|  |  |
| --- | --- |
| ДИСЦИПЛИНА | Создание программного обеспечения |
| ИНСТИТУТ | Институт перспективных технологий и индустриального программирования |
| КАФЕДРА | Кафедра индустриального программирования |
| ВИД УЧЕБНОГО МАТЕРИАЛА | Лекция |
| ПРЕПОДАВАТЕЛЬ | Макиевский Станислав Евгеньевич |
| СЕМЕСТР | 3 семестр, 2024-2025 гг. |

# 

## Лекция №5

**Технологии бэкенд разработки. Профессия бэкенд-разработчика**

**Что представляет собой BPMN: это графический стандарт моделирования бизнес-процессов, разработанный для визуализации, документирования, анализа и оптимизации бизнес-процессов.**

BPMN (Business Process Model and Notation) - это графический стандарт, разработанный для моделирования и визуализации бизнес-процессов в компаниях и организациях. Он предоставляет единый нотационный язык для представления различных бизнес-процессов и их взаимодействий.

BPMN был создан для решения ряда задач:

* Визуализация процессов: С помощью BPMN бизнес-процессы представляются в виде графических диаграмм, что делает их более понятными и доступными для всех участников организации.
* Документирование процессов: BPMN позволяет подробно описать каждый шаг процесса, его входы, выходы, промежуточные события и участников, что делает документацию более структурированной и легко читаемой.
* Анализ и оптимизация процессов: Используя BPMN, бизнес-аналитики могут проводить анализ бизнес-процессов, выявлять узкие места, определять возможности для улучшения и оптимизации процессов.
* Автоматизация процессов: BPMN диаграммы могут быть использованы в процессах автоматизации, что позволяет смоделировать и реализовать процессы в системах управления бизнес-процессами (BPMS).

**Основные элементы BPMN**

BPMN использует набор графических элементов, которые представляют различные аспекты бизнес-процессов:

* Задача (Task): Представляет действие или задачу, которую выполняет участник процесса.
* Шлюз (Gateway): Определяет различные пути и условия, по которым может развиваться процесс.
* Событие (Event): Показывает важные события, которые могут происходить в процессе.
* Поток управления (Sequence Flow): Показывает последовательность выполнения задач и связи между элементами процесса.
* Пул (Pool): Используется для разделения бизнес-процессов между различными участниками.

**Преимущества BPMN**

Единый стандарт: BPMN предоставляет единый стандарт для представления бизнес-процессов, что делает коммуникацию и взаимодействие между участниками проекта более эффективными.

Понятность и доступность: Графическое представление процессов делает BPMN диаграммы понятными для всех участников организации, даже для тех, кто не имеет технической специализации.

Улучшение производительности: Анализ и оптимизация бизнес-процессов с помощью BPMN позволяют выявить узкие места и улучшить производительность компании.

Автоматизация: BPMN диаграммы могут быть использованы для автоматизации процессов, что помогает уменьшить ручные операции и улучшить эффективность работы.

В заключении хочется сказать, что BPMN - это мощный инструмент для визуализации, документирования, анализа и оптимизации бизнес-процессов. Его использование позволяет создать более понятное представление о процессах в организации, что способствует повышению эффективности, оптимизации и автоматизации работы, что, в свою очередь, приводит к улучшению результатов и достижению поставленных бизнес-целей.

## Кто такой backend-разработчик

Backend разработчик (Backend Developer) - это специалист, который занимается разработкой серверной части программного обеспечения. Он отвечает за обработку данных, бизнес-логику и взаимодействие с базами данных. Backend разработчик работает на уровне сервера, который обеспечивает взаимодействие между клиентами и базами данных, а также управляет всеми аспектами, связанными с обработкой запросов и ответами.

**Навыки серверного разработчика**

Языки программирования: Backend разработчик часто использует языки программирования, такие как Python, Java, C#, Ruby, PHP, Node.js, Go и другие, в зависимости от конкретных требований проекта.

Базы данных: Он должен быть знаком с работой с различными системами управления базами данных (СУБД), такими как MySQL, PostgreSQL, MongoDB, Oracle и др.

Фреймворки: Знание фреймворков, которые упрощают разработку серверной части, таких как Django (Python), Spring (Java), Laravel (PHP), Express.js (Node.js), и других, также является важным.

API и протоколы: Backend разработчик должен понимать, как создавать и использовать API (Application Programming Interface) и различные протоколы обмена данными, такие как REST, SOAP, GraphQL и др.

Безопасность: Знание методов обеспечения безопасности, защиты от атак и уязвимостей также критически важно для Backend разработчика.

**Обязанности Backend разработчика**

Разработка серверной логики: Он занимается разработкой бизнес-логики и алгоритмов, обрабатывает запросы от клиентов и возвращает соответствующие данные.

Управление базами данных: Backend разработчик отвечает за создание и управление базами данных, выполнение запросов и обеспечение целостности данных.

Оптимизация производительности: Он стремится к повышению производительности сервера, оптимизируя код, уменьшая время отклика и ресурсоемкость приложения.

Обеспечение безопасности: Backend разработчик заботится о безопасности приложения, защищая данные от несанкционированного доступа и других видов угроз.

Тестирование: Он отвечает за тестирование серверной части приложения, чтобы убедиться в его работоспособности и соответствии требованиям.

Данный разработчик играет важную роль в разработке программного обеспечения, обеспечивая функциональность, производительность и безопасность серверной части приложения. Его навыки и обязанности включают работу с языками программирования, базами данных, фреймворками, а также обеспечение надежности и безопасности приложения. Благодаря работе бекенд разработчиков, клиенты могут получать надежные и эффективные взаимодействия с приложением, обеспечивая лучший пользовательский опыт.

## Что представляет собой серверная архитектура в контексте разработки программного обеспечения.

Серверная архитектура - это структура и организация серверной части программного обеспечения, которая обрабатывает запросы от клиентских устройств, выполняет логику бизнес-приложений, взаимодействует с базами данных и обеспечивает обмен данных между клиентами и сервером.

Основные компоненты серверной архитектуры:

* Серверы: Основной компонент серверной архитектуры - это серверы, которые представляют собой физические или виртуальные машины, на которых размещаются приложения и выполняется бизнес-логика.
* Базы данных: Базы данных являются важной частью серверной архитектуры и служат для хранения данных, необходимых для работы приложения.
* API (Application Programming Interface): API представляет собой набор интерфейсов и методов, которые позволяют клиентам взаимодействовать с сервером и получать доступ к его функциональности.
* Бизнес-логика: Бизнес-логика включает в себя правила и процессы, которые определяют, как приложение должно обрабатывать данные и выполнять определенные действия в ответ на запросы клиентов.
* Web-серверы: Web-серверы предоставляют доступ к веб-приложениям, обрабатывая HTTP-запросы от клиентов и возвращая соответствующие HTTP-ответы.

**Значение серверной архитектуры в разработке ПО**

Эффективность и производительность: Хорошо спроектированная серверная архитектура позволяет улучшить производительность и эффективность приложения, обеспечивая более быструю обработку запросов и минимизацию задержек.

Масштабируемость: Правильно разработанная серверная архитектура позволяет масштабировать приложение, чтобы оно могло обслуживать большое количество пользователей и увеличивать нагрузку.

Безопасность: Серверная архитектура играет важную роль в обеспечении безопасности данных и защите от различных видов атак.

Управление ресурсами: Оптимизированная серверная архитектура позволяет более эффективно использовать ресурсы серверов, такие как процессорное время и память.

Поддержка разных клиентов: Хорошо спроектированная серверная архитектура обеспечивает возможность обслуживания различных клиентских устройств, таких как веб-браузеры, мобильные приложения и десктопные приложения.

**Типы серверных архитектур**

Monolithic Architecture (Монолитная архитектура): Все компоненты приложения находятся в одной программе, что делает их простыми в разработке и тестировании, но может затруднить масштабирование и обновление.

Microservices Architecture (Микросервисная архитектура): Приложение разбивается на небольшие, независимые сервисы, что упрощает масштабирование и обновление, но требует сложной организации и управления.

Serverless Architecture (Серверная архитектура без сервера): Весь код приложения выполняется в ответ на события, что уменьшает необходимость управления серверами, но может быть ограничено функциональностью и производительностью.

*Серверная архитектура является критическим компонентом в разработке программного обеспечения. Она обеспечивает эффективность, масштабируемость и безопасность приложений, а также позволяет обеспечить поддержку различных клиентов и управление ресурсами. Выбор конкретной архитектуры зависит от требований проекта и специфики бизнес-задач, которые необходимо решить.*

**Роль сервера в клиент-серверной модели взаимодействия. Зачем нужна хорошая архитектура на серверной стороне и как она влияет на производительность, масштабируемость и надежность приложения.**

**Роль сервера в клиент-серверной модели**

В клиент-серверной модели взаимодействия приложение разделяется на две основные части: клиентскую и серверную. Сервер представляет собой центральный узел, который предоставляет доступ к данным, бизнес-логике и ресурсам, а клиенты обращаются к серверу для получения данных, обработки запросов и выполнения определенных действий.

**Функции сервера в клиент-серверной модели**

Управление данными: Сервер хранит данные приложения, такие как информацию о пользователях, товарах, заказах и других сущностях, и предоставляет доступ к этим данным клиентам.

Бизнес-логика: Сервер обрабатывает бизнес-логику приложения, что включает в себя выполнение различных операций, проверку прав доступа, расчеты и другие операции, необходимые для работы приложения.

Обработка запросов: Клиенты отправляют запросы на сервер, и сервер обрабатывает эти запросы, возвращая соответствующие ответы.

Маршрутизация: Сервер определяет, какие запросы должны быть переданы определенным компонентам приложения для обработки.

Обеспечение безопасности: Сервер отвечает за обеспечение безопасности данных и защиту приложения от различных видов атак.

**Значение хорошей серверной архитектуры**

Производительность: Хорошо спроектированная серверная архитектура обеспечивает более быструю обработку запросов и более высокую производительность приложения.

Масштабируемость: Хорошая архитектура позволяет масштабировать приложение, чтобы оно могло обслуживать большое количество пользователей и увеличивать нагрузку.

Надежность и отказоустойчивость: Хорошо спроектированный сервер обеспечивает надежность и отказоустойчивость приложения, что позволяет избежать сбоев и потери данных.

Управление ресурсами: Оптимизированная серверная архитектура позволяет более эффективно использовать ресурсы серверов, такие как процессорное время и память.

**Примеры технологий для хорошей серверной архитектуры**

RESTful API: Предоставляет универсальный и гибкий способ для клиентов обращаться к серверу и получать данные в формате JSON или XML.

Microservices Architecture: Разбивает приложение на небольшие, независимые микросервисы, что упрощает масштабирование и обновление.

Load Balancing: Распределяет трафик между несколькими серверами, что позволяет более равномерно распределить нагрузку и улучшить производительность.

Сервер играет ключевую роль в клиент-серверной модели взаимодействия, обеспечивая управление данными, бизнес-логику, обработку запросов и обеспечение безопасности приложения. Хорошая архитектура на серверной стороне важна для обеспечения высокой производительности, масштабируемости и надежности приложения, что в свою очередь способствует более эффективной работе приложения и удовлетворению потребностей пользователей.

## ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. "Clean Code: Создание, анализ и рефакторинг" - Роберт Мартин (Питер, 2019, 464 страницы)
2. "Паттерны проектирования" - Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес (Питер, 2018, 368 страниц)
3. "Рефакторинг. Улучшение существующего кода" - Мартин Фаулер (Вильямс, 2018, 448 страниц)
4. "Чистая архитектура. Искусство разработки программного обеспечения" - Роберт Мартин (Питер, 2019, 432 страницы)