ИТМО

И.В. Ананченко, Т.Е. Войтюк, Е.В. Марченко

АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

Санкт-Петербург
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

И.В. Ананченко, Т.Е. Войтюк, Е.В. Марченко

АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

УЧЕБНОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 11.03.02 Инфокоммуникационные технологии и системы связи
в качестве учебного пособия для реализации основных профессиональных образовательных программ высшего образования бакалавриата

Санкт-Петербург
2024

Ананченко И.В., Войтюк Т.Е., Марченко Е.В., АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ– СПб: Университет ИТМО, 2024. – 57 с.

Рецензент(ы):

Чумаков Сергей Иванович, кандидат технических наук, доцент, доцент кафедры системного анализа и информационных технологий, Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный технологический институт (технический университет)»;

Учебное пособие «Архитектура информационных систем» предназначено для студентов очной формы обучения, осваивающих профессиональную образовательную программу бакалавриата «Программирование в инфокоммуникационных системах» на направлении подготовки 11.03.02 «Инфокоммуникационные технологии и системы связи» (бакалавриат).

В пособии представлены общие теоретические вопросы, связанные с архитектурой информационных систем и практическое руководство по проектированию клиент-серверной архитектуры. В результате изучения материала студенты смогут: определять архитектуру информационной системы и знать ее отличительные особенности, применять паттерны и фреймворки при проектировании, осуществлять разработку масштабируемой информационной системы, отвечающей базовым принципам обеспечения безопасности.

ИТМО (Санкт-Петербург) — национальный исследовательский университет, научнообразовательная корпорация. Альма-матер победителей международных соревнований по программированию. Приоритетные направления: IT и искусственный интеллект, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication.

Лидер федеральной программы «Приоритет-2030», в рамках которой реализовывается программа «Университет открытого кода». С 2022 ИТМО работает в рамках новой модели развития — научно-образовательной корпорации. В ее основе академическая свобода, поддержка начинаний студентов и сотрудников, распределенная система управления, приверженность открытому коду, бизнес-подходы к организации работы. Образование в университете основано на выборе индивидуальной траектории для каждого студента. ИТМО пять лет подряд — в сотне лучших в области Automation & Control (кибернетика) Шанхайского рейтинга. По версии SuperJob занимает первое место в Петербурге и второе в России по уровню зарплат выпускников в сфере IT. Университет в топе международных рейтингов среди российских вузов. Входит в топ-5 российских университетов по качеству приема на бюджетные места. Рекордсмен по поступлению олимпиадников в Петербурге. С 2019 года ИТМО самостоятельно присуждает ученые степени кандидата и доктора наук.

Содержание

1. Введение. Основные понятия архитектуры информационных систем

Современные ИС и информационные технологии достигли такого уровня развития, когда на первый план выходит бизнес-оценка проектов, а не личные пристрастия разработчиков или заказчиков. В связи с этим большое внимание в настоящее время уделяется архитектуре информационных систем.

Учебное пособие «Архитектура информационных систем» предназначено для студентов очной формы обучения, осваивающих профессиональную образовательную программу бакалавриата «Программирование в инфокоммуникационных системах», направление подготовки 11.03.02 «Инфокоммуникационные технологии и системы связи». Учебное пособие включает в себя теоретический материал и практические примеры, необходимые для успешного освоения дисциплин «Жизненный цикл программного обеспечения» и «Создание клиент-серверных приложений».

Учебное пособие обобщает теоретический материал по типам архитектур информационных систем (ИС), паттернам проектирования ИС, а также освещает этапы проектирования ИС. Изложенный материал позволяет дополнить материал основного курса лекций и может быть использован в качестве дополнительной литературы для выполнения практических заданий студентами.

Изучение учебного пособия способствует формированию у обучающегося следующих компетенций:

- ОПК-2 «Способен осуществлять профессиональную деятельность с учетом экономических, финансовых, экологических, интеллектуально-правовых, социальных, этических и других ограничений на всех этапах жизненного цикла объектов профессиональной деятельности и процессов на основе оценки их эффективности и результатов» (индикатор ОПК-2.2 «Выбирает средства и технологии, в том числе с учетом последствий их применения в профессиональной сфере. Исследуют границы применения определенных решений в рамках профессиональной деятельности»);
- ОПК-4 «Способен к теоретическим и экспериментальным исследованиям в области профессиональной деятельности, включая постановку эксперимента, верификацию результатов, анализ, интерпретацию и презентацию данных» (индикатор ОПК-4.5 «Оформляет полученные результаты исследования и обосновывает их практическую и теоретическую значимость»);
- ПК-8 «Готов к определению принципов обеспечения информационной безопасности на уровне БД» (индикатор ПК-8.2 «Определяет автоматизированные процедуры выявления попыток несанкционированного доступа к данным»);
- ПК-12 Способен администрировать информационные службы и управлять базами данных инфокоммуникационной системы организации (индикатор ПК-12.1

«Оценивает необходимость инсталляции (установки) системы управления базой данных (СУБД) в информационных службах организации»).

Термин «архитектура» в применении к ИС уже давно стал привычным, так как грамотное построение информационной системы, эффективно и надежно функционирующей, является не менее сложной задачей, чем проектирование и возведение современного многофункционального здания [1].

Архитектура информационных систем является фундаментальным понятием в области разработки и проектирования компьютерных систем. Она представляет собой структуру и организацию компонентов информационной системы, их взаимодействия и принципы функционирования. Архитектура информационных систем играет важную роль в обеспечении эффективности, надежности и расширяемости системы, а также ее соответствия бизнес-требованиям.

1.1. Определение архитектуры информационных систем

Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию.

Архитектура – искусство и наука строить, проектировать здания и сооружения (включая их комплексы), а также сама совокупность зданий и сооружений, создающих пространственную среду для жизни и деятельности человека.

Известно множество определений понятия "архитектура ИС". Ниже приведены некоторые из этих определений:

1) *Архитектура ИС* – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы;
2) *Архитектура ИС* – базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и окружением, а также принципы, определяющие проектирование и развитие системы;
3) *Архитектура ИС* – набор значимых решений по поводу организации системы программного обеспечения, набор структурных элементов и их интерфейсов, при помощи которых компонуется система вместе с их поведением, определяемым во взаимодействии между этими элементами, компоновка элементов в постепенно укрупняющиеся подсистемы, а также стиль архитектуры, который направляет эту организацию (элементы и их интерфейсы, взаимодействие и компоновка) [1].

Все определения архитектуры ИС могут быть разделены на два класса – «концептуальные» и «технологические».

Концептуально архитектура ИС – набор основных решений, неизменных при изменении бизнес-технологии в рамках бизнес-видения и оказывающих определяющее влияние на совокупную стоимость владения системой [2].

Технологически архитектура обычно определяется следующими параметрами: назначением системы, составными частями системы и их размещением, связями между частями системы, принципами взаимодействия составных частей [2].

Технологически архитектура ИС – набор основных решений по выбору средств реализации, а именно, аппаратной платформы, операционной системы, телекоммуникационных средств, СУБД, влияющих на совокупную стоимость владения системой [2].

1.2. Значение архитектуры информационных систем

Архитектура информационных систем имеет огромное значение для разработки и успешной эксплуатации компьютерных систем. Она позволяет обеспечить согласованность и целостность системы, ее гибкость и масштабируемость. Архитектура информационных систем также помогает оптимизировать использование ресурсов, управлять сложностью системы, обеспечивать безопасность и конфиденциальность данных, а также удовлетворять бизнес-требованиям и потребностям пользователей.

1.3. Основные принципы архитектуры информационных систем

Существуют несколько основных принципов, которые лежат в основе архитектуры информационных систем:

1) *Модульность*. Принцип модульности заключается в разделении системы на независимые модули или компоненты, каждый из которых выполняет определенную функцию. Модульность упрощает разработку, тестирование и сопровождение системы, а также позволяет легко заменять или модифицировать отдельные компоненты без влияния на остальную систему.
2) *Иерархия*. Принцип иерархии предполагает организацию компонентов системы в иерархическую структуру, где каждый уровень имеет свою определенную функцию и ответственность. Иерархия облегчает управление системой, позволяет четко определить взаимодействие между компонентами и упростить анализ и модификацию системы.

3) *Разделение ответственности*. Принцип разделения ответственности предполагает, что каждый компонент системы должен быть ответственен только за выполнение конкретной задачи или функции. Это упрощает разработку и сопровождение системы, позволяет легко масштабировать и модифицировать компоненты, а также облегчает совместную работу разработчиков в рамках команды.
4) *Модульное взаимодействие*. Принцип модульного взаимодействия подразумевает, что компоненты системы должны взаимодействовать друг с другом через четко определенные интерфейсы. Модульное взаимодействие позволяет компонентам быть независимыми от конкретной реализации других компонентов, что обеспечивает гибкость системы и упрощает модификации и замены компонентов.
5) *Управление данными*. Принцип управления данными определяет, как система должна хранить, обрабатывать и обеспечивать доступ к данным. Включает выбор подходящих моделей данных, методов хранения и организации данных, а также обеспечение безопасности и конфиденциальности данных.

Эти принципы архитектуры информационных систем являются фундаментальными и помогают создавать эффективные и гибкие компьютерные системы.

1.4. Контрольные вопросы

Что представляет собой архитектура информационных систем?

Какое значение имеет архитектура информационных систем для разработки и эксплуатации компьютерных систем?

Какие основные принципы лежат в основе архитектуры информационных систем?

Что подразумевает принцип модульности в архитектуре информационных систем?

Что подразумевает принцип разделения ответственности в архитектуре информационных систем?

Каким образом модульное взаимодействие компонентов системы способствует гибкости и замене компонентов?

2. Архитектурные шаблоны информационных систем

Архитектурные шаблоны играют ключевую роль в разработке информационных систем, предоставляя общую структуру и организацию системы. Каждый архитектурный шаблон определяет основные принципы и концепции, которые помогают разработчикам создавать эффективные, масштабируемые и гибкие системы.

2.1. Клиент-серверная архитектура

Клиент-серверная архитектура является одним из наиболее распространенных и широко применяемых шаблонов архитектуры информационных систем. Этот шаблон разделяет систему на две основные компоненты: клиенты и серверы.

Клиенты – это пользовательские устройства или приложения, которые запрашивают данные или услуги у серверов. Они могут быть представлены в виде компьютеров, мобильных устройств или других систем, которые взаимодействуют с информационной системой.

Серверы – это специализированные компоненты, которые обрабатывают запросы от клиентов и предоставляют необходимые данные или услуги. Серверы могут быть физическими или виртуальными устройствами, работающими на серверных платформах.

Взаимодействие между клиентами и серверами осуществляется через сетевое соединение, обычно посредством протокола TCP/IP. Клиенты отправляют запросы на серверы, а серверы обрабатывают эти запросы и отвечают с необходимыми данными или результатами. (Рисунок 1)

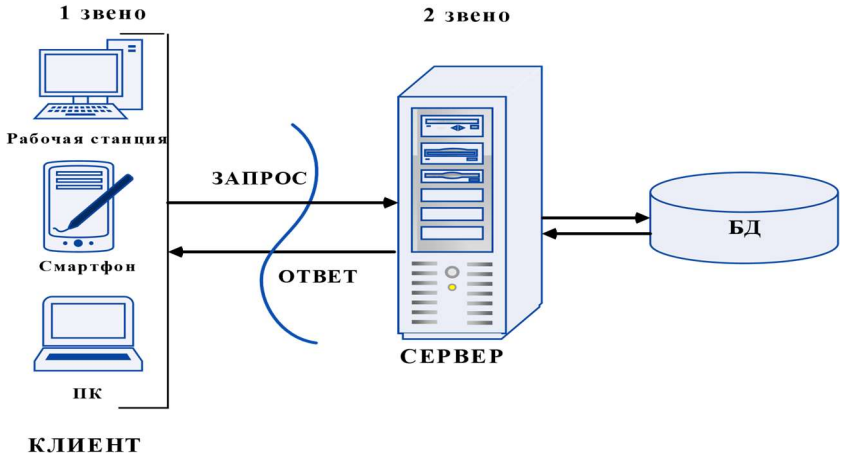


Рисунок 1 - Клиент-серверная архитектура

Характеристики архитектуры «клиент-сервер»:

1) *Асимметричность протоколов*. Между клиентами и сервером существуют отношения «один ко многим». Инициатором диалога с сервером обычно является клиент.
2) *Инкапсуляция услуг*. После получения запроса на услугу от клиента, сервер решает, как должна быть выполнена данная услуга. Модификация («апгрейд») сервера может производиться без влияния на работу клиентов, поскольку это не влияет на опубликованный интерфейс взаимодействия между ними. Иными словами, максимум, что может при этом почувствовать пользователь - незначительная задержка отклика сервера в течение небольшого времени апгрейда.
3) *Целостность*. Программы и общие данные для сервера управляются централизованно, что снижает стоимость обслуживания и защищает целостность данных. В то же время, данные клиентов остаются персонифицированными и независимыми.
4) *Местная прозрачность*. Сервер – это программный процесс, который может исполняться на той же машине, что и клиент, либо на другой машине, подключенной по сети. Программное обеспечение «клиент-сервер» обычно скрывает местоположение сервера от клиентов, перенаправляя запрос на услуги через сеть.
5) *Обмен на основе сообщений*. Клиенты и сервер являются не жёстко связанными («loosely–coupled») процессами, которые обмениваются сообщениями: запросами на услуги и ответами на них.
6) *Модульный дизайн, способный к расширению*. Модульный дизайн программной платформы «клиент-сервер» придаёт ей устойчивость к отказам, то есть, отказ в каком-то модуле не вызывает отказа всего приложения. В такой системе один или больше серверов могут отказать без остановки всей системы в целом, до тех пор, пока услуги отказавшего сервера могут быть предоставлены с резервного сервера. Другое преимущество модульности состоит в том, что приложение «клиент-сервер» может автоматически реагировать на повышение или понижение нагрузки на систему путем добавления или отключения услуг или серверов.
7) *Независимость от платформы*. Идеальное приложение «клиент-сервер» не зависит от платформ оборудования или операционной системы. Клиенты и серверы могут развертываться на различных аппаратных платформах и разных операционных системах.
8) *Масштабируемость*. Системы «клиент-сервер» могут масштабироваться как горизонтально (по числу серверов и клиентов), так и вертикально (по производительности и спектру услуг).

9) *Разделение функционала*. Система «клиент-сервер» – это соотношение между процессами, работающими на одной или на разных машинах. Сервер – это процесс предоставления услуг. Клиент – это потребитель услуг.
10) *Общее использование ресурсов*. Один сервер может предоставлять услуги множеству клиентов одновременно, и регулировать их доступ к совместно используемым ресурсам [3].

2.2. Многоуровневая архитектура

Многоуровневая архитектура, также известная как трехуровневая архитектура, является распространенным шаблоном архитектуры информационных систем. Она основана на идее разделения системы на несколько уровней функциональности, каждый из которых выполняет определенные задачи и обеспечивает определенные сервисы. (Рисунок 2)

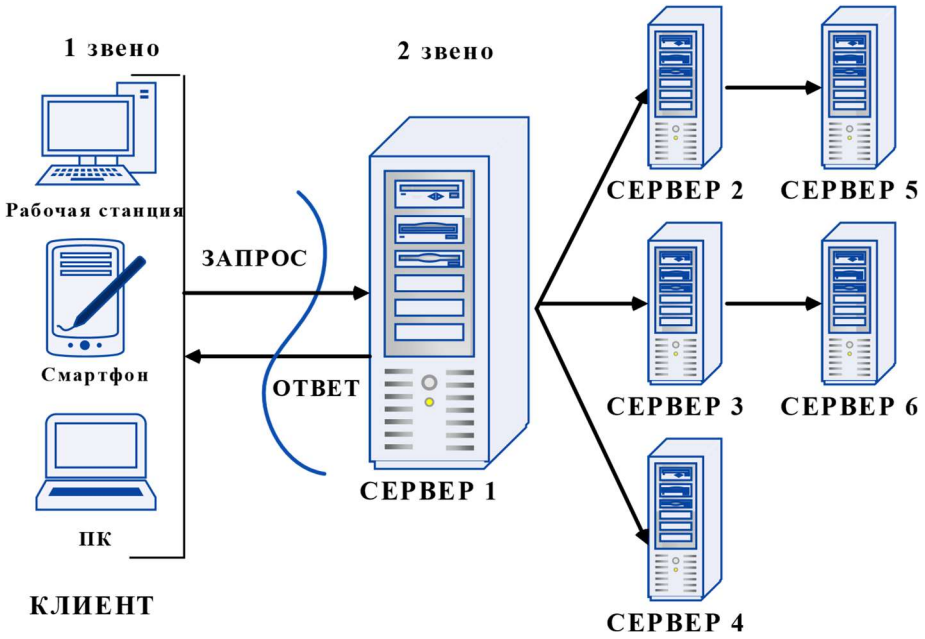


Рисунок 2 - Многоуровневая архитектура

<u>*В многоуровневой архитектуре обычно выделяют три основных уровня:*</u> 1) *Уровень представления (интерфейс пользователя).* Этот уровень отвечает за взаимодействие с пользователем и предоставляет пользовательский интерфейс для ввода и вывода данных. Он может включать в себя вебстраницы, мобильные приложения, графические интерфейсы и другие средства коммуникации с пользователем.

2) *Уровень бизнес-логики.* На этом уровне происходит обработка бизнес-логики и выполнение основных операций системы. Здесь располагаются компоненты, отвечающие за обработку данных, реализацию бизнес-правил и выполнение различных операций, связанных с логикой системы. Уровень

бизнес-логики может быть реализован в виде сервисов или классов, которые предоставляют необходимую функциональность.

3) *Уровень хранения данных*. Этот уровень отвечает за хранение и управление данными, используемыми системой. Здесь могут использоваться базы данных или другие механизмы хранения данных. Уровень хранения данных обеспечивает сохранность информации и обеспечивает доступ к ней со стороны других уровней системы.

Каждый уровень многоуровневой архитектуры имеет свои особенности и функции. Разделение системы на уровни позволяет достичь модульности, гибкости и переиспользуемости. Это означает, что изменения в одном уровне не должны влиять на другие уровни, что упрощает разработку, тестирование и сопровождение системы.

Преимущества многоуровневой архитектуры включают:

1) *Модульность*. Каждый уровень выполняет свои определенные функции, что облегчает понимание и модификацию системы. Модульность также способствует повторному использованию компонентов и упрощает тестирование и отладку.
2) *Гибкость*. Изменения в одном уровне не должны влиять на другие уровни, что позволяет легко вносить изменения в систему без необходимости перепроектирования всей архитектуры. Это делает систему гибкой и адаптивной к изменяющимся требованиям.
3) *Четкая организация и абстракция*. Многоуровневая архитектура позволяет разработчикам сосредоточиться на конкретных задачах каждого уровня, что улучшает структурированность и упрощает управление разработкой проекта. 4) *Повторное использование*. Компоненты на каждом уровне могут быть повторно использованы в различных частях системы или в других проектах. Это способствует ускорению разработки и улучшению производительности.
5) *Расширяемость*. Поскольку каждый уровень выполняет определенные функции, добавление нового функционала или изменение существующего функционала может быть относительно простым и не требует переработки всей системы.

Несмотря на множество преимуществ, многоуровневая архитектура может иметь некоторые ограничения. Например, увеличение количества уровней может привести к увеличению сложности системы и снижению производительности из-за увеличенной нагрузки на коммуникацию между уровнями.

2.3. Распределенная архитектура

Распределенная архитектура является одним из важных шаблонов архитектуры информационных систем. В этой модели система разделяется на физически отдельные компоненты, которые взаимодействуют и сотрудничают между собой через сеть.

В распределенной архитектуре каждый компонент системы может располагаться на отдельных устройствах, удаленных локациях или даже в различных облачных средах. Компоненты взаимодействуют друг с другом посредством сетевых вызовов, сообщений или использования распределенных протоколов. (Рисунок 3)

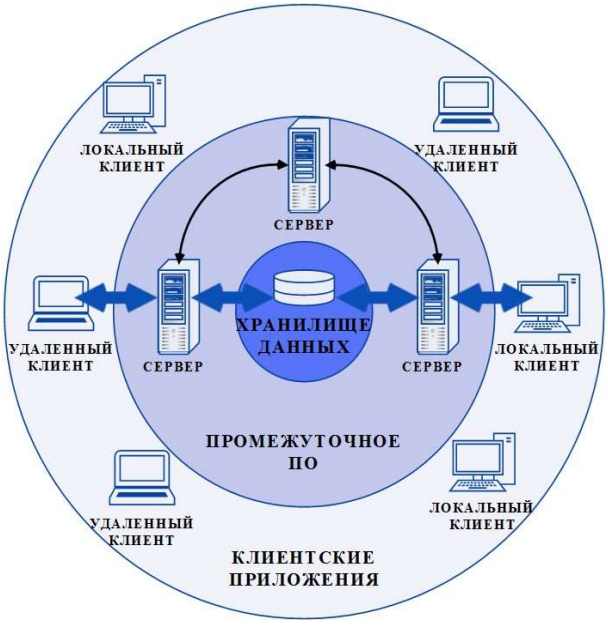


Рисунок 3 – Распределенная архитектура

<u>*Преимущества распределенной архитектуры включают:*</u>

1) *Масштабируемость*. Распределенная архитектура позволяет легко масштабировать систему путем добавления новых компонентов или увеличения ресурсов на существующих компонентах. Это особенно важно при обработке больших объемов данных или при высокой нагрузке на систему.
2) *Отказоустойчивость*. Благодаря физическому разделению компонентов на разные устройства и локации, распределенная архитектура обеспечивает высокую отказоустойчивость. Если один компонент выходит из строя, остальные компоненты могут продолжать работу, минимизируя простои и обеспечивая непрерывность работы системы.

3) *Гибкость*. Распределенная архитектура позволяет легко интегрировать новые компоненты в систему и изменять конфигурацию системы без значительных нарушений работы. Это упрощает внесение изменений в систему и адаптацию к новым требованиям.
4) *Локализация данных*. В распределенных системах данные могут храниться на ближайших узлах или узлах с наилучшим доступом к ним. Это позволяет улучшить производительность и сократить время доступа к данным.

Однако распределенная архитектура также имеет ограничения. Некоторые из них включают:

1) *Сложность разработки*. Разработка распределенных систем требует более сложного управления коммуникацией и синхронизацией между компонентами. Необходимо обеспечивать надежную передачу данных и обработку ошибок в сети.
2) *Зависимость от сети*. Распределенная архитектура сильно зависит от работоспособности и производительности сети. Если сеть нестабильна или медленная, это может привести к задержкам в обмене данными и снижению производительности системы.
3) *Безопасность*. Распределенные системы требуют особого внимания к вопросам безопасности, так как данные могут передаваться по открытым сетям. Необходимо обеспечить защиту данных и аутентификацию между компонентами системы.

2.4. Сервисно-ориентированная архитектура (SOA)

Сервисно-ориентированная архитектура (Service-Oriented Architecture, SOA) – это шаблон архитектуры информационных систем, который ориентирован на создание распределенных приложений через организацию функциональности в виде сервисов.

В SOA система разделяется на набор сервисов, которые являются автономными, логически независимыми компонентами и предоставляют функциональность через интерфейсы. Каждый сервис представляет собой логическую единицу функциональности, которая может быть вызвана другими компонентами или приложениями через стандартизированные протоколы и сообщения.

Не существует четко определенных стандартных рекомендаций по реализации сервис-ориентированной архитектуры (SOA), однако есть некоторые *общие основные принципы*:

1) *Обеспечение совместимости*. Каждый сервис в SOA включает документы описания, которые определяют функциональность сервиса и связанные с ним

условия. Любая клиентская система может запустить сервис, независимо от базовой платформы или языка программирования. Например, бизнеспроцессы могут использовать сервисы, написанные как на C#, так и на Python. Поскольку нет прямых взаимодействий, изменения в одном сервисе не влияют на другие компоненты, использующие этот сервис.

2) *Слабая взаимозависимость*. Сервисы в SOA должны быть слабосвязанными, иметь как можно меньше зависимостей от внешних ресурсов, таких как модели данных или информационные системы. Они также должны быть статичными, не сохраняя никакой информации из прошлых сессий или транзакций. Таким образом, если вы измените сервис, это не окажет существенного влияния на клиентские приложения и другие сервисы, использующие этот сервис.
3) *Абстрагирование*. Клиенты или пользователи сервисов в SOA не обязаны знать логику кода сервиса или детали его реализации. Для них сервисы должны выглядеть как черный ящик. Клиенты получают необходимую информацию о том, что делает сервис и как его использовать, через контракты на обслуживание и другие документы с описанием сервиса.
4) *Степень детализации*. Сервисы в SOA должны иметь соответствующий размер и объем, в идеале упаковывая одну дискретную бизнес-функцию в один сервис. Разработчики могут использовать несколько сервисов, чтобы создать комбинированный сервис для выполнения сложных операций [4].

Сервис-ориентированная архитектура (SOA) имеет ряд *преимуществ* перед традиционными монолитными архитектурами, в которых все процессы выполняются как единое целое. Вот некоторые из преимуществ SOA:

1) *Сокращение времени выхода на рынок*. Разработчики повторно используют сервисы в различных бизнес-процессах для экономии времени и затрат. С помощью SOA они могут создавать приложения гораздо быстрее, чем с написанием кода и выполнением интеграций с нуля.
2) *Эффективное обслуживание*. Легче создавать, обновлять и отлаживать небольшие сервисы, чем большие блоки кода в монолитных приложениях. Модификация любого сервиса в SOA не влияет на общую функциональность бизнес-процесса.
3) *Улучшенная адаптивность*. SOA лучше адаптируется к технологическому прогрессу. Вы можете модернизировать свои приложения эффективно и без лишних затрат. Например, медицинские организации могут использовать функциональность старых систем электронных медицинских карт в более новых облачных приложениях [4].

Сервисно-ориентированная архитектура (Рисунок 4) также имеет свои недостатки. Некоторые из них включают сложность управления большим

количеством сервисов, сложность обеспечения безопасности и надежности системы, а также требования к проектированию и управлению соответствующей инфраструктурой.

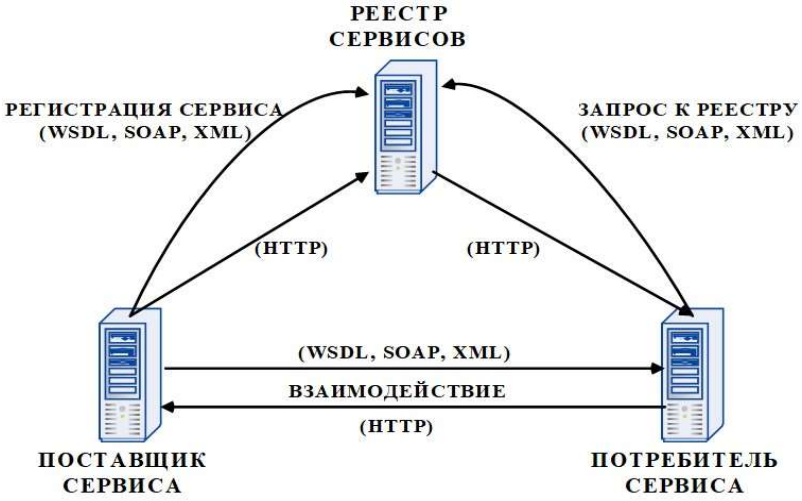


Рисунок 4 – Схема SOA

2.5. Микросервисная архитектура

Микросервисная архитектура является современным и популярным шаблоном архитектуры информационных систем. Она основана на идее разделения системы на набор маленьких, автономных и слабо связанных сервисов, каждый из которых выполняет конкретную функцию бизнес-логики. (Рисунок 5)

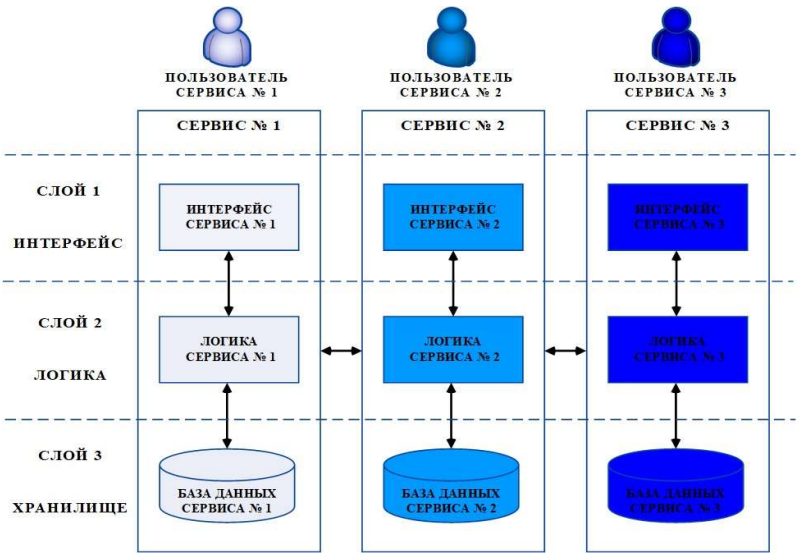


Рисунок 5 - Микросервисная архитектура

В микросервисной архитектуре каждый сервис представляет собой отдельное приложение или компонент, который может быть разработан, развернут и масштабирован независимо от других сервисов. Каждый сервис имеет свою собственную базу данных и может быть написан на различных технологиях и языках программирования в соответствии с его конкретными требованиями.

Основные принципы микросервисной архитектуры включают:

1) *Разделение по бизнес-функциональности*. Система разделяется на сервисы, которые отвечают за конкретные функции или задачи бизнес-логики. Каждый сервис имеет четко определенную область ответственности и предоставляет API для взаимодействия с другими сервисами.
2) *Слабая связанность*. Сервисы в микросервисной архитектуре должны быть слабо связаны друг с другом. Это означает, что изменения в одном сервисе не должны иметь существенного влияния на другие сервисы. Коммуникация между сервисами осуществляется через сетевые вызовы или использование сообщений.
3) *Автономность*. Каждый сервис является автономным и может быть развернут, масштабирован и обновлен независимо от других сервисов. Это позволяет командам разработчиков работать над своими сервисами независимо и ускоряет процесс разработки и внедрения изменений.
4) *Масштабируемость*. Микросервисная архитектура позволяет масштабировать отдельные сервисы по требованию. Если определенный сервис испытывает высокую нагрузку, его можно масштабировать, добавляя дополнительные экземпляры этого сервиса.

Преимущества микросервисной архитектуры:

1) *Гибкость и масштабируемость*. Микросервисная архитектура обеспечивает высокую гибкость при разработке и масштабируемость при внедрении системы. Каждый сервис может быть разработан, тестирован и масштабирован независимо, что упрощает добавление новых функций и адаптацию к изменениям требований.
2) *Независимость технологий и языков программирования*. Каждый сервис может быть написан на различных технологиях и языках программирования, в зависимости от его специфических требований. Это позволяет выбирать наиболее подходящие инструменты и технологии для каждого сервиса.
3) *Легкое масштабирование и обновление*. Благодаря независимости каждого сервиса, их можно масштабировать или обновлять независимо друг от друга.

   Это упрощает управление ресурсами и позволяет системе быть гибкой и эффективной.

4) *Легкость разработки и поддержки*. Разделение системы на маленькие сервисы упрощает разработку и поддержку кода. Команды разработчиков могут работать над отдельными сервисами, что улучшает эффективность работы и уменьшает взаимозависимость между командами.

Однако у микросервисной архитектуры есть и *недостатки:*

1) *Сетевая сложность*. Обмен данными между сервисами осуществляется через сеть, что может привести к увеличению задержек и сложностей сетевой коммуникации.
2) *Управление распределенной системой*. Управление и мониторинг распределенной системы может быть сложным и требовать специальных инструментов и практик.
3) *Повышенные требования к тестированию*. Тестирование микросервисов требует учета и проверки взаимодействия между сервисами, что может увеличить сложность тестирования.

2.6. Контрольные вопросы

Что такое клиент-серверная архитектура и какие основные компоненты она включает?

Какие характеристики архитектуры "клиент-сервер"?

Что такое многоуровневая архитектура и какие основные уровни выделяются в ней?

Какие преимущества имеет многоуровневая архитектура?

Что такое распределенная архитектура и какие преимущества она предоставляет?

Что такое сервисно-ориентированная архитектура (SOA) и какие принципы ей присущи?

Какие преимущества имеет сервисно-ориентированная архитектура?

Какие ограничения могут быть связаны с применением сервисноориентированной архитектуры?

Что такое микросервисная архитектура и какие принципы лежат в ее основе?

3. Паттерны и фреймворки в архитектуре ИС

Архитектурные паттерны и фреймворки являются важными инструментами в области разработки информационных систем (ИС). Они представляют собой набор проверенных и повторяемых решений, которые помогают разработчикам создавать эффективные, гибкие и масштабируемые архитектуры.

3.1. Паттерны

Паттерны представляют собой абстрактные модели или шаблоны, которые описывают типичные проблемы и способы их решения в проектировании ИС. Они помогают разработчикам создавать архитектуры, которые легко понять, сопровождать и модифицировать. Паттерны учитывают лучшие практики и опыт сообщества разработчиков, что позволяет избегать распространенных ошибок и повышать качество разработки.

Основное отличие паттернов от компонентов состоит в том, что компонент является модулем, который после настройки можно включить в состав системы, т.е. его можно рассматривать как готовый к употреблению строительный блок, а паттерн – это только заготовка, которую еще надо обработать, т.е. добавить код, определяющий функциональность.

Существует множество различных паттернов, каждый из которых решает конкретную проблему в архитектуре ИС. Некоторые из наиболее распространенных паттернов включают:

1) *Концептуальные паттерны* – это паттерны, функционирование которых описывается в терминах предметной области. Такие паттерны относятся к приложению в целом или крупным подсистемам ИС.
2) *Паттерны проектирования* – это паттерны, для описания которых используются термины, относящиеся к разработке программных систем, такие как объект, класс, модуль. Паттерны проектирования описывают решение общих проблем в конкретном контексте.
3) *Программные паттерны* – это паттерны, для описания которых используются низкоуровневые понятия: деревья, списки и т. п. [5].
4) *Архитектурный паттерн* описывает структуру программной системы и определяет состав подсистем, их основные функции и допустимые способы компоновки подсистем. Архитектурные паттерны называют также архитектурными стилями [5].
5) *Системные паттерны* представляют собой приложение на верхнем (системном) уровне. Системные паттерны можно рассматривать как паттерны, использование которых позволяет получить улучшенные архитектурные решения [5].
   a) *Модель-Вид-Контроллер*. Разделение приложения или подсистемы на три функциональные(логические) части (модель данных, представление (пользовательский интерфейс) и контроллер (управляющую логику)). Поскольку указанные части относительно независимы, облегчается разработка, модификация и настройка каждой части по отдельности. MVC содержится во многих фреймворках для веб-программирования.

Примерами удачного применения этого паттерна для языка PHP являются Zend Framework и CakePHP

b) *Сессия, Рабочая нить*. В системах распределённой обработки обеспечивает возможность серверам различать клиентов, что позволяет приложениям ассоциировать определенные состояния с клиентсерверными коммуникациями и повысить эффективность многопоточных приложений

c) *Обратный вызов*. Организация асинхронного взаимодействия между клиентом и сервером, что снижает загрузку на сеть, повышает эффективность использовании процессорного времени как клиента, так и сервера. Осуществляется следующим образом:

- регистрация клиента: клиент отправляет запрос серверу, предоставляя ему свою контактную информацию и разрывает соединение с сервером;
- обработка: сервер обрабатывает запрос и формирует ответ;
- обратный вызов: сервер, закончив обработку данных клиента, соединяется с клиентом и извещает его о результатах [6].

Данный принцип применяется для проектирования интернетмагазинов: с помощью асинхронного взаимодействия клиента и сервера в виде сообщений электронной почты полностью контролируется процесс покупки и доставки заказа [2].

d) *Текущее обновление*. Обеспечение возможности для клиента постоянного автоматического получения обновлений от сервера. Этот паттерн позволяет освободить пользователя от работы по обновлению. Обновления могут быть связаны с изменением данных на сервере, появлением ресурсов, обновлением ресурсов, изменениями в состоянии бизнес-модели. Примером применения паттерна является система почтовых рассылок новостей по темам, на которые подписался клиент [2].

e) *Транзакция*. Группирование коллекций методов таким образом, чтобы они либо были все успешно выполнены, либо все завершились неудачно [5].

6) *Структурные паттерны* с одинаковой эффективностью применяются как для разделения, так и для объединения элементов приложения [6].

a) *Адаптер (Adapter)*. Обеспечение взаимодействия компонентов с различными интерфейсами путем создания промежуточного объектаадаптера, преобразующего интерфейс одного из них таким

образом, чтобы им мог пользоваться другой компонент без изменения функциональности. Позволяет повторно использовать ранее созданный код.

b) *«Мост» (Bridge).* Разделение компонента на две взаимосвязанные, но независимые структуры: абстракцию и реализацию, каждая из которых является иерархической. Это позволяет повысить гибкость компонента за счёт возможности отдельного изменения (наследования) абстракции и реализации. "Полиморфизм" в классическом виде позволяет подменять наследника, но родитель у него один и тот же. Мост обеспечивает возможность "наследникам" менять "родителя", т.е. делает родителя полиморфным, позволяя менять родителя для наследников во время исполнения.

c) *Композит (Composite).* Предоставление гибкого способа создания объектов, которые могут быть представлены в виде иерархических древовидных структур произвольной сложности, и взаимодействия клиента с такими объектами на основе единого интерфейса. Упрощает работу клиента, облегчает добавление новых вариантов объектов и их элементов.

d) *«Декоратор» (Decorator).* Предоставление средства изменения функциональности компонентов без изменения их внешнего представления или функций

e) *«Фасад» (Facade).* Создание упрощенного интерфейса для сложной подсистемы путем сведения всех возможных внешних вызовов к одному объекту, направляющему их соответствующим объектам системы

f) *«Приспособленец» (Flyweight).* Уменьшение количества мелких объектов системы с большим числом низкоуровневых особенностей посредством совместного использования таких объектов

g) *Полуобъект и протокол (Half-Object Plus Protocol (HOPP)).* Разделение объекта, который должен быть доступен в более чем одном адресном пространстве, например, на клиента и на сервере, на два взаимозависимых полуобъекта (по одному в каждом адресном пространстве), взаимодействующих посредством собственного протокола. В каждом пространстве реализуется требуемая именно в нём функциональность. Протокол координирует активность и передаёт необходимую информацию между полу объектами. Используется, например, в CORBA, RMI

h) *Прокси (Proxy).* Предоставление объекта для обеспечения контроля доступа, безопасности, либо повышения скорости доступа к другому объекту [6].

7) *Поведенческие паттерны* используются в системе для передачи управления.
8) *Производящие паттерны* используются в системе для создания объектов.

Использование паттернов в архитектуре ИС позволяет создавать системы, которые легко понять, модифицировать и поддерживать. Они представляют собой ценный инструмент для разработчиков, позволяющий создавать эффективные и гибкие архитектуры, отвечающие требованиям бизнеса и пользователей.

3.2. Антипаттерны

Антипаттерны представляют собой типичные ошибки, проблемы и негативные практики, которые могут возникать при разработке информационных систем. Они противоположны паттернам и могут приводить к нежелательным результатам, таким как низкая производительность, сложность поддержки, непредсказуемость и плохое качество системы.

Понимание антипаттернов помогает разработчикам избегать распространенных ошибок и улучшать качество разрабатываемых систем. Они предоставляют уроки на основе опыта сообщества разработчиков, позволяющие избежать негативных последствий и создавать более эффективные и устойчивые архитектуры.

«Антипаттерны в управлении разработкой ПО: 1) «Дым и зеркала». Иллюстрация вида еще не реализованной системы. Демонстрация финального проекта и его функционала. Название происходит

от двух излюбленных способов, которыми фокусники скрывают свои секреты.

2) «Раздувание» ПО. Разрешение последующим версиям системы требовать все больше и больше ресурсов

3) «Функции для галочки». Превращение программы в конгломерат плохо реализованных и не связанных между собой функций (как правило, для того чтобы заявить в рекламе, что функция есть) *Антипаттерны в разработке ПО:*

1) Неопределенная точка зрения. Представления модели без спецификации ее точки рассмотрения;

2) «Большой комок грязи». Система с нераспознаваемой структурой;

3) «Бензиновая фабрика». Необоснованно усложненный дизайн;

4) «Затычка на ввод данных». Неопределенность в спецификации и выполнении поддержки возможного неверного ввода;

5) *«Раздувание интерфейса».* Чрезмерно развитый и сложный интерфейс, трудный в реализации;
6) *«Магическая кнопка».* Выполнение результатов действий пользователя в виде неподходящего (недостаточно абстрактного) интерфейса. Внедрение логики работы программы в пользовательский интерфейс, зачастую в программах, созданных в средах виртуальной разработки;
7) *«Перестыковка».* Процесс внедрения ненужной зависимости;
8) *«Дымоход».* Редко поддерживаемая сборка плохо связанных компонентов. Структура, в которой потоки информации циркулируют преимущественно вверх-вниз, а не горизонтально, между плохо связанными компонентами;
9) *«Состояние гонки».* Непредусмотренная возможность наступления событий в последовательности, отличной от предполагаемой (в многопоточных приложениях).

Антипаттерны в объектно-ориентированном программировании:

1) *Базовый класс-утилита.* Наследование функциональности из класса-утилиты вместо делегирования к нему;
2) *Вызов предка.* Для реализации прикладной функциональности методу классапотомка требуется в обязательном порядке вызвать те же методы классапредка;
3) *«Божественный» объект.* Концентрация слишком большого количества функций в одной части системы(класса);
4) *«Полтергейст».* Объекты, чье единственное предназначение – передавать информацию другим объектам;
5) *«Проблема йо-йо».* Чрезмерная размытость сильно связанного кода (например, выполняемого по порядку) по иерархии классов;
6) *Синглетонизм.* Избыточное использование паттерна Одиночка.

Антипаттерны в области программирования:

1) *Ненужная сложность.* Внесение ненужной сложности в решение;
2) *«Действие на расстоянии».* Неожиданное взаимодействие между широко разделенными частями системы;
3) *«Накопить и запустить».* Установка параметров программы в наборе глобальных переменных;
4) *«Лодочный якорь».* Сохранение неиспользуемой части системы;
5) *Активное ожидание.* Потребление ресурсов центрального процессора во время ожидания события, обычно при помощи постоянно повторяемой проверки, вместо того чтобы использовать систему сообщений;
6) *Кэширование ошибки.* Забывать сбросить флаг ошибки после ее обработки;

7) *Инерция кода.* Сверхограничение части системы путем постоянного подразумевания ее поведения в других частях системы. Кодирование путем исключения: добавление нового кода для поддержки каждого специального распознанного случая;
8) *«Таинственный» код.* Использование аббревиатур вместо мнемоничных имен;
9) *Жесткое кодирование.* Внедрение предположений об окружении системы в слишком большом количестве точек ее реализации;
10) *Мягкое кодирование.* Патологическая боязнь жесткого кодирования, приводящая к тому, что настраивается все что угодно, при этом конфигурирование системы само по себе превращается в программирование;
11) *«Поток лавы».* Сохранение нежелательного (излишнего или низкокачественного) кода по причине того, что его удаление слишком дорого или будет иметь непредсказуемые последствия;
12) *«Магические» числа.* Включение в алгоритмы чисел без объяснений их смысла;
13) *Процедурный код.* Когда другая парадигма является более подходящей; *14) «Спагетти-код».* Код с чрезмерно запутанным порядком выполнения;

15) «Мыльный пузырь». Класс, содержащий никогда не используемые данные [5].

Понимание антипаттернов помогает разработчикам избегать распространенных ошибок и создавать более эффективные и качественные архитектуры. Изучение антипаттернов помогает осознанно избегать их применения и выбирать более подходящие и эффективные решения.

3.3. Фреймворки

Фреймворки представляют собой набор инструментов, библиотек, правил и структур, которые облегчают разработку информационных систем. Они предоставляют готовые компоненты и абстракции, которые позволяют разработчикам сосредоточиться на конкретных задачах, вместо того чтобы создавать все с нуля.

Использование фреймворков в архитектуре информационных систем позволяет существенно ускорить процесс разработки и повысить качество создаваемых систем. Фреймворки предоставляют стандартные практики и рекомендации, а также облегчают решение типичных задач, таких как обработка запросов, маршрутизация, управление базами данных и пользовательским интерфейсом.

Классификация фреймворков:

1) *Архитектурные фреймворки* – это совокупность соглашений, принципов и практик, используемых для описаний архитектур и принятых применительно к некоторому предметному домену и (или) в сообществе специалистов (заинтересованных лиц) [7].
2) *Фреймворки уровня промежуточного ПО* используются для интеграции распределенных приложений и компонентов.
3) *Фреймворки, ориентированные на приложения*, используются, в первую очередь, для поддержания процесса разработки систем, ориентированных на конечного пользователя и принадлежащих некоторому конкретному предметному домену [8].
4) Использование *инфраструктурных фреймворков* упрощает разработку инфраструктурных элементов, таких как, например операционные системы. Обычно такие фреймворки используются внутри организации и не поступают в продажу.
5) *Фреймворки, используемые по принципу белого ящика,* называют также архитектурными фреймворками. Фреймворк, работающий по принципу белого ящика, определяется через интерфейсы объектов, которые разработчик может добавлять в систему.
6) *Фреймворки, используемые по принципу черного ящика*, называют также фреймворками, управляемыми данными. В качестве основных механизмов формирования приложения выступают композиция компонентов и параметризация.
7) *Фреймворки уровня приложения* обеспечивают полный набор функций, которые реализуются типовыми приложениями. Обычно сюда входят GUI, базы данных, документация. Примером таких фреймворков могут быть MFC (Microsoft Foundation Classes), которые служат для создания приложений, ориентированных на работу в среде MS Windows.
8) *Фреймворки уровня домена* используются для создания приложений, относящихся к определенному предметному домену [8].

<u>*Основные преимущества использования фреймворков включают:*</u>

1) *Ускорение разработки*. Фреймворки предлагают готовые компоненты и решения, что сокращает время, затрачиваемое на написание кода с нуля. Разработчики могут использовать функциональность, предоставляемую фреймворком, и сосредоточиться на уникальных аспектах своей системы.
2) *Согласованность и стандартизация*. Фреймворки определяют правила и структуру разработки, что обеспечивает согласованность в коде и архитектуре системы. Это позволяет разработчикам легко понимать и поддерживать код других членов команды или сторонних разработчиков.
3) *Расширяемость и модульность*. Фреймворки обычно построены с учетом расширяемости и модульности. Они позволяют добавлять новые функции и

компоненты без изменения основной структуры системы. Это упрощает поддержку и модификацию системы в будущем.

4) *Общепринятые решения*. Фреймворки часто предлагают решения, основанные на реализации общепринятых практик и опыте сообщества разработчиков. Это позволяет избежать распространенных ошибок и использовать проверенные решения для создания надежных и эффективных систем.
5) *Обновления и поддержка*. Фреймворки активно поддерживаются сообществом разработчиков и обновляются с учетом новых технологий и требований. Это обеспечивает доступность новых функций, исправление ошибок и обеспечение безопасности системы.

3.4. Примеры фреймворков

В архитектуре информационных систем существует множество фреймворков, которые облегчают разработку и ускоряют процесс создания систем. Ниже приведены некоторые примеры популярных фреймворков: 1) *Django* – это фреймворк (на языке Python) для разработки веб-приложений. Он предлагает мощные инструменты для управления базами данных, обработки HTTP-запросов, авторизации пользователей и многое другое.

Django также следует принципам шаблона проектирования MVC (ModelView-Controller). 2) *Ruby on Rails* – это фреймворк для разработки веб-приложений на языке Ruby. Он предлагает множество готовых решений, которые упрощают создание CRUD-операций, маршрутизацию, управление базами данных и другие задачи. Ruby on Rails также основан на шаблоне проектирования MVC.

3) *Angular* – это фреймворк для разработки клиентских веб-приложений на языке JavaScript. Он предоставляет мощные инструменты для создания динамических пользовательских интерфейсов, управления состоянием приложений, маршрутизации и других функций. Angular также основан на шаблоне проектирования MVC.
4) *Spring Framework* – это фреймворк для разработки приложений на языке Java. Он предлагает широкий набор функций, включая управление зависимостями, инверсию управления, обработку HTTP-запросов, управление транзакциями и другие. Spring Framework также поддерживает модульность и интеграцию с другими фреймворками и библиотеками.
5) *Laravel* – это фреймворк для разработки веб-приложений на языке PHP. Он предоставляет простой и элегантный синтаксис, а также множество готовых компонентов для управления маршрутизацией, базами данных, сессиями, аутентификацией и другими задачами.

6) *React* – это библиотека JavaScript для создания пользовательских интерфейсов. Хотя React не является полноценным фреймворком, он предлагает мощные инструменты для создания компонентов, управления состоянием и переиспользования кода.

3.5. Контрольные вопросы

Что представляют собой паттерны в контексте архитектуры информационных систем?

Какие виды паттернов существуют в архитектуре информационных систем?

Что такое антипаттерны и как они отличаются от паттернов в архитектуре информационных систем?

Перечислите антипаттерны в объектно-ориентированном программировании.

Что представляют собой фреймворки?

Какие типы фреймворков существуют в архитектуре информационных систем?

Приведите примеры популярных фреймворков.

4. Уровни архитектуры ИС

Архитектура информационных систем включает несколько уровней, каждый из которых имеет свои особенности и задачи. Каждый уровень описывает определенные аспекты исходя из потребностей бизнеса, функциональных требований, технических возможностей и процессов доставки продукта.

4.1. Архитектура бизнес-процессов

Архитектура бизнес-процессов – это описание структуры и организации бизнес-процессов внутри организации. Она включает в себя определение последовательности шагов и взаимосвязей между ними, а также ресурсы, необходимые для выполнения каждого шага. Архитектура бизнес-процессов позволяет организации лучше понять, как работают ее процессы, и оптимизировать их для достижения стратегических целей.

Основные компоненты архитектуры бизнес-процессов включают:

1) *Процессы*: описание последовательности шагов и операций, необходимых для достижения конкретной цели. Процессы могут быть высокоуровневыми, охватывающими весь цикл работы, или детализированными, фокусирующимися на конкретных аспектах работы.

2) *Роли и ответственности*: определение различных ролей, участвующих в процессе, и их ответственностей. Роли могут включать менеджеров, исполнителей, проверяющих и других участников процесса.
3) *Входные и выходные данные*: идентификация данных, необходимых для запуска и завершения процесса. Входные данные могут быть в виде информации, документов, ресурсов или запросов, а выходные данные – результатом выполнения процесса.
4) *Системы и технологии*: определение систем и инструментов, которые используются для автоматизации и управления бизнес-процессами. Это может включать программное обеспечение, системы управления, системы отчетности и другие технологии.

Проектирование архитектуры бизнес-процессов включает в себя несколько этапов:

1) *Идентификация и анализ бизнес-процессов*: определение и описание существующих процессов в организации и анализ их эффективности и эффективности.
2) *Проектирование оптимальной структуры процессов*: определение и оптимизация последовательности шагов и взаимосвязей между ними для достижения максимальной производительности и качества.
3) *Интеграция систем и технологий*: выбор и настройка подходящих систем и инструментов для автоматизации и управления бизнес-процессами.
4) *Развертывание и контроль*: внедрение новой архитектуры бизнес-процессов и непрерывное отслеживание и контроль их работы для обеспечения соответствия целям организации.

Архитектура бизнес-процессов является важной составляющей успешного управления организацией. Она помогает выявить и устранить узкие места в работе, повысить эффективность и качество процессов, а также улучшить взаимодействие между различными структурами и подразделениями.

4.2. Системная архитектура

Системная архитектура – это описание структуры и организации компьютерной системы или информационной системы (ИС) на уровне компонентов, модулей и их взаимодействия. Она определяет, как различные компоненты системы взаимодействуют друг с другом для достижения поставленных целей.

Основные цели системной архитектуры включают:

1) *Разделение ответственности*. Системная архитектура помогает определить, какие компоненты системы выполняют какие функции и как они

взаимодействуют друг с другом. Это позволяет четко определить ответственность каждого компонента и обеспечить их взаимодействие без пересечения функциональности.

2) *Модульность и повторное использование*. Системная архитектура способствует разделению системы на модули или компоненты, которые могут быть независимо разработаны, тестированы и поддерживаться. Это позволяет повторно использовать компоненты в разных проектах или контекстах, что ускоряет процесс разработки и повышает эффективность использования ресурсов.
3) *Гибкость и расширяемость*. Системная архитектура должна быть гибкой и способной адаптироваться к изменениям внешних и внутренних требований. Она должна обеспечивать возможность легкого добавления, удаления или модификации компонентов системы без значительного влияния на работу остальных компонентов.
4) *Эффективное использование ресурсов*. Системная архитектура позволяет оптимизировать использование ресурсов, таких как процессорное время, память и сетевая пропускная способность. Она должна обеспечивать эффективное распределение и управление ресурсами для достижения высокой производительности и удовлетворения требований системы.

В процессе проектирования системной архитектуры учитываются следующие аспекты:

1) *Компоненты системы*: определение основных компонентов, из которых состоит система, и их функциональность. Это может включать модули, подсистемы, службы и другие элементы, взаимодействующие между собой.
2) *Взаимодействие компонентов*: определение способов коммуникации и взаимодействия между компонентами системы. Это может быть осуществлено через интерфейсы, протоколы обмена данными или другие механизмы коммуникации.
3) *Распределение и размещение*: определение распределения компонентов системы по физическим узлам или серверам. Это может включать решения о размещении компонентов на одном сервере или их распределении по нескольким серверам для обеспечения масштабируемости и отказоустойчивости.
4) *Архитектурные шаблоны*: использование известных архитектурных шаблонов, таких как клиент-серверная архитектура, многоуровневая архитектура или микросервисная архитектура, для обеспечения эффективности и стабильности системы.

4.3. Техническая архитектура

Техническая архитектура – это уровень архитектуры информационной системы (ИС), который описывает структуру и организацию ее технических компонентов, таких как аппаратное обеспечение, программное обеспечение, сети и инфраструктура.

Основные компоненты технической архитектуры включают:

1) *Аппаратное обеспечение*: определение физических компонентов, таких как серверы, рабочие станции, хранение данных и сетевое оборудование. Это включает выбор и конфигурацию аппаратного обеспечения в соответствии с требованиями системы.
2) *Программное обеспечение*: определение программных компонентов, таких как операционные системы, базы данных, приложения и сервисы. Это включает выбор подходящих программных решений и их интеграцию в систему.
3) *Сети и коммуникации*: определение сетевой инфраструктуры, протоколов связи и механизмов коммуникации между компонентами системы. Это включает проектирование сетевой топологии, настройку сетевого оборудования и обеспечение безопасного обмена данными.
4) *Инфраструктура*: определение технической инфраструктуры, такой как системы управления и мониторинга, резервное копирование и восстановление данных, а также системы безопасности. Это включает разработку планов обслуживания системы, обеспечение доступности и защиты данных.

В процессе проектирования технической архитектуры должны учитываться: 1) *Системная архитектура*. Техническая архитектура должна соответствовать системной архитектуре, определенной на уровне компонентов и их взаимодействия. Это включает учет требований к производительности, масштабируемости, отказоустойчивости и безопасности системы [5].

2) *Интеграция*. Техническая архитектура должна обеспечивать интеграцию различных компонентов системы, чтобы они могли взаимодействовать между собой и обмениваться данными. Это может включать разработку интерфейсов, стандартов обмена данных и протоколов связи.
3) *Производительность*. Техническая архитектура должна быть спроектирована таким образом, чтобы обеспечивать высокую производительность системы. Это включает выбор оптимальных аппаратных и программных решений, оптимизацию сетевых и коммуникационных процессов, а также управление ресурсами системы.
4) *Безопасность*. Техническая архитектура должна обеспечивать защиту системы от внешних угроз и несанкционированного доступа. Это включает

внедрение механизмов аутентификации, авторизации, шифрования и контроля доступа.

Техническая архитектура играет важную роль в обеспечении стабильности, производительности и безопасности информационной системы. Ее правильное проектирование и реализация позволяют эффективно использовать ресурсы и обеспечить надежную работу системы.

4.4. Архитектура доставки продукта

Архитектура доставки продукта является важным аспектом архитектуры информационной системы (ИС) и описывает способ доставки и развертывания готового продукта или решения для его использования конечными пользователями.

Основные компоненты архитектуры доставки продукта включают:

1) *Пакетирование и сборка*. Этот компонент отвечает за создание пакета, содержащего все необходимые файлы и компоненты для развертывания продукта. Включает в себя процесс сборки и сжатия файлов, настройку зависимостей и подготовку к развертыванию.
2) *Развертывание*. Этот компонент отвечает за размещение пакета продукта на целевой среде или инфраструктуре. Включает в себя установку и конфигурирование необходимых компонентов, создание баз данных, запуск сервисов и настройку параметров системы.
3) *Миграция данных*. В случае необходимости переноса данных из предыдущей системы в новую этот компонент отвечает за перенос данных и их согласование с новой архитектурой. Включает в себя процесс извлечения данных из исходной системы, их преобразование и загрузку в целевую систему.
4) *Тестирование и верификация*. Этот компонент отвечает за проверку корректности развертывания продукта и его работоспособности в реальной среде. Включает в себя проведение функционального тестирования, нагрузочного тестирования, проверку безопасности и проверку соответствия требованиям.
5) *Мониторинг и обновление*. Этот компонент отвечает за непрерывное мониторинг и управление развернутым продуктом. Включает в себя сбор и анализ данных о производительности, доступности и безопасности системы, а также применение обновлений и патчей при необходимости.

4.5. Контрольные вопросы

Что представляет собой архитектура бизнес-процессов в информационной системе?

Какие компоненты входят в архитектуру бизнес-процессов?

Что представляет собой системная архитектура в информационной системе?

Какие основные цели системной архитектуры?

Какие компоненты входят в техническую архитектуру?

Что включает в себя архитектура доставки продукта в информационной системе?

5. Разновидности архитектур ИС

Разновидности архитектур информационных систем являются важным аспектом их проектирования и разработки. Архитектура определяет структуру и организацию системы, включая ее компоненты, взаимодействие между ними и способ достижения заданных функциональных и нефункциональных требований. Понимание различных разновидностей архитектур ИС позволяет разработчикам и архитекторам выбирать наиболее подходящую архитектурную стратегию в зависимости от потребностей и целей конкретного проекта.

5.1. По характеру решаемых задач и функциональному значению

Архитектуры информационных систем могут быть классифицированы по характеру решаемых задач и их функциональному значению. В зависимости от типа задач и функциональности, архитектуры ИС могут различаться в организации компонентов и взаимодействии между ними. Рассмотрим некоторые разновидности архитектур ИС, основанные на характере решаемых задач и функциональном значении:

1) *Архитектура транзакционных систем.* Основное предназначение этой архитектуры – обработка транзакций и поддержка операций в режиме реального времени. Транзакционные системы обеспечивают сохранность данных и согласованность выполнения операций, поддерживая надежность и целостность системы. Примеры: банковские системы, системы управления складом и системы онлайн-платежей.
2) *Архитектура управленческих систем.* Эта архитектура ориентирована на поддержку принятия решений на уровне управления и анализа данных.

Управленческие системы обеспечивают сбор, анализ и представление данных для принятия стратегических и тактических решений. Примеры: системы управления отношениями с клиентами (CRM), системы управления ресурсами предприятия (ERP).

3) *Архитектура систем поддержки принятия решений.* Эта архитектура предназначена для обработки и анализа больших объемов данных с целью выявления паттернов и трендов, что помогает в принятии решений. Системы поддержки принятия решений обычно используют методы статистики, машинного обучения и интеллектуального анализа данных. Примеры*:* системы бизнес-аналитики, системы прогнозирования и рекомендательные системы.
4) *Архитектура систем обработки информации*. Эта архитектура ориентирована на обработку, хранение и представление информации различных типов. Системы обработки информации обеспечивают эффективную работу с данными, их поиск, индексацию и управление метаданными. Примеры*:* системы управления базами данных (СУБД), системы электронного документооборота.

5.2. По предметной области, степени автоматизации, масштабности применения

Архитектуры ИС по предметной области:

1) *Финансы и банковское дело*. Архитектуры, специализированные для финансовых и банковских организаций. Эти системы обычно включают функциональность для управления счетами, транзакциями, инвестициями и другими финансовыми операциями.
2) *Здравоохранение*. Архитектуры, разработанные для медицинских учреждений и организаций. Эти системы обычно включают функциональность для хранения и обработки медицинских данных, планирования приемов пациентов, управления лекарственными препаратами и т.д.
3) *Транспорт и логистика*. Архитектуры, применяемые в транспортных и логистических компаниях. Эти системы обеспечивают управление грузами, маршрутизацию, отслеживание грузов, планирование доставки и другие связанные операции.
4) *Розничная торговля и электронная коммерция*. Архитектуры, используемые в розничных магазинах и онлайн-платформах. Эти системы включают функциональность для управления складами, обработки заказов, платежных операций, управления инвентарем и других операций розничной торговли.
5) *Производство и промышленность*. Архитектуры, применяемые в производственных предприятиях и промышленных компаниях. Эти системы обеспечивают управление производственными процессами, контроль качества, планирование производства, учет материалов и другие операции, связанные с производством.

Архитектуры ИС по степени автоматизации:

1) *Ручная система*. Это система, в которой большая часть бизнес-процессов и операций выполняется вручную без значительной автоматизации. Пользователи вручную обрабатывают данные, выполняют расчеты и принимают решения.
2) *Полуавтоматизированная система*. В этом типе системы некоторые бизнеспроцессы и операции автоматизированы, но часть работы все еще выполняется вручную. Автоматизированные модули и функции выполняют определенные задачи, но пользователи вносят ручные корректировки и принимают решения.
3) *Частично автоматизированная система*. В этом случае большая часть бизнес-процессов и операций автоматизирована, но некоторые элементы все еще требуют ручного вмешательства. Система выполняет автоматическую обработку данных и реализует определенные правила и процессы, но пользователи могут участвовать в исключительных ситуациях или в процессах, требующих человеческого вмешательства.
4) *Полностью автоматизированная система*. В данном случае все бизнеспроцессы и операции полностью автоматизированы без необходимости ручного вмешательства. Система автоматически обрабатывает данные, выполняет расчеты, принимает решения и управляет бизнес-процессами без участия пользователя.

Архитектуры ИС по масштабности применения:

1) *Локальная система*. Это система, которая предназначена для использования внутри конкретной локальной организации или подразделения. Обычно такие системы охватывают ограниченное количество пользователей и решают задачи, связанные с управлением определенной функциональной области или процесса внутри организации.
2) *Региональная система*. В этом случае система применяется на региональном уровне, охватывая несколько организаций или подразделений в определенной географической области. Региональные системы могут предоставлять совместные ресурсы, обмен данными и координацию между различными организациями в рамках региона.
3) *Национальная система*. В данном случае система применяется на уровне всей нации, охватывая множество организаций и пользователей внутри страны.

   Национальные системы могут иметь широкий охват и выполнять задачи, связанные с государственным управлением, социальными службами или другими общенациональными функциями.

4) *Глобальная система*. Глобальные системы применяются на международном или межконтинентальном уровне, охватывая организации и пользователей по всему миру. Эти системы предоставляют глобальные ресурсы, обеспечивают глобальное взаимодействие и решают сложные задачи, требующие совместной работы и обмена данными между различными странами и культурами.

5.3. По архитектурным стилям, реализации модульности

Архитектуры ИС по архитектурным стилям:

1) *Клиент-серверная архитектура*. Этот стиль базируется на разделении системы на клиентскую и серверную части. Клиенты запрашивают данные или услуги у серверов, а серверы обрабатывают запросы и предоставляют результаты. Этот стиль обеспечивает распределение функциональности между клиентами и серверами, что позволяет достичь гибкости и масштабируемости.
2) *Многоуровневая архитектура*. В многоуровневой архитектуре система разделяется на несколько уровней, каждый из которых выполняет определенные задачи и обеспечивает определенные сервисы. Обычно, выделяют уровень представления, бизнес-логики и хранения данных. Этот стиль обеспечивает модульность, гибкость и повторное использование компонентов.
3) *Распределенная архитектура*. Распределенная архитектура основана на физическом разделении системы на отдельные компоненты, которые взаимодействуют и сотрудничают между собой через сеть. Этот стиль позволяет обеспечить масштабируемость, отказоустойчивость и гибкость взаимодействия между компонентами.
4) *Сервисно-ориентированная архитектура (SOA)*. В SOA система организуется вокруг набора сервисов, которые предоставляют функциональность и могут быть вызваны другими компонентами системы. Этот стиль обеспечивает легкую интеграцию, повторное использование сервисов и гибкость внесения изменений.
5) *Микросервисная архитектура*. Микросервисная архитектура основана на идее разделения системы на набор маленьких, автономных и слабо связанных сервисов. Каждый сервис отвечает за конкретную функцию бизнес-логики и может разрабатываться, развертываться и масштабироваться независимо. Этот стиль обеспечивает гибкость, масштабируемость и возможность быстрого развертывания новых функций.

Архитектуры ИС по реализации модульности:

1) *Монолитная архитектура*. В монолитной архитектуре вся система состоит из одного цельного блока. Все компоненты и функции системы находятся внутри этого блока, и изменение одной части может затронуть всю систему. Монолитная архитектура имеет простую структуру, но может быть сложной в поддержке и изменении.
2) *Компонентно-ориентированная архитектура (COA).* COA представляет систему в виде набора независимых компонентов, которые могут быть разработаны и собираться отдельно. Компоненты в COA являются самодостаточными и имеют четкие интерфейсы для взаимодействия с другими компонентами. COA облегчает повторное использование компонентов и позволяет более гибко изменять систему.
3) *Сервисно-ориентированная архитектура (SOA).* В SOA функциональность системы предоставляется в виде сервисов, которые могут быть независимо развернуты и вызваны другими компонентами. Сервисы в SOA обычно имеют четкие интерфейсы и выполняют определенные задачи. SOA способствует повторному использованию сервисов и обеспечивает гибкость в интеграции компонентов системы.
4) *Микросервисная архитектура*. Микросервисная архитектура основана на идее разделения системы на набор независимых и автономных сервисов. Каждый сервис выполняет конкретную функцию бизнес-логики и может быть разрабатываем, развертываться и масштабироваться независимо. Микросервисная архитектура обеспечивает высокую модульность и гибкость изменений в системе.

5.4. По архитектуре аппаратных средств

1) *Централизованная архитектура*. В централизованной архитектуре все вычислительные и обрабатывающие ресурсы сосредоточены в центральном узле или сервере. Клиентские устройства получают доступ к этим ресурсам через сетевое соединение. Централизованная архитектура обеспечивает централизованное управление и контроль, но может создавать единую точку отказа.
2) *Распределенная архитектура*. В распределенной архитектуре вычислительные ресурсы и функции системы распределены по различным узлам или серверам. Каждый узел выполняет определенные задачи и взаимодействует с другими узлами для обмена информацией. Распределенная архитектура обеспечивает масштабируемость, отказоустойчивость и более равномерное распределение нагрузки.
3) *Клиент-серверная архитектура*. Клиент-серверная архитектура базируется на разделении системы на клиентские устройства и серверы. Клиенты запрашивают данные или услуги у серверов, которые обрабатывают запросы

и предоставляют результаты. Клиент-серверная архитектура обеспечивает разделение ответственности, централизованное управление и возможность масштабирования.

4) *Кластерная архитектура*. В кластерной архитектуре несколько узлов объединяются в кластер, чтобы работать вместе и обеспечивать высокую доступность и отказоустойчивость. Каждый узел в кластере выполняет те же функции, и при сбое одного узла другие узлы могут продолжать работу. Кластерная архитектура позволяет достичь высокой производительности и надежности за счет распределения нагрузки.

5.5. Контрольные вопросы

Какие разновидности архитектур информационных систем можно выделить?

Перечислите архитектуры ИС по предметной области.

Что означает степень автоматизации в контексте архитектур ИС?

Что представляет собой модульность в архитектуре ИС?

Перечислите архитектуры ИС по степени автоматизации.

Перечислите архитектуры ИС по архитектурным стилям.

Перечислите архитектуры ИС по архитектуре аппаратных средств.

6. Безопасность и масштабируемость в архитектуре информационных систем

Архитектура информационных систем должна удовлетворять не только требованиям функциональности и производительности, но также обеспечивать безопасность и масштабируемость. Безопасность является одним из ключевых аспектов, поскольку ИС сталкиваются с растущими угрозами кибербезопасности. Масштабируемость, в свою очередь, позволяет адаптировать систему к растущим потребностям и объемам данных. В этом разделе мы рассмотрим принципы обеспечения безопасности информационных систем и архитектурные подходы к обеспечению масштабируемости.

6.1. Принципы обеспечения безопасности информационных систем

Безопасность информационных систем является сложной и многогранной проблемой. Она включает в себя защиту от несанкционированного доступа, обеспечение конфиденциальности, целостности и доступности данных, а также защиту от вредоносного программного обеспечения и атак со стороны злоумышленников. Некоторые из принципов обеспечения безопасности информационных систем включают:

1) *Принцип защиты по уровням (defense-in-depth).* Этот принцип предусматривает применение нескольких уровней защиты, чтобы предотвратить компрометацию системы в случае нарушения одного уровня. Каждый уровень может включать меры, такие как брандмауэры, антивирусные программы, системы обнаружения вторжений и т. д.
2) *Принцип наименьших привилегий (principle of least privilege).* Согласно этому принципу, пользователи и процессы должны иметь только необходимые привилегии для выполнения своих задач. Это помогает ограничить потенциальный ущерб в случае компрометации аккаунта или процесса.
3) *Принцип обеспечения конфиденциальности, целостности и доступности (CIA).* Этот принцип заключается в обеспечении конфиденциальности (защите данных от несанкционированного доступа), целостности (предотвращении несанкционированного изменения данных) и доступности (гарантированном доступе к данным и сервисам в нужное время).
4) *Принцип безопасности по умолчанию (security by default).* Системы должны быть настроены таким образом, чтобы предоставлять высокий уровень безопасности с самого начала. Значения по умолчанию должны быть безопасными, а доступ к системным ресурсам должен быть ограничен.

6.2. Архитектурные подходы к обеспечению масштабируемости

Масштабируемость играет важную роль в разработке информационных систем, поскольку позволяет системе гибко реагировать на рост объемов данных, пользовательской нагрузки и требований к производительности. Некоторые архитектурные подходы к обеспечению масштабируемости включают:

1) *Горизонтальное масштабирование*. Этот подход предусматривает распределение нагрузки путем добавления дополнительных экземпляров системы. Это может быть достигнуто с помощью использования кластеров, автоматического масштабирования или использования облачных ресурсов.
2) *Вертикальное масштабирование*. В этом подходе увеличивается производительность системы путем увеличения ресурсов, таких как процессоры, память или дисковое пространство. Однако этот подход имеет свои ограничения, связанные с физическими возможностями аппаратных средств.
3) *Распределенная архитектура*. Этот подход предусматривает разделение системы на отдельные компоненты, которые могут работать независимо друг от друга и масштабироваться отдельно. Распределенная архитектура позволяет распределять нагрузку и повышать производительность системы.
4) *Кеширование*. Использование кэшей для хранения часто запрашиваемых данных позволяет улучшить производительность системы и снизить нагрузку на базу данных или другие ресурсы.

6.3. Контрольные вопросы

Перечислите основные аспекты безопасности информационных систем.

Что означает принцип защиты по уровням и как он помогает обеспечить безопасность системы?

Что предусматривает принцип наименьших привилегий и как он способствует безопасности информационных систем?

Что такое масштабируемость в контексте информационных систем?

В чем заключается горизонтальное масштабирование и какие преимущества оно предоставляет?

Что такое кеширование и как оно способствует повышению производительности и снижению нагрузки на ресурсы системы?

7. Разработка архитектуры информационных систем

Разработка архитектуры информационных систем является одним из важных этапов в жизненном цикле создания и сопровождения систем. Архитектура определяет структуру, компоненты и связи между ними, а также способы реализации требуемой функциональности системы. Она играет ключевую роль в обеспечении эффективности, надежности, безопасности и масштабируемости информационных систем.

7.1. Жизненный цикл разработки архитектуры

Жизненный цикл разработки архитектуры представляет собой последовательность этапов и деятельностей, которые проходят архитекторы и разработчики для создания и сопровождения архитектуры информационных систем. Этот жизненный цикл охватывает все основные этапы, начиная с исследования требований и заканчивая оценкой и улучшением архитектуры.

Этапы жизненного цикла разработки архитектуры:

1) *Анализ требований*. На этом этапе архитекторы анализируют требования к системе, взаимодействуют с заказчиками и заинтересованными сторонами, чтобы понять бизнес-цели, функциональные требования и ограничения проекта.
2) *Проектирование.* На этом этапе архитекторы разрабатывают общую структуру системы, определяют компоненты, связи между ними и способы взаимодействия. Они учитывают требования к безопасности, масштабируемости, производительности и другим аспектам.
3) *Реализация*. На этом этапе архитекторы работают с разработчиками для реализации архитектуры, выбирают технологии, создают прототипы, разрабатывают компоненты и интегрируют их в систему.

4) *Тестирование.* На этом этапе архитекторы и тестировщики проверяют работу системы, проводят функциональное и нагрузочное тестирование, анализируют производительность и безопасность системы.
5) *Развертывание*. На этом этапе архитекторы помогают внедрить систему в рабочую среду, проводят настройку и конфигурацию, обеспечивают ее безопасность и готовность к эксплуатации.
6) *Поддержка*. После развертывания архитекторы оказывают поддержку системы, отвечают на запросы пользователей, обновляют архитектуру при необходимости и вносят изменения в соответствии с требованиями и обратной связью.
7) *Оценка и улучшение*. На этом этапе архитекторы проводят оценку архитектуры, анализируют ее соответствие требованиям, производят аудит безопасности и производительности, идентифицируют возможности улучшения и оптимизации архитектуры.

Жизненный цикл разработки архитектуры является гибким и может быть адаптирован в зависимости от конкретных потребностей и характеристик проекта.

7.2. Методы и инструменты разработки архитектуры информационных систем

Разработка архитектуры информационных систем требует использования различных методов и инструментов, которые помогают архитекторам и разработчикам создавать эффективные и надежные системы. Некоторые из ключевых методов и инструментов, используемых в разработке архитектуры информационных систем:

1) *Методологии разработки архитектуры*. Существуют различные методологии, которые предоставляют структурированный подход к разработке архитектуры информационных систем. Некоторые из популярных методологий включают в себя:
   a) *Методология TOGAF (The Open Group Architecture Framework)*. Это фреймворк, который предоставляет набор методов и инструментов для разработки, управления и оценки архитектуры.
   b) *Agile-методологии*. Agile-методологии, такие как Scrum и Kanban, акцентируются на гибкой и итеративной разработке, позволяя архитекторам быстро реагировать на изменения и обратную связь.
   c) *DevOps*. DevOps-подход объединяет разработку и операционную деятельность, что способствует более эффективной разработке и доставке архитектуры информационных систем.

2) *Моделирование и нотации*. Моделирование является важным инструментом для визуализации и описания архитектуры информационных систем. Некоторые из популярных нотаций и языков моделирования включают в себя:
   a) *UML (Unified Modeling Language)*. UML предоставляет набор нотаций для моделирования архитектуры, включая диаграммы классов, диаграммы компонентов, диаграммы последовательности и другие.
   b) *ArchiMate*. ArchiMate является специализированным языком моделирования для описания бизнес-архитектуры, информационной архитектуры и технической архитектуры системы.
   c) *BPMN (Business Process Model and Notation)*. BPMN используется для моделирования бизнес-процессов и их взаимодействия с информационными системами.
3) *Инструменты CASE (Computer-Aided Software Engineering)*. CASEинструменты предоставляют средства автоматизации и поддержки разработки архитектуры. Они позволяют архитекторам создавать, редактировать и визуализировать модели архитектуры, проводить анализ и симуляции системы. Некоторые из популярных CASE-инструментов включают в себя Enterprise Architect, Sparx Systems, IBM Rational Software Architect и др.
4) *Метрики и показатели (Metrics and Key Performance Indicators)*.
   Использование метрик и показателей позволяет оценить производительность, эффективность и качество архитектуры системы.

7.3. Проектирование и документирование архитектуры

Проектирование архитектуры информационных систем включает следующие шаги:

1) *Анализ требований*. Архитекторы должны внимательно изучить требования к системе, включая функциональные и нефункциональные требования. Это позволяет определить основные компоненты, модули и взаимодействия системы.
2) *Выбор архитектурных стилей и паттернов*. Архитекторы выбирают соответствующие архитектурные стили и паттерны, которые наилучшим образом соответствуют требованиям и целям системы. Например, можно выбрать клиент-серверную архитектуру, микросервисную архитектуру или шаблоны проектирования, такие как MVC или MVP.
3) *Проектирование компонентов и модулей*. Архитекторы определяют компоненты и модули системы, их функциональность и взаимодействия. Это включает определение интерфейсов, структуры базы данных, логику бизнеспроцессов и другие аспекты системы.

4) *Распределение функциональности*. Архитекторы определяют, какая функциональность будет реализована на стороне клиента, сервера или других компонентов системы. Это помогает более эффективно организовать работу системы и обеспечить оптимальное распределение нагрузки.
5) *Определение протоколов и стандартов*. Архитекторы выбирают соответствующие протоколы и стандарты для обеспечения взаимодействия между компонентами системы. Это включает выбор протоколов передачи данных, форматов обмена сообщениями и других соглашений.

После проектирования архитектуры необходимо правильно документировать ее. Документирование архитектуры позволяет сохранить и передать знания о системе, обеспечить ее понимание и согласованность среди разработчиков, а также облегчить поддержку и развитие системы в дальнейшем.

Документация архитектуры может включать:

1) Описания компонентов и модулей системы.
2) Диаграммы классов, компонентов, последовательности, взаимодействия и другие виды диаграмм.
3) Описание протоколов и форматов обмена данными.
4) Технические руководства и инструкции по развертыванию и настройке системы.
5) Описание архитектурных решений и принятых решений по безопасности, масштабируемости и другим аспектам.

Документация архитектуры должна быть доступной, понятной и актуальной. Она служит важным средством коммуникации между архитекторами, разработчиками, тестировщиками и другими участниками процесса разработки.

7.4. Оценка и улучшение архитектуры информационных систем

Оценка архитектуры информационной системы включает в себя следующие шаги:

1) *Анализ соответствия требованиям*. Оценка архитектуры начинается с анализа соответствия требованиям системы. Архитекторы и другие заинтересованные стороны анализируют, насколько архитектура отвечает функциональным и нефункциональным требованиям. Это позволяет выявить потенциальные проблемы и расхождения.
2) *Оценка качественных атрибутов*. Кроме соответствия требованиям, архитектура также должна быть оценена по различным качественным атрибутам, таким как производительность, надежность, безопасность и т.д. На

этом шаге проводятся анализ и тестирование системы для определения ее сильных и слабых сторон.

3) *Идентификация проблемных областей*. На основе результатов оценки выявляются проблемные области и узкие места в архитектуре. Это могут быть недостаточная производительность, сложность интеграции, уязвимости безопасности и другие проблемы. Идентификация этих проблем позволяет сосредоточить усилия на их устранении.
4) *Планирование улучшений*. После идентификации проблемных областей разрабатывается план улучшений. На этом шаге определяются конкретные действия, необходимые для устранения проблем и улучшения архитектуры. Это может включать изменения в компонентах, оптимизацию процессов, внедрение новых технологий и другие мероприятия.
5) *Реализация и проверка улучшений*. На последнем шаге происходит реализация запланированных улучшений и их проверка на практике. Внесенные изменения тестируются, а результаты оцениваются с точки зрения достигнутых улучшений. При необходимости могут быть внесены дополнительные корректировки.

Улучшение архитектуры информационной системы может осуществляться по следующим направлениям:

1) *Оптимизация производительности*. Включает в себя оптимизацию алгоритмов, использование кэширования, улучшение масштабируемости и другие меры, направленные на улучшение производительности системы.
2) *Обеспечение безопасности*. Включает в себя применение соответствующих мер по защите данных, предотвращению несанкционированного доступа и другим аспектам безопасности.
3) *Упрощение архитектуры*. Включает в себя устранение избыточности, упрощение связей между компонентами, улучшение модульности и другие меры, направленные на снижение сложности системы.
4) *Внедрение новых технологий*. Включает в себя использование современных технологий, инструментов и подходов, которые могут улучшить архитектуру системы и обеспечить ее современность и гибкость.
5) *Исправление выявленных проблем*. Включает в себя решение конкретных проблем, выявленных в процессе оценки архитектуры. Это могут быть проблемы производительности, безопасности, надежности или другие проблемы, требующие немедленного внимания.

Улучшение архитектуры информационных систем является непрерывным процессом, который должен осуществляться на протяжении жизненного цикла системы. Система должна быть готова к изменениям требований, технологий и

бизнес-потребностей, и архитектура должна быть способной адаптироваться и улучшаться для обеспечения эффективной работы системы.

7.5. Контрольные вопросы

Что такое жизненный цикл разработки архитектуры информационных систем?

Какие этапы включает жизненный цикл разработки архитектуры?

Какие методы можно использовать при разработке архитектуры информационных систем?

Какие нотации и модели применяются для визуализации архитектуры информационных систем?

Какие этапы требуется выполнить для проектирования архитектуры информационных систем?

Какие шаги включает в себя оценка архитектуры информационных систем?

8. Практическая часть

8.1. Разработка клиент-серверного приложения (сетевого чата)

Клиент-сервер – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер – это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде:

1) *данных* (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных);
2) *сервисных функций* (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр webстраниц во всемирной паутине) [3].

В данном практическом задании будет разработан сетевой чат. Вся разработка будет реализована на языке Delphi. Для разработки сетевого чата основными компонентами являются *ServerSocket* и *ClientSocket.* Они входят в стандартный пакет Delphi.

Прежде чем приступить к разработке любого программного обеспечения, требуется определить цель его создания и задачи, которые оно будет выполнять.

Цель и задачи определяют набор требуемых компонентов. В нашем случае целью является обеспечение двух пользователей практически мгновенной коммуникацией. А задачами являются реализации функций сервера и клиента для каждого из них. Основным параметром, который определяет установление связи между пользователями, является порт. Связь установится только при идентичном значении свойства *Port*. Для того чтобы иметь возможность оперативно модифицировать значение порта, добавим в форму компонент *Edit* и назовем его *PortEdit*. Соединение будет невозможным, если не установить имя сервера или его *IP*. Для этого также размещаем на форме компонент *Edit* с именем *HostEdit*. Для указания имени пользователя и добавления текста сообщения определим ещё два компонента *Edit* с именами *NikEdit* и *TextEdit* соответственно. Компонент *Memo*, который является простым текстовым редактором, будем использовать для отображения текста принимаемых и отправляемых сообщений, назовем его *ChatMemo*. Управление логикой работы обеспечивается кнопками *Button*:

• *ServerBtn* используется для создания/закрытия сервера, назовем её «Создать сервер»;

• *ClientBtn* используется для подключения/отключения клиента к серверу, назовем её «Подключиться»;

• *SendBtn* используется для отправки сообщений, назовем её «Отправить».

На рисунке 6 определено, что будет происходить при создании формы.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    // предложенное значения порта
    PortEdit.Text:='777';
    // адрес при проверке программы на одном ПК
    HostEdit.Text:='127.0.0.1';
    // остальные поля просто очистим
    NikEdit.Clear;
    TextEdit.Clear;
    ChatMemo.Lines.Clear;
end;
```

Рисунок 6 – Описание процедуры FormCreate

Пусть в начальном состоянии программы выбран режим сервера. Для последующих переводов программы в режим сервера будем использовать кнопку «Создать сервер» (*ServerBtn*). Для минимизации количества кнопок отключение режима сервера можно производить, используя свойство *Tag* кнопки *ServerBtn*. Изменение значения данного свойства позволит определить логику работы программы в соответствующий момент времени. На рисунке 7 представлена процедура нажатия кнопки *ServerBtnClick*.

Далее описываются события, которые должны происходить при определенном состоянии *ServerSocket*.

Рисунок 8 иллюстрирует процедуру присоединения клиента к серверу *ServerSocketClientConnect*, а на рисунке 8 представлена процедура отключения клиента *ServerSocketClientDisconnect*.

```
procedure TForm1.ServerBtnClick(Sender: TObject);
begin
If ServerBtn.Tag=0 then
begin
    // блокируем ClientBtn и поля HostEdit, PortEdit
    ClientBtn.Enabled:=False;
    HostEdit.Enabled:=False;
    PortEdit.Enabled:=False;
    // запись указанного порта в ServerSocket
    ServerSocket.Port:=StrToInt(PortEdit.Text);
    // запуск сервера
    ServerSocket.Active:=True;
    // добавим в ChatMemo сообщение с временем создания
    ChatMemo.Lines.Add('['+TimeToStr(Time)+'] Сервер создан');
    // изменяем тэг
    ServerBtn.Tag:=1;
    // меняем надпись клавиши
    ServerBtn.Caption:="Закрыть сервер";
end
else
    begin
    // разблокируем ClientBtn и поля HostEdit, PortEdit
    ClientBtn.Enabled:=True;
    HostEdit.Enabled:=True;
    PortEdit.Enabled:=True;
    // закроем сервер
    ServerSocket.Active:=False;
    // выводим сообщение в ChatMemo
    ChatMemo.Lines.Add("['+TimeToStr(Time)+'] Сервер закрыт.");
    // возвращаем тэгу исходное значение
    ServerBtn.Tag:=0;
    // возвращаем исходную надпись клавиши
    ServerBtn.Caption:="Создать сервер";
end;
end;
```

Рисунок 7 – Описание процедуры нажатия клавиши ServerBtnClick

```
procedure TForm1.ServerSocketClientConnect(Sender: TObject; Socket: TCustomWinSocket);
Begin
    // добавим в ChatMemo сообщение с временем подключения клиента
    ChatMemo.Lines.Add('['+TimeToStr(Time)+'] Подключился клиент.');
end;
```

Рисунок 8 – Описание подключения клиента ServerSocketClientConnect

```
procedure TForm1.ServerSocketClientDisconnect(Sender: TObject; Socket: TCustomWinSocket);
begin
    // добавим в ChatMemo сообщение с временем отключения клиента
    ChatMemo.Lines.Add('['+TimeToStr(Time)+'] Клиент отключился.');
end;
```

Рисунок 9 – Описание отключения клиента ServerSocketClientDisconnect

При получении сервером очередного сообщения от клиента он должен сразу же его отображать. На рисунке 10 представлена процедура чтения сообщения от клиента *ServerSocketClientRead*.

```
procedure TForm1.ServerSocketClientRead(Sender: TObject; Socket: TCustomWinSocket);
begin
    // добавим в ChatMemo клиентское сообщение
    ChatMemo.Lines.Add(Socket.ReceiveText());
end;
```

Рисунок 10 – Описание процедуры чтения сообщения ServerSocketClientRead

Очевидно, что основным функционалом является отправка сообщений. В реализуемом приложении отправка осуществляется при нажатии клавиши "Отправить" (*SendBtn*). При этом требуется проверять текущий режим программы, клиент это или сервер. Выбор режима демонстрируется в процедуре *SendBtnClick* на рисунке 11.

```
procedure TForm1.SendBtnClick(Sender: TObject);
begin
    // проверка, в каком режиме находится программа
    If ServerSocket.Active=True then
    // отправляем сообщение с сервера
        ServerSocket.Socket.Connections[0].SendText('['+TimeToStr(Time)+']' +NikEdit.Text+
        ': '+TextEdit.Text);
    else
    // отправляем сообщение с клиента
    ClientSocket.Socket.SendText('['+TimeToStr(Time)+'] '+NikEdit.Text+': '+TextEdit.Text);
    // отобразим сообщение в ChatMemo
    ChatMemo.Lines.Add('['+TimeToStr(Time)+'] '+NikEdit.Text+': '+TextEdit.Text);
end;
```

Рисунок 11 – Описание процедуры выбора режима SendBtnClick

Режим клиента действует наоборот. При нажатии клавиши "Подключиться" (*ClientBtn*) блокируется *ServerBtn* и активируется *ClientSocket*. Процедура *ClientBtnClick*, приведенная на рисунке 12, демонстрирует этот режим.

Далее определим процедуру чтения сообщения клиентом с сервера *ClientSocketRead* (рисунок 13). После этого остается оформить процедуры на добавление в *ChatMemo* определенного сообщения при установлении соединения с сервером *ClientSocketConnect* (рисунок 14) и потере связи *ClientSocketDisconnect* (рисунок 15).

```
procedure TForm1.ClientBtnClick(Sender: TObject);
Begin
If ClientBtn.Tag=0 then
    begin
        // клавишу ServerBtn и поля HostEdit, PortEdit заблокируем
        ServerBtn.Enabled:=False;
        HostEdit.Enabled:=False;
        PortEdit.Enabled:=False;
        // запишем указанный порт в ClientSocket
        ClientSocket.Port:=StrToInt(PortEdit.Text);
        // запишем хост и адрес (одно значение HostEdit в оба)
        ClientSocket.Host:=HostEdit.Text;
        ClientSocket.Address:=HostEdit.Text;
        // запустим клиента
        ClientSocket.Active:=True;
        // изменим тэг
        ClientBtn.Tag:=1;
        // изменим надпись клавиши
        ClientBtn.Caption:='Отключиться';
    end
    else begin
        // клавишу ServerBtn и поля HostEdit, PortEdit разблокируем
        ServerBtn.Enabled:=True;
        HostEdit.Enabled:=True;
        PortEdit.Enabled:=True;
        // закроем клиента
        ClientSocket.Active:=False;
        // выведем сообщение в ChatMemo
        ChatMemo.Lines.Add('['+TimeToStr(Time)+'] Сессия закрыта.');
        // вернем тэгу исходное значение
        ClientBtn.Tag:=0;
        // вернем исходную надпись клавиши
        ClientBtn.Caption:='Подключиться';
    end;
end;
```

Рисунок 12 – Описание процедуры ClientBtnClick

```
procedure TForm1.ClientSocketRead(Sender: TObject; Socket: TCustomWinSocket);
begin
    // добавим в ChatMemo пришедшее сообщение
    ChatMemo.Lines.Add(Socket.ReceiveText());
end;
```

Рисунок 13 – Описание чтения сообщения с сервера ClientSocketRead

```
procedure TForm1.ClientSocketConnect(Sender: TObject; Socket: TCustomWinSocket);
begin
    // добавим в ChatMemo сообщение о соединении с сервером
    ChatMemo.Lines.Add('['+TimeToStr(Time)+'] Подключение к серверу.');
end;
```

Рисунок 14 – Описание добавления сообщения при установлении соединения ClientSocketConnect

```
procedure TForm1.ClientSocketDisconnect(Sender: TObject; Socket: TCustomWinSocket);
begin
    // добавим в ChatMemo сообщение о потере связи
   ChatMemo.Lines.Add('['+TimeToStr(Time)+'] Сервер не найден.');
end;
```

Рисунок 15 – Описание добавления сообщения при потере соединения ClientSocketDisconnect

В ходе данного практического задания был разработан сетевой чат, который может дополняться функционалом и проверками ввода данных.

8.2. Разработка клиент-серверного приложения для подключения к базе данных MS SQL

Одноуровневая архитектура «клиент-сервер» – такая, где все прикладные программы рассредоточены по рабочим станциям, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные. В целом такая архитектура очень надёжна, однако ей сложно управлять, поскольку в каждой рабочей станции данные будут присутствовать в разных вариантах. Поэтому возникает проблема их синхронизации на отдельных машинах [9].

В данном практическом задании будет разработано клиентское приложение для подключения к базе данных *MS SQL* на языке Delphi.

Вся работа будет выполняться с помощью ActiveX Data Objects (*ADO*). *ADO* – это технология от компании Microsoft для обращения к реляционным базам данных. В среде разработки на вкладке компонентов *ADO* их доступно 6 (рисунок 16): *TADOConnection*, *TADOCommand*, *TADODataSet, TADOTable*, *TADOQuery*, *TADOStoredProc*.

1) *TADOConnection* аналогичен компоненту *TDatabase* и используется для указания базы данных и работы транзакциями.
2) *TADOTable* – таблица, доступная через ADO.
3) *TADOQuery* – запрос к базе данных. Это может быть как запрос, в результате которого возвращаются данные и базы (например, *SELECT*), так и запрос, не возвращающий данных (например, *INSERT*).
4) *TADOStoredProc* – вызов хранимой процедуры. В отличие от *BDE* и *InterBase* хранимые процедуры в *ADO* могут возвращать набор данных, поэтому компонент данного типа является потомком от *TDataSet* и может выступать источником данных в компонентах типа *TDataSource*.
5) *TADOCommand* и *TADODataSet* являются наиболее общими компонентами для работы с *ADO*, но и наиболее сложными в работе. Оба компонента

позволяют выполнять команды на языке провайдера данных (так в *ADO* называется драйвер базы данных) [10].

Рисунок 16 – Компоненты ADO

Используем компонент ADOConnection. Для него зададим строку подключения: укажем подключаемые данные (сервер), имя сервера, данные пользователя и данные базы данных (рисунок 17).

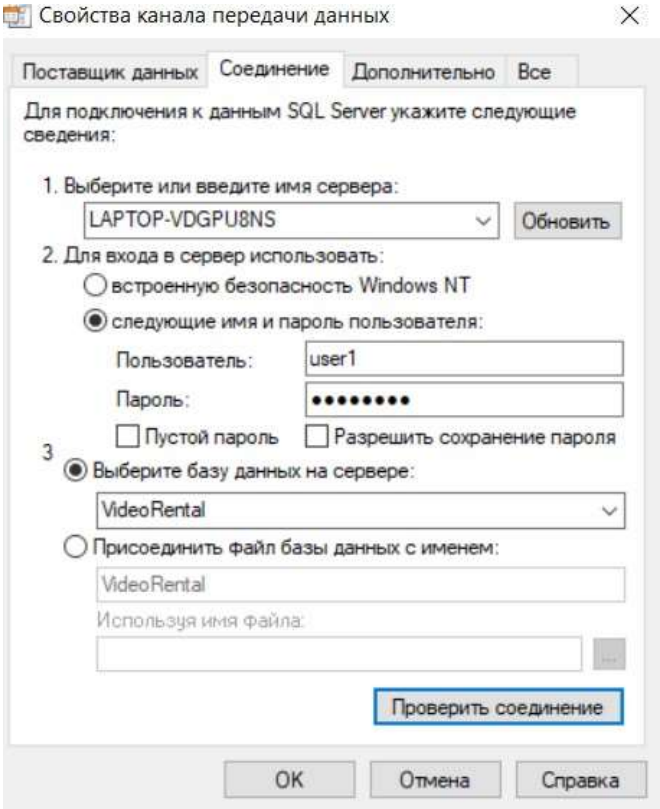


Рисунок 17 – Настройка строки подключения к базе данных

После создания строки подключения есть возможность проверки соединения по кнопке, указанной внизу окна.

Добавим остальные компоненты (*ADOQuery*, *TDataSource* и *DBGrid*) и свяжем их между собой. *ADOQuery* будет получать запрос, записывать его в

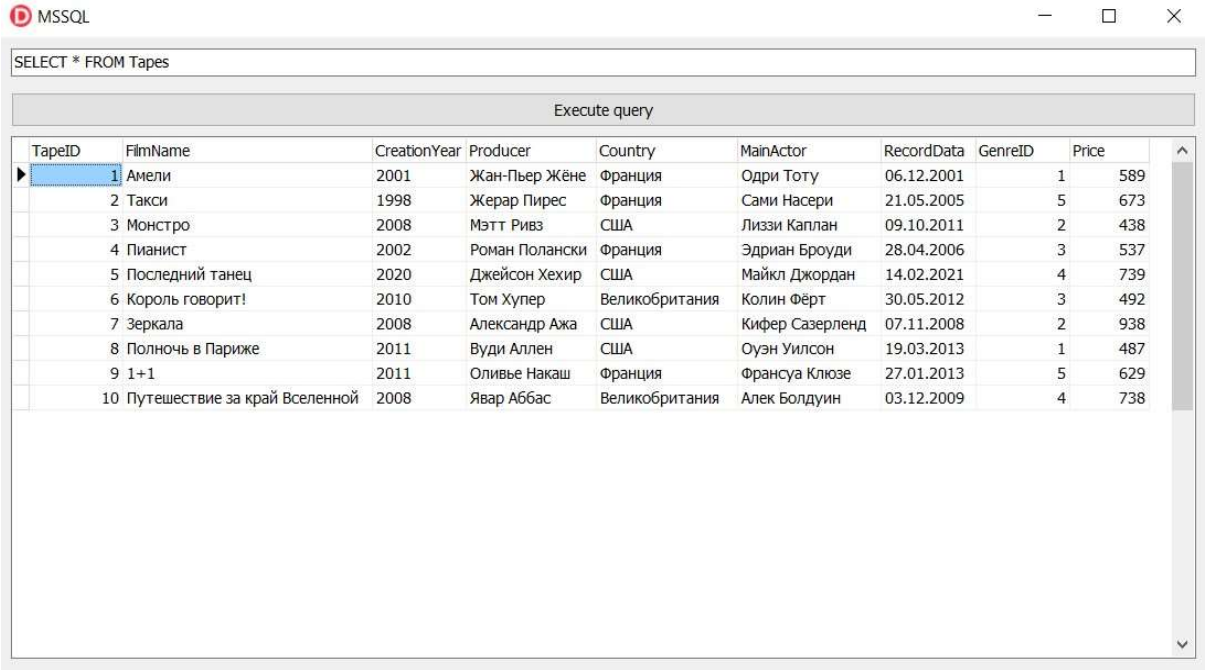
TDataSource и выводить результат с помощью компонента *DBGrid*. Связь можно реализовать с помощью указания соответствующих свойств для каждого компонента.

Также добавим кнопку и поля для ввода запроса. В событие кнопки по нажатию введем код, представленный на рисунке 18.

```
ADOQuery1.Active:=False;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:=Edit1.Text;
ADOQuery1.Active:=True;
```

Рисунок 18 – Обязательные команды при нажатии кнопки

Введем желаемый запрос к базе данных и получим результат, выведенный в соответствующую таблицу (рисунок 19).



TapeID	FilmName	CreationYear	Producer	Country	MainActor	RecordData	GenreID	Price
1	Амели	2001	Жан-Пьер Жёне	Франция	Одри Тоту	06.12.2001	1	589
2	Такси	1998	Жерар Пирес	Франция	Сами Насери	21.05.2005	5	673
3	Монстро	2008	Мэтт Ривз	США	Лиззи Каплан	09.10.2011	2	438
4	Пианист	2002	Роман Полански	Франция	Эдриан Броуди	28.04.2006	3	537
5	Последний танец	2020	Джейсон Хехир	США	Майкл Джордан	14.02.2021	4	739
6	Король говорит!	2010	Том Хупер	Великобритания	Колин Фёрт	30.05.2012	3	492
7	Зеркала	2008	Александр Ажа	США	Кифер Сазерленд	07.11.2008	2	938
8	Полночь в Париже	2011	Вуди Аллен	США	Оуэн Уилсон	19.03.2013	1	487
9	1+1	2011	Оливье Накаш	Франция	Франсуа Клюзе	27.01.2013	5	629
10	Путешествие за край Вселенной	2008	Явар Аббас	Великобритания	Алек Болдуин	03.12.2009	4	738

Рисунок 19 – Результат работы программы

В ходе данной практической работы был создан и настроен клиент, обрабатывающий запросы к базе данных MS SQL и выводящий данные в виде таблицы в компонент DBGrid.

8.3. Разработка клиент-серверного приложения для подключения к сетевой почте

К двухзвенной архитектуре «клиент-сервер» следует относить такую, в которой прикладные программы сосредоточены на сервере приложений

(*Application Server*), например, сервере 1С или сервере *CRM*, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере [9].

В данном практическом задании сервером будет являться почтовый сервер, к которому обращается данное приложение.

Для работы с сетью в Delphi используются *Indy* компоненты, которые вы можете найти на нескольких вкладках среды разработки. Найдите на вкладке *Indy Clients* – компонент *IdPOP3*, на вкладке *Indy Misc* – компонент *IdMessage*. С помощью компонента *IdPOP3* можно осуществить подключение к почтовому серверу по протоколу *POP3* и получить необходимое количество сообщений. Компонент *IdMessage* будет использоваться как буфер для получаемого письма. Также потребуется *Memo* для отображения текста письма и кнопка для загрузки очередного сообщения. Пример размещения перечисленных компонентов на форме представлен на рисунке 20, пример обработки входящей почты с использованием POP3 представлен на рисунке 21, а рисунок 22 иллюстрирует запуск приложения для обращения к почтовому серверу.

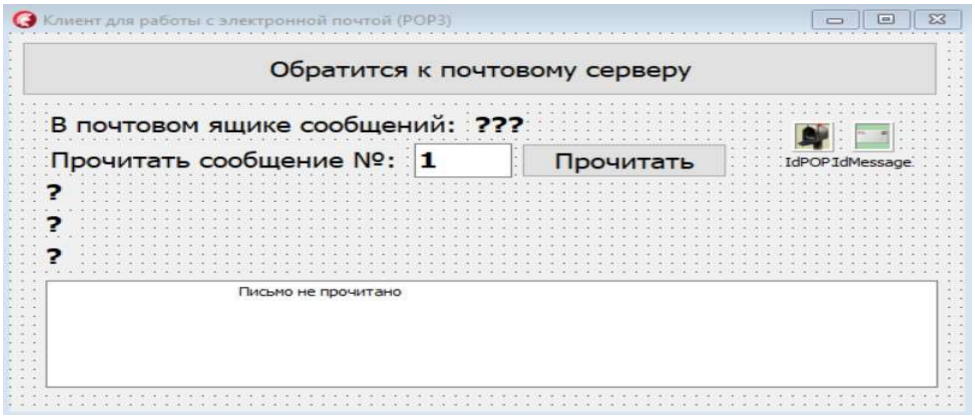


Рисунок 20 – Вид формы разрабатываемого приложения

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
    msgs: integer;
    i: integer;
    flag: boolean;
    msgcnt: integer;
begin
IdPOP3.Host:= 'mail.jino.ru';          //адрес почтового сервера
IdPOP3.Username:= 'test';              //логин
IdPOP3.Password:= 'Pa$$w0rd123';       //пароль
IdPOP3.Port:= 110;                     //порт
IdPOP3.Connect;                        //подключение к серверу
try
  msgcnt:= IdPOP3.CheckMessages;
  Label2.Caption:=IntToStr(msgcnt);
  for I:= msgcnt to 1 do
  begin
      IdMessage.Clear;                 //очистка буфера для хранения сообщений
     flag:= false;
     if (IdPOP3.Retrieve(i, IdMessage)) then //получение письма
     begin
          Application.ProcessMessages
          ProcessMessage(IdMessage.From.Address, IdMessage.Subject);
          flag:= true;
      end;
      if flag then
      begin
     IdPOP3.Delete(i);                 //удаление письма
     end;
  end;
  finally
     IdPOP3.Disconnect;
  end;
end;
```

Рисунок 21 – Обработка входящей почты

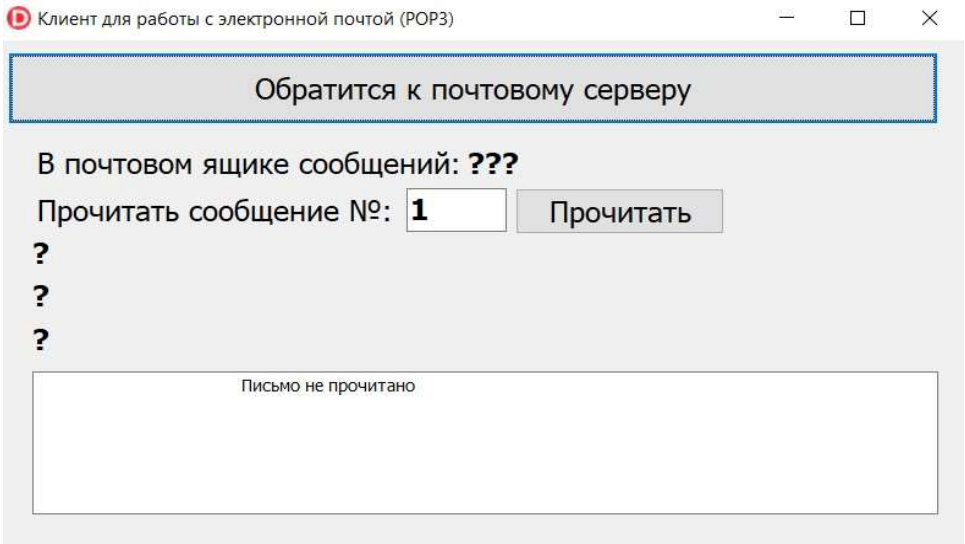


Рисунок 22 – Запуск приложения для обращения к почтовому серверу При нажатии на кнопку «Обратиться к почтовому серверу», клиент считывает

письма и их количества выводит пользователю, что представлено на рисунке 23.

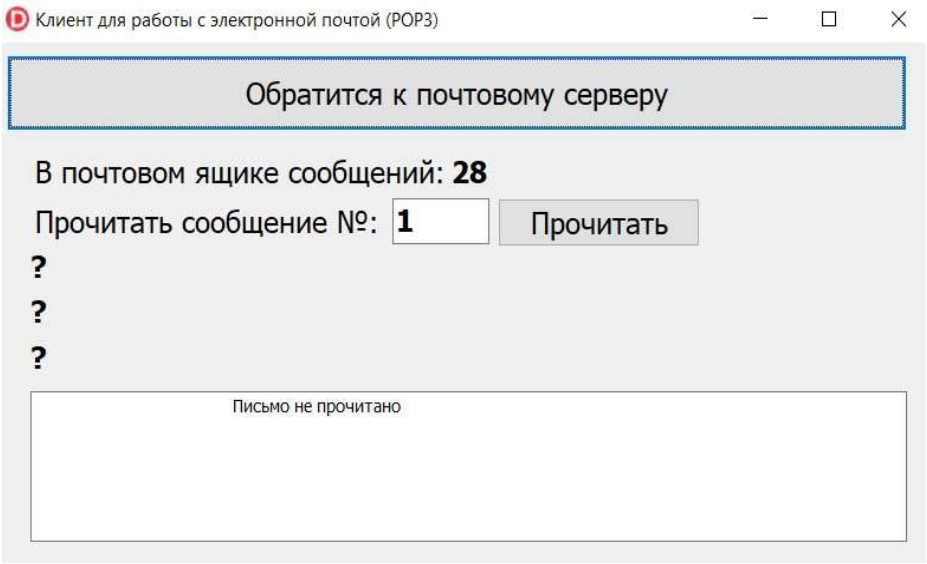


Рисунок 23 – Результат обращения к почтовому серверу

Введя соответствующий номер письма и нажав на кнопку «Прочитать», клиент снова пошлет запрос к почтовому серверу, но уже выведет содержимое запрашиваемого письма. Также будет выведена информация заголовка: от кого, тема и дата, и тело письма, его содержимое. Код кнопки «Прочитать» представлен на рисунке 24.

```
IdMessage.Clear;                // очистка буфера для сообщения
IdMessage.CharSet := 'UTF-8';
Memo1.Clear;                    // очистка компонента memo для отображения текста письма.
IdPOP3.Retrieve(StrToInt(Edit1.Text), IdMessage);  // получение одного сообщения
Label4.Caption := 'От: ' + IdMessage.From.Address;
Label5.Caption := 'Тема: ' + IdMessage.Subject;
Label6.Caption := 'Дата: ' + DateTimeToStr(IdMessage.Date);
IdMessage.SaveToFile('Test.txt');
Memo1.Lines.AddStrings(IdMessage.Body);
```

Рисунок 24 – Код кнопки «Прочитать»

Результат работы клиента для связи с почтовым сервером представлен на рисунке 25.

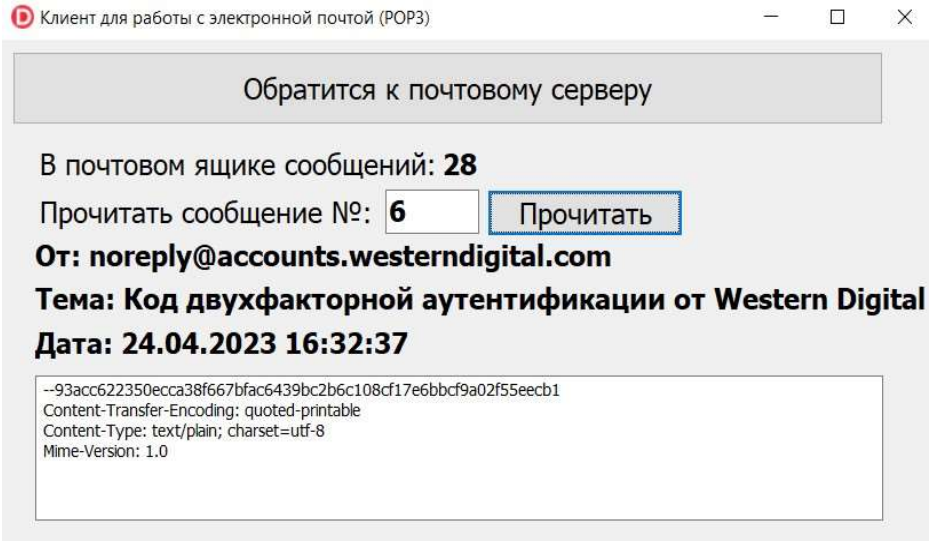


Рисунок 25 – Результат работы клиента для связи с почтовым сервером

В данном практическом задании был разработан клиент, с помощью которого было установлено соединение с почтовым сервером, а также получена требуемая информация и выведена пользователю.

Заключение

Архитектура информационных систем является фундаментом для создания успешных IT-проектов. Понимание принципов и концепций архитектуры позволяет разработчикам и архитекторам принимать обоснованные решения, гарантируя высокую производительность, безопасность и масштабируемость системы. С помощью правильной архитектуры можно создавать мощные и инновационные информационные системы, которые способны преобразовывать и улучшать организации и общество в целом.

Учебное пособие предоставляет обширный обзор концепций, принципов и шаблонов, связанных с архитектурой информационных систем. Оно помогает читателям получить общее представление о том, что такое архитектура и как она влияет на разработку и функционирование информационных систем.

Список источников

1 Советов Б.Я. Архитектура информационных систем: учебник для студ. учреждений высш. проф. образования / Б.Я. Советов, А. И. Водяхо, В.А. Дубенецкий, В.В. Цехановский. – М.: Издательский центр «Академия», 2012. – 288 с.

2 Рыбальченко М.В. Архитектура информационных систем: учеб. пособие для СПО / М. В. Рыбальченко. – М.: Издательство Юрайт, 2019. 91 с. Серия: Профессиональное образование.

3 Клиент-серверная архитектура (Client-Server Architecture). [Электронный ресурс]. - URL: https://vladislaveremeev.gitbook.io/qa_bible/setii-okolo-nikh/klient-servernaya-arkhitektura-client-server-architecture (дата обращения: 09.07.2023)

4 Что такое сервис-ориентированная архитектура? (What is SOA (ServiceOriented Architecture)?). [Электронный ресурс]. - URL: https://aws.amazon.com/ru/what-is/service-oriented-architecture/?nc1=h_ls (дата обращения: 09.07.2023)

5 Рогозов Ю.И., Свиридов А.С., Кучеров С.А. Архитектура информационных систем: учебное пособие. – Ростов-на-Дону: Изд-во ЮФУ, 2014. – 117 с.

6 Паттерны и фреймворки в архитектуре ИС. [Электронный ресурс]. - URL: https://studme.org/282507/informatika/patterny_freymvorki_arhitekture (дата обращения: 09.07.2023)

7 Стандарт ISO/IEC 15288 «Системная инженерия — процессы жизненного цикла систем». [Электронный ресурс]. - URL: http://www.iso.org/iso/support/faqs.htm (дата обращения: 18.07.2023). 8 А. Ф. Галимянов., Ф. А. Галимянов. Архитектура информационных систем:
учебное пособие. – Казань: Казан. ун-т, 2019. – 117 с.

9 Архитектура «Клиент-Сервер». [Электронный ресурс]. - URL: https://itelon.ru/blog/arkhitektura-klient-server/ (дата обращения: 29.07.2023).

10 Компоненты ADO. [Электронный ресурс]. - URL: http://www.interface.ru/home.asp?artId=3671 (дата обращения: 29.07.2023).

Ананченко Игорь Викторович
Войтюк Татьяна Евгеньевна
Марченко Елена Вадимовна

АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А