ЛЕКЦИЯ 3
ИНТЕГРАЦИЯ БАЗ ДАННЫХ В WEB-РАЗРАБОТКУ

Примеры web-приложений, использующих базы данных

Web-приложения с базами данных, позволяют хранить, обрабатывать и управлять данными, обеспечивая пользователям доступ к актуальной и структурированной информации. Рассмотрим различные типы Web-приложений, которые используют базы данных для своей работы.

1. Электронная коммерция (англ., e-commerce). Web-приложения электронной коммерции являются одним из самых популярных типов приложений, использующих базы данных. Они позволяют пользователям просматривать товары, добавлять их в корзину, оформлять заказы и производить оплату онлайн. Базы данных используются для хранения информации о продуктах, заказах, клиентах и транзакциях. Такие приложения также обеспечивают управление складскими запасами, расчет налогов и доставку.

2. Социальные сети. Web-приложения социальных сетей также широко используют базы данных для хранения информации о пользователях, их профилях, друзьях, сообщениях и других взаимодействиях. Пользователи могут создавать профили, публиковать контент, обмениваться сообщениями и просматривать активность своих друзей. Базы данных играют ключевую роль в обеспечении функциональности социальных сетей и сохранении информации.

3. Управление контентом (CMS – Content Management System). Web-приложения для управления контентом позволяют пользователям создавать, редактировать и публиковать содержимое web-сайта без необходимости знания программирования. Они используют базы данных для хранения текстового, графического и мультимедийного контента, а также для управления пользователями, правами доступа и другими функциями. Пример: WordPress – популярная CMS для создания блогов и web-сайтов, которая использует базу данных для хранения контента и настроек.

4. Аналитика и отчетность. Web-приложения для аналитики и отчетности используют базы данных для хранения и анализа больших объемов данных. Они позволяют пользователям созда-

вать отчеты, диаграммы, графики и другие визуальные способы представления информации на основе данных из базы. Такие приложения широко используются в бизнесе для принятия стратегических решений на основе аналитики. Пример: Google Analytics – сервис аналитики от Google, который использует базу данных для хранения данных о трафике на сайтах, поведении пользователей и других метриках. Еще пример: DB Engeene – сайт, определяющий популярность тех или иных СУБД, о которых встречаются любые упоминания в сети Интернет.

5. Управление задачами и проектами. Web-приложения для управления задачами и проектами помогают пользователям создавать задачи, назначать исполнителей, отслеживать прогресс выполнения и управлять проектами в целом. Они используют базы данных для хранения информации о задачах, сроках выполнения, ответственных лицах и других деталях проектов. Пример: Asana – платформа для управления задачами и проектами, которая использует базу данных для хранения информации о задачах и проектах.

6. Образовательные платформы. Web-приложения образовательных платформ используют базы данных для хранения учебного контента, информации о студентах, курсах, оценках и прогрессе обучения. Они предоставляют студентам доступ к обучающим материалам, заданиям, тестам и другим ресурсам.

7. Здравоохранение. Web-приложения в здравоохранении используют базы данных для хранения медицинских записей пациентов, результатов анализов, назначений врачей и другой информации. Они обеспечивают доступ к медицинским данным врачам, пациентам и административному персоналу. Пример: Электронная медицинская карта (ЭМК) – система хранения медицинских данных пациентов в электронном виде с использованием базы данных.

Каждый из перечисленных типов web-приложений с базами данных имеет свои особенности и применение в различных областях бизнеса и жизни. Это лишь несколько примеров web-приложений с базами данных, и их разнообразие очень велико и зависит от конкретных потребностей и целей бизнеса. Интеграция баз данных с web-приложениями открывает новые возможности для бизнеса, поскольку они становятся легко доступными откуда угодно, не ограничиваясь рабочей станцией или сетью

компании. Это не только повышает эффективность обработки данных, обеспечивает персонализированный пользовательский опыт, но и улучшает общую производительность приложения.

Базы данных и XML

XML (Extensible Markup Language) – это универсальный формат для представления и передачи данных. Он используется для структурирования информации с помощью тегов, что делает его удобным для обмена данными между различными системами. С другой стороны, базы данных представляют собой структурированные наборы данных. Эта обоюдная структурированность позволяет обмениваться данными между различными БД и XML-файлами, а также использовать XML для передачи данных между различными сисемами. Рассмотрим пример экспорта данных из базы данных в формат XML. Имеется база данных с таблицами Product, Order и Supplier (рисунок 20). Необходимо обменяться данными между этой базой данных и XML-файлом.

Table Product

ID	Name	Cost	Description	ID_ Supplier
1	Ноутбук HP Pavilion	50000	15.6" Full HD, Intel Core i5,8GB RAM, 512GB SSD	1
2	Смартфон Samsung S216	70000	6.2"Dynamic AMOLED, 8GB RAM, 128GB ROM, Snapdragon 888	2

Table Supplier

ID	Name	Adress	Contact	Email
1	ООО «Поставщик»	г. Москва, ул. Примерная, д. 12	Иванов Иван Иванович	ivanov@example.com
2	ООО «Электро»	г. Минск, ул. Новая, д. 456	Петров Петр Петрович	petrov@example.com

Table Order

ID	ID_Product	Quantity	TotalPrice	CustomerName	DeliveryAddress
1	1	2	100000	Иванова Анна Петровна	г. Москва, ул. Примерная, д. 4
2	2	2	70000	Петров Петр Петрович	г. Москва, ул. Новая, д. 789

Рисунок 20 – Набор таблиц базы данных для иллюстрации взаимодействия с XML

Для этих целей потребуется *XML-запрос* или *запрос с XML-функциями*, поскольку он использует функции для создания XML-структур из результатов запроса. Иногда такой запрос называют *комплексным*. Комплексный SQL-запрос для извлечения данных из базы данных и преобразования их в формат XML для СУБД MySQL может выглядеть так:

```
XMLElement('xml_data',
    XMLForest(
      ID as "ID",
      Name as "Name",
      Cost as "Cost",
      Description as "Description",
      (SELECT
         XMLElement("Supplier",
           XMLForest(
             s.ID as "ID",
             s.Name as "Name",
             s.Address as "Address",
             s.Contact as "Contact",
             s.Email as "Email"
           )
         )
       FROM Supplier s
       WHERE p.ID_Supplier = s.ID
       ) as "Supplier",
      (SELECT
         XMLElement("Order",
           XMLForest(
             o.ID as "ID",
             o.ID_Product as "ID_Product",
             o.Quantity as "Quantity",
             o.Total_price as "TotalPrice",
             o.CustomerName as "CustomerName",
             o.Delivery_address as "DeliveryAddress"
           )
         )
       FROM Order o
       WHERE p.ID = o.ID_Product
       ) as "Order"
    )
  ) as xml_data
FROM Product p;
```

Этот запрос выберет данные из таблицы Product и для каждой записи вставит информацию о поставщике из таблицы Supplier и информацию о заказе из таблицы Order, затем преобразует результат в формат XML. Разберем этот запрос по шагам.

```
XMLElement('xml data', XMLForest(...))
```

Здесь создается корневой элемент XML с именем 'xml_data', внутри которого будет содержаться вся информация. Функция XMLForest позволяет нам создавать XML-элементы для каждого поля, которое нужно включить в XML.

```
ID as "ID", Name as "Name", Cost as "Cost", Description as
"Description"
```

Этот код обеспечивает включение полей из таблицы Product в формате XML-элементов. Каждое поле представлено как отдельный XML-элемент с указанием имени.

```
(SELECT XMLElement("Supplier", XMLForest(...)) FROM Supplier s
WHERE p.ID_Supplier = s.ID) as "Supplier"
```

Этот подзапрос выбирает информацию о поставщике из таблицы Supplier для каждого продукта и создает XML-элемент Supplier с информацией о поставщике.

```
(SELECT XMLElement("Order", XMLForest(...)) FROM Order o WHERE
p.ID = o.ID_Product) as "Order"
```

Этот подзапрос выбирает информацию о заказе из таблицы Order для каждого продукта и создает XML-элемент Order с информацией о заказе.

```
FROM Supplier s,   FROM Order o, FROM Product p.
```

Эта часть XML-запроса указывает на псевдонимы s, o, p для используемых таблиц Supplier, Order и Product, чтобы обращаться к ним в других частях запроса.

Каждая строка результирующего XML-файла будет представлять собой информацию о товаре, заказе и поставщике:

```
<xml_data>
  <Product>
    <ID>1</ID>
    <Name>Ноутбук HP Pavilion</Name>
    <Cost>50000</Cost>
    <Description>15.6" Full HD, Intel Core i5, 8GB RAM, 512GB
SSD</Description>
    <Supplier>
      <ID>1</ID>
      <Name>ООО «Поставщик»</Name>
      <Address>г. Москва, ул. Примерная, д. 12</Address>
      <Contact>Иванов Иван Иванович</Contact>
      <Email>ivanov@example.com</Email>
    </Supplier>
    <Order>
      <ID>1</ID>
      <ID_Product>1</ID_Product>
      <Quantity>2</Quantity>
      <TotalPrice>100000</TotalPrice>
      <CustomerName>Иванова Анна Петровна</CustomerName>
      <DeliveryAddress>г. Москва, ул. Примерная, д.
4</DeliveryAddress>
    </Order>
  </Product>
  <Product>
    <ID>2</ID>
    <Name>Смартфон Samsung S216</Name>
    <Cost>70000</Cost>
    <Description>6.2"Dynamic AMOLED, 8GB RAM, 128GB ROM,
Snadragon 888</Description>
    <Supplier>
      <ID>2</ID>
      <Name>ООО «Электро»</Name>
      <Address>г. Минск, ул. Новая, д. 456</Address>
      <Contact>Петров Петр Петрович</Contact>
      <Email>petrov@example.com</Email>
    </Supplier>
    <Order>
      <ID>2</ID>
      <ID_Product>2</ID_Product>
      <Quantity>2</Quantity>
      <TotalPrice>70000</TotalPrice>
      <CustomerName>Петров Петр Петрович</CustomerName>
      <DeliveryAddress>г. Москва, ул. Новая, д.
789</DeliveryAddress>
    </Order>
  </Product>
</xml_data>
```

Можно импортировать данные из XML-файла, содержащего информацию о новых товарах и заказах в базу данных:

```
<NewData>
  <Product>
    <Name>Ноутбук Dell XPS</Name>
    <Price>80000</Price>
    <Description>13.4" 4K UHD, Intel Core i7, 16GB RAM, 512GB
SSD</Description>
    <SupplierID>1</SupplierID>
  </Product>
  <Order>
    <ProductID>3</ProductID>
    <Quantity>3</Quantity>
    <TotalPrice>240000</TotalPrice>
    <CustomerName>Иванов Иван Иванович</CustomerName>
    <ShippingAddress>г. Москва, ул. Примерная, д.
12</ShippingAddress>
  </Order>
</NewData>
```

Можно написать скрипт, который прочитает этот XML-файл и добавит новые товары и заказы в базу данных, например, используя Python и библиотеку sqlalchem:

```
from sqlalchemy import create_engine, Table, Column, Integer,
String, MetaData

# Создаем подключение к базе данных
engine = create_engine('postgresql://user:password@localhost:
5432/database')

# Создаем объект метаданных
metadata = MetaData()

# Определяем таблицы
products = Table('Product', metadata,
                 Column('ID', Integer, primary_key=True),
                 Column('Name', String),
                 Column('Cost', Integer),
                 Column('Description', String),
                 Column('ID_Supplier', Integer))

orders = Table('Order', metadata,
               Column('ID', Integer, primary_key=True),
               Column('ID_Product', Integer),
               Column('Quantity', Integer),
               Column('CustomerName', String),
               Column('DeliveryAddress', String))
```

```
# Создаем таблицы в базе данных
metadata.create_all(engine)

# Добавляем новые товары и заказы из XML-файла в базу данных
# с помощью библиотеки xml.etree.ElementTree

import xml.etree.ElementTree as ET

# Считываем данные из XML-файла
tree = ET.parse('новые_данные.xml')
root = tree.getroot()

new_products = []
new_orders = []

# Получаем данные о новых товарах и заказах из XML
for product in root.findall('Product'):
    product_data = {
        "Name": product.find('Name').text,
        "Cost": int(product.find('Price').text),
        "Description": product.find('Description').text,
        "ID_Supplier": int(product.find('SupplierID').text)
    }
    new_products.append(product_data)

for order in root.findall('Order'):
    order_data = {
        "ID_Product": int(order.find('ProductID').text),
        "Quantity": int(order.find('Quantity').text),
        "CustomerName": order.find('CustomerName').text,
        "DeliveryAddress": order.find('ShippingAddress').text
    }
    new_orders.append(order_data)

# Добавляем новые товары и заказы из XML-файла в базу данных
with engine.connect() as conn:
    for product in new_products:
        conn.execute(products.insert(), product)

    for order in new_orders:
        conn.execute(orders.insert(), order)
```

Этот скрипт считывает данные из XML-файла, подключается к базе данных и добавляет новые товары и заказы в соответствующие таблицы. В этом коде MetaData() – это класс из библиотеки sqlalchemy, которая используется для хранения метаданных о таблицах и других объектах базы данных. В предложенном выше коде оператор metadata = MetaData() используется для создания объекта метаданных, который будет хранить информацию о

структуре таблиц базы данных, определенных в коде. После того как определены таблицы с помощью sqlalchemy, можно связать их с объектом метаданных. Позже, при использовании этого объекта метаданных, sqlalchemy может автоматически создавать или обновлять структуру базы данных в соответствии с определенными таблицами.

Можно использовать более простые способы для считывания данных из XML-файла. Один из таких способов – использование сторонних библиотек, которые предоставляют удобные методы для работы с XML-данными. Например, библиотека xmltodict позволяет преобразовывать XML-данные в словари Python, что делает процесс считывания данных из XML более простым и интуитивно понятным.

С помощью XML можно создавать метаданные, описывающие схему базы данных. Это делает XML полезным инструментом для документирования структуры данных и обмена метаданными между различными системами. Приведем пример XML-файла, который описывает структуру данных для базы данных, из предыдущего примера:

```
<DatabaseSchema>
  <Tables>
    <Table name="Product">
      <Columns>
        <Column name="ID" type="Integer" primaryKey="true"/>
        <Column name="Name" type="String"/>
        <Column name="Cost" type="Integer"/>
        <Column name="Description" type="String"/>
        <Column name="ID_Supplier" type="Integer"/>
      </Columns>
    </Table>
    <Table name="Supplier">
      <Columns>
        <Column name="ID" type="Integer" primaryKey="true"/>
        <Column name="Name" type="String"/>
        <Column name="Address" type="String"/>
        <Column name="Contact" type="String"/>
        <Column name="Email" type="String"/>
      </Columns>
    </Table>
    <Table name="Order">
      <Columns>
        <Column name="ID" type="Integer" primaryKey="true"/>
        <Column name="ID_Product" type="Integer"/>
```

```
        <Column name="Quantity" type="Integer"/>
        <Column name="TotalPrice" type="Integer"/>
        <Column name="CustomerName" type="String"/>
        <Column name="DeliveryAddress" type="String"/>
      </Columns>
    </Table>
  </Tables>
</DatabaseSchema>
```

Этот XML-файл содержит описания таблиц и их столбцов для базы данных. Каждая таблица определена внутри <table> элемента, содержащего атрибут name, указывающий на имя таблицы. Внутри <columns> элемента находятся описания столбцов таблицы, каждый из которых определен внутри <column> элемента, содержащего атрибуты name (имя столбца) и type (тип данных столбца). Дополнительно, если столбец является первичным ключом, атрибут primaryKey устанавливается в true.

Кроме того, XML может использоваться в качестве промежуточного формата при обработке данных в базе данных. Например, при выполнении операций импорта или экспорта данных из базы XML-файл можно использовать в качестве промежуточного формата для представления и передачи информации.

Таким образом, XML и базы данных взаимодействуют друг с другом, обеспечивая эффективное хранение, передачу и обработку структурированных данных.

Использование специальных инструментальных средств разработки web-приложений

Одним из основных инструментов для разработки Web-приложений является интегрированная среда разработки (Integrated Development Environmen – IDE). IDE предоставляет разработчикам удобную среду для написания кода, отладки, тестирования и сборки приложений. Существует множество IDE, специализированных на разработку web-приложений, таких как Visual Studio Code, WebStorm, Sublime Text и другие.

Для работы с клиентской частью web-приложений широко используются фреймворки и библиотеки, такие как React, Angular, Vue.js. Эти инструменты позволяют создавать интерак-

тивные пользовательские интерфейсы, упрощают управление состоянием приложения и обеспечивают высокую производительность.

Для работы с серверной частью web-приложений часто используются фреймворки, такие как Express.js для Node.js, Django для Python, Ruby on Rails для Ruby. Эти фреймворки предоставляют разработчикам удобные инструменты для создания API, обработки запросов от клиентов, работы с базами данных и других задач.

Для управления зависимостями и сборкой проекта используются инструменты типа npm (Node Package Manager) или yarn (Yet Another Resource Negotiator). Они позволяют управлять версиями библиотек, устанавливать необходимые зависимости, а также выполнять различные скрипты для сборки и тестирования приложения.

Для тестирования web-приложений существует множество инструментов, таких как Jest, Mocha, Selenium. Они позволяют проводить автоматизированное тестирование приложения, проверять его работоспособность и надежность.

Использование специальных инструментальных средств разработки web-приложений позволяет ускорить процесс разработки, повысить качество и надежность приложения, а также облегчить работу разработчиков. Правильный выбор инструментов и их использование помогут создать успешное и конкурентоспособное web-приложение.

Архитектуры web-приложений с базами данных

Web-приложения с базами данных могут быть реализованы с использованием различных архитектурных подходов. К наиболее распространенным архитектурным моделям относят одно-, двух- и многоуровневые архитектуры.

Одноуровневая архитектура (англ., Single-tier). Представляет собой простую архитектурную модель, где все компоненты web-приложения размещены на одном сервере, а пользователи взаимодействуют с этим приложением через браузер со своего компьютера (рисунок 21).

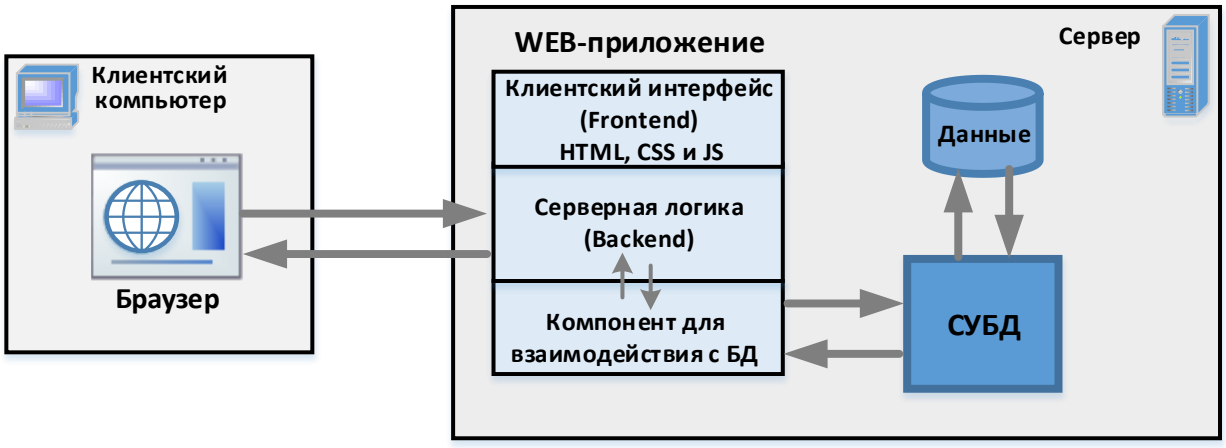

Рисунок 21 – Архитектура одноуровневого web-приложения, развернутого на сервере

Приложение включает клиентский интерфейс (файлы HTML, CSS и JavaScript), элементы которого отправляются в браузер при запросе страницы, серверную логику, обрабатывающую запросы и выполняющую бизнес-операции, и компонент, который позволяет серверной логике взаимодействовать с базой данных для выполнения операции CRUD (обычно называемый слоем доступа к данным). В одноуровневой архитектуре база данных и СУБД находятся на том же уровне, что и код приложения. И все операции чтения и записи данных выполняются непосредственно на одном уровне с компонентами приложения. Говорят, что web-приложение и база данных работают в рамках одного исполняемого процесса или контекста.

Пользователь через браузер посредством через запрашивает доступ к определённой web-странице. Серверное приложение, работающее на серверном компьютере, принимает этот запрос, обрабатывает его и генерирует соответствующий ответ, который может включать HTML, CSS и JavaScript, и отправляет этот ответ обратно в браузер. Последующие пользовательские действия в браузере, например, клики, ввод данных, инициируют запросы к серверу. Серверный код, который обрабатывает запросы от клиента, взаимодействует с базой данных и отправляет ответы обратно в браузер. Стрелки от браузера к серверной части web-приложения и обратно отражают запросы и ответы, которые происходят между клиентом и сервером. Часть программы, реализующая доступ к данным, получает SQL- от серверной логики, передает их на выполнение серверу базы данных и возвращает ре-

зультаты обратно серверной логике. Стрелки между серверной логикой и этой частью отражают этот внутренний процесс обмена данным.

Клиентская часть программы не имеет прямых стрелок взаимодействия, так как ее роль – быть отображаемой и исполняемой в браузере, но сама по себе она не отправляет или принимает данные напрямую от серверной логики или базы данных. Клиент в этом контексте, как правило, означает браузер пользователя, который получает от сервера готовую к отображению информацию. В такой монолитной архитектуре браузер не обрабатывает бизнес-логику или данные, вместо этого он просто получает готовую информацию для отображения от сервера. Другими словами, в данном случае клиентская часть не выполняет тяжелой работы по обработке данных, всё это происходит на сервере. В отличие от двухуровневой архитектуры, где браузер активно участвует в процессе (например, выполняет код JavaScript локально), в монолитном варианте браузер преимущественно является пассивным компонентом, отображающим информацию.

Одноуровневое web-приложение также может быть развернуто непосредственно на компьютере пользователя. В этом случае, все компоненты приложения, включая СУБД, находятся на одной машине(рисунок 22).

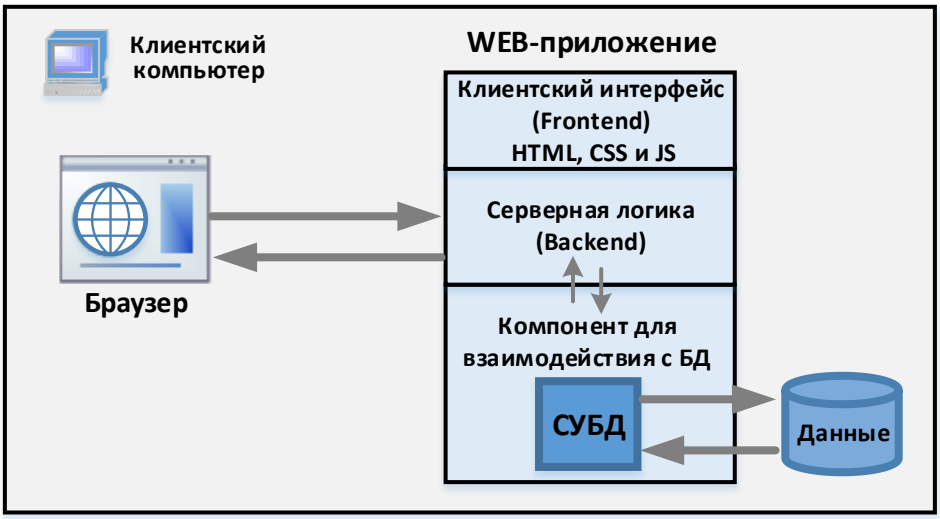

Рисунок 22 – Архитектура одноуровневого web-приложения со встроенной СУБД на компьютере пользователя

СУБД при этом может быть запущена как отдельный процесс, обеспечивая тем самым более гибкую и масштабируемую архитектуру, или может быть встроена непосредственно в приложение, что упрощает развертывание и управление, как в случае с легковесной встраиваемой СУБД SQLite

Таким образом, архитектура одноуровневого приложения предлагает гибкость в выборе развертывания и управления базами данных, адаптируясь к потребностям и условиям использования. Приложения, созданные на базе этой архитектуры, часто используются для небольших проектов, прототипов или индивидуальных программ, когда у разработчика нет необходимости разделять функциональность на отдельные компоненты или внедрять сложные архитектурные решения. Одноуровневая архитектура проста в реализации и обеспечивает быстрый доступ к данным, но она имеет свои ограничения. Например, такая модель может быть сложной для масштабирования и поддержки приложений с большим объемом данных или с развитой бизнес-логикой. В связи с этим часто используются более сложные многоуровневые архитектуры для более эффективной организации приложений.

Двухуровневая архитектура (англ., Two-tier). В этой модели web-приложение явно разделено на два основных компонента: клиентскую и серверную части. Клиентская часть отвечает за отображение информации пользователю и обеспечивает интерфейс для взаимодействия с ним. Клиентская часть фактически хранится на сервере в виде HTML, CSS, JavaScript файлов, но их исполнение производится непосредственно в браузере пользователя. Когда пользователь заходит на сайт, его браузер запрашивает у сервера страницу, сервер управляет генерацией содержимого этих файлов и отправляет их обратно в браузер. Браузер интерпретирует полученные HTML, CSS и JavaScript, предоставляя пользователю актуальную страницу. Когда пользователь выполняет действия на web-странице, например, нажимает на кнопки, заполняет формы, клиентская часть генерирует запросы к серверной части, чтобы получить нужные данные или выполнить определённые операции на сервере, например, сохранить изменения, сделанные пользователем. Стрелка между браузером и клиент-

ской частью обозначает интерпретацию браузером кода клиентской части и отображение страниц в соответствии с заданным стилем и динамикой. Обратная стрелка отражает пользовательский ввод (клики, ввод текста, выбор элементов), активизацию JavaScript-кода, который изменяет содержимое или внешний вид страницы, а также отправляет запросы на сервер.

Серверная часть обрабатывает эти запросы и отправляет ответы обратно в браузер, где клиентская часть использует полученные данные для обновления интерфейса или отображения новой информации пользователю. Серверная часть web-приложения в двухуровневой архитектуре содержит бизнес-логику, которая обрабатывает запросы от клиента, выполняет необходимые операции с данными и взаимодействует с базой данных, используя СУБД. База данных и СУБД располагаются на компьютере-сервере. Клиентская часть может запросить или изменить данные, но все операции с данными выполняются на сервере через СУБД (рисунок 23).

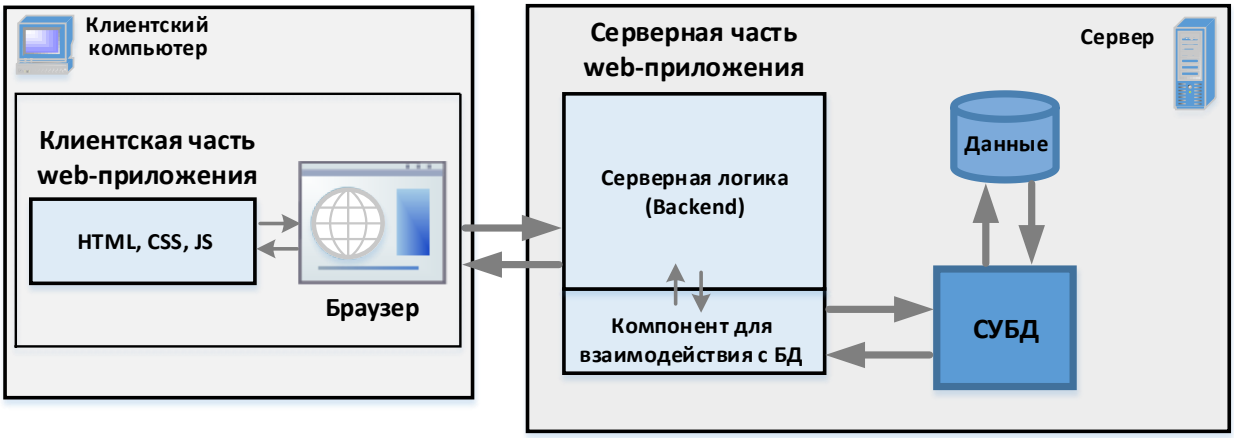

Рисунок 23 – Архитектура двухуровневого web-приложения

Двухуровневая архитектура является одной из классических моделей для web-приложений. Эта архитектура позволяет более четко разделять ответственность между клиентом и сервером, что облегчает поддержку и масштабирование.

И в одноуровневой, и в двухуровневой архитектурах взаимодействие пользователя с базой данных всегда происходит через серверную часть, которая обрабатывает логику приложения и управляет обращениями к базе данных. Это обеспечивает изоляцию и защиту данных, позволяя реализовать бизнес-логику на

сервере и предотвращать неавторизованный доступ к базе данных.

Многоуровневая архитектура (Multi-tier). Архитектуры, которые включают в себя более двух уровней, обычно классифицируются как многоуровневые или n-уровневые. Самым распространенным примером многоуровневой архитектуры является трехуровневая архитектура, но существуют и более сложные варианты, в которых могут быть выделены дополнительные уровни.

В *трехуровневом web-приложении* архитектура обычно состоит из трех основных компонентов:

1. клиентского уровня (англ., presentation layer – презентационный слой);
2. серверного уровня (англ., business logic layer – слой бизнес-логики, также известный как слой обработки данных);
3. уровня данных (data layer – слой хранения и доступа к данных в БД).

Все эти уровни могут быть физически разделены и могут находиться на разных серверах, что обычно обеспечивает более высокую безопасность и повышает производительность, позволяет независимо масштабировать каждый из уровней (рисунок 24).

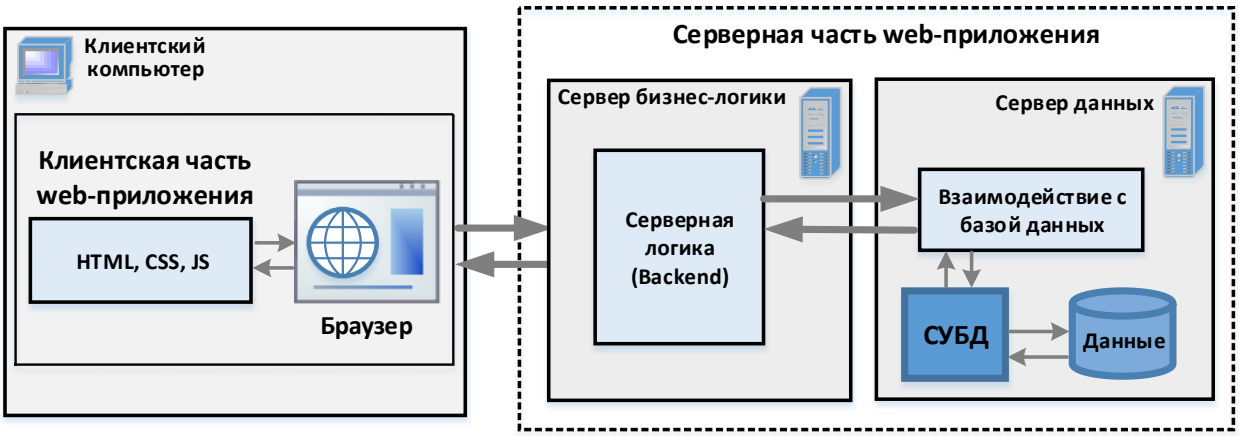

Рисунок 24 – Архитектура трехуровневого web-приложения

Тем не менее, они также могут быть размещены на одном и том же сервере, просто, будучи разделенными на разные части приложения. Это, как правило, происходит в меньших системах или в средах разработки. Вариант развертывания зависит от мно-

жества факторов, включая требования к производительности, надежности, масштабируемости, а также бюджета и доступных технических ресурсов. Оптимальная архитектура должна учитывать все эти аспекты для достижения баланса между стоимостью, производительностью и удобством поддержки.

Клиентский уровень в трехуровневом web-приложении попрежнему взаимодействует с сервером, но серверная часть web-приложения теперь разделена на уровень серверной логики и уровень данных. Теперь уровень данных объединяет в себе код взаимодействия с базой данных, саму БД и СУБД. В двухуровневой архитектуре код взаимодействия web-приложения с БД, включающий SQL-запросы, обычно смешан с бизнес-логикой приложения. Изменения в структуре базы данных или переход на другую СУБД могут потребовать значительных изменений в бизнес-логике приложения, поскольку они тесно переплетены. А теперь отдельная реализация уровня данных и выделение его в отдельный слой позволило отделить бизнес-логику от логики доступа к данным. Изменения в базе данных или смена СУБД, которые происходят в результате развития web-проекта, меньше влияют на бизнес-логику. Все ключевые функции по работе с базой данных, реализованы в коде своего собственного уровня.

Во-первых, этот уровень формирует и отправляет запросы к базе данных (например, SQL-запросы) для извлечения, обновления, вставки или удаления данных. Он также обрабатывает ответы от базы данных, преобразуя полученные данные в формат, удобный для использования серверной частью приложения.

Во-вторых, уровень данных скрывает детали обращения к СУБД, предоставляя абстрактный интерфейс для выполнения операций с данными. Это позволяет серверной части приложения общаться с базой данных через унифицированный API, не зависящий от конкретной СУБД. Уровень данных предоставляет серверной части приложения простые и понятные методы для работы с данными, тем самым упрощая разработку и поддержку приложения.

В-третьих, уровень данных берет на себя функции контроля выполнения транзакций, обеспечивая целостность и надежность данных. Он гарантирует, что все операции в рамках одной тран-

закции либо полностью выполняются, либо полностью отменяются, предотвращая частичное изменение данных.

В-четвертых, уровень данных обеспечивает безопасность хранения и обработки данных, реализуя механизмы аутентификации, авторизации и шифрования. Это предотвращает неавторизованный доступ к данным и защищает конфиденциальную информацию.

В пятых, для повышения производительности уровень данных может включать механизмы кэширования, позволяющие временно хранить часто запрашиваемые данные в памяти, сокращая количество обращений к физической базе данных.

Четырехуровневая архитектура web-приложения вводит еще один уровень разделения, что позволяет дополнительно улучшить модульность и масштабируемость приложения. В такой архитектуре выделяется уровень web-сервера и уровень бизнес-логики. Уровень web-сервера отвечает за обработку клиентских запросов, полученных через интернет, и передачу их на уровень бизнес-логики. Кроме того, web-сервер может заниматься кэшированием ответов, уменьшая нагрузку на уровень бизнес-логики, и предоставлять дополнительные слои безопасности. Он также может также обрабатывать статические ресурсы, такие как изображения, CSS и JavaScript файлы. А уровень бизнес-логики реализует логику приложения, которая обрабатывает запросы от web-сервера, выполняет необходимые операции и взаимодействует с уровнем доступа к данным.

В более ранних, более простых архитектурах web-приложений, таких как двухуровневая или трехуровневая, функции уровня web-сервера и бизнес-логики могли быть комбинированы в одном слое. Это означает, что те же серверы, которые обрабатывали входящие запросы от клиентов, также обрабатывали бизнес-логику web-приложения. С появлением более сложных и динамичных web-приложений, таких как web-сервисы и полномасштабные интерфейсы web-программирования RESTful API, стало важным разделить эти функции на отдельные слои. Это позволяет каждому слою специализироваться на своей специфической функции, улучшая эффективность, масштабируемость и надежность web-приложений. Выделение уровня web-сервера

позволяет этому слою специализироваться на обработке запросов HTTP и общении с клиентами, оставляя слой бизнес-логики свободным для обработки более сложных аспектов web-приложения, таких как обработка данных, выполнение бизнес-логики и т.д.

Многоуровневые архитектуры особенно полезны в больших и сложных системах, где нужно обеспечить высокую производительность, масштабируемость и удобство поддержки.

Микросервисная архитектура (Microservices). В этой архитектуре приложение состоит из набора независимых компонентов – *сервисов*, каждый из которых реализует свою часть бизнес-логики в виде ограниченного набора функций. Хотя микросервисная архитектура может использоваться в различных типах приложений, она имеет особые преимущества для web-приложений. Например, довольно часто web-приложения сталкиваются с ситуацией, когда определенная часть приложения испытывает больше нагрузки. С микросервисами можно масштабировать только те сервисы, которые испытывают увеличение нагрузки. Использование микросервисов позволяет достичь мгновенной реакции на действия пользователя и активно обновлять пользовательский интерфейс в ответ на изменения на сервере, что особенно важно в современных web-приложениях, где пользователи ожидают быстрой и плавной интерактивности.

При организации микросервисной архитектуры преимущественным решением является реализация концепции *Database per Service* или *один сервис – одна база данных*. Эта архитектура подразумевает, что каждый микросервис имеет свою собственную базу данных, которую другие микросервисы не могут напрямую использовать или иметь к ней доступ. Здесь ключевым является принцип разделения ответственности, согласно которому каждый микросервис отвечает только за свою базу данных. Эти базы данных могут быть разными по своей природе и содержимому в зависимости от потребностей конкретного сервиса. Например, сервисы, основанные на работы с сообщениями из очереди; микросервисы, выполняющие вычисления в реальном времени; микросервисы, реализующие функции обработки и трансформации, например, сервисы шифрования или сервисы логирования. В случае, если автономная база данных одного микро-

сервиса скомпрометирована, другие микросервисы и их данные будут защищены от этого негативного события. СУБД в таком случае могут быть расположены в облаке или на сервере, на котором размещаются микросервисы, и в этом контексте СУБД выступает в качестве отдельной сущности, которая обслуживает конкретный микросервис.

Хотя многие микросервисы имеют собственные базы данных для хранения и обработки данных, это не является обязательным. Некоторые микросервисы могут полагаться на взаимодействие с другими сервисами для получения данных, вместо использования собственной базы данных. В этих случаях, они обычно общаются через API или систему обмена сообщениями. Выбор использования базы данных или получение данных от других микросервисов или внешних систем зависит от специфики тех задач, которые должен решать микросервис.

С другой стороны, в некоторых случаях микросервисы могут использовать единую, общую базу данных, нарушая принцип один сервис – одна база данных (рисунок 25).

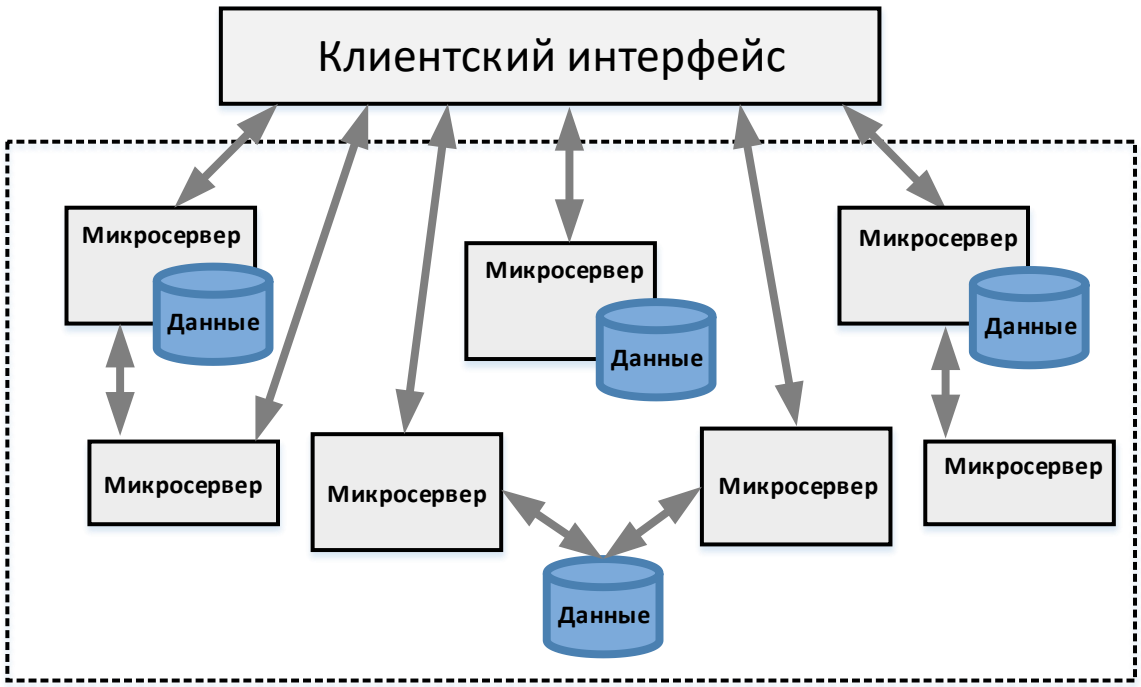

Рисунок 25 – Обобщенная архитектура микросервисного приложения

Такой подход может привести к тому, что микросервисы становятся тесно связанными из-за их общей зависимости от одной базы данных, а это может вызвать ряд потенциальных проблем. Например, использование общей базы данных может вызвать проблемы при частой смене схемы данных. Совместное использование единой базы потребует изменений в коде всех этих микросервисов. В противном случае некоторые микросервисы могут столкнуться с неожиданностями в форме разрывов в данных, конфликтов или ошибок обработки, связанных с неожиданными изменениями структуры данных. Доработки и обновления становятся сложнее, так как любое изменение должно учитываться в коде всех микросервисов. Если база данных, которая используется несколькими микросервисами, была подвергнута атаке и при этом её данные были взломаны, украдены, испорчены или любым другим способом использованы злоумышленниками, все эти микросервисы, могут оказаться под угрозой.

В микросерверной архитектуре могут использовать как SQL, так и NoSQL базы данных, в зависимости от конкретных требований и сценариев использования. Некоторые системы могут использовать смешанный подход, где одни микросервисы используют SQL базы данных, а другие – NoSQL. С одной стороны, использование однотипных БД может упростить процесс разработки, поскольку разработчики могут использовать одни и те же знания и инструменты для работы с каждым микросервисом. Это также может упростить процесс мониторинга и обслуживания баз данных, поскольку все они работают на одной и той же технологии. С другой стороны, использование смешанных БД позволяет каждому микросервису выбирать ту технологию системы управления базами данных, которая лучше всего подходит для его собственных потребностей. Это может обеспечить большую гибкость и позволить каждому микросервису оптимально использовать свои данные.

Каждая из рассмотренных архитектурных моделей характеризуется специфическими аспектами взаимодействия с базами данных и поддержки их работы (Таблица 2).

Таблица 2

Взаимодействие архитектур приложений с БД: сравнение

Роль БД	Взаимодействие с БД	Безопасность БД
Одноуровневая архитектура		
База данных служит для хранения и обработки данных непосредственно в коде приложения.	Доступ к данным осуществляется прямо из кода приложения через API базы данных.	Безопасность основывается на принципах безопасности самой базы данных. Ответственность за защиту данных лежит прямо на коде приложения.
Двухуровневая архитектура		
Работа с базой данных осуществляется на серверном уровне. База данных также может выполнять вычисления и транзакции.	Сервер контролирует все взаимодействия с базами данных, что обеспечивает более высокий уровень безопасности и гибкости.	Защита данных обеспечивается на уровне сервера, используются различные технологии и стандарты безопасности
Многоуровневая архитектура		
База данных может быть размещена на отдельном слое, что обеспечивает большую гибкость и масштабируемость.	Взаимодействие с базой данных контролируется различными уровнями архитектуры, что позволяет более тонко управлять процессами обработки данных.	Безопасность обеспечивается на каждом отдельном слое, что помогает снизить риски связанные с защитой данных.
Микросервисная архитектура		
Каждый микросервис может иметь свою базу данных, которая соответствует его специфическим требованиям. Это обеспечивает наивысшую гибкость в управлении данными.	В каждом сервисе обработка и передача данных между сервисами реализуется индивидуально. Обычно микросервисы взаимодействуют между собой через API.	Безопасность обрабатывается на уровне каждого микросервиса, что позволяет обеспечивать высокий уровень защиты данных по мере необходимости.

Выбор архитектуры влияет на манеру управлениями данными, скорость обработки запросов, а также на стратегии масштабирования и безопасности, которые разработчики выберут для своего web-приложения.

Способы интеграции баз данных в среду web

Интеграция базы данных в приложение означает установление связи между приложением и базой данных с целью сохранения, извлечения, обновления и удаления данных. Интеграция базы данных в web-приложение означает то же, что и интеграция базы данных в любое другое приложение: установление связи между web-приложением и базой данных для сохранения, извлечения, изменения и удаления данных. Однако, в контексте web-приложений, эта интеграция обычно предполагает наличие определенных дополнительных шагов и аспектов.

Рассмотрим сначала общие принципы интеграции баз данных в приложения. Как уже должно быть понятно, между приложением и базой данных располагается СУБД, которая принимает запросы, которые изначально были вписаны в программу, либо были сформированы на основе интерактивного взаимодействия пользователя с программой. Однако, для того, чтобы приложение могло обратиться к СУБД и передать ей запрос от пользовательской программы, необходимо еще одно связующее звено. Таким связующим звеном является библиотека или драйвер, которые определяют способ передачи запросов в формат, понятный конкретной СУБД. И они же определяют способ получения ответов от этой СУБД в формат, который понимает приложение. Другими словами, библиотека или драйвер определяют способ коммуникации приложения и СУБД.

Несмотря на то, что команды SQL стандартизированы, и понятны большинству реляционных СУБД, каждая из них может использовать свой уникальный протокол для коммуникации с клиентами. Например, MySQL использует свой собственный сетевой протокол для коммуникации и этот протокол отличается от протоколов, которые используются PostgreSQL, Oracle, SQL Server и т.д. Драйвер или библиотека, которые используются

приложением, знают, как общаться «со своей» СУБД с помощью этого протокола. Драйверы и библиотеки специфичны не только для конкретной СУБД, но и для языка программирования. Например, для СУБД MySQL в программах на языке Python может использоваться официальный драйвер mysql-connector-python или же библиотека pymysql. Для СУБД PostgreSQL на языке Python можно использовать библиотеку psycopg2. Для работы с MS SQL Server в C# или .NET языках, можно использовать встроенный класс SqlConnection, который является частью библиотеки System.Data.SqlClient. Библиотеки или драйверы обычно сопровождаются документацией, которая показывает, как используются их функции и методы для взаимодействия с СУБД.

В более сложных случаях, когда происходит взаимодействие приложения с базой данных по сети, может потребоваться ещё больше специфических действий, которые обеспечиваются драйверами и библиотеками: аутентификация, шифрование, сжатие данных, повторная отправка запросов в случае ошибок сети и т.д.

Практически все библиотеки Python для взаимодействия с базами данных используют понятие *курсора* или аналогичныхо механизмов. Объект курсора позволяет выполнять SQL-запросы к базе данных и извлекать результаты этих запросов. Абстракция в виде курсора унифицирует взаимодействие с базами данных и делает код более читаемым и понятным.

Вот примеры работы с курсором в некоторых популярных библиотеках Python для работы с базами данных:

```
#SQLite (встроенная библиотека sqlite3):
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

#PostgreSQL (библиотека psycopg2):
import psycopg2
conn = psycopg2.connect(database="mydb", user="myuser", pass-
word="mypassword")
cursor = conn.cursor()

#Oracle (библиотека cx_Oracle):
import cx_Oracle
conn = cx_Oracle.connect(user='myuser', password='mypassword',
dsn='localhost:1521/orcl')
cursor = conn.cursor()
```

Во всех этих примерах после установления соединения создается объект курсора, который затем используется для выполнения SQL-запросов.

Но стоит отметить, что существуют и более высокоуровневые инструменты и библиотеки, в которых использование курсоров от пользователя скрыто, благодаря чему работа с базой данных становится еще более простой и интуитивно понятной. К таким средствам относятся ORM-фреймворки, например, SQLAlchemy для Python или Hibernate для Java. Они предоставляют более удобный и объектно-ориентированный интерфейс для взаимодействия приложений с базами данных.

При установке соединения приложения с базой данных обычно требуется информация для подключения, такая как адрес хоста, порт, имя базы данных, имя пользователя и пароль. Здесь под адресом хоста понимается IP-адрес или доменное имя сервера, на котором расположена система управления базами данных. Это место может быть на том же компьютере, где запущено приложение. В этом случае обычно используется адрес "127.0.0.1" или соответствующее имя – "localhost". Или это может быть удаленный сервер в Интернете или внутренней сети компании. Например, доменное имя – "database.example.com", IP-адрес – "192.0.2.1".

Порт в контексте сетевого соединения можно рассматривать как точку подключения или доступа к определенному процессу (или службе), функционирующему на сервере. Служба базы данных – это и есть запущенная и постоянно функционирующая СУБД. Служба базы данных обрабатывает запросы, отправляемые к базе данных, и управляет всеми операциями, связанными с хранением, обработкой и извлечением данных. Каждое сетевое соединение на сервере имеет уникальный номер порта, который используется для идентификации определенного процесса, работающего на этом сервере. В случае баз данных, номер порта обычно определяется по умолчанию в зависимости от используемой СУБД. Например, для MySQL это обычно порт 3306, в то время как PostgreSQL использует порт 5432. Определение этих портов контролируется международной организацией IANA (Internet Assigned Numbers Authority).

Имя пользователя и пароль служат для аутентификации пользователя, запустившего приложение, и подтверждают, что пользователь имеет право доступа к базе данных. После успешной аутентификации СУБД также проверяет, какие именно привилегии или права доступа были назначены этому пользователю и контролирует их во время работы. Различные уровни доступа могут включать в себя права на чтение, запись, обновление или удаление данных, а также на создание и удаление таблиц, индексов, процедур и функций. Конкретный набор привилегий зависит от настроек и политики безопасности, установленных администратором базы данных. Поддержка безопасности и аутентификации при подключении к базе данных является одним из важных аспектов разработки web-приложений, поскольку обеспечивает защиту данных и предотвращает несанкционированный доступ к информации.

Помимо простого установления связи между приложением и СУБД, интеграция баз данных в web-приложение подразумевает учет дополнительных аспектов, связанных с web-технологиями и архитектурными принципами web-разработки.

Web-приложения часто используют API для извлечения данных из базы данных и передачи их на web-страницы для отображения или обработки пользователем. Многие web-приложения интегрируются с другими службами, такими как системы оплаты, сервисы электронной почты, социальные медиа и т.д. API, в этом контексте, действует как мост между базой данных и web-приложением, обеспечивая передачу данных от базы данных к внешним клиентам. От эффективности этих API зависит производительность и отзывчивость web-приложения, поэтому разработчики должны уделить особое внимание их проектированию и реализации.

Интеграция требует не только установления соединения с базой данных, но и оптимизации запросов и использования современных технологий, таких как асинхронные запросы, для обеспечения высокой производительности и отзывчивости приложения. *Асинхронные запросы* – это метод, который позволяет web-приложению продолжать выполнять другие задачи, пока оно ждет ответа от базы данных. Это повышает отзывчивость приложения, так как оно не «зависает» в ожидании ответа, и улучшает

его производительность, поскольку время обработки запроса может быть использовано для выполнения других действий. Это особенно важно в современных широкомасштабных web-приложениях, которые обрабатывают большое количество пользовательских запросов одновременно, поскольку они часто могут оказаться в состоянии, когда ожидание ответа от базы данных может значительно замедлить общую производительность.

Еще один аспект, обеспечивающий снижение нагрузки на базу данных и улучшения производительности web-приложений связан с кэшированием. *Кэширование* позволяет хранить результаты часто используемых запросов в оперативной памяти или во внешних кэш-хранилилщах, чтобы при повторном использовании ускорить доступ к этим данным. Web-приложения могут использовать кэширование на уровне сервера для хранения результатов выполнения запросов и обработки данных. Одновременно с этим web-приложения могут использовать кэширование на уровне клиента для хранения данных, которые могут быть сохранены в браузере пользователя. Это позволяет уменьшить количество запросов к серверу и ускорить загрузку страниц для пользователей.

Помимо кэширования результатов, web-приложения могут поддерживать кэширование данных, использующихся для создания отдельных частей web-страниц (например, HTML, изображения, CSS, JavaScript). Эти данные или пути к ним могут храниться в базе данных. В этом случае кэширование обеспечивает повторное использование и улучшение производительности загрузки страниц и может быть реализовано как на стороне сервера, так и на стороне клиента.

При интеграции баз данных в web-приложения необходимо обеспечить дополнительную безопасность данных, чтобы защитить информацию пользователей от несанкционированного доступа. К дополнительным мерам относятся: шифрование данных, защита от SQL-инъекций, аутентификация и авторизация пользователей и другие меры безопасности.

Многие web-приложения используют *сессии* для отслеживания информации о пользователях между запросами. Данные каждой сессии могут храниться как в базе данных web-приложения, так и в других хранилищах, таких как кэш или файловая система. Если важно, чтобы данные о сессии сохранялись даже после пе-

резапуска сервера, то хранение в базе данных может быть лучшим решением. Это значит, что информация о сессии будет сохраняться на сервере web-сайта, и даже если произойдет что-то неожиданное, вроде перезапуска сервера, web-сайт все равно сможет «вспомнить» данные о сессии, и пользователь сможет продолжить работать с web-сайтом, не замечая никаких изменений.

Таким образом, хотя концептуально интеграция базы данных в web-приложение это то же самое, что и интеграция в любое приложение, в реальности это процесс может включать в себя больше этапов и учитывать большее количество факторов.

Обеспечением вышеперечисленных аспектов занимаются разработчики web-приложений, обладающие опытом работы с базами данных и современными web-технологиями. Эти специалисты используют специализированные инструменты и фреймворки, ориентированные на особенности работы в интернете, такие как ORM, библиотеки для работы с базами данных, асинхронные библиотеки и инструменты оптимизации запросов. Одновременно для обеспечения высокой производительности и масштабируемости баз данных требуется оптимальная настройка, за которой следят разработчики баз данных и администраторы.

Интеграция базы данных в web-приложение – сложный процесс, который требует совместных усилий команды специалистов. В случае простых приложений все эти функции может выполнять один человек, в случае крупных web-приложений, обычно разработка, оптимизация и обслуживание баз данных - это коллективное усилие команды специалистов.

Технологии интеграции баз данных в среду Web: CGI, ISAPI, ASP, PHP и др.

Для интеграции баз данных в web-среду широко используются различные технологии, механизмы и стандарты, которые обеспечивают эффективное выполнение запросов к базам данных, обработку полученных данных и их динамическую передачу пользователям в интерактивном формате.

С течением времени эти технологии постепенно развивались, обогащаясь новыми возможностями и улучшениями, что существенно улучшило их эффективность и адаптируемость к изменяющимся требованиям современных web-приложений. Каждая новая технология обладает своими преимуществами, недостатками и областями применения. Важно понимать, как эти технологии работают, чтобы выбрать наиболее подходящую для конкретных нужд проекта.

Одним из первых механизмов, позволяющих серверным скриптам обрабатывать пользовательские данные, поступающие через web-формы, и взаимодействовать с базами данных, была технология *CGI* (Common Gateway Interface). CGI – это стандарт, который позволяет web-серверу передавать запросы и данные, полученные от браузера, к специализированным программам, запущенным на сервере. Эти программы являются компонентом серверной части web-приложения. Они обрабатывают полученные данные и взаимодействуют с базой данных для выполнения операций, таких как добавление, обновление или извлечение информации. Затем отправляют результаты обратно в web-сервер, который, в свою очередь, возвращает их в браузер пользователя в форме web-страницы или другого формата ответа. Таким образом, CGI действует как мост между браузером и серверными программами, облегчая динамическую обработку данных и взаимодействие с базами данных.

До широкого распространения Python в web-разработке CGI использовался в основном с такими языками программирования как Perl и C. Perl был особенно популярным выбором для CGI-скриптов во многих web-приложениях.

Ниже приведен пример подключения Perl CGI-скрипта к базе данных MySQL, выполняющего запрос на выборку данных из таблицы my_table и вывод результатов на web-страницу в виде списка. Конечно, в реальном приложении потребуются дополнительные проверки, обработка ошибок и поддержка безопасности. Когда web-сервер получает запрос на запуск данного скрипта, он запускает интерпретатор Perl для его выполнения, а затем отправляет результат выполнения скрипта обратно в браузер.

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;

# Параметры подключения к базе данных
my $dsn = "DBI:mysql:database=testdb;host=localhost";
my $username = "username";
my $password = "password";

# Подключение к базе данных
my $dbh = DBI->connect($dsn, $username, $password) or die
"Ошибка подключения к базе данных: $DBI::errstr";

# Выполнение запроса к базе данных
my $sth = $dbh->prepare("SELECT * FROM my_table");
$sth->execute() or die "Ошибка выполнения запроса:
$DBI::errstr";

# Вывод результатов запроса на web-страницу
print "Content-type: text/html\n\n";
print "<html><head><title>Пример CGI с
Perl</title></head><body>";
print "<h1>Список данных из базы данных:</h1>";
print "<ul>";

while (my @row = $sth->fetchrow_array()) {
    print "<li>$row[0] - $row[1]</li>";
}

print "</ul>";
print "</body></html>";

# Закрытие соединения с базой данных
$dbh->disconnect();
```

Хотя CGI является основополагающим методом динамической web-разработки и серверной обработки запросов, со временем он уступил место более современным и эффективным технологиям из-за своей относительной неэффективности. Каждый запрос через CGI требует создания нового процесса, что может быть ресурсоемким и медленным. CGI не является оптимальным выбором для высоконагруженных приложений из-за своей структуры и способа обработки запросов. Наконец, CGI не предлагает встроенной поддержки управления пользовательскими web-сессиями и имеет ограниченные возможности для взаимодей-

ствия с базами данных. Все это потребовало появления новых решений в ответ на растущие потребности бизнеса.

Одним из таких решений стал переход к *PHP* как к языку программирования, встраиваемому в HTML, и одновременно к технологии обработки данных на стороне сервера, которая обеспечивает лучшую производительность и масштабируемость web-приложений. PHP имеет обширные встроенные возможности для взаимодействия с различными системами управления базами данных, такими как MySQL, PostgreSQL, Oracle, и многими другими. Эти возможности включают ряд встроенных функций, которые позволяют PHP-скриптам выполнять различные операции с базами данных, такие как выполнение запросов, добавление, обновление и удаление данных, управление транзакциями и т. д. Например:

```
#!/usr/bin/php-cgi
<?php

// Установка заголовка Content-type
header('Content-type: text/html');

// Подключение к базе данных
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password,
$dbname);

// Проверка соединения
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Выполнение запроса к базе данных
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

// Вывод данных на web-страницу
echo "<html><head><title>Пример CGI с
БД</title></head><body>";
echo "<h1>Список пользователей:</h1>";
echo "<ul>";
```

```
if ($result->num_rows > 0) {
    // Вывод данных каждой строки
    while($row = $result->fetch_assoc()) {
        echo "<li>" . $row["id"]. " - " . $row["firstname"]. "
" . $row["lastname"]. "</li>";
    }
} else {
    echo "0 результатов";
}

echo "</ul>";
echo "</body></html>";

// Закрытие соединения с базой данных
$conn->close();
?>
```

Простой и интуитивно понятный интерфейс для работы с базами данных делает процесс разработки web-приложений на базе более удобным и эффективным. Разработчики могут легко написать PHP-скрипты для выполнения операций с базами данных без необходимости изучать сложные API или языки запросов к базам данных.

Функции PHP для работы с базами данных обычно оптимизированы для обеспечения быстрой и эффективной работы. Они могут использовать различные методы и стратегии для минимизации времени выполнения запросов и оптимизации использования ресурсов сервера. Функции PHP могут автоматически оптимизировать SQL-запросы, например, путем добавления индексов к таблицам, улучшения структуры запросов или использования кэширования результатов запросов. Некоторые функции базы данных в PHP позволяют использовать предварительно скомпилированные запросы, которые могут быть повторно использованы для выполнения аналогичных запросов без необходимости полного анализа и компиляции каждый раз. Вместо выполнения множества отдельных запросов к базе данных, PHP может собирать запросы в пакеты и отправлять их на выполнение одним запросом, что может снизить нагрузку на сервер базы данных и улучшить производительность. PHP может эффективно управлять соединениями с базой данных, включая их создание, управление пулом соединений и закрытие, что помогает сократить время ожидания и ресурсы, затраченные на установку соедине-

ния. PHP может использовать кэширование для временного хранения результатов запросов или данных, что позволяет избежать повторного выполнения запросов к базе данных при обращении к тем же данным в будущем. Благодаря такому широкому спектру возможностей PHP используется во множестве web-приложений, от небольших сайтов до крупных web-платформ, таких как WordPress и Facebook (хотя Facebook теперь использует Hack, язык, произошедший от PHP). Благодаря своей популярности, у PHP есть большое и активное сообщество разработчиков, что обеспечивает большое количество ресурсов для обучения. В целом, PHP – это гибкий язык, который позволяет разработчикам выбирать между разработкой «с нуля» или «без фреймворка», используя стандартные функции и библиотеки, доступные в языке. Этот подход дает полный контроль над кодом и архитектурой приложения, но также требует больше усилий и знаний. Разработчику web-приложения нужно будет самостоятельно реализовывать многие функции, такие как маршрутизация, обработка форм, работа с базой данных и т.д. PHP без фреймворка может быть хорошим выбором для небольших проектов или для разработчиков, которые хотят глубоко понимать внутреннюю работу своих приложений. Однако для PHP разработано много популярных PHP фреймворков, таких как Laravel, Symfony, CodeIgniter, Zend Framework и Yii. Каждый фреймворк имеет свои особенности, сильные и слабые стороны, но его основная цель - упростить разработку web-приложений.

ASP (Active Server Pages) – это технология, разработанная корпорацией Microsoft, ставшая одним из первых популярных методов для создания динамических web-страниц и взаимодействия с базами данных. Технология ASP предоставляла серверный скриптовый язык, позволяющий создавать интерактивные web-приложения. На начальных этапах применения ASP основным языком для написания серверных скриптов был язык VBScript (Visual Basic Scripting Edition). Однако с развитием технологий и web-стандартов, а также с появлением новых версий ASP и его альтернатив, таких как ASP.NET, более распространенным стало использование других языков, таких как JScript(JavaScript от MicroSoft), PerlScript, PythonScript, RexxScript, TCLScript. Это версии известных языков программи-

рования, которые могу быть использованы в ASP-страницах для написания серверных скриптов. Наиболее популярным из перечисленных является JScript.

ASP позволяет устанавливать соединение с различными типами СУБД, такими как MySQL, PostgreSQL, Microsoft SQL Server и другими. С помощью ASP можно выполнять любые SQL-запросы для получения, изменения и удаления данных в базе данных. Он предоставляет средства для обработки результатов SQL-запросов, позволяя фильтровать, сортировать, агрегировать и выполнять другие операции с данными. Позволяет создавать динамические веб-страницы, которые отображают данные из базы данных в реальном времени. С помощью ASP можно автоматизировать различные задачи, связанные с базами данных, например, создание отчетов, отправку уведомлений.

Пример подключения к MySQL-базе данных на языке JScript:

```
<%   # начало блока кода ASP
import win32com.client

# Подключение к базе данных MySQL
conn = win32com.client.Dispatch('ADODB.Connection')
connString = "Driver={MySQL ODBC 8.0 Unicode Driver};" & _
             "Server=your_server_address;" & _
             "Database=your_database_name;" & _
             "Uid=your_username;" & _
             "Pwd=your_password;"
conn.Open(connString)

# Выполнение SQL запроса
cmd = win32com.client.Dispatch('ADODB.Command')
cmd.ActiveConnection = conn
cmd.CommandText = "SELECT * FROM users"

# Получение результатов запроса
rs = cmd.Execute()

# Вывод данных на web-страницу
Response.Write("<h1>Список пользователей:</h1>")
Response.Write("<ul>")
while not rs.EOF:
    Response.Write("<li>" + rs.Fields("username").Value +
"</li>")
    rs.MoveNext()
Response.Write("</ul>")
```

```
# Закрытие соединения и освобождение ресурсов
rs.Close()
conn.Close()
%>  # конец блока кода ASP
```

Как видно, ASP предоставляет простой и интуитивно понятный синтаксис, основанный на скриптовом языке, позволяет быстро разрабатывать web-приложения, сокращая время настройки и развертывания. ASP обладает множеством встроенных и сторонних библиотек и модулей, которые упрощают добавление новых функциональных возможностей в web-приложения. Это делает ASP гибкой и расширяемой платформой для разработки как небольших, так и крупных проектов.

В рамках дальнейшего развития web-технологий для платформы Windows была разработана *технология ISAPI* (Internet Server Application Programming Interface). ISAPI предоставляет более низкоуровневый доступ к функциональности web-сервера и позволяет разработчикам создавать более гибкие и мощные расширения для обработки запросов на сервере. Хотя ISAPI и предоставляет возможности для создания расширений для web-серверов, она не так широко используется в сравнении с другими технологиями, такими как CGI, ASP или PHP, особенно в современной web-разработке. Его применение ограничивается в основном Windows-средой и, в определенных случаях, может быть менее эффективным или гибким по сравнению с альтернативными технологиями.

В настоящее время в контексте взаимодействия web-приложений с базами данных более распространенными и эффективными технологиями, помимо уже упомянутого PHP, являются ASP.NET, Django, Ruby on Rails. Эти технологии предоставляют более простые и эффективные способы работы с базами данных по сравнению с ранее описанными технологиями, поэтому они чаще используются в современной web-разработке.

Фреймворк ASP.NET, разработанный компанией Microsoft, предоставляет мощные средства для создания web-приложений на платформе .NET, включая средства взаимодействия с базами данных с использованием Entity Framework и других технологий. ASP.NET включает в себя множество компонентов и инструмен-

тов, которые облегчают разработку, развертывание и управление web-приложениями. ASP.NET использует языки программирования .NET, такие как C# и VB.NET, и обеспечивает более мощные возможности, такие как объектно-ориентированное программирование, встроенная безопасность, средства управления состоянием и многие другие.

Фреймворк Django для Python предоставляет инструменты для быстрой разработки web-приложений с использованием шаблонов ORM для взаимодействия с базами данных. Он автоматически создает соответствие между объектами Python и записями в базе данных, облегчая выполнение операций CRUD без явного написания SQL-запросов. Django предоставляет QuerySet API для выполнения запросов к базе данных. QuerySet позволяет строить сложные запросы, фильтровать, сортировать и агрегировать данные, обеспечивая гибкость и удобство взаимодействия с данными. Инструменты для автоматического создания и применения миграций базы данных, встроенные в Django, позволяют разработчикам легко изменять структуру базы данных и обновлять ее согласованно с изменениями в приложении, минимизируя риски ошибок и конфликтов данных. Фреймворк поставляется с встроенным административным интерфейсом, который автоматически создает интерфейс администратора для спроектированных моделей данных, что позволяет легко управлять данными приложения без необходимости разработки пользовательского интерфейса. Django поддерживает несколько популярных СУБД, включая PostgreSQL, MySQL, SQLite и другие, что позволяет выбирать наиболее подходящее решение для конкретного проекта.

Ruby on Rails, сокращенно Rails, предоставляет мощный фреймворк для создания web-приложений на языке Ruby. Одним из ключевых преимуществ Rails является встроенная поддержка ORM под названием ActiveRecord. ORM ActiveRecord облегчает взаимодействие с базами данных, предоставляя удобный способ работать с объектами базы данных через объектно-ориентированный подход. Использование ActiveRecord позволяет разработчикам создавать модели данных, которые автоматически сопоставляются с таблицами в базе данных. Это позволяет выполнять операции с данными, такие как создание, чтение, обновление и удаление, используя стандартные методы Ruby, что дела-

ет код более понятным и поддерживаемым. Кроме того, ActiveRecord обеспечивает механизмы для управления отношениями между различными сущностями в базе данных, а также поддержку *миграций базы данных* – контролируемый способ изменения структуры базы данных без необходимости вручную написания SQL-скриптов. Таким образом, благодаря ActiveRecord и другим инструментам, предоставляемым Ruby on Rails, разработчики могут эффективно создавать web-приложения, которые взаимодействуют с базами данных, сокращая время разработки и повышая читаемость и поддерживаемость кода.

В заключение приведем сравнительную таблицу, отображающую основные характеристики популярных технологий PHP, ASP.NET, Django и Ruby on Rails, оказывающих существенное влияние на интеграцию БД в web-приложения.

Таблица 3.

Сравнительная характеристика популярных web-технологий

Характеристика	PHP	ASP.NET	Django	Ruby on Rails
Язык	PHP	C#	Python	Ruby
Фреймворк	Необязательно (Laravel, Symfony, CodeIgniter)	.NET (ASP.NET Core, ASP.NET MVC)	Необязательно (Django)	Необязательно (Ruby on Rails)
Web-сервер	Apache, Nginx, IIS	IIS, Kestrel	Apache, Nginx	Puma, Passenger, Unicorn
Поддержка ORM	Да (Doctrine, Eloquent, Propel)	Да (Entity Framework Core)	Да (Django ORM)	Да (ActiveRecord)
Поддерживаемые базы данных	SQLite, MySQL, PostgreSQL, SQL Server, Oracle	SQLite, MySQL, PostgreSQL, SQL Server, Oracle	SQLite, MySQL, PostgreSQL, Oracle	SQLite, MySQL, PostgreSQL
Компоненты	Разработчик сам выбирает	Компоненты включены в платформу	Компоненты включены в платформу	Компоненты включены в платформу