МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) Институт информационных технологий
Кафедра Математическое и программное обеспечение ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Java-программирование»

на тему «Программирование на языке Java»

Выполнил студент группы:
1ИСб-00-01оп-21
группа

Направления подготовки (специальности)
09.03.02 Информационные системы
и технологии
шифр, наименование

Аксенов Иван Константинович
фамилия, имя, отчество

Руководитель:
Осколков Василий Михайлович
фамилия, имя, отчество

доцент
должность

Дата представления работы
«_____»_____2025 г.

Заключение о допуске к защите

Оценка_____, _____
количество баллов

Подпись преподавателя_____

Череповец, 2025 год

Оглавление

Введение

В условиях стремительного развития информационных технологий веб-приложения становятся неотъемлемой частью многих сфер жизни, обеспечивая эффективную автоматизацию процессов и улучшение взаимодействия с пользователями. Современные инструменты разработки веб-приложений предоставляют разработчикам возможность создавать высокоэффективные, масштабируемые и удобные системы, которые решают широкий спектр задач. Одними из самых популярных технологий для создания таких приложений являются фреймворки Spring и Vaadin.

В рамках данной курсовой работы разрабатывается веб-приложение, которое реализует систему «Магазин мониторов», предназначенную для управления продаж мониторов в магазине. Это приложение включает функционал для работы с городами, покупателями и продажами, предоставляя пользователю удобные средства для просмотра, поиска и фильтрации данных.

Использование связки технологий Spring и Vaadin позволяет создать интегрированную систему, в которой серверная логика и пользовательский интерфейс работают слаженно, обеспечивая стабильную работу и удобство взаимодействия. Важнейшими задачами, решаемыми в рамках разработки, являются оптимизация процесса обработки запросов и создание интуитивно понятного интерфейса для пользователей и администраторов системы.

1. Описание фреймворка Spring

Spring – фреймворк с открытым исходным кодом, написанный на Java. Его можно использовать для разработки на всех этих языках [1].

Spring предоставляет огромный набор инструментов и библиотек, которые упрощают и ускоряют процесс разработки, позволяя сосредоточиться на бизнес-логике приложения [1].

Фреймворк Spring программисты создали для написания бэкенда в веб-разработке. Но используют его во многих проектах как для создания десктопных, так и мобильных приложений. Почти всегда это энтерпрайз, то есть, создание крупных корпоративных порталов [1].

С помощью Spring разработчики могут быстро создавать масштабируемые и надежные приложения, а также использовать преимущества таких технологий, как Spring Boot, Spring Data и Spring Security [1].

Ключевая особенность Spring – в разнообразии возможностей. Это не один фреймворк, а целый набор готовых решений. В его состав входят дополнительные модули и библиотеки, которые совместно работают и регулярно пополняются [1].

Также Spring помогает писать код в парадигме аспектно-ориентированного программирования [1].

Основные модули и компоненты Spring Framework

1. Inversion of Control

Это базовый модуль Spring Framework, который отвечает за управление зависимостями. С его помощью можно объединить модули проекта в единую архитектуру. Ключевая особенность IoC состоит в том, что мы только предоставляем нужные зависимости контейнеру, которые потом автоматически подставляются в нужные места [1].

Благодаря технологии Dependency lnjection зависимости хранятся не в объектах, а в отдельных контейнерах. Это позволяет с легкостью вносить

изменения в проект: взаимодействие между компонентами не будет нарушено [1].

2. Аспектно-ориентированное программирование

Аспектно-ориентированное программирование (АОП) в Spring основано на концепции разделения бизнес-логики и аспектов, таких как безопасность, обработка ошибок. АОП позволяет добавлять эти аспекты в код без изменения его основной структуры, что потом упрощает внесение исправлений в приложение [1].

Spring поддерживает АОП через механизм AspectJ, который позволяет создавать аспекты, описывающие дополнительные функции, и применять их к требуемым классам или методам [1].

Для использования АОП в Spring необходимо определить аспекты и указать, к каким классам или методам они должны применяться. Это можно сделать с помощью аннотаций или конфигурационных файлов. Также можно выбрать, чтобы Spring добавлял аспекты во время компиляции приложения [1].

3. Доступ к данным

Этот модуль состоит из нескольких библиотек. Их использование дает возможность интегрировать код на Java с базами данных. С помощью фреймворка можно наладить взаимодействие, чтобы программа могла получить доступ к хранилищу информации, и он был безопасным [1].

Фреймворк работает на основе стандарта Java DataBase Connectivity. Эта технология помогает соединить базу данных через драйвер с использованием уникального URL [1].

Один из подходов, который используется для упрощения работы с реляционными базами данных в рамках объектно-ориентированного программирования, это ORM. В Spring также есть другие возможности, как например Data Access Object [1].

4. Транзакции

Транзакции – это последовательность операций, выраженная через несколько запросов, которые должны быть выполнены атомарно: то есть либо все, либо ни одна. Система собирает несколько запросов в один и отправляет их в базу данных [1].

У Spring Framework есть инструменты для безопасной работы с транзакциями. Этот модуль гарантирует сохранность данных и поддержку вложенных, локальных и глобальных транзакций [1].

5. MVC

Это шаблон для проектирования веб-приложений по системе Model-View-Controller. То есть при создании нового проекта его делят на [1]:

- Модель – данные, которые использует приложение для своей функциональности;
- Отображение – пользовательский интерфейс;
- Контроллер – принципы изменения модели в зависимости от действий пользователя.

В Spring Framework этот шаблон добавили относительно поздно. Зато этот модуль приобрел много дополнительных функций. Например, возможность привязать функциональность к выбранному интерфейсу, разделение слоев и замену интерфейсов [1].

На рис. 1 показана архитектура паттерна MVC, которую обеспечивает Spring Framework.

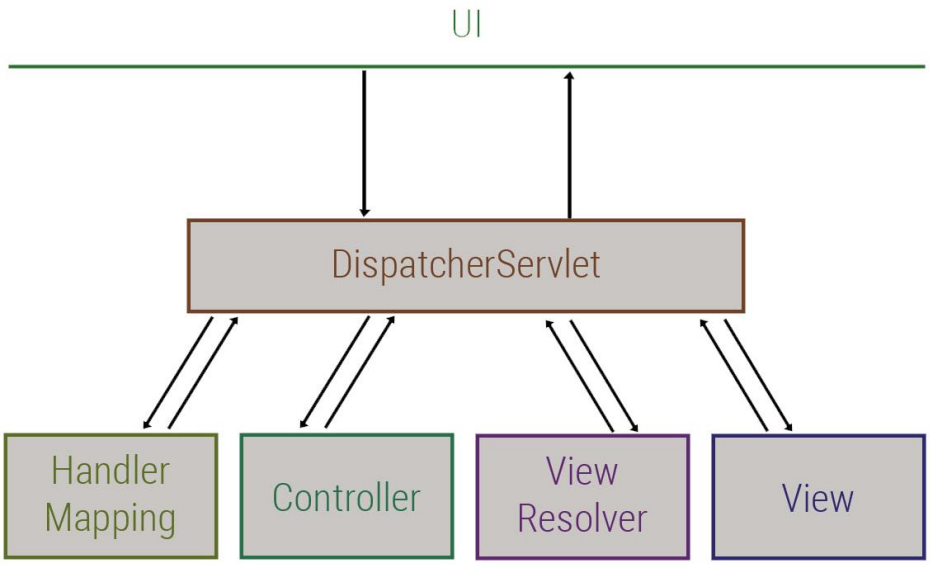

Рис. 1. Архитектура паттерна MVC

2. Описание фреймворка Vaadin

Vaadin – это фреймворк с открытым исходным кодом для создания современных веб-приложений с использованием Java. В отличие от традиционных фронтенд-фреймворков, таких как Angular или React, Vaadin позволяет разрабатывать полноценные веб-приложения на Java, управляя как серверной, так и клиентской частью приложения. Это значительно упрощает процесс разработки для Java-разработчиков, поскольку им не нужно осваивать JavaScript или другие технологии для создания динамических и интерактивных интерфейсов.

Vaadin помогает создавать современные, отзывчивые и производительные пользовательские интерфейсы, используя исключительно Java-код на серверной стороне, в то время как для клиента автоматически генерируются HTML, CSS и JavaScript. Этот подход позволяет существенно ускорить процесс разработки, сократить количество багов и улучшить поддержку кода.

Особенности Vaadin:

- Одноплатформенная разработка – весь код пишется на Java, что позволяет легко интегрировать фронтенд и бэкенд.
- Поддержка реальных веб-компонентов – Vaadin использует веб-компоненты для создания элементов интерфейса, что делает приложение более гибким и расширяемым;
- Автоматическая генерация клиентских ресурсов – Vaadin автоматически генерирует HTML, CSS и JavaScript, что избавляет от необходимости вручную работать с этими языками;
- Интерфейсы с высокой отзывчивостью – все компоненты Vaadin имеют высокую производительность и поддерживают адаптивность интерфейсов, что позволяет приложениям работать на различных устройствах и экранах.

Основные компоненты и возможности Vaadin

1. UI (User Interface) В Vaadin основное внимание уделяется созданию веб-интерфейса. Вместо того чтобы работать с отдельными компонентами

HTML, разработчики пишут код на Java, который автоматически генерирует нужный HTML для интерфейса. Весь интерфейс приложения организуется в виде объектов Java, что делает код читаемым и хорошо поддерживаемым. Vaadin использует концепцию «UI», которая инкапсулирует всю логику отображения и взаимодействия с пользователем.

2. Vaadin Components Одной из ключевых особенностей фреймворка Vaadin является набор компонентов для построения интерфейса, таких как кнопки, текстовые поля, таблицы, графики и т.д. Эти компоненты инкапсулируют все поведение UI-элементов, такие как обновление данных, обработка событий и визуальное отображение. С помощью Vaadin можно создавать как простые, так и сложные пользовательские интерфейсы с минимумом кода.

3. Data Binding Vaadin предлагает мощные средства для привязки данных (data binding), что упрощает работу с различными источниками данных. Это позволяет автоматически синхронизировать данные между моделью и пользовательским интерфейсом, а также автоматически обновлять UI при изменении данных в базе данных. Привязка данных значительно ускоряет процесс разработки, поскольку не нужно вручную отслеживать изменения и обновлять элементы интерфейса.

4. Routing Vaadin включает мощную систему маршрутизации для веб-приложений. Это позволяет легко управлять навигацией по страницам и различными представлениями, что делает приложение более структурированным и удобным для пользователя. С помощью маршрутизации можно создавать многостраничные веб-приложения, а также делать динамическую загрузку контента и подстраивать отображение страниц в зависимости от URL.

5. Layouts Vaadin предоставляет несколько типов макетов (layouts) для удобного расположения компонентов на странице, таких как HorizontalLayout, VerticalLayout, GridLayout, и другие. Это позволяет легко создавать адаптивные и отзывчивые интерфейсы, которые подстраиваются

под размер экрана. Также есть возможность создания кастомных макетов для удовлетворения специфических потребностей интерфейса.

Vaadin Flow – это основная технология для построения серверных веб-приложений. Она использует подход, при котором вся логика приложения размещается на сервере, а интерфейс автоматически обновляется на клиенте с использованием WebSocket или Server Push. Это подход позволяет эффективно работать с состоянием приложения и обеспечивать мгновенные обновления интерфейса без перезагрузки страницы. Vaadin Flow работает в связке с компонентами Vaadin и предоставляет разработчикам возможность писать бизнес-логику в одном месте, минимизируя количество ошибок.

Vaadin Fusion позволяет создавать гибридные приложения, сочетая серверную логику с клиентским кодом, написанным на TypeScript и JavaScript. Это позволяет использовать возможности фронтенд-разработки, такие как реактивные интерфейсы, с преимуществами использования Java на сервере. Такой подход идеально подходит для создания одностраничных приложений (SPA), где клиент и сервер работают в тесной интеграции.

Security Vaadin поддерживает различные механизмы безопасности для защиты приложений от несанкционированного доступа. Это включает интеграцию с Spring Security для аутентификации и авторизации, возможность добавления различных уровней доступа к компонентам и действиям внутри приложения.

Theming Vaadin предоставляет встроенные средства для создания и применения тем в приложении, чтобы настроить внешний вид интерфейса. Это позволяет изменять цветовую палитру, шрифты, стили и даже полностью адаптировать интерфейс под требования бренда. Применение тем в Vaadin также упрощает создание интерфейсов, подходящих для различных устройств и платформ.

Архитектура Vaadin Framework показана на рис. 2.

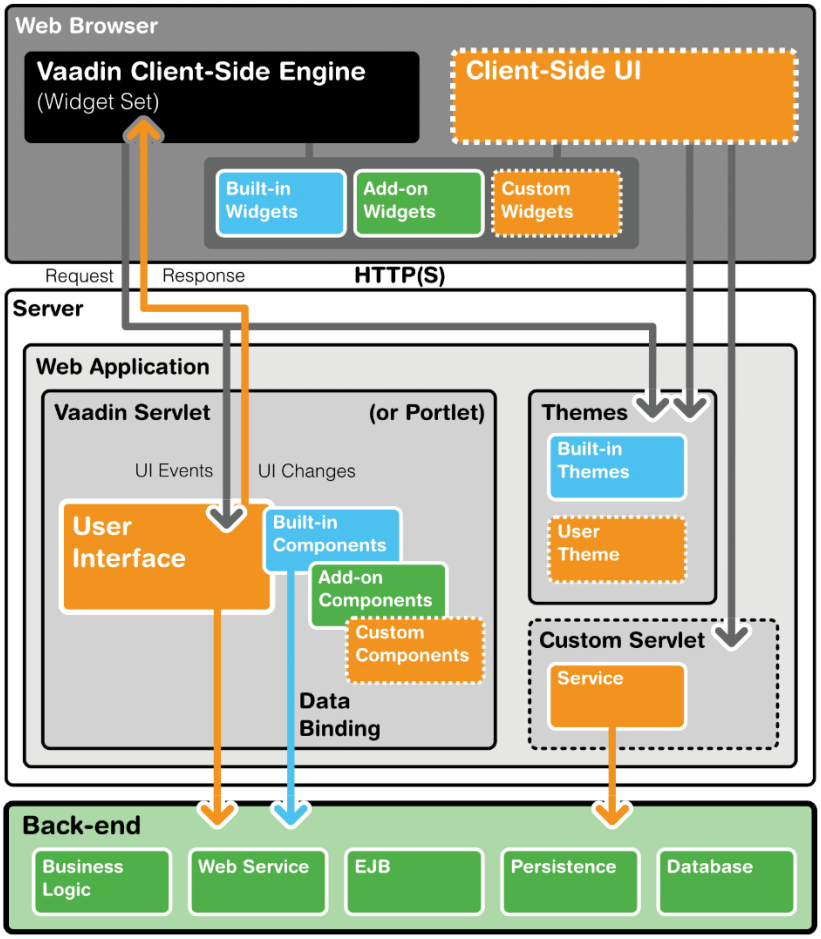

Рис. 2. Архитектура Vaadin Framework

3. Описание Spring Boot

Spring Boot – это расширение фреймворка Spring, предназначенное для упрощения процесса разработки приложений на Java. Основная цель Spring Boot – сделать процесс настройки и запуска приложений проще и быстрее, устраняя необходимость в сложной конфигурации, которая обычно требуется при использовании стандартного Spring Framework.

С помощью Spring Boot разработчики могут быстро начать работу с приложением, без необходимости вручную настраивать компоненты и зависимые библиотеки. Spring Boot позволяет сосредоточиться на бизнес-логике, минимизируя усилия, связанные с настройкой инфраструктуры приложения. Кроме того, Spring Boot автоматически конфигурирует множество аспектов работы приложения, таких как база данных, безопасность, логирование, кеширование и другие.

Основные особенности Spring Boot

1. Автоконфигурация

Одной из ключевых особенностей Spring Boot является автоконфигурация. Вместо того чтобы вручную конфигурировать все компоненты приложения, Spring Boot использует принцип автоконфигурации, который автоматически настраивает компоненты приложения на основе зависимостей, найденных в класспате. Например, если в проекте используется база данных, Spring Boot автоматически настроит подключение к базе данных и создаст необходимые компоненты для взаимодействия с ней.

Благодаря автоконфигурации разработчики не должны беспокоиться о настройках для самых популярных библиотек, таких как базы данных, кеширование, мессенджеры и другие. Spring Boot подберет наиболее подходящие настройки автоматически.

2. Встроенные серверы (Embedded Servers)

Одним из важнейших достоинств Spring Boot является возможность встраивания веб-сервера прямо в приложение. Вместо того чтобы отдельно настраивать и запускать сервер (например, Tomcat или Jetty), Spring Boot позволяет запускать приложение с уже встроенным сервером. Это позволяет значительно упростить процесс развертывания и запуска приложения.

С помощью Spring Boot можно использовать встроенные серверы, такие как Tomcat, Jetty или Undertow, и запускать приложение как самостоятельное Java-приложение. Это особенно удобно для микросервисов, которые часто разрабатываются с использованием Spring Boot.

3. Spring Boot Starter Projects

Spring Boot Starter – это набор преднастроенных зависимостей для различных типов приложений. Эти стартеры представляют собой наборы библиотек и конфигураций, которые предназначены для решения определенных задач.

Использование стартеров позволяет быстро подключать и настраивать нужные компоненты в проекте, существенно ускоряя процесс разработки.

4. Spring Boot Actuator

Spring Boot Actuator – это набор встроенных инструментов для мониторинга и управления приложением на различных стадиях его жизненного цикла. С помощью Actuator можно получить информацию о состоянии приложения, его производительности, загрузке, метриках и многом другом. Это особенно полезно для микросервисов и крупных распределенных систем.

Actuator включает такие функции, как:

- Health check – проверка состояния здоровья приложения;
- Metrics – мониторинг различных метрик работы приложения, таких как время отклика, использование памяти, количество запросов;
- Logging – доступ к журналам приложения и возможность настройки логирования в реальном времени.

5. Безопасность

Spring Security – это мощная и гибкая библиотека для обеспечения безопасности приложений, и она легко интегрируется с Spring Boot. Для большинства стандартных сценариев безопасности, таких как аутентификация и авторизация пользователей, достаточно минимальной настройки. Также в Spring Boot можно подключить поддержку различных механизмов безопасности, например, OAuth2, LDAP, а также настроить доступ к приложениям через HTTPS.

6. Микросервисная архитектура

Spring Boot идеально подходит для разработки микросервисных приложений. Его особенности, такие как простота развертывания, возможность работы с различными базами данных и эффективная интеграция с другими сервисами через REST API, делают его отличным выбором для создания микросервисов. Spring Cloud дополняет возможности Spring Boot для создания распределенных приложений и масштабируемых систем.

4. Описание созданного приложения

4.1. Постановка задачи

Предметной областью данной курсовой работы является тема «Магазин мониторов». В условиях современного рынка электроники магазины, специализирующиеся на продаже мониторов, сталкиваются с необходимостью автоматизации процессов управления ассортиментом и продаж. Развитие технологий и рост конкуренции требуют от таких магазинов внедрения эффективных систем управления запасами, отслеживания актуальных моделей и характеристик продукции.

В рамках данной работы разработано веб-приложение для управления магазином мониторов. Приложение предназначено для автоматизации процессов управления ассортиментом, учета товаров и продаж.

Целью разработки является создание программного обеспечения, включающего следующие функции:

- Управление данными о товарах: добавление новых моделей мониторов, их характеристик и описания, а также удаление устаревших позиций из ассортимента;
- Анализ продаж: сбор статистических данных о наиболее популярных моделях, периодичности покупок и предпочтениях клиентов.

Приложение будет доступно через веб-интерфейс, обеспечивая удобный доступ администраторам и сотрудникам магазина. Все данные будут представлены в удобных формах, включая таблицы и графики для анализа ключевых показателей эффективности работы магазина.

4.2. Проектирование базы данных

Проектирование базы данных – это процесс определения структуры и организации данных, которые будут храниться в базе данных. Этот этап включает создание схемы базы данных, определение таблиц, полей, связей

между таблицами, а также индексов и ограничений. Проектирование базы данных является критически важным шагом, так как оно напрямую влияет на производительность, масштабируемость и удобство поддержки приложения.

Проектирование базы данных начинается с логического проектирования, а именно создания ER-диаграммы.

ER-диаграмма – это графическое представление модели «Сущность – Связь». Она состоит из сущностей, связей и атрибутов, которые отображаются в виде прямоугольников и линий. В ER-диаграмме также используются различные символы и обозначения для указания типов связей и других аспектов моделирования.

ER-диаграмма была создана в программе Draw.io и представлена на рис. 3.

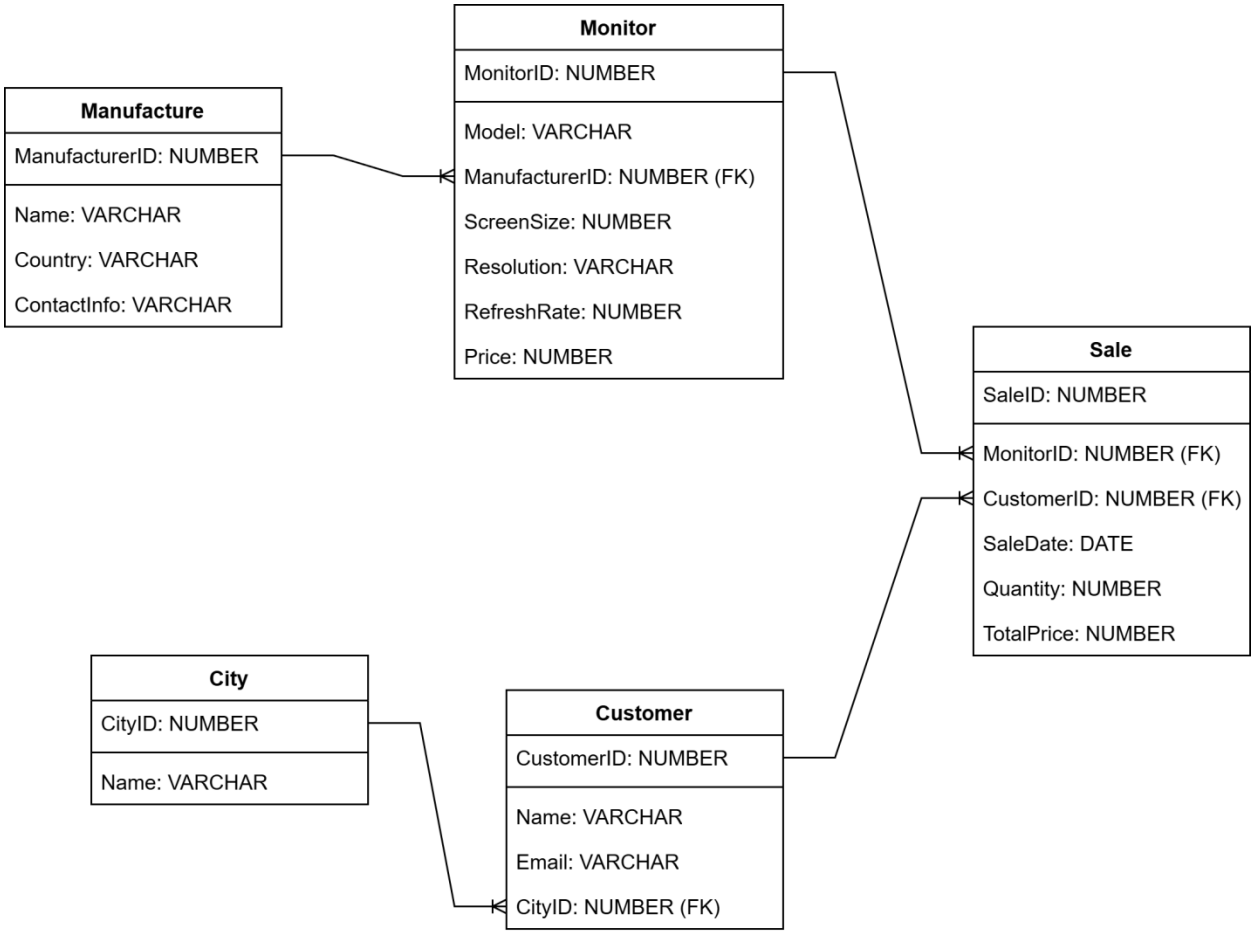

Рис. 3. ER-диаграмма предметной области

Для получения базы данных нужно провести процесс физического проектирования.

Для работы была выбрана СУБД H2. Она была выбрана для данного проекта по ряду причин, включая её легковесность, простоту интеграции с Java, поддержку SQL, а также возможность быстрого тестирования и разработки в условиях ограниченных ресурсов.

Основываясь на ER-диаграмме, была создана база данных. В табл. 1 представлены таблицы, их описание, поля и типы данных.

Таблица 1

Физическая модель

Таблица	Описание	Имя поля	Тип данных
Manufacture	Таблица хранит информацию о производителях продукции.	ManufacturerID	NUMBER
		Name	VARCHAR
		Country	VARCHAR
		ContactInfo	VARCHAR
Monitor	Таблица хранит информацию о моделях мониторов.	MonitorID	NUMBER
		Model	VARCHAR
		ManufacturerID	NUMBER (FK)
		ScreenSize	NUMBER
		Resolution	VARCHAR
		RefreshRate	NUMBER
		Price	NUMBER
Sale	Таблица хранит информацию о продажах мониторов.	SaleID	NUMBER
		MonitorID	NUMBER (FK)
		CustomerID	NUMBER (FK)
		SaleDate	DATE
		Quantity	NUMBER
		TotalPrice	NUMBER
City	Таблица хранит информацию о городах.	CityID	NUMBER
		Name	VARCHAR
Customer	Таблица хранит информацию о покупателях.	CustomerID	NUMBER
		Name	VARCHAR
		Email	VARCHAR
		CityID	NUMBER (FK)

4.3. Логическое проектирование приложения

В данном проекте будет использоваться паттерн MVC с использованием Spring и Vaadin.

Это шаблон для проектирования веб-приложений по системе Model-View-Controller. То есть при создании нового проекта его делят на [1]:

• Модель – данные, которые использует приложение для своей функциональности;

• Отображение – пользовательский интерфейс;

• Контроллер – принципы изменения модели в зависимости от действий пользователя (рис. 4).

В данном проекте будет использоваться паттерн MVC с использованием Spring и Vaadin.

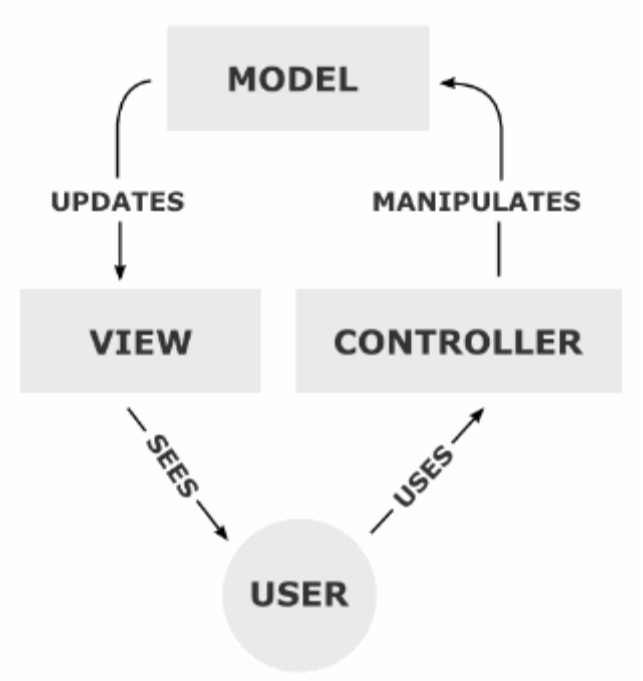

Рис. 4. Паттерн MVC

4.4. Диаграмма классов

Диаграмма классов – это важный элемент объектно-ориентированного проектирования, отображающий структуру системы с точки зрения ее сущностей и взаимосвязей между ними. В контексте разработки

электронного приложения для управления очередями в сервисном центре, диаграмма классов помогает определить ключевые сущности, их свойства, а также связи между ними [4].

В приложении для управления магазина ключевыми сущностями являются: «Города», «Покупатели», «Производители», «Мониторы», и «Продажи». Эти сущности будут представлены как классы в коде и взаимодействуют между собой через ассоциации, что позволяет поддерживать бизнес-логику и хранение данных в базе. Описание классов предоставлено в табл. 2 - 35

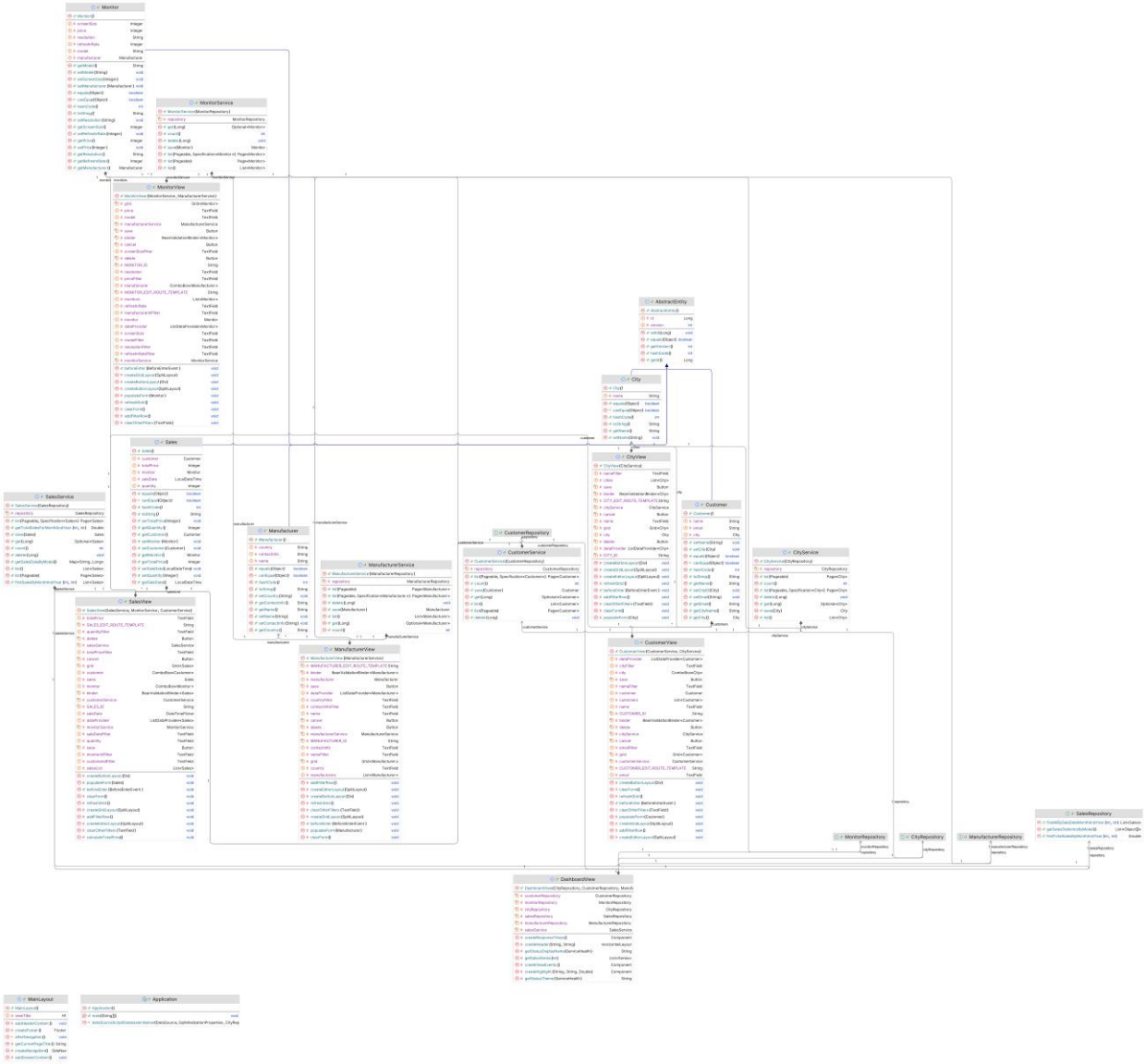

Рис. 5. Диаграмма классов

Таблица 2

Описание полей класса «Monitor»

Название поля	Тип данных	Описание поля
screenSize	Integer	Размер экрана монитора
price	Integer	Цена монитора
resolution	String	Разрешение экрана
refreshRate	Integer	Частота обновления экрана
model	String	Модель монитора
manufacturer	Manufacturer	Производитель монитора

Таблица 3

Описание методов класса «Monitor»

Название метода	Параметры	Возвращаемое значение	Описание метода
getModel	-	String	Возвращает модель монитора
setResolution	String	void	Устанавливает разрешение экрана
getScreenSize	-	Integer	Возвращает размер экрана
setScreenSize	Integer	void	Устанавливает размер экрана
getPrice	-	Integer	Возвращает цену монитора
setPrice	Integer	void	Устанавливает цену монитора
getRefreshRate	-	Integer	Возвращает частоту обновления
setRefreshRate	Integer	void	Устанавливает частоту обновления
getManufacturer	-	Manufacturer	Возвращает производителя
setManufacturer	Manufacturer	void	Устанавливает производителя
equals	Object	boolean	Проверяет равенство объектов
canEqual	Object	boolean	Проверяет, могут ли объекты быть равны
hashCode	-	int	Возвращает хэш-код объекта
toString	-	String	Возвращает строковое представление объекта

Таблица 4

Описание полей класса «AbstractEntity»

Название поля	Тип данных	Описание поля
id	Long	Уникальный идентификатор сущности
version	int	Номер версии сущности

Таблица 5

Описание методов класса «AbstractEntity»

Название метода	Параметры	Возвращаемое значение	Описание метода
setId	Long	void	Устанавливает идентификатор сущности
equals	Object	boolean	Проверяет равенство объектов
canEqual	Object	boolean	Проверяет, могут ли объекты быть равны
hashCode	-	int	Возвращает хэш-код объекта

Таблица 6

Описание полей класса «Sales»

Название поля	Тип данных	Описание поля
customer	Customer	Покупатель
monitor	Monitor	Монитор
saleDate	LocalDateTime	Дата продажи
quantity	Integer	Количество проданных единиц
totalPrice	Integer	Общая цена продажи

Таблица 7

Описание методов класса «Sales»

Название метода	Параметры	Возвращаемое значение	Описание метода
getCustomer	-	Customer	Возвращает покупателя
setCustomer	Customer	void	Устанавливает покупателя
getMonitor	-	Monitor	Возвращает монитор
setMonitor	Monitor	void	Устанавливает монитор
getSaleDate	-	LocalDateTime	Возвращает дату продажи
setSaleDate	LocalDateTime	void	Устанавливает дату продажи
getQuantity	-	Integer	Возвращает количество единиц
setQuantity	Integer	void	Устанавливает количество единиц
getTotalPrice	-	Integer	Возвращает общую цену продажи
setTotalPrice	Integer	void	Устанавливает общую цену продажи
equals	Object	boolean	Проверяет равенство объектов
canEqual	Object	boolean	Проверяет, могут ли объекты быть равны
hashCode	-	int	Возвращает хэш-код объекта
toString	-	String	Возвращает строковое представление объекта

Таблица 8

Описание полей класса «SalesService»

Название поля	Тип данных	Описание поля
repository	SalesRepository	Репозиторий для работы с данными о продажах

Таблица 9

Описание методов класса «SalesService»

Название метода	Параметры	Возвращаемое значение	Описание метода
get	Long	Optional<Sales>	Возвращает продажу по идентификатору
save	Sales	Sales	Сохраняет продажу
delete	Long	void	Удаляет продажу по идентификатору
list	Pageable	Page<Sales>	Возвращает страницу продаж
list	Pageable, Specification<Sales>	Page<Sales>	Возвращает страницу продаж по спецификации
count	-	int	Возвращает количество продаж
list	-	List<Sales>	Возвращает список продаж
findSalesByMonthAndYear	int, int	List<Sales>	Возвращает список продаж за месяц и год

Таблица 10

Описание методов класса «SalesRepository»

Название метода	Параметры	Возвращаемое значение	Описание метода
findAllBySaleDateMonthAndYear	int, int	List<Sales>	Возвращает список продаж за месяц и год
getSalesStatisticsByModel	-	List<Object[]>	Возвращает статистику продаж
findAll	-	List<Sales>	Возвращает список продаж

Таблица 11

Описание полей класса «SalesView»

Название поля	Тип данных	Описание поля
grid	Grid<Sales>	Таблица для отображения продаж
monitor	ComboBox<Monitor>	Выпадающий список для выбора монитора
customer	ComboBox<Customer>	Выпадающий список для выбора покупателя
saleDate	DateTimePicker	Выбор даты и времени продажи

quantity	TextField	Текстовое поле для ввода количества
totalPrice	TextField	Текстовое поле для отображения общей цены
cancel	Button	Кнопка "Отмена"
save	Button	Кнопка "Сохранить"
delete	Button	Кнопка "Удалить"
binder	BeanValidationBinder<Sales>	Валидатор для Sales
sales	Sales	Текущий редактируемый объект Sales
salesList	List<Sales>	Список объектов Sales
dataProvider	ListDataProvider<Sales>	Провайдер для данных
salesService	SalesService	Сервис для работы с данными продаж
monitorService	MonitorService	Сервис для работы с данными мониторов
customerService	CustomerService	Сервис для работы с данными покупателей
SALES_ID	String	Строка идентификатора продаж
SALES_EDIT_ROUTE_TEMPLATE	String	Шаблон строки для редактирования продаж

Таблица 12

Описание методов класса «SalesView»

Название метода	Параметры	Возвращаемое значение	Описание метода
beforeEnter	BeforeEnterEvent	void	Обработка события перед входом на страницу
addFilterRow	-	void	Добавляет строку фильтрации
createEditorLayout	SplitLayout	void	Создает форму редактирования
createButtonLayout	Div	void	Создает панель с кнопками
createGridLayout	SplitLayout	void	Создает сетку
refreshGrid	-	void	Обновляет данные в сетке
clearForm	-	void	Очищает форму
populateForm	Sales	void	Заполняет форму данными

Таблица 13

Описание полей класса «Manufacturer»

Название поля	Тип данных	Описание поля
id	Integer	Уникальный идентификатор
country	String	Страна производителя
contactInfo	String	Контактная информация производителя
name	String	Имя производителя

Таблица 14

Описание методов класса «Manufacturer»

Название метода	Параметры	Возвращаемое значение	Описание метода
setId	Long	void	Устанавливает идентификатор
getName	-	String	Возвращает название производителя
setName	String	void	Устанавливает название производителя
getCountry	-	String	Возвращает страну производителя
setCountry	String	void	Устанавливает страну производителя
getContactInfo	-	String	Возвращает контактную информацию
setContactInfo	String	void	Устанавливает контактную информацию
equals	Object	boolean	Проверяет равенство объектов
canEqual	Object	boolean	Проверяет, могут ли объекты быть равны
hashCode	-	int	Возвращает хэш-код объекта
toString	-	String	Возвращает строковое представление объекта

Таблица 15

Описание полей класса «ManufacturerService»

Название поля	Тип данных	Описание поля
repository	ManufacturerRepository	Репозиторий для работы с данными о производителях

Таблица 16

Описание методов класса «ManufacturerService»

Название метода	Параметры	Возвращаемое значение	Описание метода
get	Long	Optional<Manufacturer>	Возвращает производителя по идентификатору
save	Manufacturer	Manufacturer	Сохраняет производителя
delete	Long	void	Удаляет производителя по идентификатору
list	Pageable	Page<Manufacturer>	Возвращает страницу производителей
list	Pageable, Specification<Manufacturer>	Page<Manufacturer>	Возвращает страницу производителей по спецификации
list	-	List<Manufacturer>	Возвращает список производителей
count	-	int	Возвращает количество производителей

Таблица 17

Описание методов класса «ManufacturerRepository»

Название метода	Параметры	Возвращаемое значение	Описание метода
findAll	-	List<Manufacturer>	Возвращает список производителей

Таблица 18

Описание полей класса «ManufacturerRepository»

Название поля	Тип данных	Описание поля
grid	Grid<Manufacturer>	Таблица для отображения производителей
name	TextField	Текстовое поле для ввода имени
country	TextField	Текстовое поле для ввода страны
contactInfo	TextField	Текстовое поле для ввода контактной информации
cancel	Button	Кнопка "Отмена"
save	Button	Кнопка "Сохранить"
delete	Button	Кнопка "Удалить"
binder	BeanValidationBinder< Manufacturer>	Валидатор для Manufacturer
manufacturer	Manufacturer	Текущий редактируемый объект Manufacturer
manufacturers	List<Manufacturer>	Список объектов Manufacturer
dataProvider	ListDataProvider<Manufacturer>	Провайдер для данных
manufacturerService	ManufacturerService	Сервис для работы с данными производителей
MANUFACTURER_ID	String	Строка идентификатора производителя
MANUFACTURER_EDIT_ROUTE_TEMPLATE	String	Шаблон строки для редактирования производителя

Таблица 19

Описание методов класса «ManufacturerRepository»

Название метода	Параметры	Возвращаемое значение	Описание метода
beforeEnter	BeforeEnterEvent	void	Обработка события перед входом на страницу
addFilterRow		void	Добавляет строку фильтрации
createEditorLayout	SplitLayout	void	Создает форму редактирования
createButtonLayout	Div	void	Создает панель с кнопками
createGridLayout	SplitLayout	void	Создает сетку

refreshGrid	-	void	Обновляет данные в сетке
clearForm	-	void	Очищает форму
populateForm	Manufacturer	void	Заполняет форму данными

Таблица 20

Описание полей класса «Customer»

Название поля	Тип данных	Описание поля
id	Integer	Уникальный идентификатор покупателя
name	String	Имя покупателя
email	String	Email покупателя
city	City	Город покупателя

Таблица 21

Описание методов класса «Customer»

Название метода	Параметры	Возвращаемое значение	Описание метода
setId	Long	void	Устанавливает идентификатор
getName	-	String	Возвращает имя покупателя
setName	String	void	Устанавливает имя покупателя
getEmail	-	String	Возвращает email покупателя
setEmail	String	void	Устанавливает email покупателя
getCity	-	City	Возвращает город покупателя
setCity	City	void	Устанавливает город покупателя
equals	Object	boolean	Проверяет равенство объектов
canEqual	Object	boolean	Проверяет, могут ли объекты быть равны
hashCode	-	int	Возвращает хэш-код объекта
toString	-	String	Возвращает строковое представление объекта

Таблица 22

Описание полей класса «CustomerService»

Название поля	Тип данных	Описание поля
repository	CustomerRepository	Репозиторий для работы с данными о покупателях

Таблица 23

Описание методов класса «CustomerService»

Название метода	Параметры	Возвращаемое значение	Описание метода
get	Long	Optional<Customer>	Возвращает покупателя по идентификатору
save	Customer	Customer	Сохраняет покупателя
delete	Long	void	Удаляет покупателя по идентификатору
list	Pageable	Page<Customer>	Возвращает страницу покупателей
list	Pageable, Specification<Customer>	Page<Customer>	Возвращает страницу покупателей по спецификации
list	-	List<Customer>	Возвращает список покупателей
count	-	int	Возвращает количество покупателей

Таблица 24

Описание методов класса «CustomerRepository»

Название метода	Параметры	Возвращаемое значение	Описание метода
findAll	-	List<Customer>	Возвращает список покупателей
findAllByCityId	Long	List<Customer>	Возвращает всех покупателей по идентификатору города

Таблица 25

Описание полей класса «CustomerView»

Название поля	Тип данных	Описание поля
grid	Grid<Customer>	Таблица для отображения покупателей
name	TextField	Текстовое поле для ввода имени
email	TextField	Текстовое поле для ввода email
city	ComboBox<City>	Выпадающий список для выбора города
cancel	Button	Кнопка "Отмена"
save	Button	Кнопка "Сохранить"
delete	Button	Кнопка "Удалить"
binder	BeanValidationBinder<Customer>	Валидатор для Customer
customer	Customer	Текущий редактируемый объект Customer
customers	List<Customer>	Список объектов Customer
dataProvider	ListDataProvider<Custo	Провайдер для данных

	mer>	
customerService	CustomerService	Сервис для работы с данными покупателей
cityService	CityService	Сервис для работы с данными городов
CUSTOMER_ID	String	Строка идентификатора покупателя
CUSTOMER_EDIT_ROUTE_TEMPLATE	String	Шаблон строки для редактирования покупателя

Таблица 26

Описание методов класса «CustomerView»

Название метода	Параметры	Возвращаемое значение	Описание метода
beforeEnter	BeforeEnterEvent	void	Обработка события перед входом на страницу
addFilterRow	-	void	Добавляет строку фильтрации
createEditorLayout	SplitLayout	void	Создает форму редактирования
createButtonLayout	Div	void	Создает панель с кнопками
createGridLayout	SplitLayout	void	Создает сетку
refreshGrid	-	void	Обновляет данные в сетке
clearForm	-	void	Очищает форму
populateForm	Customer	void	Заполняет форму данными

Таблица 27

Описание полей класса «City»

Название поля	Тип данных	Описание поля
id	Integer	Уникальный идентификатор
name	String	Имя города

Таблица 28

Описание методов класса «City»

Название метода	Параметры	Возвращаемое значение	Описание метода
setId	Long	void	Устанавливает идентификатор
getName		String	Возвращает название города
setName	String	void	Устанавливает название города
equals	Object	boolean	Проверяет равенство объектов
canEqual	Object	boolean	Проверяет, могут ли объекты быть равны
hashCode	-	int	Возвращает хэш-код объекта
toString	-	String	Возвращает строковое представление объекта

Таблица 29

Описание полей класса «CityService»

Название поля	Тип данных	Описание поля
repository	CityRepository	Репозиторий для работы с данными о городах

Таблица 30

Описание методов класса «CityService»

Название метода	Параметры	Возвращаемое значение	Описание метода
get	Long	Optional<City>	Возвращает город по идентификатору
save	City	City	Сохраняет город
delete	Long	void	Удаляет город по идентификатору
list	Pageable	Page<City>	Возвращает страницу городов
list	Pageable, Specification <City>	Page<City>	Возвращает страницу городов по спецификации
list	-	List<City>	Возвращает список городов
count	-	int	Возвращает количество городов
getCustomersByCityId	Long	List<Customer>	Возвращает всех покупателей по идентификатору города

Таблица 31

Описание методов класса «CityRepository»

Название метода	Параметры	Возвращаемое значение	Описание метода
findAll	-	List<City>	Возвращает список городов

Таблица 32

Описание полей класса «CityView»

Название поля	Тип данных	Описание поля
grid	Grid<City>	Таблица для отображения городов
name	TextField	Текстовое поле для ввода имени
cancel	Button	Кнопка "Отмена"
save	Button	Кнопка "Сохранить"
delete	Button	Кнопка "Удалить"
binder	BeanValidationBinder<City>	Валидатор для City
city	City	Текущий редактируемый объект City
cities	List<City>	Список объектов City
dataProvider	ListDataProvider<City>	Провайдер для данных

cityService	CityService	Сервис для работы с данными городов
CITY_ID	String	Строка идентификатора города
CITY_EDIT_ROUTE_TEMPLATE	String	Шаблон строки для редактирования города

Таблица 33

Описание методов класса «CityView»

Название метода	Параметры	Возвращаемое значение	Описание метода
beforeEnter	BeforeEnterEvent	void	Обработка события перед входом на страницу
addFilterRow	-	void	Добавляет строку фильтрации
createEditorLayout	SplitLayout	void	Создает форму редактирования
createButtonLayout	Div	void	Создает панель с кнопками
createGridLayout	SplitLayout	void	Создает сетку
refreshGrid	-	void	Обновляет данные в сетке
clearForm	-	void	Очищает форму
populateForm	City	void	Заполняет форму данными

Таблица 34

Описание полей класса «DashboardView»

Название поля	Тип данных	Описание поля
cityRepository	CityRepository	Репозиторий для работы с данными о городах
customerRepository	CustomerRepository	Репозиторий для работы с данными о покупателях
manufacturerRepository	ManufacturerRepository	Репозиторий для работы с данными о производителях
monitorRepository	MonitorRepository	Репозиторий для работы с данными о мониторах
salesService	SalesService	Сервис для работы с данными о продажах
salesRepository	SalesRepository	Репозиторий для работы с данными о продажах

Таблица 35

Описание методов класса «DashboardView»

Название метода	Параметры	Возвращаемое значение	Описание метода
createHighlight	String, String, Double	Component	Создает компонент для выделения
createViewEvents	-	Component	Создает компонент для графика
createServiceHealth	-	Component	Создает компонент для

			отображения здоровья сервисов
createResponseTimes	-	Component	Создает компонент для отображения времени отклика
createHeader	String, String	HorizontalLayout	Создает заголовок
getStatusDisplayName	ServiceHealth	String	Получает строку для статуса
getStatusTheme	ServiceHealth	String	Получает тему для статуса
getSalesSeries	int	List<Series>	Получает данные для графика

4.5. Физическое проектирование

В процессе физического проектирования разрабатывается модульная структура программного обеспечения.

Модуль – это логически завершенная часть программы, которая имеет четко обозначенный функционал. Модулями могут быть файлы, пакеты, классы, функции, подпрограммы и т.п. Модули также помогают организовывать код, обеспечивая легкую навигацию при правильном наименовании, расположении и ограниченной ответственности каждого отдельного модуля [4].

Модульная структура разрабатываемого приложения представлена на рис. 6, описание представлено в табл. 36.

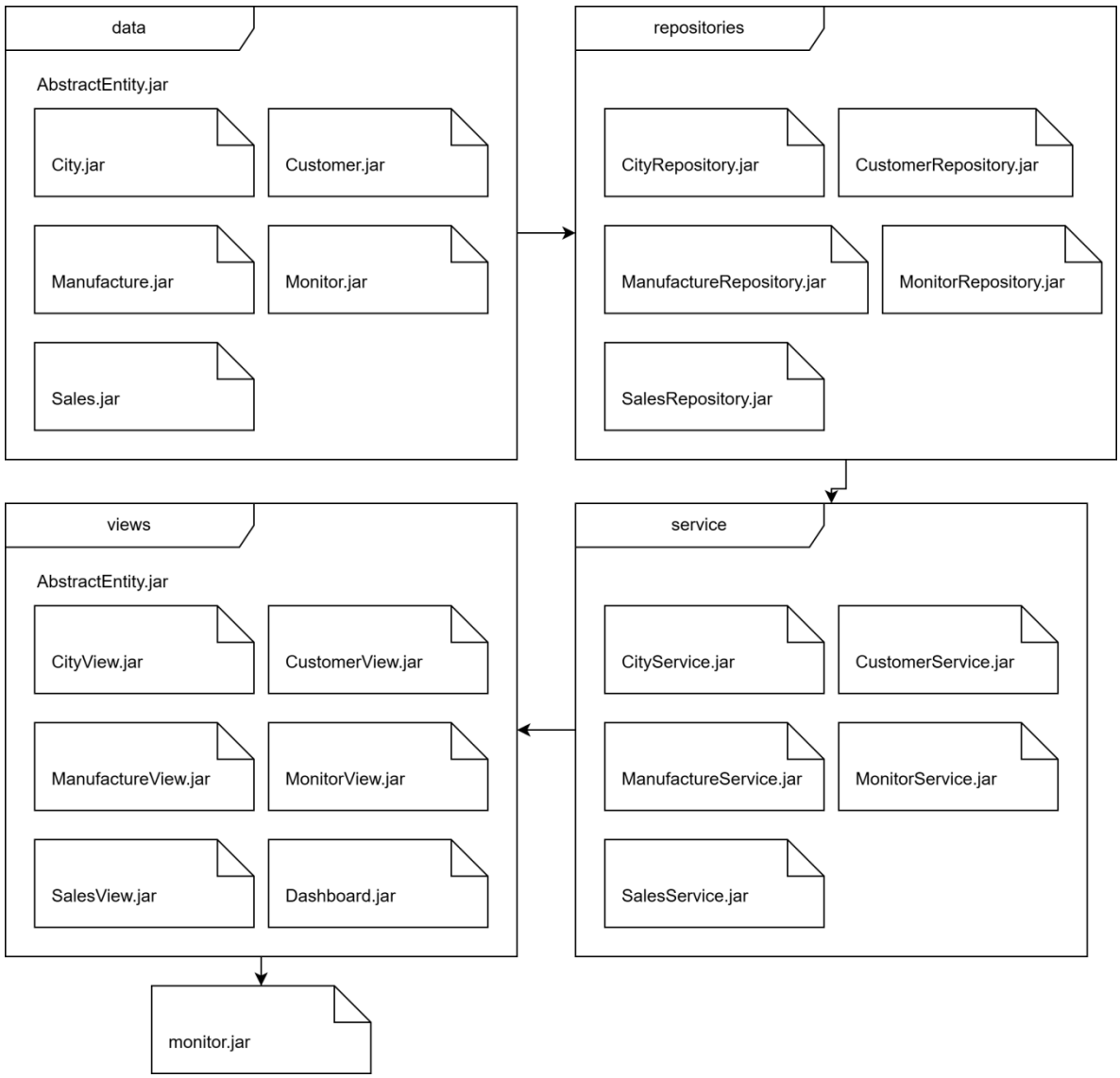

Рис. 6. Модульная структура

Таблица 36

Описание модулей

Название модуля	Описание модуля
City.jar	Модуль содержит классы, связанные с сущностью "Город" (City)
Manufacture.jar	Модуль содержит классы, связанные с сущностью "Производитель" (Manufacturer)
Monitor.jar	Модуль содержит классы, связанные с сущностью "Монитор" (Monitor)
Sales.jar	Модуль содержит классы, связанные с сущностью "Продажа" (Sales)
CityRepository.jar	Модуль содержит интерфейсы репозиториев для работы с данными о городах.
ManufacturerRepository.jar	Модуль содержит интерфейсы репозиториев для работы с данными о производителях

MonitorRepository.jar	Модуль содержит интерфейсы репозиториев для работы с данными о мониторах.
SalesRepository.jar	Модуль содержит интерфейсы репозиториев для работы с данными о продажах
CityView.jar	Модуль содержит классы для представления UI слоя городов
ManufacturerView.jar	Модуль содержит классы для представления UI слоя производителей
MonitorView.jar	Модуль содержит классы для представления UI слоя мониторов
SalesView.jar	Модуль содержит классы для представления UI слоя продаж
Dashboard.jar	Модуль содержит классы для представления UI слоя дашборда.
CityService.jar	Модуль содержит классы сервисного слоя для городов.
ManufacturerService.jar	Модуль содержит классы сервисного слоя для производителей.
MonitorService.jar	Модуль содержит классы сервисного слоя для мониторов
SalesService.jar	Модуль содержит классы сервисного слоя для продаж.
monitor.jar	Модуль содержит классы сущностей
CustomerView.jar	Модуль содержит классы для представления UI слоя покупателей
CustomerService.jar	Модуль содержит классы сервисного слоя для покупателей.
CustomerRepository.jar	Модуль содержит интерфейсы репозиториев для работы с данными о покупателях

4.6. Проектирование интерфейса

Пользовательский интерфейс реализуется с помощью фреймворка Vaadin и его компонентов. Страницы выполнены в едином стиле и имеют сквозную навигацию.

На главной странице расположена страница добавления городов. У каждой сущности есть своя страница. На рис. 7 показана страница «Города». На ней расположена таблица, кнопки для добавления/изменения/удаления записей и поля ввода и фильтры.

Рис. 7. Страница «Города»

На рис. 8 показан пример уведомления. Данные уведомления появляются при добавлении/изменении/удалении записи, а также если возникла ошибка.

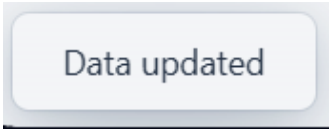

Рис. 8. Уведомление о добавлении записи

На странице дашбородов находится информация по продажах, покупателях и моделях мониторов (рис. 9).

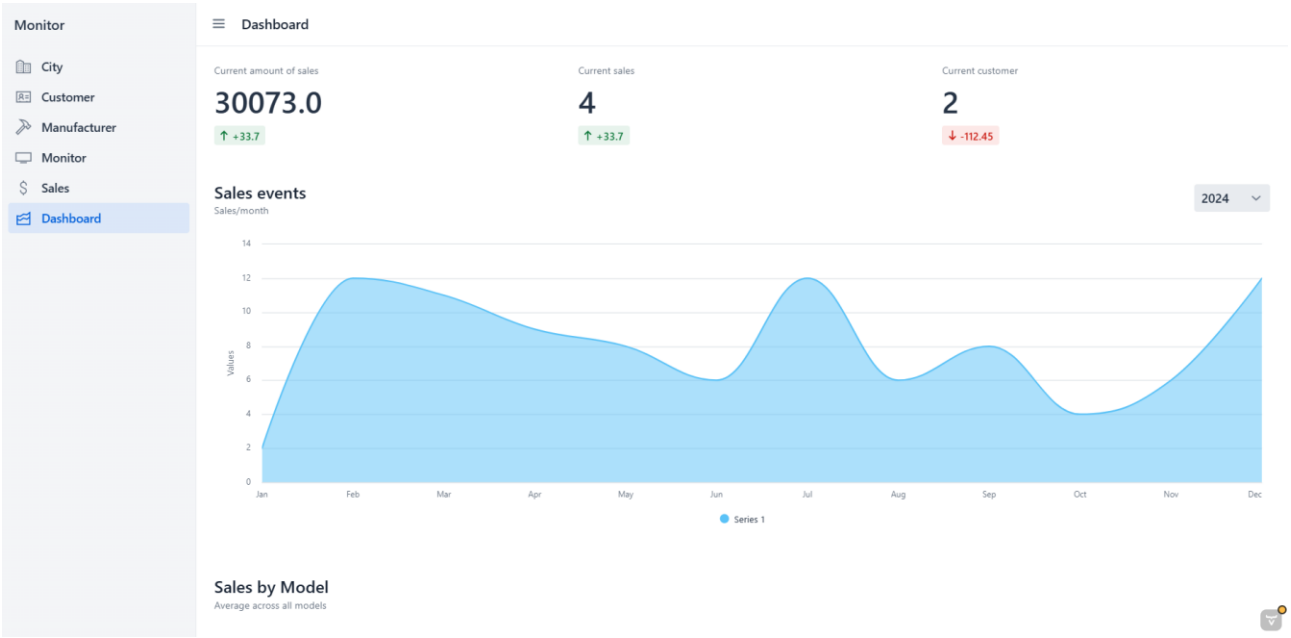

Рис. 9. Страница с дашбордами

На рис. 10. показаны хайлайты, отражающие данные о продаже на нынешний момент.

Рис. 10. Хайлайты

На рис. 11 показана линейная диаграмма, которая отражает продажи по месяцам.

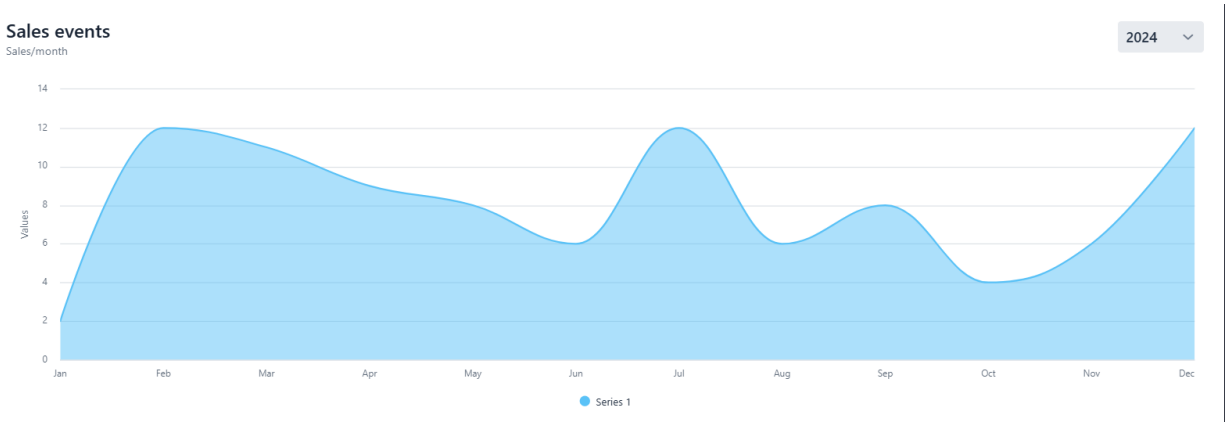

Рис. 11. Линейная диаграмма

На рис. 12 показана круговая диаграмма, которая отражает продажи по моделям за все время.

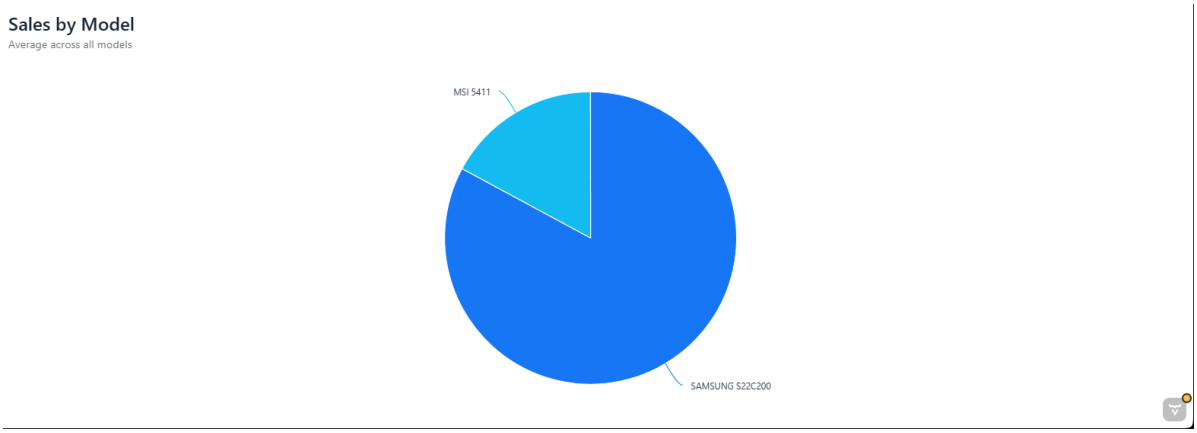

Рис. 12. Круговая диаграмма

4.7. Тестирование системы

В результате проверки программной документации были исправлены найденные грамматические и пунктуационные ошибки, было проверено соответствие документации требованиям ЕСПД и учебно-методическим пособием «Методика и организация самостоятельной работы» [1].

Результаты тестирования модулей ПО представлены в табл. 37.

Таблица 37

Результаты модульного тестирования

Дата тестирования	Тестируемая функция модуля	Кто проводил тестирование	Способ тестирования	Результаты тестирования
City.jar				
05.01.2025	CRUD операции City	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка City	Аксенов И.К.	Ручное тестирование	Успешно
Manufacture.jar				
05.01.2025	CRUD операции Manufacturer	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка	Аксенов И.К.	Ручное	Успешно

	Manufacturer		тестирование	
Monitor.jar				
05.01.2025	CRUD операции Monitor	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка Monitor	Аксенов И.К.	Ручное тестирование	Успешно
Sales.jar				
05.01.2025	CRUD операции Sales	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка Sales	Аксенов И.К.	Ручное тестирование	Успешно
CityRepository.jar				
05.01.2025	Нахождение города по id	Разработчик	Ручное тестирование	Успешно
05.01.2025	Добавление, удаление города	Разработчик	Ручное тестирование	Успешно
ManufacturerRepository.jar				
05.01.2025	Нахождение производителя по id	Разработчик	Ручное тестирование	Успешно
05.01.2025	Добавление, удаление производителя	Разработчик	Ручное тестирование	Успешно
MonitorRepository.jar				
05.01.2025	Нахождение монитора по id	Разработчик	Ручное тестирование	Успешно
05.01.2025	Добавление, удаление монитора	Разработчик	Ручное тестирование	Успешно
SalesRepository.jar				
05.01.2025	Нахождение продажи по id	Разработчик	Ручное тестирование	Успешно
05.01.2025	Добавление, удаление продажи	Разработчик	Ручное тестирование	Успешно
05.01.2025	Получение продаж по месяцу и году	Разработчик	Ручное тестирование	Успешно
CityView.jar				
05.01.2025	Отображение списка городов	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Добавление, изменение, удаление города	Аксенов И.К.	Ручное тестирование	Успешно
ManufacturerView.jar				
05.01.2025	Отображение списка производителей	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Добавление, изменение, удаление производителя	Аксенов И.К.	Ручное тестирование	Успешно
MonitorView.jar				

05.01.2025	Отображение списка мониторов	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Добавление, изменение, удаление монитора	Аксенов И.К.	Ручное тестирование	Успешно
SalesView.jar				
05.01.2025	Отображение списка продаж	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Добавление, изменение, удаление продажи	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Автоматический расчет totalPrice	Аксенов И.К.	Ручное тестирование	Успешно
Dashboard.jar				
05.01.2025	Отображение данных на панели	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Обновление данных на графиках	Аксенов И.К.	Ручное тестирование	Успешно
CityService.jar				
05.01.2025	CRUD операции City	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка городов	Аксенов И.К.	Ручное тестирование	Успешно
ManufacturerService.jar				
05.01.2025	CRUD операции Manufacturer	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка производителей	Аксенов И.К.	Ручное тестирование	Успешно
MonitorService.jar				
05.01.2025	CRUD операции Monitor	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка мониторов	Аксенов И.К.	Ручное тестирование	Успешно
SalesService.jar				
05.01.2025	CRUD операции Sales	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка продаж	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение продаж по месяцу и году	Аксенов И.К.	Ручное тестирование	Успешно
monitor.jar				
05.01.2025	Валидация полей	Аксенов И.К.	Ручное тестирование	Успешно
CustomerView.jar				
05.01.2025	Отображение списка	Аксенов И.К.	Ручное	Успешно

	покупателей		тестирование	
05.01.2025	Добавление, изменение, удаление покупателя	Аксенов И.К.	Ручное тестирование	Успешно
CustomerService.jar				
05.01.2025	CRUD операции Customer	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение списка покупателей	Аксенов И.К.	Ручное тестирование	Успешно
CustomerRepository.jar				
05.01.2025	Нахождение покупателя по id	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Добавление, удаление покупателя	Аксенов И.К.	Ручное тестирование	Успешно
05.01.2025	Получение всех покупателей по id города	Аксенов И.К.	Ручное тестирование	Успешно

Заключение

В результате выполнения курсовой работы было успешно создано веб-приложение с использованием языка Java, фреймворков Spring и Spring Boot. Был создан интерфейс с помощью фреймворка Vaadin. Приложение позволяет создавать, удалять, редактировать, сортировать и фильтровать записи. Также добавлен дашборд с информацией о количестве продаж и их распределении по моделям.

Список литературы

1. Spring – что это за фреймворк и как он устроен [Электронный ресурс]. – URL: https://ru.hexlet.io/blog/posts/spring-framework (дата обращения: 22.12.2024).

2. Spring MVC – основные принципы [Электронный ресурс] // Хабр. – URL: https://habr.com/ru/articles/336816/ (дата обращения: 25.12.2024).

3. Веб на чистой Java. Изучаем Vaadin – крутой фреймворк для создания веб-приложений [Электронный ресурс] // Хабр. – URL: https://habr.com/ru/companies/xakep/articles/244477/ (дата обращения: 05.01.2025).

4. Иванова Г.С. Технология программирования. / Г.С. Иванова - М: КноРус, 2016. - 334 с

5. Ершов Е.В., Виноградова Л.Н. Методика и организация самостоятельной работы студентов / Учебно-методическое пособие – Череповец, Череповецкий Государственный Университет, 2015. – 208 с.

Приложение 1

Техническое задание

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
наименование института (факультета)

Кафедра МПО ЭВМ
наименование кафедры

УТВЕРЖДАЮ
Зав. кафедрой МПО ЭВМ,
д.т.н., профессор _____ Ершов Е.В.
«___» _____ 20__ г.

Программирование на языке Java
Техническое задание на курсовую работу
Листов 6

Руководитель: доцент Осколков В.М.
Исполнитель: студент гр. 1ИСб-01-1оп-21
Аксенов И.К.

Череповец, 2025 год

Введение

Целью курсовой работы является написание программы на языке Java с использованием фреймворка Spring и Vaadin. Приложение должно позволять работать с базой данных моделируемой электронной очереди.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу, выданное на кафедре МПО ЭВМ ИИТ ФГБОУ ВО «Череповецкий Государственный Университет».

Дата утверждения: 11 ноября 2024 года.

Тема разработки: «Программирование на языке Java».

2. Назначение разработки

Данная программа будет предоставлять удобный интерфейс для работы с базой данных магазина мониторов.

3. Требования к разработке

3.1. Требования к функциональным характеристикам

Разработать приложение, моделирующее работу электронной очереди.

- В СУБД H2 спроектировать и создать базу данных предметной области;
- С использованием Spring Boot и Vaadin создать веб-приложение, работающее с созданной базой данных;
- Веб-приложение должно позволять:
    - просматривать таблицы БД;
    - производить добавление и удаление записей;
    - производить поиск и фильтрацию данных;

- просматривать дашборды связанные с предметной областью.

- По данным разработанной БД средствами фреймворка Vaadin создать дашборд.
- Интерфейс приложения должен быть дружественным. Графические изображения, иконки и т.д. приветствуются. Для создания интерфейса использовать фреймворк Vaadin.
- Работа всех функций должна быть проверена и результаты проверки оформлены протоколом тестирования.

3.2. Требования к надежности

Работа программного продукта должна быть стабильной. При возникновении исключений программа должна обрабатывать ошибки. Написанное приложение должно быть проверено на ошибки с помощью встроенных в среду разработки инструментов.

3.3. Условия эксплуатации

Сайт магазина мониторов предназначен для эксплуатации в стандартных офисных условиях с учетом следующих параметров:

- температура окружающей среды: от +15 до +30 °C;
- относительная влажность: от 30% до 70%;
- допустимый уровень атмосферного давления: 86-106 кПа.
- лаконичность интерфейса – минимальное количество элементов на экране;
- интерфейс должен быть интерактивным и реагировать на действия пользователя.

3.4. Требования к составу и параметрам технических средств

Минимальные системные требования к персональному компьютеру пользователя:

- процессор: Intel Core i5-4590/AMD FX 8350;
- оперативная память: 4 GB ОЗУ;
- видеокарта: NVIDIA GeForce GTX 540;

3.5. Требования к информационной и программной совместимости

Для корректной работы системы требуется использование следующих компонентов:

- операционная система Microsoft Windows 10 или аналог;
- JDK 21;

4. Требования к программной документации

4.1. Содержание программной документации

Программная документация должна содержать расчетно-пояснительную записку и приложения (текст программы, руководство пользователя).

4.2. Требования к оформлению

Требования к оформлению, установлены в ЕСПД и учебно-методическим пособием [5], а также должны быть выполнены на протяжении всей работы без каких-либо изменений.

5. Стадии и этапы разработки

Стадии и этапы разработки представлены в табл. П1.1.

Таблица П1.1

Стадии и этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
1	2	3	4
Получение задания	11.11.24	Задание получено	
Разработка ТЗ	11.11.24	Оформленное ТЗ	
Сравнительный анализ отечественных и зарубежных аналогов проектируемой системы	15.11.24	Проведен сравнительный анализ отечественных и зарубежных аналогов проектируемой системы	
Выбор технологии, среды и языка программирования	20.11.24	Проведён выбор технологии, среды и языка программирования	
Анализ процесса обработки информации, выбор методов и алгоритмов решения задач	25.11.24	Проведен анализ процесса обработки информации, выбор методов и алгоритмов решения задач	
Разработка спецификации проектируемой системы	30.11.24	Разработаны спецификации проектируемой системы	
Разработка системы	30.12.24	Разработанное программного обеспечения	
Тестирование системы	05.01.25	Приложение протестировано, ошибки исправлены	
Оформление РПЗ	23.01.25	Оформлено РПЗ	

6. Порядок контроля и приёмки

Порядок контроля и приёма представлены в таблице П1.2.

Таблица П1.2

Порядок контроля и приёма

Наименование контрольного этапа	Сроки контроля	Результат выполнения	Отметка о выполнении
1	2	3	4
Проверка ТЗ	11.11.24	Утвержденное ТЗ	
Проверка сравнительного анализа отечественных и зарубежных аналогов проектируемой системы	15.11.24	Утвержденный сравнительный анализ отечественных и зарубежных аналогов проектируемой системы	
Проверка выбранной технологии, среды и языка программирования	20.11.24	Утвержденный выбор технологии, среды и языка программирования	
Проверка анализа процесса обработки информации, выбора методов и алгоритмов решения задач	26.11.24	Утвержденный анализ процесса обработки информации, выбор методов и алгоритмов решения задач	
Демонстрация разработанной системы	15.01.25	Утверждённый прототип проектируемой системы	
Защита РПЗ	25.01.25	РПЗ защищено	

Приложение 2

Текст программы

// Application.java

```
package com.aksenov.monitor;
import com.aksenov.monitor.repositories.CityRepository;
import com.vaadin.flow.component.page.AppShellConfigurator;
import com.vaadin.flow.theme.Theme;
import javax.sql.DataSource;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.autoconfigure.sql.init.SqlDataSourceScr
iptDatabaseInitializer;
import
org.springframework.boot.autoconfigure.sql.init.SqlInitializationPr
operties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
/**
* The entry point of the Spring Boot application.
*
* Use the @PWA annotation make the application installable on
phones, tablets
* and some desktop browsers.
*
*/
@SpringBootApplication
@ComponentScan(basePackages = {"com.aksenov.monitor"})
@Theme(value = "monitor")
public class Application implements AppShellConfigurator {
public static void main(String[] args) {
SpringApplication.run(Application.class, args);
}
@Bean
SqlDataSourceScriptDatabaseInitializer
dataSourceScriptDatabaseInitializer(DataSource dataSource,
SqlInitializationProperties properties, CityRepository repository) {
// This bean ensures the database is only initialized when empty
return new SqlDataSourceScriptDatabaseInitializer(dataSource,
properties) {
@Override
public boolean initializeDatabase() {
if (repository.count() == 0L) {
return super.initializeDatabase();
}
return false;
}
};
}
}
```

// AbstractEntity.java

```
package com.aksenov.monitor.data;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import jakarta.persistence.SequenceGenerator;
import jakarta.persistence.Version;
@MappedSuperclass
public abstract class AbstractEntity {
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "idgenerator")
// The initial value is to account for data.sql demo data ids
@SequenceGenerator(name = "idgenerator", initialValue = 1000)
private Long id;
@Version
private int version;
public Long getId() {
return id;
}
public void setId(Long id) {
this.id = id;
}
public int getVersion() {
return version;
}
@Override
public int hashCode() {
if (getId() != null) {
return getId().hashCode();
}
return super.hashCode();
}
@Override
public boolean equals(Object obj) {
if (!(obj instanceof AbstractEntity that)) {
return false; // null or not an AbstractEntity class
}
if (getId() != null) {
return getId().equals(that.getId());
}
return super.equals(that);
}
}
```

// City.java

```
package com.aksenov.monitor.data;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
@Entity
@Data
@Table(name = "City")
public class City extends AbstractEntity {
@Column(name = "name")
private String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
```

// Customer.java

```
package com.aksenov.monitor.data;
import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import lombok.Data;
@Entity
@Data
@Table(name = "Customer")
public class Customer extends AbstractEntity {
private String name;
@Email
private String email;
@ManyToOne(cascade = CascadeType.DETACH)
@JoinColumn(name = "city_id", nullable = false)
private City city;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getEmail() {
return email;
}
public void setEmail(String email) {
```

```
this.email = email;
}
public City getCity() {
return city;
}
public void setCityID(City cityID) {
this.city = cityID;
}
public String getCityName() {
return city.getName();
}
}
```

// Manufacturer.java

```
package com.aksenov.monitor.data;
import jakarta.persistence.Entity;
import jakarta.persistence.Table;
import lombok.Data;
@Entity
@Data
@Table(name = "Manufacturer")
public class Manufacturer extends AbstractEntity {
private String name;
private String country;
private String contactInfo;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getCountry() {
return country;
}
public void setCountry(String country) {
this.country = country;
}
public String getContactInfo() {
return contactInfo;
}
public void setContactInfo(String contactInfo) {
this.contactInfo = contactInfo;
}
}
```

// Monitor.java

```
package com.aksenov.monitor.data;
import jakarta.persistence.*;
import lombok.Data;
@Entity
@Data
@Table(name = "Monitor")
public class Monitor extends AbstractEntity {
@Column(name = "model")
private String model;
@ManyToOne(cascade = CascadeType.DETACH)
@JoinColumn(name = "manufacturer_id", nullable = false)
private Manufacturer manufacturer;
private Integer screenSize;
@Column(name = "resolution")
private String resolution;
private Integer refreshrRate;
private Integer price;
public String getModel() {
return model;
}
public void setModel(String model) {
this.model = model;
}
public Manufacturer getManufacturer() {
return manufacturer;
}
public void setManufacturer(Manufacturer manufacturer) {
this.manufacturer = manufacturer;
}
public Integer getScreenSize() {
return screenSize;
}
public void setScreenSize(Integer screenSize) {
this.screenSize = screenSize;
}
public String getResolution() {
return resolution;
}
public void setResolution(String resolution) {
this.resolution = resolution;
}
public Integer getRefreshrRate() {
return refreshrRate;
}
public void setRefreshrRate(Integer refreshrRate) {
this.refreshrRate = refreshrRate;
}
public Integer getPrice() {
return price;
}
public void setPrice(Integer price) {
this.price = price;
}
}
```

// package-info.java

```
@NonNullApi
package com.aksenov.monitor.data;
import org.springframework.lang.NonNullApi;
```

// Sales.java

```
package com.aksenov.monitor.data;
import jakarta.persistence.*;
import lombok.Data;
import java.time.LocalDateTime;
@Entity
@Data
@Table(name = "Sales")
public class Sales extends AbstractEntity {
@ManyToOne(cascade = CascadeType.DETACH)
@JoinColumn(name = "monitor_id", nullable = false)
private Monitor monitor;
@ManyToOne(cascade = CascadeType.DETACH)
@JoinColumn(name = "customer_id", nullable = false)
private Customer customer;
private LocalDateTime saleDate;
private Integer quantity;
private Integer totalPrice;
public Monitor getMonitor() {
return monitor;
}
public void setMonitor(Monitor monitor) {
this.monitor = monitor;
}
public Customer getCustomer() {
return customer;
}
public void setCustomer(Customer customer) {
this.customer = customer;
}
public LocalDateTime getSaleDate() {
return saleDate;
}
public void setSaleDate(LocalDateTime saleDate) {
this.saleDate = saleDate;
}
public Integer getQuantity() {
return quantity;
}
public void setQuantity(Integer quantity) {
this.quantity = quantity;
}
public Integer getTotalPrice() {
return totalPrice;
}
public void setTotalPrice(Integer totalPrice) {
this.totalPrice = totalPrice;
}
}
```

// CityRepository.java

```
package com.aksenov.monitor.repositories;
import com.aksenov.monitor.data.City;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface CityRepository extends JpaRepository<City,
Long>, JpaSpecificationExecutor<City> {
}
```

// CustomerRepository.java

```
package com.aksenov.monitor.repositories;
import com.aksenov.monitor.data.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecutor;
public interface CustomerRepository extends
JpaRepository<Customer, Long>,
JpaSpecificationExecutor<Customer> {
}
```

// ManufacturerRepository.java

```
package com.aksenov.monitor.repositories;
import com.aksenov.monitor.data.Manufacturer;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecutor;
public interface ManufacturerRepository
extends
JpaRepository<Manufacturer, Long>,
JpaSpecificationExecutor<Manufacturer> {
}
```

// MonitorRepository.java

```
package com.aksenov.monitor.repositories;
import com.aksenov.monitor.data.Monitor;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecutor;
public interface MonitorRepository extends
JpaRepository<Monitor, Long>,
JpaSpecificationExecutor<Monitor> {
}
```

// SalesRepository.java

```
package com.aksenov.monitor.repositories;
import com.aksenov.monitor.data.Sales;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import java.util.List;
import java.util.Map;
public interface SalesRepository extends JpaRepository<Sales,
Long>, JpaSpecificationExecutor<Sales> {
@Query("SELECT s FROM Sales s WHERE
FUNCTION('MONTH', s.saleDate) = :month AND
FUNCTION('YEAR', s.saleDate) = :year")
List<Sales> findAllBySaleDateMonthAndYear(@Param("month")
int month, @Param("year") int year);
@Query("SELECT SUM(s.totalPrice) FROM Sales s WHERE
FUNCTION('MONTH', s.saleDate) = :month AND
FUNCTION('YEAR', s.saleDate) = :year")
Double findTotalSalesByMonthAndYear(@Param("month") int
month, @Param("year") int year);
@Query("SELECT m.model, SUM(s.totalPrice) FROM Sales s
JOIN s.monitor m GROUP BY m.model")
List<Object[]> getSalesStatisticsByModel();
}
```

// CityService.java

```
package com.aksenov.monitor.services;
import com.aksenov.monitor.data.City;
import com.aksenov.monitor.repositories.CityRepository;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
@Service
public class CityService {
private final CityRepository repository;
@Autowired
public CityService(CityRepository repository) {
this.repository = repository;
}
public Optional<City> get(Long id) {
return repository.findById(id);
}
public City save(City entity) {
return repository.save(entity);
}
public void delete(Long id) {
repository.deleteById(id);
}
public Page<City> list(Pageable pageable) {
return repository.findAll(pageable);
}
public Page<City> list(Pageable pageable, Specification<City>
filter) {
return repository.findAll(filter, pageable);
}
public int count() {
return (int) repository.count();
}
public List<City> list() {
return repository.findAll();
}
}
```

// CustomerService.java

```
package com.aksenov.monitor.services;
import com.aksenov.monitor.data.Customer;
import com.aksenov.monitor.repositories.CustomerRepository;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
@Service
public class CustomerService {
private final CustomerRepository repository;
public CustomerService(CustomerRepository repository) {
this.repository = repository;
}
public Optional<Customer> get(Long id) {
return repository.findById(id);
}
public Customer save(Customer entity) {
return repository.save(entity);
}
public void delete(Long id) {
repository.deleteById(id);
}
public Page<Customer> list(Pageable pageable) {
return repository.findAll(pageable);
}
public Page<Customer> list(Pageable pageable,
Specification<Customer> filter) {
return repository.findAll(filter, pageable);
}
public int count() {
return (int) repository.count();
}
public List<Customer> list() {
```

```
return repository.findAll();
}
}
```

// ManufacturerService.java

```
package com.aksenov.monitor.services;
import com.aksenov.monitor.data.Manufacturer;
import com.aksenov.monitor.repositories.ManufacturerRepository;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
@Service
public class ManufacturerService {
private final ManufacturerRepository repository;
public ManufacturerService(ManufacturerRepository repository) {
this.repository = repository;
}
public Optional<Manufacturer> get(Long id) {
return repository.findById(id);
}
public Manufacturer save(Manufacturer entity) {
return repository.save(entity);
}
public void delete(Long id) {
repository.deleteById(id);
}
public Page<Manufacturer> list(Pageable pageable) {
return repository.findAll(pageable);
}
public Page<Manufacturer> list(Pageable pageable,
Specification<Manufacturer> filter) {
return repository.findAll(filter, pageable);
}
public List<Manufacturer> list() {
return repository.findAll();
}
public int count() {
return (int) repository.count();
}
}
```

// MonitorService.java

```
package com.aksenov.monitor.services;
import com.aksenov.monitor.data.Customer;
import com.aksenov.monitor.data.Monitor;
import com.aksenov.monitor.repositories.MonitorRepository;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
@Service
public class MonitorService {
private final MonitorRepository repository;
public MonitorService(MonitorRepository repository) {
this.repository = repository;
}
public Optional<Monitor> get(Long id) {
return repository.findById(id);
}
public Monitor save(Monitor entity) {
return repository.save(entity);
}
public void delete(Long id) {
repository.deleteById(id);
}
public Page<Monitor> list(Pageable pageable) {
return repository.findAll(pageable);
}
public Page<Monitor> list(Pageable pageable,
Specification<Monitor> filter) {
return repository.findAll(filter, pageable);
}
public int count() {
return (int) repository.count();
}
public List<Monitor> list() {
return repository.findAll();
}
}
```

// package-info.java

```
@NonNullApi
package com.aksenov.monitor.services;
import org.springframework.lang.NonNullApi;
```

// SalesService.java

```
package com.aksenov.monitor.services;
import com.aksenov.monitor.data.Sales;
import com.aksenov.monitor.repositories.SalesRepository;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
@Service
public class SalesService {
private final SalesRepository repository;
public SalesService(SalesRepository repository) {
this.repository = repository;
}
public Optional<Sales> get(Long id) {
return repository.findById(id);
}
public Sales save(Sales entity) {
return repository.save(entity);
}
public void delete(Long id) {
repository.deleteById(id);
}
public Page<Sales> list(Pageable pageable) {
return repository.findAll(pageable);
}
public Page<Sales> list(Pageable pageable, Specification<Sales>
filter) {
return repository.findAll(filter, pageable);
}
public int count() {
return (int) repository.count();
}
public List<Sales> list() {
return repository.findAll();
}
public List<Sales> findSalesByMonthAndYear(int month, int
year) {
return repository.findAllBySaleDateMonthAndYear(month, year);
}
public Double getTotalSalesForMonthAndYear(int month, int
year) {
return repository.findTotalSalesByMonthAndYear(month, year);
}
public Map<String, Long> getSalesDataByModel() {
Map<String, Long> salesData = new HashMap<>();
List<Object[]> results = repository.getSalesStatisticsByModel();
for (Object[] result : results) {
String model = (String) result[0];
Long total = (Long) result[1];
salesData.put(model, total);
}
return salesData;
}
}
```

// CityView.java

```
package com.aksenov.monitor.views.city;
import com.aksenov.monitor.data.City;
import com.aksenov.monitor.data.Customer;
import com.aksenov.monitor.services.CityService;
```

```
import com.vaadin.flow.component.UI;
import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.button.ButtonVariant;
import com.vaadin.flow.component.combobox.ComboBox;
import com.vaadin.flow.component.datepicker.DatePicker;
import com.vaadin.flow.component.formlayout.FormLayout;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.grid.GridVariant;
import com.vaadin.flow.component.grid.HeaderRow;
import com.vaadin.flow.component.html.Div;
import com.vaadin.flow.component.notification.Notification;
import
com.vaadin.flow.component.notification.Notification.Position;
import
com.vaadin.flow.component.notification.NotificationVariant;
import
com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.splitlayout.SplitLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.binder.BeanValidationBinder;
import com.vaadin.flow.data.binder.ValidationException;
import com.vaadin.flow.data.provider.ListDataProvider;
import com.vaadin.flow.data.value.ValueChangeMode;
import com.vaadin.flow.function.SerializablePredicate;
import com.vaadin.flow.router.BeforeEnterEvent;
import com.vaadin.flow.router.BeforeEnterObserver;
import com.vaadin.flow.router.Menu;
import com.vaadin.flow.router.PageTitle;
import com.vaadin.flow.router.Route;
import com.vaadin.flow.router.RouteAlias;
import com.vaadin.flow.spring.data.VaadinSpringDataHelpers;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import org.apache.commons.lang3.StringUtils;
import
org.springframework.orm.ObjectOptimisticLockingFailureExcepti
on;
import org.vaadin.lineawesome.LineAwesomeIconUrl;
@PageTitle("City")
@Route("/:cityID?/:action?(edit)")
@Menu(order = 0, icon = LineAwesomeIconUrl.CITY_SOLID)
@RouteAlias("")
public class CityView extends Div implements
BeforeEnterObserver {
private final String CITY_ID = "cityID";
private final String CITY_EDIT_ROUTE_TEMPLATE =
"/%s/edit";
private final Grid<City> grid = new Grid<>(City.class, false);
private TextField name;
private final Button cancel = new Button("Cancel");
private final Button save = new Button("Save");
private final Button delete = new Button("Delete");
private final BeanValidationBinder<City> binder;
private City city;
private List<City> cities;
private final CityService cityService;
private ListDataProvider<City> dataProvider;
public CityView(CityService cityService) {
this.cityService = cityService;
addClassNames("city-view");
// Create UI
SplitLayout splitLayout = new SplitLayout();
createGridLayout(splitLayout);
createEditorLayout(splitLayout);
add(splitLayout);
// Configure Grid
grid.addColumn("name").setAutoWidth(true);
//grid.setItems(query ->
cityService.list(VaadinSpringDataHelpers.toSpringPageRequest(q
uery)).stream());
grid.addThemeVariants(GridVariant.LUMO_NO_BORDER);
cities = cityService.list();
dataProvider = new ListDataProvider<>(cities);
grid.setDataProvider(dataProvider);
// when a row is selected or deselected, populate form
grid.asSingleSelect().addValueChangeListener(event -> {
if (event.getValue() != null) {
UI.getCurrent().navigate(String.format(CITY_EDIT_ROUTE_TE
MPLATE, event.getValue().getId()));
} else {
clearForm();
UI.getCurrent().navigate(CityView.class);
}
});
// Configure Form
binder = new BeanValidationBinder<>(City.class);
// Bind fields. This is where you'd define e.g. validation rules
binder.bindInstanceFields(this);
addFilterRow();
cancel.addClickListener(e -> {
clearForm();
refreshGrid();
});
delete.addClickListener(e -> {
try {
if (this.city == null) {
return;
}
binder.writeBean(this.city);
cityService.delete(this.city.getId());
clearForm();
refreshGrid();
Notification.show("Data deleted");
UI.getCurrent().navigate(CityView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
save.addClickListener(e -> {
try {
if (this.city == null) {
this.city = new City();
}
if (name.isEmpty() ) {
Notification.show("Please fill all required fields");
return;
}
binder.writeBean(this.city);
cityService.save(this.city);
clearForm();
refreshGrid();
Notification.show("Data updated");
UI.getCurrent().navigate(CityView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
}
@Override
public void beforeEnter(BeforeEnterEvent event) {
Optional<Long> cityId =
event.getRouteParameters().get(CITY_ID).map(Long::parseLong);
if (cityId.isPresent()) {
Optional<City> cityFromBackend = cityService.get(cityId.get());
if (cityFromBackend.isPresent()) {
populateForm(cityFromBackend.get());
} else {
Notification.show(String.format("The requested city was not
found, ID = %s", cityId.get()), 3000,
Notification.Position.BOTTOM_START);
// when a row is selected but the data is no longer available,
// refresh grid
```

```
refreshGrid();
event.forwardTo(CityView.class);
}
}
}
private void addFilterRow() {
HeaderRow headerRow = grid.appendHeaderRow();
// Filter for name column
TextField nameFilter = new TextField();
nameFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if (filterValue.isEmpty()) {
dataProvider.clearFilters();
}
else {
dataProvider.setFilter(city ->
city.getName().toLowerCase().contains(filterValue.toLowerCase()
));
}
// Clear other filters
clearOtherFilters(nameFilter);
clearForm();
refreshGrid();
});
nameFilter.setPlaceholder("Filter");
nameFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("name")).setComponen
t(nameFilter);
}
private void clearOtherFilters(TextField currentFilter) {
if (currentFilter != nameFilter) {
nameFilter.clear();
}
}
private void createEditorLayout(SplitLayout splitLayout) {
Div editorLayoutDiv = new Div();
editorLayoutDiv.setClassName("editor-layout");
Div editorDiv = new Div();
editorDiv.setClassName("editor");
editorLayoutDiv.add(editorDiv);
FormLayout formLayout = new FormLayout();
name = new TextField("Name");
formLayout.add(name);
editorDiv.add(formLayout);
createButtonLayout(editorLayoutDiv);
splitLayout.addToSecondary(editorLayoutDiv);
}
private void createButtonLayout(Div editorLayoutDiv) {
HorizontalLayout buttonLayout = new HorizontalLayout();
buttonLayout.setClassName("button-layout");
cancel.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
save.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
delete.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
buttonLayout.add(save, cancel, delete);
editorLayoutDiv.add(buttonLayout);
}
private void createGridLayout(SplitLayout splitLayout) {
Div wrapper = new Div();
wrapper.setClassName("grid-wrapper");
splitLayout.addToPrimary(wrapper);
wrapper.add(grid);
}
private void refreshGrid() {
SerializablePredicate<City> currentFilter =
dataProvider.getFilter();
// 2. Обновить список
cities = cityService.list();
// 3. Создать новый dataProvider и применить фильтр
dataProvider = new ListDataProvider<>(cities);
grid.setDataProvider(dataProvider);
dataProvider.setFilter(currentFilter);
//  grid.select(null);
dataProvider.refreshAll();
}
private void clearForm() {
populateForm(null);
}
private void populateForm(City value) {
this.city = value;
binder.readBean(this.city);
}
private TextField nameFilter;
}
```

// CustomerView.java

```
package com.aksenov.monitor.views.customer;
import com.aksenov.monitor.data.City;
import com.aksenov.monitor.data.Customer;
import com.aksenov.monitor.services.CityService;
import com.aksenov.monitor.services.CustomerService;
import com.aksenov.monitor.views.city.CityView;
import com.vaadin.flow.component.UI;
import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.button.ButtonVariant;
import com.vaadin.flow.component.combobox.ComboBox;
import com.vaadin.flow.component.formlayout.FormLayout;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.grid.GridVariant;
import com.vaadin.flow.component.grid.HeaderRow;
import com.vaadin.flow.component.html.Div;
import com.vaadin.flow.component.notification.Notification;
import
com.vaadin.flow.component.notification.Notification.Position;
import
com.vaadin.flow.component.notification.NotificationVariant;
import
com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.splitlayout.SplitLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.binder.BeanValidationBinder;
import com.vaadin.flow.data.binder.ValidationException;
import com.vaadin.flow.data.provider.ListDataProvider;
import com.vaadin.flow.function.SerializablePredicate;
import com.vaadin.flow.router.BeforeEnterEvent;
import com.vaadin.flow.router.BeforeEnterObserver;
import com.vaadin.flow.router.Menu;
import com.vaadin.flow.router.PageTitle;
import com.vaadin.flow.router.Route;
import java.util.List;
import java.util.Optional;
import java.util.logging.Filter;
import
org.springframework.orm.ObjectOptimisticLockingFailureExcepti
on;
import org.vaadin.lineawesome.LineAwesomeIconUrl;
@PageTitle("Customer")
@Route("customer/:customerID?/:action?(edit)")
@Menu(order = 1, icon =
LineAwesomeIconUrl.ADDRESS_CARD)
public class CustomerView extends Div implements
BeforeEnterObserver {
private final String CUSTOMER_ID = "customerID";
private final String CUSTOMER_EDIT_ROUTE_TEMPLATE =
"customer/%s/edit";
private final Grid<Customer> grid = new Grid<>(Customer.class,
false);
private TextField name;
private TextField email;
private ComboBox<City> city;
private final Button cancel = new Button("Cancel");
private final Button save = new Button("Save");
private final Button delete = new Button("Delete");
private final BeanValidationBinder<Customer> binder;
private Customer customer;
private List<Customer> customers;
private final CustomerService customerService;
private final CityService cityService;
private ListDataProvider<Customer> dataProvider;
public CustomerView(CustomerService customerService,
CityService cityService) {
this.customerService = customerService;
this.cityService = cityService;
addClassNames("customer-view");
// Create UI
SplitLayout splitLayout = new SplitLayout();
createGridLayout(splitLayout);
createEditorLayout(splitLayout);
```

```
add(splitLayout);
// Configure Grid
grid.addColumn("name").setAutoWidth(true);
grid.addColumn("email").setAutoWidth(true);
grid.addColumn("city.name").setAutoWidth(true).setHeader("City
");
//grid.setItems(query ->
customerService.list(VaadinSpringDataHelpers.toSpringPageRequ
est(query)).stream());
grid.addThemeVariants(GridVariant.LUMO_NO_BORDER);
customers = customerService.list();
dataProvider = new ListDataProvider<>(customers);
grid.setDataProvider(dataProvider);
// when a row is selected or deselected, populate form
grid.asSingleSelect().addValueChangeListener(event -> {
if (event.getValue() != null) {
UI.getCurrent().navigate(String.format(CUSTOMER_EDIT_ROU
TE_TEMPLATE, event.getValue().getId()));
} else {
clearForm();
UI.getCurrent().navigate(CustomerView.class);
}
});
// Configure Form
binder = new BeanValidationBinder<>(Customer.class);
addFilterRow();
// Bind fields. This is where you'd define e.g. validation rules
binder.bindInstanceFields(this);
cancel.addClickListener(e -> {
clearForm();
refreshGrid();
});
save.addClickListener(e -> {
try {
if (this.customer == null) {
this.customer = new Customer();
}
if (name.isEmpty() || email.isEmpty() || city.isEmpty() ) {
Notification.show("Please fill all required fields");
return;
}
binder.writeBean(this.customer);
customerService.save(this.customer);
clearForm();
refreshGrid();
Notification.show("Data updated");
UI.getCurrent().navigate(CustomerView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
delete.addClickListener(e -> {
try {
if (this.city == null) {
return;
}
binder.writeBean(this.customer);
customerService.delete(this.customer.getId());
clearForm();
refreshGrid();
Notification.show("Data deleted");
UI.getCurrent().navigate(CustomerView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
}
@Override
public void beforeEnter(BeforeEnterEvent event) {
Optional<Long> customerId =
event.getRouteParameters().get(CUSTOMER_ID).map(Long::pars
eLong);
if (customerId.isPresent()) {
Optional<Customer> customerFromBackend =
customerService.get(customerId.get());
if (customerFromBackend.isPresent()) {
populateForm(customerFromBackend.get());
} else {
Notification.show(String.format("The requested customer was not
found, ID = %s", customerId.get()),
3000, Notification.Position.BOTTOM_START);
// when a row is selected but the data is no longer available,
// refresh grid
refreshGrid();
event.forwardTo(CustomerView.class);
}
}
}
private void addFilterRow() {
HeaderRow headerRow = grid.appendHeaderRow();
// Filter for name column
nameFilter = new TextField();
nameFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if (filterValue.isEmpty()) {
dataProvider.clearFilters();
}
else {
dataProvider.setFilter(customer ->
customer.getName().toLowerCase().contains(filterValue.toLowerC
ase()));
}
clearOtherFilters(nameFilter);
clearForm();
refreshGrid();
});
nameFilter.setPlaceholder("Filter");
nameFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("name")).setComponen
t(nameFilter);
// Filter for email column
emailFilter = new TextField();
emailFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(customer ->
customer.getEmail().toLowerCase().contains(filterValue.toLowerC
ase()));
}
clearOtherFilters(emailFilter);
clearForm();
refreshGrid();
});
emailFilter.setPlaceholder("Filter");
emailFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("email")).setComponen
t(emailFilter);
// Filter for cityName column
cityFilter = new TextField();
cityFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
dataProvider.setFilter(customer -> customer.getCityName() != null
&&
customer.getCityName().toLowerCase().contains(filterValue.toLo
werCase()));
clearOtherFilters(cityFilter);
clearForm();
refreshGrid();
});
cityFilter.setPlaceholder("Filter");
cityFilter.setWidthFull();
```

```
headerRow.getCell(grid.getColumnByKey("city.name")).setComp
onent(cityFilter);
}
private void clearOtherFilters(TextField currentFilter) {
if (currentFilter != nameFilter) {
nameFilter.clear();
}
if (currentFilter != emailFilter) {
emailFilter.clear();
}
if (currentFilter != cityFilter) {
cityFilter.clear();
}
}
private void createEditorLayout(SplitLayout splitLayout) {
Div editorLayoutDiv = new Div();
editorLayoutDiv.setClassName("editor-layout");
Div editorDiv = new Div();
editorDiv.setClassName("editor");
editorLayoutDiv.add(editorDiv);
FormLayout formLayout = new FormLayout();
name = new TextField("Name");
email = new TextField("Email");
city = new ComboBox<City>("City");
city.setItemLabelGenerator(currCity ->
String.valueOf(currCity.getId()));
city.setAllowCustomValue(false);
city.addFocusListener(event -> {
List<City> cityList = cityService.list();
city.setItems(cityList);
});
city.setItemLabelGenerator(City::getName);
formLayout.add(name, email, city);
editorDiv.add(formLayout);
createButtonLayout(editorLayoutDiv);
splitLayout.addToSecondary(editorLayoutDiv);
}
private void createButtonLayout(Div editorLayoutDiv) {
HorizontalLayout buttonLayout = new HorizontalLayout();
buttonLayout.setClassName("button-layout");
cancel.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
save.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
delete.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
buttonLayout.add(save, cancel, delete);
editorLayoutDiv.add(buttonLayout);
}
private void createGridLayout(SplitLayout splitLayout) {
Div wrapper = new Div();
wrapper.setClassName("grid-wrapper");
splitLayout.addToPrimary(wrapper);
wrapper.add(grid);
}
private void refreshGrid() {
SerializablePredicate<Customer> currentFilter =
dataProvider.getFilter();
// 2. Обновить список
customers = customerService.list();
// 3. Создать новый dataProvider и применить фильтр
dataProvider = new ListDataProvider<>(customers);
grid.setDataProvider(dataProvider);
dataProvider.setFilter(currentFilter);
//  grid.select(null);
dataProvider.refreshAll();
}
private void clearForm() {
populateForm(null);
}
private void populateForm(Customer value) {
this.customer = value;
// Загрузить список городов
List<City> cityList = cityService.list();
// Установить список городов в ComboBox
city.setItems(cityList);
// Установить значения клиента
binder.readBean(this.customer);
if (value != null && value.getCity() != null) {
city.setValue(value.getCity());
} else {
city.clear();
}
}
private TextField nameFilter;
private TextField emailFilter;
private TextField cityFilter;
}
```

// DashboardView.java

```
package com.aksenov.monitor.views.dashboard;
import com.aksenov.monitor.data.Sales;
import com.aksenov.monitor.repositories.*;
import com.aksenov.monitor.services.SalesService;
import
com.aksenov.monitor.views.dashboard.ServiceHealth.Status;
import com.vaadin.flow.component.Component;
import com.vaadin.flow.component.board.Board;
import com.vaadin.flow.component.charts.Chart;
import com.vaadin.flow.component.charts.model.*;
import com.vaadin.flow.component.grid.ColumnTextAlign;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.grid.GridVariant;
import com.vaadin.flow.component.html.H2;
import com.vaadin.flow.component.html.Main;
import com.vaadin.flow.component.html.Span;
import com.vaadin.flow.component.icon.Icon;
import com.vaadin.flow.component.icon.VaadinIcon;
import
com.vaadin.flow.component.orderedlayout.FlexComponent;
import
com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.orderedlayout.VerticalLayout;
import com.vaadin.flow.component.select.Select;
import com.vaadin.flow.data.renderer.ComponentRenderer;
import com.vaadin.flow.router.Menu;
import com.vaadin.flow.router.PageTitle;
import com.vaadin.flow.router.Route;
import com.vaadin.flow.spring.annotation.SpringComponent;
import com.vaadin.flow.theme.lumo.LumoUtility.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.vaadin.lineawesome.LineAwesomeIconUrl;
@PageTitle("Dashboard")
@Route("dashboard")
@Menu(order = 6, icon =
LineAwesomeIconUrl.CHART_AREA_SOLID)
@SpringComponent
@Scope("prototype")
public class DashboardView extends Main {
private final CityRepository cityRepository;
private final CustomerRepository customerRepository;
private final ManufacturerRepository manufacturerRepository;
private final MonitorRepository monitorRepository;
private final SalesService salesService;
private final SalesRepository salesRepository;
public DashboardView(CityRepository cityRepository,
CustomerRepository customerRepository, ManufacturerRepository
manufacturerRepository, MonitorRepository monitorRepository,
SalesService salesService, SalesRepository salesRepository) {
this.cityRepository = cityRepository;
this.customerRepository = customerRepository;
this.manufacturerRepository = manufacturerRepository;
this.monitorRepository = monitorRepository;
this.salesService = salesService;
this.salesRepository = salesRepository;
addClassName("dashboard-view");
Board board = new Board();
board.addRow(
createHighlight("Current amount of sales",
String.valueOf(salesRepository.findTotalSalesByMonthAndYear(
LocalDateTime.now().getMonthValue(),
LocalDateTime.now().getYear())), 33.7),
createHighlight("Current sales", String.valueOf((long)
salesRepository.findAllBySaleDateMonthAndYear(LocalDateTim
```

```
e.now().getMonthValue(),
LocalDateTime.now().getYear()).size()), 33.7),
createHighlight("Current customer",
String.valueOf(customerRepository.count()), -112.45));
board.addRow(createViewEvents());
board.addRow(createResponseTimes());
add(board);
}
private List<Series> getSalesSeries(int year) {
List<Series> seriesList = new ArrayList<>();
String[] categories = new String[]{"Jan", "Feb", "Mar", "Apr",
"May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
List<DataSeriesItem> dataSeriesItems = new ArrayList<>();
for (int i = 0; i < 12; i++) {
List<Sales> sales = salesService.findSalesByMonthAndYear(i + 1,
year);
int salesCount = sales != null ? sales.size() : 0;
dataSeriesItems.add(new DataSeriesItem(categories[i],
salesCount));
}
DataSeries dataSeries = new DataSeries();
dataSeries.setData(dataSeriesItems);
seriesList.add(dataSeries);
return seriesList;
}
private Component createHighlight(String title, String value,
Double percentage) {
VaadinIcon icon = VaadinIcon.ARROW_UP;
String prefix = "";
String theme = "badge";
if (percentage == 0) {
prefix = "±";
} else if (percentage > 0) {
prefix = "+";
theme += " success";
} else if (percentage < 0) {
icon = VaadinIcon.ARROW_DOWN;
theme += " error";
}
H2 h2 = new H2(title);
h2.addClassNames(FontWeight.NORMAL, Margin.NONE,
TextColor.SECONDARY, FontSize.XSMALL);
Span span = new Span(value);
span.addClassNames(FontWeight.SEMIBOLD,
FontSize.XXXLARGE);
Icon i = icon.create();
i.addClassNames(BoxSizing.BORDER, Padding.XSMALL);
Span badge = new Span(i, new Span(prefix +
percentage.toString()));
badge.getElement().getThemeList().add(theme);
VerticalLayout layout = new VerticalLayout(h2, span, badge);
layout.addClassName(Padding.LARGE);
layout.setPadding(false);
layout.setSpacing(false);
return layout;
}
private Component createViewEvents() {
// Header
Select<String> year = new Select<>();
year.setItems("2011", "2012", "2013", "2014", "2015", "2016",
"2017", "2018", "2019", "2020", "2021", "2022", "2023", "2024",
"2025");
year.setValue("2024");
year.setWidth("100px");
// Chart
Chart chart = new Chart(ChartType.AREASPLINE);
Configuration conf = chart.getConfiguration();
conf.getChart().setStyledMode(true);
XAxis xAxis = new XAxis();
xAxis.setCategories("Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
conf.addxAxis(xAxis);
conf.getyAxis().setTitle("Values");
PlotOptionsAreaspline plotOptions = new
PlotOptionsAreaspline();
plotOptions.setPointPlacement(PointPlacement.ON);
plotOptions.setMarker(new Marker(false));
conf.addPlotOptions(plotOptions);
//  Load data for sales chart.
List<Series> salesSeries =
getSalesSeries(Integer.parseInt(year.getValue()));
conf.setSeries(salesSeries);
year.addValueChangeListener(event -> {
chart.getConfiguration().setSeries(getSalesSeries(Integer.parseInt(
event.getValue())));
chart.drawChart();
});
// Add it all together
HorizontalLayout header = createHeader("Sales events",
"Sales/month");
header.add(year);
VerticalLayout viewEvents = new VerticalLayout(header, chart);
viewEvents.addClassName(Padding.LARGE);
viewEvents.setPadding(false);
viewEvents.setSpacing(false);
viewEvents.getElement().getThemeList().add("spacing-l");
return viewEvents;
}
private Component createResponseTimes() {
HorizontalLayout header = createHeader("Sales by Model",
"Average across all models");
// Chart
Chart chart = new Chart(ChartType.PIE);
Configuration conf = chart.getConfiguration();
conf.getChart().setStyledMode(true);
chart.setThemeName("gradient");
DataSeries series = new DataSeries();
Map<String, Long> salesData =
salesService.getSalesDataByModel();
for (Map.Entry<String, Long> entry : salesData.entrySet()) {
series.add(new DataSeriesItem(entry.getKey(), entry.getValue()));
}
conf.addSeries(series);
// Add it all together
VerticalLayout serviceHealth = new VerticalLayout(header, chart);
serviceHealth.addClassName(Padding.LARGE);
serviceHealth.setPadding(false);
serviceHealth.setSpacing(false);
serviceHealth.getElement().getThemeList().add("spacing-l");
return serviceHealth;
}
private HorizontalLayout createHeader(String title, String subtitle)
{
H2 h2 = new H2(title);
h2.addClassNames(FontSize.XLARGE, Margin.NONE);
Span span = new Span(subtitle);
span.addClassNames(TextColor.SECONDARY,
FontSize.XSMALL);
VerticalLayout column = new VerticalLayout(h2, span);
column.setPadding(false);
column.setSpacing(false);
HorizontalLayout header = new HorizontalLayout(column);
header.setJustifyContentMode(FlexComponent.JustifyContentMod
e.BETWEEN);
header.setSpacing(false);
header.setWidthFull();
return header;
}
private String getStatusDisplayName(ServiceHealth serviceHealth)
{
Status status = serviceHealth.getStatus();
if (status == Status.OK) {
return "Ok";
} else if (status == Status.FAILING) {
return "Failing";
} else if (status == Status.EXCELLENT) {
return "Excellent";
} else {
return status.toString();
}
}
private String getStatusTheme(ServiceHealth serviceHealth) {
Status status = serviceHealth.getStatus();
String theme = "badge primary small";
if (status == Status.EXCELLENT) {
theme += " success";
} else if (status == Status.FAILING) {
theme += " error";
```

```
}
return theme;
}
}
```

// MainLayout.java

```
package com.aksenov.monitor.views;
import com.vaadin.flow.component.applayout.AppLayout;
import com.vaadin.flow.component.applayout.DrawerToggle;
import com.vaadin.flow.component.html.Footer;
import com.vaadin.flow.component.html.H1;
import com.vaadin.flow.component.html.Header;
import com.vaadin.flow.component.html.Span;
import com.vaadin.flow.component.icon.SvgIcon;
import com.vaadin.flow.component.orderedlayout.Scroller;
import com.vaadin.flow.component.sidenav.SideNav;
import com.vaadin.flow.component.sidenav.SideNavItem;
import com.vaadin.flow.router.Layout;
import com.vaadin.flow.server.auth.AnonymousAllowed;
import com.vaadin.flow.server.menu.MenuConfiguration;
import com.vaadin.flow.server.menu.MenuEntry;
import com.vaadin.flow.theme.lumo.LumoUtility;
import java.util.List;
/**
* The main view is a top-level placeholder for other views.
*/
@Layout
@AnonymousAllowed
public class MainLayout extends AppLayout {
private H1 viewTitle;
public MainLayout() {
setPrimarySection(Section.DRAWER);
addDrawerContent();
addHeaderContent();
}
private void addHeaderContent() {
DrawerToggle toggle = new DrawerToggle();
toggle.setAriaLabel("Menu toggle");
viewTitle = new H1();
viewTitle.addClassNames(LumoUtility.FontSize.LARGE,
LumoUtility.Margin.NONE);
addToNavbar(true, toggle, viewTitle);
}
private void addDrawerContent() {
Span appName = new Span("Monitor");
appName.addClassNames(LumoUtility.FontWeight.SEMIBOLD,
LumoUtility.FontSize.LARGE);
Header header = new Header(appName);
Scroller scroller = new Scroller(createNavigation());
addToDrawer(header, scroller, createFooter());
}
private SideNav createNavigation() {
SideNav nav = new SideNav();
List<MenuEntry> menuEntries =
MenuConfiguration.getMenuEntries();
menuEntries.forEach(entry -> {
if (entry.icon() != null) {
nav.addItem(new SideNavItem(entry.title(), entry.path(), new
SvgIcon(entry.icon())));
} else {
nav.addItem(new SideNavItem(entry.title(), entry.path()));
}
});
return nav;
}
private Footer createFooter() {
Footer layout = new Footer();
return layout;
}
@Override
protected void afterNavigation() {
super.afterNavigation();
viewTitle.setText(getCurrentPageTitle());
}
private String getCurrentPageTitle() {
return
MenuConfiguration.getPageHeader(getContent()).orElse("");
}
}
```

// ManufacturerView.java

```
package com.aksenov.monitor.views.manufacturer;
import com.aksenov.monitor.data.Customer;
import com.aksenov.monitor.data.Manufacturer;
import com.aksenov.monitor.services.ManufacturerService;
import com.vaadin.flow.component.UI;
import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.button.ButtonVariant;
import com.vaadin.flow.component.formlayout.FormLayout;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.grid.GridVariant;
import com.vaadin.flow.component.grid.HeaderRow;
import com.vaadin.flow.component.html.Div;
import com.vaadin.flow.component.notification.Notification;
import
com.vaadin.flow.component.notification.Notification.Position;
import
com.vaadin.flow.component.notification.NotificationVariant;
import
com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.splitlayout.SplitLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.binder.BeanValidationBinder;
import com.vaadin.flow.data.binder.ValidationException;
import com.vaadin.flow.data.provider.ListDataProvider;
import com.vaadin.flow.function.SerializablePredicate;
import com.vaadin.flow.router.BeforeEnterEvent;
import com.vaadin.flow.router.BeforeEnterObserver;
import com.vaadin.flow.router.Menu;
import com.vaadin.flow.router.PageTitle;
import com.vaadin.flow.router.Route;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import
org.springframework.orm.ObjectOptimisticLockingFailureExcepti
on;
import org.vaadin.lineawesome.LineAwesomeIconUrl;
@PageTitle("Manufacturer")
@Route("manufacturer/:manufacturerID?/:action?(edit)")
@Menu(order = 2, icon =
LineAwesomeIconUrl.HAMMER_SOLID)
public class ManufacturerView extends Div implements
BeforeEnterObserver {
private final String MANUFACTURER_ID = "manufacturerID";
private final String
MANUFACTURER_EDIT_ROUTE_TEMPLATE =
"manufacturer/%s/edit";
private final Grid<Manufacturer> grid = new
Grid<>(Manufacturer.class, false);
private TextField name;
private TextField country;
private TextField contactInfo;
private final Button cancel = new Button("Cancel");
private final Button save = new Button("Save");
private final Button delete = new Button("Delete");
private final BeanValidationBinder<Manufacturer> binder;
private Manufacturer manufacturer;
private List<Manufacturer> manufacturers;
private final ManufacturerService manufacturerService;
private ListDataProvider<Manufacturer> dataProvider;
public ManufacturerView(ManufacturerService
manufacturerService) {
this.manufacturerService = manufacturerService;
addClassNames("manufacturer-view");
// Create UI
SplitLayout splitLayout = new SplitLayout();
createGridLayout(splitLayout);
createEditorLayout(splitLayout);
add(splitLayout);
// Configure Grid
grid.addColumn("name").setAutoWidth(true);
grid.addColumn("country").setAutoWidth(true);
grid.addColumn("contactInfo").setAutoWidth(true);
```

```
//grid.setItems(query ->
manufacturerService.list(VaadinSpringDataHelpers.toSpringPageR
equest(query)).stream());
grid.addThemeVariants(GridVariant.LUMO_NO_BORDER);
manufacturers = manufacturerService.list();
dataProvider = new ListDataProvider<>(manufacturers);
grid.setDataProvider(dataProvider);
// when a row is selected or deselected, populate form
grid.asSingleSelect().addValueChangeListener(event -> {
if (event.getValue() != null) {
UI.getCurrent().navigate(String.format(MANUFACTURER_EDIT
_ROUTE_TEMPLATE, event.getValue().getId()));
} else {
clearForm();
UI.getCurrent().navigate(ManufacturerView.class);
}
});
// Configure Form
binder = new BeanValidationBinder<>(Manufacturer.class);
addFilterRow();
binder.bindInstanceFields(this);
cancel.addClickListener(e -> {
clearForm();
refreshGrid();
});
save.addClickListener(e -> {
try {
if (this.manufacturer == null) {
this.manufacturer = new Manufacturer();
}
if (name.isEmpty() || country.isEmpty() || contactInfo.isEmpty()) {
Notification.show("Please fill all required fields");
return;
}
binder.writeBean(this.manufacturer);
manufacturerService.save(this.manufacturer);
clearForm();
refreshGrid();
Notification.show("Data updated");
UI.getCurrent().navigate(ManufacturerView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
delete.addClickListener(e -> {
try {
if (this.manufacturer == null) {
return;
}
binder.writeBean(this.manufacturer);
manufacturerService.delete(this.manufacturer.getId());
clearForm();
refreshGrid();
Notification.show("Data deleted");
UI.getCurrent().navigate(ManufacturerView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
}
@Override
public void beforeEnter(BeforeEnterEvent event) {
Optional<Long> manufacturerId =
event.getRouteParameters().get(MANUFACTURER_ID).map(Lo
ng::parseLong);
if (manufacturerId.isPresent()) {
Optional<Manufacturer> manufacturerFromBackend =
manufacturerService.get(manufacturerId.get());
if (manufacturerFromBackend.isPresent()) {
populateForm(manufacturerFromBackend.get());
} else {
Notification.show(
String.format("The requested manufacturer was not found, ID =
%s", manufacturerId.get()), 3000,
Notification.Position.BOTTOM_START);
// when a row is selected but the data is no longer available,
// refresh grid
refreshGrid();
event.forwardTo(ManufacturerView.class);
}
}
}
private void addFilterRow() {
HeaderRow headerRow = grid.appendHeaderRow();
// Filter for name column
nameFilter = new TextField();
nameFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if (filterValue.isEmpty()) {
dataProvider.clearFilters();
}
else {
dataProvider.setFilter(manufacturer -> manufacturer.getName() !=
null &&
manufacturer.getName().toLowerCase().contains(filterValue.toLo
werCase()));
}
clearOtherFilters(nameFilter);
clearForm();
refreshGrid();
});
nameFilter.setPlaceholder("Filter");
nameFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("name")).setComponen
t(nameFilter);
// Filter for country column
countryFilter = new TextField();
countryFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(manufacturer ->  manufacturer.getCountry()
!= null &&
manufacturer.getCountry().toLowerCase().contains(filterValue.toL
owerCase()));
}
clearOtherFilters(countryFilter);
clearForm();
refreshGrid();
});
countryFilter.setPlaceholder("Filter");
countryFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("country")).setCompon
ent(countryFilter);
// Filter for contactInfo column
contactInfoFilter = new TextField();
contactInfoFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
dataProvider.setFilter(manufacturer ->
manufacturer.getContactInfo() != null &&
manufacturer.getContactInfo().toLowerCase().contains(filterValue.
toLowerCase()));
clearOtherFilters(contactInfoFilter);
clearForm();
refreshGrid();
});
contactInfoFilter.setPlaceholder("Filter");
contactInfoFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("contactInfo")).setCom
ponent(contactInfoFilter);
}
private void clearOtherFilters(TextField currentFilter) {
if (currentFilter != nameFilter) {
```

```
nameFilter.clear();
}
if (currentFilter != countryFilter) {
countryFilter.clear();
}
if (currentFilter != contactInfoFilter) {
contactInfoFilter.clear();
}
}
private void createEditorLayout(SplitLayout splitLayout) {
Div editorLayoutDiv = new Div();
editorLayoutDiv.setClassName("editor-layout");
Div editorDiv = new Div();
editorDiv.setClassName("editor");
editorLayoutDiv.add(editorDiv);
FormLayout formLayout = new FormLayout();
name = new TextField("Name");
country = new TextField("Country");
contactInfo = new TextField("Contact Info");
formLayout.add(name, country, contactInfo);
editorDiv.add(formLayout);
createButtonLayout(editorLayoutDiv);
splitLayout.addToSecondary(editorLayoutDiv);
}
private void createButtonLayout(Div editorLayoutDiv) {
HorizontalLayout buttonLayout = new HorizontalLayout();
buttonLayout.setClassName("button-layout");
cancel.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
save.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
delete.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
buttonLayout.add(save, cancel, delete);
editorLayoutDiv.add(buttonLayout);
}
private void createGridLayout(SplitLayout splitLayout) {
Div wrapper = new Div();
wrapper.setClassName("grid-wrapper");
splitLayout.addToPrimary(wrapper);
wrapper.add(grid);
}
private void refreshGrid() {
SerializablePredicate<Manufacturer> currentFilter =
dataProvider.getFilter();
// 2. Обновить список
manufacturers = manufacturerService.list();
// 3. Создать новый dataProvider и применить фильтр
dataProvider = new ListDataProvider<>(manufacturers);
grid.setDataProvider(dataProvider);
dataProvider.setFilter(currentFilter);
//  grid.select(null);
dataProvider.refreshAll();
}
private void clearForm() {
populateForm(null);
}
private void populateForm(Manufacturer value) {
this.manufacturer = value;
binder.readBean(this.manufacturer);
}
private TextField nameFilter;
private TextField countryFilter;
private TextField contactInfoFilter;
}
```

// MonitorView.java

```
package com.aksenov.monitor.views.monitor;
import com.aksenov.monitor.data.Manufacturer;
import com.aksenov.monitor.data.Monitor;
import com.aksenov.monitor.services.ManufacturerService;
import com.aksenov.monitor.services.MonitorService;
import com.vaadin.flow.component.UI;
import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.button.ButtonVariant;
import com.vaadin.flow.component.combobox.ComboBox;
import com.vaadin.flow.component.formlayout.FormLayout;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.grid.GridVariant;
import com.vaadin.flow.component.grid.HeaderRow;
import com.vaadin.flow.component.html.Div;
import com.vaadin.flow.component.notification.Notification;
import
com.vaadin.flow.component.notification.Notification.Position;
import
com.vaadin.flow.component.notification.NotificationVariant;
import
com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.splitlayout.SplitLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.binder.BeanValidationBinder;
import com.vaadin.flow.data.binder.ValidationException;
import com.vaadin.flow.data.converter.StringToIntegerConverter;
import com.vaadin.flow.data.provider.ListDataProvider;
import com.vaadin.flow.function.SerializablePredicate;
import com.vaadin.flow.router.BeforeEnterEvent;
import com.vaadin.flow.router.BeforeEnterObserver;
import com.vaadin.flow.router.Menu;
import com.vaadin.flow.router.PageTitle;
import com.vaadin.flow.router.Route;
import java.util.List;
import java.util.Optional;
import
org.springframework.orm.ObjectOptimisticLockingFailureExcepti
on;
import org.vaadin.lineawesome.LineAwesomeIconUrl;
@PageTitle("Monitor")
@Route("monitor/:monitorID?/:action?(edit)")
@Menu(order = 3, icon = LineAwesomeIconUrl.TV_SOLID)
public class MonitorView extends Div implements
BeforeEnterObserver {
private final String MONITOR_ID = "monitorID";
private final String MONITOR_EDIT_ROUTE_TEMPLATE =
"monitor/%s/edit";
private final Grid<Monitor> grid = new Grid<>(Monitor.class,
false);
private TextField model;
private ComboBox<Manufacturer> manufacturer;
private TextField screenSize;
private TextField resolution;
private TextField refreshrRate;
private TextField price;
private final Button cancel = new Button("Cancel");
private final Button save = new Button("Save");
private final Button delete = new Button("Delete");
private final BeanValidationBinder<Monitor> binder;
private Monitor monitor;
private List<Monitor> monitors;
private final MonitorService monitorService;
private final ManufacturerService manufacturerService;
private ListDataProvider<Monitor> dataProvider;
public MonitorView(MonitorService monitorService,
ManufacturerService manufacturerService) {
this.monitorService = monitorService;
this.manufacturerService = manufacturerService;
addClassNames("monitor-view");
// Create UI
SplitLayout splitLayout = new SplitLayout();
createGridLayout(splitLayout);
createEditorLayout(splitLayout);
add(splitLayout);
// Configure Grid
grid.addColumn("model").setAutoWidth(true);
grid.addColumn("manufacturer.name").setAutoWidth(true).setHea
der("Manufacturer");
grid.addColumn("screenSize").setAutoWidth(true);
grid.addColumn("resolution").setAutoWidth(true);
grid.addColumn("refreshrRate").setAutoWidth(true);
grid.addColumn("price").setAutoWidth(true);
//grid.setItems(query ->
monitorService.list(VaadinSpringDataHelpers.toSpringPageReque
st(query)).stream());
grid.addThemeVariants(GridVariant.LUMO_NO_BORDER);
monitors = monitorService.list();
dataProvider = new ListDataProvider<>(monitors);
grid.setDataProvider(dataProvider);
// when a row is selected or deselected, populate form
grid.asSingleSelect().addValueChangeListener(event -> {
if (event.getValue() != null) {
```

```
UI.getCurrent().navigate(String.format(MONITOR_EDIT_ROUT
E_TEMPLATE, event.getValue().getId()));
} else {
clearForm();
UI.getCurrent().navigate(MonitorView.class);
}
});
// Configure Form
binder = new BeanValidationBinder<>(Monitor.class);
// Bind fields. This is where you'd define e.g. validation rules
binder.forField(model).bind("model");
binder.forField(manufacturer).bind("manufacturer");
binder.forField(resolution).bind("resolution");
binder.forField(screenSize).withConverter(new
StringToIntegerConverter("Only numbers are allowed"))
.bind("screenSize");
binder.forField(refreshrRate).withConverter(new
StringToIntegerConverter("Only numbers are allowed"))
.bind("refreshrRate");
binder.forField(price).withConverter(new
StringToIntegerConverter("Only numbers are
allowed")).bind("price");
addFilterRow();
cancel.addClickListener(e -> {
clearForm();
refreshGrid();
});
save.addClickListener(e -> {
try {
if (this.monitor == null) {
this.monitor = new Monitor();
}
if (model.isEmpty() || manufacturer.isEmpty() ||
screenSize.isEmpty() || resolution.isEmpty() ||
refreshrRate.isEmpty() || price.isEmpty()) {
Notification.show("Please fill all required fields");
return;
}
if (binder.validate().isOk()) {
binder.writeBean(this.monitor);
monitorService.save(this.monitor);
clearForm();
refreshGrid();
Notification.show("Data updated");
UI.getCurrent().navigate(MonitorView.class);
}else{
Notification.show("Please fill all required fields");
}
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
delete.addClickListener(e -> {
try {
if (this.monitor == null) {
return;
}
binder.writeBean(this.monitor);
monitorService.delete(this.monitor.getId());
clearForm();
refreshGrid();
Notification.show("Data deleted");
UI.getCurrent().navigate(MonitorView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
}
@Override
public void beforeEnter(BeforeEnterEvent event) {
Optional<Long> monitorId =
event.getRouteParameters().get(MONITOR_ID).map(Long::parse
Long);
if (monitorId.isPresent()) {
Optional<Monitor> monitorFromBackend =
monitorService.get(monitorId.get());
if (monitorFromBackend.isPresent()) {
populateForm(monitorFromBackend.get());
} else {
Notification.show(String.format("The requested monitor was not
found, ID = %s", monitorId.get()), 3000,
Notification.Position.BOTTOM_START);
// when a row is selected but the data is no longer available,
// refresh grid
refreshGrid();
event.forwardTo(MonitorView.class);
}
}
}
private void addFilterRow() {
HeaderRow headerRow = grid.appendHeaderRow();
// Filter for model column
modelFilter = new TextField();
modelFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if (filterValue.isEmpty()) {
dataProvider.clearFilters();
}
else {
dataProvider.setFilter(monitor ->
monitor.getModel().toLowerCase().contains(filterValue.toLowerC
ase()));
}
clearOtherFilters(modelFilter);
clearForm();
refreshGrid();
});
modelFilter.setPlaceholder("Filter");
modelFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("model")).setCompone
nt(modelFilter);
// Filter for manufacturerId column
manufacturerIdFilter = new TextField();
manufacturerIdFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(monitor ->
String.valueOf(monitor.getManufacturer()).contains(filterValue));
}
clearOtherFilters(manufacturerIdFilter);
clearForm();
refreshGrid();
});
manufacturerIdFilter.setPlaceholder("Filter");
manufacturerIdFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("manufacturer.name")).
setComponent(manufacturerIdFilter);
// Filter for screenSize column
screenSizeFilter = new TextField();
screenSizeFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(monitor ->
String.valueOf(monitor.getScreenSize()).contains(filterValue));
}
clearOtherFilters(screenSizeFilter);
clearForm();
refreshGrid();
});
screenSizeFilter.setPlaceholder("Filter");
```

```
screenSizeFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("screenSize")).setCom
ponent(screenSizeFilter);
// Filter for resolution column
resolutionFilter = new TextField();
resolutionFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(monitor ->
monitor.getResolution().toLowerCase().contains(filterValue.toLow
erCase()));
}
clearOtherFilters(resolutionFilter);
clearForm();
refreshGrid();
});
resolutionFilter.setPlaceholder("Filter");
resolutionFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("resolution")).setComp
onent(resolutionFilter);
// Filter for refreshrRate column
refreshrRateFilter = new TextField();
refreshrRateFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(monitor ->
String.valueOf(monitor.getRefreshrRate()).contains(filterValue));
}
clearOtherFilters(refreshrRateFilter);
clearForm();
refreshGrid();
});
refreshrRateFilter.setPlaceholder("Filter");
refreshrRateFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("refreshrRate")).setCo
mponent(refreshrRateFilter);
// Filter for price column
priceFilter = new TextField();
priceFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(monitor ->
String.valueOf(monitor.getPrice()).contains(filterValue));
}
clearOtherFilters(priceFilter);
clearForm();
refreshGrid();
});
priceFilter.setPlaceholder("Filter");
priceFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("price")).setComponent
(priceFilter);
}
private void clearOtherFilters(TextField currentFilter) {
if (currentFilter != modelFilter) {
modelFilter.clear();
}
if (currentFilter != manufacturerIdFilter) {
manufacturerIdFilter.clear();
}
if (currentFilter != screenSizeFilter) {
screenSizeFilter.clear();
}
if (currentFilter != resolutionFilter) {
resolutionFilter.clear();
}
if (currentFilter != refreshrRateFilter) {
refreshrRateFilter.clear();
}
if (currentFilter != priceFilter) {
priceFilter.clear();
}
}
private void createEditorLayout(SplitLayout splitLayout) {
Div editorLayoutDiv = new Div();
editorLayoutDiv.setClassName("editor-layout");
Div editorDiv = new Div();
editorDiv.setClassName("editor");
editorLayoutDiv.add(editorDiv);
FormLayout formLayout = new FormLayout();
model = new TextField("Model");
model.setRequired(true);
manufacturer = new ComboBox<Manufacturer>("Manufacturer");
manufacturer.setItemLabelGenerator(currManufacturer ->
String.valueOf(currManufacturer.getId()));
manufacturer.setAllowCustomValue(false);
manufacturer.setRequired(true);
manufacturer.addFocusListener(event -> {
List<Manufacturer> manufacturerList =
manufacturerService.list();
manufacturer.setItems(manufacturerList);
});
manufacturer.setItemLabelGenerator(manufacturer ->
manufacturer != null ? manufacturer.getName() : "");
screenSize = new TextField("Screen Size");
screenSize.setRequired(true);
resolution = new TextField("Resolution");
resolution.setRequired(true);
refreshrRate = new TextField("Refreshr Rate");
refreshrRate.setRequired(true);
price = new TextField("Price");
price.setRequired(true);
formLayout.add(model, manufacturer, screenSize, resolution,
refreshrRate, price);
editorDiv.add(formLayout);
createButtonLayout(editorLayoutDiv);
splitLayout.addToSecondary(editorLayoutDiv);
}
private void createButtonLayout(Div editorLayoutDiv) {
HorizontalLayout buttonLayout = new HorizontalLayout();
buttonLayout.setClassName("button-layout");
cancel.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
save.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
delete.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
buttonLayout.add(save, cancel, delete);
editorLayoutDiv.add(buttonLayout);
}
private void createGridLayout(SplitLayout splitLayout) {
Div wrapper = new Div();
wrapper.setClassName("grid-wrapper");
splitLayout.addToPrimary(wrapper);
wrapper.add(grid);
}
private void refreshGrid() {
SerializablePredicate<Monitor> currentFilter =
dataProvider.getFilter();
// 2. Обновить список
monitors = monitorService.list();
// 3. Создать новый dataProvider и применить фильтр
dataProvider = new ListDataProvider<>(monitors);
grid.setDataProvider(dataProvider);
dataProvider.setFilter(currentFilter);
//  grid.select(null);
dataProvider.refreshAll();
}
private void clearForm() {
populateForm(null);
}
private void populateForm(Monitor value) {
this.monitor = value;
// Загрузить список городов
List<Manufacturer> manufacturerList =
manufacturerService.list();
// Установить список городов в ComboBox
manufacturer.setItems(manufacturerList);
binder.readBean(this.monitor);
}
private TextField modelFilter;
private TextField manufacturerIdFilter;
private TextField screenSizeFilter;
private TextField resolutionFilter;
private TextField refreshrRateFilter;
```

```
private TextField priceFilter;
}
```

// SalesView.java

```
package com.aksenov.monitor.views.sales;
import com.aksenov.monitor.data.Customer;
import com.aksenov.monitor.data.Monitor;
import com.aksenov.monitor.data.Sales;
import com.aksenov.monitor.services.CustomerService;
import com.aksenov.monitor.services.MonitorService;
import com.aksenov.monitor.services.SalesService;
import com.vaadin.flow.component.UI;
import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.button.ButtonVariant;
import com.vaadin.flow.component.combobox.ComboBox;
import
com.vaadin.flow.component.datetimepicker.DateTimePicker;
import com.vaadin.flow.component.formlayout.FormLayout;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.grid.GridVariant;
import com.vaadin.flow.component.grid.HeaderRow;
import com.vaadin.flow.component.html.Div;
import com.vaadin.flow.component.notification.Notification;
import
com.vaadin.flow.component.notification.Notification.Position;
import
com.vaadin.flow.component.notification.NotificationVariant;
import
com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.splitlayout.SplitLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.binder.BeanValidationBinder;
import com.vaadin.flow.data.binder.ValidationException;
import com.vaadin.flow.data.converter.StringToIntegerConverter;
import com.vaadin.flow.data.provider.ListDataProvider;
import com.vaadin.flow.function.SerializablePredicate;
import com.vaadin.flow.router.BeforeEnterEvent;
import com.vaadin.flow.router.BeforeEnterObserver;
import com.vaadin.flow.router.Menu;
import com.vaadin.flow.router.PageTitle;
import com.vaadin.flow.router.Route;
import java.time.Duration;
import java.util.List;
import java.util.Optional;
import
org.springframework.orm.ObjectOptimisticLockingFailureExcepti
on;
import org.vaadin.lineawesome.LineAwesomeIconUrl;
@PageTitle("Sales")
@Route("sales/:salesID?/:action?(edit)")
@Menu(order = 4, icon =
LineAwesomeIconUrl.DOLLAR_SIGN_SOLID)
public class SalesView extends Div implements
BeforeEnterObserver {
private final String SALES_ID = "salesID";
private final String SALES_EDIT_ROUTE_TEMPLATE =
"sales/%s/edit";
private final Grid<Sales> grid = new Grid<>(Sales.class, false);
private ComboBox<Monitor> monitor;
private ComboBox<Customer> customer;
private DateTimePicker saleDate;
private TextField quantity;
private TextField totalPrice;
private final Button cancel = new Button("Cancel");
private final Button save = new Button("Save");
private final Button delete = new Button("Delete");
private final BeanValidationBinder<Sales> binder;
private Sales sales;
private List<Sales> salesList;
private ListDataProvider<Sales> dataProvider;
private final SalesService salesService;
private final MonitorService monitorService;
private final CustomerService customerService;
public SalesView(SalesService salesService, MonitorService
monitorService, CustomerService customerService) {
this.salesService = salesService;
this.monitorService = monitorService;
this.customerService = customerService;
addClassNames("sales-view");
// Create UI
SplitLayout splitLayout = new SplitLayout();
createGridLayout(splitLayout);
createEditorLayout(splitLayout);
add(splitLayout);
// Configure Grid
grid.addColumn("monitor.model").setAutoWidth(true).setHeader("
Monitor");
grid.addColumn("customer.name").setAutoWidth(true).setHeader(
"Customer");
grid.addColumn("saleDate").setAutoWidth(true);
grid.addColumn("quantity").setAutoWidth(true);
grid.addColumn("totalPrice").setAutoWidth(true);
//grid.setItems(query ->
salesService.list(VaadinSpringDataHelpers.toSpringPageRequest(q
uery)).stream());
grid.addThemeVariants(GridVariant.LUMO_NO_BORDER);
salesList = salesService.list();
dataProvider = new ListDataProvider<>(salesList);
grid.setDataProvider(dataProvider);
// when a row is selected or deselected, populate form
grid.asSingleSelect().addValueChangeListener(event -> {
if (event.getValue() != null) {
UI.getCurrent().navigate(String.format(SALES_EDIT_ROUTE_T
EMPLATE, event.getValue().getId()));
} else {
clearForm();
UI.getCurrent().navigate(SalesView.class);
}
});
// Configure Form
binder = new BeanValidationBinder<>(Sales.class);
// Bind fields. This is where you'd define e.g. validation rules
binder.forField(monitor).bind("monitor");
binder.forField(customer).bind("customer");
binder.forField(saleDate).bind("saleDate");
binder.forField(quantity).withConverter(new
StringToIntegerConverter("Only numbers are allowed"))
.bind("quantity");
binder.forField(totalPrice).withConverter(new
StringToIntegerConverter("Only numbers are allowed"))
.bind("totalPrice");
addFilterRow();
cancel.addClickListener(e -> {
clearForm();
refreshGrid();
});
save.addClickListener(e -> {
try {
if (this.sales == null) {
this.sales = new Sales();
}
if (monitor.isEmpty() || customer.isEmpty() || saleDate.isEmpty()
|| quantity.isEmpty() ){
Notification.show("Please fill all required fields");
return;
}
// Check if the binder has errors before writing
if (binder.validate().isOk()) {
binder.writeBean(this.sales);
salesService.save(this.sales);
clearForm();
refreshGrid();
Notification.show("Data updated");
UI.getCurrent().navigate(SalesView.class);
}else{
Notification.show("Please fill all required fields");
}
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
```

```
});
delete.addClickListener(e -> {
try {
if (this.sales == null) {
return;
}
binder.writeBean(this.sales);
salesService.delete(this.sales.getId());
clearForm();
refreshGrid();
Notification.show("Data deleted");
UI.getCurrent().navigate(SalesView.class);
} catch (ObjectOptimisticLockingFailureException exception) {
Notification n = Notification.show(
"Error updating the data. Somebody else has updated the record
while you were making changes.");
n.setPosition(Position.MIDDLE);
n.addThemeVariants(NotificationVariant.LUMO_ERROR);
} catch (ValidationException validationException) {
Notification.show("Failed to update the data. Check again that all
values are valid");
}
});
}
@Override
public void beforeEnter(BeforeEnterEvent event) {
Optional<Long> salesId =
event.getRouteParameters().get(SALES_ID).map(Long::parseLon
g);
if (salesId.isPresent()) {
Optional<Sales> salesFromBackend =
salesService.get(salesId.get());
if (salesFromBackend.isPresent()) {
populateForm(salesFromBackend.get());
} else {
Notification.show(String.format("The requested sales was not
found, ID = %s", salesId.get()), 3000,
Notification.Position.BOTTOM_START);
// when a row is selected but the data is no longer available,
// refresh grid
refreshGrid();
event.forwardTo(SalesView.class);
}
}
}
private void addFilterRow() {
HeaderRow headerRow = grid.appendHeaderRow();
// Filter for monitor column
monitorIdFilter = new TextField();
monitorIdFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(sales ->
String.valueOf(sales.getMonitor()).contains(filterValue));
}
clearOtherFilters(monitorIdFilter);
clearForm();
refreshGrid();
});
monitorIdFilter.setPlaceholder("Filter");
monitorIdFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("monitor.model")).setC
omponent(monitorIdFilter);
// Filter for customerId column
customerIdFilter = new TextField();
customerIdFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(sales ->
String.valueOf(sales.getCustomer()).contains(filterValue));
}
clearOtherFilters(customerIdFilter);
clearForm();
refreshGrid();
});
customerIdFilter.setPlaceholder("Filter");
customerIdFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("customer.name")).set
Component(customerIdFilter);
// Filter for saleDate column
saleDateFilter = new TextField();
saleDateFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(sales ->
String.valueOf(sales.getSaleDate()).contains(filterValue));
}
clearOtherFilters(saleDateFilter);
clearForm();
refreshGrid();
});
saleDateFilter.setPlaceholder("Filter");
saleDateFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("saleDate")).setCompo
nent(saleDateFilter);
// Filter for quantity column
quantityFilter = new TextField();
quantityFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(sales ->
String.valueOf(sales.getQuantity()).contains(filterValue));
}
clearOtherFilters(quantityFilter);
clearForm();
refreshGrid();
});
quantityFilter.setPlaceholder("Filter");
quantityFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("quantity")).setCompo
nent(quantityFilter);
// Filter for totalPrice column
totalPriceFilter = new TextField();
totalPriceFilter.addValueChangeListener(event -> {
String filterValue = ((TextField) event.getSource()).getValue();
if(filterValue.isEmpty()) {
dataProvider.clearFilters();
} else {
dataProvider.setFilter(sales ->
String.valueOf(sales.getTotalPrice()).contains(filterValue));
}
clearOtherFilters(totalPriceFilter);
clearForm();
refreshGrid();
});
totalPriceFilter.setPlaceholder("Filter");
totalPriceFilter.setWidthFull();
headerRow.getCell(grid.getColumnByKey("totalPrice")).setComp
onent(totalPriceFilter);
}
private void clearOtherFilters(TextField currentFilter) {
if (currentFilter != monitorIdFilter) {
monitorIdFilter.clear();
}
if (currentFilter != customerIdFilter) {
customerIdFilter.clear();
}
if (currentFilter != saleDateFilter) {
saleDateFilter.clear();
}
if (currentFilter != quantityFilter) {
quantityFilter.clear();
}
if (currentFilter != totalPriceFilter) {
totalPriceFilter.clear();
}
}
private void createEditorLayout(SplitLayout splitLayout) {
Div editorLayoutDiv = new Div();
editorLayoutDiv.setClassName("editor-layout");
```

```
Div editorDiv = new Div();
editorDiv.setClassName("editor");
editorLayoutDiv.add(editorDiv);
FormLayout formLayout = new FormLayout();
monitor = new ComboBox<Monitor>("Monitor");
monitor.setItemLabelGenerator(currMonitor ->
String.valueOf(currMonitor.getId()));
monitor.setAllowCustomValue(false);
monitor.setRequired(true);  // Make it required
monitor.addValueChangeListener(event -> {
if(event.getValue() != null) {
calculateTotalPrice();
} else {
totalPrice.clear();
}
});
monitor.addFocusListener(event -> {
List<Monitor> manufacturerList = monitorService.list();
monitor.setItems(manufacturerList);
});
monitor.setItemLabelGenerator(monitor ->
monitor != null ? monitor.getModel() : "");
customer = new ComboBox<Customer>("Customer");
customer.setItemLabelGenerator(currMonitor ->
String.valueOf(currMonitor.getId()));
customer.setAllowCustomValue(false);
customer.setRequired(true);  // Make it required
customer.addFocusListener(event -> {
List<Customer> customerList = customerService.list();
customer.setItems(customerList);
});
customer.setItemLabelGenerator(customer ->
customer != null ? customer.getName() : "");
saleDate = new DateTimePicker("Sale Date");
saleDate.setStep(Duration.ofSeconds(1));
saleDate.setRequiredIndicatorVisible(true); // Make it required
quantity = new TextField("Quantity");
quantity.setRequired(true); // Make it required
quantity.addValueChangeListener(event -> calculateTotalPrice());
totalPrice = new TextField("Total Price");
totalPrice.setReadOnly(true);
formLayout.add(monitor, customer, saleDate, quantity, totalPrice);
editorDiv.add(formLayout);
createButtonLayout(editorLayoutDiv);
splitLayout.addToSecondary(editorLayoutDiv);
}
private void calculateTotalPrice() {
if(monitor.getValue() != null && !quantity.isEmpty()) {
Integer quantityValue = Integer.parseInt(quantity.getValue());
Integer monitorPrice = monitor.getValue().getPrice();
Integer totalPriceValue = quantityValue * monitorPrice;
totalPrice.setValue(String.valueOf(totalPriceValue));
} else {
totalPrice.clear();
}
}
private void createButtonLayout(Div editorLayoutDiv) {
HorizontalLayout buttonLayout = new HorizontalLayout();
buttonLayout.setClassName("button-layout");
cancel.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
save.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
delete.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
buttonLayout.add(save, cancel, delete);
editorLayoutDiv.add(buttonLayout);
}
private void createGridLayout(SplitLayout splitLayout) {
Div wrapper = new Div();
wrapper.setClassName("grid-wrapper");
splitLayout.addToPrimary(wrapper);
wrapper.add(grid);
}
private void refreshGrid() {
SerializablePredicate<Sales> currentFilter =
dataProvider.getFilter();
salesList = salesService.list();
dataProvider = new ListDataProvider<>(salesList);
grid.setDataProvider(dataProvider);
dataProvider.setFilter(currentFilter);
dataProvider.refreshAll();
}
private void clearForm() {
populateForm(null);
}
private void populateForm(Sales value) {
this.sales = value;
// Загрузить список городов
List<Monitor> monitorList = monitorService.list();
// Установить список городов в ComboBox
monitor.setItems(monitorList);
List<Customer> customerList = customerService.list();
// Установить список городов в ComboBox
customer.setItems(customerList);
// Установить значения клиента
binder.readBean(this.sales);
if (value != null && value.getMonitor() != null) {
monitor.setValue(value.getMonitor());
} else {
monitor.clear();
}
if (value != null && value.getCustomer() != null) {
customer.setValue(value.getCustomer());
} else {
customer.clear();
}
if(value != null && value.getSaleDate() != null) {
saleDate.setValue(value.getSaleDate()); // Set the date from the
Sales entity
} else {
saleDate.clear();
}
calculateTotalPrice();
}
private TextField monitorIdFilter;
private TextField customerIdFilter;
private TextField saleDateFilter;
private TextField quantityFilter;
private TextField totalPriceFilter;
}
```

Приложение 3

Руководство пользователя

1. Общие сведения о программе

Исполнительный файл: monitor\monitor.jar

Файл базы данных: monitor \monitorDB

2. Описание установки

Данная программа не требует установки.

3. Описание запуска

Для запуска программы необходимо дважды нажать левую кнопку мыши по файлу monitor.jar (рис. П3. 1)

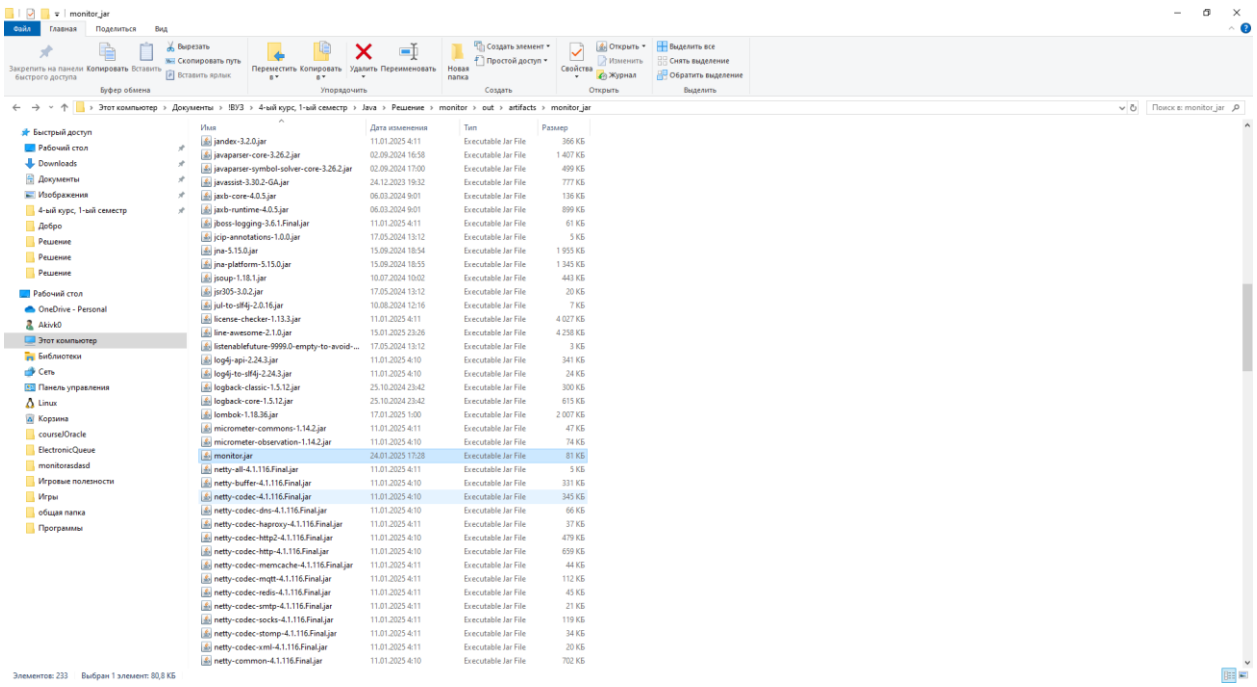

Рис. П3. 1. Запуск программы

4. Инструкция по работе

На странице сущности () имеются следующие объекты:

- таблица записей;
- поля ввода данных записи;
- кнопка добавления записи «Save»;

- кнопка удаления выбранной записи «Delete»;
- кнопка отмены выбора или ввода данных «Cancel».

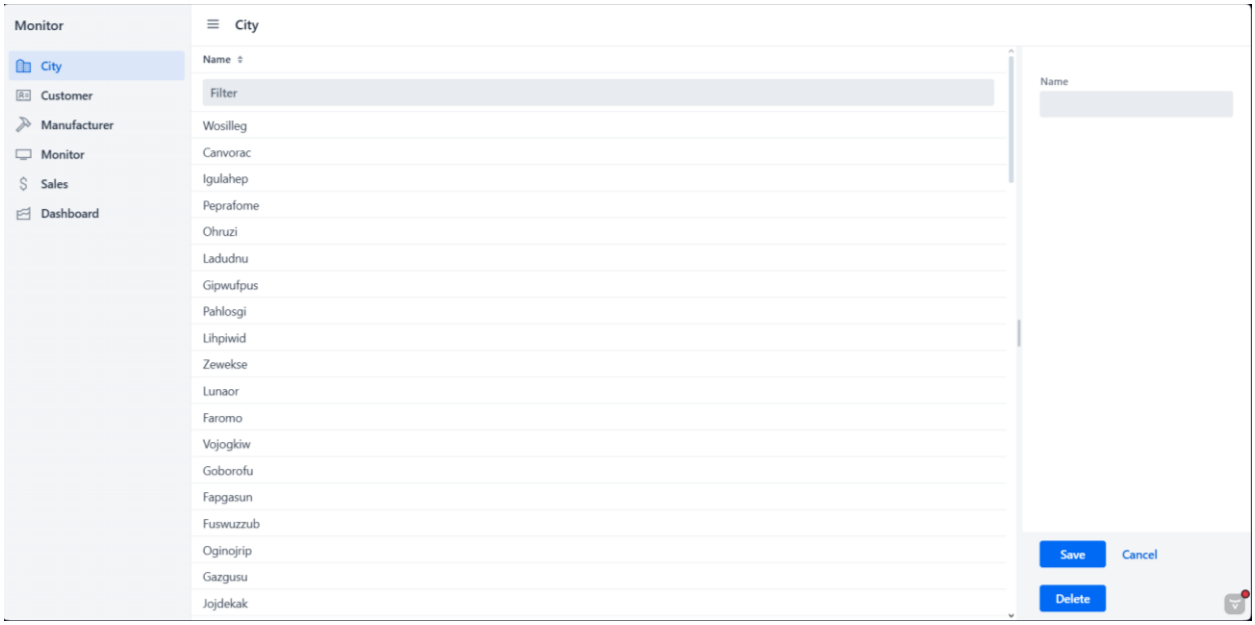

Рис. П3. 2. Страница сущности

Для добавления сущности необходимо внести данные в поле ввода, нажать кнопку «Save». После успешного добавления записи в левом углу страницы появится уведомление (рис. П3. 5). При вводе некорректных данных или при вводе не всех данных появится предупреждение (рис. П3. 6).

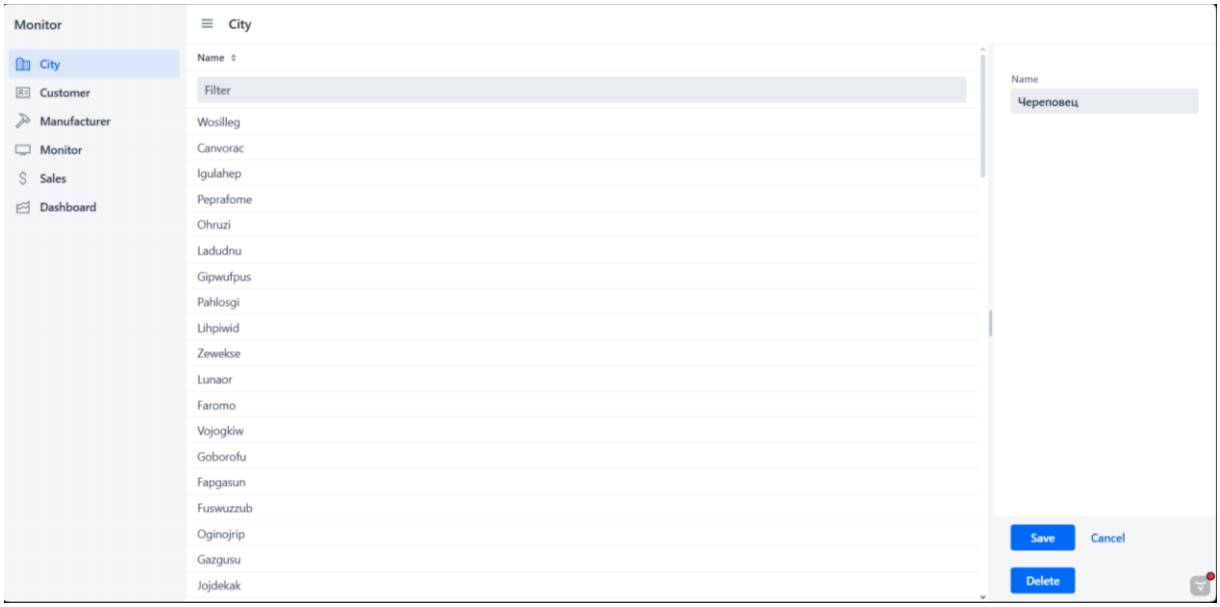

Рис. П3. 3. Ввод данных записи

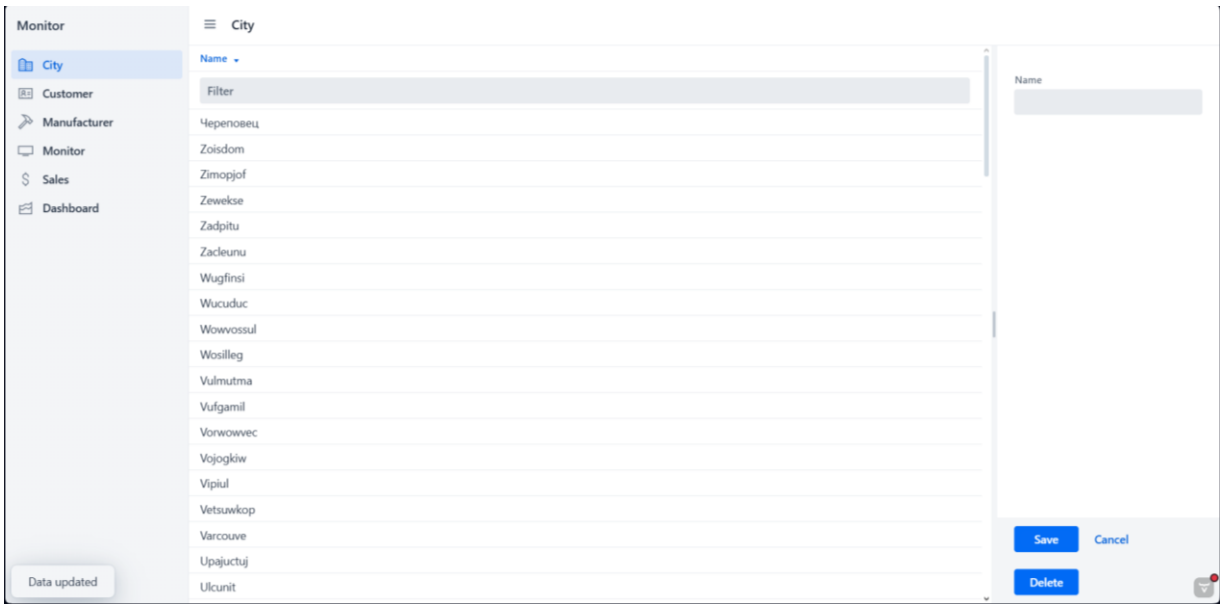

Рис. П3. 4. Добавленная запись

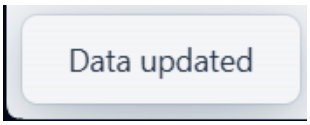

Рис. П3. 5. Уведомление успешного изменения данных

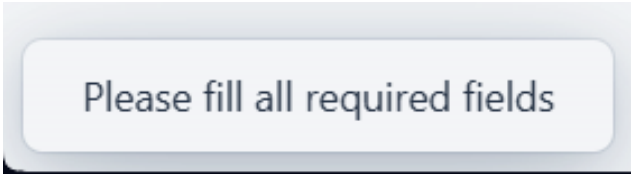

Рис. П3. 6. Уведомление неполного ввода данных

Для изменения записи необходимо выбрать запись в таблице, ввести новые данные, нажать копку «Save» (рис. П3. 7 - 9).

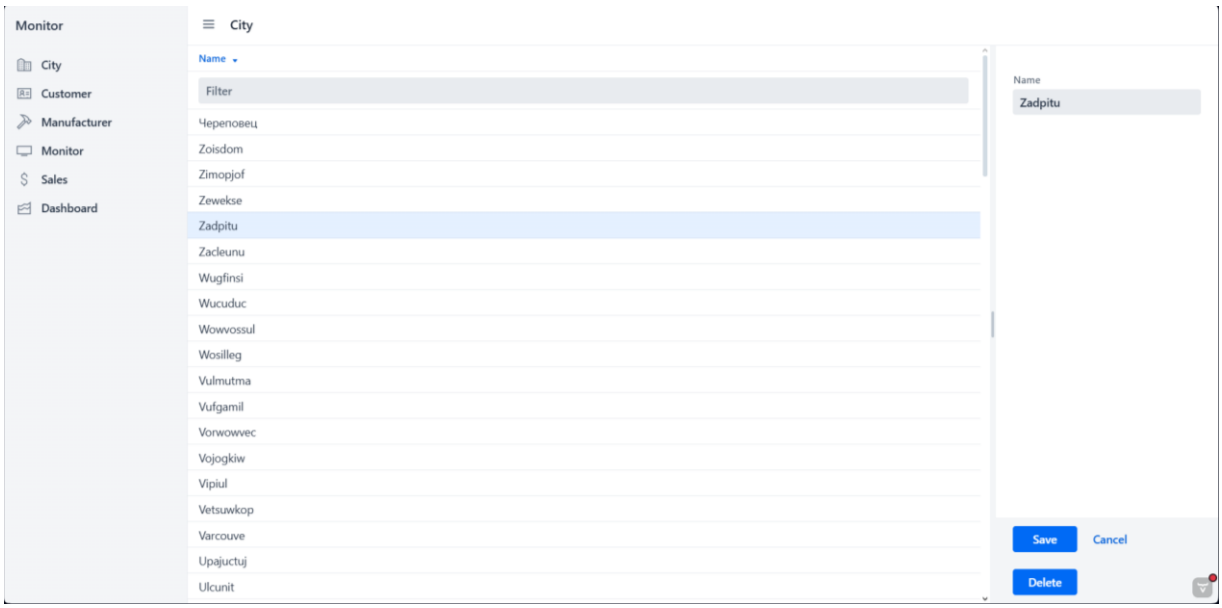

Рис. П3. 7. Выбранная запись

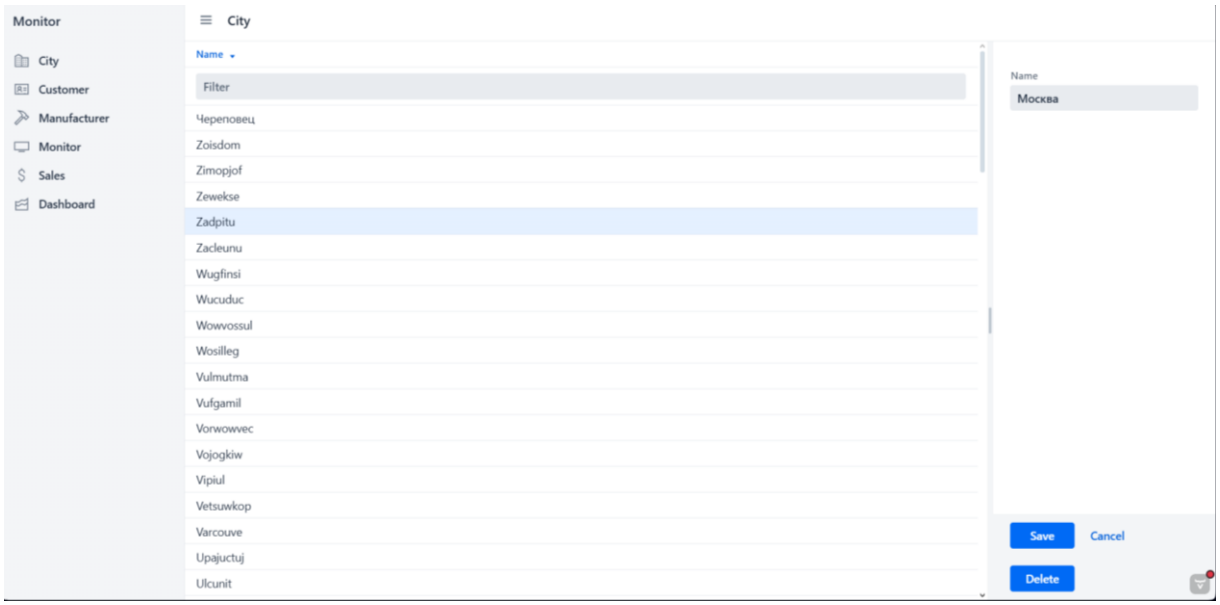

Рис. П3. 8. Ввод новых данных записи

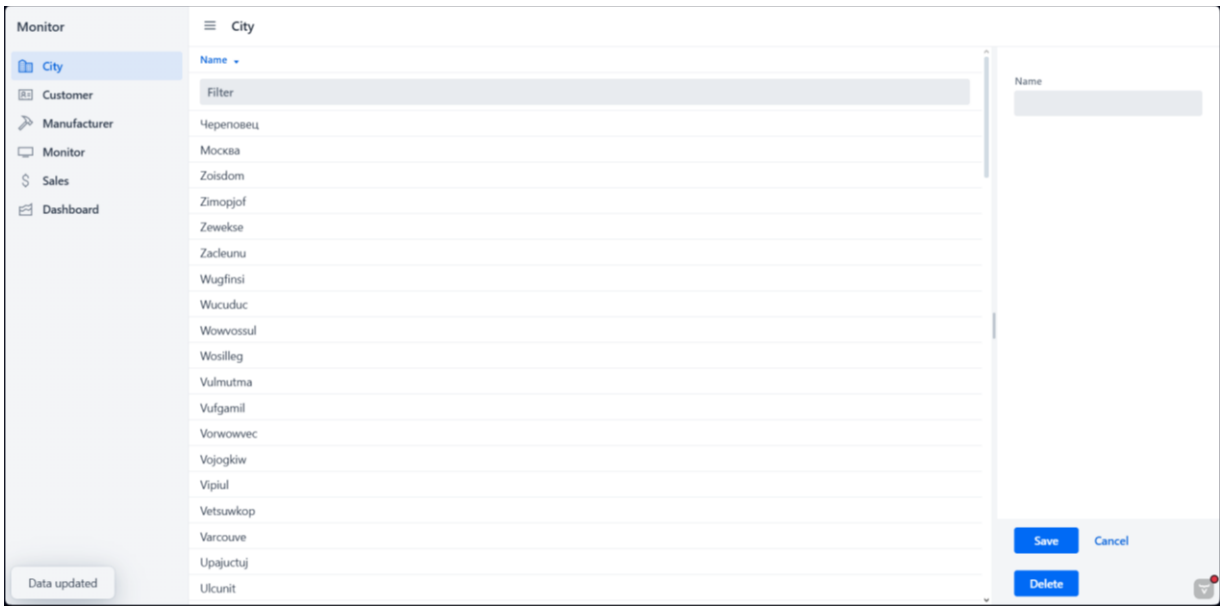

Рис. П3. 9. Измененная запись

Для удаления записи необходимо выбрать запись в таблице и нажать кнопку «Delete» (рис. П3. 10 - 11).

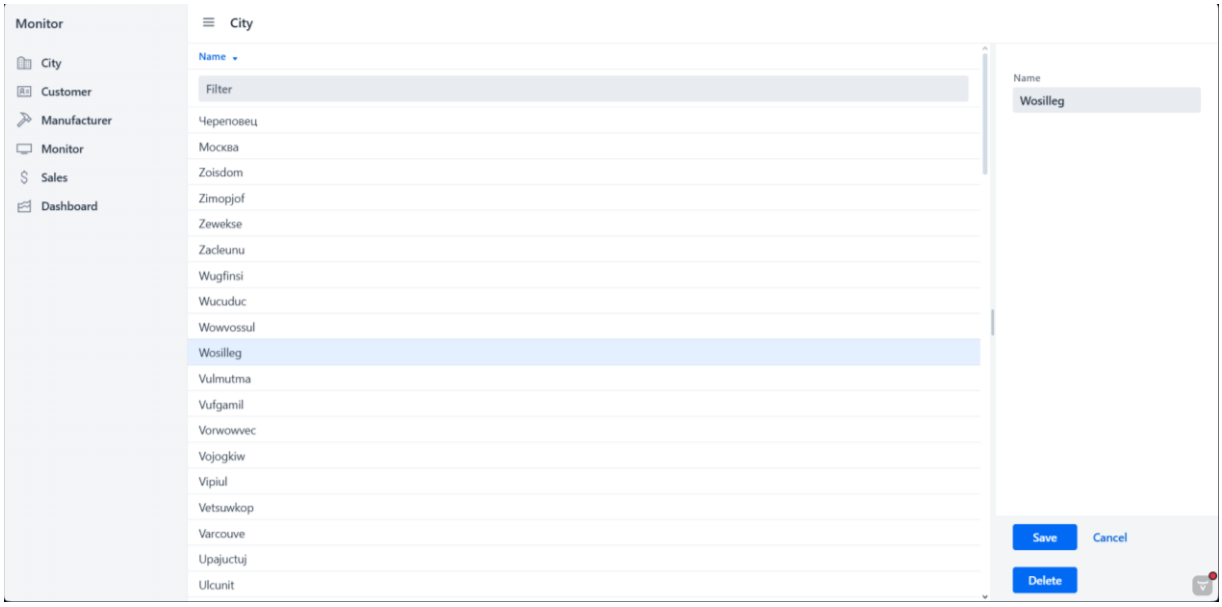

Рис. П3. 10. Выбора записи для удаления

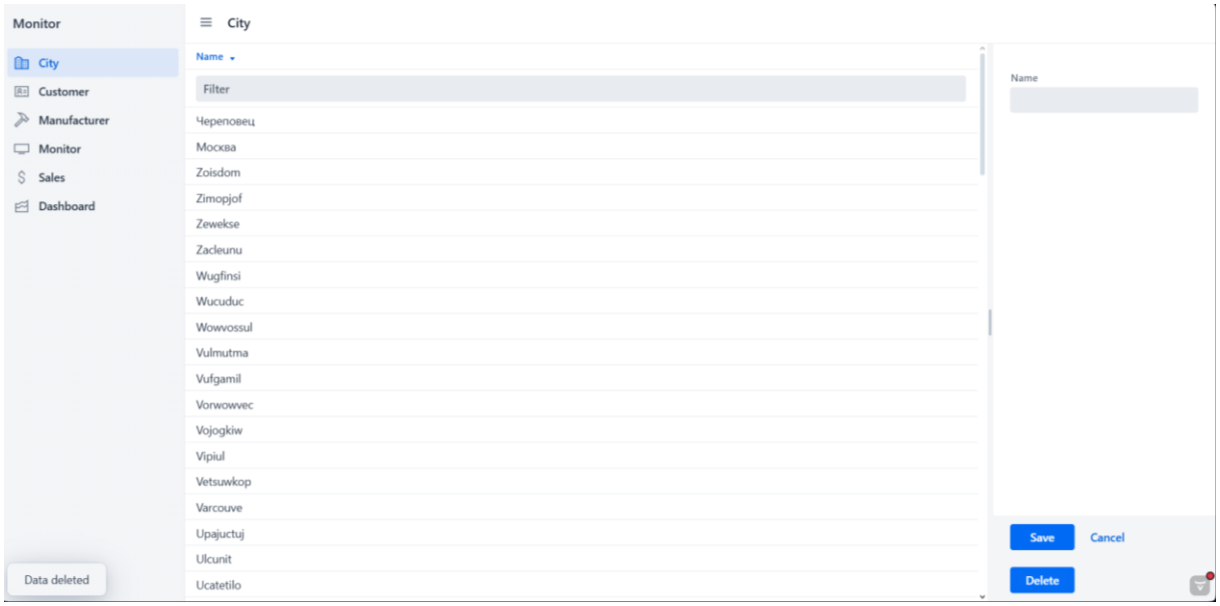

Рис. П3. 11. Таблица после удаления записи

Для перехода на другую страницу нужно нажать на нужную вкладку в левом меню (рис. П3. 12).

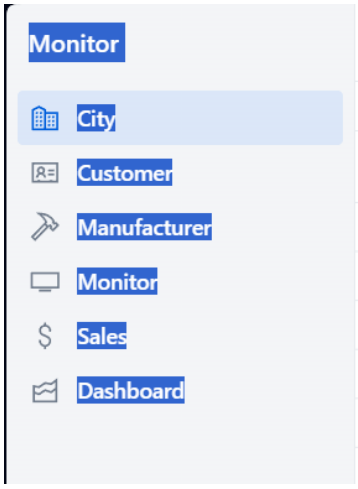

Рис. П3. 12. Меню перехода

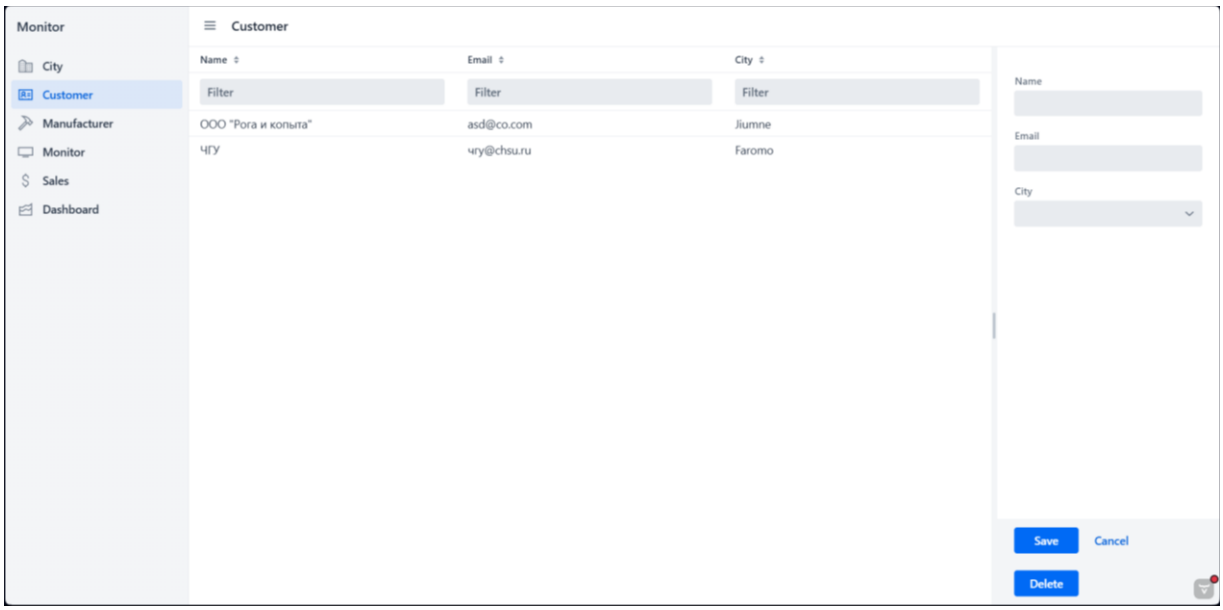

Рис. П3. 13. Страница «Покупатели»

5. Закрытие программы

Для закрытия программы нужно закрыть вкладку браузера сайта.