Java EE

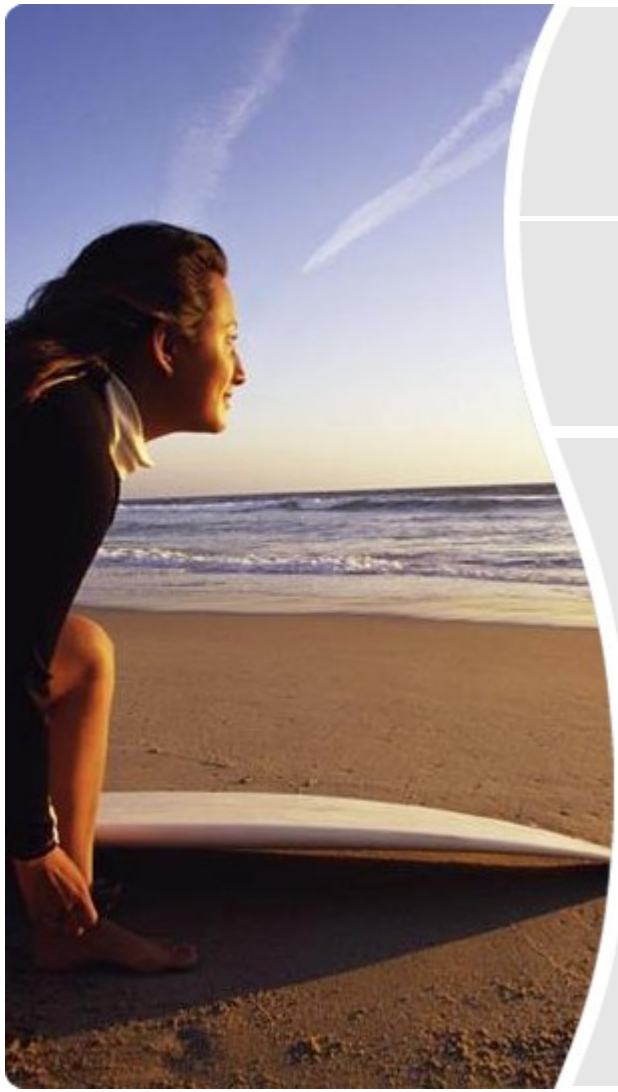
I. rész: Bevezetés

OSUM rendezvény

Nagy Zoltán Arnold

nagyz@nefty.hu

Miről lesz ma szó?



Mitől lesz egy alkalmazás “nagyvállalati”?

Mi az a Java EE?

Mikor használjuk, és mikor ne?

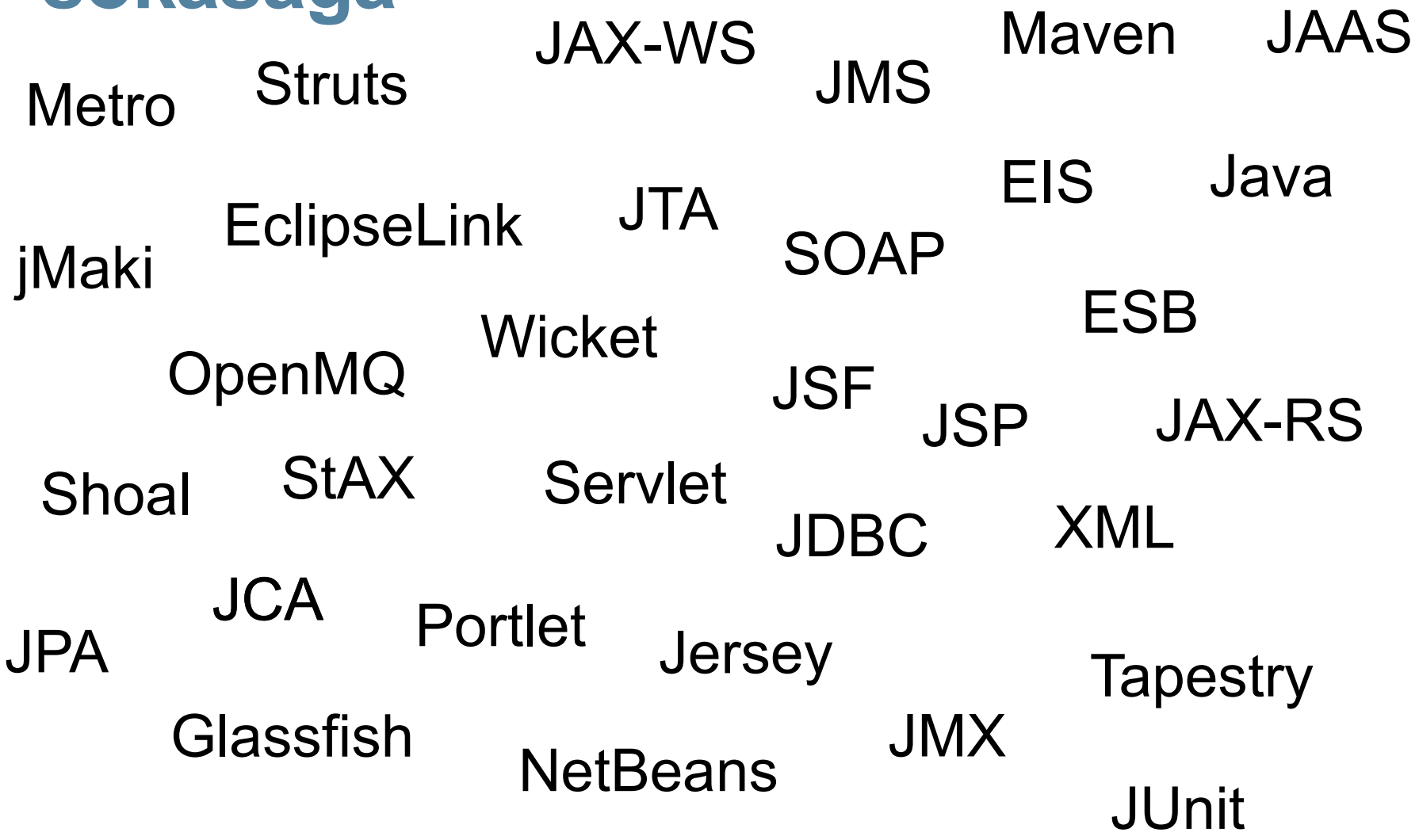
Egy alkalmazás felépítése, azaz mit nyújt a platform

Kérdések és válaszok

Sunos történések

- Sun Campus Ambassador program
- Open Source University Meetup (OSUM)
  - http://osum.sun.com/group/elte

A technológiák és eszközök sokasága

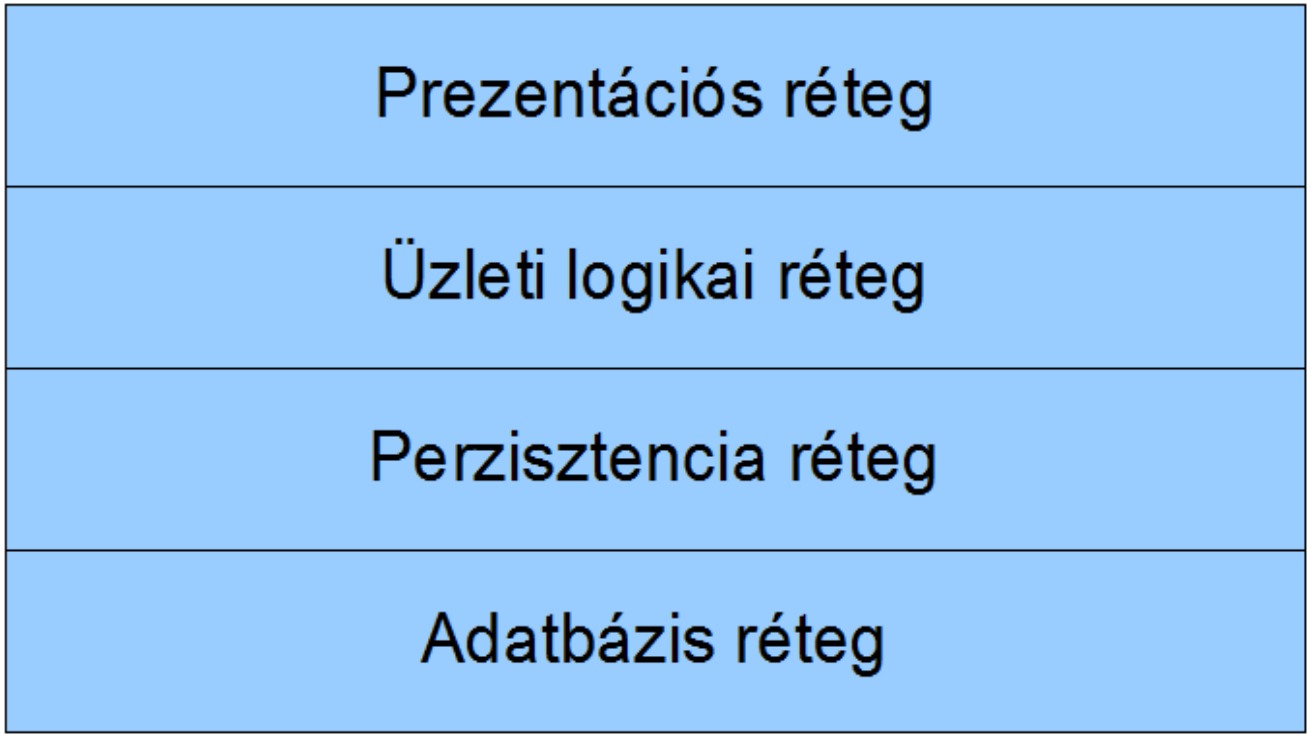


Nagyvállalati alkalmazásfejlesztés

- Milyen tulajdonságokkal kell rendelkeznie az alkalmazásnak?
  - Nagy rendelkezésreállású → hibatűrő
  - Nagy teljesítményű → elosztott
  - Tranzakciós
  - Biztonságos
  - Moduláris → komponens alapú fejlesztés
  - Validálható (helyes bemenetre helyes kimenet)
  - Újrafelhasználható

- Mindehez normális architectúra kell...

Architectúra kialakítása



A rétegek feladatai

- Prezentációs réteg: a felhasználói interfész megvalósítása, gyakorlatilag webes elérhetőség biztosítása
- Üzleti logikai réteg: az alkalmazásspecifikus kód helye, a rendszerből elérhető bárhonnan
- Perzisztencia réteg: az állapottér leképzése relációs adatokra és vissza
- Adatbázisréteg: általában egy relációs adatbáziskelő, például MySQL

Nagy rendelkezésreállás

- Minden rendszerkomponenst legalább duplázunk, amin átmegy adat; ha az egyik kiesik, a másik azonnal átveszi a szerepét → fontos az adatok replikálása a komponensek között
  - aktív-aktív failover cluster
- Nem csak az uptime számít, SLA lehet egy szolgáltatás válaszidejére is!
- 99.99% (négy kilenc): 4.32 perc/hó, évi 52.6 perc
- 99.999% (öt kilenc): 25.2 mp/hó, évi 5.26 perc

Skálázhatóság

- Vertikális skálázás (scale up): minél nagyobb gép (CPU, memória, diszkek...)
- Horizontális skálázás (scale out): minél több gép (fontos a gyors egymás közötti kommunikáció biztosítása)

- Mindkettőhöz normális tervezés szükséges: a rosszúl megírt kód nem skálázódik

Tranzakciókezelés

- Nem lehet inkonzisztencia a rendszerben
- A tranzakció egy logikai egység (unit of work)
  - egymástól függő, nem elválasztható részek
  - vagy sikeres a tranzakció: commit
  - vagy nem sikeres: rollback
- Az egyidejű hozzáférést valahogy figyelni kell
  - Optimistic locking
  - Pessimistic locking
- Több rendszernél two phase commit

Biztonság

- Adott erőforrás felhasználójának ellenőrzése
- Mindent lehessen szabályozni
- AAA:
  - Authentication: az adott személy beléphet-e a rendszerbe?
  - Authorization: az adott felhasználónak (bejelentkezés után!) van-e joga egy adott szolgáltatáshoz?
  - Accounting: szolgáltatás felhasználásnak naplózása (kezdet, vég)... → számlázási, biztonsági célok

Modularitás

- Konfiguráció változatása esetén nem szükséges az egész rendszert újraindítani: elég az adott komponenst
  - > a cél: minimalizálni a kiesést, gyorsítani a fejlesztést
- Ha egy komponens nem teljesít elég jól, könnyen lehessen cserélni egy másikra

Validálható

- Minden komponens a többitől függetlenül tesztelhető
- Unit Testing

- Test Driven Development – TDD
  - > nem a kész szoftvert teszteljük, hanem a teszteknek konform szoftvert fejlesztünk

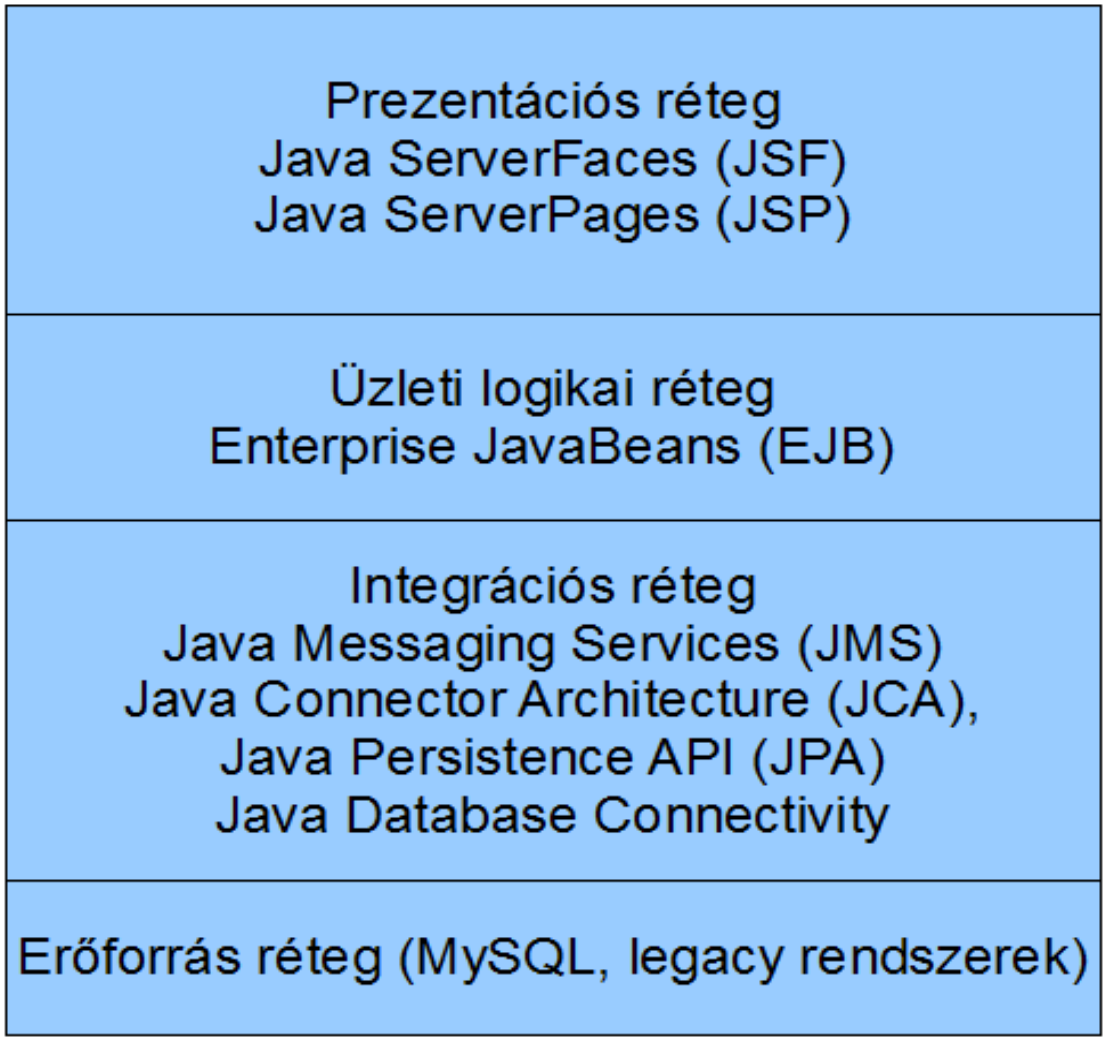
Mi az a Java EE?

- Java Enterprise Edition
- Egységes platform szerveroldali fejlesztésekhez
- Nem kell feltalálnunk a spanyolviaszt: a fenti szükségleteket a konténer segítségével akár transzparensen kielégíthetjük
- Java SE -re épít
- Egy hatalmas API gyüjtemény egységes integrációja
- Lehetővé teszi hibatűrő, elosztott, többrétegű alkalmazások fejlesztését moduláris elemekből
- Legacy rendszerek támogatása

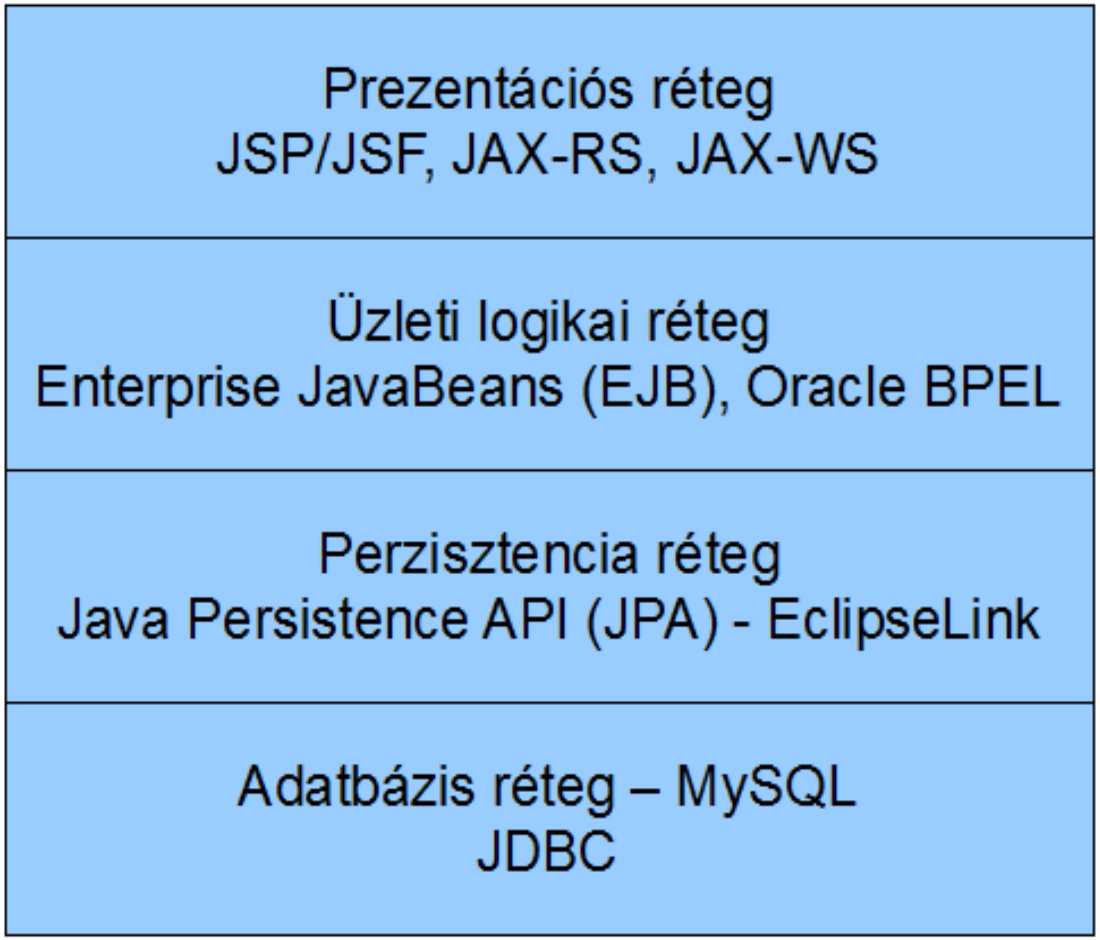
A konténer

- Szolgáltatásokat nyújt a benne futó alkalmazás(ok)nak
- Alkalmazásszerver része (GlassFish)
- Erre deployolódnak az alkalmazások egy instanceba
- Egy fizikai gépen több instance
- Egy instance akárhány fizikai gépből álhat
- Szolgáltatások JNDI fába regisztrálódnak, elérhetőek minden rétegből
- Az erőforrásokat (életciklus, elérhetőség, ...) majd a konténer menedzseli

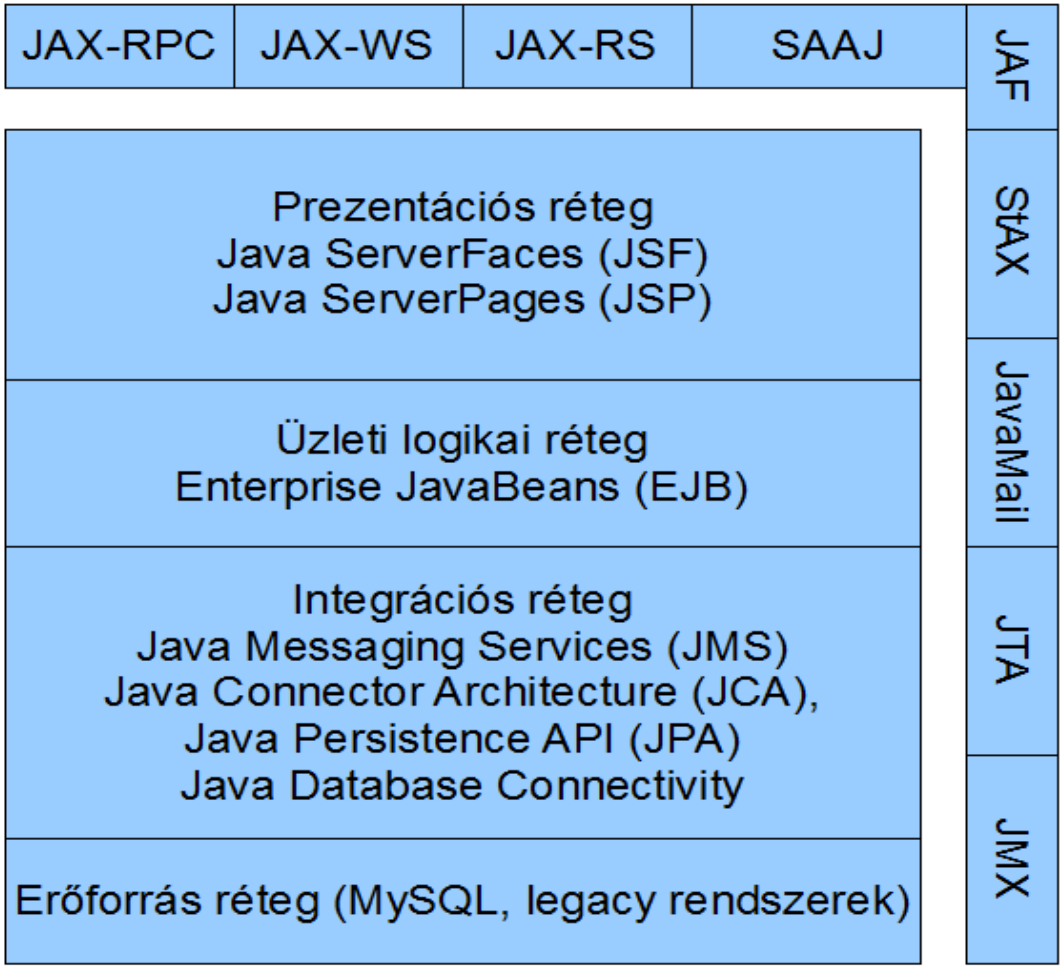
A rétegekhez tartozó Java EE API-k



A modell kicsit átalakítva...



Kiegészítve a platformszolgáltatásokkal



Alkalmazásfejlesztés alulról felfelé

- Bármilyen JDBC-t beszélő erőforrás használható perzisztens adatok tárolására
- Létre kell hoznunk egy connection poolt
  - > Finomhangolás: minimum-maximum méret, átméretezési mérték
  - > egyszerre sok kapcsolat van nyitva az erőforrás felé
  - > a kérések transzparens elosztása a kapcsolatok között, kívűlről a pool egyként viselkedik

Az építőelemek: entitások

- Egy speciális osztály, amely perzisztálható (JPA)
- Speciális annotációkkal a kódból szabályozható a működés
- Minden adat objektum
- Definiálhatóak kapcsolatok
  - egy-egy, egy-sok, sok-sok
- Öröklődés támogatása
- A kezeléshez kell egy EntityManager, illetve egy PersistenceContext; a konténer kezeli ezeket is
  - egy PersistenceContext egy adatbázis

```
@Entity
@Table(name="persons")
public class Person implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name="id")
   private Integer id;
   @Column(name="firstname")
   private String firstname;
   @Column(name="lastname")
   private String lastname;
   @Version
   private Integer version; ...
}
```

Az üzleti logikai réteg: EJB

- Enterprise JavaBeans (EJB): szerver oldali komponens
- Szolgáltatások megvalósítására alkalmas
- Az összes üzleti logika itt helyezhető el
- Három típus
  - stateless: ha nincs szükség a hívások között információcserére (conversation state)
  - stateful: ha van. például egy regisztrációs folyamat, vásárlás, rendelési folyamat

    → a már bevitt adatokat jó tárolni

Az üzleti logikai réteg: EJB (folyt)

- message driven bean: valamilyen üzenetre reagál aszinkron módon, a forrás általában egy JMS üzenet
- Szintén pooled service: itt is sok példány dolgozik a háttérben
  - egyenletes terhelésnél csökkenthetőek az erőforrásköltségek
- Távolról is hívható, nem csak egy alkalmazáson belülről
  - dependency injectionnel könnyen elérhető bárhonnan

```
@Stateless
public class PersonManagerBean implements PersonManagerBeanLocal {
    @PersistenceContext
    private EntityManager em;
    public void createEntity(String firstname, String lastname) {
        Person person = new Person();
        person.setFirstname(firstname);
        person.setLastname(lastname);
        em.persist(person);
    }

    public String getFullname(Integer id) throws NoSuchPersonException {
        Person person = em.find(Person.class, id);
        if(person == null)
            throw new NoSuchPersonException();
        return person.getFirstname() + "" + person.getLastname();
    }
}
```

```
@MessageDriven(mappedName = "jndi/pool",
    ActivationConfig =  {
            @ActivationConfigProperty(propertyName =
"acknowledgeMode", propertyValue = "Auto-acknowledge"),
            @ActivationConfigProperty(propertyName =
"destinationType", propertyValue = "javax.jms.Queue")
    })
public class DistributorHandlerBean implements
MessageListener {
    public DistributorHandlerBean() {
    }
    public void onMessage(Message message) {
    }
}
```

Integrációs réteg: JMS

- Java Messaging Service (JMS)
- Üzenetküldési- és fogadási API
- Message Oriented Middleware (MOM)
- Laza csatolású rendszerek létrehozása
- Flexibilis, megbízható, skálázható
- Pont-Pont kapcsolat, Publish-Subscribe modell üzenetszóráshoz
- Nyílt megvalósítás: OpenMQ

Integrációs réteg: JCA

- Java EE Connection Architecture (JCA)
- Egy sztenderd API definiálása több Enterprise Information Systemmel (EIS) való integrációhoz
  - > SAP, PeopleSoft, ...
- JCA nélkül minden alkalmazásszerverhez eltérő csatlakozófelületet kellett fejleszteni
  - > (Ndb alkalamzásszerver) x (Mdb EIS)
- JCA támogatás mellett elég az API -t megvalósítani
  - > 1 db implentáció minden EIShez

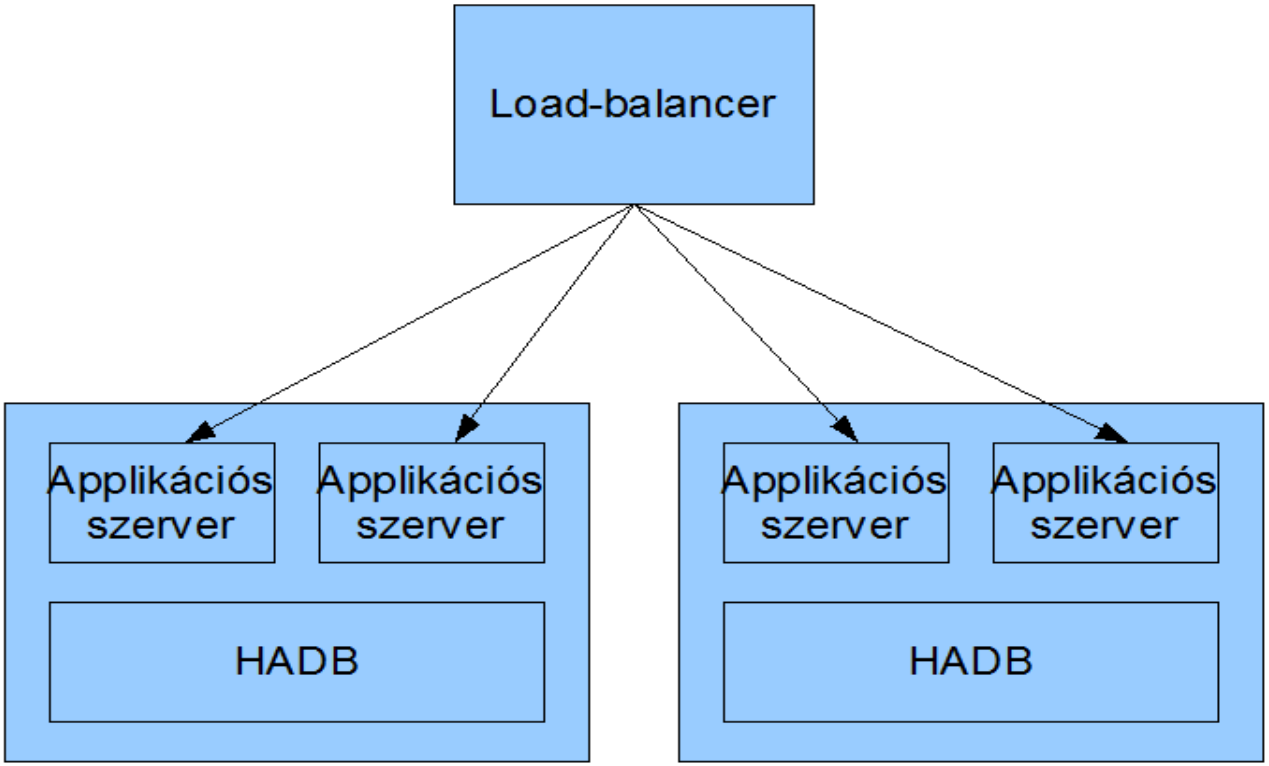
Prezentációs réteg megvalósítása

- Java ServerFaces: komponens alapú fejlesztés
- Egy bean adott példányát könnyű használni
- Általában csak az üzleti logikai réteggel kell kapcsolatban lennie, hiszen adatok be- és kivitelére szolgál
- Servletek: Java kód kezeli a HTTP kérést, alacsony szintű oldalgenerálás

SOA: Webszervizek megvalósítása

- Java Architecture for XML Binding (JAXB): xml schemából oszálygenerálás, és vica-versa
- Java API for XML-Based Web Services (JAX-WS): xmlek segítségével kommunikálnak szolgáltatások
  - SOAP üzenetek
  - lehetőség van digitális aláírásra, titkosításra, megbízhatósági garanciák betartására
- Java API for RESTful Web Services (JAX-RS):
  - A HTTP GET/PUT metódusai a setterek/gettereket hívják meg
  - Leightweight, kevésbé erőforrásigényes

Terheléseloszlás és nagy rendelkezésreállás megvalósítása



Nem elég szétosztani...

- Ha a rendszer egy komponense nem tud skálázódni, akkor az egész rendszer nem tud skálázódni
- Replikálni is kell
  - > stateful replication nagyon költséges, azonban teljesen transzparens
- round-rubin terheléselosztás nem biztos, hogy elegendő: egy nagy kérés megfoghat egy gépet, arra már hiába küldenénk több kérést → SLA
- Java Management eXtension (JMX) – az instanceok monitorozása, az alapján döntés
  - > Erre született a GlassFish plugin a Sun webszerverhez

Tranzakciókezelés

- Java Transaction API – JTA
- Támogatnia kell az integrációs rétegnek is, amivel épp dolgozunk
- Az üzleti rétegben lehet CMT vagy BMT
  - > Container Managed Transactions
  - > Bean Manager Transactions
- Alapból minden EJB-metódus tranzakciós
  - > felüldefiniálható ahogy szükséges
- A MOM küldése-fogadása is tranzakciós

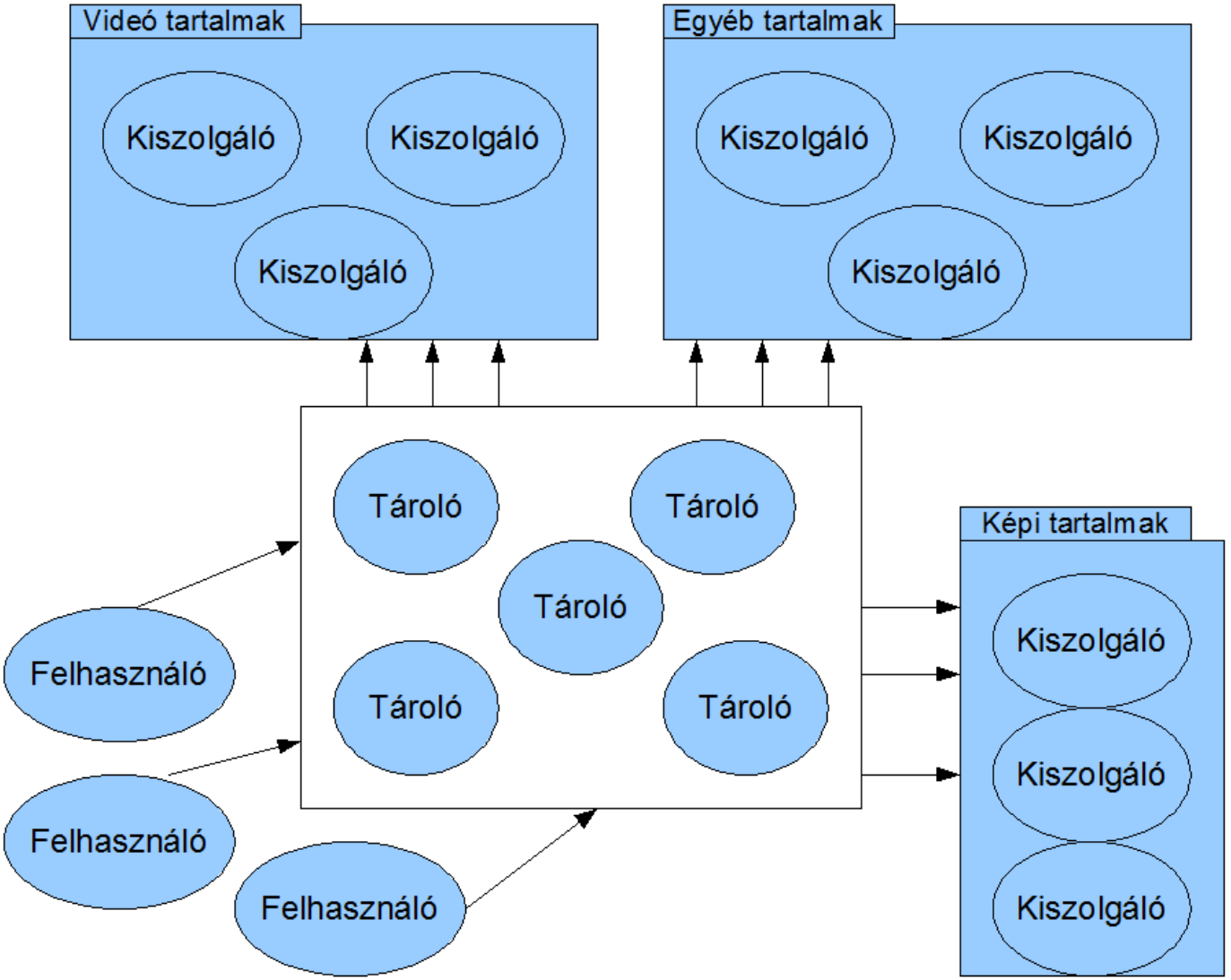
Biztonságos

- Java Authentication and Authorization Service (JAAS)
- Komponens alapú: minden része felüldefiniálható/cserélhető
  - > Jelszavas, smartcard, kerberos, biometrikus, … azonosítások támogatása
- Szerepkörök (role) definiálása
  - > Bármilyen JDBC-képes erőforrás használható a tárolásra/lekérésre
- Az alkalmazásban bárhol definiálható hozzáférésszabályozás

CDN: követelmények

- Sok tartalom kezelése
- Sok felhasználó egyidejű kezelése (több ezer kérés/ másodperc)
- Lineáris skálázhatóság
- Biztonságos tárolás
- Felhasználói oldalról transzparencia

CDN: lehetséges megvalósítás



CDN funkcionalitáslista

- Tetszőleges felhasználó feltölthet
- A feltöltött tartalmak szétszórása
  - > lehet automatikus
  - > lehet periódikusan
  - > lehet adminisztrátori jóváhagyás után
- A felhasználói interfésznek egyszerűnek kell lennie

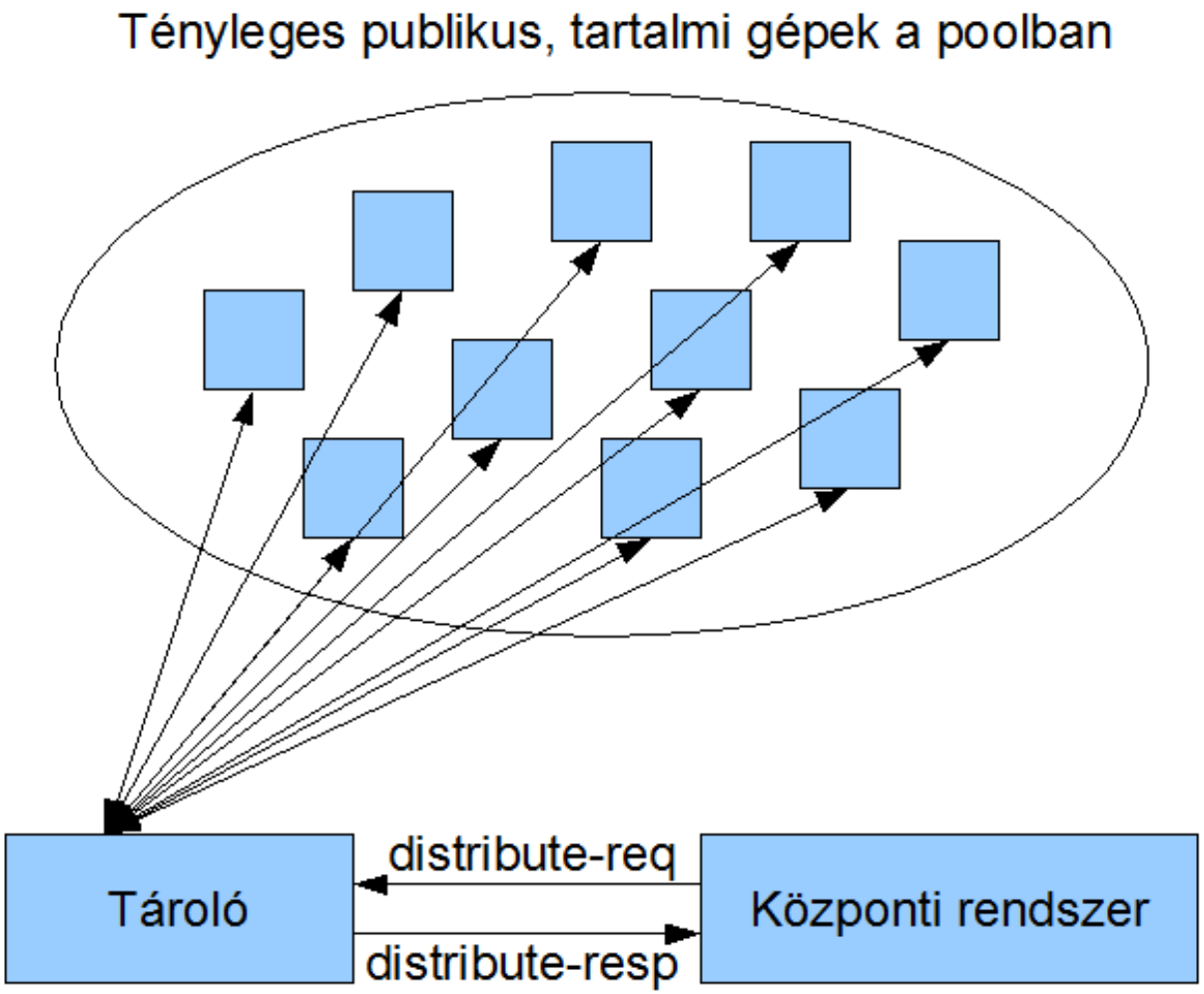
CDN: a funkciók megvalósítása 1. feltöltés

- A felhasználó meghív egy webszervízt egy load-balancolt instanceon a háttérben (pl weblap, tetszőleges kliensalkalmazás, ...)
  - > Visszakapja a feltöltéshez szükséges adatokat
  - > feltölti a tartalmát, majd jelzi ezt a központi gépnek
  - > kikerül egy “import-request” üzenet a queuere, amit az adott tároló feldolgoz
  - > válaszként megkapja a fájladatokat a központi rendszer egy “import-response” üzenetben
  - > kikerül egy “finalize-import-request” üzenet
  - > A kliens kirakja egy publikusan is elérhető helyre (még nem osztja szét)

CDN: a funkciók megvalósítása 2. tartalomszétosztás

- A központi rendszer indítja a szétosztást
  - > distribute-request (url, size, hash)
- Az adott tartalmat tároló gép kiküld egy broadcast üzenetet (Pub/Sub modell!) annak a csoportnak, akinek tárolni kell a tartalmat
  - > get-request (url, size, hash)
- A poolból lévő gépek egyenként visszaküldik a választ a tartalmi gépnek, mely jelez a központi rendszernek
  - > get-response → distribute-response

CDN: a funkciók megvalósítása
2. tartalomszétosztás



Támpontok a tanuláshoz

- A hivatalos Java EE tutorial
  - > http://java.sun.com/javaee/5/docs/tutorial/doc/
- JavaFórum projekt
  - > www.javaforum.hu
- Debu Panda, Reza Rahman, Derek Lane: EJB 3 in Action (Manning, 2007)
- Paul R. Allen, Joseph J. Bambara: Sun Certified Enterprise Architect for Java EE Study Guide (McGraw-Hill, 2007)

Kérdések és válaszok

Köszönöm a figyelmet!

Nagy Zoltán Arnold
OSUM – ELTE / Sun Microsystems
www.architectnotes.com
nagyz@nefty.hu