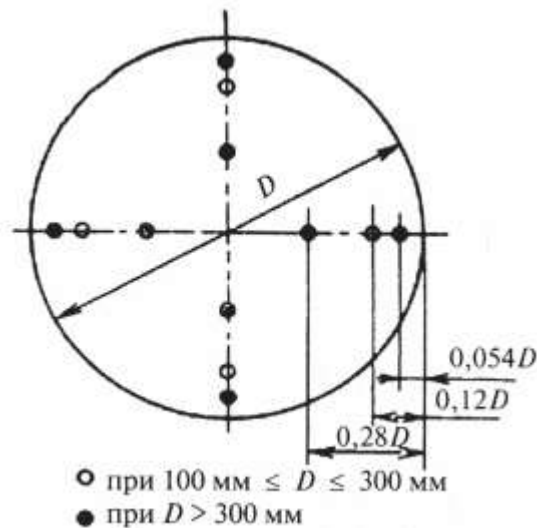


Рисунок 1.21 – Координаты точек измерения скоростей в воздуховодах круглого сечения

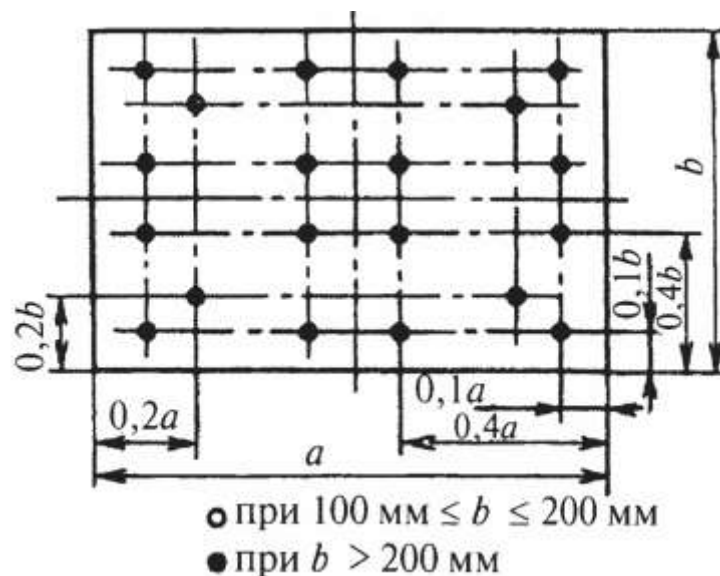


Рисунок 1.22 – Координаты точек измерения скоростей в воздуховодах прямоугольного сечения

В каждой точке измерения необходимо зафиксировать значения дважды, и их разница должна составлять менее 5%, иначе необходимо делать дополнительные измерения.

В открытых отверстиях с площадью сечения до 1м² можно производить измерения на всей площади путем медленного равномерного перемещения анемометра по всей площади. В случае, когда отверстие больше данного значения необходимо делить его на сектора, в которых производятся такие же замеры и после чего высчитывается среднее арифметическое значение всех этих замеров.

В отверстиях, которые закрыты распределительными решетками, измерения производятся с использованием анемометра большого диапазона по скорости воздуха от 0 до 60 м/с. Кроме того, необходимо использовать насадку, которая собирает выходящий воздух и сглаживает профили скорости за решеткой.



Рисунок 1.23 – Воронка для вентиляционных решеток

Соответственно при известной скорости воздуха в воздуховоде мы можем вычислить расход воздуха. В открытых проемах и при использовании воронки расход воздуха следует рассчитывать по следующей формуле:

$$L = 3600 \cdot V \cdot F, \quad (1)$$

где F – площадь сечения открытого отверстия или воронки, м²;

V – скорость воздуха, м/с.

Соответственно, пройдя всю вентиляционную систему и замерив расход воздуха на всех точках системы, необходимо будет сделать корректировки с помощью дроссель-клапанов, а иногда и частотой вращения вентилятора и повторить проходку с замерами. Данные действия необходимо выполнять до

момента, когда во всех точках системы будет соблюдена погрешность до $\pm 8\%$, согласно ГОСТ 34060-2017. В системах противодымной вентиляции данной погрешностью является значение не более 15% согласно ГОСТ Р 53300-2009.

1.4 Выводы по разделу

По итогам данного раздела было выяснено, что вентиляция появилась еще в древнем Египте и, начиная с того времени, технологии систем вентиляции постоянно развивались и усложнялись. В зависимости от варианта исполнения различают несколько видов вентиляционных систем, такие как: естественная, принудительная, рекуперационная и приточно-вытяжная варианты вентиляции.

Более сложные варианты вентиляции отличаются друг от друга наполнением используемых технологий. Вентилятор является основным компонентом принудительных вентиляционных систем, так как именно он создает поток воздуха в воздуховодах. Кроме этого, для защиты основного компонента и очищения входного воздуха существуют вспомогательные компоненты, такие как фильтры. Фильтры тоже бывают различного варианта исполнения и различной степени очистки, в зависимости от требований на объекте делается выбор.

Существует так же вариант, при котором внутри вентиляционной системы оптимизируется процесс обмена тепла и влаги. Такой вариант возможен за счет специального элемента системы, называемого рекуператором. Но в целом вентиляция способна достигать постоянной температуры при любых погодных условиях и без рекуператора. Достаточно добавить в систему блок нагревателя, который может работать как от воды, так и от электричества, что тоже позволяет делать выбор в зависимости от ситуации. Соответственно элемент, который позволяет охлаждать помещение до определенной температуры называют охладителем, который имеет

высокую холодопроизводительность по сравнению с другими системами охлаждения.

Контроль по подаче определенного количества объема воздуха осуществляют как программно, за счет изменения частоты работы двигателя вентилятора, так и механически, за счет использования еще одного элемента вентиляционных систем таких как клапаны и заслонки. Все эти элементы в целом объединяются одной системой управления и питания соответственно, без которой вентиляционная система не будет работать. Этим элементом является щит вентиляции, который управляет температурой, влажностью и количеством входного воздуха.

По завершению всех строительно-монтажных работ наступает фаза пусконаладочных работ, где настраиваются и запускаются все вентиляционные системы и подготавливаются для передачи в эксплуатацию. Фаза пусконаладочных работ в основном делится на несколько этапов таких как:

1. Подготовка к пусконаладке
2. Измерение и настройка основных параметров системы
3. Настройка работы автоматических систем
4. Проверка системы на герметичность
5. Пусконаладочные испытания
6. Окончательная регулировка и вывод на рабочий режим
7. Составление актов

Каждый из этапов по-своему важен, так как все они ведут к тому, чтобы работы были выполнены успешно, и заказчик смог официально принять данную работу. Например, если выполнить все этапы, но пропустить первый, в котором подготавливаются поверенные инструменты, то заказчик может указать на отсутствие поверки на инструменты за счет чего работы не будут считаться выполненными. Как только все этапы будут последовательно выполнены, и на руках будут готовые подписанные акты о выполненных работах, можно будет считать объект завершенным.

2. Выбор основных технологий и сущностей для веб-приложения

2.1 Что такое веб-приложение и его особенности

В настоящее время у многих компаний появляется необходимость в разработке или же внедрения веб-приложений в свою бизнес-экосистему для запуска продуктов или в помощь оптимизации своих бизнес-процессов. Но часто сталкиваются с путаницей, связанной с тем, что термины «веб-приложение» и «сайт» на данный момент времени используют взаимозаменяемо, хоть это таковым не является. Приведем определения каждого из терминов, их основные характеристики и различия для большего понимания.

Термин сайт представляет собой набор связанных веб-страниц, которые используются для предоставления пользователям какой-либо информации. Сайты могут включать в себя текстовые блоки, изображения, видео и другие элементы мультимедиа. Они могут быть как статическими, так и динамическими, но от этого их основная цель не меняется. В основном в разработке веб-страниц используются такие технологии как HTML, CSS и базовый JavaScript, что позволяет удобно их поддерживать и разрабатывать.

В основе всех сайтов чаще всего стоят статические страницы, которые вне зависимости от действий пользователей предоставляют информацию. Примерами таких сайтов можно назвать блоги людей, страницы брендов и другие схожие по смыслу страницы. За счет того, что чаще всего они не имеют сложной логики и подключаемых баз данных, такие веб-страницы легки и доступны в разработке и поддержании.

Также чаще всего сайты характеризуются простотой взаимодействия, которая обычно ограничивается несколькими основными действиями, такими как: навигация по страницам, чтение контента и, иногда, отправление обратной связи за счет заполнения форм.

Термин веб-приложение представляет собой интерактивное программное обеспечение, которое доступно для использования в веб-

браузере. Для его работы нет необходимости в установке приложения, для его использования можно воспользоваться любым устройством, имеющим доступ к Интернету и установленным веб-браузером. В основе каждого веб-приложения есть какая-либо определенная задача, выполнением которой собственно программа и занимается.

Относительно сайтов, веб-приложения способны вести себя подобно настольным приложениям, а именно включать в себя сложные интерфейсы и функционал. Это становится возможно за счет того, что они разрабатываются на основе более сложных современных технологий, таких как: HTML5, CSS3, JavaScript, фреймворки и библиотеки для представлений (например, React, Angular или Vue.js) и серверные языки программирования (например, Java, Node.js, Ruby и другие) со своими фреймворками.

К характеристикам веб-приложений можно отнести то, что они динамичны, а именно, в основном от действий пользователя зависит контент, размещаемый на представлении. К примеру, приложение онлайн-банка, где динамичность позволяет предоставить актуальную информацию в реальном времени и при этом реагировать на действия пользователя.

Также к характеристикам веб-приложения можно отнести сложную логику, а именно то, что они включают в себя более сложные алгоритмы и функционал. К примеру, интернет-магазин, который обрабатывает заказы, рассчитывает стоимость доставки до определенного адреса и при этом контролирует наличие товара на складе.

2.2 Архитектура веб-приложения

Для того, чтобы веб-приложение было максимально производительным, возможным к масштабированию и долговечным в работе, в самом начале разработки проекта необходимо ответственно подойти к продумыванию архитектуры всего приложения. Соответственно, архитектура веб-приложения представляет собой подход к проектированию, написанию и

развертыванию приложений. Она включает в себя большое количество элементов, которые входят в основу веб-приложения.

При выборе архитектуры приложения стоит обращать внимание, что от нее зависят такие факторы как:

- *Стоимость разработки и поддержки проекта* – архитектура, которая имеет большое количество компонентов, интеграций и используемых технологий, в свою очередь занимает больше времени и ресурсов, за счет чего повышается стоимость;
- *Скорость и возможность модифицирования проекта* – если есть необходимость в изменении отдельных частей проекта, стоит задуматься о выборе микросервисной архитектуры, которая позволяет делать это без необходимости перекомпилирования всего приложения;
- *Производительность проекта* – архитектура в которой предусмотрена балансировка нагрузок, кэширование данных, оптимизированное устройство баз данных и других вещей, влияющих на производительность, не только влияют на скорость работы приложения, но и на отзывы пользователей;
- *Безопасность* – защита данных от атак, возможность аутентификации и авторизации тоже является неотъемлемой частью архитектуры.

Архитектура веб-приложения, в основе своей, включает в себя три основных слоя, которые распределяют между собой различные задачи, используя различные компоненты для их выполнения.

Презентационный слой (веб-браузер, клиентский или же интерфейсный компонент). Данный слой представляет с собой инструмент взаимодействия с пользователем и, соответственно, отображение данных, которые получены с серверной части. Для организации этого слоя

используются следующие компоненты: HTML, CSS, JavaScript как базовые компоненты, а также фреймворки и библиотеки React, Angular, Vue.js.

Логический слой (веб-сервер, внутренний компонент). Этот слой занимается выполнением бизнес-логики, обработкой данных, которые были получены от презентационного слоя, взаимодействием с базой данных через слой доступа к данным и отправкой данных на презентационный слой. При выполнении этого слоя используются следующие компоненты: серверные языки программирования (Java, Node.js, Ruby, PHP, Python и другие) и также фреймворки к ним (Spring, Express.js, Ruby on Rails, Laravel, Django и другие).

Слой доступа к данным (сервер базы данных). У этого слоя достаточно понятная задача, а именно взаимодействие с базами данных. В основные задачи входит хранение, извлечение и внесение данных, поддержка транзакций. Для выполнения этих задач используют следующие компоненты: реляционные (MySQL, PostgreSQL) и нереляционные (MongoDB) базы данных.

Есть еще один архитектурный слой, который собирает в себе вспомогательные компоненты и выполняет дополнительные функции, этот слой называют **инфраструктурным**. В его задачи входят такие аспекты как: развертывание приложения, мониторинг и управление состояниями и ресурсами, и безопасность. Для его работы могут использоваться такие компоненты как: веб-серверы (Nginx, Apache), контейнеризация проекта (Docker), оркестрация контейнеров (Kubernetes) и мониторинг и логирование (Grafana, Prometheus).

В зависимости от подхода различают несколько видов архитектуры, основные из них являются **монолитные** и **микросервисные** приложения. **Монолитным** приложением называют архитектуру, в которой все компоненты приложения объединяются в единый неделимый блок кода. В таком подходе программирования вся функциональность управляется в рамках одного блока (монолита) и выполняется в одном процессе.



Рисунок 2.1 – Монолитная архитектура

Проекты с таким подходом в архитектуре было проще разрабатывать и разворачивать, так как все находится в одном модуле, поэтому такой вид архитектуры долгое время считался традиционным подходом к разработке ПО. Кроме того, за счет отсутствия затрат на связь между модулями, так как они обмениваются данными в одном блоке, монолиты работают лучше.

Из минусов такого подхода является трудоемкий процесс масштабирования и сложность ремонта. При масштабировании монолитных систем, если только один компонент требует дополнительных ресурсов, придется масштабировать полностью весь монолит. Также одно незначительное изменение в одном компоненте может повлиять на все приложение целиком.

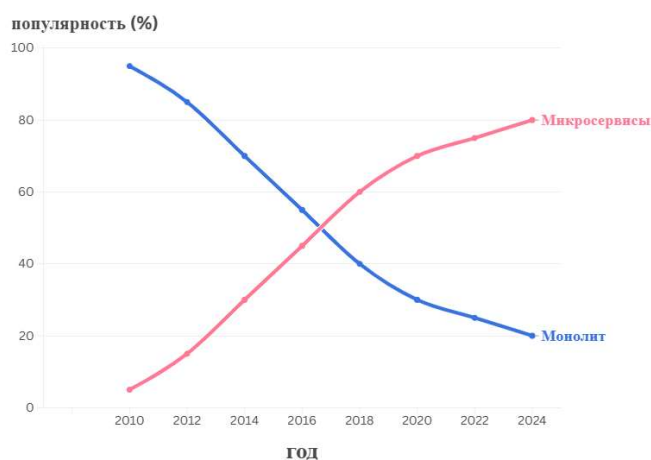


Рисунок 2.2 – Популярность монолитов и микросервисов (2010-2024)

Микросервисная архитектура представляет собой подход к программированию, в котором приложение состоит из набора децентрализованных и независимо развертываемых сервисов. В настоящее время такой тип архитектуры становится более популярным, чем монолит.

К преимуществам микросервисной архитектуры можно отнести то, что это очень гибкий подход к программированию, так как есть возможность независимо от всего проекта масштабировать одну службу. Также из преимуществ можно отметить, что при необходимости обновить какой-либо определенный блок, разработчикам не понадобится отключать все приложение целиком. Еще один полезный момент, что в различных блоках проекта могут использоваться различные стек технологий, что позволяет максимально эффективно использовать различные инструменты.

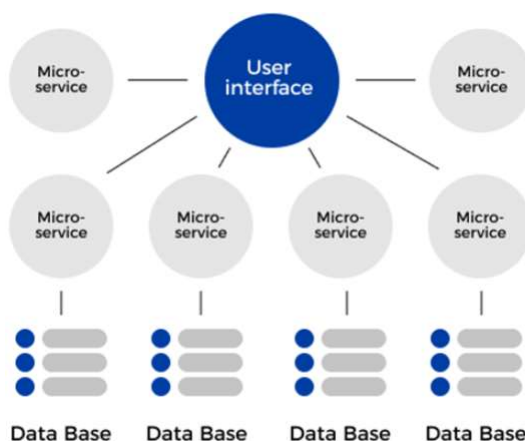


Рисунок 2.3 – Микросервисная архитектура

К недостаткам микросервисной архитектуры можно отнести то, что такой подход имеет повышенную сложность в разработке, а именно необходимо иметь уверенные знания об управлении взаимодействиями между блоками и умение управлять распределенными данными. Кроме того, разработка и эксплуатация микросервисов занимает больше времени и ресурсов, соответственно стоимость такого продукта будет выше. Также за счет того, что микросервисы независимы друг от друга для их взаимодействия

требуется время, поэтому могут возникать некоторые задержки и увеличение время отклика.

Для того чтобы микросервисы начали свою работу непосредственно в рабочей среде, а не в среде разработки, используют специальный механизм, называемый контейнеризацией. Для использования такого механизма чаще всего используют специальную платформу для разработки и запуска контейнерных веб-приложений, которая называется Docker. Она позволяет обернуть любое веб-приложение в контейнер, жизненным циклом которого соответственно можно управлять через Docker, при этом есть возможность автоматизировать процесс запуска и развертывания приложения.

Данная платформа позволяет запускать и тестировать код в изолированной среде любого рода, что является очень полезной функцией при разработке ПО, так как можно проверить есть ли какие-либо проблемы с зависимостями или же проблемы с какой-либо рабочей средой. Соответственно такой подход исключает проблему, когда приложение полноценно работало в тестовом режиме, а при переносе в рабочую среду отказывается работать. Также при необходимости, например при увеличении нагрузки, есть возможность развернуть новый экземпляр необходимого контейнера для распределения нагрузки и убирать лишние работающие контейнеры при отсутствии загруженности.

В целом, при росте популярности веб-приложения для решения проблемы нагрузки существует термин масштабирование, который представляет собой выделение дополнительных ресурсов под приложение. В зависимости от варианта масштабирования существует два типа:

- **Вертикальное масштабирование** представляет собой увеличение ресурсов таких как память, вычислительная мощность и объем хранилища, на существующем серверном оборудовании. Это означает, что необходимо сменить машину на более производительную, то есть, например с 4 ядер на 8 и более ядер, и

тоже самое с ОЗУ с 8 Гб на 16 – 64 Гб ОЗУ. Соответственно характеристики новой машины подбираются в зависимости от потребностей приложения и расчета запаса под дополнительные нагрузки;

- **Горизонтальное масштабирование** представляет собой способ расширения ресурсов приложения за счет добавления новых серверов. Это означает, что на новом сервере будет развернут дополнительный экземпляр с помощью которого будет балансироваться нагрузка.

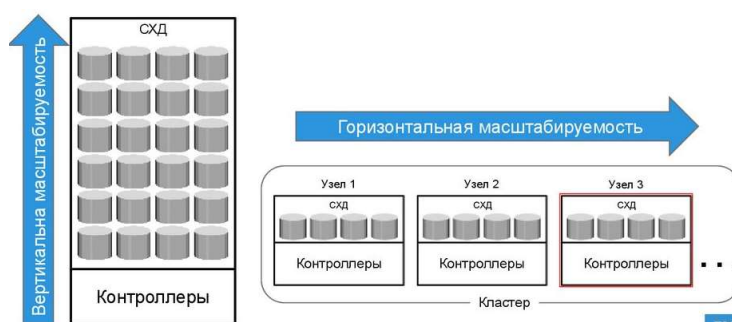


Рисунок 2.4 – Вертикальное и горизонтальное масштабирование

В свою очередь вертикальное масштабирование имеет преимущества за счет того, что нет необходимости доработки и наладки приложения, так как меняется только компьютерное железо, но в таком варианте есть предел как по количеству ядер процессора, так и по количеству ОЗУ. Как только предел будет достигнут, будет доступна только горизонтальное масштабирование.

2.3 Выбор технологий для веб-приложения

Для написания веб-приложения в основном будет использоваться язык программирования Java и фреймворк Spring.

Java – один из самых популярных и эффективных объектно-ориентированных языков программирования (ООП), который строится на основе классов. ООП это одно из свойств, которое позволяет программе хорошо расширяться и масштабироваться за счет того, что вся программа состоит из отдельных блоков.

Также у Java есть особенность, что она совмещает в себе лучшее из интерпретируемых и компилируемых языков. Интерпретирование представляет собой процесс, когда специально установленная на компьютер программа – интерпретатор, проходит по коду и выполняет его на ходу, не используя машинный код. Компилирование представляет собой процесс, когда до запуска программы написанный код переводится в машинный для дальнейшей отправки в процессор. В таком варианте процессор при запуске программы начинает сразу выполнять скомпилированный код. Соответственно Java является компилируемым языком, но сначала процесс компиляции переводит код в байткод (код для JVM), а после происходит процесс интерпретации с байткода в машинный код.

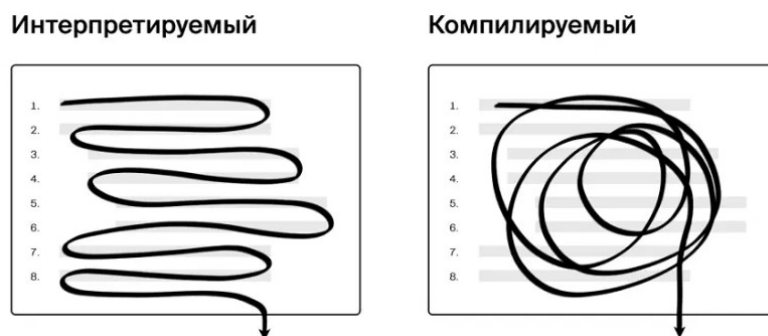


Рисунок 2.5 – Выполнение программ интерпретируемых и компилируемых ЯП

Основным преимуществом Java является кросс-платформенность. Это означает то, что несмотря на платформу, на которой была написана программа, к примеру Windows, при необходимости запустить эту программу на Linux или macOS – не возникнет никаких проблем. Это возможно за счет виртуальной Java-машины (JVM), которая соответственно является посредником между написанной программой и железом.

На языке Java пишется достаточно большое количество различных программ, к примеру: банковские приложения, настольные приложения, приложения для Android, промышленные программы, веб-приложения и множество других. Соответственно, в нашем случае Java будет использоваться для написания веб-приложения.

Далее рассмотрим, что такое Spring и для чего он будет использован. Spring Framework представляет собой мощный и гибкий фреймворк для разработки программ на Java, который предоставляет большое количество инструментов для создания различных приложений и микросервисов. Его основной целью является упрощение процесса разработки в том плане, что улучшается структура кода, повышается его читаемость, за счет использования разных паттернов и принципов проектирования.

Spring Framework включает в себя несколько различных модулей, которые отвечают за определенные задачи:

- ***Spring Core Container.*** Данный модуль является основным. В его задачи входит предоставление механизма управления объектами и их зависимостями, что позволяет писать гибкие и удобно расширяемые приложения. Для это используется инверсия управления или Inversion of Control (IoC) и внедрение зависимостей или Dependency Injection (DI).
- ***Spring Web.*** Данный модуль является ответственным за организацию создания веб-приложений. В его работу входит поддержка для работы с сервлетами, кроме того, в его работу входит интеграция с другими веб-технологиями, к примеру REST или WebSocket.
- ***Spring Data Access/Integration.*** Данный модуль предоставляет инструменты, которые позволяют работать с базами данных, а также интегрировать с различными технологиями для работы с данными, такими как JPA, JDBC и Hibernate.
- ***Spring Security.*** Кроме того, существует модуль, который позволяет обеспечить безопасность приложения. В его задачи входит осуществление аутентификации, авторизации и защиты от атак.
- ***Spring AOP (Aspect-Oriented Programming).*** Данный модуль позволяет воспользоваться аспектно-ориентированным программированием, то есть, к примеру отделить от бизнес-логики такой функционал как логирование.

Spring Framework имеет достаточно широкое использование в различных задачах и проектах за счет своих разнообразных модулей. Помимо описанных выше основных компонентов, существуют и другие модули относящиеся к Spring. К примеру, для более упрощенной интеграции с другими технологиями существуют *Spring Integration* и *Spring Batch*.

Еще одним из вариантов использования Spring является создание микросервисов, для быстрого создания и удобной поддержки которых используются такие модули как *Spring Boot* и *Spring Cloud*. Spring Boot отвечает за удобство развертывания и настройки приложений, так как имеет встроенные шаблоны, которые ускоряют все эти процессы. А Spring Cloud позволяет управлять конфигурацией распределенных систем, балансировать нагрузки, обнаруживать новые микросервисы, в целом предоставляет инструменты для поддержания распределенных систем.

Кроме того, есть модули, которые позволяют создавать тесты для компонентов приложения. *Spring Test* включает набор аннотаций и позволяет тестировать компоненты внутри контекста Spring, не запуская всё приложение. Также для тестирования используют еще такие инструменты как Mockito и JUnit. Они занимают процесс Mocking'а, который включает в себя замену реальных объектов на специально имитированные для тестов. Это позволяет исключить влияние зависимостей, так как Mock-объекты работают, следуя своим настройкам, и исключить влияние состояния внешних зависимостей, то есть тестирование происходит не зависимо от других компонентов.

Также есть модули, которые предоставляют инструменты для разработки веб-приложений. *Spring MVC* – это веб-фреймворк, который позволяет разрабатывать приложение с четким разделением проблем и задач, для чего используется паттерн программирования MVC (Model-View-Controller). За счет этого, происходит прием и обработка HTTP-запросов, отображение необходимых представлений и маршрутизация по этим представлениям. Для ускорения работы и повышения

производительности веб-приложений есть модуль Spring WebFlux, который позволяет использовать асинхронное программирование и реактивные потоки для ввода/вывода данных.

Собственно, в основе написанного веб-приложения заложена архитектура model-view-controller (MVC). Данный подход к программированию включает в себя пять уровней, каждый из которых имеет определенные задачи.

1. Уровень представления или *Presentation Layer*. В задачи этого слоя входит взаимодействие с пользователями и, соответственно, визуализация интерфейса. Для данного уровня в написании приложения использовались такие технологии как HTML5, CSS3, JavaScript, Bootstrap 5, Thymeleaf.

Для включения Bootstrap 5 необходимо поместить тег <link> между тегами <head> для подключения CSS и тег <script> перед закрывающим тегом </body>.

Bootstrap 5 представляет собой популярный бесплатный фреймворк для фронтенд-разработки сайтов и веб-приложений с открытым исходным кодом. Он включает в себя большой набор инструментов для разработки, например такие как:

- ***Адаптивная система сеток на основе Flexbox.*** С ее помощью можно удобно динамически размещать контент с колонками, которые автоматически адаптируются под различный размер экрана.
- ***Готовые компоненты интерфейса.*** В Bootstrap 5 появляется возможность брать для использования готовые элементы, такие как кнопки, формы, уведомления, панели навигации, карточки и другие.
- ***Утилитные классы стилизации.*** Они позволяют использовать стилизацию без написания пользовательского CSS, например обеспечение отступов, теней, границ и различных других эффектов.

- **Плагины.** Позволяют использовать динамические компоненты такие как модальные окна, всплывающие подсказки и другие, с использованием чистого JavaScript.

Далее перейдем к следующей технологии, а именно Thymeleaf. Он представляет собой шаблонизатор для языка программирования Java, который дает возможность динамически создавать HTML-страницы и веб-шаблоны. Его основной задачей является предоставление естественных HTML-шаблонов, которые можно открыть двойным щелчком и просмотреть результат без запуска на сервере.

Для того чтобы Thymeleaf был интегрирован в проект, необходимо добавить соответствующую зависимость в файл pom.xml.

Листинг 1. Зависимость Thymeleaf:

Внедрение зависимости Thymeleaf в проект

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-thymeleaf</artifactId>
4 </dependency>
```

2. Уровень контроллера или *Controller Layer*. Получает запросы от пользователя и назначает их соответствующему сервису или компоненту. Это позволяет использовать такие элементы, как контроллеры, которые отвечают за управление запросами и выдачу ответов. Уровень контроллера служит мостом между уровнем представления и уровнем сервиса.

3. Уровень сервиса или *Service Layer*. Отвечает за реализацию бизнес-логики приложения. Состоит из таких частей, как сервисы и менеджеры, которые обрабатывают данные и обеспечивают функциональность приложения. Между уровнем контроллера и уровнем доступа к данным в качестве интерфейса служит уровень сервиса.

4. Уровень доступа к данным или *Data Access Layer*. Отвечает за хранение и извлечение данных из базы данных. Он состоит из таких частей, как DAO (объекты доступа к данным), которые отвечают за выполнение

операций с базой данных. Уровень доступа к данным служит связующим звеном между уровнем сервиса и уровнем базы данных. Для данного уровня в написании приложения использовалась технология Liquibase.

Liquibase – это инструмент, который позволяет управлять миграциями баз данных, а именно эта открытая библиотека берет на себя задачи отслеживания, управления и изменения схем базы данных. Из преимуществ данной технологии можно отметить, что каждое изменение схемы базы данных отслеживается и логируется. Кроме того, есть функционал автоматической проверки применимости миграции, то есть при конфликтующих изменениях срабатывают механизмы блокировки параллельного применения, что позволяет защитить целостность данных.

5. *Уровень базы данных или Database Layer.* Хранение и извлечение данных из базы данных являются обязанностями уровня базы данных. Сама база данных и система управления базами данных (СУБД) составляют эту систему. Уровень базы данных, отвечает за взаимодействие с фактическим хранилищем данных. В данном уровне приложения использовалась технология PostgreSQL.

2.4 Выделение основных сущностей и их атрибутов

Первой сущностью для производства пуско-наладочных работ можно назвать объект, на котором эти работы будут производиться, так как это первое место куда необходимо приехать для их проведения.

Далее, мы получаем проект ОВиК (отопление вентиляция и кондиционирование), в котором есть список вентиляционных систем. Каждая вентиляционная система чаще всего имеет различные параметры. Соответственно следующей сущностью можно выделить вентиляционную систему.

У каждой вентиляционной системы есть различное количество точек выброса/забора воздуха. У каждой такой точки есть параметры, которые для каждой точки задаются в зависимости от типа помещения, объема помещения

и других условий. Соответственно, следующей сущностью будет точка измерения.

Так как пуско-наладочные работы чаще всего производятся не за один проход, вероятнее всего замеров в каждой точке будет в любом случае более двух. Результаты замеров соответственно меняются в зависимости от процента открытия клапана, частоты работы вентилятора и других условий. Соответственно, еще одной сущностью в проекте будет само измерение.

Основные сущности были выявлены, теперь необходимо определить какими атрибутами они будут обладать. В целом у сущности объекта достаточно понятные атрибуты, а именно название объекта и адрес объекта, для того чтобы можно было различать их между собой. Кроме того, у каждого объекта будет уникальный id для идентификации и, соответственно, список вентиляционных систем, которые относятся к данному объекту.

Атрибуты вентиляционной системы в целом тоже достаточно несложные. У каждой вентиляционной системы есть название, соответственно это будет одним из атрибутов, также есть полный объем расхода воздуха, который система должна выдавать по проекту. Для идентификации вентиляционная система будет иметь атрибут уникального id. Для того, чтобы определять к какому объекту данная вентиляционная система относится будет добавлен атрибут id объекта. Также у каждой вент системы есть точки измерения, для которых добавим атрибут списка точек измерения, которые относятся к данной системе.

Для сущности точки измерения будет больше различных атрибутов, так как она включает в себя большее количество параметров. Соответственно для идентификации будет уникальный id точки измерения. Для того чтобы различать точки измерения между собой, будет добавлен атрибут названия. Чтобы было понятно какой прибор необходим для проведения измерения, добавим атрибут типа измерения, заодно будет добавлен тип отверстия, чтобы было понимание какого характера отверстия. Также будет добавлен атрибут вентиляционной системы, к которой относится точка измерения и также будет

список измерений, которые относятся к данной точке. Кроме того, у точки измерения необходимо добавить атрибут площади сечения и объема воздуха в данной точке, имея эти атрибуты можно пересчитать еще один атрибут скорость потока воздуха. Добавим также атрибуты текущих объема воздуха и скорости потока воздуха в точке измерения и расхождение с проектными, для подсвечивания готовности точки.

Для сущности измерения нам нет необходимости в добавлении большого количества атрибутов, поэтому будет добавлены уникальное id для идентификации измерения и точку к которой данное измерение относится. Также необходимо добавить соответственно значение измерения и добавим заметки, чтобы можно было указать при каких условиях было сделано измерение.

2.5 Выводы по разделу

Веб-приложение представляет собой интерактивное программное обеспечение, доступ к которому осуществляется через веб-браузер. В основном каждое веб-приложение имеет под собой необходимость в решении какой-либо проблемы и, соответственно, разрабатывается под определенные цели. В отличие от сайтов, которые в основном являются статической страницей для предоставления информации, веб-приложения обновляются в режиме реального времени, то есть предоставляется более актуальная информация, а также приложение реагирует на действия пользователя.

Архитектура веб-приложений играет большую роль на разработку, так как при ее выборе необходимо учитывать такие факторы как: стоимость разработки и поддержки проекта, скорость разработки и модификации проекта, производительность и безопасность приложения. В основе архитектуры чаще всего есть три слоя, каждый из которых отвечает за определенную задачу. Первый слой является презентационным, он отвечает за взаимодействие с пользователями и отображение данных. Далее следует логический слой, который включает в себя бизнес-логику приложения,

обрабатывает полученные данные и взаимодействует со слоем данных. Соответственно следующим компонентом является слой доступа к данным, который непосредственно взаимодействует с базами данных.

В зависимости от подхода к программированию различают несколько видов архитектур, разберем более популярные – монолитные и микросервисные приложения. Монолитами являются приложения структура кода которого объединена в единый неделимый блок. В таком варианте все процессы протекают последовательно и весь функционал находится в одном блоке. Микросервисная архитектура наоборот же представляет собой набор децентрализованных и независимо развертываемых сервисов, каждый из которых отвечает за какую-либо задачу. Соответственно при выборе архитектуры необходимо продумать какой вариант подходит больше, так как и выбор сервера необходимо производить, отталкиваясь от архитектур.

Существует два варианта масштабирования проектов, а именно вертикальное и горизонтальное масштабирование. Они отличаются тем, что при вертикальном масштабировании необходимо увеличивать ресурсы сервера, то есть менять железо на более мощное и производительное. В варианте горизонтального программирования увеличение ресурсов происходит за счет добавления дополнительных экземпляров, то есть можно открыть еще один экземпляр приложения на другом сервере, который начнет делить нагрузку. Соответственно монолитной архитектуре подходит только вариант с вертикальным масштабированием так как все приложение должно находиться в одном месте. Для микросервисной архитектуры подходят оба варианта, так как можно масштабировать отдельный компонент проекта.

Для написания веб-приложения были подобраны язык программирования Java и фреймворк к нему Spring Framework. Java является одним из самых популярных объектно-ориентированных языков. Она является кроссплатформенной за счет использования своей виртуальной Java-машины (JVM), что позволяет разрабатывать и запускать приложения вне зависимости от установленной операционной системы, что является

огромным преимуществом. На данный момент она занимает широкую позицию на рынке, так как может использоваться в различного типа программ. Spring Framework представляет собой мощный и гибкий фреймворк для разработки программ на Java. Он включает в себя различные инструменты для разработки, которые позволяют более удобно и легче подходить к программированию различных систем.

В основе написанного веб-приложения лежит архитектура model-view-controller (MVC), которая включает в себя разбиение на пять уровней, каждый из которых будет отвечать за определенную задачу, а именно: уровень представления, уровень контроллера, уровень сервиса, уровень доступа к данным и уровень базы данных. Уровень представления берет на себя задачу взаимодействия с пользователями и визуализацию данных с использованием соответствующих технологий. Далее идет уровень контроллера, который получает запросы пользователя и направляет их в соответствующий сервис или компонент. Уровень сервиса в свою очередь уже реализует бизнес-логику с поступившими данными и при необходимости передает их далее. Уровень доступа к данным берет на себя хранение, ввод и вывод данных из базы данных. И соответственно уровень базы данных, который отвечает уже за взаимодействие с фактическим хранилищем данных.

Для написания приложения необходимо понимать какие сущности будут в нем участвовать. Самой первой сущностью будет рабочий объект, который включает в себя id, название, адрес и список вентиляционных систем. Далее соответственно переходим к сущности вентиляционной системы, которая тоже имеет свой id, наименование, полный расход воздуха, список точек, которые принадлежат данной системе, и объект, к которому принадлежит вент система. Далее переходим к точке измерения которая имеет такие атрибуты как: id, наименование, вентиляционную систему, к которой она принадлежит, тип измерения в данной точке, тип отверстия, площадь сечения отверстия, объемы воздуха текущие и по проекту, скорости потока воздуха текущие и по проекту, процент расхождения и список измерений

которые производились в данной точке. Далее заключительная сущность, которая представляет собой измерение в точке и имеет следующие атрибуты такие как: id, наименование измерения, результат измерения и примечание для записи условий, при которых производились измерения. Данных сущностей должно быть достаточно для написания приложения.

3. Разработка приложения для наладки вентиляционных систем

3.1 Инициализация и создание структуры приложения

Разработка приложения будет производиться в профессиональной интегрированной среде разработки IntelliJ IDEA 2023.2.2 (Ultimate Edition). Для инициализации проекта воспользуемся функцией среды разработки, которая позволяет создать проект с сервера <https://start.spring.io>. Для этого необходимо нажать New → Project... Далее в меню выбрать в списке Generators – Spring Initializr и заполнить необходимые пункты.

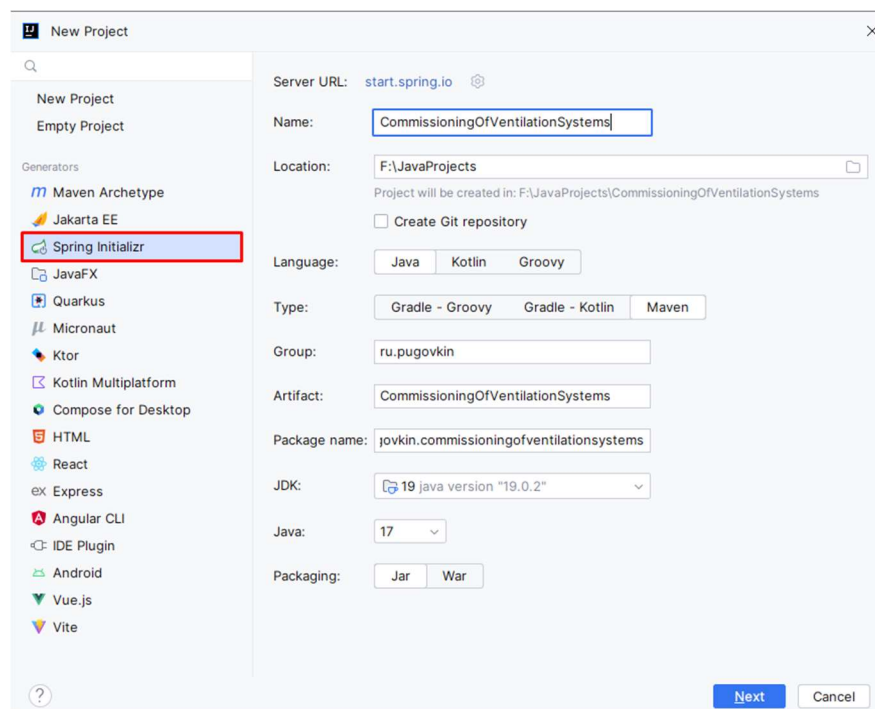


Рисунок 3.1 – Инициализация проекта

Далее переходим на следующее меню, в котором предлагается выбрать зависимости, которые будут использоваться в нашем проекте. Соответственно

необходимо добавить интересующие зависимости и нажать Create для создания проекта.

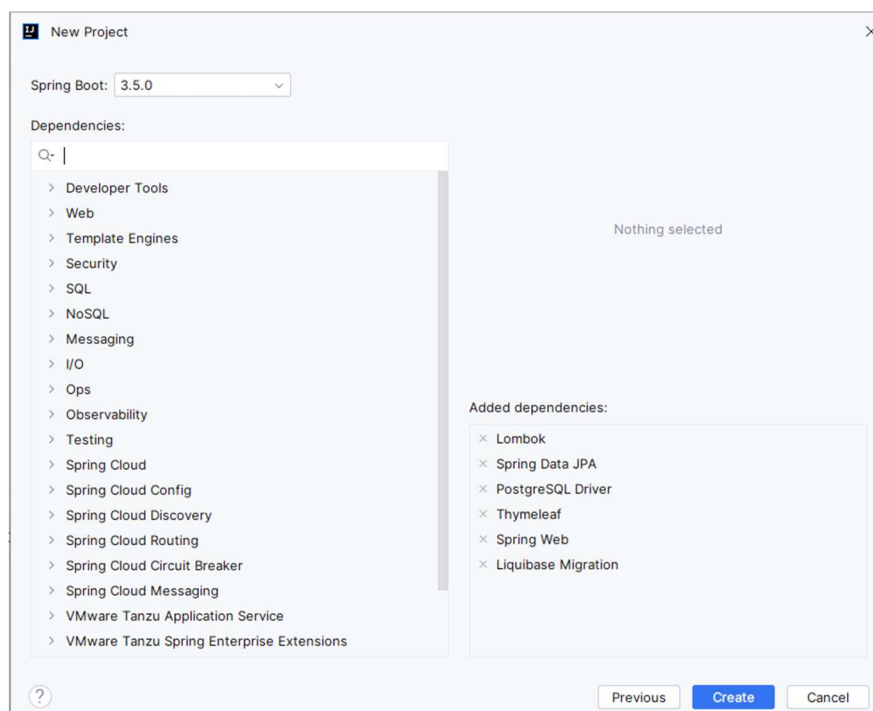


Рисунок 3.2 – Выбор зависимостей

Далее получаем чистый проект с которого начинается наша разработка. Для удобства программирования необходимо добавить некоторое разделение на Package. Создадим необходимые папки для дальнейшего размещения в них, а именно следующие пакеты: controllers, dto, repository, services и domain. В папку controllers будут направлены классы контроллеры. В папке dto будут складываться классы Data Transfer Object сущностей и dtoMapper. Папка repository предназначена для классов репозитория для всех сущностей. В папку services складываются классы сервисы для всех сущностей. Папка domain предназначена для хранения моделей, в нашем случае будут основные модели и нумераторы.

Также в папке resources для правильной работы приложения необходимо чтобы были папки templates, liquibase с внутренней папкой changesets и preliquibase. В папку templates будем размещать шаблоны страниц нашего приложения. Папки liquibase и preliquibase нужны для использования

инструмента Liquibase который предназначен для управления миграциями базы данных.

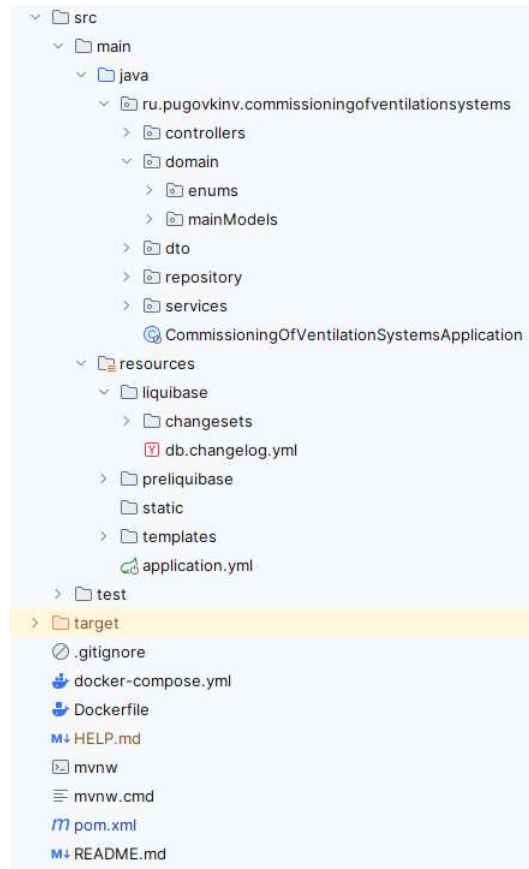


Рисунок 3.3 – Конечная структура приложения

3.2 Написание классов приложения

Первыми классами в нашем приложении будут сущности, которые были описаны ранее. Начнем с сущности рабочего объекта, а именно он будет называться PlaceOfWork. Так как этот класс у нас является сущностью помечаем его аннотацией `@Entity`. Для того чтобы были доступны основные функции класса, но не был загроможден код можно воспользоваться специальной аннотацией Lombok, а именно `@Data`, которая включает в себя геттеры и сеттеры, методы `toString()`, `equals()` и `hashCode()`, а также конструктор. Для того, чтобы связать класс с таблицей в базе данных с определенным наименованием воспользуемся аннотацией `@Table (name = "placeOfWork")` для явного указания.

Листинг 2. Класс рабочего объекта:

Структура класса рабочего объекта

```
1 package ru.pugovkinv.commissioningofventilationsystems.domain.mainModels;
2
3 import jakarta.persistence.*;
4 import lombok.Data;
5
6 import java.util.List;
7
8 /**
9  * Рабочий объект
10 */
11 @Entity
12 @Data
13 @Table(name = "placeOfWork")
14 public class PlaceOfWork {
15     /**
16      * Уникальный айди объекта
17      */
18     @Id
19     @GeneratedValue(strategy = GenerationType.IDENTITY)
20     @Column(name = "object_id")
21     private Long objectId;
22     /**
23      * Название объекта
24      */
25     private String nameOfObject;
26     /**
27      * Адресс объекта
28      */
29     private String addressOfObject;
30     /**
31      * Список систем, которые относятся к данному объекту
32      */
33     @OneToMany(mappedBy = "placeOfWork")
34     private List<VentilationSystem> ventilationSystems;
35 }
```

Соответственно, как и было сказано ранее будет четыре атрибута, один из которых уникальное id. Для такой необходимости есть аннотация `@Id`, которая используется для пометки первичного ключа в соответствующей таблицы. Для настройки стратегии генерации первичного ключа, можно воспользоваться аннотацией `@GeneratedValue` (strategy = `GenerationType.IDENTITY`).

Следующей сущностью по которой будет написан класс будет вентиляционная система. Далее все по аналогии с предыдущим классом мы используем аннотации. Но в данном случае мы также добавляем аннотацию `@JoinColumn` с помощью которой будет производиться связь.

Листинг 3. Класс вентиляционной системы:

Структура класса вентиляционной системы

```
1 package ru.pugovkinv.commissioningofventilationsystems.domain.mainModels;
2
3 import jakarta.persistence.*;
4 import lombok.Data;
5
6 import java.util.List;
7
8 /**
9  * Вентиляционная система
10  */
11 @Entity
12 @Data
13 @Table(name = "ventilation_system")
14 public class VentilationSystem {
15     /**
16      * Уникальное айди вентиляционной системы
17      */
18     @Id
19     @GeneratedValue(strategy = GenerationType.IDENTITY)
20     @Column(name = "ventilation_system_id")
21     private Long ventilationSystemId;
22     /**
23      * Название системы (Например, В1 - коридоры корпуса А)
24      */
25     private String nameOfSystem;
26     /**
27      * Полный расход объема воздуха который система должна выдавать по проекту
28      */
29     private Double fullAirVolume;
30     /**
31      * Список точек измерений, которые относятся к данной системе
32      */
33     @OneToMany(mappedBy = "ventilationSystem")
34     private List<Point> pointsOfSystem;
35     /**
36      * Рабочий объект на котором находится данная система (Например, РУДН)
37      */
38     @ManyToOne @JoinColumn(name = "object_id")
39     private PlaceOfWork placeOfWork;
40 }
```

Далее также необходимо написать класс сущности точки измерения. Это делается по аналогии с прошлыми классами, но добавляется большее количество атрибутов. Здесь также будет добавлена аннотация `@Enumerated` для отображения перечислений в базе данных (см. Приложение А, листинг 1).

Для данного класса нужны дополнительные вспомогательные нумераторы. Один из нумераторов будет отвечать за тип измерения, а второй за тип отверстия.

Листинг 4. Класс нумератор типа измерения:

Структура класса нумератора типа измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.domain.enums;
2
3 /**
4  * Типы замеров: внутри вентиляционного канала, на вентиляционной решетке
5  */
6 public enum TypeMeasuring {
7     INSIDE_THE_DUCT, ON_THE_GRATE
8 }
```

Листинг 5. Класс нумератор типа отверстия:

Структура класса нумератора типа отверстия

```
1 package ru.pugovkinv.commissioningofventilationsystems.domain.enums;
2
3 /**
4  * Типы отверстия: круглое, прямоугольное
5  */
6 public enum TypeOfHole {
7     CIRCULAR, RECTANGULAR
8 }
```

Следующей сущностью для которой будет написан класс будет измерение. В данном классе все пишется по аналогии с прошлыми классами (см. Приложение Б. листинг 1).

Далее напишем классы DTO для всех типов сущностей, а также dtoMapper для них. Начнем с первой сущности, а именно с рабочего объекта. Данный класс будет заниматься передачей данных между различными уровнями приложения. Также для определенных атрибутов будет добавлена аннотация `@NotEmpty(message = "Название объекта должно быть не пустым!")`, которая указывает, что данное поле не должно быть пустым, иначе будет отображаться сообщение об ошибке message.

Листинг 6. Класс dto для объекта работы:

Структура класса dto рабочего объекта

```
1 package ru.pugovkinv.commissioningofventilationsystems.dto;
2
3 import jakarta.validation.constraints.NotEmpty;
4 import lombok.Data;
5 import domain.mainModels.VentilationSystem;
6
7 import java.util.List;
8
9 @Data
10 public class PlaceOfWorkDto {
11
12     /**
13      * Уникальный айди объекта
14      */
15     private Long objectId;
16
17     /**
18      * Название объекта
19      */
20     @NotEmpty(message = "Название объекта должно быть не пустым!")
21     private String nameOfObject;
22
23     /**
24      * Адресс объекта
25      */
26     @NotEmpty(message = "Адресс объекта должен быть не пустым!")
27     private String addressOfObject;
28
29     /**
30      * Список систем, которые относятся к данному объекту
31      */
32     private List<VentilationSystem> ventilationSystems;
33 }
```

Далее у нас следует dto класс для сущности вентиляционной системы. В целом структура выстраивается по аналогии с предыдущим классом, но добавляется еще одна аннотация `@Pattern()`, в скобках которой также указывается сообщение об ошибке, которое будет выводиться и также regex, который представляет собой определенный текстовый шаблон, который необходимо задать. Листинг данного кода можно посмотреть в Приложении В, листинг 1.

Следующим классом будет dto сущности точки измерения. Соответственно пишется он аналогично, в нем также используются аннотации `@NotEmpty` и `@Pattern`, изменения только в атрибутах, которые добавляются (см. Приложение В, листинг 2).

Последним dto для сущностей будет dto для измерений. В его структуре также используются прошлые аннотации. Структура также схожа с прошлыми (см. Приложение В, листинг 3).

Далее идет более сложный класс, который занимается преобразованием сущностей в dto и наоборот соответственно. Таким классом является `dtoMapper`. Весь его функционал направлен на то, чтобы получать на вход сущность и преобразовывать ее в dto и подавать на вывод или же наоборот. Приведем пример кода на примере преобразования сущности рабочего объекта.

Листинг 7. Класс `dtoMapper`:

Пример преобразования сущности рабочего объекта

```
1  @Component
2  public class DtoMapper {
3      /**
4       * Преобразование объекта в его dto
5       *
6       * @param placeOfWork объект
7       * @return его dto
8       */
9      public PlaceOfWorkDto toPlaceOfWorkDto(PlaceOfWork placeOfWork) {
10         PlaceOfWorkDto placeOfWorkDto = new PlaceOfWorkDto();
11         placeOfWorkDto.setObjectId(placeOfWork.getObjectId());
12         placeOfWorkDto.setNameOfObject(placeOfWork.getNameOfObject());
13         placeOfWorkDto.setAddressOfObject(placeOfWork.getAddressOfObject());
14         placeOfWorkDto.setVentilationSystems(placeOfWork.getVentilationSystems());
15         return placeOfWorkDto;
16     }
17     /**
18      * Преобразование dto объекта в объект
19      *
20      * @param placeOfWorkDto dto объекта
21      * @return объект
22      */
23     public PlaceOfWork toPlaceOfWork(PlaceOfWorkDto placeOfWorkDto) {
24         PlaceOfWork placeOfWork = new PlaceOfWork();
25         placeOfWork.setObjectId(placeOfWorkDto.getObjectId());
26         placeOfWork.setNameOfObject(placeOfWorkDto.getNameOfObject());
27         placeOfWork.setAddressOfObject(placeOfWorkDto.getAddressOfObject());
28         placeOfWork.setVentilationSystems(placeOfWorkDto.getVentilationSystems());
29         return placeOfWork;
30     }
31 }
```

Далее напишем классы сервисы для наших сущностей. Для того чтобы класс был распознан необходимо воспользоваться аннотацией `@Service`,

которая указывает на то, что класс является частью сервисного слоя приложения. Также добавим аннотацию Lombok `@RequiredArgsConstructor`, которая заменяет написание конструктора в коде для более удобной читаемости. Рассмотрим код класса службы для рабочего объекта. В целом для работы приложения нам понадобится выполнение операций CRUD: операции создания (Create), чтения (Read), обновления (Update) и удаление (Delete). Для этого соответственно пишем функции, которые будут реализовывать эти операции с использованием соответствующего репозитория (см. Приложение Г, листинг 1).

Аналогично пишем классы сервисы для работы с вентиляционными системами (см. Приложение Г, листинг 2), с точками измерения (см. Приложение Г, листинг 3) и с измерениями (см. Приложение Г, листинг 4).

Далее напишем код репозитория для всех сущностей. Для этого необходимо создать интерфейс, который расширяем с использованием `extends JpaRepository<>`. Для репозитория объекта необходимо разместить в скобках необходимый класс и тип данных, который используется для id, а именно получится `<PlaceOfWork, Long>`. Кроме того, необходимо добавить аннотацию `@Repository`, которая указывает на то, что данный класс выполняет роль репозитория. При подключении интерфейса `JpaRepository` автоматически предоставляется доступ к базовым CRUD операциям. Если данного функционала недостаточно, то можно написать определение необходимого метода и интерфейс автоматически сгенерирует его наполнение. Если и этого будет недостаточно, то можно воспользоваться аннотацией `@Query`.

Листинги кода классов репозитория для всех сущностей расположены в приложении Д, а именно для репозитория рабочего объекта (см. Приложение Д, листинг 1), вентиляционной системы (см. Приложение Д, листинг 2), точки измерения (см. Приложение Д, листинг 3), измерений (см. Приложение Д, листинг 4).

Далее после сервисов и репозитория нам необходимо получить класс контроллер, который будет отвечать за получение и обработку входящих

HTTP-запросов и формировать ответ. Для этого необходимо написать класс и пометить его соответствующей аннотацией `@Controller`. Также в нашем случае мы еще добавим аннотацию Lombok `@AllArgsConstructor` для формирования конструктора и еще `@RequestMapping`, которая указывает путь сопоставления HTTP-запросов с методами.

Листинг 8. Аннотации класса `CommissioningController`:

Аннотации класса `CommissioningController`

```
@Controller
@AllArgsConstructor
@RequestMapping("/commissioning")
```

Далее для полноценной работы необходимо добавить зафиналенные поля сервисов и dto маппер.

Листинг 9. Инициализация полей `CommissioningController`:

Инициализация полей `CommissioningController`

```
private final PlaceOfWorkService placeOfWorkService;
private final VentilationSystemService ventilationSystemService;
private final PointService pointService;
private final MeasurementsService measurementsService;
private final DtoMapper dtoMapper;
```

Первое что необходимо сделать, это функционал с рабочими объектами, а именно получение всех объектов, добавление объектов, удаление объекта, обновления данных об объекте. Соответственно функция получения всех объектов будет возвращать макет страницы со всеми рабочими объектами (см. Приложение Е, листинг 1). Добавление объекта разбивается на две функции, одна будет с GET запросом, который будет выводить страницу добавления нового объекта, другая с POST, которая отправляет объект в базу данных (см. Приложение Е, листинг 2). Для удаления объекта также будет две функции, но обе с GET запросами. Одна из функций удаляет объект, если он был добавлен случайно и не имеет дальнейших зависимостей, а вторая будет форсировано удалять объект и все его зависимости (см. Приложение Е, листинг 3). И также для обновления объекта будет две функции с GET запросом для вывода

страницы обновления значений объекта и с POST запросом для отправления обновленного объекта в базу данных (см. Приложение Е, листинг 4).

Далее необходимо по аналогии с данным кодом для каждой сущности написать соответствующие функции. Соответственно в данных функциях меняются сущности, путь запроса и добавляются переменные.

3.3 Составление и написание шаблонов HTML страниц

Для начала необходимо сделать страницу, на которой будут расположены рабочие объекты. Для этого сделаем некоторый макет, на котором будут расположены в два столбца объекты. В каждом объекте будет добавлено название, адрес и соответственно три кнопки с просмотром, изменением и удалением объекта. Код данной страницы можно посмотреть на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/object_main_page.html.

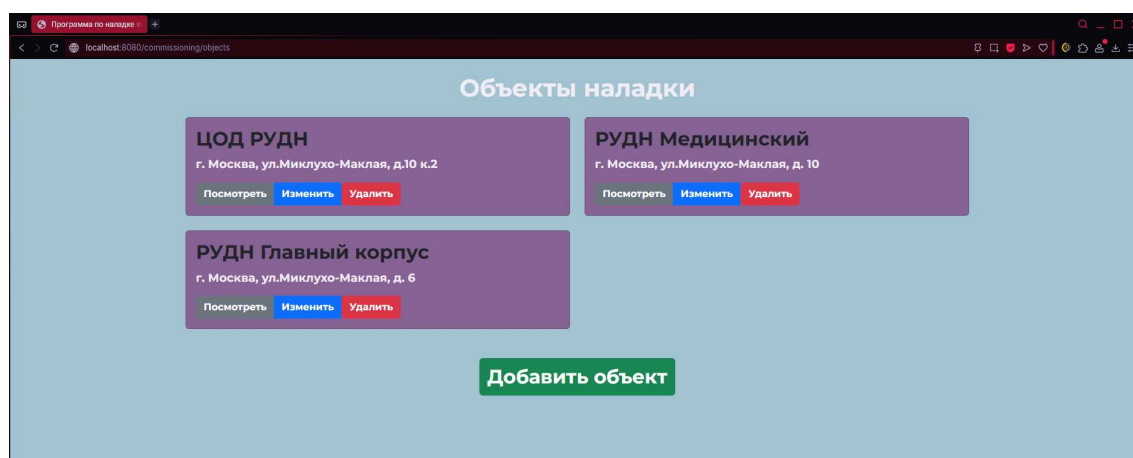


Рисунок 3.4 – Страница с объектами

Следующим шагом необходимо сделать страницу с добавлением нового объекта, на которой должно быть поле ввода названия нового объекта, его адреса, кнопка добавления объекта и кнопка вернуться назад. Код данной страницы можно также посмотреть на github по ссылке ниже: https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/object_add_page.html.

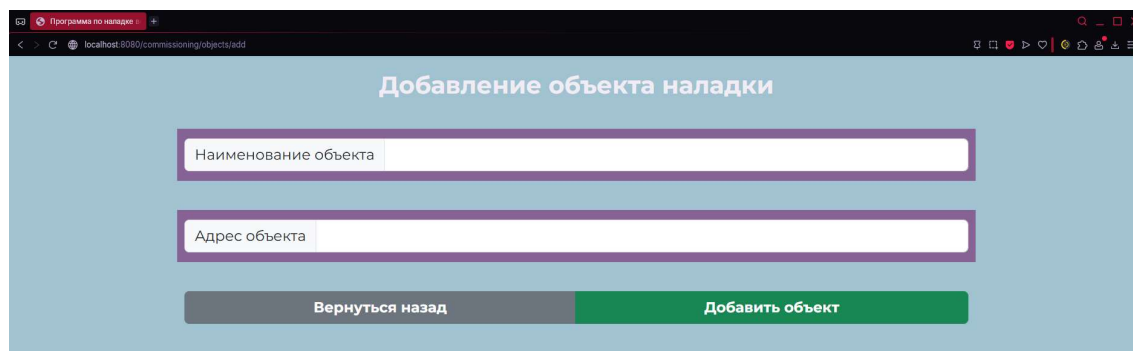


Рисунок 3.5 – Страница с добавлением объекта

Кнопка удаления объекта просто удаляет объект из базы данных и перенаправляет обратно на страницу с объектами, но если объект имеет под собой заполненные вентиляционные системы, то всплывает модальное окно, в котором необходимо подтвердить свой выбор.

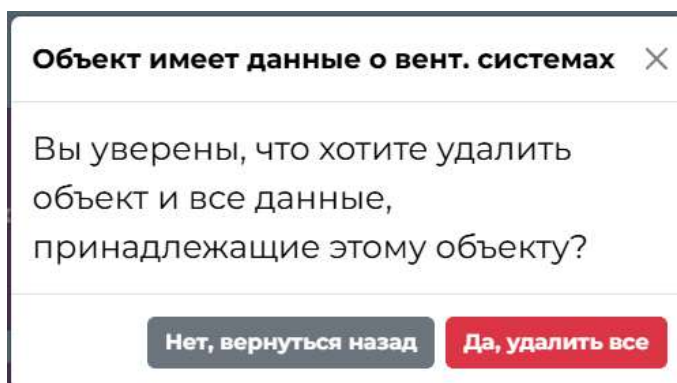


Рисунок 3.6 – Модальное окно удаления объекта

Далее идет кнопка изменения объекта, которая перенаправляет на страницу с изменением значений данного объекта. На данной странице можно заменить поля наименование объекта и адрес объекта. Код страницы на github: https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/object_update_page.html.

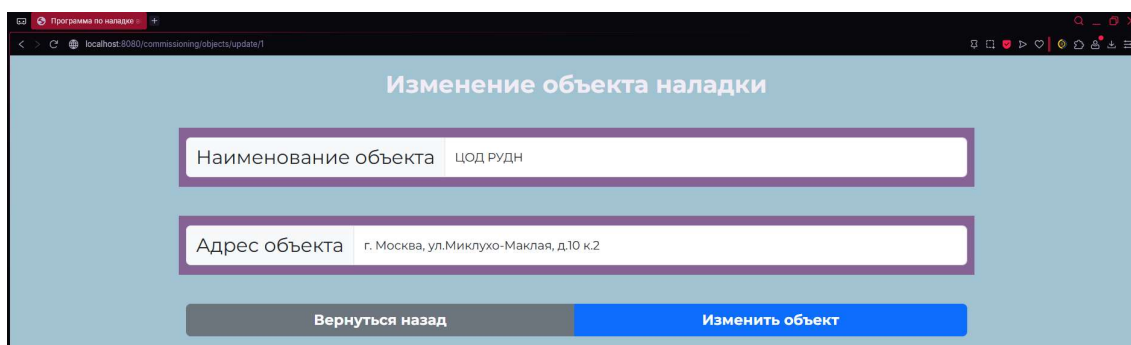


Рисунок 3.7 – Страница изменения объекта

Далее соответственно у нас идет кнопка «Посмотреть», которая ведет на страницу с просмотром объекта, на которой расположены вентиляционные системы, которые принадлежат данному объекту, кнопка «Вернуться к списку объектов» и кнопка «Добавить вентиляционную систему». Код данной страницы можно посмотреть на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/ventsystem_main_page.html.

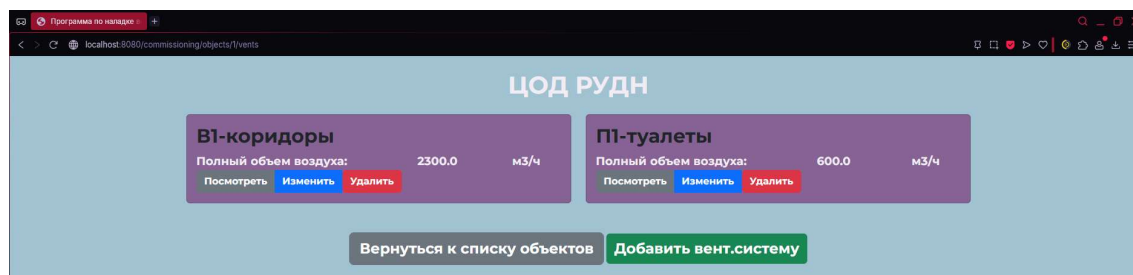


Рисунок 3.8 – Страница просмотра объекта

Кнопка добавить вентиляционную систему переносит нас на страницу, на которой расположены поля формы ввода наименования вентиляционной системы и полного расхода воздуха, также кнопка «Вернуться назад», которая возвращает на страницу вент системы, и кнопка «Добавить вент. систему». Код страницы на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/ventsystem_add_page.html.

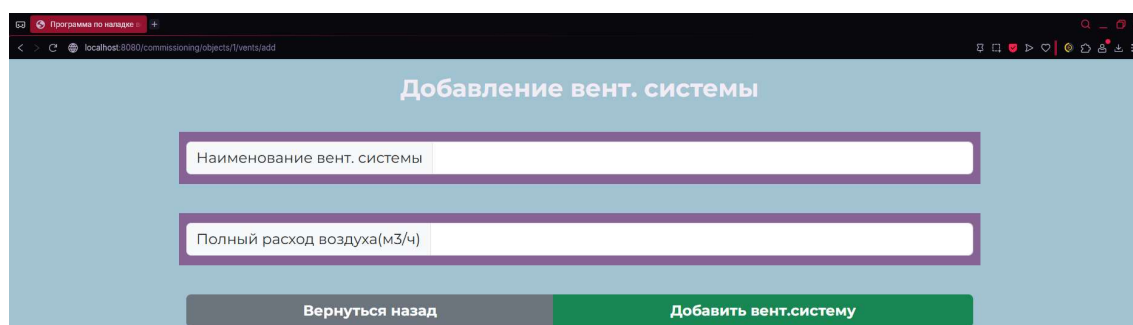


Рисунок 3.9 – Страница добавления вентиляционной системы

Далее идет кнопка изменения вентиляционной системы, которая направляет на страницу, где можно изменить данные вентиляционной системы. Соответственно можно изменить данные наименования и полного

расхода воздуха и применить изменения или же вернуться назад. Код этой страницы на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/ventsystem_update_page.html.

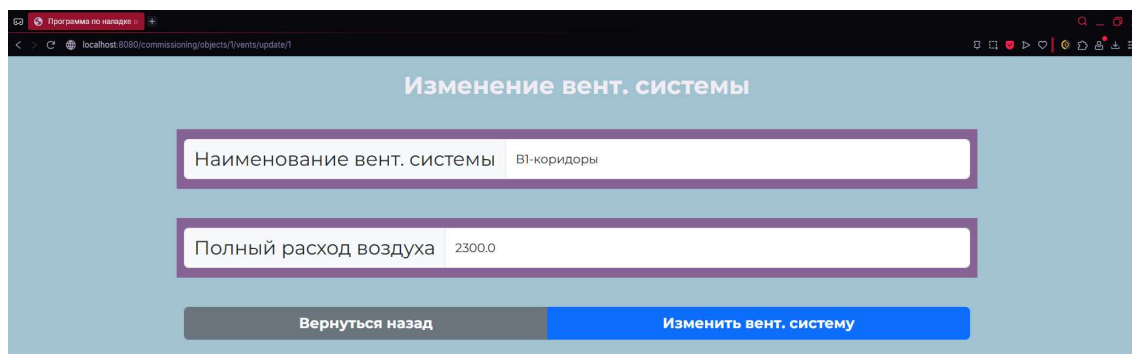


Рисунок 3.10 – Страница изменения вентиляционной системы

Кнопка удаления вентиляционной системы выполняет функцию удаления системы из базы данных, но в случае, когда вентиляционная система имеет данные о точках измерения, то будет предложено модальное окно, в котором можно подтвердить выбор или вернуться назад.

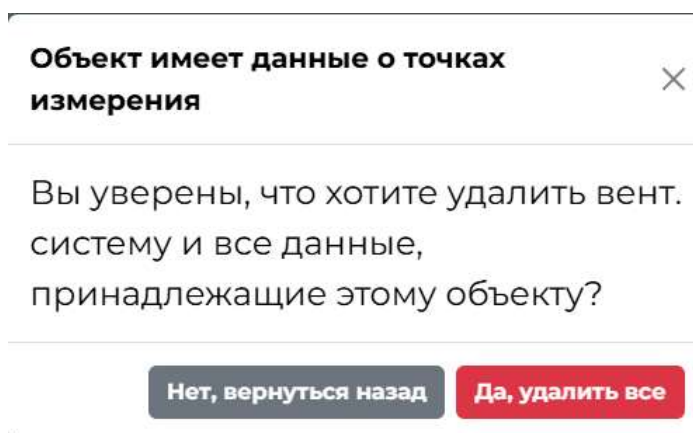


Рисунок 3.11 – Модальное окно удаления вентиляционной системы

Далее следует кнопка просмотра вентиляционной системы, которая переносит на страницу с списком точек измерений данной вентиляционной системы. У каждой точки измерения выводится название, проектные расход и скорость потока воздуха, также если были добавлены измерения, то и отображаются текущие расход и скорость потока воздуха. Если все добавлено, то еще показывается % расхождения данных в данной точке, который становится зеленым если данный процент находится в допуске. Также

присутствуют кнопки добавить точку измерения и вернуться к списку вентиляционных систем. У каждой точки соответственно присутствует кнопки «Удалить», «Изменить» и «Посмотреть т.и. целиком». Код на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/point_main_page.html.

Рисунок 3.12 – Страница точек измерения вентиляционной системы

При нажатии кнопки «Добавить точку измерения» происходит переход на страницу добавления точки, на которой необходимо заполнить все необходимые поля. На данной странице, при вводе размеров, можно автоматически рассчитать площадь сечения с помощью специальной кнопки, а также, при вводе объема воздуха, можно будет рассчитать скорость потока воздуха. И соответственно далее кнопки «Добавить точку измерения» и «Вернуться назад». Код страницы на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/point_add_page.html.

Рисунок 3.13 – Страница добавления точек изменения вентиляционной системы

Для добавления измерения к точке необходимо нажать кнопку «Добавить измерение», которое перенесет на страницу с добавлением измерения, на которой необходимо ввести значение измерения и, при желании, можно ввести примечание к измерению. Код страницы на github: https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/measure_add_page.html.

Рисунок 3.14 – Страница добавления измерения точки

Далее идет кнопка «Изменить точку измерения», которая переносит на страницу с изменением параметров точки. На данной странице можно изменить поля, которые не подсвечиваются красным и соответственно либо сохранить изменение, либо вернуться назад. Код страницы на github: https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/s/point_update_page.html.

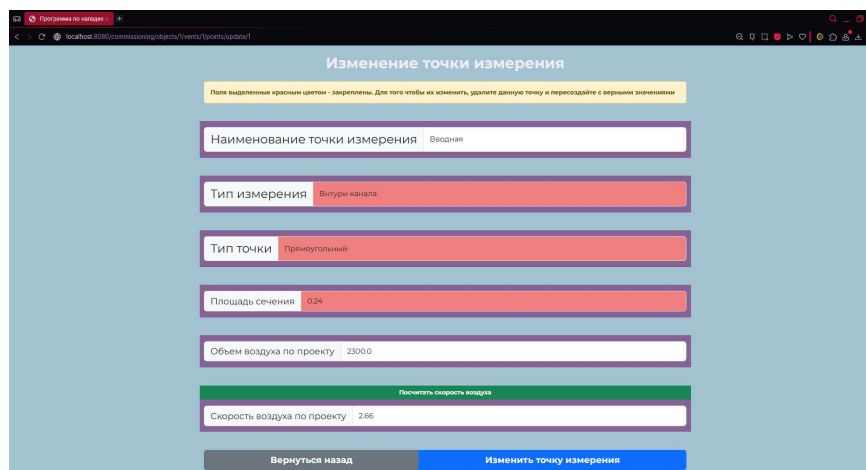


Рисунок 3.15 – Страница изменения точки измерения системы

Кнопка удалить точку измерения, при отсутствии сохраненных измерений, удаляет точку из БД, иначе открывает модальное окно с подтверждением выбора.

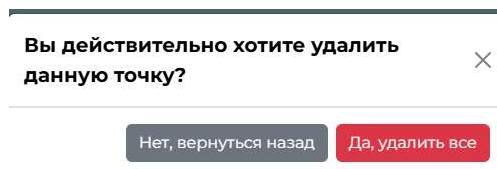


Рисунок 3.16 – Модальное окно удаления точки измерения

Далее кнопка «Просмотреть т.и. целиком», которая переносит на страницу просмотра точки измерения на которой показывает всю информацию о данной точке измерения и показывает список измерений, которые были произведены ранее. Код страницы на github:

https://github.com/lpand4/Graduate_work/blob/master/src/main/resources/templates/point_page.html.

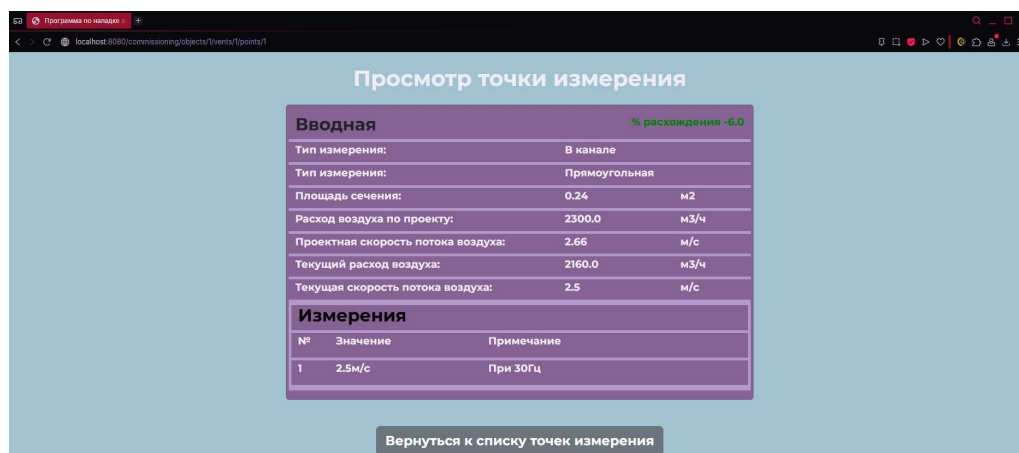


Рисунок 3.17 – Страница просмотра точки измерения

3.4 Подготовка и вывод приложения для работы в контейнере

Для полноценной работы приложения необходимо подготовить конфигурации приложения, также настроить миграции баз данных и подготовить к работе в контейнере. Начнем с liquibase, для инициализации таблиц необходимо в папке changesets создать файл inittable.sql, в котором будут SQL-запросы для инициализации таблиц.

Листинг 10. Инициализация таблиц:

Инициализация таблиц inittable.sql

```
1  --Создание таблицы объектов в случае ее отсутствия
2  CREATE TABLE IF NOT EXISTS place_of_work
3  (
4      object_id BIGSERIAL PRIMARY KEY,
5      name_of_object VARCHAR(255) NOT NULL,
6      address_of_object VARCHAR(255)
7  );
8  --Создание таблицы вентиляционных систем в случае ее отсутствия
9  CREATE TABLE IF NOT EXISTS ventilation_system
10 (
11     ventilation_system_id BIGSERIAL PRIMARY KEY,
12     object_id INT,
13     FOREIGN KEY (object_id) REFERENCES place_of_work,
14     name_of_system VARCHAR(255) NOT NULL,
15     full_air_volume INT NOT NULL
16 );
17 --Создание таблицы точек измерения системы в случае ее отсутствия
18 CREATE TABLE IF NOT EXISTS points
19 (
20     point_id BIGSERIAL PRIMARY KEY,
21     ventilation_system_id INT,
22     FOREIGN KEY (ventilation_system_id) REFERENCES ventilation_system ,
23     name_of_point VARCHAR(255) NOT NULL,
24     type_measuring VARCHAR(255) NOT NULL,
25     type_of_hole VARCHAR(255) NOT NULL,
26     cross_sectional_area DECIMAL NOT NULL,
27     air_volume DECIMAL NOT NULL,
28     air_flow_rate DECIMAL NOT NULL,
29     current_air_flow_rate DECIMAL,
30     current_air_volume DECIMAL,
31     discrepancy DECIMAL
32 );
33 --Создание таблицы замеров в случае ее отсутствия
34 CREATE TABLE IF NOT EXISTS measurements
35 (
36     measurement_id BIGSERIAL PRIMARY KEY,
37     point_id INT,
38     FOREIGN KEY (point_id) REFERENCES points,
39     note VARCHAR(255),
40     value_of_measure DECIMAL NOT NULL
41 );
```

Далее для удобства пользования добавим еще changeset, который будет вставлять данные insertdata.sql с которыми можно будет поработать.

Листинг 11. Инициализация данных в таблицы:

Инициализация данных в таблице insertdata.sql

```
1 insert into place_of_work (name_of_object, address_of_object)
2 values ('ЦОД РУДН', 'г. Москва, ул.Миклухо-Маклая, д.10 к.2'),
3        ('РУДН Медицинский', 'г. Москва, ул.Миклухо-Маклая, д. 10'),
4        ('РУДН Главный корпус', 'г. Москва, ул.Миклухо-Маклая, д. 6')
5 ON CONFLICT DO NOTHING;
6
7 insert into ventilation_system(object_id, name_of_system, full_air_volume)
8 values (1, 'В1-коридоры', 2300),
9        (1, 'П1-туалеты', 600),
10       (2, 'В1-зал', 1600),
11       (2, 'П1-зал', 1600),
12       (2, 'В2-туалеты', 600),
13       (3, 'В1', 600),
14       (3, 'П1', 600)
15 ON CONFLICT DO NOTHING;
16
17 insert into points( ventilation_system_id, name_of_point, type_measuring,
18                    type_of_hole, cross_sectional_area, air_volume, air_flow_rate)
19 values ( 1, 'Вводная', 'INSIDE_THE_DUCT', 'RECTANGULAR', 0.24, 2300, 2.66),
20       ( 2, 'Вводная', 'INSIDE_THE_DUCT', 'CIRCULAR', 0.0491, 600, 3.39),
21       ( 3, 'Вводная', 'INSIDE_THE_DUCT', 'RECTANGULAR', 0.16, 1600, 2.78),
22       ( 4, 'Вводная', 'INSIDE_THE_DUCT', 'RECTANGULAR', 0.16, 1600, 2.78),
23       ( 5, 'Вводная', 'INSIDE_THE_DUCT', 'RECTANGULAR', 0.09, 600, 1.85),
24       ( 6, 'Вводная', 'INSIDE_THE_DUCT', 'CIRCULAR', 0.0314, 600, 5.31),
25       ( 7, 'Вводная', 'INSIDE_THE_DUCT', 'CIRCULAR', 0.0314, 600, 5.31)
26 ON CONFLICT DO NOTHING;
```

Для того чтобы данные файлы были активны, в папке liquibase необходимо создать файл db.changelog.yml в котором пропишем путь к ним.

Листинг 12. Конфигурация liquibase:

Конфигурация db.changelog.yml

```
1 databaseChangeLog:
2   - includeAll:
3     path: /changesets
4     relativeToChangelogFile: true
```

Также в папку preliquibase добавим файл, который будет создавать схему, если она не будет создана.

Листинг 13. Инициализация схемы:

Инициализация схемы postgresql.sql

```
1 CREATE SCHEMA IF NOT EXISTS commissioning_of_ventilation_system;
```

Далее рассмотрим конфигурацию приложения, которая находится в файле application.yml. В данной конфигурации будет указано наименование приложения, параметры для подключения к базе данных и, соответственно, параметры для liquibase.

Листинг 14. Конфигурация приложения:

Конфигурация application.yml

```
1  spring:
2    application:
3      name: CommissioningOfVentilationSystems
4    datasource:
5      url: jdbc:postgresql://localhost:5432/commissioning_of_ventilation_system
6      username: postgres
7      password: password
8      driver-class-name: org.postgresql.Driver
9    jpa:
10     hibernate:
11       ddl-auto: update
12       show-sql: true
13     liquibase:
14       change-log: classpath:liquibase/db.changelog.yml
15       enabled: true
16       default-schema: commissioning_of_ventilation_system
```

После настройки всех конфигураций, необходимо сделать, чтобы можно было приложение вынести на сервер. Для этого необходимо собрать приложение, чтобы получить jar файл, который представляет собой Java-архив. Далее, когда все успешно собрано, создаем Dockerfile в котором опишем запуск нашего приложения в контейнере. Для этого соответственно воспользуемся образом, который поддерживает Java, далее необходимо скопировать собранный jar файл в контейнер, добавить переменные для подключения в БД и, соответственно, запустить приложение внутри контейнера.

Листинг 15. Конфигурация контейнера:

Конфигурация Dockerfile

```
1 FROM openjdk:19-slim
2
3 COPY target/*.jar /app/commissioning.jar
4
5 ENV POSTGRES_DB commissioning_of_ventilation_system
6 ENV POSTGRES_USER postgres
7 ENV POSTGRES_PASSWORD password
8
9 CMD ["java", "-jar", "/app/commissioning.jar"]
```

Для того, чтобы приложение полноценно выполняло свою функцию, сделаем чтобы запускались сразу два контейнера: один контейнер с приложением и один контейнер с базой данных. Для этого необходимо создать `docker-compose.yml` файл, который является конфигурацией нашего многоконтейнерного приложения. В нем соответственно указывается конфигурация нашего приложения, его контекст, который представляет собой `Dockerfile`, переменные для подключения к базе данных и порты. Кроме того указывается конфигурация контейнера с базой данных, а именно образ и переменные для подключения.

Листинг 16. Конфигурация многоконтейнерного приложения:

Конфигурация docker-compose.yml

```
1 services:
2   app:
3     build:
4       context: .
5     environment:
6       - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/commissioning_of_ventilation_system
7       - SPRING_DATASOURCE_USERNAME=postgres
8       - SPRING_DATASOURCE_PASSWORD=password
9     ports:
10      - "8080:8080"
11     depends_on:
12      - db
13   db:
14     image: postgres:13.2-alpine
15     environment:
16       POSTGRES_DB: commissioning_of_ventilation_system
17       POSTGRES_USER: postgres
18       POSTGRES_PASSWORD: password
```

ЗАКЛЮЧЕНИЕ

Подводя итоги, мы получили веб-приложение для наладки вентиляционных систем, которое можно выгрузить на сервер и использовать для работы. В рамках работы были поставлены и решены следующие задачи:

1. Рассмотрение основных сведений о вентиляционных системах и способах их наладки;
2. выбор технологий для создания веб-приложения по наладке вентиляционных систем;
3. выбор и описание основных сущностей и их атрибутов для дальнейшей разработки приложения;
4. написание кода для основных и вспомогательных классов программы;
5. написание шаблонов HTML страниц;
6. вывод приложения для полноценной самостоятельной работы в контейнере.

По результатам решения поставленных задач можно сделать следующие выводы:

1. Вентиляционные системы появились еще в древнем Египте, все это время технологии развивались и на данный момент есть несколько различных видов систем, таких как естественная, принудительная, рекуперационная и приточно-вытяжная варианты вентиляции. Наладка вентиляционных систем делится на несколько этапов в которые входит: подготовка к пусконаладке, измерение и настройка основных параметров системы, настройка автоматических систем, проверка на герметичность, пусконаладочные испытания, регулировка и вывод на рабочий режим и составление актов.

2. Архитектура веб-приложения играет большую роль на разработку, так как влияет на некоторые факторы, например, такие как стоимость разработки и поддержания проекта, скорость разработки и модификации

приложения, производительность и безопасность. В основном приложение делится на три слоя, каждый из которых отвечает за определенную задачу, а именно первый слой является презентационным, он отвечает за взаимодействие с пользователями и отображение данных, после чего следует логический слой, который включает в себя бизнес-логику приложения, обрабатывает полученные данные и взаимодействует со слоем данных и, соответственно, следующим компонентом является слой доступа к данным, который непосредственно взаимодействует с базами данных.

3. Для веб-приложения по наладке вентиляционных систем были выделены такие сущности как рабочий объект, вентиляционная система, точка измерения и измерение, к каждой из которых были подобраны соответствующие атрибуты. Соответственно были написаны подходящие классы для данных сущностей, вспомогательные классы нумераторы, классы репозитории, классы сервисы и класс контроллер.

В итоге работы получено веб-приложение для наладки вентиляционных систем, которое можно найти в репозитории на github по ссылке https://github.com/lpand4/Graduate_work и далее есть возможность расположить его на сервере и использовать в пуско-наладочных работах.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- [1] Е. Белова, «Центральные системы кондиционирования воздуха в зданиях», Москва: Евроклимат, 2006.
- [2] *ГОСТ 12.3.018-79 «Система стандартов безопасности труда. Системы вентиляционные. Методы аэродинамических испытаний».*
- [3] *ГОСТ 34060-2017 «Инженерные сети зданий и сооружений внутренние. Испытание и наладка систем вентиляции и кондиционирования воздуха».*
- [4] *ГОСТ Р 53300-2009 «Противодымная защита зданий и сооружений. Методы приемосдаточных и периодических испытаний (с Изменением N 1)».*
- [5] «Spring Framework Documentation,» [В Интернете]. Available: <https://docs.spring.io/spring-framework/reference/index.html>.
- [6] «Документация Bootstrap,» [В Интернете]. Available: <https://getbootstrap.ru/docs/5.3/getting-started/introduction/>.
- [7] «Документация PostgreSQL и Postgres Pro,» [В Интернете]. Available: <https://postgrespro.ru/docs/>.
- [8] «Документация Thymeleaf,» [В Интернете]. Available: <https://www.thymeleaf.org/documentation.html>.
- [9] «Документация Java,» [В Интернете]. Available: <https://docs.oracle.com/en/java/>.
- [10] М. Фаулер, Архитектура корпоративных программных приложений, Вильямс, 2007.

ПРИЛОЖЕНИЕ А

Листинг 1. Класс точки измерения:

Структура класса точки измерения

```
15 @Table(name = "points")
16 public class Point {
17     /**
18      * Уникальное айди точки измерения
19      */
20     @Id
21     @GeneratedValue(strategy = GenerationType.IDENTITY)
22     private Long pointId;
23     /**
24      * Вентиляционная система к которой принадлежит точка измерения
25      */
26     @ManyToOne @JoinColumn(name = "ventilation_system_id")
27     private VentilationSystem ventilationSystem;
28     /**
29      * Название точки измерения (Например Этаж 1, т. 1)
30      */
31     private String nameOfPoint;
32     /**
33      * Тип измерения (На решетке или в канале)
34      */
35     @Enumerated(EnumType.STRING)
36     private TypeMeasuring typeMeasuring;
37     /**
38      * Тип отверстия измерения (Круглый воздуховод/решетка или прямоугольный)
39      */
40     @Enumerated(EnumType.STRING)
41     private TypeOfHole typeOfHole;
42     /**
43      * Площадь сечения точки измерения (Если круглый, то  $(3,14 * D^2) / 4$ ,
44      * если прямоугольный, то  $W * H$ )
45      */
46     private Double crossSectionalArea;
47     /**
48      * Объем воздуха, который должен быть на данной точке по проекту
49      */
50     private Double airVolume;
51     /**
52      * Скорость потока воздуха в точке измерения по проекту
53      */
54     private Double airFlowRate;
55     /**
56      * Текущая скорость потока воздуха в данной точке
57      */
58     private Double currentAirFlowRate;
59     /**
60      * Текущий объем воздуха
61      */
62     private Double currentAirVolume;
63     /**
64      * Расхождение проектного от текущего
65      */
66     private Double discrepancy;
67     /**
68      * Измерения в данной точке,
69      * которые могли происходить при разных условиях
70      */
71     @OneToMany(mappedBy = "pointId")
72     private List<Measurements> listAirFlowRate;
73 }
```

ПРИЛОЖЕНИЕ Б

Листинг 1. Класс измерения:

Структура класса измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.domain.mainModels;
2
3 import jakarta.persistence.*;
4 import lombok.Data;
5 /**
6  * Измерения, которые происходят при разных условиях
7  */
8 @Entity
9 @Data
10 @Table(name = "measurements")
11 public class Measurements {
12     /**
13      * Уникальное айди измерения
14      */
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     @Column(name = "measurement_id")
18     private Long measurementsId;
19     /**
20      * Точка измерения, к которое принадлежит определенное измерение
21      */
22     @ManyToOne @JoinColumn(name = "point_id")
23     private Point pointId;
24     /**
25      * Примечание к измерению
26      */
27     private String note;
28     /**
29      * Значение скорости потока воздуха при данных условиях
30      */
31     private Double valueOfMeasure;
32 }
```

ПРИЛОЖЕНИЕ В

Листинг 1. Класс dto вентиляционной системы:

Структура класса dto вентиляционной системы

```
1 package ru.pugovkinv.commissioningofventilationsystems.dto;
2
3
4 import jakarta.validation.constraints.NotEmpty;
5 import jakarta.validation.constraints.Pattern;
6 import lombok.Data;
7 import domain.mainModels.PlaceOfWork;
8 import domain.mainModels.Point;
9
10 import java.util.List;
11
12 @Data
13 public class VentilationSystemDto {
14     /**
15      * Уникальное айди вентиляционной системы
16      */
17     private Long ventilationSystemId;
18     /**
19      * Название системы (Например, В1 - коридоры корпуса А)
20      */
21     @NotEmpty(message = "Название системы не должно быть пустым!")
22     private String nameOfSystem;
23     /**
24      * Полный расход объема воздуха который система должна выдавать по проекту
25      */
26     @NotEmpty(message = "Полный расход воздуха системы не должно быть пустым!")
27     @Pattern(regex = "^-?(0|[1-9][0-9]*) (?:[.]\d+|)$",
28             message = "Полный расход воздуха системы должен иметь числовое значение," +
29                     " и, если оно не целое, то запись через точку!")
30     private String fullAirVolume;
31     /**
32      * Список точек измерений, которые относятся к данной системе
33      */
34     private List<Point> pointsOfSystem;
35     /**
36      * Рабочий объект на котором находится данная система (Например, РУДН)
37      */
38     private PlaceOfWork placeOfWork;
39 }
```


ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В

Листинг 2. Класс dto точки измерения:

Структура класса dto точки измерения

```
13 @Data
14 public class PointDto {
15     /**
16      * Уникальное айди точки измерения
17      */
18     private Long pointId;
19     /**
20      * Вентиляционная система к которой принадлежит точка измерения
21      */
22     private VentilationSystem ventilationSystem;
23     /**
24      * Название точки измерения (Например Этаж 1, т. 1)
25      */
26     @NotEmpty(message = "Название точки измерения не должно быть пустым!")
27     private String nameOfPoint;
28     /**
29      * Тип измерения (На решетке или в канале)
30      */
31     private TypeMeasuring typeMeasuring;
32     /**
33      * Тип отверстия измерения (Круглый воздуховод/решетка или прямоугольный)
34      */
35     private TypeOfHole typeOfHole;
36     /**
37      * Площадь сечения точки измерения (Если круглый, то  $(3,14 * D^2) / 4$ , если прямоугольный, то  $W * H$ )
38      */
39     @NotEmpty(message = "Название точки измерения не должно быть пустым!")
40     private String crossSectionalArea;
41     /**
42      * Объем воздуха, который должен быть на данной точке по проекту
43      */
44     @NotEmpty(message = "Объем воздуха точки измерения не должен быть пустым!")
45     @Pattern(regexp = "^~?(0|[1-9][0-9]*) (?:[.]\\d+)?$ ", message = "Объем воздуха точки измерения должен иметь числовое значение," +
46         " и, если оно не целое, то запись через точку!")
47     private String airVolume;
48     /**
49      * Скорость потока воздуха в точке измерения по проекту
50      */
51     @NotEmpty(message = "Скорость потока воздуха точки измерения не должен быть пустым!")
52     @Pattern(regexp = "^~?(0|[1-9][0-9]*) (?:[.]\\d+)?$ ", message = "Скорость потока воздуха точки измерения должен иметь числовое значение," +
53         " и, если оно не целое, то запись через точку!")
54     private String airFlowRate;
55     /**
56      * Текущая скорость потока воздуха в данной точке
57      */
58     private Double currentAirFlowRate;
59     /**
60      * Текущий объем воздуха
61      */
62     private Double currentAirVolume;
63     /**
64      * Расхождение проектного от текущего
65      */
66     private Double discrepancy;
67     /**
68      * Измерения в данной точке, которые могли происходить при разных условиях
69      */
70     private List<Measurements> listAirFlowRate;
71 }
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В

Листинг 3. Класс dto измерения:

Структура класса dto измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.dto;
2
3
4 import jakarta.validation.constraints.NotEmpty;
5 import jakarta.validation.constraints.Pattern;
6 import lombok.Data;
7 import domain.mainModels.Point;
8
9 @Data
10 public class MeasurementsDto {
11     /**
12      * Уникальное айди измерения
13      */
14     private Long measurementsId;
15     /**
16      * Точка измерения, к которое принадлежит определенное измерение
17      */
18     private Point pointId;
19     /**
20      * Примечание к измерению
21      */
22     private String note;
23     /**
24      * Значение скорости потока воздуха при данных условиях
25      */
26     @NotEmpty(message = "Значение измерения не должно быть пустым!")
27     @Pattern(regex = "^-?(0|[1-9][0-9]*)?(:[.]\\d+)?$ ",
28             message = "Значение измерения должно быть числовым," +
29                     " и, если оно не целое, то запись через точку!")
30     public String valueOfMeasure;
31
32 }
```

ПРИЛОЖЕНИЕ Г

Листинг 1. Класс сервис рабочего объекта:

Структура класса сервиса рабочего объекта

```
1 package ru.pugovkinv.commissioningofventilationsystems.services;
2 import lombok.RequiredArgsConstructor;
3 import org.springframework.stereotype.Service;
4 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.PlaceOfWork;
5 import ru.pugovkinv.commissioningofventilationsystems.repository.PlaceOfWorkRepository;
6 import java.util.List;
7 import java.util.Optional;
8 /**
9  * Сервис работы с рабочими объектами
10 */
11 @Service
12 @RequiredArgsConstructor
13 public class PlaceOfWorkService {
14     /**
15      * Репозиторий объектов
16      */
17     private final PlaceOfWorkRepository placeOfWorkRepository;
18     /**
19      * Поиск всех объектов
20      * @return Лист объектов
21      */
22     public List<PlaceOfWork> findAll(){
23         return placeOfWorkRepository.findAll();
24     }
25     /**
26      * Поиск объекта по айди
27      * @param id айди
28      * @return объект найденный по айди
29      */
30     public Optional<PlaceOfWork> findById(Long id){
31         return placeOfWorkRepository.findById(id);
32     }
33     /**
34      * Добавление объекта в базу данных
35      * @param placeOfWork новый объект
36      */
37     public void save(PlaceOfWork placeOfWork){
38         placeOfWorkRepository.save(placeOfWork);
39     }
40     /**
41      * Удаление объекта по айди
42      * @param id айди
43      */
44     public void deleteById(Long id){
45         placeOfWorkRepository.deleteById(id);
46     }
47     /**
48      * Обновление объекта
49      * @param placeOfWork обновленный объект
50      */
51     public void updatePlaceOfWork(PlaceOfWork placeOfWork){
52         placeOfWorkRepository.save(placeOfWork);
53     }
54 }
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г

Листинг 2. Класс сервис вентиляционной системы:

Структура класса сервиса вентиляционной системы

```
1 package ru.pugovkinv.commissioningofventilationsystems.services;
2 import lombok.RequiredArgsConstructor;
3 import org.springframework.stereotype.Service;
4 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.PlaceOfWork;
5 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.VentilationSystem;
6 import ru.pugovkinv.commissioningofventilationsystems.repository.VentilationSystemRepository;
7 import java.util.List;
8 import java.util.Optional;
9 /**
10  * Сервис работы с вентиляционными системами
11  */
12 @Service
13 @RequiredArgsConstructor
14 public class VentilationSystemService {
15     /**
16      * Репозиторий вентиляционных систем
17      */
18     private final VentilationSystemRepository ventilationSystemRepository;
19     /**
20      * Поиск всех вентиляционных систем
21      * @return Все вентиляционные системы
22      */
23     public List<VentilationSystem> findAll(PlaceOfWork placeOfWork) {
24         return ventilationSystemRepository.findAllByPlaceOfWork(placeOfWork);
25     }
26     /**
27      * Поиск вентиляционной системы по айди
28      * @param id айди
29      * @return нужная вентиляционная система
30      */
31     public Optional<VentilationSystem> findById(Long id) {
32         return ventilationSystemRepository.findById(id);
33     }
34     /**
35      * Сохранение вентиляционной системы
36      * @param ventilationSystem новая вентиляционная система
37      */
38     public void save(VentilationSystem ventilationSystem) {
39         ventilationSystemRepository.save(ventilationSystem);
40     }
41     /**
42      * Удаление вентиляционной системы по айди
43      * @param id айди
44      */
45     public void deleteById(Long id) {
46         ventilationSystemRepository.deleteById(id);
47     }
48     /**
49      * Обновление вент. системы
50      * @param ventilationSystem обновленная вент. система
51      */
52     public void updateVentilationSystem(VentilationSystem ventilationSystem) {
53         ventilationSystemRepository.save(ventilationSystem);
54     }
55 }
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г

Листинг 3. Класс сервис точки измерения:

Структура класса сервиса точки измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.services;
2
3 import lombok.RequiredArgsConstructor;
4 import org.springframework.stereotype.Service;
5 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.Point;
6 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.VentilationSystem;
7 import ru.pugovkinv.commissioningofventilationsystems.repository.PointRepository;
8
9 import java.util.List;
10 import java.util.Optional;
11
12 @Service
13 @RequiredArgsConstructor
14 public class PointService {
15     /**
16      * Репозиторий точек измерения
17      */
18     private final PointRepository pointRepository;
19     /**
20      * Поиск всех точек измерения, которые принадлежат вентиляционной системе
21      * @param ventilationSystem необходимая вентиляционная система
22      * @return все точки измерения
23      */
24     public List<Point> findAll(VentilationSystem ventilationSystem) {
25         return pointRepository.findAllByVentilationSystem(ventilationSystem);
26     }
27     /**
28      * Поиск точки измерения по айди
29      * @param id необходимое айди
30      * @return нужная точка измерения
31      */
32     public Optional<Point> findById(Long id) { return pointRepository.findById(id); }
33     /**
34      * Сохранение точки измерения
35      * @param point новая точка измерения
36      * @return точка измерения
37      */
38     public Point save(Point point) { return pointRepository.save(point); }
39     /**
40      * Удаление точки измерения по айди
41      * @param id нужное айди
42      */
43     public void deleteById(Long id) { pointRepository.deleteById(id); }
44     /**
45      * Обновление точки измерения
46      * @param point обновленная точка измерения
47      */
48     public void updatePoint(Point point) {
49         pointRepository.save(point);
50     }
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г

Листинг 4. Класс сервис измерения:

Структура класса сервиса измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.services;
2
3 import lombok.AllArgsConstructor;
4 import lombok.RequiredArgsConstructor;
5 import org.springframework.stereotype.Service;
6 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.Measurements;
7 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.Point;
8 import ru.pugovkinv.commissioningofventilationsystems.repository.MeasurementsRepository;
9 import ru.pugovkinv.commissioningofventilationsystems.repository.PointRepository;
10
11 import java.util.List;
12 import java.util.Optional;
13
14 @Service
15 @AllArgsConstructor
16 public class MeasurementsService {
17     /**
18      * Репозиторий измерений в точке
19      */
20     private final MeasurementsRepository measurementsRepository;
21
22     /**
23      * Поиск всех измерений, которые принадлежат точке
24      * @param point нужная точка
25      * @return все измерения
26      */
27     public List<Measurements> findAll(Point point){
28         return measurementsRepository.findAllByPointId(point);
29     }
30     /**
31      * Поиск измерения по его айди
32      * @param id нужное айди
33      * @return искомое измерение
34      */
35     public Optional<Measurements> findById(Long id){
36         return measurementsRepository.findById(id);
37     }
38     /**
39      * Сохранение измерения
40      * @param measurements новое измерение
41      * @return измерение
42      */
43     public Measurements save(Measurements measurements){
44         return measurementsRepository.save(measurements);
45     }
46     /**
47      * Удаление измерения по айди
48      * @param id нужное айди
49      */
50     public void deleteById(Long id){measurementsRepository.deleteById(id);}
51 }
```

ПРИЛОЖЕНИЕ Д

Листинг 1. Класс репозиторий рабочего объекта:

Структура класса репозитория рабочего объекта

```
1 package ru.pugovkinv.commissioningofventilationsystems.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.PlaceOfWork;
6
7 @Repository
8 public interface PlaceOfWorkRepository extends JpaRepository<PlaceOfWork, Long> {
9 }
```

Листинг 2. Класс репозиторий вентиляционной системы:

Структура класса репозитория вентиляционной системы

```
1 package ru.pugovkinv.commissioningofventilationsystems.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.PlaceOfWork;
6 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.VentilationSystem;
7
8 import java.util.List;
9
10 @Repository
11 public interface VentilationSystemRepository extends JpaRepository<VentilationSystem, Long> {
12     List<VentilationSystem> findAllByPlaceOfWork(PlaceOfWork placeOfWork);
13 }
```

Листинг 3. Класс репозиторий точки измерения:

Структура класса репозитория точки измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.Point;
6 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.VentilationSystem;
7
8 import java.util.List;
9
10 @Repository
11 public interface PointRepository extends JpaRepository<Point, Long> {
12     List<Point> findAllByVentilationSystem(VentilationSystem ventilationSystem);
13 }
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Д

Листинг 4. Класс репозиторий измерения:

Структура класса репозитория измерения

```
1 package ru.pugovkinv.commissioningofventilationsystems.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.data.jpa.repository.Modifying;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.stereotype.Repository;
7 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.Measurements;
8 import ru.pugovkinv.commissioningofventilationsystems.domain.mainModels.Point;
9
10 import java.util.List;
11
12 @Repository
13 public interface MeasurementsRepository extends JpaRepository<Measurements, Long> {
14     List<Measurements> findAllByPointId(Point pointId);
15 }
```


ПРИЛОЖЕНИЕ Е

Листинг 1. Получение всех объектов:

Код получения всех объектов и вывод страницы с объектами

```
/**
 * Вывод всех доступных объектов наладки
 *
 * @param model модель
 * @return страница с списком объектов наладки
 */
@GetMapping("/objects")
public String getAllObjects(Model model) {
    List<PlaceOfWork> placeOfWorkList = placeOfWorkService.findAll();
    model.addAttribute("objects", placeOfWorkList.stream()
        .map(dtoMapper::toPlaceOfWorkDto)
        .collect(Collectors.toList()));
    return "object_main_page";
}
```

Листинг 2. Добавление объекта:

Код вывода страницы с добавлением объекта и отправления его в БД

```
/**
 * Страница добавления нового объекта наладки
 *
 * @param model модель
 * @return страница создания нового объекта
 */
@GetMapping("/objects/add")
public String addObject(Model model) {
    model.addAttribute("new_object", new PlaceOfWorkDto());
    return "object_add_page";
}

/**
 * Отправление нового объекта в БД
 *
 * @param newObject новый объект
 * @return возвращает на страницу с списком объектов
 */
@PostMapping("/objects/add")
public String addObject(@ModelAttribute("new_object") @Validated
    PlaceOfWorkDto newObject, BindingResult br) {
    if (br.hasErrors()) {
        return "object_add_page";
    }
    placeOfWorkService.save(dtoMapper.toPlaceOfWork(newObject));
    return "redirect:/commissioning/objects";
}
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е

Листинг 3. Удаление объекта:

Код обычного и форсированного удаления объекта

```
/**
 * Удаление объекта наладки по кнопке
 *
 * @param id айди нужного объекта
 * @return возвращает на страницу с списком объектов
 */
@GetMapping("/objects/delete/{objectId}")
public String deleteObject(@PathVariable("objectId") Long id) {
    placeOfWorkService.deleteById(id);
    return "redirect:/commissioning/objects";
}

/**
 * Форсированное удаление объекта
 *
 * @param objectId айди нужного объекта
 * @return страница с объектами
 */
@GetMapping("/objects/forcedelete/{objectId}")
public String forceDeleteObject(@PathVariable("objectId") Long objectId) {
    PlaceOfWork placeOfWork = placeOfWorkService.findById(objectId).get();
    for (VentilationSystem vent : placeOfWork.getVentilationSystems()) {
        forceDeleteVent(objectId, vent.getVentilationSystemId());
    }
    placeOfWorkService.deleteById(objectId);
    return "redirect:/commissioning/objects";
}
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е

Листинг 4. Обновление объекта:

Код вывода страницы с обновления объекта и отправления его в БД

```
/**
 * Изменение данных объекта наладки
 *
 * @param id айди необходимого объекта
 * @param model модель
 * @return страница изменения объекта наладки
 */
@GetMapping("/objects/update/{objectId}")
public String updateObject(@PathVariable("objectId") Long id, Model model) {
    PlaceOfWork placeOfWorkToUpdate = placeOfWorkService.findById(id).get();
    model.addAttribute("update_object",
        dtoMapper.toPlaceOfWorkDto(placeOfWorkToUpdate));
    return "object_update_page";
}

/**
 * Отправка измененного объекта
 *
 * @param updatedObject измененный объект
 * @return возвращает на страницу с списком объектов
 */
@PostMapping("/objects/update")
public String updateObject(@ModelAttribute("update_object")
    @Validated PlaceOfWorkDto updatedObject,
    BindingResult br) {
    if (br.hasErrors()) {
        return "object_update_page";
    }
    placeOfWorkService.updatePlaceOfWork(dtoMapper.toPlaceOfWork(updatedObject));
    return "redirect:/commissioning/objects";
}
```