**Problema:**

*TI:* Como posso te ajudar?

*Cliente:* Conta uma história...

*Dica:* Leve um notebook, gravador de voz (com a devida permissão) e registre toda a história. Não é feito nenhum tipo de detalhe técnico.

*O que fazer depois da reunião?*

Analisar tudo o que foi levantado, estimar o número de horas e propor o orçamento para o cliente.

Caso, aprovado, é elaborado um contrato. Que contem, tudo o que foi levantado na reunião, tudo que vai ter no sistema e o que não vai ter também. O contrato deve ser o mais claro possível.

Obs: tudo o que não estiver no contrato, será feito adequando o orçamento.

--

*Fase de prototipagem - Apresentando o protótipo*

É uma parte fundamental, até para fazer um contrato com o cliente. Envolve uma documentação de forma mais confortável e não oficial junto ao cliente, exemplo:

- HTML - Fazer o protótipo com html puro e imprimir.

- No Papel

É a primeira oportunidade de validar se os ambos entenderam a mesma coisa. Poupando tempo antes mesmo de começar o desenvolvimento.

--

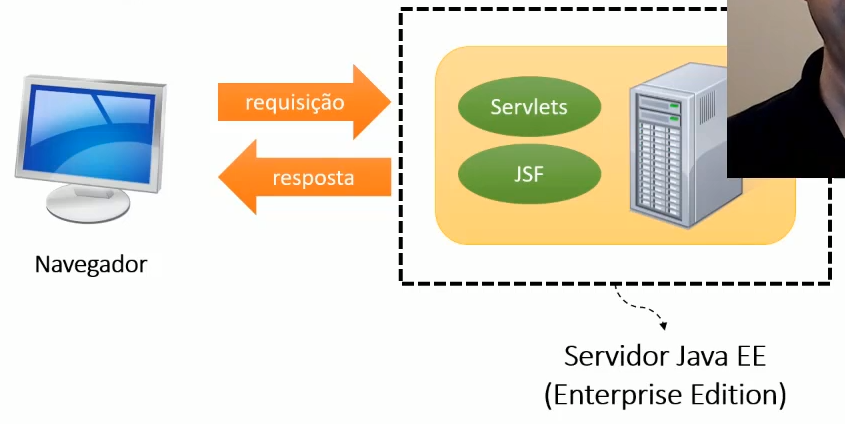
**O Modelo Cliente-Servidor**

O *cliente* é o cara que inicia a comunicação. Ele faz uma *requisição* para o *servidor*. Por sua vez, o *servidor* processa a *requisição* e devolve uma *resposta*. Quem faz o papel do cliente, é o navegador. Já o servidor, fornece as páginas para o cliente.



* Servidores web (web server) podem entregar apenas conteúdos estáticos (arquivos que já existem).

Para gerar conteúdo dinâmico, é preciso mais do que um web server. É necessário um servidor de aplicações:

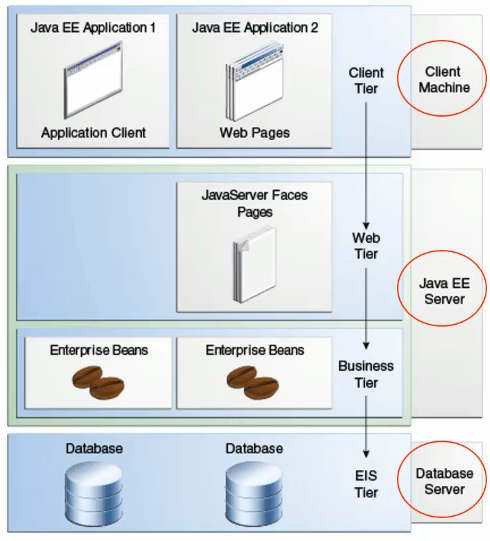


Exemplo: Tomcat, WildFly, GlassFish etc.

A resposta é montada dinamicamente, mas o browser continua recebendo o conteúdo HTML. Para o cliente só interessa o HTML, não importa se do lado do servidor a aplicação é Java, .net, php etc. O que o cliente sabe fazer? Ele sabe receber HTML do servidor e renderizar. O papel do cliente é, requisitar, recebe e mostrar.

**Servidor Java EE**

Ao falarmos de servidor Java EE, estamos falando de *divisões em camadas*.

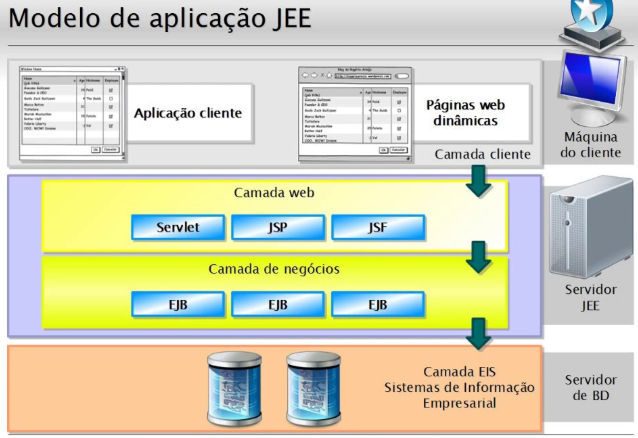
Client Tier: É onde o cliente está acessando a aplicação. Através do navegador. Representadas por páginas HTML.

Servidor Java EE: Temos duas camadas.

1. Camada web: JSF é o responsável por montar as páginas HTML dinâmicas que serão entregues para o cliente.
2. Camada de negócio: é onde será executada a lógica de negócio. EJB + JPA + CDI (API responsável pela injeção de dependência, para fornecer objetos).

O Java EE é uma coleção de tecnologias, ali dentro existe várias APIs e algumas são para trabalhar com a parte web.

1. Servidor do Banco de Dados. É um servidor separado do Servidor Java EE. MySQL, Oracle, SQL Server etc. EIS = Enterprise Information Systems.



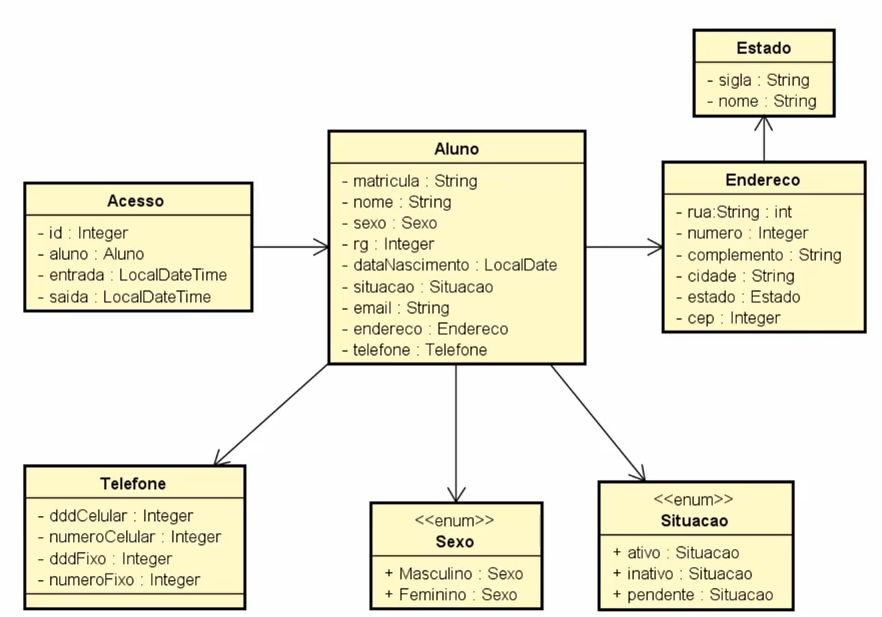
**Modelando as Classes de Domínio**

Esse passo deve ser muito bem pensando e muito bem-feito. Pois se não for encontrado problemas nesta fase, lá na frente uma série de problemas surgirá. *O que é classe de domínio?*

São as classes que representam o núcleo do problema a ser resolvido. Representam *os conceitos* do sistema. São todas as classes do sistema? Não, de forma nenhuma.

*UML*, linguagem de modelagem e documentação de sistemas, composta por vários diagramas.

Exemplo:



É muito comum em sistemas orientados a objetos, alguns atributos ter o tipo, como outra classe. Gerando um mapeando/relacionamento entre as classes, exemplo Aluno -> Endereço; Aluno -> Telefone; importante, cada classe representa um conceito em específico.

Cada objeto Acesso representa uma entrada ou saída da academia.

**DDD Domain-Driven Design**

Significa Projeto Orientado a Domínio. É antes de tudo, comunicação. No DDD modelagem e implementação andam juntos. O domínio de um software são as atividades desempenhadas pelo usuário e a área de interesse destes. E um modelo é “uma simplificação. É uma interpretação da realidade que abstrai os aspectos relevantes para resolver o problema em questão e ignorar detalhes alheios a isto” (Evans - tradução livre).

<https://www.lambda3.com.br/2017/10/desmistificando-o-ddd/>

–

**Serviços**

No modelo DDD existe o conceito de *Application Service* que são serviços utilizados para coordenar a execução dos objetivos. Exemplo: Você precisa que o seu Bean busque os estados no BD, existirá um serviço que fará a busca.

O *Application Service*, por sua vez, não acessa diretamente o BD. O acesso é feito por classes que implementam o conceito de *Repository*. Que são as classes que tem acesso a informações externas.

--

**O que é arquitetura JSF?**

É um framework para construir interfaces de usuários baseadas em componentes para aplicativos da web.

*return null* = retornar para a própria página.

O JSF trabalha com a extensão XHTML.

*O que são templates de JSF?*

Uma aplicação normalmente tem várias páginas e normalmente todas tem um layout em comum, seja o menu, topo, rodapé etc. A estrutura das páginas é a mesma. E é por isso que existe os templates JSF.

*O que é o Bean do JSF?*

A página JSF, sempre trabalha em conjunto com uma classe Java. Como se fosse um representante do Java na página JSF, esse representante é chamado Bean. *Binding*, é a ligação do campo do formulário com o atributo do Bean, exemplo: <input type="text" \*jsf:value="#{alunoBean.aluno.nome}" name="nome" id="nome" />

–

**Anotações**

@Embedded → A JPA sabe que a entidade Endereco e Telefone fazem parte da entidade Aluno e compartilharão o mesmo *id* do Aluno. A classe receberá o compartilhamento é anotada com @Embeddable em vez de @Entity ou @Table.

@PersistenceContext, utilizado para fazer a injeção de dependências (CDI).

@EJB, injeta a referência de um EJB.

É um componente que executa a lógica de negócio,, pode utiliza o CDI para injetar dependência é o componente ideal para executar lógica. Existem EJB do tipo Stateless e Statefull.

@Stateless é um EJB que não mantêm estado, isso significa que tudo o que é executado (chamada de método) é terminado naquele método. Não pode usar atributos, pois ele é muito escalável, pode ter acessos simultâneos e não tem um cliente em específico. Ele é reaproveitado entre clientes e caso tenha atributos isso pode gerar inconsistência de dados.

@RequestScoped -> significa que o objeto só vai sobreviver durante o processo de requisição, na resposta ele já não existe mais.

@SessionScoped garante que o Bean ficara ativo enquanto o navegador do usuário estiver aberto e fazer com que a lógica de exclusão funcione.

Dica: Primeiro a gente faz a coisa funcionar, para depois acertar os detalhes.

Sempre que fazer uma alteração, testa o que está fazendo na hora, porque senão tu só vai descobrir o erro lá na frente.