ТИТУЛ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Задание…

СОДЕРЖАНИЕ

ВВЕДНИЕ

В современном мире технологии постоянно развиваются, что приводит к значительным изменениям во всех аспектах нашей жизни. Электросамокаты за последние годы стали популярным средством передвижения в городских условиях, обеспечивая удобство и экологичность. Этот тренд стимулировал спрос на удобные и эффективные решения для покупки и аренды этих транспортных средств. В связи с этим разработка магазина электросамокатов представляется как актуальная и перспективная задача. Она не только открывает новые возможности для бизнеса, но и способствует продвижению экологически чистых видов транспорта.

Разработка такого проекта требует продуманной архитектуры, включая как бэкэнд, так и фронтэнд компоненты, что предполагает использование современных технологий и платформ. Важной частью проектирования информационных систем является обеспечение их масштабируемости, надежности и удобства в использовании.

Одним из аспектов, который будет рассмотрен в данной курсовой работе, является использование архитектурного подхода CORBA (Common Object Request Broker Architecture), который позволяет объектам в сети взаимодействовать друг с другом независимо от того, на каких платформах они реализованы. Это эссе рассмотрит CORBA и её роль в разработке распределённых приложений, что поможет понять, как можно интегрировать различные компоненты системы для создания эффективного и надежного сервиса продажи и аренды электросамокатов.

1 ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ АРХИТЕКТУРНЫ ПРИЛОЖЕНИЙ И ДАННЫХ. ОПИСАНИЕ CORBA – Common object request broker architecture и ее роли.

Введение

В настоящее время архитектура Common Object Request Broker Architecture (CORBA) продолжает играть значимую роль в разработке распределённых систем, несмотря на развитие и появление новых технологий. Созданная под эгидой Object Management Group (OMG), CORBA стала одним из первых и наиболее влиятельных стандартов, направленных на упрощение взаимодействия между программными компонентами, расположенными на различных платформах и написанных на разных языках программирования. Эта архитектура предоставляет механизмы для того, чтобы объекты могли прозрачно вызывать методы друг друга, находясь в одной локальной или разных сетевых средах, что делает её крайне важной для создания масштабируемых и гибких систем.

Введение CORBA в индустрию ИТ вызвало революцию в способах разработки программного обеспечения, позволяя разработчикам концентрироваться на бизнес-логике, а не на деталях взаимодействия между компонентами. Это значительно ускорило процессы разработки и внедрения сложных систем. В данном эссе будет рассмотрена история развития CORBA, её архитектурные особенности, роль в создании распределённых приложений, а также оценка преимуществ и недостатков использования данной архитектуры на фоне современных альтернатив. Этот анализ позволит лучше понять место CORBA в современном мире ИТ и её потенциал на будущее.

История и развитие CORBA

Исторический контекст

Common Object Request Broker Architecture (CORBA) была разработана в начале 1990-х годов организацией Object Management Group (OMG), которая состояла из нескольких сотен компаний, включая крупнейшие технологические фирмы того времени, такие как IBM, Microsoft, и Sun Microsystems. OMG была создана с целью разработки стандартов для облегчения взаимодействия между приложениями, написанными на разных языках программирования и работающими в разных средах.

Первые версии и основные принципы

Первая версия спецификации CORBA была выпущена в 1991 году. Основной идеей CORBA было создание стандарта, который позволял бы объектам в одних приложениях вызывать методы объектов в других приложениях, находящихся в любой части сети, независимо от платформы и языка программирования. Это было достигнуто благодаря использованию Object Request Broker (ORB), который действовал как посредник между клиентом и сервером, обрабатывая все детали связи.

Развитие и усовершенствования

С течением времени CORBA продолжала развиваться, включая новые возможности, такие как поддержка реального времени, транзакции, управление безопасностью и обработка исключений. Версии спецификации были расширены для обеспечения более высокой производительности и масштабируемости, что сделало CORBA более привлекательной для использования в критически важных приложениях.

Золотой век и пик популярности

Конец 1990-х — начало 2000-х годов можно считать "золотым веком" CORBA, когда она стала де-факто стандартом для разработки распределённых систем в корпоративном секторе. Благодаря своей универсальности и мощным возможностям CORBA нашла применение в различных отраслях — от финансовых услуг до телекоммуникаций и оборонной промышленности.

Конкуренция и спад

С появлением новых технологий, таких как веб-сервисы и архитектура ориентированных на услуги (SOA), популярность CORBA начала снижаться. Эти новые подходы предлагали большую гибкость и были более простыми в использовании по сравнению с относительно сложной архитектурой CORBA. Несмотря на это, CORBA по-прежнему используется в некоторых специализированных областях, где её возможности централизованного управления и строгой типизации оказываются критически важными.

Современное состояние и будущее

На сегодняшний день CORBA остаётся в арсенале некоторых систем, особенно там, где требуется высокая надёжность и совместимость с существующими системами. Организация OMG продолжает поддерживать стандарт, обновляя его для соответствия современным требованиям безопасности и взаимодействия. Однако в целом, с учётом развития современных архитектурных подходов, CORBA становится все более нишевым решением в мире программного обеспечения.

Техническое описание CORBA

Архитектура CORBA основана на нескольких ключевых компонентах, которые вместе обеспечивают ее функциональность:

Object Request Broker (ORB)

Object Request Broker (ORB) представляет собой основной компонент архитектуры CORBA, функционирующий как посредник между клиентскими запросами и серверными объектами. ORB выступает в роли спинного мозга распределённой системы, обеспечивая прозрачность взаимодействия между объектами, распределёнными по различным сетям.

Маршрутизация запросов: ORB определяет местоположение объекта-адресата, управляет соединениями и передачей данных между клиентом и сервером. Это включает в себя поиск объекта на сервере, установление соединения, сериализацию данных для передачи через сеть и последующую десериализацию на приёмной стороне.

Управление жизненным циклом объектов: ORB управляет созданием, удалением и другими аспектами жизненного цикла объектов CORBA. Это гарантирует, что ресурсы используются эффективно, и обеспечивает корректное управление памятью и другими системными ресурсами.

Обработка исключений: ORB также отвечает за обработку исключительных ситуаций, которые могут возникнуть во время взаимодействия объектов, например, сетевые ошибки, ошибки доступа к объектам или нарушения безопасности.

Interface Definition Language (IDL)

Interface Definition Language (IDL) — это специализированный язык программирования, используемый в CORBA для описания интерфейсов объектов, которые могут быть вызваны удалённо. IDL обеспечивает строгую типизацию и совместимость интерфейсов между различными языками программирования.

Независимость от платформы и языка: IDL является языком, независимым от конкретных платформ и языков программирования, что позволяет разработчикам определять интерфейсы объектов, которые могут быть

реализованы на любом языке, поддерживающем CORBA. Это способствует созданию действительно мультиплатформенных приложений.

Генерация кода: Из описаний IDL автоматически генерируется код "стабов" и "скелетов", которые используются соответственно на стороне клиента и сервера для маршрутизации вызовов и ответов через ORB. Это существенно упрощает процесс разработки и обеспечивает согласованное использование интерфейсов на всех этапах работы приложения.

Интеграция существующих систем: IDL позволяет включить в CORBA-систему уже существующие компоненты, обеспечивая их взаимодействие без необходимости полной переработки кода.

Использование ORB и IDL в CORBA делает возможным разработку распределённых приложений, в которых взаимодействие между компонентами является прозрачным, надёжным и эффективным, позволяя предприятиям создавать масштабируемые, безопасные и устойчивые системы.

Dynamic Invocation Interface (DII) и Dynamic Skeleton Interface (DSI): DII позволяет клиентам CORBA динамически составлять и отправлять запросы к объектам без необходимости статически компилировать стабы IDL. DSI выполняет обратную функцию, позволяя серверам динамически принимать и обрабатывать запросы, не зная заранее точных интерфейсов объектов.

Portable Object Adapter (POA): POA управляет жизненным циклом серверных объектов, их активацией, деактивацией и уникальным идентификатором объекта. Это позволяет объектам быть переносимыми между различными реализациями ORB.

CORBA-средства

CORBA-средства включают различные программные инструменты и библиотеки, предназначенные для поддержки разработки, внедрения и эксплуатации распределенных приложений, использующих архитектуру CORBA. Эти инструменты помогают разработчикам в создании, тестировании,

отладке и управлении жизненным циклом CORBA-приложений. Примеры таких инструментов включают:

IDL компиляторы: Преобразуют описания интерфейсов, написанные на языке IDL, в код на языках программирования, таких как Java, C++, или Python, который затем может быть использован для разработки клиентских и серверных частей приложения.

ORB реализации: Различные поставщики предлагают свои реализации ORB, которые могут отличаться по производительности, поддержке различных операционных систем и дополнительных функций.

Средства мониторинга и управления: Позволяют администраторам следить за работой приложений в реальном времени и управлять поведением компонентов системы.

GIOP/IIOP

General Inter-ORB Protocol (GIOP) — это обобщенный протокол для взаимодействия между различными реализациями ORB, которые могут быть развернуты в разных операционных системах и на различных платформах. GIOP определяет формат сообщений и последовательность операций для управления запросами и ответами между клиентом и сервером. Internet Inter-ORB Protocol (IIOP) является реализацией GIOP для сред, использующих протоколы TCP/IP, и представляет собой стандарт для взаимодействия в интернете.

Роль GIOP/IIOP: Протоколы GIOP и IIOP гарантируют, что запросы и ответы, передаваемые между ORB, интерпретируются корректно, несмотря на различия в машинах, операционных системах и языках программирования. Это обеспечивает интероперабельность и масштабируемость в распределенных системах.

Практическое использование: IIOP часто используется для развертывания распределенных приложений в интернете или внутрикорпоративных сетях, обеспечивая надежную и стандартизированную основу для обмена данными между приложениями.

Включение этих инструментов и протоколов в ваше эссе не только расширит понимание технологических аспектов CORBA, но и подчеркнет значимость стандартизации и интероперабельности в создании эффективных и масштабируемых распределенных систем.

Сервисы CORBA

Сервисы CORBA представляют собой набор предопределённых интерфейсов и функций, которые предоставляют расширенные возможности для разработки распределённых приложений. Эти сервисы облегчают управление сложными операциями и предоставляют дополнительную поддержку, необходимую для обеспечения корректного взаимодействия различных компонентов в распределённых системах. Вот подробное описание некоторых ключевых сервисов CORBA:

Naming Service

Назначение: Naming Service обеспечивает механизм для регистрации и поиска объектов по уникальным именам в распределённой сети. Это позволяет клиентам находить серверные объекты на основе логических имен, не зная физических адресов этих объектов.

Реализация: Сервис предоставляет иерархическую структуру для хранения имен объектов, что облегчает их систематизацию и поиск.

Transaction Service

Назначение: Transaction Service управляет группировкой нескольких операций в одну транзакцию, обеспечивая их атомарность, согласованность, изоляцию и долговечность (свойства ACID).

Реализация: Сервис позволяет разработчикам определять начало и конец транзакций, а также управлять их поведением при возникновении ошибок, обеспечивая откат или подтверждение изменений.

Security Service

Назначение: Security Service предоставляет функциональность для аутентификации пользователей, авторизации доступа, шифрования данных и управления политиками безопасности.

Реализация: Сервис интегрирует механизмы безопасности на уровне транспорта и приложений, позволяя настраивать уровни безопасности для различных компонентов системы.

Event Service

Назначение: Event Service позволяет объектам публиковать события, которые могут быть получены другими заинтересованными объектами, подписанными на эти события.

Реализация: Он поддерживает как синхронные, так и асинхронные модели взаимодействия, что делает его гибким инструментом для построения реактивных систем.

Life Cycle Service

Назначение: Life Cycle Service управляет жизненным циклом объектов CORBA, включая создание, удаление, копирование и перемещение объектов между доменами.

Реализация: Сервис предоставляет интерфейсы для управления состоянием объектов, что важно для поддержания целостности и устойчивости распределенных систем.

Каждый из этих сервисов играет критически важную роль в управлении сложностью распределенных приложений и помогает разработчикам сосредоточиться на бизнес-логике, минимизируя необходимость заниматься низкоуровневыми деталями взаимодействия и управления данными. Это существенно упрощает создание масштабируемых, безопасных и надежных приложений в современных корпоративных и коммерческих средах.

Как работает ORB

ORB действует как посредник между клиентскими запросами и серверными объектами. Когда клиент хочет вызвать метод удалённого объекта, он отправляет запрос ORB, который затем использует информацию, предоставленную через IDL, для маршрутизации и активации соответствующего серверного объекта. Ответ сервера возвращается клиенту через ORB, который обрабатывает все аспекты передачи данных, включая сериализацию и десериализацию данных.

ORB может работать в разных конфигурациях, поддерживая как синхронные, так и асинхронные вызовы методов, что обеспечивает гибкость при разработке распределённых систем.

Таким образом, CORBA предлагает комплексный набор инструментов и служб, обеспечивающих создание мощных и гибких распределенных приложений, способных работать в широком диапазоне операционных сред и платформ.

Взаимодействие клиента и сервера в архитектуре CORBA

На представленной диаграмме иллюстрируется процесс взаимодействия между клиентом и сервером в архитектуре CORBA. Начало взаимодействия инициируется с клиентской стороны, где пользовательский код (1) вызывает функцию или метод, представленный в стабе (2). Стаб — это клиентский прокси, сгенерированный на основе IDL (Interface Definition Language), который маршрутизирует вызовы от клиента к серверу через Object Request Broker (ORB). Затем ORB клиента (3) передает запрос через сетевой протокол TCP/IP (8) к ORB сервера, который направляет этот запрос к серверному скелету (4). Скелет, также сгенерированный по IDL, принимает вызов и перенаправляет его к конечному компоненту сервера — вызываемому объекту (callee) (5), который обрабатывает запрос и генерирует ответ.

После обработки запроса серверный объект отправляет ответ обратно к скелету (6), который через серверный ORB (7) передает его обратно клиентскому

ORB. Клиентский ORB, в свою очередь, передает данные обратно стабу (9), который завершает цикл, возвращая результат пользовательскому коду (10). Этот механизм обеспечивает эффективное и безопасное взаимодействие между удаленными объектами в распределенной среде, позволяя клиентским приложениям вызывать методы серверных объектов, как будто они находятся локально.

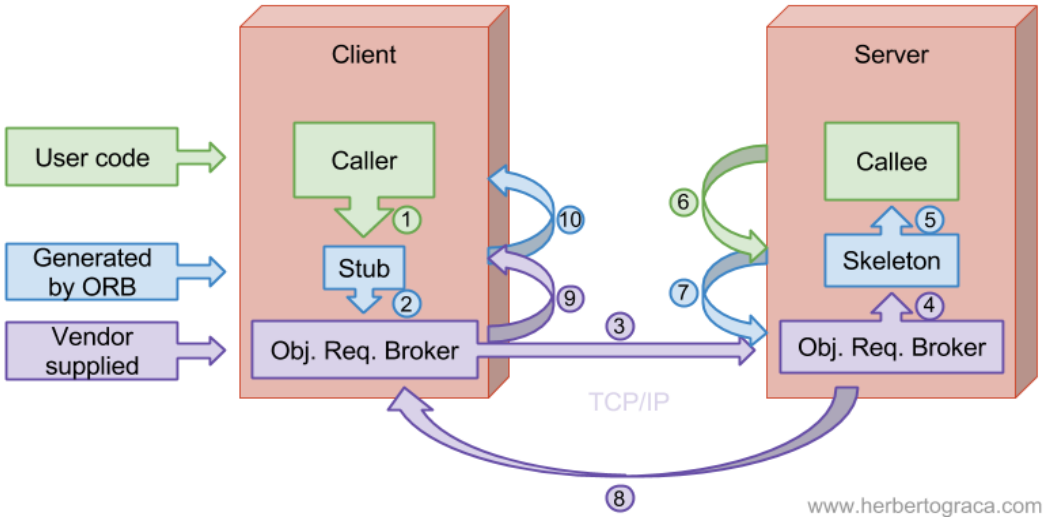

Рисунок 1 – диаграмма взаимодействия

Роль CORBA в разработке распределенных приложений

Взаимодействие объектов в различных средах выполнения

CORBA была разработана для того, чтобы упростить взаимодействие между объектами, расположенными в различных средах выполнения, что является ключевой характеристикой распределенных систем. При помощи ORB (Object Request Broker), CORBA обеспечивает прозрачное взаимодействие: клиенты могут вызывать методы удаленных объектов так, как если бы эти объекты находились локально. Это достигается благодаря стандартизированному интерфейсу IDL (Interface Definition Language), который описывает, как объекты должны взаимодействовать, не зависимо от языка программирования, на котором они написаны.

Интеграция с различными языками программирования

Одной из наиболее ценных особенностей CORBA является ее способность интегрироваться с множеством языков программирования. IDL поддерживает создание кода для интерфейсов объектов, который может быть использован в приложениях, написанных на различных языках, включая Java, C++, Python и многие другие. Это позволяет разработчикам использовать наиболее подходящий для задачи язык, сохраняя при этом способность объектов взаимодействовать друг с другом.

Прозрачность распределения

Концепция прозрачности распределения в CORBA позволяет разработчикам абстрагироваться от сложности сетевых взаимодействий. Разработчики могут фокусироваться на бизнес-логике, в то время как сложные аспекты взаимодействия между объектами на разных серверах или даже разных континентах обрабатываются автоматически. Это освобождает разработчиков от необходимости писать много кода, связанного с управлением сетью, сериализацией данных и обработкой ошибок.

Масштабируемость и гибкость

CORBA предоставляет мощные возможности для масштабирования приложений. Системы, построенные с использованием CORBA, могут легко расширяться по мере роста потребностей организации. Благодаря использованию ORB, который может эффективно управлять сетевыми запросами и балансировкой нагрузки, приложения могут масштабироваться для обслуживания большего числа пользователей без значительной переработки кода.

Безопасность

CORBA включает в себя обширные возможности по обеспечению безопасности. Security Service, одна из многочисленных служб, предоставляемых

CORBA, позволяет управлять аутентификацией пользователей, авторизацией доступа к объектам и защитой конфиденциальности данных. Эти возможности критически важны для корпоративных приложений, где требования к безопасности чрезвычайно высоки.

Адаптация к изменениям в бизнес-требованиях

Гибкость CORBA позволяет легко адаптировать системы к изменениям в бизнес-процессах и требованиях. Изменения в интерфейсах объектов могут быть внесены с минимальными нарушениями для клиентов, что делает CORBA идеальной для динамично развивающихся бизнес-сред.

Таким образом, благодаря своей универсальности, масштабируемости и поддержке разнообразия языков программирования, CORBA остается важным инструментом в арсенале разработчика при создании сложных, надежных и безопасных распределенных приложений.

Преимущества и недостатки использования CORBA

Преимущества CORBA

Межплатформенная интеграция: Одним из основных преимуществ CORBA является её способность обеспечивать взаимодействие между приложениями, работающими на различных платформах и написанными на разных языках программирования. Это достигается благодаря использованию Object Request Broker (ORB), который действует как посредник между клиентами и серверами.

Масштабируемость: CORBA поддерживает разработку распределенных приложений, которые легко масштабируются от небольших до очень крупных систем. Это делает CORBA подходящей для предприятий, которые ожидают роста или изменений в объемах данных и числе пользователей.

Безопасность: Сервисы безопасности CORBA предоставляют механизмы для аутентификации, контроля доступа, шифрования и управления политиками безопасности, что критически важно для корпоративных приложений.

Независимость от языка программирования: Использование Interface Definition Language (IDL) позволяет разработчикам определять интерфейсы объектов, которые могут быть реализованы на множестве языков программирования, обеспечивая гибкость в выборе технологий для разработки компонентов системы.

Поддержка сложных транзакций: CORBA включает в себя службы, которые поддерживают управление сложными транзакциями, что является необходимостью в многих корпоративных приложениях, особенно в финансовом секторе.

Недостатки CORBA

Сложность: Одним из основных недостатков CORBA является её сложность. Настройка, развертывание и поддержка CORBA могут потребовать значительных усилий и глубоких знаний, что увеличивает временные и финансовые затраты на проекты.

Производительность: в некоторых случаях использование ORB может вносить дополнительную нагрузку на систему, что снижает общую производительность приложений, особенно если не обеспечивается оптимальная конфигурация и тюнинг системы.

Альтернативные технологии: С развитием новых технологий, таких как веб-сервисы, RESTful API и микросервисы, многие организации переориентируются на более современные и гибкие решения, которые проще внедрять и поддерживать.

Избыточность для некоторых проектов: для некоторых проектов, особенно малых или с ограниченными требованиями к распределенным системам, CORBA может представлять избыточное решение, увеличивающее сложность без ощутимой выгоды.

Трудности в обслуживании и обновлении: С уменьшением популярности CORBA и уменьшением числа специалистов, владеющих этой технологией, поддержка и модернизация существующих систем могут стать проблематичными.

Исходя из этих факторов, организации должны тщательно взвешивать преимущества и недостатки использования CORBA в контексте своих специфических требований и целей, чтобы принять обоснованное решение о применении этой технологии в своих проектах.

Примеры применения CORBA в различных отраслях

Финансовый сектор

В финансовой индустрии CORBA была широко применена для создания высоконадежных и масштабируемых систем, таких как торговые и брокерские платформы, банковские информационные системы, системы управления активами и рисков. CORBA позволяет интегрировать различные финансовые приложения в единую сеть, что обеспечивает быстрый обмен данными и выполнение транзакций в режиме реального времени. Её способность управлять сложными транзакциями и обеспечивать надежную межплатформенную связь делает CORBA идеальной для использования в критически важных финансовых операциях.

Телекоммуникации

В отрасли телекоммуникаций CORBA применяется для управления сетевыми элементами и обслуживания клиентов в распределенных сетях. Системы на основе CORBA могут управлять сотнями тысяч событий в секунду, таких как маршрутизация вызовов, биллинг и управление сетевым трафиком. Эффективность CORBA в обработке распределенных событий и её способность

интегрироваться с различными программными платформами делают ее подходящим решением для этой динамично развивающейся отрасли.

Промышленность и производство

CORBA используется в промышленности для управления производственными процессами, автоматизации заводов и интеграции различных производственных систем. Например, CORBA может служить связующим звеном между системами управления производством (MES), планирования ресурсов предприятия (ERP) и управления оборудованием. Это обеспечивает централизованное управление данными и процессами, что критически важно для современных высокотехнологичных производств.

Оборонная промышленность

В оборонных и аэрокосмических системах CORBA применяется для создания сложных систем управления и мониторинга, которые требуют высоких стандартов надежности и безопасности. Эти системы часто включают интеграцию с широким спектром оборудования и программного обеспечения, где CORBA обеспечивает необходимую гибкость и масштабируемость.

Здравоохранение

В медицинских информационных системах CORBA используется для обеспечения взаимодействия различных медицинских приложений, таких как системы электронных медицинских записей, лабораторные системы и системы визуализации. Это обеспечивает надежный обмен данными между различными подсистемами и устройствами, что критически важно для оперативного предоставления медицинских услуг.

Кейсы успеха и проблемные ситуации

Во всех этих отраслях были случаи как успешного, так и проблемного применения CORBA. Успех часто обусловлен тщательным планированием,

глубоким пониманием требований к системе и правильным использованием возможностей CORBA для создания гибких, надежных и масштабируемых приложений. Проблемные ситуации, в свою очередь, часто возникают из-за недостатков в планировании, недопонимания возможностей CORBA или ошибок в интеграции с существующими системами, что подчеркивает важность компетенций и опыта в работе с этой технологией.

Актуальность CORBA

Сравнение с современными технологиями

В последние десятилетия IT-индустрия заметно преобразилась с появлением новых архитектурных подходов, таких как микросервисы, веб-сервисы и RESTful API. Эти технологии предлагают большую гибкость и простоту развертывания по сравнению с CORBA, что сделало их популярными в современных разработках. В отличие от CORBA, которая может быть избыточной и сложной в некоторых случаях, микросервисы предоставляют более легковесные и модульные решения, облегчающие непрерывное внедрение и масштабирование приложений.

Будущее CORBA и её место в современных ИТ-структурах

Несмотря на снижение популярности, CORBA по-прежнему сохраняет свою актуальность в определенных сегментах индустрии, где требуются высокая надежность и строгая типизация. В частности, в корпоративных системах, где уже внедрены решения на основе CORBA, полностью отказываться от этой технологии может быть нецелесообразно из-за высоких затрат на миграцию и потенциальных рисков, связанных с переходом на новые технологии.

Адаптация к новым реалиям

Чтобы оставаться релевантной, CORBA должна адаптироваться к изменяющемуся IT-ландшафту. Одним из возможных направлений развития

может стать улучшение интеграции CORBA с современными протоколами и архитектурами, например, обеспечение взаимодействия с RESTful API или улучшение поддержки облачных технологий. Это позволило бы CORBA более эффективно взаимодействовать с новыми системами и сохранить свою актуальность в разработке крупномасштабных и межплатформенных приложений.

Перспективы на будущее

В то время как многие организации могут рассматривать переход на более новые технологии, для некоторых секторов, таких как оборона и авиация, где системы должны соответствовать строгим стандартам безопасности и надежности, CORBA может продолжать оставаться предпочтительным выбором. В этих случаях, возможности CORBA в области распределенных транзакций и управления сложными взаимодействиями остаются важными.

Вкратце, хотя общий тренд в IT-индустрии движется в сторону более новых и адаптивных технологий, CORBA все еще имеет свое место в определенных нишах, где ее уникальные возможности по-прежнему востребованы. Оценивая перспективы на будущее, ключевым аспектом является гибкость и способность к адаптации к новым вызовам и требованиям современного программного обеспечения.

Вывод

Common Object Request Broker Architecture (CORBA) представляет собой заметную веху в истории развития информационных технологий, благодаря своей роли в фасилитации распределенного программирования и межплатформенной интеграции. Несмотря на смену технологических парадигм и появление новых архитектур, таких как микросервисы и RESTful API, CORBA

продолжает оставаться важным инструментом в некоторых специализированных областях, где требуется высокий уровень надежности и строгость типизации.

Значение CORBA в современной ИТ-индустрии

CORBA вносит вклад в разработку приложений, где важны стабильность, масштабируемость и способность к обширной интеграции. Она обеспечивает строгую согласованность интерфейсов и надежное взаимодействие между компонентами системы, что по-прежнему ценится в отраслях с высокими требованиями к безопасности и устойчивости систем.

Проблемы и возможности

Несмотря на свои достоинства, CORBA сталкивается с вызовами, связанными с устареванием и снижением популярности. Для сохранения релевантности CORBA необходимо адаптироваться к меняющимся технологическим требованиям и ландшафту. Улучшенная интеграция с новыми протоколами и технологиями, упрощение разработки и поддержки, а также увеличение гибкости архитектуры могли бы помочь CORBA оставаться востребованной.

Перспективы на будущее

CORBA, вероятно, продолжит играть ключевую роль в определенных нишевых областях, в частности, там, где требования к системам особенно строги. Однако, для широкого применения в более динамичных и меняющихся условиях современной разработки, CORBA нужно будет эволюционировать или интегрироваться с другими более современными технологиями.

В заключение, CORBA остается значимым элементом в пантеоне ИТ-технологий, обладая потенциалом для дальнейшего использования и развития. Она продолжает служить напоминанием о важности строгой стандартизации и возможностей межязыковой интеграции в разработке программного обеспечения, участвуя в создании устойчивых и надежных систем.

2 ПРИКЛАДНЫЕ АСПЕКТЫ РАЗРАБОТКИ АРХИТЕКТУРЫ ПРИЛОЖЕНИЙ И ДАННЫХ «МАГАЗИН ЭЛЕКТРОСАМОКАТОВ»

2.1 Описание проекта команды и разрабатываемого программного приложения «Магазин электросамокатов»

1. Цель создания приложения

Целью создания магазина электросамокатов является предоставление удобной, быстрой и доступной платформы для покупки и аренды этих транспортных средств. Проект направлен на удовлетворение растущего спроса на экологически чистые средства передвижения в городских условиях. Основная задача сервиса — облегчить доступ к электросамокатам, предоставляя функциональный и надежный интерфейс для выбора и заказа, а также способствовать экологической устойчивости через продвижение экологически безопасных транспортных средств. Этот сервис не только улучшит качество жизни городских жителей, обеспечивая больше возможностей для мобильности, но и способствует созданию более чистого и зеленого городского окружения.

2. Краткое описание создаваемого приложения

Магазин электросамокатов — это веб-приложение, позволяющее пользователям покупать и арендовать электросамокаты, а также управлять своими заказами. Приложение предусматривает различные уровни доступа с возможностями, адаптированными для разных типов пользователей:

неавторизованные посетители, зарегистрированные пользователи, модераторы и администраторы. Разработка направлена на создание удобного и функционального интерфейса с целью обеспечения простоты и комфорта при выборе и аренде транспортных средств. Приложение включает следующий функционал:

- Создать аккаунт
- Войти/выйти из аккаунта
- Просматривать и заказывать электросамокаты
- Управлять заказами и личными данными
- Модерировать контент и управлять пользователями

3. Описание объекта исследования

Исследование фокусируется на теоретических и практических аспектах создания архитектуры приложений и баз данных для магазина электросамокатов. Этот объект исследования включает в себя процессы выбора, покупки и аренды электросамокатов, а также управление пользователями и заказами через клиент-серверное приложение. Выбор такой архитектуры обусловлен несколькими ключевыми причинами:

Масштабируемость и управляемость: Клиент-серверная структура разделяет функциональные возможности между сервером и клиентами, что способствует улучшению общей масштабируемости и управляемости системы.

Безопасность данных: Данные централизованно хранятся на сервере, что позволяет осуществлять их защиту и контролировать доступ к ним более эффективно.

Эффективность обработки данных: Сервер обрабатывает все сложные запросы и операции, позволяя клиентской стороне концентрироваться на отображении результатов, тем самым увеличивая общую производительность приложения.

Гибкость в управлении доступом: Архитектура предоставляет возможности для детальной настройки прав доступа к различным функциям приложения, улучшая контроль и безопасность.

Таким образом, выбор клиент-серверной архитектуры является оптимальным решением для разработки магазина электросамокатов, учитывая его требования к масштабируемости, безопасности и эффективности.

4. Физически и/или юридические лица, которые непосредственно вовлечены в реализацию проекта, либо чьи интересы могут быть затронут при осуществлении проекта.

Проект разработки магазина электросамокатов предполагает активное участие не только технических специалистов, но и сторон, которые могут быть заинтересованы в продукте или его использовании. Ключевые участники проекта включают разработчиков, менеджеров, а также потенциальных пользователей и инвесторов, которые заинтересованы в коммерческом успехе и функциональности платформы. Важно, чтобы каждый участник проекта понимал свои роли и ответственности для обеспечения эффективной и скоординированной работы над проектом.

Ниже представлена таблица ответственности команды для полного представления области ответственности каждого участника команды.

Таблиц 1 - Матрица ответственности команды

Активности	Гуров Никита product manager	Мартынов Артем frontend dev	Самойлов Михаил backend dev	Курышкин Артем backend dev
Постановка целей и требований к проекту	RA	CI	CI	CI
Написание технического задания	RA	I	CI	CI

Продолжение таблицы 1

Организация митапов, поддерживание принципов SCRUM	RA	CI	CI	CI
Разработка пользовательского интерфейса	I	RA	CI	CI
Тестирование приложения	I	I	RA	RA
Развертывание приложения	AI	R	R	R
Поддержка приложения	A	R	R	R

Архитектура приложения "магазин электросамокатов" представлена в нотации Archimate и включает три основных слоя: бизнес-слой, слой приложений и технологический слой.

Бизнес-слой (жёлтый)

Верхний уровень архитектуры описывает бизнес-процессы и взаимодействия пользователей:

Пользователи:

- Незарегистрированный пользователь.
- Зарегистрированный пользователь.
- Модератор.
- Администратор.

Процессы и функции:

- Регистрация.
- Авторизация.
- Редактирование аккаунта.
- Просмотр товаров.
- Управление товарами в корзине.
- Управление заказами.
- Работа с отзывами.
- Работа с товарами.

- Работа с пользователями.

Информация:

- Данные пользователя (ФИО, почта, пароль, телефон, адрес доставки).

Слой приложений (голубой)

Средний уровень архитектуры описывает программные компоненты и сервисы, поддерживающие бизнес-процессы:

Сервисы:

- Сервис регистрации.
- Сервис авторизации.
- Сервис управления аккаунтом.
- Сервис просмотра товаров.
- Сервис управления корзиной.
- Сервис управления заказами.
- Сервис управления отзывами.
- Сервис управления товарами.
- Сервис работы с пользователями.

Данные:

- Данные регистрации.
- Данные авторизации.
- Данные аккаунта.
- Данные о товарах.
- Данные корзины.
- Данные заказов.
- Данные отзывов.
- Данные пользователей.

Технологический слой (зелёный)

Нижний уровень архитектуры описывает технологическую инфраструктуру, необходимую для работы приложений:

Серверы и инфраструктура:

- Сервер приложений:
- ОС Ubuntu Server.
- Docker.
- Сервер БД:
- PostgreSQL.

Сервисы инфраструктуры:

- Сервис доступа к приложениям.
- Сервис хранения данных.
- Сервис взаимодействия с сайтом.

Устройства пользователей:

- Операционные системы (Windows 11/7, MacOS, Linux).
- Магазин электросамокатов (веб-приложение).

Сеть и интернет:

- Интернет как средство взаимодействия пользователей с магазином.

Описание взаимодействий

Пользователи взаимодействуют с системой через регистрацию и авторизацию, после чего могут редактировать свои аккаунты, просматривать товары, управлять корзиной и заказами, оставлять отзывы и взаимодействовать с администраторами и модераторами.

Все бизнес-функции поддерживаются соответствующими приложениями и сервисами, которые обрабатывают и хранят данные в специализированных хранилищах данных. Эти данные управляются на сервере приложений и сервере баз данных, которые работают на ОС Ubuntu Server и используют контейнеризацию через Docker. Таким образом, архитектура приложения "магазин электросамокатов" организована по трёхуровневой модели, обеспечивая ясное разделение бизнес-логики, приложений и технологий.

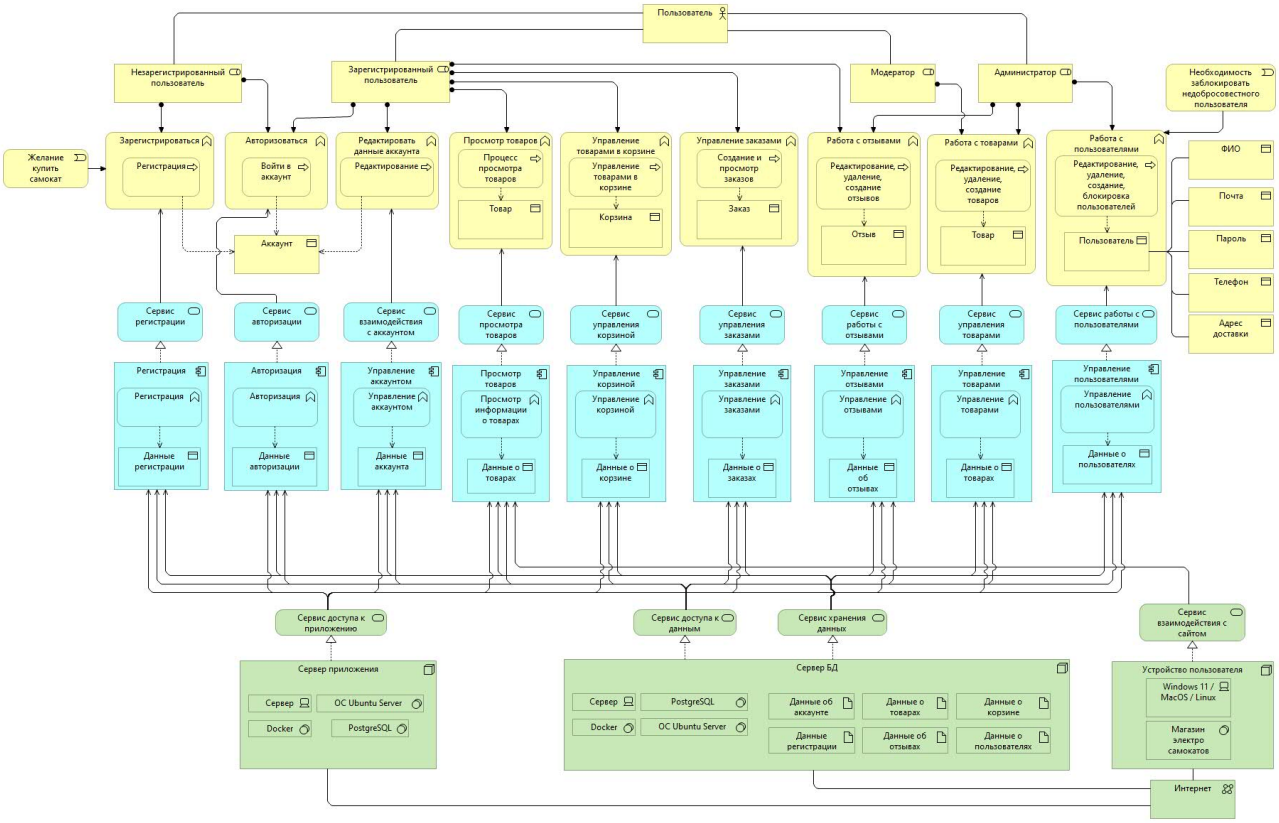

Рисунок 2 – общая архитектурная модель приложения

2.2 Описание роли неавторизованный пользователь в команде

1. Ролевая модель проекта

Таблица 2 – Ролевая модель проекта

Роль	Мера ответственности	Функционал
Неавторизованный пользователь	Просмотр ассортимента товаров	Просмотр страницы товара, чтение описаний и характеристик электросамокатов
Авторизованный пользователь	Использование функционала покупки электросамокатов	Вход в систему, выбор товаров, добавление их в корзину, оформление заказов, просмотр истории покупок.

Продолжение таблицы 2

Модератор	Управление каталогом товаров	Добавление новых товаров в каталог, удаление товаров из каталога, редактирование информации о товарах.
Администратор	Полный контроль над платформой	Все права модератора плюс управление отзывами пользователей, блокирование или разблокирование пользователей, настройка параметров системы.

В представленной выше таблице (Таблица 2 – Ролевая модель проекта) изложена ролевая модель онлайн-магазина электросамокатов. В таблице описаны основные роли пользователей платформы, а также их мера ответственности и соответствующий функционал, доступный каждой роли. В частности, таблица выделяет следующие роли:

- Неавторизованный пользователь: Ограниченный доступ, позволяющий только просматривать ассортимент товаров и получать информацию о продуктах.
- Авторизованный пользователь: Имеет возможности для выполнения покупок, включая вход в систему, выбор товаров, добавление их в корзину, оформление заказов и просмотр истории своих покупок.
- Модератор: Отвечает за управление каталогом товаров, включая добавление новых товаров, удаление и редактирование существующих записей.
- Администратор: Обладает полным контролем над платформой, выполняет все действия модератора, а также управляет отзывами пользователей, блокировкой и разблокировкой пользователей, настройкой параметров системы.

Эта модель является ключевой для понимания распределения обязанностей и доступного функционала в рамках различных уровней доступа к

системе, обеспечивая таким образом эффективное и целесообразное взаимодействие пользователей с онлайн-магазином.

2. Перечень основных обязанностей для роли неавторизованный пользователь

Роль "Неавторизованный пользователь" в магазине электросамокатов имеет следующие обязанности в соответствии с правилами и политикой платформы:

- Соблюдение правил и политик платформы: Неавторизованный пользователь обязан соблюдать все правила и политику магазина. Это включает запрет на размещение неприемлемого или нарушающего правила контента, дискриминации или оскорблений в комментариях или сообщениях.
- Уважительное поведение и коммуникация: Неавторизованный пользователь должен поддерживать уважительное и дружелюбное поведение во всех взаимодействиях с другими пользователями. Он должен избегать споров, конфликтов и провокаций, а также стараться создавать позитивную и поддерживающую атмосферу в сообществе.
- Содействие безопасности и модерации: Неавторизованный пользователь обязан помогать в обеспечении безопасности платформы. Это включает сообщение о любых нарушениях правил или нежелательных поведениях, а также сотрудничество с администрацией и модераторами при необходимости.
- Соблюдение правил конфиденциальности: Неавторизованный пользователь обязан соблюдать правила конфиденциальности и не разглашать личную информацию других пользователей без их явного согласия.

- Использование платформы по назначению: Неавторизованный пользователь обязан использовать платформу только по назначению, соблюдая все установленные правила и политики, что способствует улучшению качества услуг и функционала платформы.

Соблюдение этих обязанностей позволяет неавторизованным пользователям безопасно и эффективно взаимодействовать с платформой, поддерживать её порядок и вносить вклад в улучшение сервиса для всех пользователей.

3. Перечень основных взаимодействий с другими ролями и способы осуществления взаимодействий для роли неавторизованный пользователь

Неавторизованный пользователь: Взаимодействие с системой для этой роли в основном носит информационный характер и ограничивается следующими аспектами:

- Просмотр информации о продуктах: Неавторизованные пользователи могут просматривать страницы с описанием товаров, включая характеристики, цены, и фотографии электросамокатов. Это основное взаимодействие позволяет им собирать информацию о товарах без необходимости регистрации или входа в систему.
- Взаимодействие с контентом: Неавторизованные пользователи могут читать отзывы и комментарии, оставленные другими пользователями. Они могут также просматривать рейтинги товаров, что помогает формировать мнение о качестве и популярности продуктов.
- Получение информации о текущих акциях и специальных предложениях: Неавторизованные пользователи имеют доступ к информации о специальных предложениях, скидках и акциях, что может стимулировать

их на регистрацию и последующее использование полного функционала платформы.

- Информационные уведомления: Неавторизованные пользователи могут видеть уведомления и важные сообщения на главной странице магазина, такие как изменения в условиях обслуживания или важные новости относительно ассортимента.

Эти взаимодействия обеспечивают базовое знакомство с платформой и её предложениями, создавая основу для принятия решения о регистрации и активном использовании системы.

4. Основные артефакты проекта, с которыми взаимодействует роль «Неавторизованный пользователь»

Неавторизованный пользователь в магазине электросамокатов не создает артефактов, а взаимодействует с уже существующим контентом. Основные действия, доступные для неавторизованного пользователя, включают:

- Просмотр товаров: Неавторизованный пользователь может просматривать каталог электросамокатов, включая описание, характеристики и изображения каждого товара.
- Чтение отзывов: Неавторизованный пользователь может читать отзывы и оценки, оставленные другими пользователями, что помогает ему принять решение о покупке или аренде электросамоката.
- Поиск и фильтрация: Неавторизованный пользователь может использовать функции поиска и фильтрации для быстрого нахождения нужных товаров по различным параметрам, таким как цена, модель, характеристики и т.д.

Эти действия помогают неавторизованным пользователям эффективно использовать платформу для поиска и выбора электросамокатов, а также получать актуальную информацию о продукции и специальных предложениях.

2.3 Описание архитектуры программного приложения и данных «Магазин электросамокатов» для роли «неавторизованный пользователь»

1. Архитектурная модель приложения

Архитектурная модель приложения для пользователей магазина электросамокатов с ролью "Неавторизованный пользователь" включает следующие элементы и их взаимодействие:

- Взаимодействие с каталогом товаров: Неавторизованный пользователь имеет возможность просматривать каталог товаров, доступных на платформе. Архитектурная модель включает компоненты, отвечающие за отображение списка товаров, фильтрацию и сортировку по различным параметрам, а также представление детальной информации о каждом товаре, включая описание, характеристики и изображения.
- Авторизация: Неавторизованный пользователь имеет возможность зарегистрироваться или войти в систему, используя учетные данные. Архитектурная модель включает компоненты, отвечающие за обработку данных авторизации, проверку учетных данных, создание и управление учетными записями пользователей.

Эти взаимодействия обеспечивают неавторизованным пользователям доступ к основным функциям платформы, позволяя им получать информацию о товарах и осуществлять вход в систему для дальнейшего взаимодействия с приложением.

Ниже на рисунке 3 приведена архитектурная модель магазина электросамокатов для роли «Неавторизованный пользователь», сконструированная в Archi.

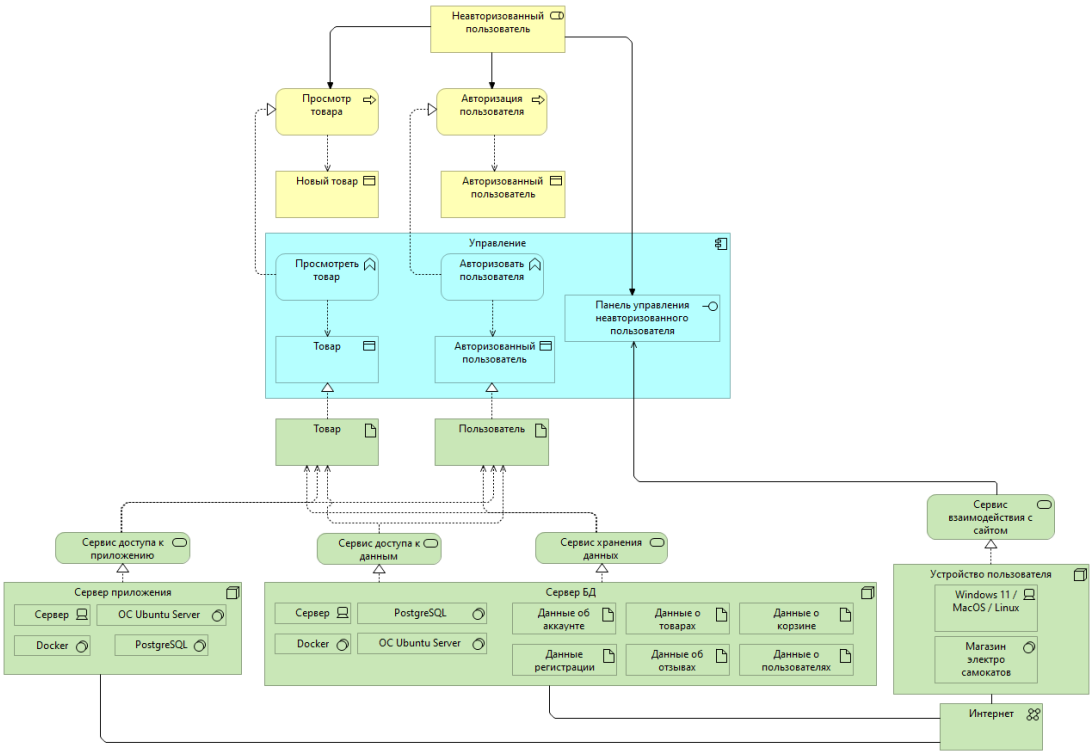

Рисунок 3 - Архитектурная модель разрабатываемого приложения, выполненная в нотации ArchiMate для роли «Неавторизованный пользователь»

2. Архитектурная модель данных

Архитектурная модель данных для неавторизованного пользователя в онлайн-магазине электросамокатов включает следующие сущности и их взаимосвязи:

- **Товар (product):** Сущность, представляющая отдельный товар в магазине. Товар содержит информацию о названии, стоимости, описании, дате добавления, характеристиках батареи, мощности, скорости и времени работы. Неавторизованный пользователь может просматривать все данные о товаре, чтобы сделать информированный выбор.
- **Отзыв (comment):** Сущность, представляющая отзыв пользователя на товар. Неавторизованный пользователь может просматривать отзывы, чтобы получить дополнительную информацию о товарах. Отзыв содержит

текст, оценку, дату создания и обновления, что помогает пользователям оценить качество и популярность товара.

- **Изображение (image):** Сущность, представляющая изображение товара. Включает тип контента, оригинальное название файла, дату создания. Неавторизованный пользователь может просматривать изображения товаров для лучшего понимания их внешнего вида и характеристик.

Эти сущности и их взаимодействия обеспечивают неавторизованным пользователям доступ к необходимой информации о товарах, что помогает им принимать осознанные решения при выборе электросамокатов. Просмотр изображений и чтение отзывов дополнительно улучшают их пользовательский опыт, предоставляя подробное представление о продукции магазина.

На рисунке 4 представлена модель данных для роли «Неавторизованный пользователь», на которой отражены все вышеописанные элементы.

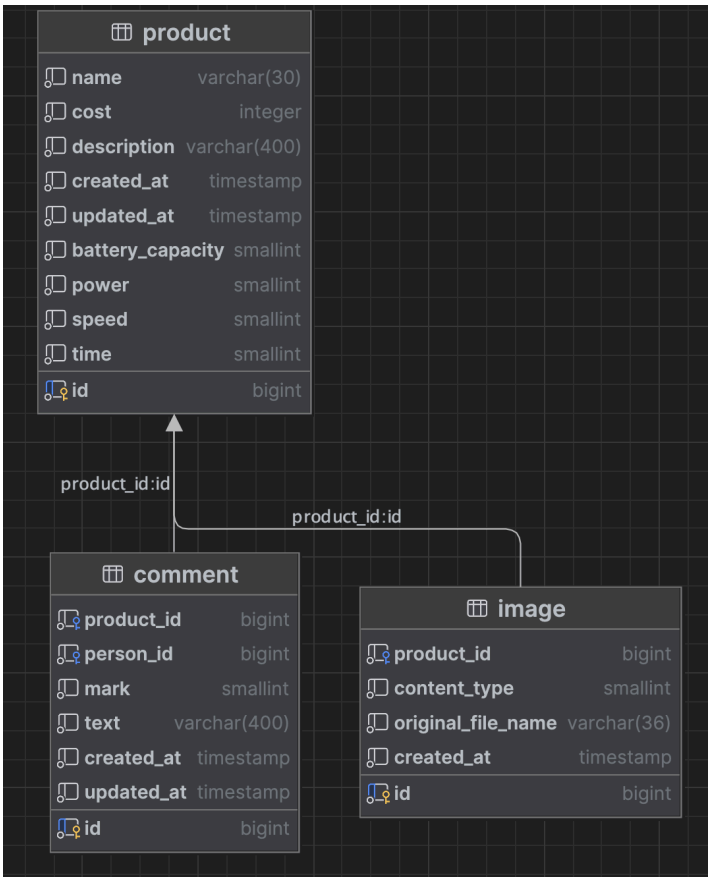

Рисунок 4 - Архитектурная модель данных разрабатываемого приложения для роли «Неавторизованный пользователь»

2.4 Варианты архитектурных моделей приложения «Магазин электросамокатов» для развития программного приложения

Для развития приложения «Магазин электросамокатов» можно выбрать микросервисную архитектуру. Микросервисная архитектура разделяет приложение на множество независимых сервисов, каждый из которых отвечает за выполнение конкретной бизнес-логики. Для данной архитектуры присуще такие положительные качества, как гибкость и возможность независимого развёртывания отдельных сервисов, улучшенное масштабирование засчёт возможности масштабировать только необходимые компоненты и упрощенная поддержка, и обновление отдельных частей приложения. Однако, появятся и следующие проблемы: усложнённое управление межсервисными взаимодействиями и данными, требование к более сложной инфраструктуре и DevOps процессам, а также необходимость тщательно прорабатывать механизмы безопасности и мониторинга.

В нашем случае микросервисная архитектура не была выбрана из-за сложности развертывания и тестирования, и выбор пал именно на клиент-сервисную архитектуру. Модель для микросервисной архитектуры нашего приложения продемонстрирована ниже, на рисунке 5.

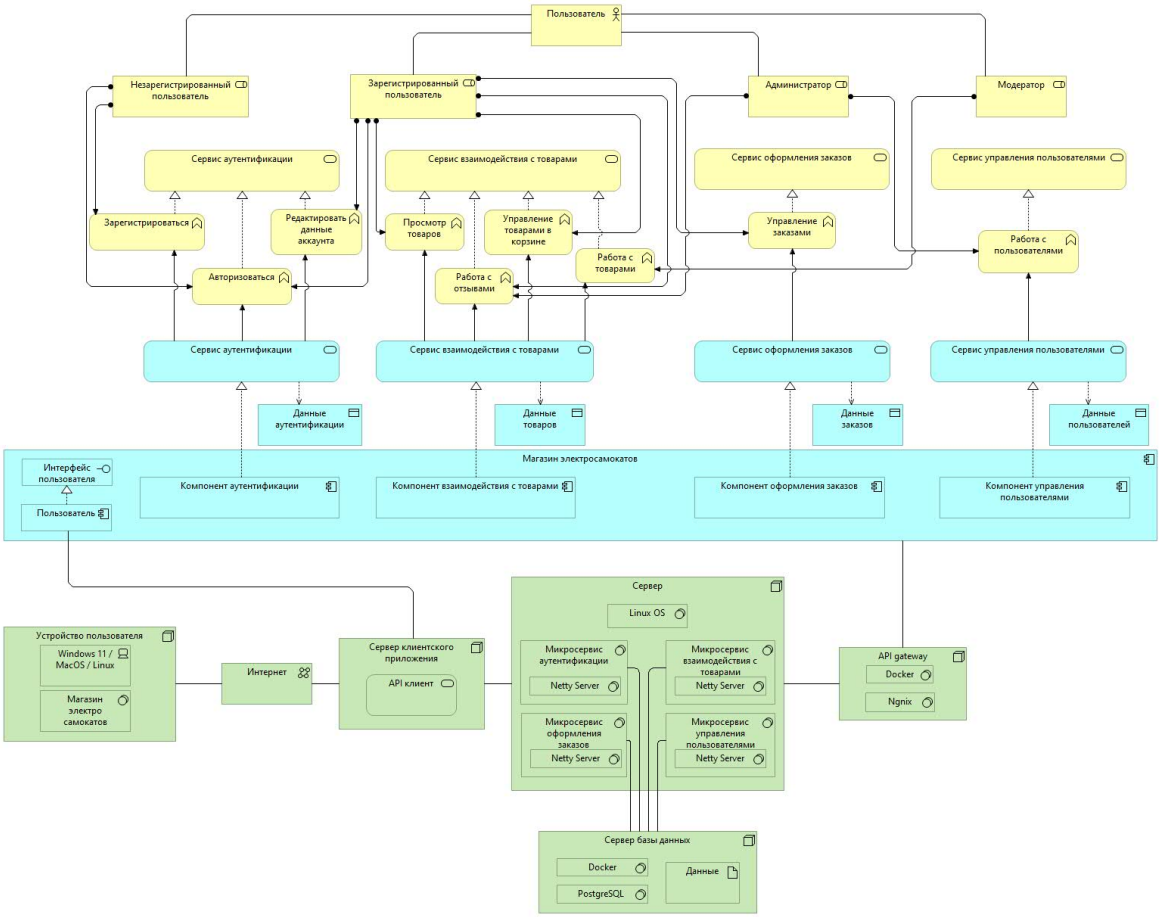

Рисунок 5 - Архитектурная модель данных для микросервисного подхода

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом была разработана комплексная система для онлайн-торговли электросамокатами. Проект включал создание как фронтэнд, так и бэкэнд компонентов, что позволило создать полноценный, удобный в использовании сервис. В процессе разработки были применены современные технологии и подходы, что обеспечило высокую производительность и масштабируемость системы.

В части работы, посвящённой архитектурному подходу CORBA, было продемонстрировано, как этот подход способствует созданию мощных и гибких распределённых систем. CORBA позволяет различным компонентам приложения взаимодействовать между собой в условиях распределенной сети, что существенно расширяет возможности проектирования и реализации разнообразных функциональных требований.

Таким образом, разработка данного магазина не только отвечает современным требованиям к технологиям и удобству использования, но и способствует продвижению экологически чистых транспортных средств. Этот проект демонстрирует, как с помощью инновационных технологий можно решать актуальные экологические и социальные задачи, предоставляя удобные и доступные решения для потребителей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Object Management Group (OMG) – Официальный сайт // URL: https://www.omg.org/ (дата обращения: 07.05.2024)
2. Robert Orfali and Dan Harkey «Client/Server Programming with Java and CORBA» // P. Орфали, Д. Харки. 1998. — 500 с. (дата обращения: 10.05.2024)
3. Alan R. Feuer «CORBA For Dummies / A. P. Фойер» // A. P. Фойер. 1998. — 350 с. (дата обращения: 13.05.2024)
4. Thomas J. Mowbray and Ron Zahavi «The Essential CORBA: Systems Integration Using Distributed Objects» // T. Дж. Моубрей, P. Захави. — 300 с. (дата обращения: 15.05.2024)
5. Archimate®3.0 Specification [Электронный ресурс]. – 2024. URL: https://pubs.opengroup.org/architecture/archimate30-doc/ (дата обращения 17.05.2024)
6. Archi – archimate modelling [Электронный ресурс]. – 2014. URL: https://www.archimatetool.com/ (дата обращения 20.05.2024)
7. Клиент-серверная архитектура в картинках [Электронный ресурс]. – 2020. URL: https://habr.com/ru/articles/495698/ (дата обращения 10.04.2024)
8. Микросервисная архитектура [Электронный ресурс]. – 2024. URL: https://www.atlassian.com/ru/microservices/microservices-architecture/ (дата обращения 15.04.2024)