



ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ

Государственное бюджетное профессиональное образовательное учреждение города Москвы
КОЛЛЕДЖ АРХИТЕКТУРЫ, ДИЗАЙНА И РЕИНЖИНИРИНГА № 26 (ГБПОУ «26 КАДР»)

ул. Цимлянская, д. 7, стр. 1, Москва, 109559

тел.: +7 (495) 679-47-21, тел.факс: +7 (495) 710-21-03, e-mail: spo-26@edu.mos.ru

ОГРН 1057723001731 ИНН/КПП 7723356160/772301001



**ОТДЕЛЕНИЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ
СТРУКТУРНОЕ ПОДРАЗДЕЛЕНИЕ 10**

Допустить к защите
Заведующий отделения
_____ Селезнева А.В.
_____ 2023 год

КУРСОВАЯ РАБОТА

Создание платформы для управления ресурсами и
эффективностью работы компании.

Специальность 09.02.07 Информационные системы и программирование
код, название

КР.09.02.07.1.41.4

(шифр работы)

Выполнил
студент:

_____	_____	_____
(подпись)	(дата)	(Ф.И.О.)

Проверил
преподаватель:

_____	_____	_____
(подпись)	(дата)	(Ф.И.О.)

Оценка:

Москва 2023г.

Оглавление

Введение	3
Глава 1. Теоретическая часть.....	6
1.1 Понятие информационной системы.....	6
1.2 Виды приложений	6
1.3 Определение клиент-серверной архитектуры	7
1.4 Spring Framework	11
1.5 Spring Boot	12
1.6 Git	13
1.7 Шаблон MVC	15
1.8 База данных.....	15
1.9 PostgreSQL	17
1.10 Spring Data JPA.....	21
1.11 Thymeleaf.....	23
Глава 2. Практическая часть.....	25
2.1Создание проекта	25
2.2 Создание базы данных	26
2.3 Создание объекта и репозитория сотрудника	29
2.4 Создание главной страницы	31
2.5 Создание страницы Сотрудники	33
2.6 Создание формы создания нового сотрудника	35
2.7 Обработка удаления записи из таблицы	36
Заключение.....	39
Список использованной литературы.....	41

Введение

В современном мире, где информационные технологии проникают во все сферы жизни и деятельности, успешное управление компанией становится невозможным без интеграции современных информационных решений. Качественное внедрение информационных технологий в компанию — играет решающую роль в создании и поддержании конкурентоспособности современных организаций.

Информационные технологии переплелись с нашей повседневной жизнью и стали неотъемлемой частью бизнес-процессов. Современная компания в наши дни не может функционировать без активного использования IT-решений. Например, интеграция систем управления предприятием (ERP) в структуру организации позволяет автоматизировать учёт и управление финансами, управление персоналом и логистикой, что существенно повышает эффективность работы компании. Технологии облачных вычислений и мобильных приложений обеспечивают гибкость и мобильность рабочей силы, что важно в условиях современной динамичной бизнес-среды.

Однако, современная компания — это не просто сумма отдельных систем и процессов. Она представляет собой динамичное органическое образование, которое постоянно меняется и адаптируется к новым вызовам и возможностям рынка. Например, в сфере розничной торговли, современная компания может использовать информационные технологии для анализа покупательского поведения, оптимизации ценообразования и управления запасами, а также для установления более тесных связей с клиентами через онлайн-платформы и социальные сети. Это требует гибкости и адаптивности в управлении, что можно обеспечить с помощью современных информационных решений.

Основой успешной современной компании становится эффективное управление, которое включает в себя не только контроль за финансами и

					КР.09.02.07.1.41.4	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

ресурсами, но и оптимизацию рабочих процессов. Например, с помощью эффективного управления компания может сократить издержки на производстве и снизить временные задержки в поставках. Это достигается благодаря автоматизации процессов управления производством и логистикой, а также оптимизации цепочек поставок. Современные технологии позволяют компаниям отслеживать каждый этап производственного процесса и оперативно реагировать на изменения в спросе или поставках.

Анализ данных становится критически важным элементом эффективного управления. Например, компания в сфере медицинских услуг может использовать анализ данных для прогнозирования потребности в медицинских ресурсах и оптимизации распределения персонала и оборудования в разных клиниках. Это позволяет компании предоставлять более качественное обслуживание пациентов и одновременно снижать операционные расходы. Современные аналитические инструменты и технологии искусственного интеллекта позволяют обрабатывать большие объемы данных и выявлять скрытые закономерности, что помогает компаниям принимать более обоснованные и стратегические решения.

Создание единой платформы для управления всеми аспектами компании становится необходимым, чтобы справиться с множеством вызовов, с которыми сталкиваются современные организации. Платформа может объединить данные о финансах, производстве, логистике и персонале, предоставляя руководителям компании полную картину её состояния. Это помогает компании принимать обоснованные решения, основанные на данных, а не на интуиции. Платформа также может автоматизировать рутинные операции, упростить процессы принятия решений и обеспечить прозрачность и доступность данных для всех участников организации.

Таким образом, создание платформы для управления ресурсами и эффективности работы компании становится актуальной задачей, которая не

					КР.09.02.07.1.41.4	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

только повышает конкурентоспособность организации, но и способствует её долгосрочному успеху.

Цель: разработать платформу для управления ресурсами и эффективностью работы компании.

Задачи:

1. Провести анализ рынка разработки ПО для выявления оптимальной платформы разработки;
2. Изучить процессы и особенности выбранной платформы;
3. Спроектировать базу данных;
4. Разработать программный интерфейс взаимодействия пользователя и ИС.

					КР.09.02.07.1.41.4	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Глава 1. Теоретическая часть

1.1 Понятие информационной системы

Под информационной системой обычно понимается прикладная программная подсистема, ориентированная на сбор, хранение, поиск и обработку текстовой и/или фактографической информации. Подавляющее большинство информационных систем работает в режиме диалога с пользователем.

Автоматизированная информационная система предназначена для более удобной работы с данными. Когда объемы информации, с которыми приходится иметь дело, довольно велики, а сама она имеет достаточно сложную структуру, то при организации работы с ней возникает немало различных проблем. Для препятствия возникновения различных проблем, необходимо разработать такую систему работы с информацией, которая позволила бы реализовать автоматизированный сбор, обработку и работу с данными.

1.2 Виды приложений

Начало разработки информационной системы начинается с выбора доступных платформ и путей для создания информационной системы. Существует три основных подхода в представлении взаимодействия между пользователем и программным обеспечением: консольное приложение, приложение с графическим интерфейсом и веб-приложение.

Консольное приложение — на программном уровне для ввода и вывода информации консольные программы используют стандартные устройства ввода — вывода, хотя могут открывать и другие файлы, сетевые соединения и совершать иные действия, доступные в выполняющей их среде. Вывод печатных символов приводит к появлению этих символов на устройстве вывода, то есть к их получению пользователем.

					<i>КР.09.02.07.1.41.4</i>	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

В простейшем случае консольная программа использует интерфейс командной строки, однако многие из таких программ с помощью управляющих последовательностей терминалов создают более дружелюбный интерфейс, приближающийся к графическому. Некоторые консольные программы пригодны лишь для определённой реализации текстового интерфейса, например, текстовые программы операционных систем.

Приложение с графическим интерфейсом — программа с использованием средств для взаимодействия пользователя с компьютером, основанной на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т. п.). При этом, в отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам — элементам интерфейса, а на экране реализуется модель мира в соответствии с некоторой метафорой и осуществляется прямое манипулирование.

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

После изучения достоинств и недостатков каждого из подходов, было выбрано веб-приложение в качестве разрабатываемого продукта.

1.3 Определение клиент-серверной архитектуры

Клиент-сервер — это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками

					КР.09.02.07.1.41.4	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, передача файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр веб-страниц во всемирной паутине).

Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

Характеристика клиент-сервер описывает отношения взаимодействующих программ в приложении. Серверный компонент предоставляет функцию или услугу одному или нескольким клиентам, которые инициируют запросы на такие услуги. Серверы классифицируются по предоставляемым ими услугам. Например, веб-сервер обслуживает веб-страницы, а файловый сервер обслуживает компьютерные файлы. Общий ресурс может быть любой из программного обеспечения и электронных компонентов компьютера — сервера, от программ и данных в процессорах и

					КР.09.02.07.1.41.4	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

запоминающих устройств. Совместное использование ресурсов сервера представляет собой услугу.

Является ли компьютер клиентом, сервером или и тем, и другим, определяется характером приложения, которому требуются сервисные функции. Например, на одном компьютере могут одновременно работать веб-серверы и программное обеспечение файлового сервера, чтобы обслуживать разные данные для клиентов, отправляющих различные типы запросов. Клиентское программное обеспечение также может взаимодействовать с серверным программным обеспечением на том же компьютере. Связь между серверами, например, для синхронизации данных, иногда называется межсерверной.

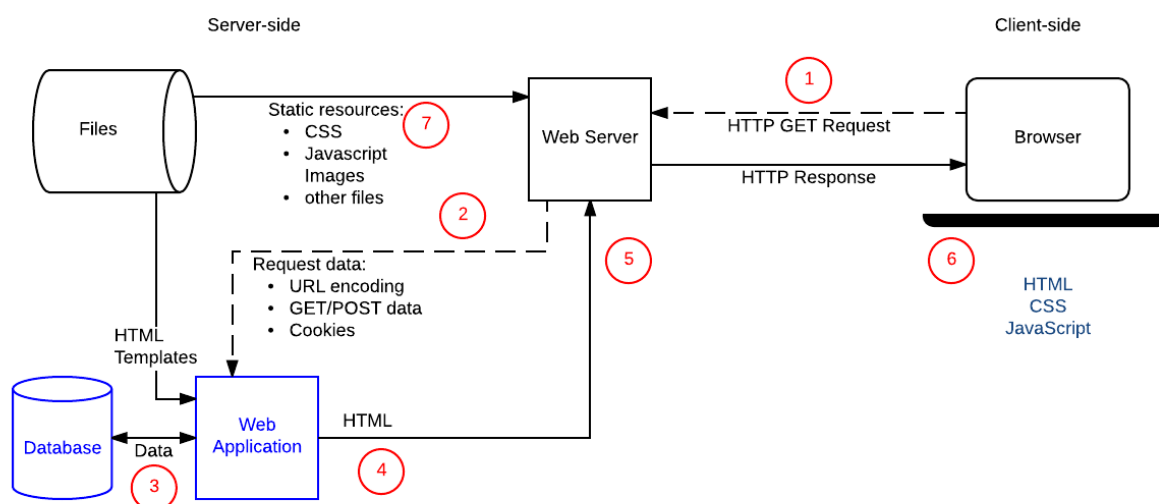


Рисунок 1 – Схема работы клиент-серверной архитектуры

Виды клиент-серверной архитектуры:

- Одноуровневая архитектура — все прикладные программы рассредоточены по рабочим станциям, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные. В целом, такая архитектура очень надежна, однако, ей сложно управлять, поскольку в каждой

рабочей станции данные будут присутствовать в разных вариантах. Поэтому возникает проблема их синхронизации на отдельных машинах.

- Двухуровневая архитектура — прикладные программы сосредоточены на сервере приложений (Application Server), например, сервере 1С или сервере CRM, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере. Такая архитектура представляется наиболее логичной для архитектуры «клиент-сервер». В ней можно выделить два варианта. Когда общие данные хранятся на сервере, а логика их обработки и бизнес-данные хранятся на клиентской машине, то такая архитектура носит название “fat client thin server” (толстый клиент, тонкий сервер). Когда не только данные, но и логика их обработки и бизнес-данные хранятся на сервере, то это называется “thin client fat server” (тонкий клиент, толстый сервер). Такая архитектура послужила прообразом облачных вычислений (Cloud Computing);

- Трехуровневая архитектура — сервер баз данных, файловый сервер и другие представляют собой отдельный уровень, результаты работы которого использует сервер приложений. Логика данных и бизнес-логика находятся в сервере приложений. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение (middleware), которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность;

- Многоуровневая архитектура — несколько серверов приложений используют результаты работы друг друга, а также данные от различных серверов баз данных, файловых серверов и других видов серверов. Преимуществом многоуровневой архитектуры является гибкость предоставления услуг, которые могут являться комбинацией работы различных приложений серверов разных уровней и элементов этих приложений. Очевидным недостатком является сложность, многокомпонентность такой архитектуры.

					КР.09.02.07.1.41.4	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

1.4 Spring Framework

Для проекта был выбран фреймворк Spring в качестве основы для разработки информационной системы на Java платформе. Spring имеет инструменты для создания веб-приложений и большую библиотеку дополнительных проектов, входящих в экосистему Spring. Для более удобной и быстрой разработки будет использоваться проект Spring Boot в качестве основной точки взаимодействия приложения с фреймворком.

Spring — это фреймворк с открытым исходным кодом для языка программирования Java. Он был создан для упрощения разработки и поддержки масштабируемых, слабосвязанных и повторно используемых приложений. Фреймворк нужен, чтобы разработчикам было легче проектировать и создавать приложения. Spring не связан с конкретной парадигмой или моделью программирования, поэтому его могут использовать как каркас для разных видов приложений.

Чаще всего Spring используется в крупных корпоративных проектах: это характерно для Java и связанных с ним инструментов. Но благодаря универсальности применять Spring можно и в других случаях. Его задача — дать разработчикам больше свободы в проектировании и реализации.

Spring — открытый бесплатный проект, просмотреть его исходный код может любой желающий. Он написан на Java, Kotlin и Groovy, поэтому в теории может использоваться с любым из этих языков. На практике Spring чаще всего применяют с Java.

Для чего нужен Spring:

- Для более быстрого и легкого создания приложений — набор инструментов фреймворка позволяет выполнять те же задачи с меньшим количеством затрат, чем при написании с нуля;
- Для архитектурной «гибкости»: Spring универсален, поэтому позволяет реализовать нестандартные решения;

					КР.09.02.07.1.41.4	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

- Для гибкого использования возможностей — к проекту можно подключать разнообразные модули и тем самым настраивать инструментарий под свои нужды;
- Для удобного построения зависимостей — благодаря Spring разработчики могут сконцентрироваться на логике приложения, а не на том, как подключить одно к другому;
- Для решения задач с межкомпонентной связью — Spring позволяет удобно работать со связями между компонентами или разными приложениями, для доступа различных частей системы друг к другу и многого другого.

1.5 Spring Boot

Spring Boot — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать веб-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода. Spring Boot обладает большим функционалом.

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

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

					КР.09.02.07.1.41.4	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

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

Ещё одной превосходной возможностью Spring Boot является автоматическая конфигурация приложения после выбора подходящего starter-пакета — Spring Boot попытается автоматически настроить Spring-приложение на основе добавленных jar-зависимостей.

Каждое Spring Boot веб-приложение включает встроенный веб-сервер. Посмотрите на список контейнеров сервлетов, которые поддерживаются «из коробки».

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

При необходимости использовать отдельный HTTP-сервер достаточно исключить зависимости по умолчанию. Spring Boot предоставляет отдельные starter-пакеты для разных HTTP-серверов.

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

1.6 Git

Современная разработка программного обеспечения не обходится без системы управления версиями. Без управления версиями разработчики заманчивы хранить несколько копий кода на своем компьютере. Это опасно, так как легко изменить или удалить файл в неправильной копии кода,

					КР.09.02.07.1.41.4	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

потенциально потеряв работу. Системы управления версиями решают эту проблему, управляя всеми версиями кода, но предоставляя команде одну версию одновременно. Раньше существовало множество различных популярных системы управления версиями, но на сегодня существует лишь одна популярная и всемирно используемая система — Git.

Git — это распределенная система управления версиями, которая означает, что локальный клон проекта — это полный репозиторий управления версиями. Полнофункциональные локальные репозитории упрощают работу как в автономном, так и в удаленном режиме. Разработчики фиксируют свою работу локально, а затем синхронизируют копию репозитория с копией на сервере. Эта парадигма отличается от централизованных систем управления версиями, где клиенты должны синхронизировать код с сервером перед созданием новой версии кода.

Гибкость и популярность Git делают его отличным выбором для любой команды. Сообщество пользователей Git создало ресурсы для обучения разработчиков и популярности Git, что упрощает получение помощи при необходимости. Почти каждая среда разработки поддерживает Git и средства командной строки Git, реализованные в каждой основной операционной системе.

При каждом сохранении работы Git создает фиксацию. Фиксация — это моментальный снимок всех файлов в определенный момент времени. Если файл не изменился с одной фиксации на следующую, Git использует ранее сохраненный файл. Эта конструкция отличается от других систем, которые хранят начальную версию файла и сохраняют запись разностных с течением времени.

Фиксации создают ссылки на другие фиксации, формируя граф журнала разработки. Можно вернуть код к предыдущей фиксации, проверить, как файлы изменились с одной фиксации на следующую, а также просмотреть сведения о том, где и когда были внесены изменения. Фиксации определяются

					КР.09.02.07.1.41.4	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

в Git уникальным криптографическим хэшем содержимого фиксации. Так как все хэшируется, невозможно вносить изменения, терять информацию или повреждены файлы без обнаружения Git.

1.7 Шаблон MVC

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. MVC архитектура широко применяется для построения веб-приложений, особенно во фреймворке Spring.

Статическая страница на HTML не умеет реагировать на действия пользователя. Для двухстороннего взаимодействия нужны динамические веб-страницы. MVC — ключ к пониманию разработки динамических веб-приложений.

Компоненты MVC:

- Модель — этот компонент отвечает за данные, а также определяет структуру приложения;
- Представление — этот компонент отвечает за взаимодействие с пользователем. Код компонента view определяет внешний вид приложения и способы его использования;
- Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя.

1.8 База данных

Ни одна современная информационная система не обходится без базы данных. База данных — это центральный блок приложения, который содержит все данные, касающиеся различных компонентов приложения. Это могут

					КР.09.02.07.1.41.4	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

быть, например, конфигурационные данные, сведения о самом приложении, сведения о пользователях, данные, принадлежащие пользователям.

Обычно базы данных делят на два вида:

- Нереляционные — NoSQL;
- Реляционные — SQL.

Реляционная база данных — это классический и наиболее часто используемый вид баз данных. SQL-базы данных используют структурированный язык запросов для управления данными, для их обработки, хранения, обновления, удаления. В реляционных хранилищах данных используются реляционные системы управления базами данных (РСУБД, Relational Database Management Systems, RDBMS). В РСУБД данные хранятся в табличном формате. Таблица — это базовая единица базы данных, она состоит из строк и столбцов, в которых хранятся данные. Таблицы — это самый часто используемый тип объектов баз данных, или структур, которые в реляционной БД хранят данные или ссылки на них.

Нереляционная база данных — Этим термином обозначают базу данных любого типа, которая хранит информацию не так, как хранят её РСУБД. NoSQL-базы данных отлично подходят для многих современных приложений, которым нужны гибкие, масштабируемые, высокопроизводительные базы данных, обладающие широким набором функциональных возможностей. Среди таких приложений — проекты из мобильной и веб-сферы, компьютерные игры. Возможности NoSQL-баз данных позволяют таким проектам давать своим пользователям наилучшие впечатления от работы с ними.

Причина роста популярности NoSQL-баз данных, в основном, кроется в необходимости работы с данными, которые имеют самую разную структуру и самые разные размеры. Это могут быть структурированные, полуструктурированные и полиморфные данные. При работе с такими

					КР.09.02.07.1.41.4	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

данными определить схему данных практически невозможно. NoSQL, кроме того, даёт разработчику большую гибкость в тех случаях, когда ему нужно быстро адаптировать систему к изменениям. Например — это характерно для стартапов. NoSQL-базы данных предлагают пользователям API с широкими функциональными возможностями, а также — типы данных, спроектированные специально для определённых моделей данных.

Базы данных NoSQL, кроме того, легче, чем БД SQL, поддаются масштабированию, так как NoSQL БД могут масштабироваться горизонтально. Делается это путём добавления дополнительных узлов при возникновении необходимости обрабатывать больше трафика, чем обычно. Это упрощает и расширение мощности в ситуациях пиковых сетевых нагрузок, и её сокращение в случаях, когда трафика не очень много. Это улучшает масштабируемость приложений. Это не значит, что SQL-базы данных не поддерживают горизонтальное масштабирование. При работе с РСУБД это, просто, делается сложнее. SQL-хранилища, в основном, масштабируют вертикально, то есть — давая больше вычислительной мощности серверу, на котором работает SQL-база данных.

1.9 PostgreSQL

Для создания веб-приложения был выбран PostgreSQL в качестве СУБД. PostgreSQL — это объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире. Имеет открытый исходный код и является альтернативой коммерческим базам данных.

Ранние версии системы были основаны на старой программе POSTGRES University, созданной университетом Беркли: так появилось название PostgreSQL. И сейчас СУБД часто называют «Постгрес». Существуют сокращения PSQL и PgSQL — они тоже обозначают PostgreSQL.

СУБД позволяет гибко управлять базами данных. С ее помощью можно создавать, модифицировать или удалять записи, отправлять транзакцию — набор из нескольких последовательных запросов на языке запросов SQL.

Особенности PostgreSQL:

- Работа с большими объемами данных — В большинстве СУБД, рассчитанных на средние и небольшие проекты, есть ограничения по объему базы и количеству записей в ней. В PostgreSQL ограничений нет. Существуют ограничения только на размеры конкретных записей;
- Поддержка сложных запросов — PostgreSQL работает со сложными, составными запросами. Система справляется с задачами разбора и выполнения трудоемких операций, которые подразумевают и чтение, и запись, и валидацию одновременно. В операциях чтения она медленнее аналогов, но в других аспектах зачастую превосходит конкурентов;
- Написание функций на нескольких языках — В PostgreSQL можно писать собственные функции — пользовательские блоки кода, которые выполняют различные действия. В отличие от конкурентов, PostgreSQL поддерживает больше языков. Кроме стандартного SQL, в PostgreSQL можно писать на C и C++, Java, Python, PHP, Lua и Ruby. При необходимости можно расширить систему под другие языки программирования;
- Высокая мощность и широкая функциональность — PostgreSQL — единственная бесплатная СУБД с открытым исходным кодом, которая рассчитана на работу с объемными и сложными проектами. Она мощная, производительная, способна эффективно работать с большими массивами данных. Существуют примеры реального использования СУБД для баз данных в несколько петабайт с сотнями тысяч запросов в секунду;
- Минимальное количество багов — PostgreSQL — проект, который известен высоким качеством отладки. Каждая версия системы появляется в доступе только после полной проверки, поэтому СУБД очень стабильна;

- Кроссплатформенность — Чаще всего PostgreSQL используют на серверах с операционными системами семейства Linux, но СУБД поддерживает и другие ОС. Ее можно установить в системы на базе Windows, BSD, macOS и Solaris. Кроме того, у PostgreSQL есть автономный веб-сервер PostgREST, с которым можно работать с помощью REST API. СУБД можно развернуть и в облаке.

1.10 Объектно-реляционная модель (ORM)

ORM (Object-Relational Mapping) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

Использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, заставляя программистов писать программное обеспечение, которое должно уметь обрабатывать данные в объектно-ориентированном виде, а хранить эти данные в реляционной форме. Эта постоянная необходимость в преобразовании между двумя разными формами данных не только сильно снижает производительность, но и создаёт трудности для программистов, так как обе формы данных накладывают ограничения друг на друга.

Так как системы управления реляционными базами данных обычно не реализуют реляционного представления физического уровня связей, выполнение нескольких последовательных запросов (относящихся к одной «объектно-ориентированной» структуре данных) может быть слишком затратно.

Некоторые реализации ORM автоматически синхронизируют загруженные в память объекты с базой данных. Для того чтобы это было возможным, после создания объект-в-SQL-преобразующего SQL-запроса

(класса, реализующего связь с БД) полученные данные копируются в поля объекта, как во всех других реализациях ORM. После этого объект должен следить за изменениями этих значений и записывать их в базу данных.

Системы управления реляционными базами данных показывают хорошую производительность на глобальных запросах, которые затрагивают большой участок базы данных, но объектно-ориентированный доступ более эффективен при работе с малыми объёмами данных, так как это позволяет сократить семантический провал между объектной и реляционной формами данных.

Существует множество пакетов, устраняющих необходимость в преобразовании объектов для хранения в реляционных базах данных.

Некоторые пакеты решают эту проблему, предоставляя библиотеки классов, способных выполнять такие преобразования автоматически. Имея список таблиц в базе данных и объектов в программе, они автоматически преобразуют запросы из одного вида в другой.

С точки зрения программиста система должна выглядеть как постоянное хранилище объектов. Он может просто создавать объекты и работать с ними как обычно, а они автоматически будут сохраняться в реляционной базе данных.

Все системы ORM обычно проявляют себя в том или ином виде, уменьшая в некотором роде возможность игнорирования базы данных. Более того, слой транзакций может быть медленным и неэффективным (особенно в терминах сгенерированного SQL). Всё это может привести к тому, что программы будут работать медленнее и использовать больше памяти, чем программы, написанные «вручную». Но ORM избавляет программиста от написания большого количества кода, часто однообразного и подверженного ошибкам, тем самым значительно повышая скорость разработки. Кроме того, большинство современных реализаций ORM позволяют программисту при необходимости самому жёстко задать код SQL-запросов, который будет

использоваться при тех или иных действиях (сохранение в базу данных, загрузка, поиск и т. д.) с постоянным объектом.

1.11 Spring Data JPA

Spring Data JPA— это модуль Spring Framework, используемый для удобного взаимодействия с сущностями базы данных, организации их в репозитории, извлечения данных их изменения из кода в виде объектов на основе спецификации Java Persistence API.

Спецификация JPA описывает систему управления сохранением java объектов в таблицы реляционных баз данных в удобном виде. Сама Java не содержит реализации JPA, однако есть существует много реализаций данной спецификации от разных компаний

Spring Data JPA — это абстракция над Hibernate, которая значительно упрощает жизнь разработчика: быстрее и удобнее аналогичных решений.

Hibernate Framework — это фреймворк для языка Java, предназначенный для работы с базами данных, реализующий объектно-реляционную модель.

					КР.09.02.07.1.41.4	Лист
						21
Изм.	Лист	№ докум.	Подпись	Дата		

Например, для создания и удаления таблиц можно добавить соответствующую строку в `application.properties`.

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

1.12 Thymeleaf

Thymeleaf — современный серверный механизм Java-шаблонов для веб и автономных сред, способный обрабатывать HTML, XML, JavaScript, CSS и даже простой текст.

Основной целью Thymeleaf является создание элегантного и удобного способа шаблонизации. Чтобы достичь этого, Thymeleaf основывается на концепции Natural Templates, чтобы внедрить свою логику в файлы шаблонов таким образом, чтобы этот шаблон не влиял на отображение прототипа дизайна. Это улучшает коммуникацию в команде и уменьшает разрыв между дизайнерско-программистскими группами.

Thymeleaf — чрезвычайно расширяемый механизм (на самом деле его можно назвать платформой шаблонов), который позволяет вам определять и настраивать способ обработки ваших шаблонов до тонкого уровня детализации.

Объект, который применяет некоторую логику к артефакту разметки (тегу, тексту, комментарию) называется процессором, а набор этих процессоров — является диалектом. Основная библиотека Thymeleaf

					КР.09.02.07.1.41.4	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

предоставляет диалект, называемый стандартным диалектом, которого должно быть достаточно для большинства пользователей.

Официальные пакеты интеграции «thymeleaf-spring3» и «thymeleaf-spring4» определяют диалект, называемый «SpringStandard Dialect», который в основном совпадает с стандартным диалектом, но с небольшими адаптациями, чтобы лучше использовать некоторые функции Spring Framework.

					КР.09.02.07.1.41.4	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		

Глава 2. Практическая часть

2.1 Создание проекта

Создание нового проекта на Spring Boot принято начинать с помощью официального инициализатора. На сайте выбираются необходимые зависимости и параметры проекта. После нажатия кнопки «generate» произойдёт автоматическая сборка проекта, и начнётся скачивание. Используем модули Spring Web, Spring Data JPA, а также Lombok.

The screenshot shows the Spring Initializr web interface. At the top is the 'spring initializr' logo. Below it, the 'Project' section has radio buttons for 'Gradle - Groovy' (selected), 'Gradle - Kotlin', and 'Maven'. The 'Language' section has radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has radio buttons for versions: '3.2.0 (SNAPSHOT)', '3.2.0 (RC2)', '3.1.6 (SNAPSHOT)', '3.1.5' (selected), '3.0.13 (SNAPSHOT)', '3.0.12', '2.7.18 (SNAPSHOT)', and '2.7.17'. The 'Project Metadata' section includes fields for 'Group' (ru.rambler.ptvkz1000), 'Artifact' (companyResourceManagement), 'Name' (companyResourceManagement), 'Description' (Курсовая работа студента колледжа КАДР26), and 'Package name' (ru.rambler.ptvkz1000.companyResourceManagement). The 'Packaging' section has radio buttons for 'Jar' (selected) and 'War'. The 'Java' section has radio buttons for versions: '21', '17' (selected), '11', and '8'. On the right, the 'Dependencies' section has a button 'ADD DEPENDENCIES... CTRL + B'. Below it, there are three dependency cards: 'Lombok' with a 'DEVELOPER TOOLS' tag, 'Spring Web' with a 'WEB' tag, and 'Spring Data JPA' with an 'SQL' tag. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. The background is light blue with a subtle grid pattern.

Рисунок 3 – Spring инициализатор

Для завершения этапа инициализации проекта открываем скачанный проект в IDE и производим стартовый GIT коммит.

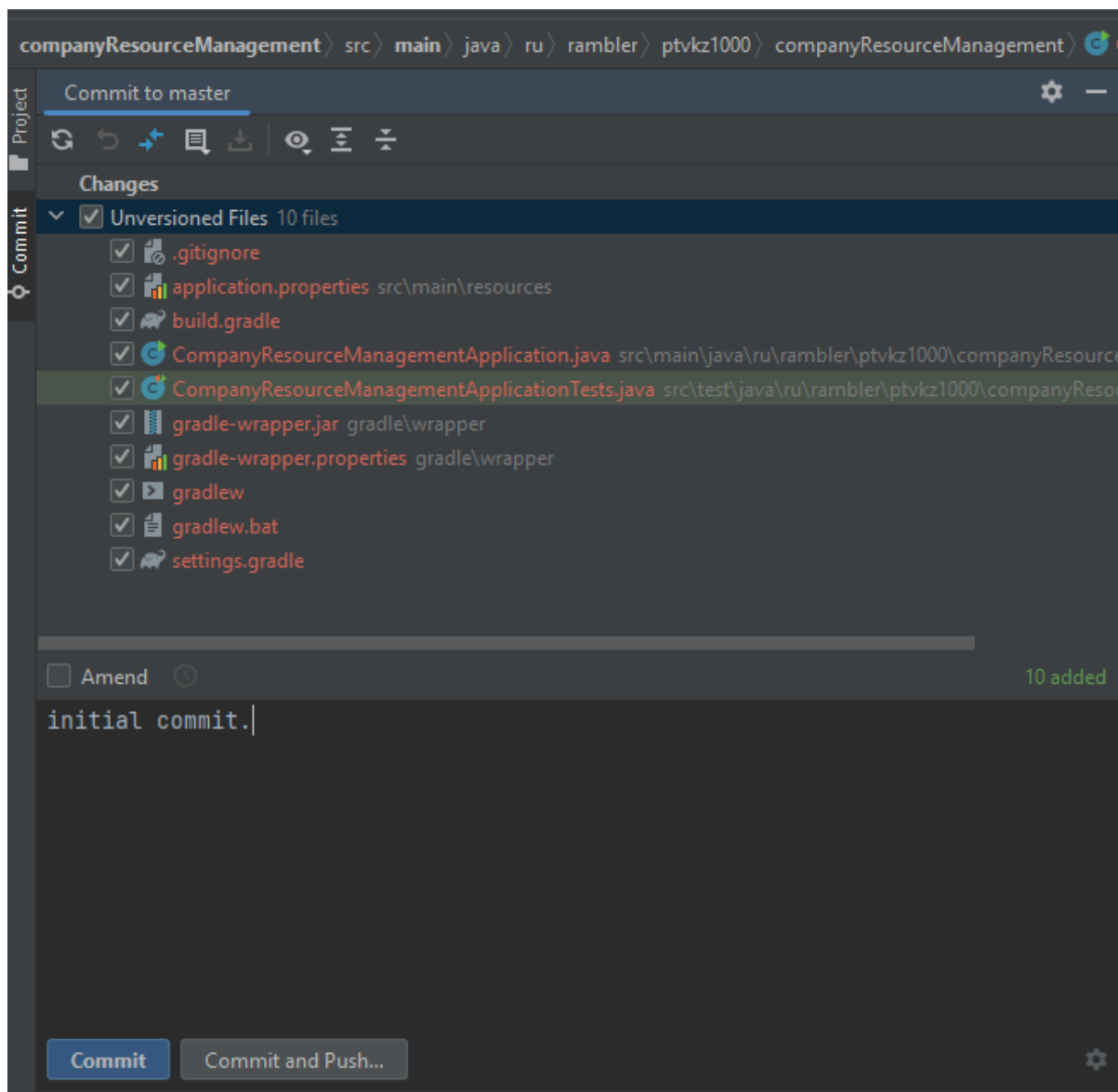


Рисунок 4 – Стартовый коммит

2.2 Создание базы данных

В качестве базы данных был выбран PostgreSQL. Для запуска сервера базы данных был использован Docker. Устанавливаем образ сервера PostgreSQL, а затем создаём на его основе контейнер. Вписываем 0 в поле Ports, чтобы автоматически привязать порт хоста к виртуальному порту контейнера. В качестве переменной среды добавляем «POSTGRES_HOST_AUTH_METHOD» со значением trust, чтобы сервер не требовал пароль для использования сервера со всеми правами. Данный подход

крайне не рекомендуется по соображениям безопасности для полноценных проектов, но был использован для удобства пользования учебным проектом.

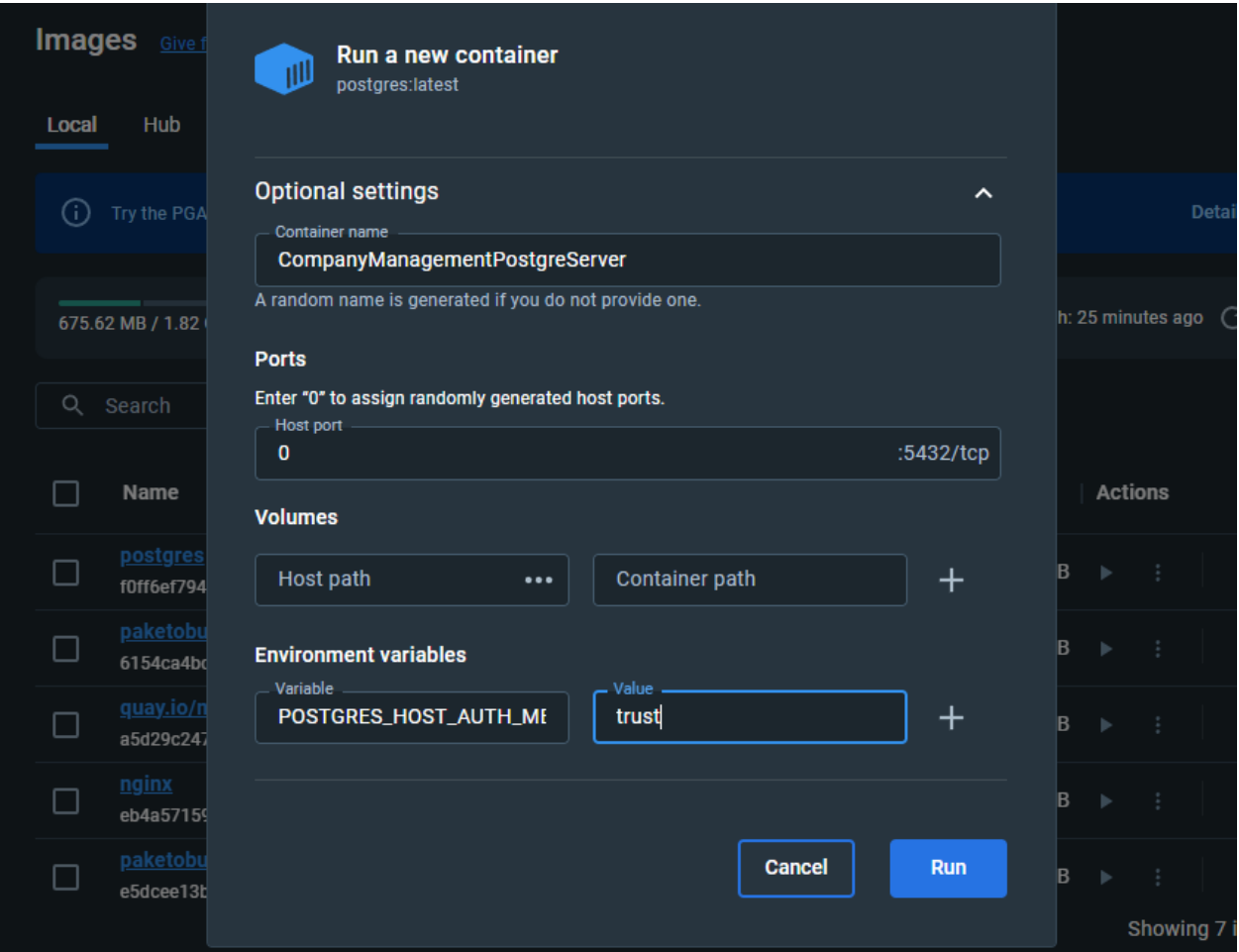


Рисунок 5 – Окно создания контейнера в Docker

Теперь, когда сервер Postgre «поднят», необходимо подключиться к нему с помощью клиента. Клиентом выступает pgAdmin 4. В открытом клиенте выбираем Servers – Register – Server для подключения к созданному локальному серверу. Во вкладке General добавляем отображаемое имя сервера, а во вкладке Connection вводим «localhost» для адреса сервера и порт, определённый в Docker.

The screenshot shows the 'Register - Server' dialog box in PgAdmin. The 'Connection' tab is selected. The following fields are visible:

- Host name/address: localhost
- Port: 32769
- Maintenance database: postgres
- Username: postgres
- Kerberos authentication?: ☐
- Password: (empty)
- Save password?: ☐
- Role: (empty)
- Service: (empty)

At the bottom right, there are three buttons: 'Close', 'Reset', and 'Save'.

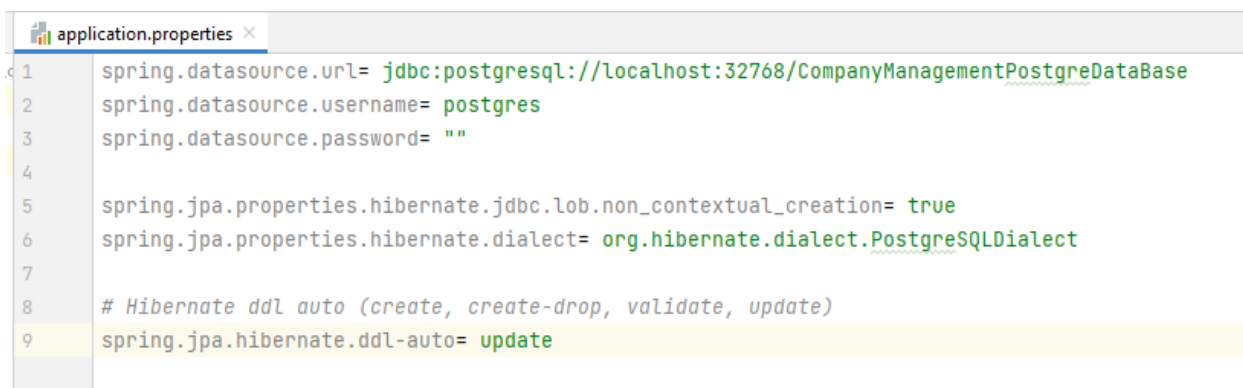
Рисунок 6 – Окно добавления сервера в PgAdmin

Далее создаём новую базу данных на добавленном сервере под названием CompanyManagementPostgreDataBase. Теперь, когда база данных создана, к ней можно подключиться из приложения. Для этого используется Spring Data JPA.

Для подключения прописываем необходимые параметры в application.properties файл:

- spring.datasource.url отвечает за драйвер и адрес базы данных. Вписываем для него «jdbc:postgresql» в качестве драйвера и адрес до базы данных через комбинацию символов «://»;
- spring.datasource.username отвечает за имя пользователя, подключаемого к базе данных. Используем стандартное имя «postgres», для подключения к пользователю администратора;

- `spring.datasource.password` отвечает за пароль подключаемого пользователя к базе данных. Так как указано значение «trust» во время создания базы данных в Docker, пароль не требуется;
- `spring.jpa.properties.hibernate.dialect` отвечает за выбор диалекта SQL-запросов. Для значения используем полный путь до класса: «`org.hibernate.dialect.PostgreSQLDialect`»;
- `spring.jpa.hibernate.ddl-auto` отвечает за модель работы фреймворка с базой данных. Значение «update» означает, что объектная модель, созданная на маппингах, будет автоматически сверяться с существующей схемой и обновлять таблицы в соответствии с моделью. Но удаление избыточных и неиспользуемых таблиц осуществляться не будет.



```

1 spring.datasource.url= jdbc:postgresql://localhost:32768/CompanyManagementPostgreDataBase
2 spring.datasource.username= postgres
3 spring.datasource.password= ""
4
5 spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation= true
6 spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.PostgreSQLDialect
7
8 # Hibernate ddl auto (create, create-drop, validate, update)
9 spring.jpa.hibernate.ddl-auto= update

```

Рисунок 7 – «application.properties» файл

2.3 Создание объекта и репозитория сотрудника

Для внесения и изменения данных в базе данных с помощью JPA создаём новый класс «Employee», с нужными полями «id», «firstName» и «lastName». К классу приписываем аннотацию Entity, чтобы явно объявить принадлежность класса к сущности JPA. Для id указываем аннотации Id и GeneratedValue, чтобы определить id как первичный ключ и задать автоматическую генерацию значения.

```

@Entity 3 usages  Andrei
public class Employee {

    @Id 2 usages
    @GeneratedValue
    private Long id;
    private String firstName; 3 usages
    private String lastName; 3 usages
    @Nullable no usages
    private Date birthDate;
    private Date registrationDate; 2 usages

    public Long getId() { return id; }
    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }
    public Date getRegistrationDate() { return registrationDate; }

    public void setFirstName(String firstName) { this.firstName = firstName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
    public void setRegistrationDate(Date registrationDate) { 1 usage  Andrei
        this.registrationDate = registrationDate;
    }

    @Override  Andrei
    public String toString() {
        return String.format(
            "Employee[id=%d, firstName='%s', lastName='%s']",
            id, firstName, lastName);
    }
}

```

Рисунок 8 – Класс Employee

Теперь, для непосредственного взаимодействия с базой данных с помощью созданной сущности, необходимо создать соответствующий репозиторий. Для этого объявляем новый интерфейс с аннотацией Repository. Этот интерфейс должен наследоваться от одного из Spring репозиториев. Для сущности сотрудника используем CrudRepository, позволяющий производить все базовые операции над сущностью.

					КР.09.02.07.1.41.4	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

```

1 usage  Andrei*
@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Long> {
}

```

Рисунок 9 – Интерфейс EmployeeRepository

Инстансируем репозиторий в основном классе с помощью аннотации `Autowired`. Также необходимо добавить аннотацию `EnableJpaRepositories` к классу приложения, чтобы Spring начал корректно обрабатывать модуль `SpringDataJpa`.

```

@Controller
@SpringBootApplication
@EnableJpaRepositories
public class CompanyResourceManagementApplication {

    4 usages
    @Autowired
    private EmployeeRepository repository;
}

```

Рисунок 10 – Поле репозитория в классе приложения

2.4 Создание главной страницы

Добавляем аннотацию «`Controller`», к классу приложения, чтобы Spring Framework искал контроллеры среди методов главного класса.

Создаем контроллер GET-запросов для главной страницы с помощью помеченного аннотацией `GetMapping` метода основного класса приложения. Главная страница будет выводить текущее количество сотрудников, поэтому добавляем атрибут для выводимого сообщения в представление.

					КР.09.02.07.1.41.4	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

```

no usages  Andrei *
@GetMapping("/")
public String index(Model model){
    String employeeCount = String.valueOf(repository.count());
    model.addAttribute( attributeName: "employeeCountMessage",
                        attributeValue: "В нашей компании уже "
                        + employeeCount + " сотрудников!");
    return "index";
}

```

Рисунок 11 – GET-контроллер для главной страницы

Создаём представление в директории `resources.templates` в виде файла `index.html`. Это представление контроллер будет возвращать пользователю, ссылаясь на строковое значение, возвращённое из метода контроллера.

Объявляем пространство имён `thymeleaf` для корректной работы генератора шаблонов. С помощью `thymeleaf` и `Spring` диалекта можно пользоваться атрибутами и сущностями `Spring` классов.

В созданном файле оформляем страницу с помощью языка разметки. В тэге «`head`» указываем кодировку и название страницы.

Добавляем «шапку» страницы для навигации между страницами веб-приложения. В отдельном блоке «`div`» указываем ссылки на отдельные страницы приложения в виде ссылочных тэгов «`a`» и атрибутов «`href`». В атрибуте «`href`» указываем эндпоинты на GET-контроллеры.

Для вывода действующего количества сотрудников на главную страницу веб-приложения используем атрибут `thymeleaf` «`th:text`». С его помощью можно генерировать части `html`-страницы через передаваемые в представления из GET-контроллера значения.


```

1  <!DOCTYPE HTML>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <title>Управление компанией</title>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6  </head>
7  <body>
8
9      <div>
10         <a style="color:red" href="/">Главная</a>
11         <a href="/employees">Список сотрудников</a>
12
13     </div>
14
15     <h1 style="text-align:center">Добро пожаловать.</h1>
16     <h2 style="text-align:center" th:text="|${employeeCountMessage}|"></h2>
17
18 </body>
19 </html>

```

Рисунок 12 – Представление главной страницы

2.5 Создание страницы «Сотрудники»

Создаём GET-контроллер для новой страницы. В качестве атрибута представления передаём лист всех сотрудников из репозитория для вывода списка сотрудников в таблице.

```

@GetMapping("employees") no usages Andrei *
public String employeesGet(Model model) {
    model.addAttribute( attributeName: "employeeList",
        repository.findAll());
    return "employees";
}

```

Рисунок 13 – GET-контроллер страницы «Сотрудники»

Создаем представление для страницы. В файл прописываем навигационную шапку, ссылку на регистрацию нового сотрудника и таблицу. С помощью «th:each» производим итерацию по каждой записи сотрудников для вывода на страницу всей таблицы из базы данных.

Для создания кнопки удаления создаём новую форму внутри table data блока. Чтобы каждая сгенерированная кнопка отвечала за свою отдельную запись сотрудника, создаём новый input внутри формы с типом «hidden». Этой кнопке назначаем имя «id» и значение в виде поля «id» объекта текущей итерации.

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Управление компанией</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<div>
    <a href="/">Главная</a>
    <a style="color: red" href="/employees">Список сотрудников</a>
</div>

<h1><a href="/newEmployee">Новый сотрудник</a></h1>

<table>
    <thead>
        <tr>
            <th>Имя</th>
            <th>Фамилия</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="employee : ${employeeList}">
            <td><span th:text="${employee.firstName}">Имя</span></td>
            <td><span th:text="${employee.lastName}">Фамилия</span></td>
            <td>
                <form action="#" th:action="@{/employees/delete}" method="post">
                    <input type="hidden" th:name="id" th:value="${employee.id}"/>
                    <button type="submit">Delete</button>
                </form>
            </td>
        </tr>
    </tbody>
</table>

</body>
</html>
```

Рисунок 14 – Представление страницы «Сотрудники»

2.6 Создание формы регистрации нового сотрудника

В отличие от предыдущих страниц, для этой страницы создаём два контроллера: для запросов GET и POST.

Объявляем параметр класса Model в методе GET-контроллера. Spring Framework автоматически передаст в него объект представления.

С помощью метода «addAttribute» передаём JPA сущность сотрудника в представление. С помощью этого объекта Thymeleaf сможет автоматически генерировать POST-запросы к контроллеру, передавая заполненные данные формы в объектном представлении.

```
@GetMapping("newEmployee")
public String newEmployeeGet(Model model) {
    model.addAttribute("employee", new Employee());
    return "newEmployee";
}

@PostMapping("newEmployee")
public String employeesPost(@ModelAttribute Employee employee) {
    repository.save(employee);
    return "redirect:/employees";
}
```

Рисунок 15 – Контроллеры страницы формы нового сотрудника

Для представления страницы с формой используем возможность thymeleaf для формирования POST-запросов к эндпоинтам Spring контроллеров прямо из html представления и использования Java объектов внутри разметки.

Создаем тэг формы внутри html документа. В качестве атрибута «action» используем филлер в виде знака решётки, потому что для замещения тэга «action» необходимо использовать специализированный тэг thymeleaf «th:action». Он позволяет обращаться POST-запрос напрямую к эндпоинту POST-контроллера. Также добавляем атрибут «th:object» со значением названия класса необходимой JPA сущности. Благодаря этому возможно

напрямую использовать значения полей из формы в виде удобных полей объекта в POST-контроллере.

В POST-контроллере в качестве возвращаемого значения используем «redirect:/employees», чтобы после отправки формы нового сотрудника пользователь переправлялся на GET-контроллер страницы и получал страницу сотрудников с обновленной таблицей.

```
employees.html × Employee.java × CompanyResourceManagementApplication.java × EmployeeRepository.java × newEmployee.html × index.html ×
1 <!DOCTYPE HTML>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <title>Управление компанией</title>
5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6 </head>
7 <body>
8 <div>
9 <a href="/">Главная</a>
10 <a href="/employees">Список сотрудников</a>
11 </div>
12
13 <h1>Регистрация нового сотрудника</h1>
14 <form action="#" th:action="@{/newEmployee}" th:object="${employee}" method="post">
15 <p>Имя: <input type="text" th:field="*{firstName}"/> Фамилия: <input type="text" th:field="*{lastName}"/> </p>
16 <p><input type="submit" value="Подтвердить" /> <input type="reset" value="Сбросить" /></p>
17 </form>
18
19 </body>
20 </html>
```

Рисунок 16 – Представление формы регистрации нового сотрудника

2.7 Обработка удаления записи из таблицы

Создаем новый POST-контроллер для эндпоинта «employees/delete», который будет отвечать за удаление записи сотрудника. Кнопка POST-запроса из страницы сотрудников отправляет параметр «id» с целочисленным значением идентификатора сотрудника, поэтому объявляем один параметр метода с аннотацией «RequestParam».

Теперь с помощью репозитория возможно удаление записи сотрудника из базы данных через его уникальный идентификатор.

Чтобы пользователь сразу увидел изменение, переадресуем пользователя на GET-контроллер с таблицей сотрудников.

```

@PostMapping("employees/delete")
public String deleteEmployee(@RequestParam(name="id") Long id) {
    repository.deleteById(id);
    return "redirect:/employees";
}

```

Рисунок 17 – POST-контроллер удаления записи

2.8 Создание страницы профиля сотрудника

Создаём GET-контроллер с переменной в эндпоинте. Эта переменная является уникальным идентификатором сотрудника, по которому будет производиться поиск в базе данных. Чтобы использовать переменную из эндпоинта внутри метода, объявляем параметр с аннотацией «PathVariable».

Чтобы передать объект сотрудника в представление, делаем запрос к базе данных по id сотрудника и, в случае нахождения записи, добавляем найденный объект в атрибут модели. В противном случае возвращаем страницу ошибки с сообщением: «Профиль не найден.».

```

}

@GetMapping("employees/{id}") no usages  Andrei
public String employeeProfile(@PathVariable long id, Model model) {
    AtomicReference<String> view = new AtomicReference<>( initialValue: "profile");
    employeeRepository.findById(id).ifPresentOrElse(employee -> model.addAttribute( attributeName: "employee", employee),
        () -> {
            view.set("error");
            model.addAttribute( attributeName: "message", attributeValue: "Профиль не найден.");
        });
    return view.get();
}

```

Рисунок 18 – GET-контроллер профиля сотрудника

Создаём HTML-шаблон в папке шаблонов. Задаём значение тэга «title» в виде строки, составленной через поля имени и фамилии сотрудника. В отдельном блоке отрисовываем карточку сотрудника. В поле даты регистрации выводим дату, при условии наличия значения этой даты в поле объекта.

					КР.09.02.07.1.41.4	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

```

<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">

<head>
  <title th:text="${employee.firstName + ' ' + employee.lastName}"></title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<div>
  <a href="/">Главная</a>
  <a style="color:red" href="/employees">Список сотрудников</a>
</div>

<div style="width:50%; margin:auto; border:solid; border-width:1px; padding:20px">
  <h1 style="text-align:center" th:text="${employee.firstName + ' ' + employee.lastName}"></h1><br>
  <h3 th:text="${employee.registrationDate == null ?
    'Дата регистрации: до начала времён' :
    'Дата регистрации: ' + employee.registrationDate}"></h3>
</div>

</body>
</html>

```

Рисунок 19 – Представление профиля сотрудника

Заключение

В рамках данной курсовой работы была проведена разработка информационной системы с использованием Spring Framework, JPA и Thymeleaf. Этот процесс охватывал множество ключевых задач, каждая из которых играла важную роль в создании современного веб-приложения. В ходе выполнения работы были реализованы следующие задачи:

1. Анализ рынка ПО. Первым этапом работы был анализ современного рынка программного обеспечения. Этот этап помог определить актуальность выбранных технологий и инструментов для разработки информационной системы. Исследование рынка также позволило выявить лучшие практики и подходы к разработке, что важно для создания конкурентоспособного продукта.;

2. Изучение технологий. В процессе выполнения работы было необходимо углубленно изучить Spring Framework, Java Persistence API (JPA), PostgreSQL и Thymeleaf. Это включало в себя ознакомление с архитектурой, основными компонентами, их взаимодействием и спецификой применения. Полученные знания стали фундаментом для успешной реализации информационной системы.;

3. Проектирование базы данных. Одним из ключевых этапов работы было проектирование базы данных для информационной системы. Здесь важно было правильно определить структуру данных, связи между сущностями и обеспечить её эффективное функционирование. Этот этап позволил создать надежную основу для хранения и управления информацией;

4. Разработка программного интерфейса. С использованием Spring Framework и JPA была разработана серверная часть веб-приложения, а Thymeleaf был использован для создания пользовательского интерфейса. Этот этап включал в себя создание логики приложения, обработку запросов и генерацию динамических HTML-страниц. Результатом этой работы стало веб-

					КР.09.02.07.1.41.4	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

приложение, способное взаимодействовать с пользователями и обеспечивать им удобный доступ к данным и функциональности.

Таким образом, курсовая работа позволила успешно освоить современные технологии разработки информационных систем, а также продемонстрировать навыки анализа, проектирования и реализации веб-приложений. Разработанная информационная система представляет собой полезный инструмент для управления данными и может быть доработана и расширена в дальнейшем.

					КР.09.02.07.1.41.4	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованной литературы

1. Алексеев А.В., Рогозин Д.В. Создание веб-приложений на Java с использованием фреймворка Spring Boot. 2022.
2. Официальная документация PostgreSQL. URL: <https://www.postgresql.org/docs/> (дата обращения: 18.11.2023).
3. Официальная документация Thymeleaf. URL: <https://www.thymeleaf.org/documentation.html> (дата обращения: 22.11.2023).
4. Курняван, Б. Программирование WEB-приложений на языке Java / Б. Курняван. - М.: Лори, 2023. - 880 с.
5. Вершинин А.И., Мустафиев Т.С. Разработка приложений на основе фреймворка Spring. 2020.
6. Официальная документация Spring. URL: <https://docs.spring.io/spring-framework/reference/> (дата обращения: 20.11.2023).
7. Атенцио, Л Функциональное программирование на JavaScript: как улучшить код JavaScript-программ / Л Атенцио. - М.: Диалектика, 2022. – 304с.
8. Орм М. Hibernate в действии. 2020.
9. Рамальхо К. Java. Эффективное программирование. 2021.
10. Савин А. Spring 5 на примерах. 2021.
11. Сен М. Spring Boot в действии. 2020.
12. Сметана А.В., Гусельников А.А. Реализация серверной части информационной системы на фреймворке Spring Boot. 2020.
13. Фаулер М. Разработка корпоративного приложения на Java EE. 2020.
14. Быстрый старт Spring. URL: <http://spring-projects.ru/projects/spring-framework/> (дата обращения: 23.11.2023).
15. Шейман Р., Вагнер В. Spring JPA. Полное руководство. 2020.
16. ORM // Википедия URL: <https://ru.wikipedia.org/wiki/ORM> (дата обращения: 24.11.2023).