Com essa estrutura de arquivos, o fluxo do seu projeto **AllSkinWR** fica muito claro. Vou explicar como a lógica se distribui entre esses arquivos específicos, usando o exemplo real de quando o usuário seleciona um campeão:

1. Funcionalidade: Champions (A Escolha)

Esta feature cuida de listar e gerenciar a seleção do personagem.

Infra (`championsRepository.js`): É o único que toca no arquivo `champions.json`. Ele lê os dados brutos e os entrega quando solicitado.
Domain (`Champion.js`): Define o que é um Campeão. Se você precisar de uma regra como "este campeão é um Atirador?", essa lógica fica aqui dentro como um método.
Application (`getChampionsUseCase.js`): É o "gerente". Ele pede os dados ao repositório, transforma os dados do JSON em objetos da classe `Champion` e entrega a lista pronta para a interface.
UI (`championsSelectController.js` & `View.js`): O **Controller** ouve o clique do usuário no select; a **View** preenche as opções do `<select>` no HTML.

2. Funcionalidade: Skins (A Busca de Dados)

Esta feature foca em buscar e organizar as aparências do campeão escolhido.

Infra (`skinsRepository.js`): Acessa o arquivo `skinColections.json`. Ele tem a função técnica de filtrar as skins que pertencem ao ID do campeão selecionado.
Domain (`Skin.js`): Modela a skin (nome, raridade, URL da imagem). Aqui ficaria a lógica para decidir, por exemplo, se a borda da skin deve ser dourada (raridade lendária).
Application (`getSkinsByChampionUseCase.js`): Orquestra a busca. Ele recebe o ID da Ashe (vindo da UI), pede ao repositório as skins dela, e garante que o sistema receba uma lista de objetos `Skin` prontos para uso.

3. Funcionalidade: Gallery (A Exibição no Carrossel)

Esta feature é responsável pelo comportamento visual do carrossel.

Domain (`Gallery.js`): **Diferente das outras, esta não lida com JSON.** Ela cuida da lógica do slide: "qual é o índice atual?", "quem é o próximo?". O método `nextSlide()` para calcular o loop do carrossel fica aqui.

Application (`loadGalleryUseCase.js`): Prepara a galeria. Ele pega as skins que vieram da feature de `skins` e as organiza dentro do modelo de dados da `Gallery`.
UI (`galleryController.js` & `View.js`): O **Controller** inicia o *autoplay* (setInterval) e gerencia os cliques nas setas. A **View** faz o trabalho pesado de manipular o DOM: cria as tags `<img>`, aplica o efeito de *fade* com CSS e atualiza os títulos e parágrafos na tela.

Resumo do "Telefone Sem Fio" entre os arquivos:

O usuário muda o select → **`championsSelectController.js`** (UI) percebe.
O Controller avisa o **`getSkinsByChampionUseCase.js`** (App).
O Use Case pede as skins para o **`skinsRepository.js`** (Infra).
O Repositório lê o JSON e cria objetos **`Skin.js`** (Domain).
O Use Case devolve essas skins para o **`galleryController.js`** (UI).
O Gallery Controller usa a lógica de **`Gallery.js`** (Domain) para saber quem mostrar primeiro.
O Gallery Controller manda a **`galleryView.js`** (UI) desenhar tudo no HTML.

Essa separação garante que, se você mudar o design do carrossel (alterando apenas a `galleryView.js`), toda a lógica de busca de dados e regras de campeões continuará funcionando perfeitamente.