Техническое задание – e-commerce платформа «Кедр»

1. Введение

Проект «Кедр» представляет собой разработку многофункциональной e-commerce платформы для компании, специализирующейся на оптовой продаже дверной фурнитуры в Украине. Платформа предназначена для обслуживания **оптовых клиентов** (ключевой сегмент бизнеса) с возможностью обработки и **розничных продаж** через единый веб-сайт. Планируется создание современного веб-приложения с удобным интерфейсом и интеграцией с корпоративной системой учета 1С. Платформа объединит в себе каталог товаров, онлайн-заказы, **личный кабинет клиента** и **административную панель** для управления всеми аспектами интернет-магазина.

В рамках данного технического задания описываются цели и задачи проекта, архитектура системы и используемые технологии, функциональные возможности для разных ролей пользователей, интеграция с внешними системами (в первую очередь с 1С), требования по SEO и производительности, меры безопасности, а также условия развёртывания, хостинга, документации и поддержки. Данный документ структурирован согласно принятым стандартам для крупных IT-проектов и предназначен для команды разработки уровня senior/architect.

2. Цели и задачи проекта

Цели проекта:

- **Расширение каналов сбыта:** Обеспечить компании выход в онлайн-канал продаж, чтобы увеличивать охват клиентов по всей Украине, особенно среди оптовых покупателей, а также упростить процесс розничных продаж.

- **Повышение эффективности заказов:** Автоматизировать прием и обработку заказов через интернет-магазин. Это позволит снизить нагрузку на менеджеров, уменьшить количество ошибок при оформлении заказов и ускорить цикл продажи.

- **Актуальность информации:** Гарантировать, что информация о товарах (наличие на складе, цены, описание) на сайте всегда актуальна за счет интеграции с 1С и регулярного обмена данными.

- **Персонализация для оптовых клиентов:** Предоставить зарегистрированным оптовым покупателям персональные цены и условия (индивидуальные прайс-листы, специальные скидки), доступные через личный кабинет.

- **Улучшение сервиса для клиентов:** Дать клиентам возможность самостоятельно получать всю необходимую информацию – от выбора товара и оформления заказа до отслеживания статуса заказа и просмотра истории покупок – без обращения к менеджеру.

- **Рост конкурентоспособности:** Создать современный, быстрый и SEO-оптимизированный веб-сайт, который повысит видимость компании в поисковых системах, привлечет новых клиентов и укрепит имидж компании как технологичного участника рынка.

Задачи проекта:

1. **Разработка веб-сайта с полнофункциональным интернет-магазином:** Реализовать каталог товаров с удобной структурой и поиском, функционал корзины и оформление заказа с учетом спецификаций оптовой торговли (например, возможность быстрых заказов по артикулу, выбор количества товаров оптом) и стандартов розницы.

2. **Создание личного кабинета клиента:** Предоставить пользователям после авторизации доступ к личному кабинету, где они смогут просматривать историю своих заказов, актуальные статусы заказов, свои индивидуальные цены на товары (для оптовиков) и скачивать персональные прайс-листы.

3. **Разработка административной панели:** Реализовать защищенную админ-панель для сотрудников компании (администраторов и менеджеров) с возможностями управления товарами (категории, позиции, остатки, цены), пользователями и их ролями, заказами (просмотр, изменение статусов, экспорта данных), а также настройками маркетинговых акций, скидок и промо-кампаний.

4. **Интеграция с системой 1С:** Обеспечить двунаправленную интеграцию с учетной системой 1С для автоматического обмена данными: регулярная выгрузка из 1С на сайт информации о товарах, остатках на складе и актуальных ценах; передача оформленных на сайте заказов в 1С для дальнейшей обработки. Интеграция должна работать надежно и в режиме близком к реальному времени, чтобы ни один заказ не был утерян и данные всегда были синхронизированы.

5. **Использование современного стека технологий:** Спроектировать архитектуру решения на основе **ASP.NET Core Blazor** (WebAssembly или Server) в сочетании с **Entity Framework Core** и СУБД **PostgreSQL**. Это обеспечит высокую

производительность, кроссплатформенность, и масштабируемость приложения.

6. **SEO-оптимизация и производительность:** Реализовать все основные технические SEO-настройки (человекопонятные URL, корректные мета-теги, **sitemap.xml**, **robots.txt** и пр.) для эффективного индексирования сайта поисковиками. Обратить внимание на оптимизацию скорости загрузки и отзывчивости сайта, поскольку это влияет на пользовательский опыт и SEO.

7. **Безопасность и надежность:** Обеспечить защиту данных пользователей и коммерческой информации компании. Реализовать современные механизмы аутентификации и авторизации, использовать шифрование для передачи конфиденциальных данных, защищать систему от распространенных веб-уязвимостей. Также обеспечить резервное копирование данных и отказоустойчивость системы.

8. **Документирование и поддержка:** Подготовить полный комплект документации по системе (пользовательской и технической), а также регламентировать процесс поддержки и развития платформы после запуска (обновления, исправление ошибок, добавление нового функционала).

Критериями успешности проекта будут запуск платформы «Кедр» в промышленную эксплуатацию, положительные отзывы пользователей (оптовых и розничных клиентов) о удобстве работы, увеличение доли онлайн-продаж в общей выручке компании, а также сокращение времени обработки заказов за счет автоматизации.

3. Архитектура и технологии

Общая архитектура системы: Платформа разрабатывается как монолитное веб-приложение на базе **ASP.NET Core Blazor Web App**, с разделением на многоуровневые компоненты (слои) для обеспечения чистой архитектуры. Выделяются следующие основные уровни:

- **Представление (UI):** Одностраничное приложение (SPA) на Blazor с использованием Razor-компонентов. Blazor обеспечит интерактивный пользовательский интерфейс в браузере без перезагрузки страниц, что улучшит опыт работы с каталогом и корзиной. Возможна реализация либо в режиме Blazor Server (рендеринг и обработка событий на сервере через SignalR-сессии) либо Blazor WebAssembly (логика работает на стороне клиента), выбор конкретного варианта будет зависеть от требований SEO и производительности. В любом случае, при использовании Blazor будет применяться компонентный подход, позволяющий повторно использовать UI-элементы и отделять логику отображения от бизнес-логики.

- **Бизнес-логика (Backend):** На серверной стороне ASP.NET Core реализует всю доменную логику приложения. Будут созданы сервисы и модули, отвечающие за обработку операций: управление корзиной, оформление заказа, расчёт цен с учетом скидок, применение бизнес-правил (например, минимальная сумма оптового заказа, проверка остатков и пр.). Логика будет централизована в сервисах (или менеджерах) таким образом, чтобы UI-компоненты Blazor вызывали эти сервисы для операций (следуя принципу разделения ответственности: UI-компоненты только отображают данные и принимают ввод, а всю обработку выполняют сервисы). Это упростит тестирование и поддержку кода.

- **Слой данных:** Используется **Entity Framework Core** (ORM) для доступа к базе данных **PostgreSQL**. База данных проектируется отдельно, на выделенном сервере или инстансе, и хранит данные о товарах, категориях, пользователях, заказах, ценах и т.д. EF Core обеспечит работу с данными в виде объектов .NET и позволит избежать SQL-инъекций благодаря параметризированным запросам. Предусмотрена миграция схемы БД через EF Migrations для удобства развёртывания и обновления. При проектировании БД будут учтены оптимизации для PostgreSQL: добавление необходимых индексов для ускорения поиска по каталогу и заказам, нормализация/денормализация данных в разумных пределах для баланса между производительностью и сложностью.

- **Интеграционный модуль:** Специальный компонент (или набор сервисов) для интеграции с 1С. Архитектурно он может быть реализован как фоновые задачи (Background Services) внутри ASP.NET Core приложения или как отдельный микросервис, взаимодействующий через очередь сообщений/ API. Этот модуль отвечает за обмен данными с 1С: получение обновлений остатков и цен, загрузку новых товаров, передачу заказов и др. Он будет работать автономно от веб-интерфейса (например, по расписанию или по событиям) и взаимодействовать со слоем данных (БД) для чтения/записи интегрируемой информации.

Основные технологии и стек:

- **Язык и платформа:** C# и .NET 7/8 (последняя стабильная версия .NET Core на момент разработки) для серверной части. Это обеспечит высокую производительность и современный функционал языка.

- **Веб-фреймворк:** ASP.NET Core Blazor. Данный фреймворк позволит реализовать интерактивный SPA с возможностью повторного использования .NET-кода на клиенте и сервере, а также облегчить последующую поддержку (вместо применения JavaScript-фреймворков).

- **ORM и БД:** Entity Framework Core для работы с реляционной СУБД PostgreSQL. PostgreSQL выбран как отказоустойчивая и масштабируемая СУБД с открытым исходным кодом, поддерживающая сложные запросы и хранимые процедуры при

необходимости. База данных размещается отдельно (на отдельном сервере или сервисе), что повышает безопасность и позволяет масштабировать хранилище независимо от веб-приложения.

- **Frontend-стек:** В основе фронтенда – Blazor (генерирует HTML/CSS из Razor-компонентов). Для стилевого оформления и адаптивности интерфейса будут использованы HTML5/CSS3, фреймворк Bootstrap 5 (или аналогичный CSS-фреймворк) для обеспечения responsive-дизайна, корректного отображения на разных устройствах. Также возможна минимальная необходимость JavaScript (например, для интеграции сторонних виджетов оплаты или аналитики), которая будет подключаться отдельно через Interop по мере необходимости.

- **Аутентификация и аккаунты:** Планируется использовать встроенную систему **ASP.NET Identity** для управления учетными записями пользователей (хранение хэшей паролей, сброс паролей, ролевая модель). Identity легко интегрируется с EF Core и PostgreSQL, позволяя хранить пользователей, роли и права доступа в базе данных. Это ускорит реализацию безопасной регистрации и входа, включая возможности расширения (например, добавление полей профиля пользователя).

- **Логирование и мониторинг:** Будут внедрены механизмы логирования событий и ошибок (например, с помощью библиотеки **Serilog** или **NLog**). Логи важны как для отладки во время разработки, так и для мониторинга в продакшене. В критических местах (например, при обмене с 1С, оформлении платежей) будут предусмотрены дополнительные журналы и оповещения об ошибках. В будущем возможно подключение системы мониторинга (Application Insights, Prometheus + Grafana, Graylog и т.п.) для отслеживания производительности и доступности.

- **Прочие компоненты:** Для отправки email-уведомлений (например, подтверждение заказа, восстановление пароля) будет интегрирован SMTP-сервис или API почтового сервиса. Если потребуется онлайн-оплата – возможно подключение SDK платежного шлюза (например, LiqPay, Fondy или другого, в зависимости от предпочтений компании). В случае наличия большого количества статических файлов (изображения товаров) – может быть применено хранение и раздача через CDN или облачное хранилище, но на этапе ТЗ это опционально.

Принципы архитектуры: При разработке будет применяться модульный подход и принципы SOLID. Кодовая база будет разделена на проекты/модули (например: Web/UI, Core/Domain, Data, Integration) для явного отделения ответственности. Внутри решения используется **Dependency Injection** (встроенный в ASP.NET Core контейнер) для гибкого связывания компонентов и упрощения тестирования. Также возможно использование шаблонов проектирования, таких как Репозиторий и Unit of Work для работы с данными, фабричный метод для создания определенных объектов (например, разных типов промо-акций) и т.д., чтобы облегчить поддержку и расширение функциональности.

Архитектура должна обеспечить горизонтальную масштабируемость: при росте нагрузки можно будет запустить несколько экземпляров веб-приложения «Кедр» за балансировщиком. Для этого приложение будет спроектировано статически масштабируемым (stateless) насколько возможно. Если используется Blazor Server (SignalR), для масштабирования потребуется поддержка sticky sessions или использование службы Azure SignalR для распределения соединений. Если Blazor WebAssembly + API – фронтенд может быть развернут на CDN, а бэкенд (Web API) масштабируется независимо. Окончательное решение по Blazor режиму будет принято с учетом компромисса между SEO (Blazor Server дает серверный рендеринг) и нагрузкой (Blazor WASM снижает нагрузку на сервер, но требует дополнительных API).

4. Пользовательские роли и функциональность

В системе «Кедр» предполагаются следующие основные **роли пользователей** и соответствующий функционал:

- **Неавторизованный посетитель (гость):** Любой пользователь, зайдя на сайт, может просматривать каталог товаров без авторизации. Доступные возможности для гостя включают:
    - Просмотр каталога товаров: навигация по категориям, фильтрация и поиск товаров по названиям, артикулам, характеристикам.
    - Просмотр страниц товара: детальная информация о товаре – фотографии, описание, характеристики, цена (розничная цена для гостей), наличие на складе или срок поставки.
    - Добавление товаров в корзину: гость может выбрать количество и добавить товар в корзину, просматривать содержимое корзины, изменять количество или удалять позиции.
    - Оформление заказа без регистрации (опционально): В случае поддержки гостевого оформления, пользователь сможет ввести необходимые данные для доставки и контакта при оформлении заказа. Однако, для оптовых заказов эта опция, скорее всего, будет отключена – оптовые клиенты должны быть зарегистрированы. Гостевое оформление может применяться лишь для розничных клиентов с небольшими заказами.
    - Регистрация и вход: возможность создать учетную запись, указав необходимые данные (имя, email, пароль, возможно телефон). После регистрации гость может стать зарегистрированным клиентом.
- **Зарегистрированный розничный клиент:** Пользователь, создавший учетную запись на сайте (или зарегистрированный администратором). Для розничных

клиентов (частные покупатели) функциональность включает все возможности гостя, а также:

- **Личный кабинет**: доступ к персональной странице. В личном кабинете розничный клиент может:
  - Просматривать историю своих заказов с детализацией (дата, состав заказа, сумма, статус выполнения).
  - Повторять предыдущие заказы (функция "заказать снова") или формировать шаблоны заказов.
  - Управлять персональными данными: обновлять контактную информацию (телефон, адрес доставки, email), изменять пароль.
  - Отслеживать статусы текущих заказов в режиме реального времени. Статус заказа обновляется либо вручную менеджером в админ-панели, либо автоматически на основании данных из 1С (например, оплачен, отправлен, выполнен).
  - Скачивать общедоступные материалы, например, прайс-лист розничных цен (если компания выкладывает общий файл).
- **Способы оплаты и доставки:** При оформлении заказа зарегистрированный пользователь выбирает способ оплаты (например, онлайн-оплата картой, оплата по счету для опта, наложенный платеж и т.д.) и способ доставки (курьером, через службу доставки Новая Почта и т.п.). Система должна запоминать последние использованные адреса доставки для удобства.
- **Уведомления:** Клиент получает уведомления по электронной почте о важных событиях: подтверждение регистрации, подтверждение оформления заказа, изменение статуса заказа (например, отправлен, ожидает оплаты), а также может получать маркетинговые рассылки (опционально, при согласии).

- **Зарегистрированный оптовый клиент:** Это ключевая категория пользователей – партнеры/дилеры компании, осуществляющие закупки оптом. Как правило, такие клиенты регистрируются на сайте, но их учетная запись привязывается к определенной организации и может требовать подтверждения администратором (для присвоения статуса "оптовик"). **Оптовый кабинет** включает функциональность розничного клиента, плюс дополнительные возможности:

    - **Персональные цены:** После входа под оптовой учетной записью, все отображаемые цены на сайте пересчитываются согласно индивидуальным условиям данного клиента. Например, каждому оптовому клиенту может соответствовать определенный тип цен или скидочный коэффициент, который хранится в 1С и передается на сайт. Клиент видит товары по своим тарифам (вместо розничных цен).

    - **Скачивание прайс-листа:** В личном кабинете оптовика должна быть возможность скачать актуальный прайс-лист с индивидуальными ценами. Формат прайса – XLSX или PDF, генерируемый системой. Данные для прайса берутся из БД (с учетом последних обновлений из 1С). Файл может содержать список товаров, упаковок, минимальные партии и цены конкретно для данного клиента. Это избавляет менеджеров от ручной рассылки прайсов.

    - **Быстрое оформление крупных заказов:** Для удобства оптовых клиентов можно реализовать функции ускоренного заказа, такие как ввод списка позиций по артикулу (например, загрузка CSV-файла со списком товаров и количеств), либо быстрый поиск по артикулу прямо в корзине. Это позволит оптовику, уже зная коды товаров, быстро собрать заказ.

    - **Отсрочка платежа / кредитный лимит:** (Если применимо) В аккаунте может отображаться информация о доступном кредитном лимите, состоянии баланса или отсроченных платежах, если это выгружается из 1С. Новые заказы оптовика могут учитываться в этом лимите.

    - **Индивидуальные условия:** Отображение договорных условий сотрудничества – например, прикрепление к аккаунту копий договоров, спецификаций, персональных скидок. Возможно, раздел "Документы" для оптового клиента, где он может скачать счет, акт и т.д., сформированные менеджером (неявно следует, что могут быть интегрированы документы из 1С или генерироваться на сайте).

- **Администратор / Менеджер системы:** Административные пользователи – сотрудники компании, имеющие доступ в **админ-панель** сайта. В зависимости от внутренних ролей компании, можно выделить:

    - *Администратор IT:* полный доступ ко всем настройкам системы, управление учетными записями других админов, конфигурация интеграции.

    - *Контент-менеджер / Менеджер каталога:* ведает наполнением каталога – добавление новых категорий и товаров (если это делается через сайт, а не только через 1С), редактирование описаний, загрузка фотографий,

управление атрибутами товаров.

- *Менеджер по продажам/заказам:* обрабатывает поступающие заказы в системе – просматривает новые заказы, проверяет корректность, меняет статусы (например, "Принят в обработку", "Отправлен", "Завершен"), помечает, оплачены ли они (если есть офлайн-оплата), и т.д. Также этот сотрудник может создавать заказы от имени клиента (например, если заказ поступил по телефону, менеджер может завести его через админ-панель).

- *Маркетолог:* управляет разделом скидок и акций – создание промокодов, настройка скидок на категории или товары, запуск промо-баннеров на сайт (если предусмотрен такой функционал).

- **Функциональность админ-панели** (доступная в совокупности вышеперечисленным ролям, в зависимости от привилегий):

  - **Управление товарами и каталогом:** Создание и изменение категорий (иерархия каталога), добавление новых товаров или редактирование существующих (название, описание, изображения, SEO-поля, привязка к категориям). Поля товара включают SKU (артикул), описание, бренд, характеристики, и **цены** – при интеграции с 1С часть этих данных будет синхронизирована, однако администратор может добавлять маркетинговую информацию (расширенное описание, дополнительные фото, теги для поиска). Если товар поступает из 1С, возможно, в админке он редактируется ограниченно (например, нельзя менять цену, иконка "управляется из 1С").

  - **Управление складскими остатками:** В идеале остатки импортируются из 1С автоматически, однако админ-панель может отображать текущие остатки по товарам, а также предупреждать о товарах с нулевым остатком. Возможно, администратор сможет вручную корректировать остатки или снимать товар с продажи (отмечая как "не отображать на сайте"), если нужно срочно скрыть позицию.

  - **Управление ценами и скидками:** Интерфейс для настройки ценовой политики. Если персональные цены задаются на стороне 1С, то администратор через сайт может только выбрать для пользователя соответствующий "тип цен" 1С. Однако, система может также позволять задавать скидки прямо на сайте: например, акция -20% на определенную категорию, промокод на скидку, специальные цены на товары (override стандартной цены). Эти маркетинговые акции админ может задавать и они будут применяться поверх базовых цен от 1С.

  - **Управление заказами:** Просмотр списка всех заказов, фильтрация по статусам (новые, в обработке, отправленные, выполненные, отмененные и

т.д.). Для каждого заказа админ видит все детали: товары, суммы, данные клиента, выбранные доставка/оплата. Менеджер может редактировать статус, добавлять внутренний комментарий (например, номер TTN отправления, примечание о созвоне с клиентом). Также должна быть возможность экспортировать заказы (например, в Excel) за период, либо интегрировано получение отчетов по продажам.

- **Управление пользователями:** Админ может создавать учетные записи для новых оптовых клиентов или сотрудников. Также – просматривать список зарегистрированных пользователей, сбрасывать им пароль при необходимости, присваивать или изменять роли (например, отметить учетную запись как "Оптовый клиент" – что может быть сигналом системе использовать индивидуальные цены). Для оптовиков – настройка их параметров сотрудничества (тип цены, скидка, закрепленный менеджер и т.п.).

- **Контент и информационные страницы:** Раздел для редактирования неглавных страниц сайта – таких как "О компании", "Контакты", FAQ, условия доставки и оплаты. Этот функционал обеспечит возможность контент-менеджеру править тексты без участия программиста.

- **SEO-настройки:** Для каждой страницы каталога и товара должны быть поля для ввода SEO-тегов: уникальный meta title, meta description, возможность задать **человекопонятный URL (ЧПУ)** вместо автоматически сгенерированного. По умолчанию ЧПУ генерируются из названия (транслитерация на латиницу), но админ может их откорректировать. Поддержка ЧПУ крайне важна, так как понятные URL улучшают ранжирование в поисковиках. Кроме того, админ-панель может иметь инструменты генерации sitemap.xml и управления robots.txt (например, встроенный редактор для robots.txt).

- **Логистика и интеграции:** Если сайт поддерживает интеграцию с курьерскими службами или платежными шлюзами, админ-панель должна позволять настраивать ключи API, точки отправки, тарифы доставки. Также, в случае интеграции с 1С, админы могут видеть статус соединения (например, дату последнего успешного обмена, ошибки обмена) и, при необходимости, вручную запускать внеочередной обмен.

- **Система уведомлений:** Админ-панель может отображать оперативные уведомления: о новых заказах (приходящие в режиме реального времени, чтобы менеджеры сразу видели), о сбоях интеграции, об ошибках на сайте. Возможно, реализована простая **панель мониторинга** (dashboard) на главной странице админки: сегодняшние продажи, количество заказов, топ

товаров, и т.д., для быстрого обзора состояния онлайн-продаж.

Каждый раздел админ-панели будет доступен только пользователям с соответствующими правами. Для разграничения прав используется ролевая модель (RBAC). Например, менеджер по контенту может иметь доступ только к каталогу и контенту страниц, но не видеть раздел пользователей и настройки. Безопасность админ-панели – особый приоритет, подробнее описана в разделе "Безопасность".

5. Интеграции и обмен данными

Одним из ключевых требований системы «Кедр» является тесная интеграция с корпоративной системой **1С:Предприятие** (конфигурация управления торговлей/складом, которая используется компанией для учета товаров, остатков, цен и оформления продаж). Интеграция должна быть надежной, автоматической и максимально бесшовной для пользователя. Ниже описаны основные аспекты и требования к обмену данными:

Данные, передаваемые из 1С на веб-сайт:

- **Каталог товаров:** Номенклатура (список товаров) должна регулярно выгружаться из 1С на сайт. Это включает уникальный идентификатор/артикул товара, наименование, базовую информацию. **Полная выгрузка** каталога необходима при инициализации системы (первичное наполнение БД сайта данными 1С). Далее предусмотрено обновление только изменившихся позиций (инкрементальные обновления), чтобы снизить нагрузку. Т.е. система должна получать из 1С только новые товары или изменения в существующих (например, изменение названия или параметров).

- **Остатки на складе:** Актуальное количество каждого товара на складе (или складах) будет передаваться на сайт. Эти данные критично обновлять часто (по возможности в режиме, близком к реальному времени, либо с частотой, например, каждые 10-15 минут), чтобы покупатели видели точное наличие. Поддерживается вариант, когда товары без остатка помечаются как "Нет в наличии" или не доступны для заказа.

- **Цены:** Для синхронизации цен используется механизм передачи типов цен из 1С. В 1С могут быть заведены несколько типов цен (розничная, оптовая, дилерская и т.д.). Необходимо выгружать актуальные цены для всех соответствующих типов. Сайт сопоставляет тип цены с категорией пользователя: неавторизованным и розничным клиентам показывается базовая розничная цена, оптовому клиенту – его специальная цена (например, оптовая или индивидуальная договорная). **Персональные прайсы**: если у каждого оптовика своя ценовая сетка, то на стороне 1С должны быть настроены индивидуальные типы цен или скидки по контрагенту. Сайт должен поддерживать получение этих индивидуальных цен

(например, по ID контрагента) либо расчёт скидок при отображении. Поскольку обновление цен – частая операция, она может выполняться несколько раз в день (или в момент изменения в 1С).

- **Прочие сведения:** При необходимости могут быть интегрированы и другие объекты: информация о контрагентах (клиентах), остатки денежных средств/кредитов, данные об оплатах. Однако, обязательный минимум для запуска – товары, остатки, цены.

Данные, передаваемые с веб-сайта в 1С:

- **Заказы клиентов:** Каждое оформленное на сайте **заказ** должно быть автоматически передано в 1С (создан документ "Заказ клиента" или аналогичный, связанный с контрагентом). Передача должна происходить немедленно после подтверждения заказа на сайте, чтобы менеджеры могли сразу обработать заказ в 1С. В случае временной недоступности 1С (например, отключен сервер), заказы должны ставиться в очередь и отправляться, как только соединение восстановится, чтобы ничего не потерялось. Важно предусмотреть уникальную идентификацию заказов, чтобы избежать дублирования (например, если повторно отправится тот же заказ).

- **Регистрация новых клиентов:** Если на сайте зарегистрировался новый оптовый клиент, его данные (название организации, контактные лица, реквизиты) могут быть переданы в 1С для заведения контрагента. Возможно, это делается не автоматически, а вручную менеджером, однако система должна поддерживать экспорт списка новых регистраций.

- **Обновление статусов заказов:** Обработка заказа может происходить в 1С (например, формирование расходной накладной, проведение оплаты и т.д.). Необходимо, чтобы изменения статуса заказа (например, "отгружен", "отменен", "оплачен") синхронизировались обратно на сайт. Это можно реализовать либо путем периодической выгрузки статусов заказов из 1С, либо по событию (1С сама дергает API сайта). В результате, клиент в личном кабинете всегда увидит актуальное состояние, а также может получить уведомление. **Пример:** после проведения отгрузки в 1С менеджер устанавливает статус "Отправлен", это должно отразиться на сайте и, возможно, отослать email клиенту.

- **Обратные данные:** Если на сайте ведется какая-то активность, влияющая на 1С кроме заказов (например, клиент изменил свои данные в личном кабинете – адрес, телефон), имеет смысл передать и эти изменения в 1С, чтобы карточка контрагента была актуальной. Такие интеграции можно запланировать на второй этап, основное – синхронизация заказов.

Технология и механизмы интеграции:

- **Протокол обмена:** Интеграция может быть реализована через веб-сервисы (REST/SOAP API) или через обмен файлами (XML/CSV) в оговоренной директории/FTP. Оптимальным современным подходом является использование **Web API** на стороне платформы «Кедр», который будет вызываться из 1С. 1С, начиная с версии 8.3, поддерживает HTTP-запросы и OData, что позволяет настроить прямое взаимодействие. Например, 1С может делать POST запросы на сайт для отправки обновлений, и GET запросы для получения новых заказов. Альтернативно, возможно использовать готовый механизм обмена с сайтами, присутствующий в типовых конфигурациях 1С (как используется для интеграции с Bitrix). В любом случае обмен должен быть защищенным (HTTPS, авторизация по токену или логин/пароль).

- **Частота обмена:** Для разных типов данных устанавливается разная частота синхронизации:

    - Остатки и цены – высокочастотный обмен. Желательно обновлять остатки постоянно или с минимальной задержкой (например, каждые 5 минут или по событию изменения в 1С). Цены можно выгружать несколько раз в день или в момент изменения прайсов.

    - Заказы – передача сразу же при появлении заказа (push-метод). Подтверждение получения заказа 1С может отправлять обратно (например, устанавливая признак/номер 1С у заказа в БД сайта).

    - Каталог – полный обмен (обновление ассортимента) можно делать реже, например, раз в ночь, если номенклатура меняется не часто. Однако, новые товары, добавленные в 1С, желательно оперативно публиковать на сайте; это может достигаться либо мгновенной выгрузкой из 1С при добавлении товара, либо промежуточным статусом "скрыт" и появлением товара после заполнения контента на сайте.

- **Двунаправленность:** Интеграция должна быть **двунаправленной**, то есть не только 1С снабжает сайт данными, но и сайт отдает в 1С результаты работы (заказы, регистрации). Лучшим решением для интернет-магазина является постоянный двусторонний обмен в реальном времени, что гарантирует непрерывность продаж и учета. Например, как только товар куплен на сайте – остаток в 1С уменьшается, как только в 1С обновили цену – сайт почти сразу показывает новую цену.

- **Обмен только измененными данными:** Во избежание избыточного трафика и нагрузки, система обмена должна по возможности пересылать только изменения. Например, если из 10,000 товаров обновились цены у 50 – выгружать только эти

50, а не весь прайс. Аналогично по остаткам: передавать только изменившиеся остатки. Такой подход уменьшит объем данных и ускорит интеграцию. Для этого придется вести журнал изменений в 1С или опрашивать с фильтром "дата изменения > последней синхронизации".

- **Обработка конфликтов:** Требуется определиться, какая система является "источником истины" для различных данных:
    - Для товарных остатков и цен – безусловно, **1С главная**, сайт просто отображает полученное.
    - Для описаний товаров, фотографий – вероятно, сайт основной (эти данные могут вообще не храниться в 1С). Нужно обеспечить, чтобы при обновлении каталога из 1С контент, добавленный на сайте, не затёрся. Решение: разделить поля, которые обновляются из 1С (артикул, название, цена, остаток) и поля, которые ведутся на сайте (описание, фото, SEO). Обновление из 1С не должно трогать поля контента.
    - Для заказов – 1С будет обрабатывать после передачи, но изменения статуса должны синхронизироваться обратно. Здесь нужна логика, чтобы например, администратор случайно не изменил статус на сайте, который конфликтует с фактическим в 1С. Вероятно, статус на сайте должен являться отражением статуса в 1С для тех заказов, что уже выгружены.
- **Логирование и надежность интеграции:** Каждый сеанс обмена нужно логировать. При возникновении ошибок (не удалось подключиться, неверные данные, таймаут) – система должна уведомлять ответственного (например, email разработчику или вывод сообщения в админ-панели). Должны быть механизмы повторной попытки передачи данных. Например, если отправка заказа в 1С не удалась, система повторит через 5 минут, и так несколько раз, пока не получится, либо пока админ не заберет данные вручную.
- **Тестирование интеграции:** Перед запуском на продуктиве должна быть выполнена обширная проверка обмена на тестовом контуре с боевыми данными, чтобы убедиться в корректном сопоставлении полей (mapping). Например, убедиться что все типы цен из 1С правильно распознаны, что приходящий из сайта заказ корректно создается в 1С (правильные единицы измерения, скидки, привязка к существующему контрагенту или создание нового).
- **Другие интеграции:**
    - **Платежные шлюзы:** Если внедряется онлайн-оплата, понадобится интеграция с платежной системой (через их API). Это косвенно интеграция, но ее тоже надо учитывать в архитектуре безопасности (webhook от

платежки для подтверждения оплаты).

- **Службы доставки:** Возможна интеграция с API служб доставки (например, «Новая Почта» в Украине) для расчета стоимости доставки и для отправки номера отслеживания обратно в систему. Если решено подключать, админ-панель может иметь модуль для генерации накладных доставки.

- **Аналитические сервисы:** Интеграция с Google Analytics, Яндекс.Метрика или другими – для сбора статистики посетителей. В техническом плане – подключение соответствующих скриптов в шаблон сайта.

Интеграция с 1С – критически важная часть проекта, от неё зависит консистентность данных между онлайн-платформой и офлайн-системой компании. Будет подготовлена отдельная спецификация (или раздел документации) с описанием деталей API/обмена: формат передаваемых файлов или запросов, поля, идентификаторы, и т.д., чтобы команды, отвечающие за поддержку сайта и 1С, говорили на одном языке.

6. SEO и производительность

Для успешного функционирования интернет-магазина «Кедр» необходимо уделить внимание **поисковой оптимизации (SEO)** и **производительности** сайта. Ниже перечислены требования и меры по этим направлениям:

SEO-настройки:

- **Человекопонятные URL (ЧПУ):** Все страницы сайта (особенно страницы товаров и категорий) должны иметь понятные URL-адреса, отражающие их содержимое. Вместо параметров и идентификаторов в адресе должны использоваться слова (название категории или товара транслитом). Пример: `/catalog/mezhkomnatnye-zamki/seriya-123` вместо `/catalog?id=1234`. Такие дружелюбные URL улучшают восприятие пользователями и положительно влияют на ранжирование в поисковиках. Система должна автоматически генерировать ЧПУ при создании новых сущностей, а также позволять администратору их редактировать для внесения SEO-ключевых слов.

- **Управление мета-тегами:** На каждой странице предусмотрены заполненные мета-теги `<title>` и `<meta name="description">`, соответствующие содержанию. Для динамических страниц (товары, категории) эти теги генерируются шаблонно (например, "Купить {Название товара} в Украине – {Бренд} – цена, описание" для title товара), но с возможностью ручной корректировки через админ-панель. Метатеги `keywords` опциональны (менее значимы для современных поисковых систем). Также важно наличие правильных заголовков H1

на страницах (для товара – обычно название товара).

- **Файл sitemap.xml:** Платформа должна автоматически генерировать и обновлять **sitemap.xml**, в котором перечислены все важные страницы сайта для индексации: страницы товаров, категорий, статические страницы. Sitemap помогает поисковым роботам быстрее находить и индексировать содержимое. При обновлении ассортимента или добавлении страниц sitemap должен актуализироваться (либо динамически, либо регенерироваться по расписанию, например раз в день).

- **Файл robots.txt:** Необходимо настроить **robots.txt** для указания поисковым роботам, что индексировать, а что нет. Например, закрыть от индексации служебные разделы (`/admin`, `/account` и т.д.), страницы фильтрации или пагинации, дублирующий контент (если такой есть). Также в robots.txt указывается ссылка на sitemap.xml. Robots.txt должен быть доступен по стандартному пути `/<robots.txt>`.

- **Микроразметка (структурированные данные):** Рекомендуется внедрить микроразметку schema.org для товаров (Product schema: название, цена, наличие, рейтинг если есть отзывы). Это поможет поисковикам формировать расширенные сниппеты (rich results) для страниц товаров (например, показывать цену и наличие прямо в результатах поиска).

- **Локализация контента:** Если планируется мультиязычный сайт (например, украинский и русский языки для охвата более широкой аудитории), необходимо предусмотреть структуру URL для языков и `hreflang` метки. Однако, в рамках текущего проекта язык интерфейса, скорее всего, один (предположительно русский или украинский), поэтому мультиязычность можно отложить или заложить как возможность на будущее.

- **Мобильная оптимизация:** Сайт должен иметь адаптивный дизайн (responsive) для корректной работы на мобильных устройствах. Это не только требование удобства для пользователей, но и фактор SEO (Google отдаёт предпочтение мобильным сайтам). Все элементы (меню, каталог, таблицы в кабинете) должны корректно масшабироваться под маленькие экраны.

- **Быстродействие и Core Web Vitals:** Технически, сайт должен соответствовать метрикам Core Web Vitals (Largest Contentful Paint, First Input Delay, Cumulative Layout Shift) – т.е. быстро отображать основной контент, быстро реагировать на действия пользователя и не иметь скачков макета. Для этого – минимизировать лишние скрипты, использовать сжатие ресурсов, обеспечить загрузку изображений оптимизированных размеров (например, создание thumbnail для превью товаров). В Blazor WebAssembly случае – воспользоваться prerendering (предварительный рендер на сервере), чтобы улучшить показатель LCP. В Blazor Server – контролировать объем данных, пересылаемых по SignalR, и использовать **lazy**

loading для тяжелых компонент (например, загрузка дополнительных данных по мере прокрутки).

Производительность и масштабируемость:

- **Оптимизация загрузки страниц:** Страницы каталога и товара должны генерироваться/отправляться пользователю с минимальной задержкой. Это достигается за счёт эффективных запросов к БД (использование необходимых индексов, **ленивая загрузка** связанных данных через EF Core или предварительная загрузка только нужных полей). При проектировании SQL-структуры и EF-моделей нужно избегать "N+1" проблем в запросах. В местах, где требуются сложные агрегаты (например, подсчет товаров в категориях) – возможно использование кэширования или подготовленных представлений.

- **Кэширование данных:** Следует применять кэш там, где данные относительно статичны и это даст выигрыш. Например, кэширование списка категорий (который меняется редко) в памяти приложения, чтобы не запрашивать его из БД при каждом обращении. Можно рассмотреть использование **Distributed Cache** (например, Redis) для распределенного кэша, если предполагается несколько серверов приложений. Также HTTP-кэширование на стороне клиента для статических ресурсов (CSS, JS, изображения) – настроить длительный `Cache-Control` заголовок, потому что эти файлы можно версионировать при деплое.

- **Виртуализация списков:** Если каталоги большие (тысячи товаров), в интерфейсе следует применять постраничную загрузку или виртуальный скроллинг, чтобы не загружать сразу весь список. Компоненты Blazor могут отрисовывать списки постранично. Также поиск по каталогу должен быть оптимизирован (можно добавить полнотекстовый индекс в PostgreSQL для поиска или использовать внешние поисковые движки по необходимости).

- **Производительность Blazor:** В случае Blazor WASM – обратить внимание на размер загружаемого .NET DLL в браузер, отключить неиспользуемые части (tpimming). В случае Blazor Server – контролировать нагрузку на сервер: каждый пользователь держит сигналR-соединение, поэтому сервер должен выдерживать одновременные подключения. Для больших нагрузок, возможно, потребуется scale-out (как упомянуто ранее) или перерасчет на Blazor WASM.

- **Тестирование нагрузкой:** До запуска провести нагрузочное тестирование (например, JMeter, k6) на ключевые сценарии: одновременное оформление заказа, одновременное просмотр каталога большим количеством пользователей. Цель – убедиться, что при пиковых нагрузках (например, 100-200 одновременных пользователей для начала, с возможностью роста) система реагирует приемлемо

(страницы открываются без больших задержек, не происходит сбоев). По результатам тестирования, возможно, нужно будет настроить connection pool к БД, оптимизировать тяжелые запросы, увеличить ресурсы сервера.

- **CDN и сжатие:** Использование CDN для статического контента (изображения товаров, библиотечные файлы JS/CSS) может значительно ускорить доставку контента пользователю, особенно если аудитория распределена географически. Также веб-сервер должен отдавать контент сжатым (gzip/br) для текстовых ресурсов. ASP.NET Core изначально поддерживает сжатие ответов.

- **Асинхронность и очередь задач:** Некоторые операции, которые могут быть медленными, лучше выполнять асинхронно. Например, генерация PDF-прайс-листа для большого каталога – стоит выносить во внешний процесс или поток, чтобы не держать пользователя. Пользователь инициирует – ему показывается уведомление, а файл генерируется и становится доступен для скачивания, или высылается на email. Другой пример – отправка email или интеграция с 1С – делается через фоновые задачи, чтобы веб-ответ пользователю не ждал окончания этих процессов.

В целом, архитектура должна быть спроектирована с учетом дальнейшего роста нагрузки: оптимизация запросов, потенциал для масштабирования, контроль потребления памяти. Придерживаясь этих мер, платформа «Кедр» будет быстрой, стабильной и поисково-видимой, что напрямую отразится на успехе e-commerce направления компании.

7. Безопасность

Безопасность является неотъемлемой частью разработки платформы «Кедр», учитывая наличие конфиденциальных данных (персональные данные клиентов, информация о заказах, цены, доступ к учетной системе 1С). В проекте будут реализованы следующие меры и требования по безопасности:

- **Безопасность соединений (HTTPS):** Весь трафик между пользователем и сайтом должен передаваться по защищенному протоколу HTTPS. Будет настроен SSL-сертификат для домена платформы. Это гарантирует шифрование передаваемых данных (защита от перехвата паролей, персональных данных, номеров заказов и т.д.). Также взаимодействие между веб-приложением и 1С (если происходит по HTTP) обязательно должно быть защищено SSL/TLS, особенно если 1С находится на внешнем сервере. Если используются файловые обмены – то через VPN или защищенное подключение к общему ресурсу.

- **Аутентификация и управление доступом:** Для входа пользователей используется **ASP.NET Identity**, обеспечивающий хранение паролей с современными методами хеширования (алгоритм PBKDF2 или Argon2). Пароли

никогда не хранятся в открытом виде. Будет введена политика паролей (минимальная длина, сложность) и, возможно, двухфакторная аутентификация (2FA) для учетных записей администраторов. Система **ролей и прав** четко разграничивает доступ: оптовик не может видеть чужие цены или чужие заказы, розничный клиент – доступ только к своему кабинету, администратор – доступ к админ-панели. Админ-панель будет требовать отдельного входа (можно тот же аккаунт с ролью админ). Попытки доступа к защищенным разделам без авторизации должны перенаправляться на страницу входа.

- **Защита от OWASP-уязвимостей:** Приложение изначально строится на платформе, которая уже предоставляет защиту от многих типичных уязвимостей:
    - **SQL Injection:** используемый ORM (EF Core) параметризует запросы, предотвращая инъекции. Однако, нужно быть внимательными с любыми ручными SQL-запросами (если они появятся) – все параметры должны экранироваться или параметризоваться.
    - **XSS (Cross-Site Scripting):** Blazor по умолчанию экранирует вывод переменных в разметке. Нужно избегать небезопасного рендеринга HTML без необходимости. Также, все вводимые пользователем данные (например, названия, отзывы, если будут) надо сохранять и показывать с экранированием. Для страниц админ-панели, где может вставляться произвольный HTML (например, описание товара), нужно либо позволять только безопасный поднабор тегов, либо предупреждать админов.
    - **CSRF (Cross-Site Request Forgery):** Формы и важные запросы должны быть защищены анти-CSRF токенами. В ASP.NET Core есть встроенная защита (Cookie Authentication + Antiforgery Tokens), она должна быть задействована для всех форм, особенно для POST запросов в личном кабинете и админке.
    - **Проверка входных данных:** Все данные, поступающие от пользователя (формы регистрации, оформления заказа, параметры в URL для фильтров) должны проходить валидацию на сервере. Это предотвратит, например, отправку некорректных значений в 1С или ошибки БД. Особое внимание полям, которые могут влиять на логику (цена, количество товара в заказе – их нельзя доверять с клиента, нужно пересчитывать на сервере).
- **Безопасность API и интеграции:** Если сторонние сервисы или 1С обращаются к сайту через API, эти API должны быть недоступны публично без авторизации. Будут использоваться как минимум **API-ключи или токены** для аутентификации интеграционных запросов, либо HTTP Basic Auth через защищённый канал. Запросы от 1С, например, могут включать в URL или заголовках секрет, который

проверяется на стороне сайта.

- **Логирование и аудит:** Вести журнал важных действий:
    - Входы пользователей (успешные и неудачные) – с фиксацией IP, времени (для возможного анализа атак).
    - Действия администраторов в панели (добавление/удаление товара, изменение цены, удаление пользователя, изменение прав) – все такие действия должны протоколироваться с указанием кто и когда сделал. Это позволит провести аудит при инцидентах.
    - Лог обмена с 1С – все операции (сколько товаров обновлено, сколько заказов отправлено) фиксируются, а при ошибках – с деталями, чтобы можно было быстро выявить проблему.
- **Защита от ботов и спама:** Поскольку сайт публичный, целесообразно внедрить защиту регистрационных форм и форм обратной связи от злоупотреблений. Например, reCAPTCHA на форму регистрации или ограничение частоты отправки запросов (throttling) для API. Это предотвратит массовую автоматическую регистрацию или скриптовое добавление товаров в корзину для DDoS.
- **Обновления и патчи:** Проект должен быть построен с возможностью легкого обновления зависимостей. В рамках поддержки безопасности важно регулярно обновлять .NET фреймворк, библиотеки (особенно связанные с безопасностью, например, Identity), а также саму ОС/СУБД. В требования включается использование поддерживаемых LTS-версий платформы .NET и своевременное применение security patches.
- **Шифрование данных в хранилище:** База данных PostgreSQL будет настроена с шифрованием соединения. При необходимости (если на уровне политики компании) – может использоваться шифрование данных "at rest" (например, Transparent Data Encryption) или хотя бы шифрование резервных копий, особенно для таблиц с чувствительной информацией (персональные данные клиентов). Однако на этапе разработки это дополнительная мера, которая может зависеть от инфраструктуры.
- **Резервное копирование и аварийное восстановление:** Безопасность – это еще и сохранность данных. Должна быть настроена регулярная резервная копия базы данных (например, ежедневный backup с хранением копий за 7 последних дней). Также, если хранятся на сервере файлы (изображения товаров, выгруженные прайсы) – их тоже нужно бэкапить. Необходимо документировать процедуру восстановления на случай сбоя сервера или потери данных.

- **Защита инфраструктуры:** Сервер или сервис хостинга, на котором разместится платформа, должен быть настроен безопасно: ограничить открытые порты (только 80/443 для веб, 5432 для БД если нужно, остальное закрыто), использовать firewall. Если развертывание на Windows – обеспечить актуальность антивируса и настройку прав; если на Linux – использовать best practices безопасности (не запускать приложение под root и т.д.).

- **Тестирование безопасности:** Перед запуском рекомендуется провести **пентест/сканирование уязвимостей** приложения. Это может быть автоматизированное сканирование (OWASP ZAP, Nikto) и ручная проверка критических сценариев. Выявленные уязвимости должны быть устранены. Также, настроить мониторинг аномалий (например, резкое увеличение 500 ошибок может свидетельствовать о попытке SQLi, а множество неудачных логинов – о переборе паролей).

Отдельно отметим, что интеграция с 1С тоже не должна компрометировать безопасность 1С. Поэтому, если веб-сайт обращается к 1С (которая может находиться внутри корпоративной сети), нужно обеспечить защищенный канал (VPN или DMZ). Возможно, стоит задействовать промежуточный API-шлюз для обмена, который будет торчать наружу, а 1С будет стучаться к нему, чтобы не открывать 1С сервер напрямую.

Все эти меры направлены на то, чтобы защитить систему и данные клиентов от угроз, сохранить доверие пользователей к платформе и соответствовать требованиям законодательства (в частности, законы о защите персональных данных в Украине).

8. Требования к развёртыванию и хостингу

Для успешного развертывания и последующей эксплуатации платформы «Кедр» необходимо предусмотреть соответствующую инфраструктуру и процессы. Ниже перечислены требования и рекомендации по окружению, хостингу и развертыванию:

- **Серверное окружение:** Приложение построено на .NET Core, что дает гибкость в выборе ОС. Возможны варианты:

    - **Windows-сервер + IIS:** Традиционный подход, учитывая интеграцию с 1С (которая сама под Windows). Можно использовать Windows Server 2019/2022, развернуть приложение под IIS (в режиме in-process). Плюс – возможность разместить 1С и сайт близко, упрощая обмен, минус – стоимость лицензий Windows.

    - **Linux-сервер:** .NET Core и PostgreSQL отлично работают на Linux. Например, Ubuntu Server 22.04 LTS. Веб-приложение будет работать под встроенным Kestrel-сервером, за которым может стоять Nginx как reverse

proxy для управления SSL и статикой. Это более экономичный вариант, и многие предпочитают Linux для веб-хостинга.

    - **Контейнеризация:** Альтернативно, подготовить Docker-контейнеры: один для веб-приложения (образ на основе `mcr.microsoft.com/dotnet/aspnet`), другой для PostgreSQL (стандартный образ Postgres). Это позволит запускать приложение в оркестраторе (Docker Compose, Kubernetes) и упростит перенос с сервера на сервер.

- **Характеристики сервера:** На этапе старта, нагрузка неизвестна точно, но предполагая десятки одновременных пользователей (оптовых клиентов) и периодические пики посещаемости, можно рекомендовать:

    - **App-сервер:** 4 ядра CPU, 8-16 ГБ RAM, SSD 100 ГБ (для системы и логов). RAM особенно важна, если Blazor Server, т.к. на каждого пользователя держится объект в памяти. CPU важен для генерации страниц и фоновых задач (но .NET довольно эффективен).

    - **DB-сервер (PostgreSQL):** Можно совместить с app-сервером на начальном этапе, но лучше выносить отдельно. Требования: 4 ядра CPU, 16 ГБ RAM, SSD-диск 100+ ГБ (в зависимости от размера каталога; лучше с запасом для роста данных заказов). PostgreSQL любит хорошее быстродействие дисков (NVMe) для транзакций.

    - Если БД и приложение на одном физическом сервере, железо стоит усилить (8 ядер, 32 ГБ) и настроить грамотную совместимость.

- **Отказоустойчивость:** На стартовом этапе можно развернуть в одном экземпляре (single node). Но в перспективе, особенно для критичного оптового бизнеса, стоит заложить возможности:

    - **Масштабирование веб-приложения:** Возможность добавить второй экземпляр приложения на другой ноде для балансировки нагрузки или failover. Потребуется настроить load balancer (например, Nginx или AWS ELB, Azure Traffic Manager и т.п. в зависимости от среды).

    - **Кластеризация БД:** PostgreSQL можно настроить в режиме потоковой репликации на реплику (hot-standby) для аварийного переключения. Или использовать управляемый сервис, если в облаке (Azure Database for PostgreSQL, Amazon RDS).

    - **Резервное копирование:** Упомянуто ранее – обязателен механизм регулярного backup. Например, ежедневный дамп PostgreSQL, сохраняемый

на удаленное хранилище или облако.

    - **Мониторинг инфраструктуры:** Желательно иметь мониторинг ресурсов (CPU, RAM, диски, сеть) – например, связка Prometheus + Grafana, либо готовые решения (Datadog, Zabbix). Это поможет быстро реагировать, если, скажем, диск заполнится или память перегружена.

- **Процесс развертывания (CI/CD):** Настроить конвейер CI/CD для автоматизации сборки и деплоя:

    - При изменениях кода и прохождении код-ревью, код сливается в основную ветку репозитория (например, Git). Должен запускаться автоматический Pipeline: сборка проекта, прогон юнит-тестов, сборка Docker-образа (если выбран контейнерный подход) или публикация артефактов.

    - **Тестовый стенд:** Следует иметь отдельную тестовую (staging) среду, максимально приближенную к боевой. Туда будут накатываться сборки для тестирования фич и приемки от бизнес-пользователей. Эта среда может содержать обезличенные копии данных из боевой (для реалистичных тестов).

    - **Продакшн-деплой:** Деплой на продакшн должен быть максимально без простоев. Для этого можно применять стратегию Blue-Green Deployment или Rolling (если позволяет инфраструктура). Если однонодовое развёртывание – то, как минимум, выполнять деплой в ночное время и по инструкции, минимизируя время перезапуска сервиса.

    - Версии приложения должны тегироваться; должна быть возможность отката на предыдущую стабильную версию в случае выявления критической проблемы.

- **Средства хостинга:** Возможны варианты:

    - **Облако (IaaS/PaaS):** Например, Azure (Virtual Machines или App Service + Azure Database for PostgreSQL), Amazon AWS (EC2 + RDS), либо локальные украинские облачные провайдеры. Облако дает удобство масштабирования и готовые сервисы безопасности, но может быть дороже.

    - **Собственный сервер (on-premises):** Компания может разместить сервер у себя или в датацентре. В этом случае необходимо обеспечить канал связи с достаточной пропускной способностью и резервирование интернета, если сайт для внешних клиентов. Также понадобится статический IP, настройка DNS для домена интернет-магазина.

    - **Хостинг-провайдеры для .NET:** Существуют площадки, специализирующиеся на .NET хостинге. Но для серьёзного проекта лучше контролируемое окружение или облако.

- **Домен и SSL:** Название домена для проекта (например, cedr.ua или по бренду компании) должно быть зарегистрировано. SSL-сертификат – либо приобретен у доверенного центра, либо можно использовать бесплатный Let's Encrypt с автоматическим обновлением каждые 3 месяца (подойдет, если устраивает доверие браузеров).

- **Хранение файлов и контента:** Изображения товаров, файлы прайс-листов, которые будут загружаться клиентами, надо где-то хранить. Возможности:

    - Локально на сервере (в папке wwwroot/uploads) – просто, но надо следить за резервным копированием и объемом.

    - Внешнее хранилище – например, AWS S3, Azure Blob Storage – оттуда раздавать через CDN. Это масштабируемо, но сложнее в настройке. На начальном этапе можно хранить локально, заложив интерфейсы, чтобы потом переключиться на облако.

- **Соображения развёртывания 1С:** Интеграция с 1С подразумевает, что 1С либо сама инициирует соединение, либо сайт как-то обращается к 1С. Если сайт в облаке, а 1С внутри офиса, может потребоваться VPN туннель или "проброс" API 1С наружу через безопасный канал. Эти моменты надо продумать совместно с 1С специалистами, чтобы при деплое всё соединилось.

Итого инфраструктура: Минимально – один сервер под Linux (NGinx + .NET приложение) + PostgreSQL (можно на том же). Лучше – два сервера (app + db). Оптимально – контейнеризация и оркестрация, но это более сложный путь. Решение примется исходя из бюджета и компетенций команды эксплуатации. В любом случае, документ с требуемыми параметрами будет передан ИТ-отделу или DevOps-инженерам для подготовки окружения.

9. Требования к документации и поддержке

Для успешного использования и дальнейшего развития платформы «Кедр» необходим комплекс документации, а также план поддержки. Ниже перечислены требования:

Документация:

- **Техническая документация разработчика:** Должен быть подготовлен подробный дизайн-документ по архитектуре системы. Он во многом пересекается с данным

ТЗ, но может содержать дополнительные диаграммы (например, диаграмма компонентов, схема базы данных, описание интеграционных потоков). Также необходимо документировать нестандартные технические решения, алгоритмы (например, расчет персональных скидок, алгоритм синхронизации).

- **Документация по API/интеграции:** Описать все точки интеграции с 1С: какие методы API используются, какие форматы запросов/ответов (JSON, XML), примеры. Если используются файлы обмена – образцы файлов, поле соответствия полей 1С и сайта. Это облегчит поддержку и доработку интеграции.

- **Руководство администратора (системного):** Документ, описывающий процесс деплоя приложения, его конфигурацию. В него входят: инструкции по установке необходимых компонент (.NET runtime, PostgreSQL), настройке переменных окружения (строки соединения, ключи API), шаги по обновлению версии. Также описание структуры проекта, чтобы другой разработчик/системный администратор мог быстро разобраться.

- **Руководство пользователя админ-панели:** Пошаговые инструкции для менеджеров компании, которые будут работать в админ-панели. Описать, как добавить товар, как изменить цену, как обработать заказ, как создать скидку и т.д. Желательно с иллюстрациями (скриншоты) после того, как интерфейс реализован. Это руководство позволит быстро обучить новых сотрудников и снизит количество вопросов к разработчикам после запуска.

- **Руководство пользователя сайта (опционально):** В принципе, сам сайт должен быть интуитивно понятным. Но если есть специфические возможности (например, оптовый быстрый заказ), можно подготовить раздел FAQ или отдельный документ, объясняющий клиентам, как этим пользоваться. Возможно, раздел "Помощь" на самом сайте.

- **Комментарирование кода:** Кодовая база должна содержать необходимый уровень комментариев (особенно в сложных местах). Публичные методы могут иметь XML-комментарии. Важно, чтобы следующий разработчик, читая код, смог понять логику. Критичные классы/методы должны быть описаны.

- **Сохранение знаний:** Желательно вести wiki или knowledge base в ходе проекта – где фиксировать архитектурные решения, решения по настройке сервера, часто возникающие проблемы и способы их решения. Это поможет при передаче проекта от команды разработки к команде сопровождения (если они разные).

Поддержка и сопровождение:

- **Тестирование и приемка:** Перед запуском проекта необходимо провести этап приемочного тестирования с участием ключевых пользователей от

компании-заказчика (например, менеджеров, которые будут использовать админку, и нескольких оптовых клиентов). Все выявленные дефекты фиксируются и устраняются. К моменту запуска система должна быть в состоянии **UAT-подтверждено** (User Acceptance Testing passed).

- **Режим поддержки после запуска:** Определяется период гарантированной поддержки разработчиками после релиза (например, 1-3 месяца), в течение которого они оперативно исправляют выявленные баги и помогают настраивать систему. Далее возможны два варианта:

    - Передача проекта на внутреннюю поддержку ИТ-отделу компании. В этом случае должна быть организована встреча/обучение для внутренних разработчиков или DevOps по особенностям системы.

    - Заключение договора на дальнейшую техническую поддержку с командой-разработчиком. Тогда прописываются SLA: время реакции на инциденты, время восстановления работы при сбоях, время на внесение мелких правок. Также возможна поддержка по расширению функционала (эволюционное развитие проекта).

- **Обращение в поддержку:** Нужно определить, как пользователи и персонал будут сообщать о проблемах. Например, завести почтовый ящик или трекер (Jira, Redmine) для багрепортов. Менеджеры должны знать, куда писать при возникновении ошибки на сайте или сбоя интеграции.

- **Обновления и улучшения:** В ходе эксплуатации, вероятно, появятся запросы на новые функции или изменения. Должен быть процесс управления изменениями: сбор запросов, оценка, планирование новых версий. Чтобы система оставалась актуальной, рекомендуется периодически (например, раз в квартал) планировать мини-версии с улучшениями или необходимыми обновлениями безопасности.

- **Мониторинг и реагирование:** Организовать мониторинг доступности сайта (например, с помощью внешнего сервиса, который пингует сайт и проверяет ответ). Если сайт падает – ответственные лица (админ, разработчик на поддержке) должны получить уведомление (смс, email) и принять меры. Также мониторинг ошибок: настроить сбор необработанных исключений (через Application Insights или Sentry) – при критической ошибке сразу уведомление разработчикам, чтобы быстро исправить, пока пользователи массово не столкнулись.

- **Производительность и аудит безопасности периодически:** Периодически (раз в год или по необходимости) выполнять аудит безопасности приложения, особенно если меняется окружение или появляются новые угрозы. Также анализировать производительность под возросшей нагрузкой – возможно, понадобится оптимизация запросов или масштабирование инфраструктуры по мере роста базы

данных и числа пользователей.

Все описанное выше должно помочь создать не только качественный продукт на момент запуска, но и обеспечить его стабильную и длительную работу. Документация и налаженная поддержка снизят риски зависания проекта или длительных простоев при возникновении проблем. Платформа «Кедр» станет стратегически важным инструментом бизнеса, поэтому ее сопровождение должно осуществляться на высоком уровне, в тесном взаимодействии между командой разработки и командой эксплуатации.