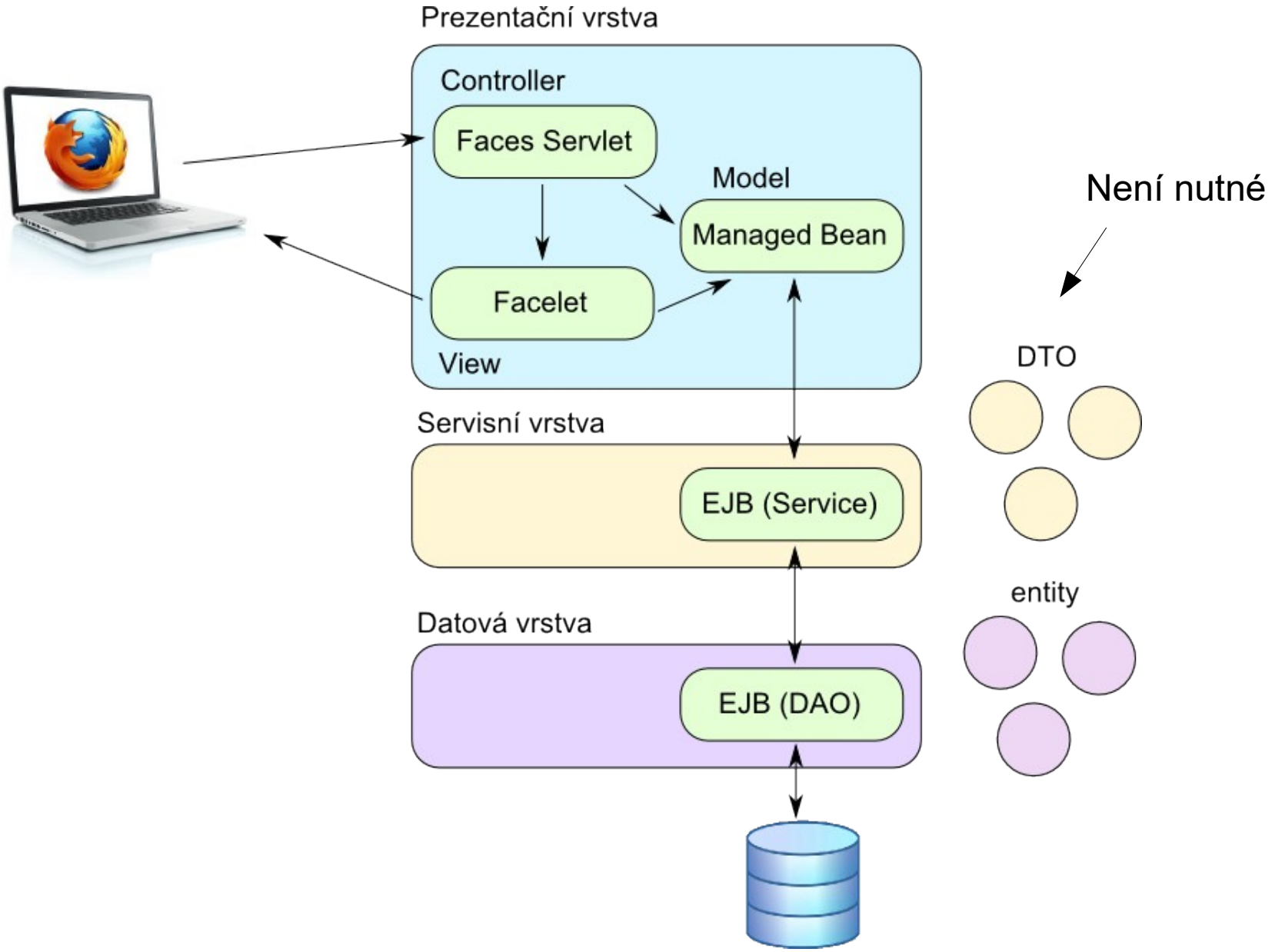
Architektura JSF aplikace

JSF & MVC & třívrstvá architektura



Managed Bean v modelu

- Managed bean v modelu se také dříve říkalo backing bean.
- Je to fyzicky mini-controller, který spojuje View (facelet) a entity / DTO, se kterými pracuje pomocí servisní vrstvy (EJB).
- U těchto tříd se používá jedna z následujících jmenných konvencí (neexistuje oficiální konvence):
  - `Item`
  - `ItemBean`
  - `ItemBacking`
  - `ItemManager`
  - `ItemController`
  - `ItemBackingBean`
  - `ItemManagedBean`
  - `ItemBB`
  - `ItemMB`
  - `ItemView`

Poznámka: Názvy bean pro entitu s názvem `Item`

Best Practices

- Pokud máte v celé aplikaci pouze jednu Managed Beanu, která má stovky nebo tisíce řádků, tak je něco špatně …
  - Best practice je, aby pro každou entitu existovala alespoň jedna Managed Bean. V žádném případě ale nevytvářejte Managed Bean jako potomka entity! Managed Bean je svázaná se stránkou a používá entity (jako atributy).
- Managed Bean nesmí obsahovat business logic, ta patří do servisní vrstvy. Ani práci s databází, ta patří do datové vrstvy (nebo servisní vrstvy, podle typu aplikace).
- Managed Bean nesmí obsahovat statické atributy (kromě konstant), protože by byly sdílené pro všechny uživatele, což by popíralo význam scope Managed Beany.

Managed Bean scope I.

- Managed Bean může mít jeden z těchto oborů platnosti (scope):
  - `@RequestScoped @Scope("request")`: Po zavolání HTTP requestu se vytvoří, po odeslání HTTP response se zruší. Použijete, když chcete poslat klientovi dynamická data na základě vstupních parametrů.
  - `@ViewScoped`: Je v paměti po dobu interakce se stejnou JSF stránkou v prohlížeči. Vytvoří se po HTTP requestu, zruší se po navigaci na jinou stránku. Použijete zejména ve spojení s AJAXem, tabulkami a atributy, jejichž stav potřebujete udržet při více requestech na stejnou stránku. Z action / listener metod se musí vracet `void` nebo `null`! Ve Springu je nutné implementovat vlastní view scope, Spring ho out-of-the-box nemá.
  - `@SessionScoped @Scope("session")`: Je v paměti po dobu platnosti session klienta.
  - `@ApplicationScoped Ve springu výchozí`: Je v paměti po celou dobu běhu aplikace.

Managed Bean scope II.

- Další (zřídka používaný) scope:
  - `@ConversationScoped`: K dispozici pouze při použití CDI nebo Spring Web Flow, používá se pro tvorbu wizardů.
- Další scope používané v minulosti:
  - `@FlashScoped`: Používá se při POST-Redirect-GET (PRG). K dispozici pouze při použití CDI.
  - `@NoneScoped`: Je vytvořena pro každé vykonání EL (Expression Language). Používá se pouze na beanách, které jsou součástí jiných s delším oborem platnosti.
  - `@CustomScoped`: Můžete si vytvořit vlastní scope

- Poznámka: V drtivé většině případů se používá request a view scope. Pokud často používáte session scope anebo application scope, pak máte špatný návrh managed bean.

Serializace

- Je best practice, aby všechny Managed Beany, které mají scope session, view a conversation, implementovaly rozhraní Serializable.

ItemManager

```
@ViewScoped
@ManagedBean
public class ItemManager {

   private Item item;

   @EJB private ItemService itemService;

   @PostConstruct
   public void init() {
      item = new Item();
   }
   // gettery, settery
   // dalsi metody
}
```

Obyčejně se používá místo konstruktoru. Uvnitř konstruktoru ještě nejsou nastaveny injectované atributy, uvnitř této metody nastaveny jsou.

Při vytváření instance `ItemManager` můžete vytvořit objekt, se kterým budete dál v JSF pracovat

Managed bean v modelu

- Managed beany, které mají jako atributy entity, musí nějak do těchto entit dostat data z databáze / do databáze. Jak na to? V žádném případě tento kód nedávejte do getteru / setteru, protože je komponenty mohou volat vícekrát a tím pádem byste degradovali výkon Vaší aplikace.
- Je best practice nemít v getterech / setterech žádnou logiku, pouze získat / uložit data do atributu a mít metody, které s těmito atributy pracují (loadOne(), loadAll(), save(), delete() …).
- Načtení záznamů se provádí na jednom z těchto míst:
  - Metoda s anotací `@PostConstruct`
  - V JSF pomocí `<f:event type="preRenderView" ...`
  - …

@Inject I.

- Je možné mít z jedné beany referenci na druhou pomocí anotace `@Inject` (dříve: `@ManagedProperty`)
- Například:

```
@ViewScoped @Named
public class ItemManager {

   private Item item;

   private int quantity;

   @Inject
   @Named("#{basket}")
   private Basket basket;

   public void addItemToBasket() {
      basket.add(item, quantity);
   }
}
```

Poznámka: @Named je nepovinné

```
@SessionScoped
@Named
public class Basket {

}
```

Poznámka: Toto je typické použití. (ve ViewScoped bean přistupujeme k SessionScoped bean). Obráceně to nelze.

@Inject II.

- Tato funkcionalita je velice užitečná, ale zase na druhou stranu bych se moc „nerozlítnul“ ve vytváření takových atributů, zejména pokud by jedna managed bean obsahovala referenci na další (která by byla v session scope) a ta na další (opět v session scope), …

Komponentový strom I.

- Všechny komponenty, které máte na stránce se musí pro každého uživatele na serveru vytvořit (JSF je stateful).
  - Když u nějaké komponenty nastavíte atribut `rendered` na `false`, pak se tato komponenta klientovi nepošle, ale na serveru stejně zabírá místo (v operační paměti).
  - Proto čím větší komponentový strom, tím bude aplikace pomalejší.
    - Pokud stejnou službu udělá HTML tag i JSF tag (například h:panelGroup vygeneruje tag span), z pohledu serveru je efektivnější použít HTML tag.
    - Pokud nepotřebujete pro konkrétní vlastnost AJAX, tak ho nepoužívejte.
    - Když bude aplikace rozdělená do více XHTML souborů, pak bude komponentový strom pro každý soubor menší.

Komponentový strom II.

- Optimalizační tip: Pokud se můžete rozhodnout, jestli použijete HTML nebo JSF komponentu, použijte HTML, je to výkonnější.