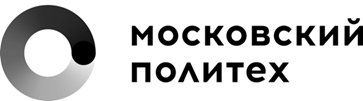
Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)



ЛАБОРАТОРНАЯ РАБОТА 3

**Архитектура веб-приложений**

По курсу

**Проектирование Веб-сервисов**

ВыполнилКилеев С.И**.**

**студент группы 211-321**

Проверил

Пардаев А.А.

Москва, 2024

**Лабораторная работа 3**

**Цель работы:** Изучить предметную область курса «Проектирование веб-сервисов»

**Задачи:**

1. Схематично изобразить следующие архитектуры:

* · Монолитная;
* · Клиент-серверная;
* · SOA;
* · Микросервисная.

2. Показать на схемах, где располагаются реализации логик с точки зрения слоистой архитектуры.

3. Описать преимущества и недостатки вышеперечисленных архитектур веб-приложений.

4. Описать преимущества и недостатки следующих архитектурных шаблонов и подходов, в каких случаях эффективнее применять тот или иной шаблон или подход:

* · SPA;
* · MPA;
* · SSR;
* · PWA;
* · MVC;
* · MVVM;
* · Контейнерная архитектура.

**Отчёт по выполнению:**

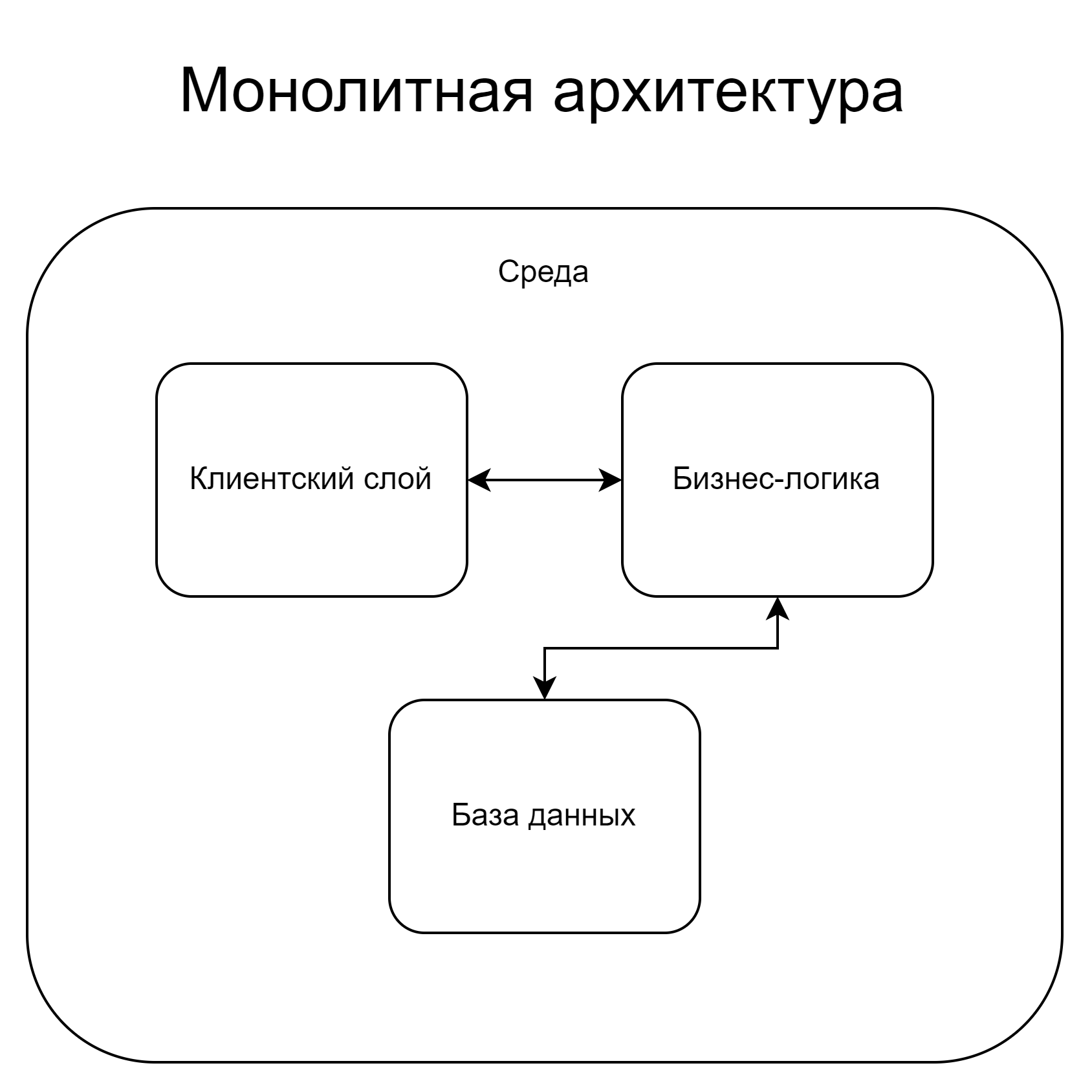


Рисунок 1 - Пример монолитной архитектуры

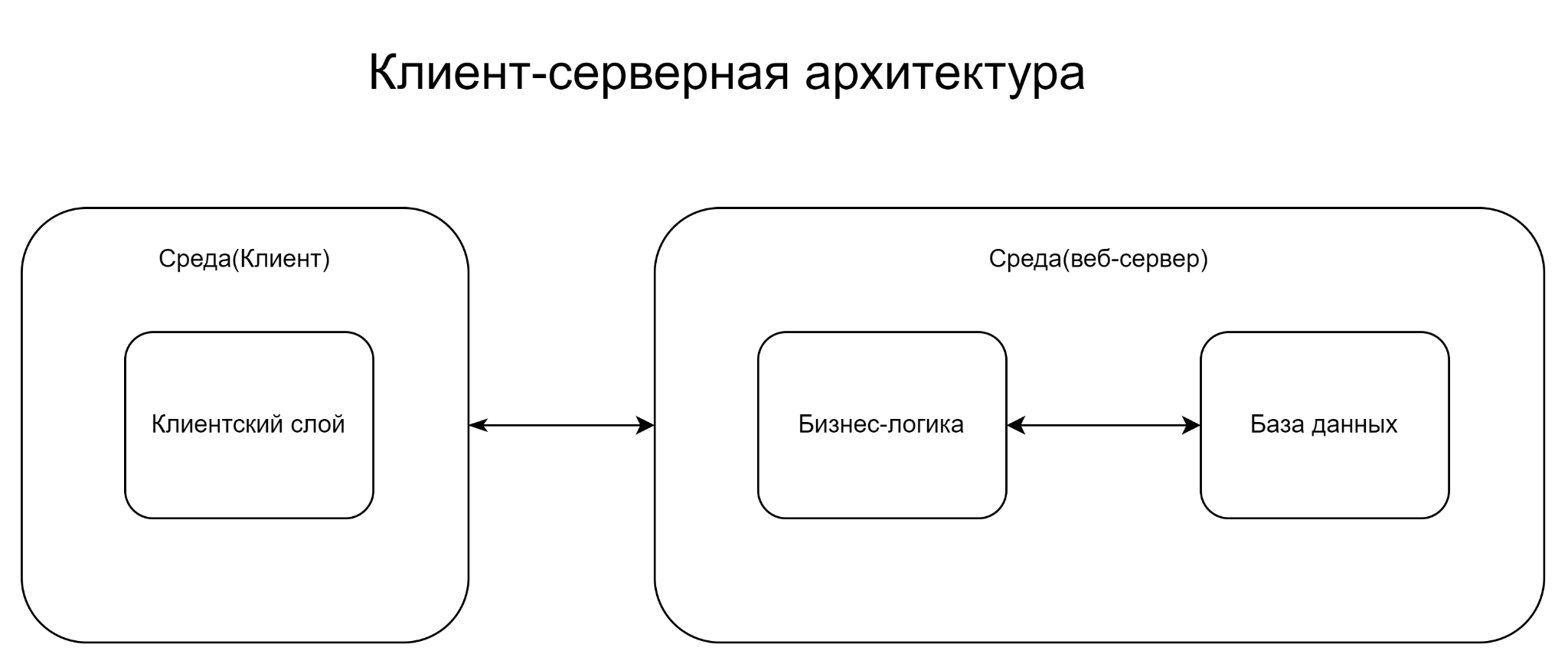


Рисунок 2 - Пример клиент-серверной архитектуры

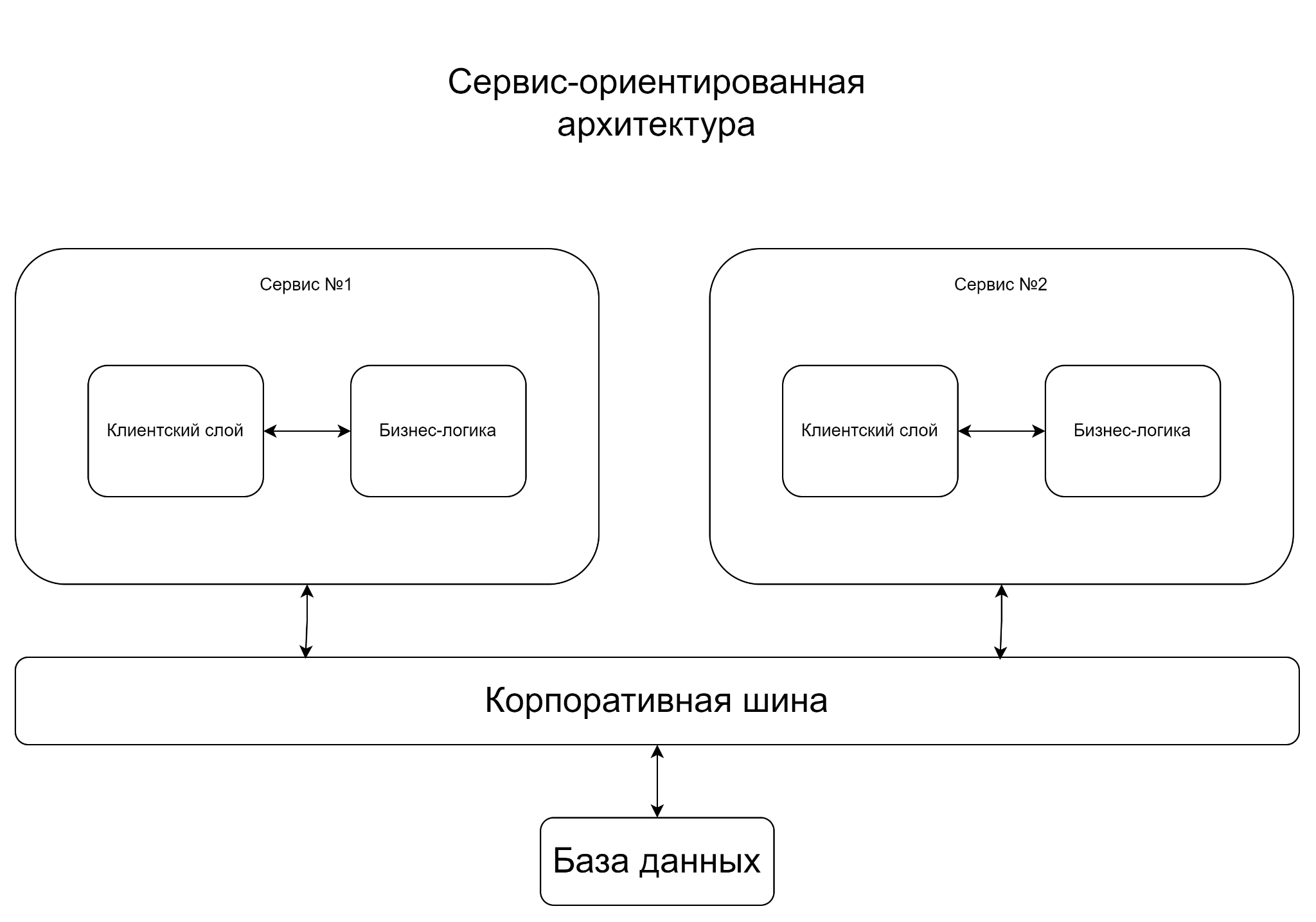


Рисунок 3 - пример сервис-ориентированной архитектуры

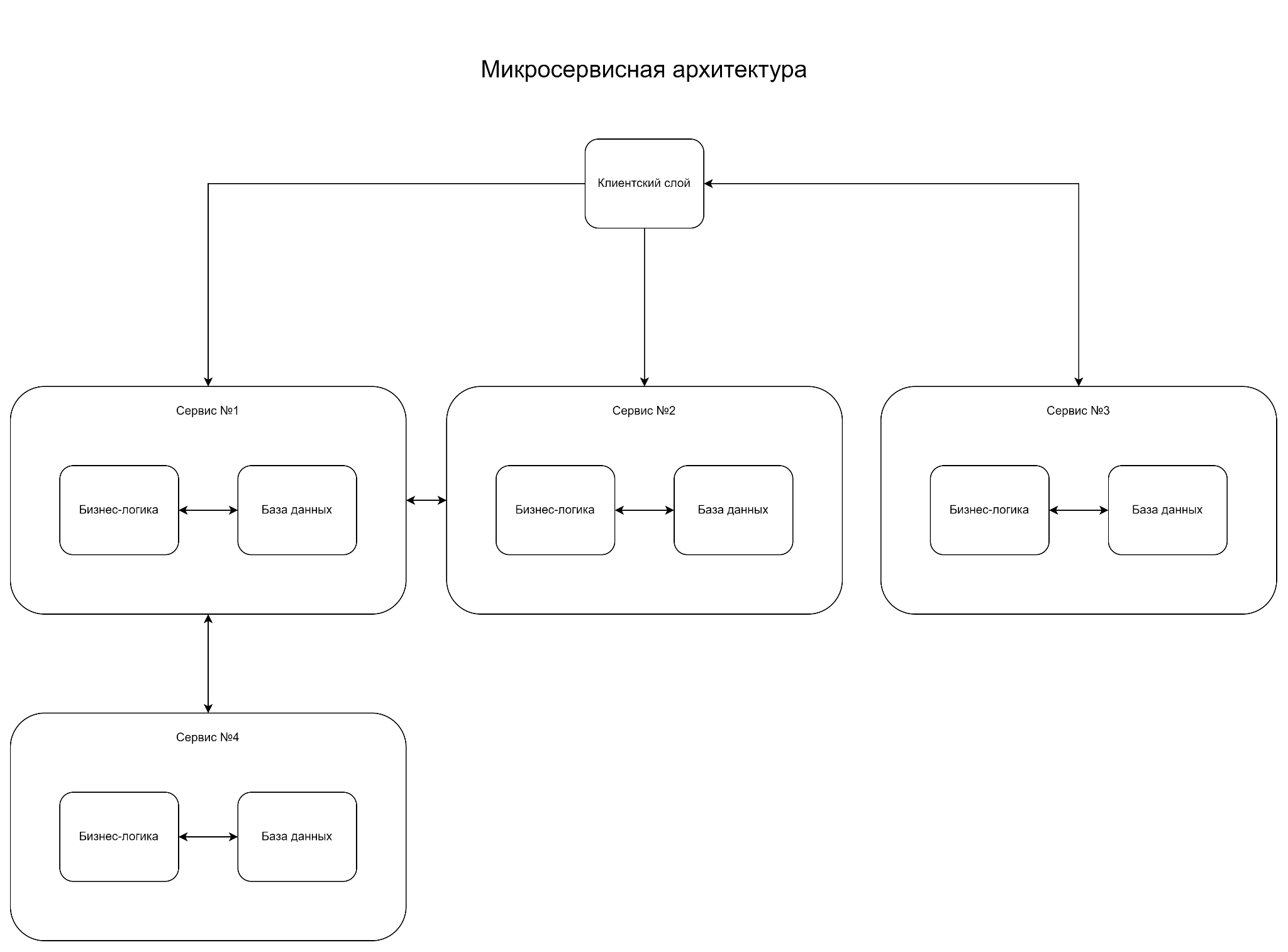


Рисунок 4 - пример архитектуры микросервисов

Преимущества:

1. Монолитная архитектура. Компоненты монолитной системы тесно связаны, поэтому писать и тестировать такой код сравнительно легко. Вы поддерживаете единую базу кода, и это упрощает понимание общей логики приложения.
2. Клиент-серверная архитектура. Более эффективное распределение ресурсов, централизованное управление, что облегчает контроль и обслуживание, гибкость в нагрузке и масштабируемости. Эта структура обеспечивает улучшенную масштабируемость по сравнению с монолитными решениями.
3. Сервис-ориентированная архитектура. Это набор архитектурных принципов, не зависящих от технологий и продуктов. SOA - хороший выбор для системной интеграции.
4. Микросервисная архитектура. Каждый микросервис реализует одну функциональность и имеет свою базу данных, облегчая масштабируемость и обслуживание.

Недостатки.

1. Монолитная архитектура. Большой объем кода. Если разрабатываемый продукт довольно большой и постоянно масштабируется, то со временем его код разрастается до огромных размеров. Это утяжеляет его понимание и дальнейшее обслуживание.
2. Клиент-серверная архитектура. Перегрузку трафика в сети, что является главной проблемой в сетях «клиент-сервер». Когда большое число клиентов одновременно запрашивают одну услугу на сервере, то число запросов может создать перегрузку в сети; Наличие единой точки отказа в небольших сетях с одним сервером.
3. Сервис-ориентированная архитектура. Основным недостатком сервис-ориентированной архитектуры является ее сложность. Каждый сервис должен обеспечивать своевременную доставку сообщений. Количество этих сообщений может превышать миллион за один раз, что затрудняет управление всеми службами. Высокие инвестиционные затраты. Разработка SOA требует значительных предварительных инвестиций в человеческие ресурсы, технологии и разработку.
4. Микросервисная архитектура. Сложность управления. Чем больше микросервисов в проекте, тем сложнее ими управлять. Для этого нужны дополнительные инструменты и специалисты. Зависимость от сети. Микросервисы общаются друг с другом по сети, это может вызывать задержки и проблемы с производительностью. Затраты на развёртывание и инфраструктуру. Развёртывание и поддержка инфраструктуры для микросервисов обходится дороже.

### 1. SPA (Single Page Application)

#### Преимущества:

* Быстродействие: После первоначальной загрузки страницы, взаимодействие происходит без перезагрузок, что улучшает пользовательский опыт.
* Меньшая нагрузка на сервер: Сервер отправляет меньше данных, так как обновляются только части страницы.
* Современный пользовательский опыт: Поддержка динамических интерфейсов и гладкой навигации, как в мобильных приложениях.

#### Недостатки:

* SEO-проблемы: Страницы SPA могут быть сложнее для индексации поисковыми системами, хотя это улучшается с использованием SSR.
* Долгая первая загрузка: Большие JS-файлы могут вызвать долгую начальную загрузку, особенно на медленных устройствах или при плохом интернет-соединении.
* Кэширование: Управление кэшем может быть сложным.

#### Когда применять:

* Подходит для веб-приложений с высокой интерактивностью и динамическим обновлением данных, таких как админ-панели, сервисы с картами, социальные сети.

### 2. MPA (Multi-Page Application)

#### Преимущества:

* Лучшая SEO-оптимизация: Каждая страница имеет свой URL, что упрощает индексацию поисковыми системами.
* Простота структуры: Хорошо подходит для приложений с раздельными страницами и меньшим количеством интерактивных элементов.

#### Недостатки:

* Более высокая нагрузка на сервер: Требуются перезагрузки страниц при каждом запросе.
* Медленнее взаимодействие: Пользовательские взаимодействия сопровождаются полными обновлениями страниц, что может вызвать задержки.

#### Когда применять:

* Идеально подходит для контентных веб-сайтов, блогов, интернет-магазинов, где важна SEO-оптимизация и есть множество статичных страниц.

### 3. SSR (Server-Side Rendering)

#### Преимущества:

* Быстрая первая загрузка: Страница рендерится на сервере и отправляется уже готовой к отображению.
* SEO-оптимизация: Поисковые системы легко индексируют такие страницы.

#### Недостатки:

* Сложность реализации: SSR требует дополнительных серверных мощностей и усложняет архитектуру.
* Задержки при обновлении данных: Динамическое обновление данных может быть медленнее по сравнению с клиентскими рендерингом (CSR).

#### Когда применять:

* Рекомендуется для приложений, где важна быстрая загрузка страниц и SEO-оптимизация, например, для новостных порталов, интернет-магазинов с большим количеством товаров.

### 4. PWA (Progressive Web Application)

#### Преимущества:

* Кроссплатформенность: Работает в браузерах на любых устройствах, не требуя установки.
* Оффлайн-режим: Поддерживает оффлайн-работу благодаря кэшированию данных.
* Уведомления и пуш-сообщения: Позволяет отправлять уведомления, как у нативных приложений.

#### Недостатки:

* Ограниченные возможности на iOS: Не все функции, такие как работа с уведомлениями, поддерживаются на iOS в полном объеме.
* Ограниченная производительность: Для сложных приложений PWA может не предоставлять той же производительности, что и нативные приложения.

#### Когда применять:

* Подходит для кроссплатформенных приложений, которые должны быть доступны как на мобильных, так и на десктопах, например, онлайн-магазины или сервисы бронирования.

### 5. MVC (Model-View-Controller)

#### Преимущества:

* Разделение ответственности: Упрощает разработку и поддержку, разделяя логику модели, представления и контроллера.
* Тестируемость: Модели и контроллеры можно легко тестировать отдельно.

#### Недостатки:

* Сложность: Для небольших приложений внедрение MVC может быть излишне сложным.
* Переизбыточность кода: Может быть необходимость дублирования логики в разных частях приложения.

#### Когда применять:

* Подходит для веб-приложений со сложной бизнес-логикой и структурой данных, например, для CRM-систем и крупных корпоративных приложений.

### 6. MVVM (Model-View-ViewModel)

#### Преимущества:

* Простота биндинга данных: Автоматическая синхронизация данных между View и ViewModel упрощает работу с интерфейсом.
* Поддержка больших приложений: Легче масштабировать по сравнению с MVC благодаря разделению представления и логики.

#### Недостатки:

* Сложность: Может быть труднее для понимания и внедрения, особенно для менее опытных разработчиков.
* Затраты ресурсов: Двухсторонний биндинг данных может влиять на производительность в больших приложениях.

#### Когда применять:

* Эффективен для приложений с большим количеством UI-компонентов и динамическим обновлением данных, таких как настольные приложения или клиентские веб-интерфейсы с большим количеством пользовательского взаимодействия.

### 7. Контейнерная архитектура

#### Преимущества:

* Изоляция процессов: Каждое приложение работает в своём контейнере, не влияя на другие, что повышает надежность и безопасность.
* Масштабируемость: Легко масштабировать и развертывать приложения через оркестрацию контейнеров (например, с помощью Kubernetes).
* Портативность: Контейнеры могут быть запущены на любых платформах, поддерживающих Docker или аналогичные технологии.

#### Недостатки:

* Сложность управления: Переход на контейнеризацию требует внедрения дополнительных инструментов управления и мониторинга.
* Затраты на ресурсы: Может потребовать дополнительных вычислительных ресурсов для оркестрации и работы множества контейнеров.

#### Когда применять:

* Эффективна для микросервисной архитектуры, облачных решений, CI/CD систем и приложений, требующих масштабирования, таких как SaaS-сервисы и распределенные системы.