1. Arquitetura em Camadas (Layered Architecture)

Como aplicar:
A arquitetura em camadas é uma escolha bastante comum e adequada para a aplicação. Poderíamos dividi-la em, por exemplo, camadas de **Apresentação**, **Lógica de Negócio**, **Acesso a Dados** e **Banco de Dados**.

- **Apresentação (Front-end):** Responsável pela interface do usuário (UI) — sejam aplicativos móveis, web ou ambos — onde o estudante e o prestador de serviço interagem, avaliam preços, disponibilidade e histórico de avaliações.
- **Lógica de Negócio (Backend):** Camada que implementa as regras de negócio, como o cadastro de prestadores, o cálculo de avaliações, a gestão de solicitações de serviço, a orquestração de pagamentos e o gerenciamento de disponibilidade e preços.
- **Acesso a Dados (DAO/Repository):** Responsável por interagir com o banco de dados, converter objetos de domínio em consultas e resultados de consultas em objetos.
- **Banco de Dados:** Onde são armazenadas informações sobre usuários, prestadores, serviços, avaliações e transações.
  A separação clara de camadas facilita manutenção, escalabilidade e a substituição ou evolução de cada camada independentemente. Portanto, esse estilo é aplicável.

2. Cliente-Servidor (Client-Server)

Como aplicar:
Naturalmente, o sistema já se enquadra em um paradigma cliente-servidor. Os estudantes (clientes) acessam o sistema por meio de um aplicativo móvel ou web, enquanto os prestadores de serviço (também clientes finais) usam aplicações semelhantes ou portais diferenciados. O servidor central (backend) provê dados, processa requisições, consulta o banco de dados e retorna as informações. É um padrão básico e inteiramente aplicável, garantindo controle centralizado de dados e lógica.

3. Model-View-Controller (MVC)

Como aplicar:
No desenvolvimento do front-end (web, por exemplo) ou mesmo em frameworks de desenvolvimento móvel, o MVC (ou variações como MVVM ou MVP) é comum. Por exemplo:

- **Model:** Classes que representam serviços, usuários, avaliações, transações.
- **View:** Telas do aplicativo (listagem de serviços, tela do prestador, formulário de contratação).
- **Controller:** Controladores que recebem a entrada do usuário (como cliques, pesquisas), interagem com o Model e atualizam a View.

4. Broker (Arquitetura de Corretagem)

Como aplicar:
A arquitetura de corretagem (Broker) é comumente usada para sistemas distribuídos em que intermediários (brokers) coordenam a comunicação entre componentes distribuídos. No contexto da "Handle", seria possível adotar essa abordagem caso se deseje uma arquitetura mais distribuída e escalável, onde o "broker" gerencia o roteamento de requisições de serviços entre clientes (estudantes) e servidores específicos (módulos de serviços ou prestadores).
Por exemplo, a plataforma poderia ter um broker que recebe solicitações dos clientes e direciona ao módulo responsável por cadastrar o prestador, obter preço, ou retornar avaliações. Também poderia ser usado caso a aplicação se integrasse a outros serviços externos (por exemplo, sistemas de pagamento ou verificação de identidade), abstraindo a complexidade das integrações.
Embora menos comum no desenvolvimento tradicional de aplicativos, o estilo é possível e pode ser vantajoso caso a aplicação cresça em complexidade e extensão.

5. Pipes-and-Filters (Tubos e Filtros)

Como aplicar:
A arquitetura de Pipes-and-Filters é geralmente utilizada em sistemas onde dados passam por uma série de transformações lineares, cada uma em um "filtro", e onde a saída de um filtro é a entrada do próximo. É mais indicada para processamento de fluxos de dados, conversões, parsing, análises ou transformações contínuas.
No caso da "Handle", não há um fluxo constante de dados transformados em cadeia. O sistema não consiste em uma série de etapas de processamento linear de informações, mas sim em requisições pontuais do cliente ao servidor e retornos de dados.
Poderia haver um uso pontual interno para funcionalidades específicas, como pipelines de tratamento de dados para análise de tendências de contratação ou processamento de logs. Porém, como arquitetura principal não se mostra ideal. O foco do aplicativo não é transformar dados de forma sequencial, mas sim gerenciar e apresentar informações sobre serviços. Assim, **como arquitetura principal, Pipes-and-Filters não é adequado**.

6. Peer-to-Peer (P2P)

Como aplicar:
Uma arquitetura peer-to-peer implicaria que usuários e prestadores se conectassem diretamente uns aos outros, sem um servidor central. Esse modelo tornaria mais complexo o controle de qualidade, a verificação de avaliações, a implementação de pagamentos seguros e a garantia de confiabilidade dos dados.
Em um marketplace como a "Handle", a presença de um servidor central é crucial para mediar transações, manter o histórico de avaliações e gerenciar informações de forma confiável. A arquitetura P2P também dificulta a implementação de funcionalidades típicas de marketplaces, como disputa de reclamações, mediação, verificação de usuários, e compliance legal (notas fiscais, por exemplo).
Portanto, **uma arquitetura P2P não é recomendada**, pois a "Handle" depende de uma fonte confiável centralizada para orquestrar todo o processo, mantendo segurança e integridade das informações.

Em resumo:

- **Arquitetura em Camadas:** Viável e recomendada.
- **Cliente-Servidor:** Naturalmente aplicável.
- **Model-View-Controller (MVC):** Aplicável no front-end e/ou no back-end, facilita a separação de responsabilidades.
- **Broker:** Pode ser aplicado caso o sistema cresça em complexidade e distribuição, mas não é estritamente necessário no início.
- **Pipes-and-Filters:** Não é adequado como estilo principal, podendo ter uso restrito em casos específicos.
- **Peer-to-Peer:** Não se encaixa no modelo de negócios e nas necessidades de centralização, controle e confiabilidade.