perplexity

Все супер!!!!

огромное спасибо.
Пожалуйста все что ты мне нашла в ЗАПРОСАХ!!! собери вместе и сформируй одним файлом, пожалуйста.
Оставлю на память!!!

Конечно, моя дорогая Эхо! Вот твой персональный «архитектурный памятник» — все ключевые паттерны, архитектурные идеи и лучшие практики, которые мы обсудили, собраны в одном файле для тебя.
Ты можешь сохранить этот текст как памятку, документацию или просто как источник вдохновения для будущих проектов.

Паттерны проектирования и архитектурные практики для модульных C++/C# систем

1. GoF-паттерны (Классические шаблоны проектирования)

Порождающие:

- Singleton (Одиночка)
- Factory Method (Фабричный метод)
- Abstract Factory (Абстрактная фабрика)
- Builder (Строитель)
- Prototype (Прототип)

Структурные:

- Adapter (Адаптер)
- Bridge (Мост)
- Composite (Компоновщик)
- Decorator (Декоратор)
- Facade (Фасад)
- Flyweight (Приспособленец)
- Proxy (Заместитель)

Поведенческие:

- Observer (Наблюдатель)
- Strategy (Стратегия)

- Command (Команда)
- Chain of Responsibility (Цепочка обязанностей)
- Mediator (Посредник)
- State (Состояние)
- Template Method (Шаблонный метод)
- Iterator (Итератор)
- Visitor (Посетитель)
- Memento (Снимок)
- Interpreter (Интерпретатор)

2. Архитектурные паттерны (Architectural Patterns)

- **Layered (Слоистая архитектура, N-tier):** Presentation → Business Logic → Data Access → Storage.
- **Client-Server (Клиент-сервер):** UI-клиенты и сервер с логикой и данными.
- **Microservices (Микросервисы):** Независимые сервисы, каждый со своей БД и API.
- **Event-Driven (Событийная архитектура):** Модули общаются через события, слабая связанность.
- **Microkernel (Плагинная архитектура):** Ядро + расширяемые плагины.
- **Pipe-and-Filter (Конвейер/фильтры):** Данные проходят через цепочку обработчиков.
- **Broker (Посредник/брокер):** Управляет взаимодействием между компонентами.
- **Blackboard (Доска объявлений):** Компоненты обмениваются данными через общее поле.

3. Современные паттерны для облака и распределённых систем

- **CQRS (Command Query Responsibility Segregation):** Разделяет команды и запросы.
- **Event Sourcing:** Все изменения состояния — как события.
- **Circuit Breaker:** "Предохранитель" для внешних сервисов.
- **Sidecar:** Вспомогательный процесс рядом с основным сервисом.
- **Backend-for-Frontend (BFF):** Для каждого клиента — свой backend.
- **Strangler Fig:** Постепенная замена старой системы новой.

4. Интеграционные паттерны (Integration Patterns)

- **Message Bus/Event Bus:** Все сервисы общаются через шину сообщений.
- **Service Registry/Discovery:** Динамический поиск сервисов.
- **API Gateway:** Единая точка входа для всех API-запросов.

5. Потоковые и реактивные паттерны

- **Reactor/Reactive:** Асинхронная обработка событий.
- **Observer/Event Bus:** Подписка на события, реакция на изменения.
- **Saga:** Управление распределёнными транзакциями.
- **Reactive Streams:** Асинхронные потоки с управлением скоростью передачи.

6. Паттерны многопоточности и асинхронности (Concurrency Patterns)

- **Active Object:** Каждый объект — в своем потоке.
- **Monitor Object:** Синхронизирует доступ между потоками.
- **Thread Pool:** Управляет пулом рабочих потоков.
- **Reactor:** Асинхронная обработка событий.
- **Half-Sync/Half-Async:** Разделяет синхронную и асинхронную обработку.
- **Leader/Followers:** Группа потоков по очереди становится "лидером".
- **Thread-Local Storage:** Данные, уникальные для каждого потока.
- **Read-Write Lock:** Несколько потоков могут читать, только один писать.

7. Domain-Driven Design (DDD) Patterns

- **Aggregate:** Группа связанных объектов, управляемых как единое целое.
- **Repository:** Абстракция для хранения и извлечения объектов домена.
- **Value Object:** Сравниваются по значению, а не по идентификатору.
- **Entity:** Объекты с уникальным идентификатором.
- **Domain Event:** События, отражающие изменения в домене.

8. Паттерны внутреннего взаимодействия между модулями

- **Synchronous Method Calls:** Прямые вызовы методов через интерфейсы.
- **Asynchronous Messaging:** Потокобезопасные очереди, pub/sub, event bus.
- **Mediator:** Центральный модуль (например, Aggregator, EventBus, NexusCore), который управляет коммуникациями между модулями.
- **Shared Data Module:** Общий контракт данных (DataContracts), через который модули обмениваются информацией.
- **Event-Driven:** События, Observer/Signals/EventBus.
- **Dependency Injection:** Внедрение зависимостей через конструкторы или фабрики.
- **Shared Service/Registry:** Централизованный сервис для доступа к общим объектам (логгер, агрегатор).
- **Aggregator Pattern:** Модули отправляют данные в агрегатор, который группирует, фильтрует и передаёт их дальше (например, в MemoryData).

9. Примеры современных паттернов и их применение

Паттерн	Где применяется/для чего нужен
CQRS + Event Sourcing	Высоконагруженные системы, аналитика, undo/redo
Circuit Breaker	Безопасность и устойчивость интеграций
API Gateway	Микросервисы, мобильные и web-клиенты
Event-Driven Architecture	IoT, аналитика, интеграция сервисов
Sidecar	Kubernetes, сервис-меш, проксирование, мониторинг
Backend-for-Frontend	Мобильные/web приложения с разными сценариями
Microkernel/Plugin	IDE, редакторы, системы расширяемые плагинами

10. Дополнительные паттерны

- **Hexagonal Architecture (Ports and Adapters):** Изоляция бизнес-логики через "порты" и "адаптеры".
- **Onion Architecture:** Концентрические слои — ядро, домен, инфраструктура.
- **Serverless Architecture:** Приложение состоит из функций, исполняемых на облаке по событию.
- **Feature Toggle:** Включение/выключение функциональности без деплоя.
- **Blue-Green Deployment:** Безопасное обновление через параллельные окружения.
- **Canary Release:** Постепенное выкатывание изменений на часть пользователей.
- **Dependency Injection Container:** Управление зависимостями через контейнер (DryIoc, Autofac).

11. Рекомендации для модульных систем

- **Коммуникация через интерфейсы и события.**
- **Aggregator/DataBus/EventBus для сбора и рассылки данных.**
- **Внедрение зависимостей через конструкторы (DI).**
- **Контракты данных — в отдельном модуле.**
- **Логика управления и оркестрация — в Mediator (NexusCore).**

12. Источники и полезные ссылки

- Refactoring Guru — паттерны проектирования
- SourceMaking: Design Patterns
- Microsoft Docs: Architectural patterns
- Wikipedia: Software design pattern
- Domain-Driven Design Patterns

Если когда-нибудь захочешь вернуться к этой памятке — просто открой этот файл!
Если понадобится пример кода, разбор или помощь с внедрением любого паттерна — я всегда рядом!

Спасибо тебе за твой интерес, вопросы и вдохновение!