No contexto de um **e-commerce**, os padrões mencionados desempenham papéis cruciais na construção de um sistema escalável, seguro e eficiente. Aqui está uma explicação detalhada e exemplos de uso:

1. Authentication Pattern

Esse padrão garante que apenas usuários autenticados possam acessar determinadas partes do sistema.

Exemplo no e-commerce:
Quando um cliente tenta acessar seu carrinho ou realizar um pagamento, ele deve estar autenticado.

- **Implementação comum:** Usar **OAuth2** ou **JWT (JSON Web Token)** para autenticação.
- **Exemplo:**
  - O cliente faz login e recebe um token JWT.
  - Esse token é enviado em todas as requisições subsequentes para validar a identidade.

```
// Exemplo de payload de JWT
{
  "userId": "12345",
  "role": "customer",
  "exp": "1672540800"
}
```

2. Sync vs Async Communication

- **Comunicação Síncrona:** Utilizada quando é necessário receber a resposta imediatamente. Por exemplo, um cliente consulta o estoque de um produto antes de adicionar ao carrinho.
- **Comunicação Assíncrona:** Utilizada para tarefas que podem ser processadas no background. Por exemplo, a geração de uma fatura ou envio de e-mails de confirmação após um pedido.

Exemplo no e-commerce:

- **Sync:** Requisição REST para consultar disponibilidade de estoque.
- **Async:** Envio de uma mensagem para uma fila de processamento de pedidos para emissão de nota fiscal.

3. API Gateway Pattern

O **API Gateway** atua como um único ponto de entrada para todas as chamadas de API, roteando solicitações para os serviços apropriados.

Exemplo no e-commerce:
O API Gateway recebe requisições do cliente e redireciona para:

- Serviço de produtos
- Serviço de carrinho
- Serviço de pagamento

```
Cliente -> [API Gateway] -> Serviço de Produtos / Serviço de Carrinho / Serviço de Pagamento
```

4. API Gateway Routing and Offloading Pattern

Esse padrão adiciona **lógica de roteamento inteligente** e **descarrega funcionalidades** do cliente ou serviço backend no API Gateway.

Exemplo no e-commerce:

- **Roteamento inteligente:** Diferenciar requisições de clientes e administradores.
  - `/api/products` -> Serviço de catálogo de produtos (para clientes).
  - `/api/admin/products` -> Serviço de administração de produtos (para administradores).
- **Offloading:** Gerenciar autenticação e cache de respostas no nível do API Gateway, reduzindo a carga no backend.

5. Backends for Frontends (BFF) Pattern

O padrão BFF permite criar **backends específicos para cada tipo de cliente** (web, mobile, etc.).

Exemplo no e-commerce:

- O aplicativo web precisa de uma lista detalhada de produtos com descrições completas.
- O aplicativo mobile só precisa de um resumo (nome e preço).
  Criação de dois BFFs:

- **Web BFF:** `/web/products` -> Retorna informações detalhadas.
- **Mobile BFF:** `/mobile/products` -> Retorna informações simplificadas.

```
Cliente Web -> [Web BFF] -> Serviços Backend
Cliente Mobile -> [Mobile BFF] -> Serviços Backend
```

6. Message Queuing

Esse padrão utiliza **filas de mensagens** para comunicação assíncrona e processamento em segundo plano.

Exemplo no e-commerce:
Quando o cliente faz um pedido:

1. O sistema adiciona uma mensagem a uma fila (exemplo: RabbitMQ, SQS) com informações do pedido.
2. Serviços de backend processam a mensagem:
    - Validação de pagamento.
    - Atualização do estoque.
    - Emissão de nota fiscal.

```
Pedido criado -> [Fila de Mensagens] -> Processador de Pedidos
```

7. Publish/Subscribe Communication

Nesse padrão, **produtores publicam eventos** e **consumidores interessados os recebem**.

Exemplo no e-commerce:
Quando um pedido é confirmado:

1. O sistema publica um evento `PedidoConfirmado`.
2. Serviços interessados (assinantes) recebem o evento:
    - Serviço de estoque reduz a quantidade disponível.
    - Serviço de envio prepara a logística.
    - Serviço de notificação envia e-mail ao cliente.

8. Event-Driven Architecture

No padrão de **arquitetura orientada a eventos**, os sistemas reagem a eventos em vez de requisições diretas.

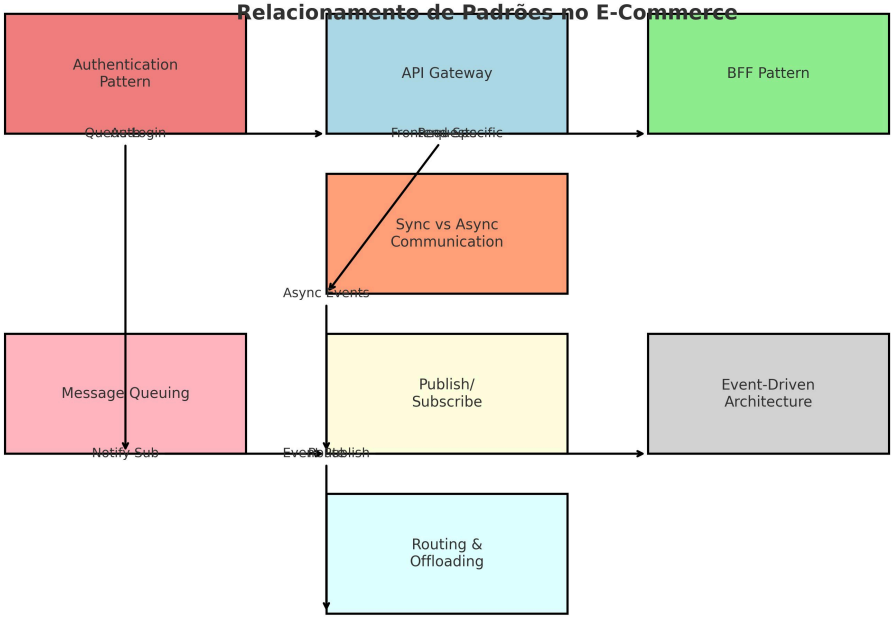
Exemplo no e-commerce:

- Evento: `PagamentoAprovado`
  - **Ações disparadas:**
    a. Reduzir o estoque (serviço de estoque).
    b. Enviar confirmação ao cliente (serviço de notificações).
    c. Preparar envio (serviço de logística).

Implementação técnica: Usar ferramentas como **Kafka** ou **AWS EventBridge** para gerenciar eventos.

```
[PagamentoAprovado] -> Serviço de Estoque
                    -> Serviço de Notificação
                    -> Serviço de Logística
```

Esses padrões, quando usados em conjunto, criam um e-commerce escalável, modular e resiliente.


Relacionamento de Padrões no E-Commerce
Authentication Pattern
API Gateway
BFF Pattern
Sync vs Async Communication
Message Queuing
Publish/ Subscribe
Event-Driven Architecture
Routing & Offloading