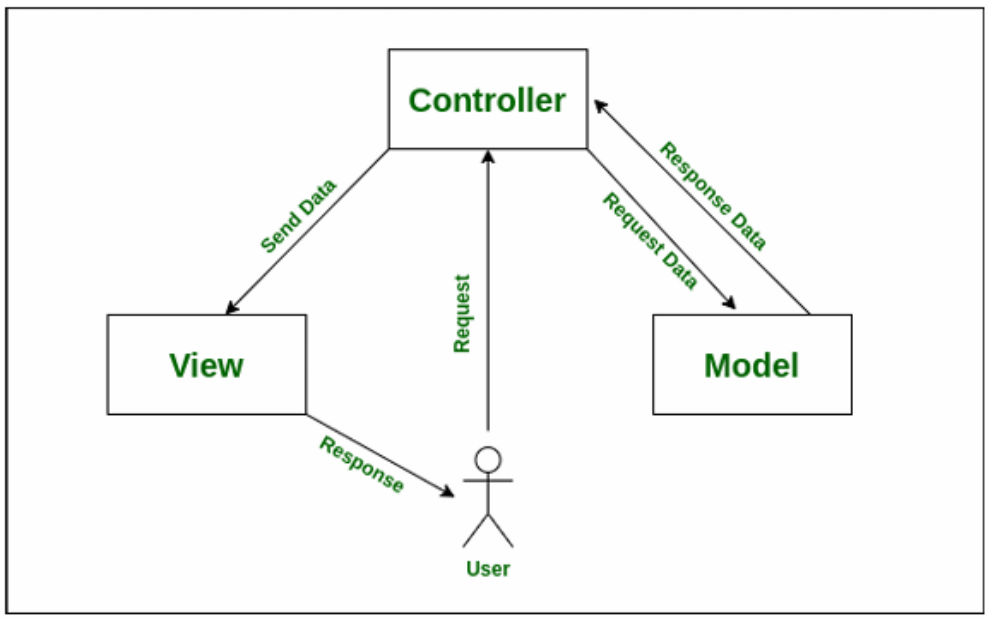
Опишите шаблон MVC, структуру и назначение компонентов.

MVC

- Model
- View
- Controller

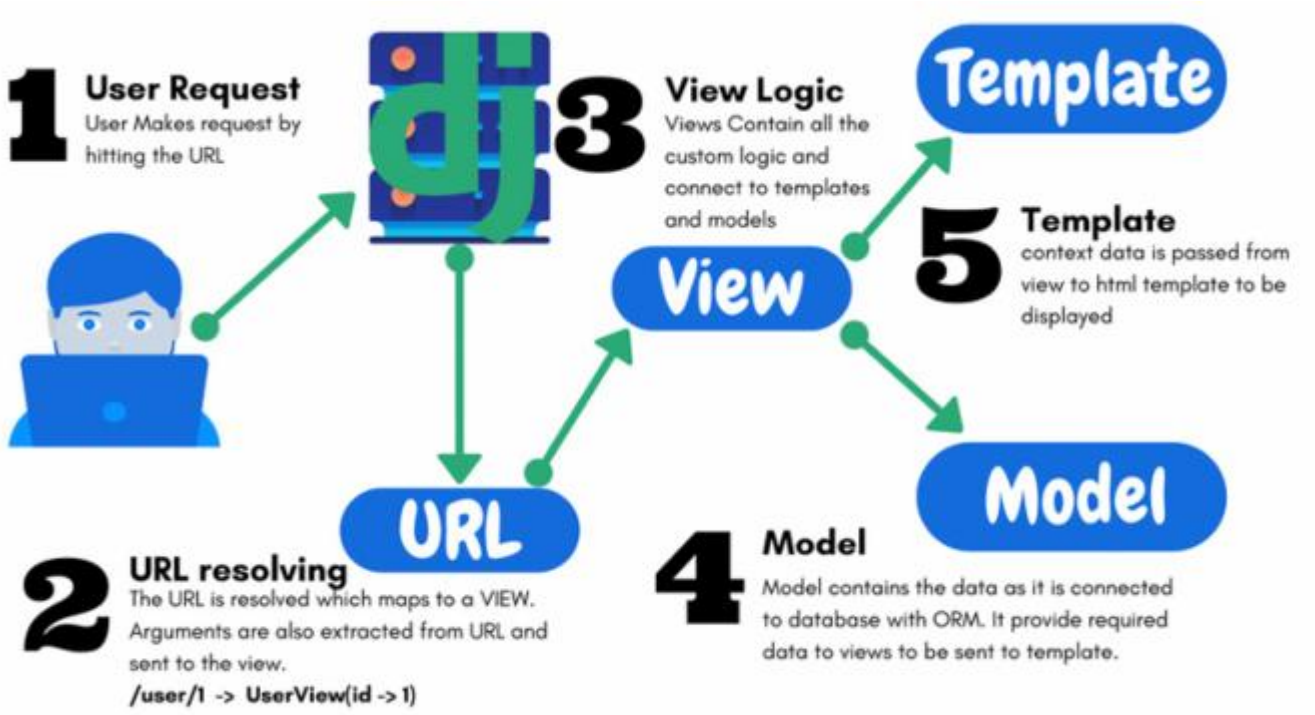


Model (модель) отвечает за данные приложения и логику работы с ними. Она управляет данными, их сохранением, извлечением и изменением, а также может реализовывать бизнес-логику, которая не зависит от пользовательского интерфейса.

View (представление) отвечает за отображение данных пользователю. Оно интерпретирует информацию, полученную от модели, и отображает её в удобном виде. Представление не должно содержать бизнес-логики или управлять данными, оно лишь показывает их пользователю.

Controller (контроллер) связывает модель и представление, управляя их взаимодействием. Он получает пользовательский ввод (через представление), обрабатывает его (например, с помощью модели), а затем обновляет представление для отображения новых данных.

Опишите схему, как реализован шаблон MVC в фреймворке Django.



- Django – это MVC фреймворк
- При обработке запроса сначала обрабатывается URL
- Решается, какой view будет его обрабатывать
- View обращается к БД или нейросети
- Результаты вносятся в шаблон Template, получается HTML

Что такое Django? Его назначение и возможности.

Django — это высокоуровневый фреймворк для веб-разработки на языке Python, который способствует быстрому созданию веб-приложений, обеспечивая «из коробки» множество функциональностей для решения типичных задач в разработке.

Основное назначение Django — это предоставление разработчикам мощного инструмента для создания веб-приложений, который упрощает рутинные задачи, такие как управление базой данных, обработка URL-адресов, создание форм, аутентификация пользователей и т.д. Он ориентирован на быстрое создание проектов и облегчение работы с данными.

Основные возможности Django:

1. Модульная архитектура: Model-Template-View (MTV).
2. Административная панель: Django автоматически генерирует административную панель для управления данными в приложении.
3. ORM (Object-Relational Mapping): Django включает встроенный инструмент ORM, который позволяет работать с базой данных через Python-классы.
4. Безопасность: Django предоставляет защиту от множества распространенных уязвимостей.
5. Роутинг (URL-маршрутизация): В Django с помощью системы маршрутизации можно задавать URL-адреса для различных представлений.
6. Шаблоны (Template Engine): Django включает систему шаблонов, которая позволяет разработчикам разделить логику приложения и представление.
7. Миграции базы данных: Django предоставляет систему миграций, которая позволяет легко изменять структуру базы данных при изменении моделей данных.
8. Кеширование: Django поддерживает различные способы кеширования, которые могут ускорить работу приложения, уменьшив нагрузку на сервер и базу данных.
9. Поддержка REST API: С использованием библиотек, таких как Django REST Framework, можно быстро создавать и управлять API для веб-приложений.
10. Тестирование: Django включает средства для создания юнит-тестов для проверки функциональности приложения.

Что такое шаблонизация Django? Приведите примеры.

Шаблонизация в Django — это процесс генерации динамических HTML-страниц с использованием системы шаблонов Django. Она позволяет отделить логику приложения от представления (UI), что способствует чистоте и поддерживаемости кода. В Django используется встроенная система шаблонов, которая позволяет вставлять переменные, выполнять циклы и условия, а также использовать фильтры для изменения данных перед выводом.

Основные принципы шаблонизации в Django

1. Вставка переменных: В шаблонах можно вставлять данные, передаваемые из представлений.
2. Условия: Использование условных операторов (if, else).
3. Циклы: Работа с коллекциями данных с помощью циклов (for).
4. Фильтры: Применение фильтров для обработки данных перед их выводом.
5. Включение шаблонов: Можно использовать другие шаблоны внутри основного шаблона для удобства и повторного использования кода.
6. Теги: Django предоставляет теги шаблонов для выполнения различных операций, например, для работы с блоками, загрузки статических файлов и т.д.

```
<h1>Список товаров</h1>
<ul>
    {% for product in products %}
        <li>{{ product.name }} - {{ product.price }} руб.</li>
    {% empty %}
        <li>Товары не найдены.</li>
    {% endfor %}
</ul>
```

```
<h1>Добро пожаловать, {{ user.username }}!</h1>

{% if user.is_authenticated %}
    <p>Вы вошли в систему.</p>
{% else %}
    <p>Пожалуйста, войдите, чтобы продолжить.</p>
{% endif %}
```

```
from django import template

register = template.Library()

@register.filter
def currency(value):
    return f"{value} руб."
```

```
<p>Сегодня: {{ current_date|date:"d-m-Y" }}</p>
<p>Привет, {{ user.username|upper }}!</p>
```

```
{% load filters %}

<p>Цена: {{ product.price|currency }}</p>
```

```
<a href="{% url 'product_detail' product.id %}">Посмотреть подробности</a>
```

Опишите протокол HTTP. Схему работы и основные понятия.

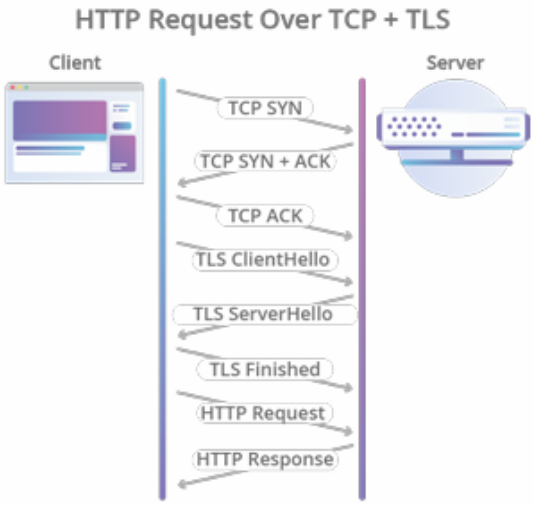
- HTTP – протокол передачи гипертекста. Сейчас для произвольных данных

Протокол является клиент-серверным, то есть существуют:

- Клиент – потребитель, он инициирует взаимодействие
- Сервер – поставщик, ждет запроса, обрабатывает его и возвращает обратно ответ

- Данный тип взаимодействия накладывает ограничение при получении уведомлений, сообщений в чате и тд
- HTTP – прокол без хранения состояния между разными запросами. Но компоненты могут хранить например куки (клиенты) или сессии (сервер)

- Протокол HTTP (поверх TCP)
- Протокол HTTPS (поверх TLS и TCP)
- Протокол HTTP/2
- Протокол HTTP/3 (поверх QUIC)

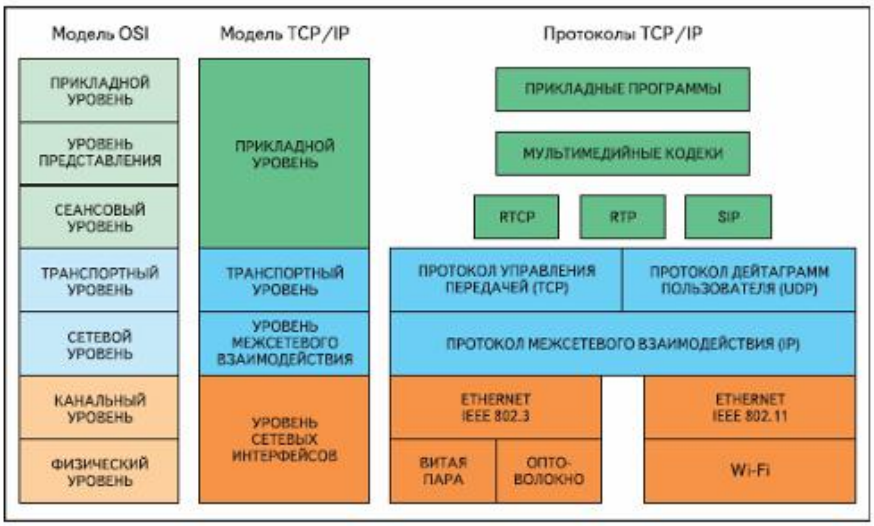


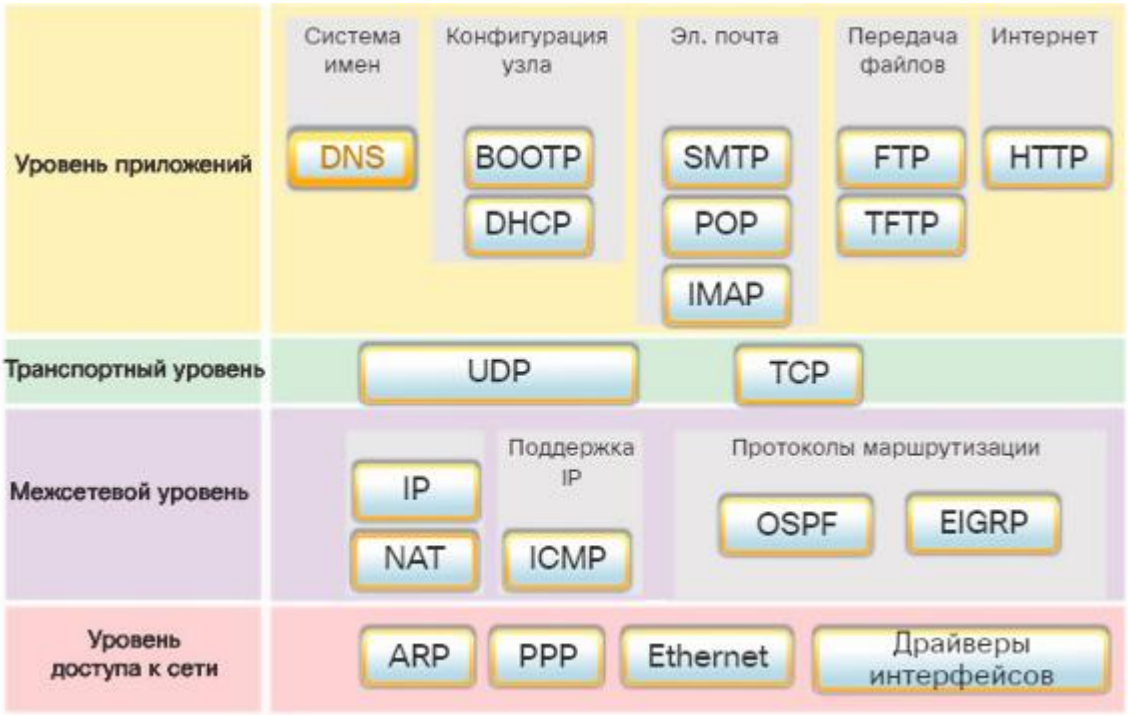
- Протоколы на основе HTTP:

– XML-RPC
– SOAP
– WebDAV
- JSON-RPC (AJAX)
- gPRC

Опишите стек протоколов интернета TCP/IP.

- TCP/IP – стек протоколов на которых базируется Интернет





Перечислите основные составляющие web и опишите их.

- Тим Бернерс-Ли создал три основных компонента WWW:
- язык гипертекстовой разметки документов HTML (HyperText Markup Language);
- универсальный способ адресации ресурсов URI (Universal [Uniform] Resource Identifier);
- протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol – протокол передачи гипертекста).
- Позже к этим трем компонентам добавился четвертый CGI: исполняемая часть, с помощью которой можно создавать динамические HTML-документы.

- HTML-HyperText Markup Language.
- В HTML версии 1.0 были реализованы все элементы разметки, связанные с выделением параграфов, шрифтов, стилей и т.п., т.к. уже первая реализация подразумевала графический интерфейс. Важным компонентом языка стало описание гипертекстовых ссылок, графики и обеспечение возможности поиска по ключевым словам.
- В качестве базы для разработки языка гипертекстовой разметки HTML был выбран SGML (Standard Generalised Markup Language – стандартный общий язык разметки). Тим Бернерс-Ли описал HTML в терминах SGML как описывают языки программирования в терминах формы Бекуса-Наура.

- Вторым важным компонентом WWW стал универсальный способ адресации ресурсов URI (Universal Resource Identifier).
- Кроме термина URI можно также встретить термины:
  - URL (Universal Resource Locator),
  - URN (Universal Resource Name).
- Наиболее общим термином является URI, который может быть или URL или URN. В соответствии со спецификацией URL определяет ресурс по механизму доступа к ресурсу, а URN по уникальному имени (это не имя файла).
- В результате терминологической путаницы термины URI и URL часто стали использоваться как синонимы. Термин URN используется достаточно редко. Некоторое применение он нашел в технологии XML.

- HTTP – протокол передачи гипертекста. Сейчас для произвольных данных

Протокол является клиент-серверным, то есть существуют:

- Клиент – потребитель, он инициирует взаимодействие
- Сервер – поставщик, ждет запроса, обрабатывает его и возвращает обратно ответ

- Данный тип взаимодействия накладывает ограничение при получении уведомлений, сообщений в чате и тд
- HTTP – прокол без хранения состояния между разными запросами. Но компоненты могут хранить например куки (клиенты) или сессии (сервер)

Что такое HTML, CSS? Приведите примеры.

- HTML-HyperText Markup Language.
- В HTML версии 1.0 были реализованы все элементы разметки, связанные с выделением параграфов, шрифтов, стилей и т.п., т.к. уже первая реализация подразумевала графический интерфейс. Важным компонентом языка стало описание гипертекстовых ссылок, графики и обеспечение возможности поиска по ключевым словам.
- В качестве базы для разработки языка гипертекстовой разметки HTML был выбран SGML (Standard Generalised Markup Language – стандартный общий язык разметки). Тим Бернерс-Ли описал HTML в терминах SGML как описывают языки программирования в терминах формы Бекуса-Наура.

```html
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0
    <title>Простой пример</title>
</head>
<body>
    <h1>Привет, мир!</h1>
    <p>Это простой пример страницы на HTML.</p>
    <a href="https://example.com">Перейти на сайт</a>
</body>
</html>
```

CSS — это язык описания стилей, который используется для определения внешнего вида HTML-элементов на веб-странице. С помощью CSS можно изменять шрифты, цвета, отступы, расположение элементов и другие стилистические характеристики.

```css
h1 {
    font-family: Arial, sans-serif;
    color: #333;
    text-align: center;
}
```

```css
p {
    font-size: 18px;
    line-height: 1.6;
    color: #666;
}
```

Что такое URI? Опишите элементы URI для HTTP.

- Вторым важным компонентом WWW стал универсальный способ адресации ресурсов URI (Universal Resource Identifier).
- Кроме термина URI можно также встретить термины:
  - URL (Universal Resource Locator),
  - URN (Universal Resource Name).
- Наиболее общим термином является URI, который может быть или URL или URN. В соответствии со спецификацией URL определяет ресурс по механизму доступа к ресурсу, а URN по уникальному имени (это не имя файла).
- В результате терминологической путаницы термины URI и URL часто стали использоваться как синонимы. Термин URN используется достаточно редко. Некоторое применение он нашел в технологии XML.

URI – схема HTTP

- http:// хост : порт / путь и имя файла ? параметры # якорь гиперссылки

- Пример:

http:// 127.0.0.1 :8080/index.html

http://localhost:8080/file.html

http://iu5.bmstu.ru:8080/cat1/cat2/script.asp?param1=1&param2=2#anchor1

- Порт по умолчанию – 80.

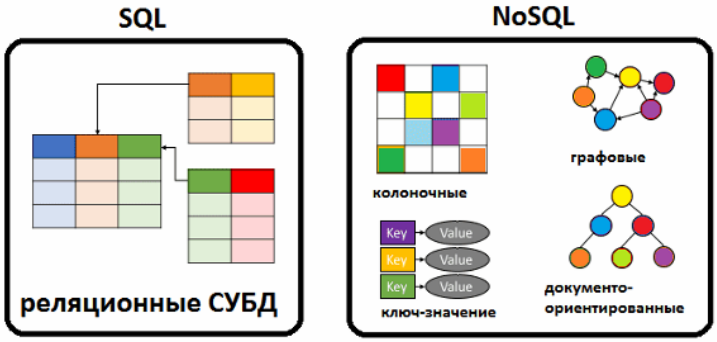
- Рекомендуется использовать наиболее общий термин URI, хотя во многих спецификациях можно также встретить термин URL. Фактически, все адреса в WWW обозначающие ресурсы, являются URL.
- URI (URL) используется в гипертекстовых ссылках для обозначения ресурсов. С помощью URL можно адресовать как гипертекстовые документы формата HTML, так и другие ресурсы, например электронную почту, ftp.
- Для создания URI на национальных языках разрабатывается стандарт IRI.

- Нормализация URI

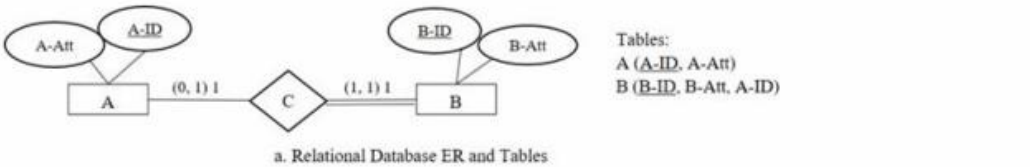
- Семантический URI

Виды баз данных. Приведите примеры и отличия.

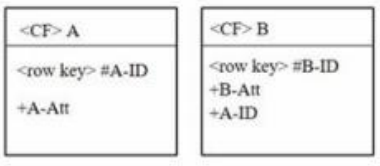
- Документо-ориентированные
- Реляционные
- in memory
- Графовые
- Ключ-значение



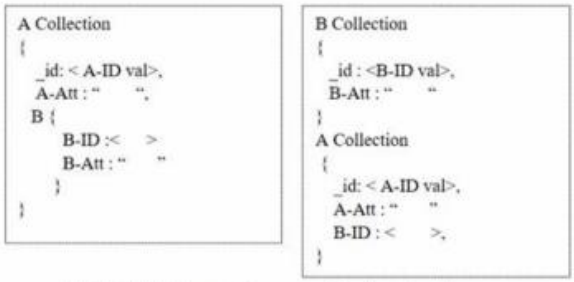
Тип БД	Структура хранения данных	Преимущества	Недостатки
Реляционные БД (RDBMS)	Таблицы (строки и столбцы)	Поддержка SQL, нормализация, транзакции	Не подходят для хранения неструктурированных данных
Документно-ориентированные БД	Документы (JSON/BSON)	Гибкость в схеме, масштабируемость	Отсутствие строгой схемы, не всегда поддерживают SQL
Ключ-значение	Пара ключ-значение	Простота, высокая производительность	Ограниченная структура, трудности с обработкой сложных данных
Графовые БД	Граф (вершины и рёбра)	Идеальны для работы с данными, где важны связи	Меньше стандартов, сложность в интеграции
Колонно-ориентированные БД	Колонки	Оптимальны для аналитики и работы с большими объёмами данных	Меньше подходящих решений для транзакционной обработки



a. Relational Database ER and Tables

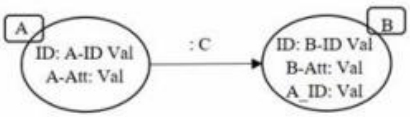


b. Column-based NoSQL Columns



1-Embedded document 2-Referencing document

c. Document-based NoSQL Collections



d. Graph-based NoSQL Nodes

Объясните назначение ORM, ее составляющие. Укажите преимущества и недостатки ORM.

ORM (Object-Relational Mapping) — это подход и технология, которая позволяет разработчикам работать с реляционными базами данных, используя объекты и классы объектно-ориентированных языков программирования. ORM автоматически преобразует данные между объектами в коде и таблицами в базе данных. Вместо того, чтобы вручную писать SQL-запросы для работы с данными, разработчики используют классы и методы, которые ORM интерпретирует в SQL-запросы, делая взаимодействие с базой данных проще и более интуитивно понятным.

Модели представляют собой классы, которые определяют структуру данных в базе данных. Каждое поле класса соответствует столбцу таблицы в базе данных.

ORM использует сессии или контексты для отслеживания изменений в объектах и взаимодействия с базой данных. Сессии предоставляют методы для добавления, обновления, удаления и извлечения объектов.

Маппинг – процесс сопоставления таблиц и полей в базе данных с объектами и атрибутами в коде. ORM автоматически выполняет этот маппинг, что избавляет от необходимости писать SQL-запросы вручную.

ORM предоставляет возможности для выполнения операций с базой данных через объектно-ориентированные запросы. Запросы часто реализуются через методы классов или специализированные API.

ORM часто поддерживает механизм миграций, который позволяет отслеживать изменения в структуре базы данных и автоматически их применять.

Преимущества ORM: ускоряет разработку, уменьшает количество ошибок, кроссплатформенность, чистота кода, упрощение миграций, инкапсуляция бизнес-логики.

Недостатки ORM: проблемы с производительностью, сложность оптимизации запросов, часто бывает избыточным, сложность при миграции на другую СУБД, отсутствие полного контроля над SQL.

Что такое модель и миграция?

Модель — это объектно-ориентированное представление данных в рамках структуры базы данных. В контексте ORM (Object-Relational Mapping), модель используется для описания таблицы базы данных с её полями, которые могут быть представлением столбцов таблицы.

Каждая модель соответствует таблице в базе данных, и каждый экземпляр модели (объект) соответствует строке в этой таблице. Модели используются для выполнения CRUD-операций (создание, чтение, обновление и удаление) с базой данных через объектно-ориентированный интерфейс.

Миграция — это способ отслеживания и применения изменений в структуре базы данных (например, создание, удаление или изменение таблиц и полей) через ORM. Миграции позволяют автоматически синхронизировать структуру базы данных с моделями, определенными в коде.

Миграции помогают управлять изменениями в базе данных без необходимости вручную писать SQL-запросы для создания или модификации схемы.

Укажите группы SQL запросов, их примеры и назначение.

SQL (Structured Query Language) — это язык запросов, используемый для взаимодействия с реляционными базами данных.

- DDL – Data Definition Language
CREATE, ALTER, DROP

- DML – Data Manipulation Language
SELECT, INSERT, UPDATE, DELETE

- DCL – Data Control Language
GRANT, REVOKE

- TCL – Transaction Control Language
COMMIT, ROLLBACK, SAVEPOINT

Что такое веб-сервис? Отличие от веб-сервера.

- Веб-служба, веб-сервис (web service) — идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами.

- Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (SOAP, XML-RPC и т. д.) и соглашениях (REST). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения.

Веб-сервер — это сервер, который отвечает за прием и обработку HTTP-запросов и отправку ответов пользователю (например, веб-страниц).

Веб-сервис — это приложение, которое предоставляет данные или функциональность другим приложениям через стандартизированные веб-протоколы (например, REST API или SOAP).

Что такое Web API? Назначение и применение.

Web API (Web Application Programming Interface) — это интерфейс программирования приложений, который позволяет двум различным программным системам обмениваться данными и функциональностью через интернет с использованием стандартных веб-протоколов, чаще всего HTTP.

Web API может возвращать данные в различных форматах, наиболее популярными из которых являются JSON и XML.

Веб-API используется для предоставления доступных для использования сервисов, таких как получение данных, выполнение операций или обмен информацией с другими приложениями.

Основные компоненты Web API:

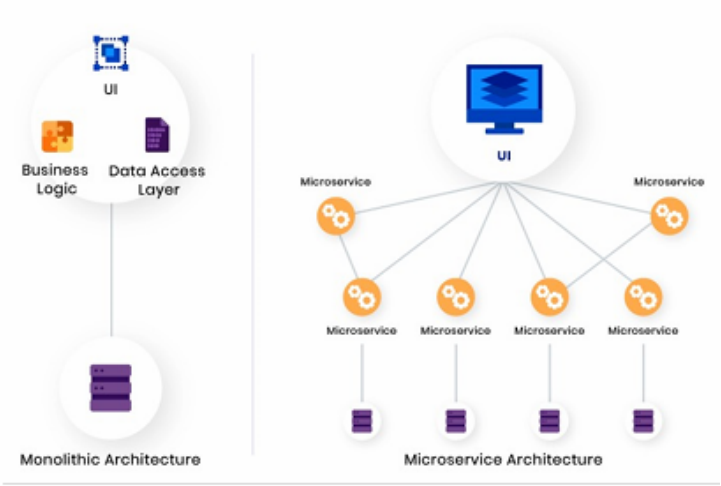
1. URL-адрес — это адрес, по которому можно получить доступ к функционалу API.
2. Методы HTTP — определяют действия, которые можно выполнить с помощью API: GET, POST, PUT, DELETE.
3. Формат данных — данные обычно передаются в формате JSON или XML.
4. Ответы и ошибки — API возвращает ответы, которые могут содержать как успешные результаты (например, данные), так и коды ошибок.

Применение Web API:

1. Мобильные приложения — для получения данных с серверов, например, о погоде или новостях.
2. Веб-сайты — для интеграции с внешними сервисами, такими как платёжные системы или социальные сети.
3. Интернет вещей (IoT) — для обмена данными между устройствами и приложениями.
4. Автоматизация процессов — для интеграции бизнес-систем, например, обработки заказов или управления инвентарём.
5. Реальное время — для получения обновлений в реальном времени, например, информации о состоянии оборудования или событий.

Микросервисная архитектура. Отличия от монолитной архитектуры.

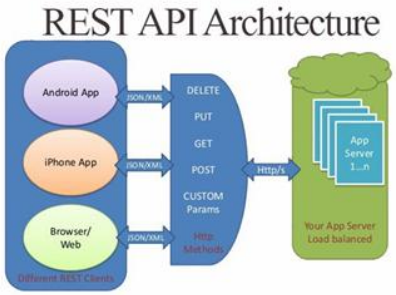
- **Микросервисная архитектура** — вариант сервис-ориентированной архитектуры программного обеспечения на основе взаимодействия небольших, слабо связанных и легко изменяемых модулей - микросервисов
- Теперь каждая система состоит из множества отдельных сервисов, выполняющих самостоятельную бизнес-функцию
- Например можно разделить услуги, заявки и авторизацию
- Получила распространение в середине 2010-х годов в связи с развитием практик гибкой разработки и DevOps



Характеристика	Микросервисная архитектура	Монолитная архитектура
Структура	Разделение на независимые сервисы	Всё приложение в одном целом
Масштабирование	Можно масштабировать каждый сервис отдельно	Масштабируется всё приложение целиком
Развитие и обновление	Легко обновлять или заменять отдельные сервисы	Требуется обновлять всё приложение целиком
Технологии	Возможность использовать разные технологии для разных сервисов	Одинаковая технология для всего приложения
Устойчивость к сбоям	Система более устойчива к сбоям: сбой в одном сервисе не влияет на другие	Сбой в одном компоненте может повлиять на всё приложение
Сложность	Более высокая сложность управления и координации сервисов	Простота на старте, но высокая сложность с ростом приложения
Тестирование	Требует тестирования каждого сервиса отдельно	Легче тестировать как целое приложение

Перечислите требования REST, опишите их.

- REST - Representational State Transfer. Набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать
- RESTful – веб-службы не нарушающие ограничений
- **Ограничения (знать!)**: клиент-сервер, отсутствие состояния, кэширование, единообразие интерфейса, слои, код по требованию



Статусность: запросы от клиента должны быть независимы, без сохранения состояния на сервере.

Клиент-серверная архитектура: чёткое разделение между клиентом и сервером.

Отсутствие состояния передачи (Cacheability): ответы должны поддерживать кэширование.

Единый интерфейс: стандартизированные API, методы и структуры данных.

Слои системы: возможность использования промежуточных слоёв для управления, безопасности и масштабируемости.

Код по требованию: сервер может передавать клиенту исполняемый код (необязательно).

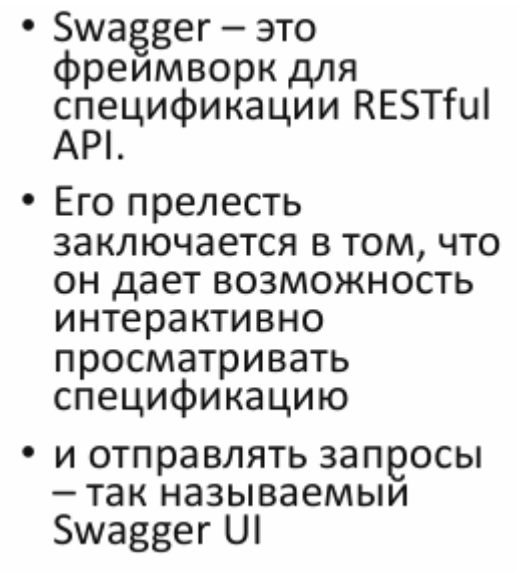
Что такое RPC? Варианты RPC и их отличия.

- RPC - remote procedure call. Удалённый вызов процедур
- Реализация RPC-технологии включает два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур для необъектных RPC)

- JSON-RPC
- XML RPC
- gRPC
- SOAP

Тип RPC	Формат передачи данных	Протокол передачи	Поддержка языков	Преимущества	Недостатки
XML-RPC	XML	HTTP	Широкая	Простота, совместимость	Высокий объём данных, медленная обработка
JSON-RPC	JSON	HTTP, WebSocket	Широкая	Лёгкость и компактность	Отсутствие встроенной безопасности
gRPC	Protocol Buffers (Protobuf)	HTTP/2	Широкая	Высокая производительность, асинхронность	Требует дополнительных настроек и компиляции
SOAP	XML	HTTP, SMTP	Широкая	Строгая спецификация безопасности	Большой объём данных, сложность настройки

Что такое Swagger? Назначение и использование.



Использование:

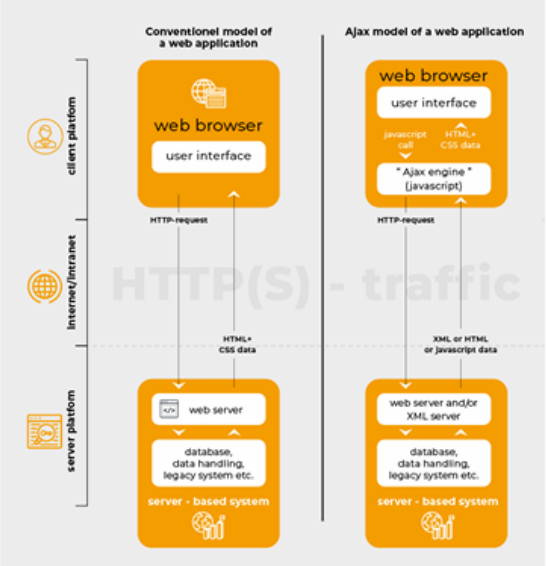
1. Документирование API: Swagger помогает создать актуальную и машиночитаемую документацию для API.
2. Интерактивное тестирование: Swagger UI позволяет пользователям тестировать API напрямую через веб-интерфейс.
3. Генерация кода: С помощью Swagger Codegen можно автоматически генерировать клиентские и серверные библиотеки для различных языков программирования.
4. Управление API: Swagger Hub позволяет управлять версиями API и совместной работой над проектами.

Что такое AJAX? Схема работы и назначение.

AJAX

- **AJAX**, Ajax (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.
- В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

- **JSON-RPC** (JavaScript Object Notation Remote Procedure Call — JSON-вызов удалённых процедур) — протокол удалённого вызова процедур, использующий JSON для кодирования сообщений.



Назначение JSON и XML. Примеры и отличия.

JSON — это легковесный формат обмена данными, который легко читается человеком и машиной. Он используется для передачи структурированных данных в основном в веб-приложениях, API и веб-сервисах.

Основное назначение:

- Обмен данными между клиентом и сервером.
- Хранение и передача структурированных данных.
- Используется в большинстве современных API, включая RESTful API.

XML — это расширяемый метаязык разметки, который используется для представления данных в виде иерархической структуры. XML предоставляет больше возможностей для описания данных и их структуры с помощью тегов и атрибутов.

Основное назначение:

- Представление данных с формальной схемой.
- Использование в старых веб-сервисах (SOAP).
- Применяется в ситуациях, где важна строгая иерархия и возможность описания метаданных (атрибутов).

```json
{
  "name": "Alice",
  "age": 25,
  "isStudent": false,
  "address": {
    "street": "Main St.",
    "city": "Wonderland"
  },
  "phoneNumbers": ["123-456-7890", "987-654-3210"]
}
```

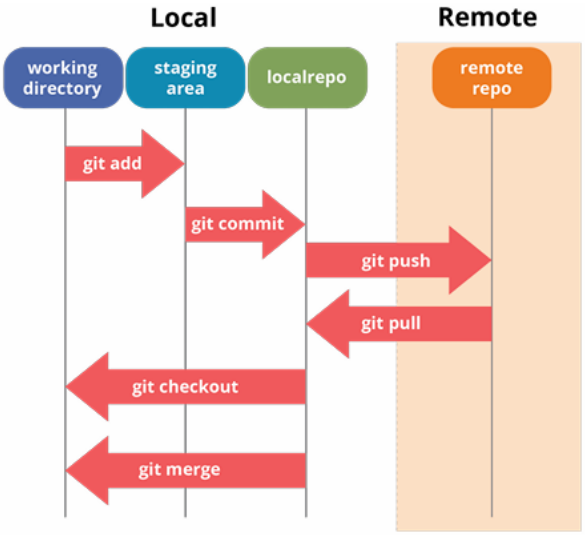
```xml
<person>
  <name>Alice</name>
  <age>25</age>
  <isStudent>false</isStudent>
  <address>
    <street>Main St.</street>
    <city>Wonderland</city>
  </address>
  <phoneNumbers>
    <phone>123-456-7890</phone>
    <phone>987-654-3210</phone>
  </phoneNumbers>
</person>
```

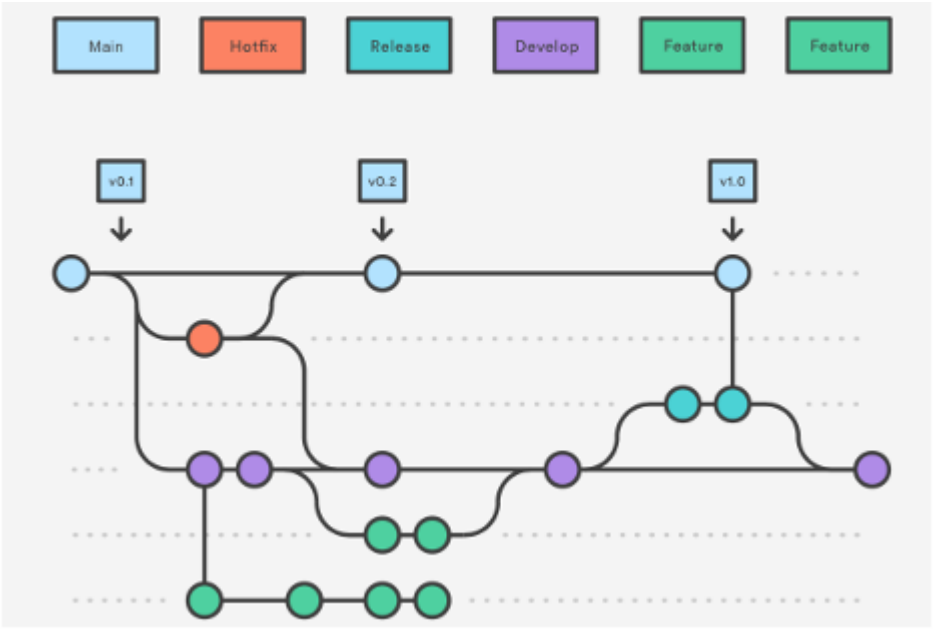
Характеристика	JSON	XML
Читаемость человеком	Легко читается, краткий формат.	Читаемость хуже, из-за большого объема тегов.
Сложность структуры	Простой и минималистичный формат.	Более сложный, поддерживает схемы и атрибуты.
Размер данных	Меньше по объему данных (меньше тегов).	Более громоздкий из-за дополнительных тегов.
Поддержка типов данных	Поддерживает строки, числа, массивы, булевы.	Не поддерживает типы данных напрямую (только строки).
Использование	В основном для веб-приложений, API, фронтенда.	Используется в старых системах, для строгих схем (SOAP).
Поддержка метаданных	Нет явной поддержки метаданных.	Легко добавлять атрибуты и описания в тегах.
Парсинг и обработка	Быстрое и простое парсирование (например, в JavaScript).	Требует больше ресурсов для парсинга.
Популярность	Широко используется в современных веб-технологиях.	Используется в более старых технологиях и системах.

Что такое git? Опишите схему работы с ветками GitHub.

Git

- Git – распределенная система управления версиями
- Позволяет хранить несколько версий одного и того же документа





Методология разработки Agile. Состав IT команды.

Состав IT команды:

- Product Owner
- Agile Coach
- Development Team
- DevOps Engineers
- UI/UX дизайнеры
- QA-инженеры (тестировщики)

Перечислите основные диаграммы UML и их назначение.

Диаграмма развёртывания, диаграмма последовательности.

Диаграмма развертывания (Deployment Diagram)

- Назначение: Показывает, как компоненты системы развёртываются на физических устройствах.
- Использование: Полезна для определения архитектуры развертывания, серверов, баз данных и сетевых узлов.

Диаграмма последовательности (Sequence Diagram)

- Назначение: Показывает взаимодействие объектов в системе в последовательности во времени.
- Использование: Используется для моделирования сценариев взаимодействия между объектами на протяжении определенного времени.

Что такое Web реального времени? Что такое WebSocket?

Web реального времени — это концепция, при которой веб-приложения или веб-сайты могут передавать данные и обновления пользователю без необходимости обновления страницы или запроса от клиента. Такой подход позволяет обеспечивать мгновенные или почти мгновенные отклики на события, что особенно полезно в приложениях, где необходимо постоянно обновлять данные в реальном времени.

WebSocket — это протокол связи, предназначенный для двусторонней передачи данных между клиентом (например, браузером) и сервером через одно постоянное соединение. Он позволяет устанавливать постоянное соединение, по которому данные могут передаваться в обоих направлениях, то есть и от клиента к серверу, и от сервера к клиенту, без необходимости повторного открытия соединения.

Укажите отличия XmlHttpRequest и fetch. Приведите примеры.

Характеристика	XMLHttpRequest	fetch
Синтаксис	Сложный и многословный, использует коллбэки.	Более современный, использует промисы и async/await.
Поддержка промисов	Не поддерживает промисы по умолчанию. Нужно использовать коллбэки или сторонние библиотеки для работы с асинхронностью.	Работает с промисами по умолчанию. Это упрощает работу с асинхронными запросами.
Поддержка обработки ошибок	Ошибки нужно обрабатывать вручную (например, через `onerror` и `onreadystatechange`).	Легче обрабатывать ошибки с помощью `catch` на уровне промисов.
Поддержка CORS	Требует больше настроек для работы с CORS.	Простой и удобный способ работы с CORS (некоторые настройки могут быть сделаны через параметры fetch).
Поддержка потоков	Не поддерживает потоки и загрузку данных по частям.	Поддерживает загрузку данных по частям с использованием `ReadableStream`.
Обработка JSON	Требуется явная обработка ответа как JSON (например, через `JSON.parse`).	Простой способ обработки JSON с помощью `response.json()`.
Методы отправки запроса	Использует методы `open()` и `send()` для отправки запроса.	Использует метод `fetch()`, который является глобальной функцией.
Поддержка запросов с телом	Можно использовать, но требует дополнительных шагов для настройки.	Легко поддерживает запросы с телом, например, через метод `fetch()`.
Размер API	Больший и старый API.	Меньший и современный API.

```
// Создаем объект XMLHttpRequest
let xhr = new XMLHttpRequest();

// Настройка запроса
xhr.open('GET', 'https://api.example.com/data', true);

// Определяем функцию обработки ответа
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        // Преобразуем ответ в JSON
        let data = JSON.parse(xhr.responseText);
        console.log(data);
    }
};

// Отправляем запрос
xhr.send();
```

```
// Используем fetch для отправки GET-запроса
fetch('https://api.example.com/data')
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();  // Преобразуем ответ в JSON
    })
    .then(data => {
        console.log(data);  // Выводим данные
    })
    .catch(error => {
        console.error('There was a problem with the fetch operation:', error);
    });
```

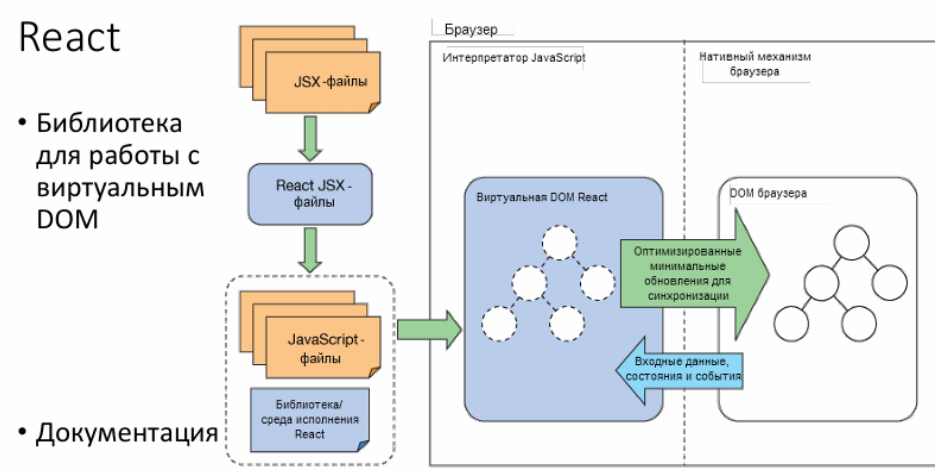
Перечислите отличия Axios от fetch. Приведите примеры.

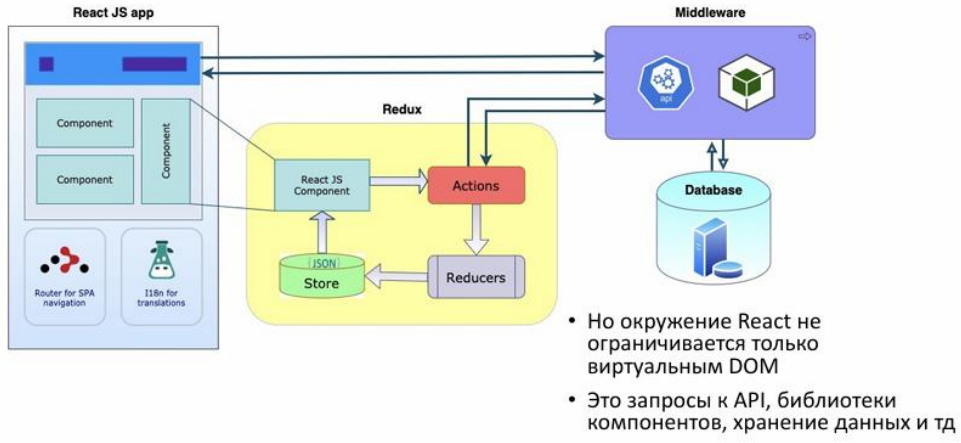
Характеристика	Axios	fetch
Поддержка браузеров	Поддерживает старые браузеры, включая Internet Explorer (IE11 и выше).	Не поддерживает старые браузеры (например, IE) без полифилов.
Работа с JSON	Автоматически преобразует ответ в формат JSON, если ответ в этом формате.	Требуется вручную вызвать `.json()` для преобразования ответа в JSON.
Обработка ошибок HTTP	Автоматически выбрасывает ошибку при получении ответов с кодами ошибок (например, 404, 500).	Не выбрасывает ошибку для кодов HTTP, если статус ответа не 2xx. Нужно вручную проверять `response.ok`.
Поддержка отмены запросов	Легко поддерживает отмену запросов через `CancelToken`.	Поддерживает отмену запросов с помощью `AbortController`, но это требует дополнительной настройки.
Поддержка отправки данных	Легко отправлять данные в запросах с телом (например, JSON, формы, файлы).	Также поддерживает отправку данных, но требует явного указания метода и типа контента.
Работа с заголовками	Легче управлять заголовками, например, для авторизации или сессий.	Для работы с заголовками нужно использовать `headers` в настройках.
Интерсепторы	Поддерживает интерсепторы запросов и ответов для глобальной обработки.	Не поддерживает встроенные интерсепторы, но можно реализовать самостоятельно.
Размер библиотеки	Большая библиотека по сравнению с fetch (около 12KB).	Стандартный API браузера, без дополнительных зависимостей.
Поддержка отправки запросов с куки	Поддерживает отправку запросов с куки по умолчанию (если используется правильная конфигурация).	Не отправляет куки по умолчанию в кросс-доменных запросах (нужно установить `credentials: 'include'`).

```javascript
// Установка Axios (если используете через npm)
import axios from 'axios';

// Отправка GET-запроса
axios.get('https://api.example.com/data')
  .then(response => {
    console.log('Data:', response.data); // Данные ответа
  })
  .catch(error => {
    console.error('Error:', error); // Обработка ошибок
  });

// Отправка POST-запроса с телом запроса
axios.post('https://api.example.com/data', {
    name: 'John',
    age: 30
  })
  .then(response => {
    console.log('Data:', response.data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

```javascript
// Используем fetch для отправки GET-запроса
fetch('https://api.example.com/data')
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();  // Преобразуем ответ в JSON
    })
    .then(data => {
        console.log(data);  // Выводим данные
    })
    .catch(error => {
        console.error('There was a problem with the fetch operation:', error);
    });
```

Что такое React? Что такое компонент, его состояния и свойства.



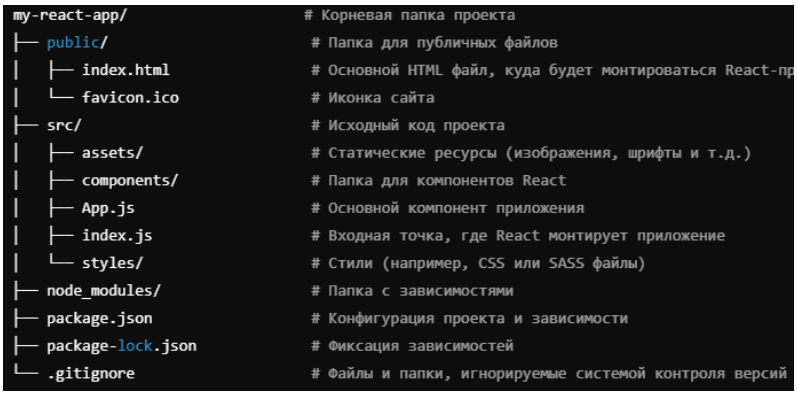


Компонент в React — это независимая, переиспользуемая часть интерфейса, которая может содержать как UI, так и логику. Компоненты могут быть функциональными или классовыми. Каждый компонент может иметь свое состояние (state) и свойства (props).

Props (свойства): Это данные, которые компонент получает от родительского компонента. Свойства являются неизменяемыми внутри компонента. Они позволяют передавать информацию и конфигурацию от одного компонента к другому.

State (состояние): Это данные, которые управляются внутри компонента и могут изменяться в процессе работы приложения. Когда состояние компонента изменяется, React перерисовывает компонент, чтобы отобразить новые данные.

Структура React проекта. Назначение Babel и WebPack.



Babel — это транспайлер JavaScript, который позволяет использовать последние возможности языка (например, ES6 и JSX) в браузерах, которые могут не поддерживать их.

Основные задачи Babel:

1. Преобразование JSX в JavaScript: JSX — это синтаксис, который позволяет писать код, похожий на HTML, в JavaScript. Babel преобразует JSX в обычные JavaScript функции React.createElement().
2. Транспиляция ES6+ в ES5: Babel может преобразовывать более новые версии JavaScript в старые версии, чтобы поддерживать старые браузеры.

Webpack — это модульный сборщик, который упрощает сборку, управление зависимостями и оптимизацию фронтенд-ресурсов. Webpack помогает с процессом компиляции, упаковки и минификации всех файлов проекта.

Основные задачи Webpack:

1. Модульная система: Webpack позволяет работать с различными типами модулей, такими как JavaScript, CSS, изображения и шрифты, и обрабатывать их как отдельные модули. Он объединяет эти модули в единый выходной файл или несколько файлов.
2. Транспиляция и минификация: С помощью плагинов и лоадеров Webpack может транспилировать файлы, а также минифицировать код, уменьшая его размер для продакшн-сборки.
3. Генерация оптимизированных бандлов: Webpack генерирует оптимизированные выходные файлы, которые содержат только необходимые части кода, улучшая производительность.

Жизненный цикл React компонента.

Жизненный цикл приложения

- **1: Монтирование**

компонент запускает getDerivedStateFromProps(), потом запускается render(), возвращающий JSX. React «монтируется» в DOM

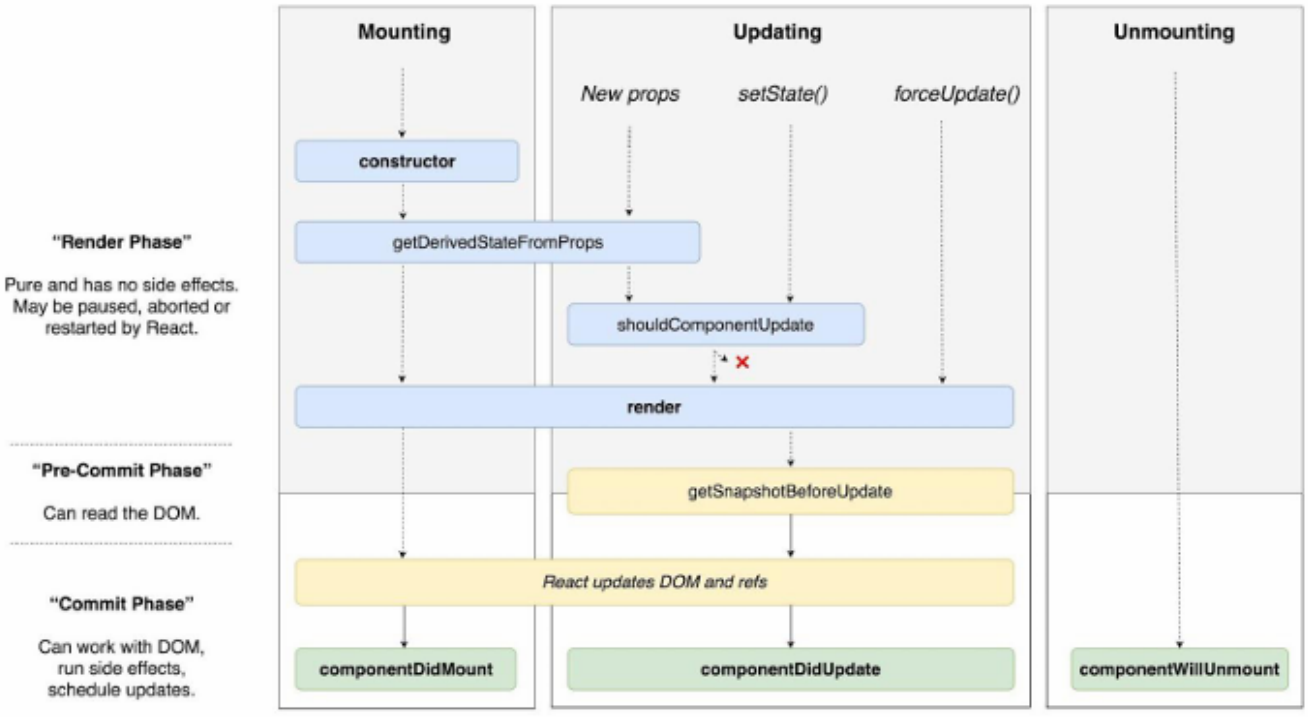
- **2: Обновление**

Данный этап запускается во время каждого изменения состояния либо свойств

- **3: Размонтирование**

React выполняет запуск componentWillUnmount() непосредственно перед удалением из DOM

Методы жизненного цикла компонента



Назначение хуков useState и useEffect.

Хук useState используется для создания и управления состоянием внутри функциональных компонентов. Он позволяет компоненту хранить данные, которые могут изменяться со временем, и обновлять интерфейс при изменении этих данных.

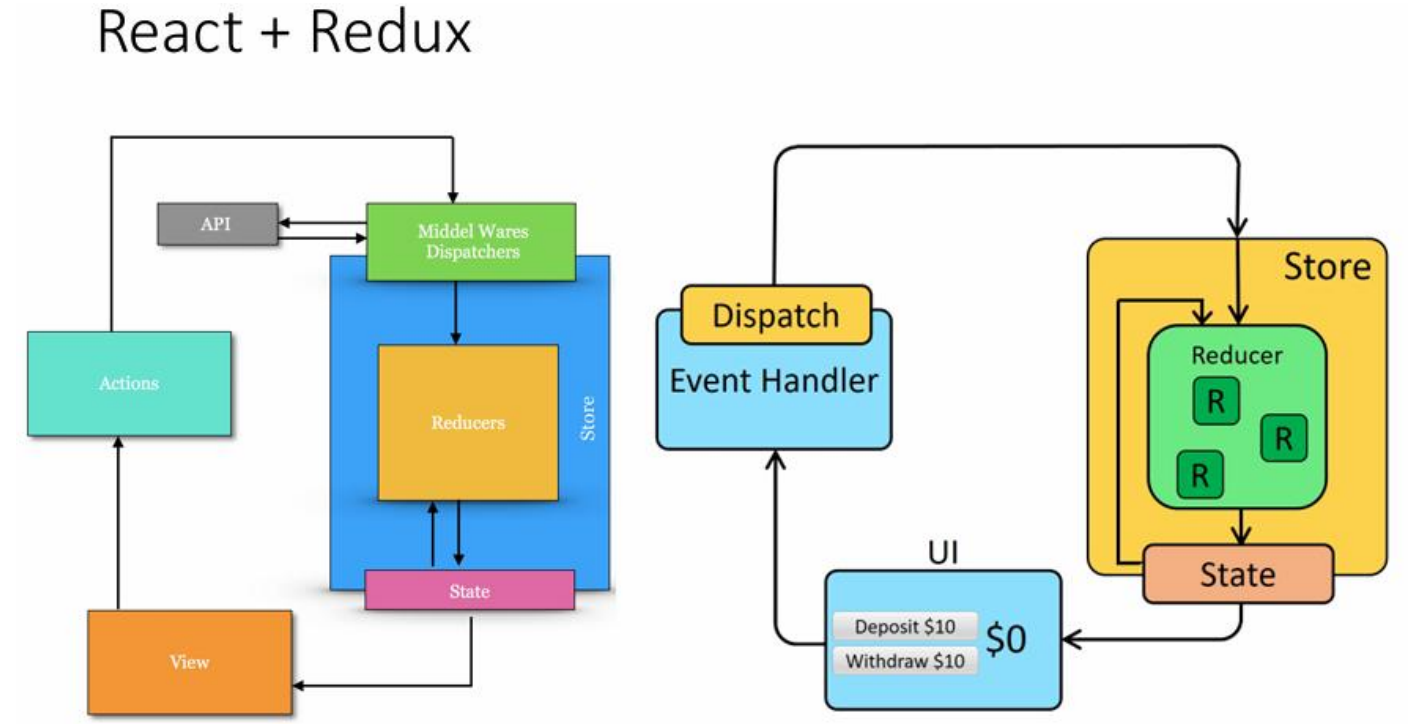
Хук useEffect используется для выполнения побочных эффектов в функциональных компонентах. Он позволяет выполнять операции, такие как запросы к API, подписки на события, манипуляции с DOM или таймеры, которые не связаны напрямую с рендером компонента, но должны быть выполнены после него.

Назначение хуков useContext и useReducer.

Хук useContext позволяет компоненту получать доступ к значению контекста (context) без необходимости использовать Consumer компонент. Это упрощает передачу данных через дерево компонентов, особенно когда необходимо делиться состоянием между удалёнными компонентами.

Хук useReducer используется для более сложного управления состоянием, особенно когда состояние зависит от предыдущего состояния или когда обновление состояния требует нескольких шагов. Это похоже на useState, но с большими возможностями для более сложных состояний.

Опишите схему работы менеджера состояний Redux.



Инициализация состояния:

- Когда приложение запускается, состояние приложения (store) инициализируется.

Действие (Action):

- Когда пользователь или приложение вызывает действие, это действие отправляется в Redux через функцию dispatch.

Редуктор (Reducer):

- Когда действие отправляется, редуктор получает текущее состояние и действие.
- Редуктор решает, как изменить состояние, основываясь на типе действия.
- Редуктор возвращает новое состояние, которое отличается от предыдущего только теми изменениями, которые были выполнены в ответ на действие.

Обновление состояния:

- После выполнения редуктора, новое состояние сохраняется в store. Это приводит к обновлению всех подписанных компонентов, которым необходимо это новое состояние.

Подписка и обновление UI:

- Компоненты, которые подписаны на store через функцию connect, получают новое состояние и перерисовываются с обновленными данными.

Опишите работу Redux на диаграмме последовательности.

Инициализация состояния (Store):

- Когда приложение запускается, создается хранилище состояния (store), которое инициализируется редуктором, например, с начальным состоянием.

Отправка действия (Action Dispatch):

- Когда пользователь выполняет действие (например, кликает на кнопку или взаимодействует с интерфейсом), отправляется action через dispatch.

Обработка действия в редукторе (Reducer):

- Store передает действие (action) в редуктор.
- Редуктор проверяет тип действия и обновляет состояние на основе этого действия. Редуктор не изменяет текущее состояние, а возвращает новое.

Обновление состояния (State Update):

- Редуктор возвращает новое состояние.
- Хранилище обновляет состояние, сохраняет его в своем внутреннем объекте.

Подписка и рендер (React Components):

- Компоненты, подписанные на изменения состояния через connect или useSelector (в случае с React), получают новое состояние.
- Компоненты перерисовываются с новым состоянием.

Какие параметры передаются при создании Store? Их назначение.

При создании store в Redux используется функция createStore, которая принимает несколько параметров для настройки хранилища состояния.

reducer (редуктор):

- Назначение: Это обязательный параметр, который представляет собой функцию, отвечающую за обновление состояния приложения на основе поступающих действий (actions).
- Редуктор принимает два аргумента:
  - state — текущее состояние приложения.
  - action — объект действия, который был отправлен (с типом и данными).
- Редуктор возвращает новое состояние, которое будет сохранено в хранилище.

preloadedState (предварительное состояние):

- Назначение: Этот параметр позволяет задать начальное состояние хранилища.
- Это полезно, если вы хотите, чтобы состояние было задано не по умолчанию (например, при серверном рендеринге или восстановлении состояния из локального хранилища).
- Если этот параметр не передан, состояние по умолчанию будет пустым или задано в редукторе.

enhancer (пауэр-расширитель):

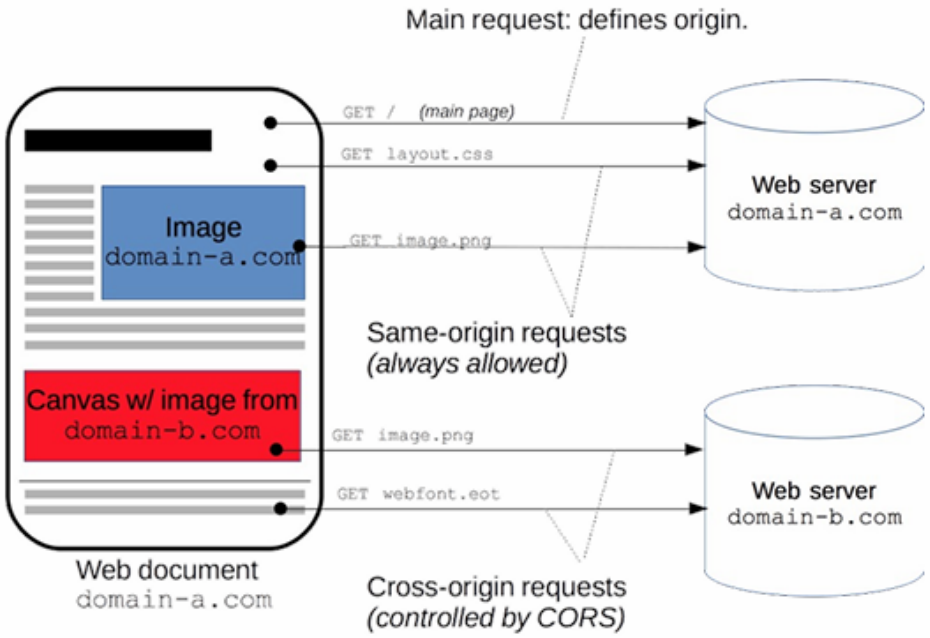
- Назначение: Этот параметр используется для добавления промежуточных функций, которые могут расширить возможности store.
- Обычно это используется для интеграции с Redux DevTools, middleware (например, Redux Thunk, для работы с асинхронными действиями), или других расширений.
- Это необязательный параметр, но его использование позволяет улучшить функциональность и удобство работы с Redux.
- Вы можете использовать applyMiddleware или compose для настройки enhancer.

Что такое Cors? Укажите варианты решения.

- CORS - мы получили страницу с одного домена, а запросы отправляем на другой

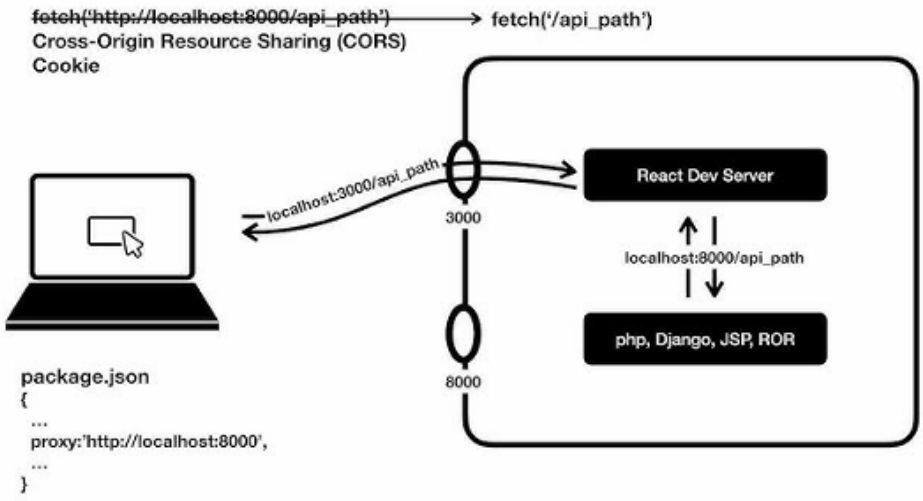
Как решить?

- CORS – заголовки на бекенде
- Проксирование через сервер фронтенда



- Одно из решений - отправляем запросы на напрямую в веб-сервис, а проксируем через наш сервер фронтенда

- Похоже на prod решение при проксировании через Nginx



Что такое Redis? Его назначение и варианты применения.

Redis — это высокопроизводительная система управления базами данных, которая работает в памяти (in-memory) и используется в качестве кэша, брокера сообщений и базы данных NoSQL. Redis сохраняет данные в памяти для очень быстрых операций чтения и записи, что делает его идеальным для приложений, требующих высокой скорости обработки данных.

Назначение:

- Кеширование данных
- Система очередей и брокер сообщений
- Поддержка структур данных
- Сохранение данных на диск
- Секционирование данных
- Реализация механизмов блокировки

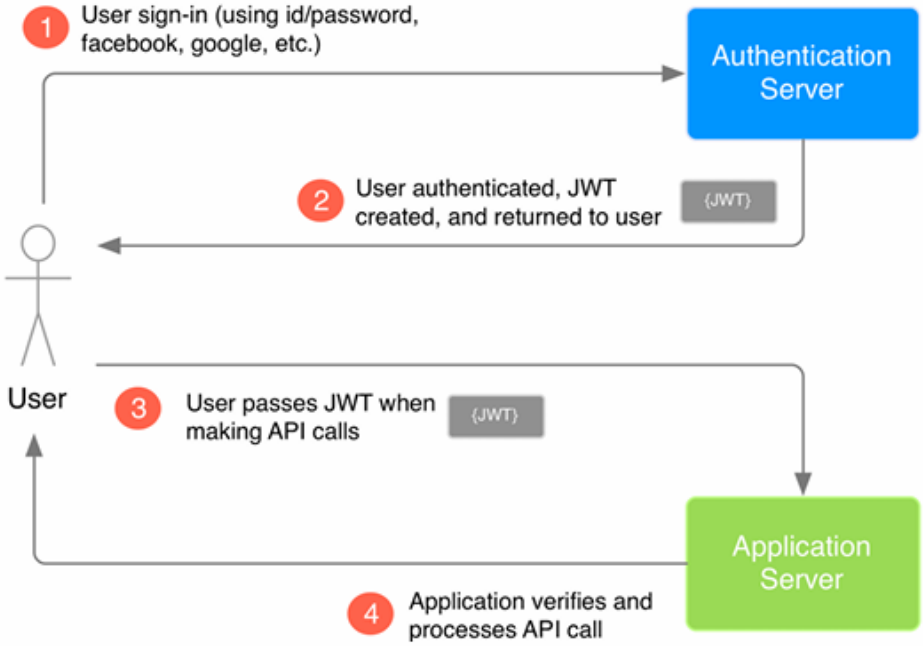
Варианты применения:

- Кэширование
- Очереди сообщений
- Подсчёт и мониторинг
- Таймера и сессии
- Реализация сложных структур данных
- Распределённые блокировки и синхронизация
- Веб-сокеты и push-уведомления

Опишите схему авторизации с помощью JWT.

JWT

- **JSON Web Token**
- Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.
- Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения подлинности аккаунта.



Опишите схему авторизации с помощью сессий.

Авторизация с использованием сессий — это стандартный способ хранения информации о пользователе на сервере в ходе его сессии в веб-приложении. Этот процесс включает несколько шагов, с помощью которых можно подтвердить личность пользователя и отслеживать его действия без необходимости повторной аутентификации на каждом запросе.

Пользователь вводит данные для входа (логин и пароль):

- Пользователь открывает страницу входа и вводит свои учетные данные. Данные передаются на сервер с помощью HTTP-запроса.

Сервер проверяет данные пользователя:

- Сервер получает запрос с данными пользователя. Сервер проверяет, существует ли такой пользователь в базе данных и совпадают ли введенные данные с сохраненными.

Генерация уникальной сессии:

- Если аутентификация прошла успешно, сервер генерирует уникальный идентификатор сессии.

Отправка идентификатора сессии в браузер пользователя:

- Сервер отправляет клиенту (браузеру) cookie с уникальным идентификатором сессии.

Клиент отправляет идентификатор сессии с каждым запросом:

- Каждый последующий запрос от клиента будет содержать cookie с идентификатором сессии. Сервер извлекает этот идентификатор из cookie и использует его для нахождения данных о сессии в хранилище.

Проверка сессии на сервере:

- После каждого действия сервер извлекает session_id из cookie и проверяет, существует ли сессия с таким идентификатором. Если сессия найдена и она действительна, сервер позволяет пользователю выполнить требуемое действие. Иначе сервер может запросить повторную аутентификацию.

Окончание сессии:

- Когда пользователь выходит из системы, сервер уничтожает сессию и удаляет соответствующие данные с сервера.

Что такое авторизация и аутентификация? Укажите варианты авторизации и их отличия.

Аутентифика́ция (*authentication*) — процедура проверки подлинности, например:

- проверка подлинности пользователя путём сравнения введённого им пароля (для указанного логина) с паролем, сохранённым в базе данных пользовательских логинов;
- подтверждение подлинности электронного письма путём проверки цифровой подписи письма по открытому ключу отправителя;
- проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла.

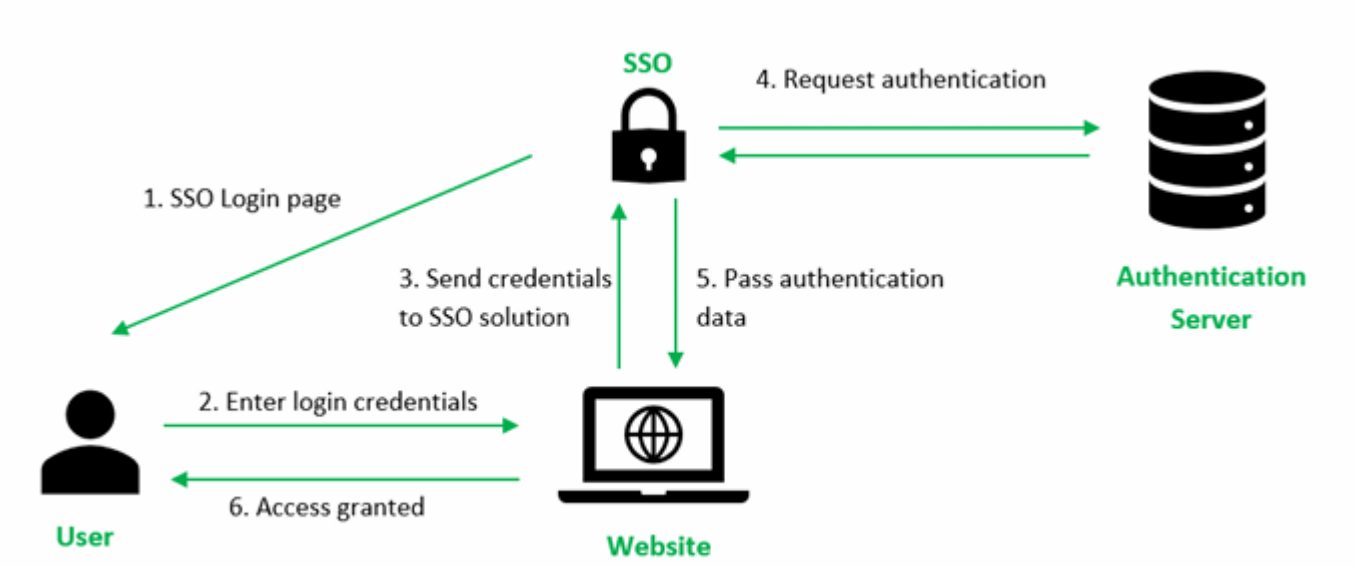
- **Авториза́ция** (*authorization* «разрешение; уполномочивание») — предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

- Авторизация производит контроль доступа к различным ресурсам системы в процессе работы легальных пользователей после успешного прохождения ими аутентификации.

Вариант авторизации	Описание	Преимущества	Недостатки
Basic Authentication	Логин и пароль передаются в заголовке HTTP-запроса (Base64).	Простота реализации, широко поддерживается.	Не безопасно без HTTPS, передача пароля в открытом виде.
Token-based Authentication (Токен-ориентированная)	Пользователь получает токен (например, JWT), который используется для дальнейшей авторизации.	Безопасность, токен передается вместо пароля.	Необходима реализация хранения и управления сроком действия токенов.
OAuth	Протокол для авторизации через сторонние сервисы (например, Google, Facebook).	Удобство для пользователей, интеграция с другими сервисами.	Зависимость от сторонних сервисов, сложность настройки.
SAML (Security Assertion Markup Language)	Протокол для обмена данными аутентификации и авторизации, используется в основном для единого входа (SSO).	Централизованное управление доступом, безопасность.	Сложность настройки и интеграции, используется в основном в корпоративных системах.
Multi-factor Authentication (MFA)	Требуется несколько факторов (например, пароль + SMS-код или приложение аутентификации).	Повышенная безопасность, защита от взлома.	Усложнение процесса входа, возможные проблемы с доступом в случае потери второго фактора.

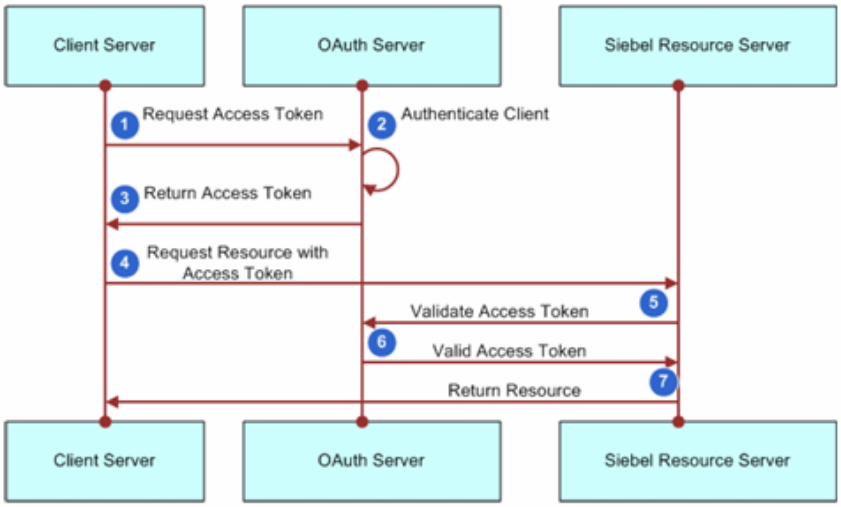
Что такое SSO? Схема работы.

- **Технология единого входа** (Single Sign-On) — технология, при использовании которой пользователь переходит из одного раздела портала в другой, либо из одной системы в другую, не связанную с первой системой, без повторной аутентификации.



Протокол OAuth. Схема работы.

- **OAuth** — открытый протокол (схема) авторизации, обеспечивающий предоставление третьей стороне ограниченный доступ к защищённым ресурсам пользователя.
- Без передачи ей (третьей стороне) логина и пароля



Отличия мобильных и веб-приложения. Языки и технологии для разработки мобильных приложений.

Критерий	Мобильные приложения	Веб-приложения
Платформа	Разрабатываются для конкретных мобильных операционных систем (iOS, Android).	Работают в браузере и доступны на любой платформе с интернет-соединением.
Установка	Необходима установка через магазины приложений (Google Play, App Store).	Не требует установки, доступно через URL в браузере.
Доступ к устройствам	Могут напрямую взаимодействовать с функционалом устройства (камера, GPS, сенсоры, и т.д.).	Ограничены функциональностью браузера, доступ к устройствам ограничен (через Web API).
Производительность	Обычно быстрее и эффективнее, так как они работают непосредственно на устройстве.	Могут быть медленнее, так как зависят от производительности браузера и скорости интернета.
Обновления	Требуют обновлений через магазины приложений.	Обновления происходят автоматически на сервере, пользователи получают новую версию сразу.
Доступность офлайн	Могут работать офлайн, если приложение поддерживает такую функциональность.	Требуют постоянного интернет-соединения для полноценной работы.
Опыт пользователя	Чаще всего имеют более глубокую и персонализированную интеграцию с устройством.	Зависит от качества веб-разработки, часто менее интерактивные, чем мобильные приложения.
Разработка	Требуется знание специфических языков и SDK для каждой платформы (например, Swift для iOS, Kotlin для Android).	Разработка чаще всего использует одни и те же технологии для всех платформ (HTML, CSS, JavaScript).

IOS: Swift, Objective-C, Xcode

Android: Kotlin, Java, Android Studio

Кросс-платформенные технологии: React Native, Flutter, Xamarin, Ionic

Гибридные мобильные приложения: Apache Cordova, Electron

Что такое pwa? Отличия от других вариантов приложений.

PWA (Progressive Web App) — это тип веб-приложений, который использует современные веб-технологии для предоставления пользователям качественного опыта, схожего с нативными мобильными приложениями. PWA работает как веб-страница, но имеет возможность работать в офлайн-режиме, отправлять push-уведомления и быть установленным на устройстве пользователя, как обычное мобильное приложение.

Характеристика	PWA	Нативное мобильное приложение	Гибридное приложение
Платформа	Работает в браузере на всех платформах (Android, iOS, Windows и другие).	Требуется отдельная разработка для каждой платформы (iOS, Android).	Требуется отдельная разработка для каждой платформы, но код может быть общим.
Установка	Устанавливается с веб-сайта (без магазинов приложений), добавляется на главный экран.	Требуется установка через магазин приложений (Google Play, App Store).	Требуется установка через магазин приложений.
Интернет	Работает в офлайн-режиме благодаря сервис-воркерам.	Могут работать в офлайн-режиме, зависит от реализации.	Могут работать в офлайн-режиме, но зависят от реализации.
Доступ к устройствам	Ограничен возможностями браузера (например, доступ к камере, GPS через Web API).	Полный доступ ко всем функциональным возможностям устройства.	Может быть ограничен, хотя и обеспечивает доступ к нативным функциям через плагины.
Обновления	Обновляется автоматически через сервер, без необходимости для пользователя.	Требует загрузки обновлений через магазины приложений.	Обновления через магазины приложений, иногда с использованием кросс-платформенных решений.
Производительность	Высокая производительность благодаря кэшированию и оптимизации, но может быть ниже, чем у нативных приложений.	Обычно высокая производительность, так как приложения работают напрямую с операционной системой.	Может быть ниже по производительности, чем у нативных приложений.
Push-уведомления	Да, поддерживает push-уведомления в браузере.	Да, поддерживает push-уведомления.	Да, поддерживает push-уведомления.

Плюсы и минусы разработки на React Native.

Преимущество	Описание
Кросс-платформенность	React Native позволяет писать один код, который будет работать и на iOS, и на Android, что сокращает время и ресурсы на разработку.
Быстрая разработка	С использованием библиотеки React и возможности "горячей перезагрузки" (Hot Reloading) разработка становится более быстрой и удобной.
Большое сообщество	Огромное сообщество разработчиков, активная поддержка и множество готовых решений и библиотек, которые ускоряют процесс разработки.
Использование нативных компонентов	React Native предоставляет доступ к нативным API и позволяет использовать нативные компоненты для повышения производительности и гибкости.
Поддержка JavaScript	Разработка ведется на JavaScript, что является популярным и широко используемым языком программирования, что облегчает найм разработчиков.
Реализация нативных функций	Возможность интеграции с нативными модулями для использования специфичных для платформы функций (например, камеры, GPS, сенсоров и т.д.).
Поддержка сторонних библиотек	Можно интегрировать сторонние библиотеки и SDK для реализации функционала, который не предоставляется стандартно в React Native.
Быстрое тестирование	Благодаря возможности работы в эмуляторах и на реальных устройствах, тестирование и отладка происходят гораздо быстрее.

Недостаток	Описание
Проблемы с производительностью	Хотя React Native хорош для большинства приложений, для очень сложных, ресурсоемких приложений, таких как игры или 3D-приложения, может быть хуже, чем нативная разработка.
Не полная нативная поддержка	Некоторые нативные функции могут быть недоступны или ограничены, особенно для более новых возможностей платформ. Для их использования может потребоваться написание дополнительных нативных модулей.
Зависимость от сторонних библиотек	Часто для реализации сложных или нестандартных функций требуется использование сторонних библиотек, которые могут не быть идеальными или поддерживаться с ошибками.
Производительность на слабых устройствах	На старых или слабых устройствах производительность может быть ниже по сравнению с нативными приложениями.
Сложность обновлений и совместимости	Некоторые обновления React Native могут нарушить совместимость с ранее используемыми версиями, что требует дополнительных усилий по обновлению и тестированию.
Проблемы с многозадачностью	Для сложных многозадачных операций React Native может не всегда работать так эффективно, как нативные решения. Например, многозадачность или фоновые операции могут требовать написания нативного кода.
Отсутствие поддержки всех нативных API	Несмотря на возможность интеграции с нативными модулями, не все функции могут быть доступны в React Native, и иногда придется прибегать к нативной разработке.

Назначение фреймворков Electron и Tauri. Их отличия.

Electron — это фреймворк для разработки кросс-платформенных десктопных приложений с использованием веб-технологий (HTML, CSS, JavaScript). Он позволяет создавать приложения для Windows, macOS и Linux, используя единую кодовую базу.

- Технологии: Основой Electron являются Chromium (для рендеринга интерфейса) и Node.js (для работы с серверной логикой и доступом к файловой системе).
- Основное преимущество: Приложения, разработанные на Electron, могут использовать те же веб-технологии, что и веб-приложения, что облегчает процесс разработки, особенно для разработчиков, уже знакомых с фронтенд-технологиями.

Tauri — это фреймворк для создания кросс-платформенных десктопных приложений с использованием веб-технологий. В отличие от Electron, Tauri ориентирован на создание легковесных приложений с меньшими требованиями к ресурсам, используя веб-технологии только для интерфейса, а нативный код для реализации серверной логики.

- Технологии: В отличие от Electron, Tauri использует Rust для создания нативных компонентов приложения и WebView для рендеринга интерфейса. Это обеспечивает более легкие приложения с лучшей производительностью.
- Основное преимущество: Приложения, созданные с помощью Tauri, имеют гораздо меньший размер и потребляют меньше системных ресурсов, чем Electron-приложения.

Критерий	Electron	Tauri
Технологии	Chromium + Node.js	Rust + WebView (использует встроенный браузер)
Размер приложений	Обычно большие (от 50 МБ и выше)	Компактные (до 10 МБ)
Производительность	Может быть более ресурсоемким, из-за использования Chromium	Легковесные и быстрые благодаря использованию Rust
Кросс-платформенность	Windows, macOS, Linux	Windows, macOS, Linux
Безопасность	Поддерживает безопасную среду, но в некоторых случаях уязвим.	Более строгие меры безопасности благодаря Rust
Поддержка нативных функций	Хорошая поддержка нативных функций через Node.js и Electron API	Ограниченная поддержка нативных функций, но улучшенная через Rust
Размер сообщества	Большое сообщество и экосистема библиотек	Меньше по размеру, но активно растет
Использование	Удобно для приложений с богатым интерфейсом и высокой сложностью	Идеально для легких приложений с минимальными требованиями

Опишите этапы подхода DevOps. Назначение GitHub Pages.

GitHub Pages — это сервис для хостинга статических сайтов, который предоставляет GitHub. Он позволяет размещать веб-страницы непосредственно из репозиториев GitHub.

Назначение GitHub Pages

- Хостинг статических сайтов: GitHub Pages используется для хостинга простых статических сайтов и блогов. Это идеально подходит для сайтов с ограниченной динамичностью, например, персональных портфолио, документации, блогов и landing-страниц.
- Документация и портфолио: GitHub Pages часто используется для размещения документации проектов (например, для библиотек или инструментов, размещенных на GitHub), а также для создания личных или корпоративных сайтов.
- Легкость настройки: В отличие от традиционных сервисов хостинга, GitHub Pages не требует дополнительной настройки серверной части или базы данных, так как работает только с файловыми структурами.
- Интеграция с GitHub: Прямо из репозитория GitHub можно настроить автоматический деплой сайта при каждом коммите в основной ветке (например, через GitHub Actions или CI/CD).
- Поддержка пользовательских доменов: GitHub Pages позволяет использовать кастомные домены для сайта.