УДК [004.774+330.33]:629.08

2. ВЕБ-СЕРВИС ДЛЯ ПОДДЕРЖКИ ОСНОВНЫХ БИЗНЕС-ПРОЦЕССОВ ФУНКЦИОНИРОВАНИЯ СТАНЦИИ ТЕХНИЧЕСКОГО ОБСЛУЖИВАНИЯ АВТОМОБИЛЕЙ

Котягова А.Д., студентка гр. 072304, Литвинова В.А., ассистент каф. ЭИ

Белорусский государственный университет информатики и радиоэлектроники г. Минск, Республика Беларусь

Сторожев Д.А. – ст. преподаватель каф. ЭИ

Аннотация. Актуальность разработки данного проекта обусловлена тем, что в современном обществе наблюдается растущий спрос на сервисы, которые помогают оптимизировать процессы организации работы и управления задачами организаций. Приложение, способное эффективно поддерживать рабочие процессы, может успешно занять свою экономическую нишу в данной сфере, обеспечив себе развитие и прибыль.

Ключевые слова. Станция технического обслуживания, клиент, сервер, модели UML, схемы алгоритмов работы, TypeScript, Java, Spring, React, MySQL, база данных, бизнес-процессы, веб-сервис.

Исследование и понимание основных аспектов функционирования станции технического обслуживания являются ключевыми в современном бизнесе. Эффективное управление бизнес-процессами и автоматизация операций становятся необходимостью для оптимизации работы и повышения производительности [1]. В данном контексте разработка программного средства, направленного на поддержку основных бизнес-процессов и автоматизацию рабочего места специалиста станции технического обслуживания, выступает важной задачей [2]. Цель данного проекта заключается в повышении эффективности работы станции технического обслуживания через создание программного инструмента, способного оптимизировать основные процессы функционирования.

Объектом исследования являются основные бизнес-процессы станции технического обслуживания автомобилей. На основе вышеизложенного была поставлена следующая гипотеза: создание и внедрение ПО для поддержки основных бизнес-процессов функционирования станции технического обслуживания автомобилей позволит ускорить, упростить, стабилизировать и оптимизировать работу станции в целом.

Пользовательский интерфейс системы реализуется на языке TypeScript с помощью библиотеки React (см. рисунок 1).

Основными требованиями к интерфейсу являются:

- за каждым действием пользователя внутри приложения должна следовать обратная связь: осуществление навигации, начало загрузки с соответствующим отображением (индикатором загрузки), подтверждение выполнения действия (отображение диалогового окна с подтверждением успешного выполнения действия или сообщением об ошибке и ее описанием) и так далее;
- предусмотрение защиты от ввода данных в некорректном формате с установкой соответствующих ограничений (списки возможных значений, маски ввода, блокировка кнопок и так далее);
- содержимое экранов должно быть адаптировано для различных устройств;
- экраны должны сохранять и восстанавливать свое состояние.

Программное средство имеет интерфейс веб-приложения, т.е. приложение располагается во вкладке в браузере. А в роли посредников между пользовательскими командами и приложением выступают меню, отдельные кнопки и поля ввода различных форматов.

Расположение всех элементов пользовательского интерфейса спроектировано таким образом, чтобы поведение программы было максимально понятным и предсказуемым. Таким образом, основное содержание страниц помещается в центр, навигационные кнопки располагаются в углах. Кнопки, ведущие к наиболее предпочтительным действиям пользователя, имеют яркий фон. Текст, в зависимости от его значимости, отображается соответствующим цветом и размером (ярче или бледнее и больше или меньше соответственно).

Приложение имеет общий стиль: окна и их элементы выполнены в разных оттенках серого цвета и акцентного кукурузного, текст использует два основных шрифта.

Рисунок 1 – Главная страница приложения

Была реализована клиент-серверная архитектура приложения: взаимодействие двух самостоятельных процессов – клиента и сервера, которые могут выполняться как на одном, так и на разных компьютерах, обмениваясь данными по сети. Клиент-серверная архитектура является широко используемым подходом в разработке программного обеспечения, разделяющим функционал между клиентской и серверной частями. Она представляет из себя модель взаимодействия, где клиенты и серверы обмениваются данными через сеть. Этот подход обеспечивает эффективное разделение ответственности: клиенты (например, веб-браузеры или приложения) обращаются к серверу (который может быть удаленным или работать локально) для получения данных или выполнения операций.

В архитектуре «клиент-сервер» клиенты не имеют прямого доступа к базе данных. Вместо этого они отправляют запросы на сервер, который обрабатывает эти запросы, выполняет необходимые операции с базой данных и отправляет обратно клиенту результаты выполнения запроса. Это обеспечивает централизованное управление данными, что повышает безопасность и согласованность информации.

Программное средство реализует паттерн «Наблюдатель» или «Observer» на клиентской части приложения с помощью библиотек JavaScript React и Redux.

Концепция паттерна «Наблюдатель» используется для установления зависимости между объектами. Основная идея заключается в том, чтобы иметь один объект, называемый субъектом, и несколько зависимых от него объектов, называемых наблюдателями. Субъект содержит список наблюдателей и предоставляет методы для добавления, удаления и оповещения наблюдателей об изменениях своего состояния. Наблюдатели оповещаются об изменениях в субъекте и могут реагировать соответствующим образом. Паттерн наблюдатель позволяет создавать слабосвязанные системы, где изменения в одном объекте автоматически приводят к изменениям в других объектах. «Наблюдатель» позволяет реализовать асинхронную коммуникацию между объектами. Он способствует уменьшению дублирования кода, так как позволяет разделить логику обновления объектов от их основной функциональности. Использование этого паттерна упрощает добавление новых наблюдателей и изменение способа оповещения об изменениях.

Store – главный элемент хранилища состояния. Будучи основной функцией Redux, он возвращает дерево состояния и в нем же реализованы методы подписки и уведомления. Объект содержит в себе данные о текущем пользователе и его корзине автозапчастей. Информация о пользователе включает в себя идентификационный номер пользователя в системе, его электронную почту, роль в приложении и выбранное самим клиентом имя пользователя. Информация о выбранных автозапчастях хранится в виде JavaScript-объекта Map, где в роли ключа выступает идентификационный номер товара, а в роли значения – его количество. Дерево состояний приложения в инструментах разработчика представлено на рисунке 2.

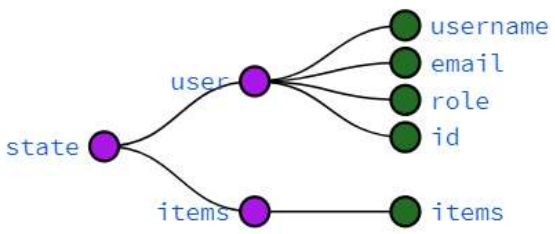

Рисунок 2 – Дерево состояний приложения в инструментах разработчика

Помимо этого, использование собственных хуков в React можно рассматривать в качестве паттерна «Компоновщик» или «Composite». Использование собственных хуков позволяет строить более гибкие и масштабируемые приложения, разбивая сложную логику на меньшие части, которые могут быть легко подключены и переиспользованы в различных контекстах. В ходе разработки веб-сервиса были созданы собственные хуки: useInput и useInputValidation для корректной обработки событий ввода пользователей.

На рисунке 3 представлена диаграмма вариантов использования проектируемого приложения. Определены три основных актора: неавторизованный пользователь, авторизованный пользователь и администратор. Каждый актор располагает специфическим набором возможных действий [3]. Все варианты использования нацелены на максимально эффективное использование системы: пользователям доступна информация о сервисе в различных видах, покупка реализуемых станцией технического обслуживания автозапчастей. Администратор же регулирует основное содержимое приложения, обновляя его актуальными данными.

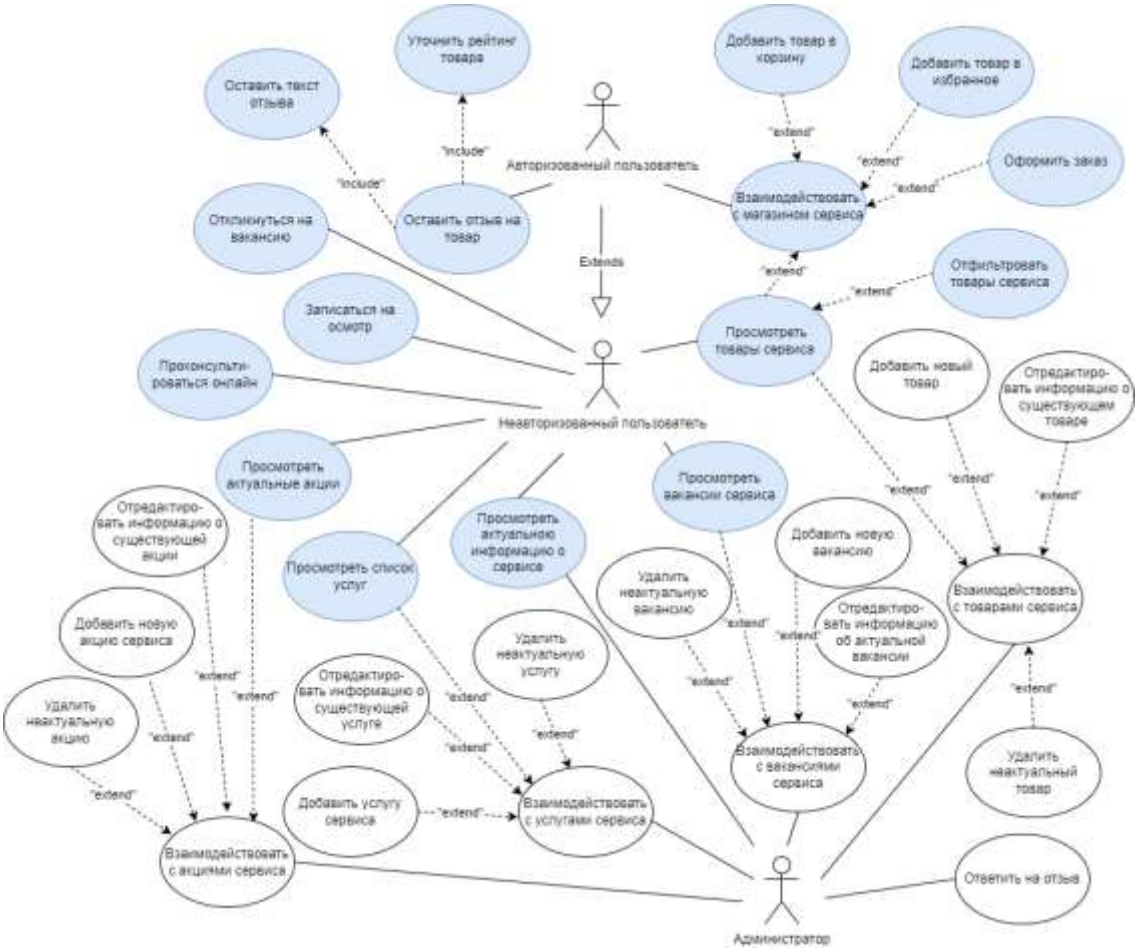

Рисунок 3 – Диаграмма вариантов использования программного средства

Создание базы данных играет важную роль в построении информационных систем. Она гарантирует эффективное хранение, доступ и управление данными, что критически важно для успешной работы бизнеса. Базы данных организуют информацию, обеспечивая её доступность для пользователей в нужный момент в нужном объёме. Это способствует ускорению процессов, обеспечивает безопасность данных и упрощает анализ для принятия обоснованных решений. В конечном итоге, разработка баз данных является ключевым элементом для оптимизации бизнес-процессов и успешной деятельности компании.

Даталогическая модель данных конкретизирует абстрактное представление структуры данных, определяя их типы, ограничения, индексы и другие технические детали, превращая абстрактную концепцию в конкретные таблицы, поля и отношения, готовые для реализации в базе данных. Даталогическая модель была разработана в MySQL Workbench, предоставляющей инструменты для создания структуры данных. Этот инструмент обеспечивает удобный интерфейс для создания таблиц, определения связей и атрибутов, а также позволяет визуализировать схему базы данных.

Использование MySQL Workbench позволило создать даталогическую модель, отражающую структуру данных и их взаимосвязи для дальнейшей работы с базой данных (см. рисунок 4).

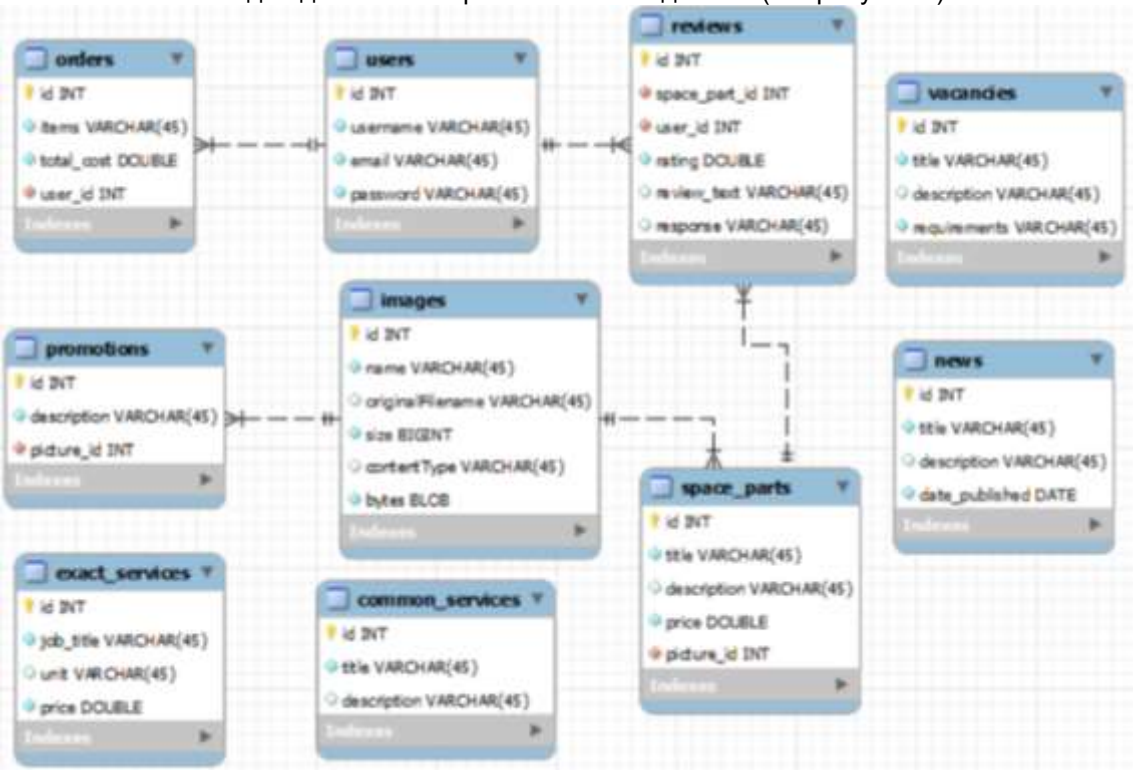

Рисунок 4 – Даталогическая модель базы данных

Диаграмма развертывания в UML используется для визуализации физической структуры системы, показывая, как программные компоненты размещаются на аппаратном оборудовании [4]. Она представляет конфигурацию устройств, серверов, связанных узлов и соединений, а также показывает взаимодействие между ними [5]. Диаграмма развертывания разрабатываемой системы представлена на рисунке 5. Эта диаграмма разъясняет, как разные части системы распределены по оборудованию, показывая, на каких устройствах функционируют программные компоненты. Её цель – помочь понять, какие аппаратные компоненты используются для поддержки приложений или сервисов и как они взаимосвязаны. Главная цель заключается в представлении физической конфигурации системы, что облегчает планирование и управление установкой, настройкой и поддержкой приложения.

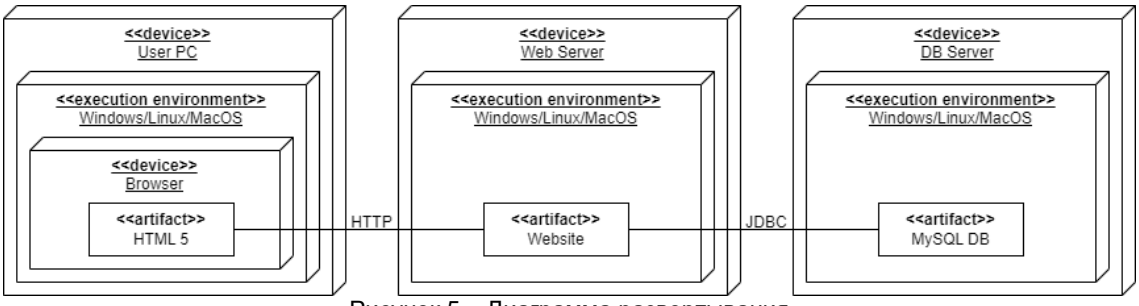

Рисунок 5 – Диаграмма развертывания

Диаграмма последовательности в UML – это визуальное представление взаимодействия между объектами или компонентами системы в определенном порядке. Она демонстрирует, какие сообщения передаются между ними в конкретной ситуации или в ответ на определенное событие.

Эта диаграмма иллюстрирует динамику взаимодействия объектов, показывая последовательность действий для выполнения определенной функции или сценария. Она помогает понять, как объекты обмениваются сообщениями и взаимодействуют друг с другом в процессе или операции.

Главная цель диаграммы последовательности заключается в наглядном отображении порядка событий и обмена сообщениями между объектами или компонентами системы [6]. Это способствует более полному пониманию и описанию логики работы системы, её функциональности и взаимодействия между её элементами. Диаграмма последовательности оформления заказа на автозапчасти представлена на рисунке 3.6.

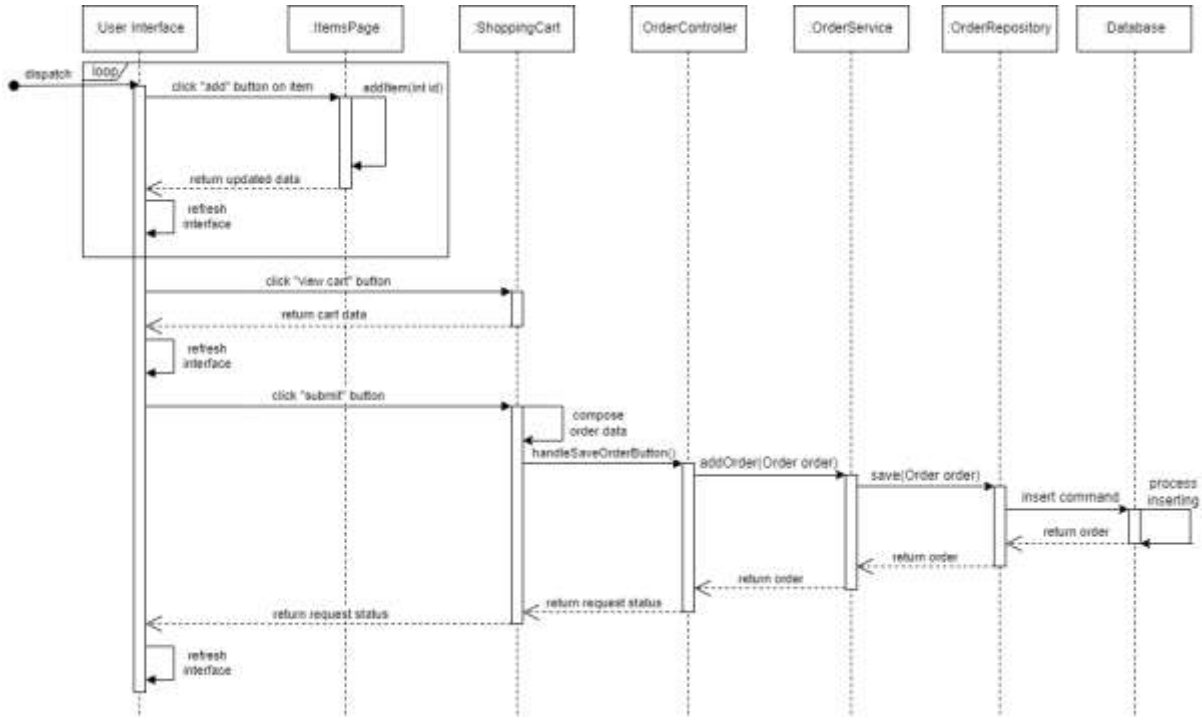

Рисунок 6 – Диаграмма последовательности оформления заказа на автозапчасти

Таким образом, клиенты имеют возможность воспользоваться функциями онлайн-консультации, записаться на осмотр и приобрести необходимые автозапчасти, в то время как администраторы контролируют и регулируют данные процессы, обновляя информацию по мере необходимости.

Разработанное приложение обладает удобным и интуитивно понятным пользовательским интерфейсом, отвечает всем упомянутым требованиям и успешно решает задачу совершенствования основных бизнес-процессов станции технического обслуживания.

Для достижения этой цели была тщательно исследована предметная область, изучены процессы работы станций технического обслуживания в Республике Беларусь и за ее пределами. Была выбрана методология работы над проектом, описаны технические требования. Были разработаны соответствующие UML-диаграммы и схемы, описывающие программное средство, были выбраны оптимальные технологии и архитектурные решения. Система была протестирована. Для удобства пользователей было составлено руководство по эксплуатации.

Приложение готово к использованию для оптимизации бизнес-процессов станции, совершенствуя их в сторону автоматизации и персонализации. Более того, оно легко масштабируется и может быть дополнено с течением времени, соответствуя изменяющимся требованиям в данной области.

Таким образом, все поставленные цели были успешно достигнуты, а разработанное программное средство готово для интеграции в информационную систему любой станции технического обслуживания автомобилей.

Список использованных источников:

1. Дехтеринский, Л.В. Ремонт автомобилей: учебник для вузов / Л.В. Дехтеринский, К.Х. Акмаев, В.П. Апсин. – М.: Транспорт, 2019. – С.47-53.

2. Волгин, В. Автосервис. Стандарты управления: Практическое пособие / В. Волгин. – М.: Дашков и К., 2023. – с. 58-71.

3. Диаграмма вариантов использования (UseCase diagram) [Электронный ресурс]. – Режим доступа: https://flexberry.github.io/ru/fd_use-case-diagram.html .– Дата доступа: 30.03.2024.

4. Unified Modeling Language (UML) | An Introduction [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction. – Дата доступа: 30.03.2024.

5. Что такое диаграмма развертывания? [Электронный ресурс]. – Режим доступа: https://www.guru99.com/ru/deployment-diagram-uml-exampl e.html. – Дата доступа: 30.03.2024.

6. Теория и практика UML. Диаграмма последовательности [Электронный ресурс]. – Режим доступа: http://it-gost.ru/articles /view_articles/94 .– Дата доступа: 30.03.2024.