Pokročilé informační systémy

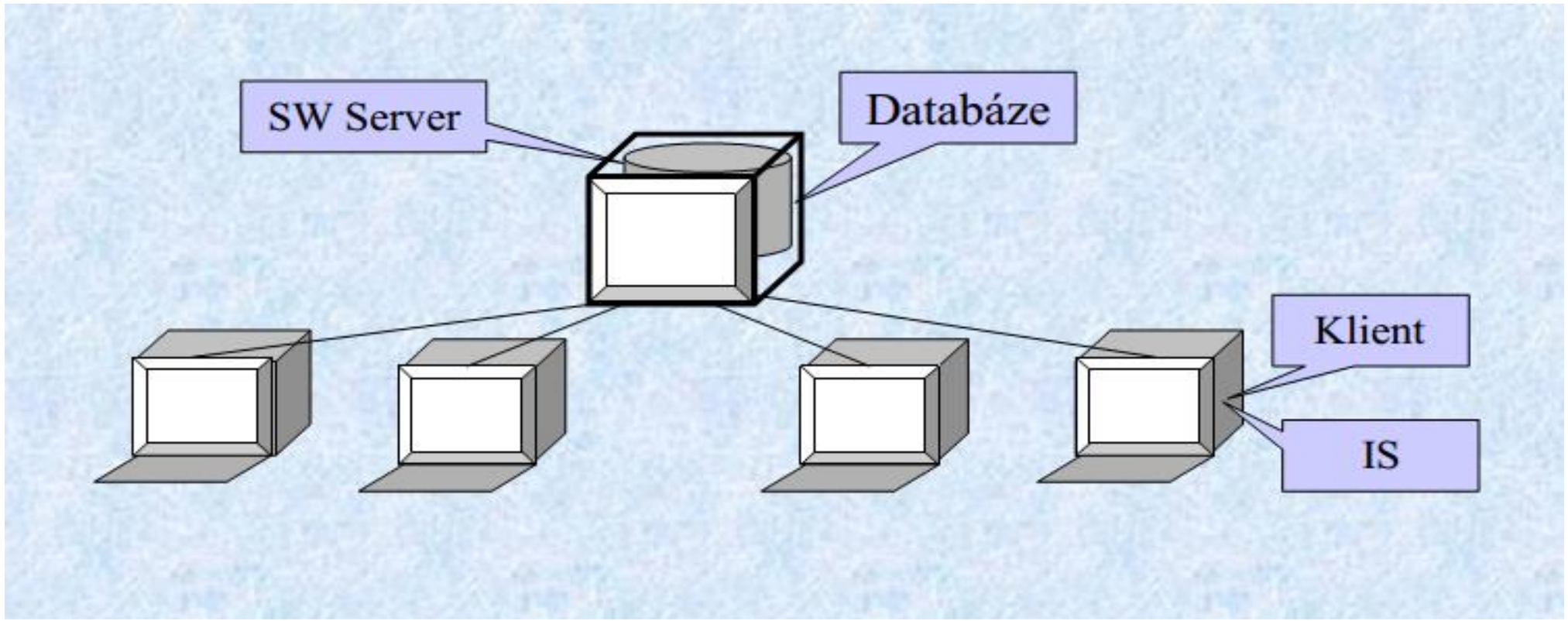
Backend a platforma Jakarta EE

Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Architektury a implementace informačních systémů

Architektura klient-server (dvouvrstvá)

- Užity dva druhy oddělených výpočetních systémů ***klient*** a ***server***.
- ***Tloušťka*** klienta odpovídá jeho "***inteligenci"***



Architektura klient-server

- Na nižší úrovni použita síťová komunikace standardizovaná protokoly Internetu TCP/IP
- Chování klienta a serveru rovněž standardizováno
  - Server specializovaný pro databázové dotazy
  - Po síti se přenášejí pouze dotazy a výsledky
- Ve vyšších vrstvách aplikačních protokolů se nejčastěji komunikuje ***serializovanými daty***, případně v SQL

V současnosti nejvíce užívaná architektura – oblast našeho zájmu.

Třívrstvá architektura

- ***Třívrstvá architektura (three-tier architecture)***
- ***Prezentační vrstva*** – ***vizualizuje*** informace pro uživatele, většinou formou grafického uživatelského rozhraní, může kontrolovat zadávané vstupy, neobsahuje však zpracování dat
- ***Aplikační vrstva*** – jádro aplikace, logika a funkce, výpočty a zpracování dat
- ***Datová vrstva*** – nejčastěji databáze. Může zde být ale také (síťový) souborový systém, webová služba nebo jiná aplikace.

Terminologická odbočka

- **Tier** – fyzická vrstva – jednotka nasazení (deployment)
  - Fyzické členění systému – klient, aplikační server, DB server
  - Tomu odpovídá volba technologií pro realizaci jednotlivých částí
- **Layer** – logická vrstva – jednotka organizace kódu
  - Obvykle řešena v rámci aplikační vrstvy
  - *Data layer* – část řešící komunikaci s databází
  - *Business layer* – část implementující logiku aplikace
  - *Presentation layer* – komunikace s klientem

Schéma třívrstvé architektury

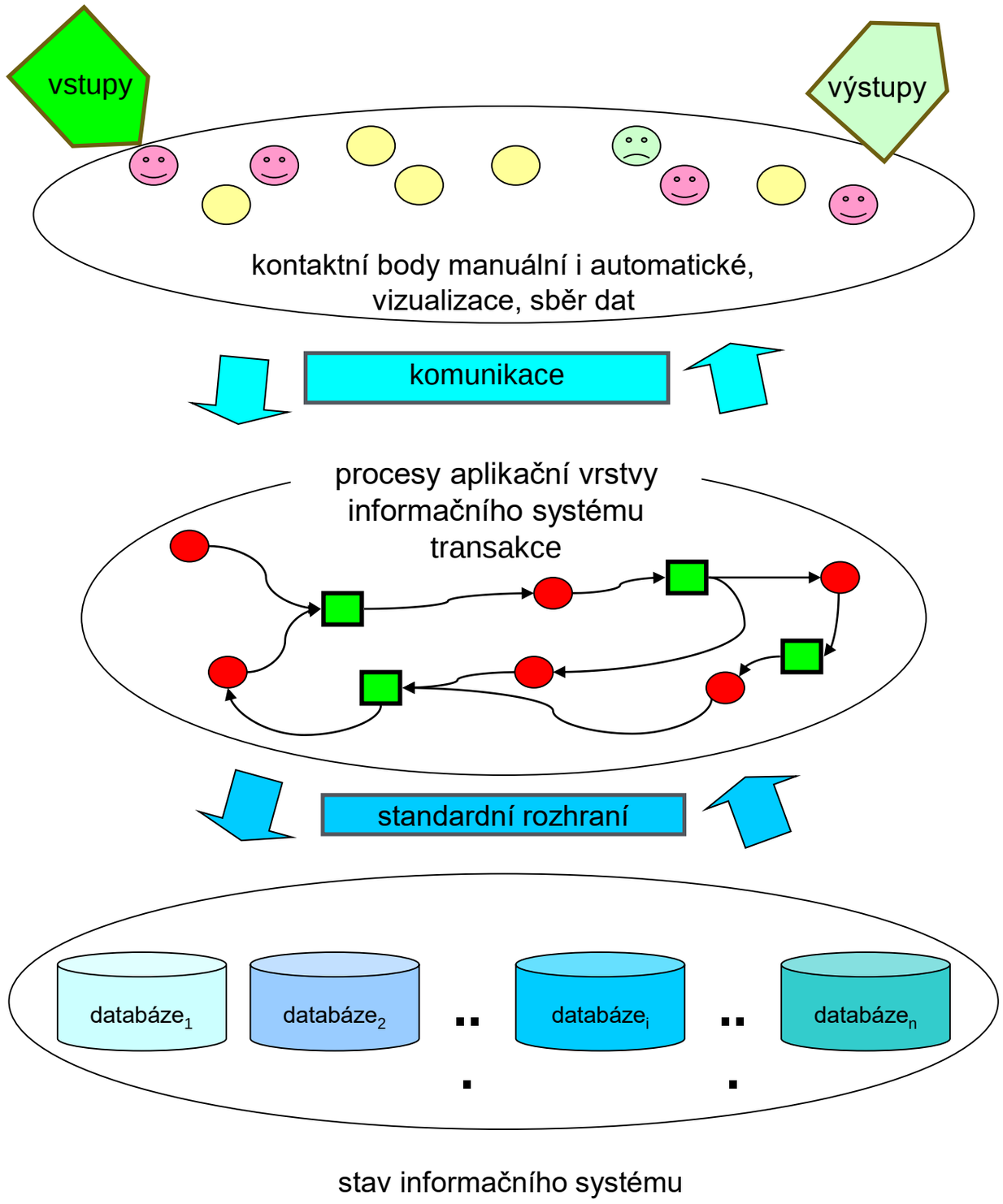
prezentační vrstva, *vizualizace*

komunikace s kontaktními body, síť, přenos - *serializace*

aplikační vrstva – business logic, transakce, dotazy, procesy

komunikace s databází, serverové skripty - PHP, ADL, SQL

datová vrstva (relační, objektová), případně distribuovaná



Webový IS

Klientská část (frontend)

Webový prohlížeč

HTTP

- HTML
- Jednoduchá data (formuláře)

Serverová část (backend)

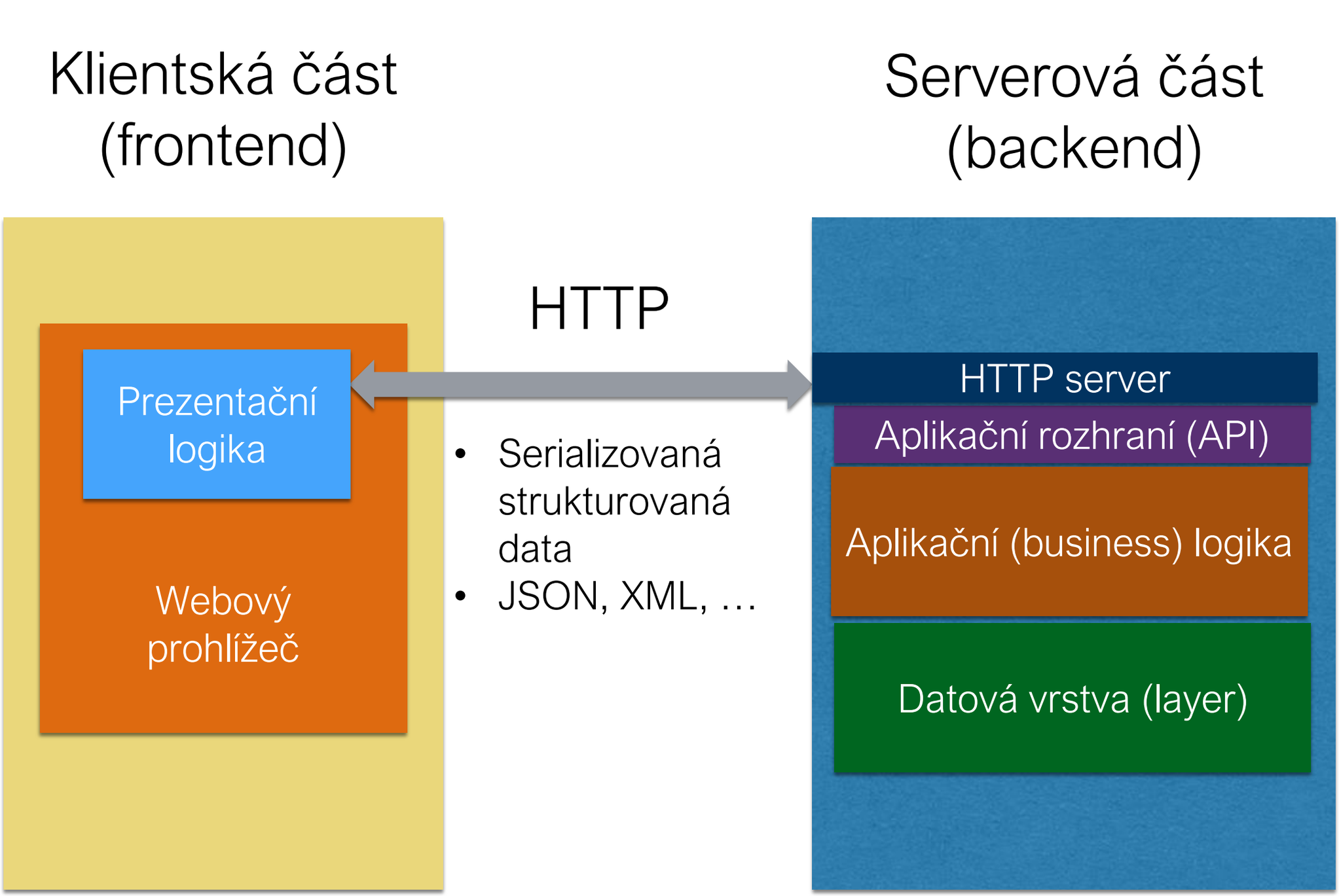
HTTP server

Prezentační logika

Aplikační (business) logika

Datová vrstva (layer)

Webový IS s aplikačním rozhraním



Technologie třívrstvé architektury

- ***Klientská část***
  - HTML, CSS, JavaScript (+ Angular, React, Vue.js, …)
- ***Komunikace***
  - **HTTP**, serializace
    - Aplikační rozhraní – REST, JAX-RS, SOAP, GraphQL
- ***Serverová část*** *– jednotlivé vrstvy*
  - Java, .NET, PHP, JavaScript, Python, Ruby, …
  - Různá rámcová řešení (frameworky)

Technologie třívrstvé architektury

- ***Komunikace mezi aplikační a datovou vrstvou***
  - *S*tandardizované databázové rozhraní (SQL)

- ***Datová vrstva*** - relační nebo objektový databázový model
  - *Relační model* viz kurz Databázové systémy
  - *Objektový model* v kurzu Pokročilé informační systémy

Aplikační vrstva – Java

- Java EE umožňuje implementovat *monolitický* IS s *třívrstvou architekturou:*

1. Databázová vrstva
    - JPA – definice entit, persistence (*PersistenceManager*)
    - Alternativně: Relační databáze (JDBC), NoSQL (MongoDB), …
2. Logická (business) vrstva
    - Enterprise Java Beans (EJB) nebo CDI beans
    - Dependency injection – volné propojení
3. Prezentační vrstva
    - Webové rozhraní (JSF) nebo API (REST, JAX-RS)

Další platformy – přehled

- Java
  - Existuje mnoho možností kromě „standardní“ J EE
- .NET (Core / Framework)
  - Mnoho řešení na všech vrstvách
- PHP
  - Různé frameworky, důraz na webovou vrstvu
- JavaScript
  - Node.js + frameworky, důraz na web a mikroslužby
- Python, Ruby, … - podobné principy

Distribuované architektury

- Monolitický systém (typické pro třívrstvou architekturu)
  - Vyvíjí se a nasazuje jako jeden celek
  - + snáze zvládnutelný vývoj, testování
  - - obtížnější a pomalejší nasazování nových verzí
- Distribuované architektury
  - Service-oriented architecture (SOA)
  - Mikroservices (mikroslužby)

Jakarta EE

Serverová část informačního systému

Java

- Programovací jazyk
  - Silně typovaný, objektově orientovaný
- Platforma pro vývoj a provoz aplikací
  - Virtuální stroj
  - Podpůrné nástroje
    - (překladač, debugger, dokumentace, …)
- Balík Java SE (JRE nebo **JDK**)
  - Alternativní implementace (i open source)

Java EE

- V současnosti Jakarta EE 8 (září 2019)
  - Plně kompatibilní s Java EE 8
- Platforma pro vývoj podnikových aplikací a IS v Javě
- Množina standardních technologií a API
  - Enterprise Java Beans (EJB)
  - Tansaction API (JTA)
  - Java Persistence API (JPA)
  - Java Message Service (JMS)
  - Java Server Faces (JSF)
  - … a další

Vrstvy aplikace Java EE

1. Databázová vrstva
2. Business vrstva
   - Implementace chování aplikace
   - Potenciálně distribuovaná
3. Webová vrstva
   - Webové API nebo komponentový serverový framework

Nejdůležitější součásti specifikace

- **Datová vrstva**
  - Java Persistence API (JPA)
- **Business vrstva**
  - Java Transactions API (JTA)
  - Enterprise Java Beans (EJB)
  - Java Messaging Service (JMS)
- **Webová vrstva**
  - Java Server Faces (JSF), Servlet, WebSocket
  - Java API for RESTful Web Services (JAX-RS)
  - Java API for XML Web Services (JAX-WS) – SOAP, …

Struktura aplikace Java EE

- EJB moduly
  - Definují veřejná rozhraní
  - Implementují chování
  - Lze je odděleně nasadit na servery
    - EJB vrstva zajišťuje distribuované volání
    - Dependency injection, kontrola souběžnosti, přístupu, …
- Webové moduly
  - Webové rozhraní

Běhové prostředí – Kontejnery

- Prostředí pro běh aplikace na serveru
- EJB kontejner
  - Běh EJB modulů, volání funkcí
- Webový kontejner
  - Běh webové vrstvy
- Java EE kontejner
  - Webový + EJB kontejner

Dostupné kontejnery

- Java EE kontejnery (aplikační servery)
  - GlassFish (Oracle, open source)
    - **Payara** (https://www.payara.fish/)
  - WildFly (Red Hat, dříve JBoss AS)
  - TomEE (Apache)
  - WebSphere, **Open Liberty** (IBM)
- Pouze webové servery
  - Tomcat (Apache)
  - Jetty (Mort Bay Consulting)

Instalace vývojového prostředí

Instalace

- Java 11+ – **musí být JDK**
- Java EE server
  - **Payara** (Glassfish)
  - TomEE
  - WildFly, Open Liberty
- Vývojové prostředí
  - **Eclipse for Java EE developers**
  - NetBeans, IntelliJ IDEA
- Databázový server
  - Jakýkoliv relační (MySQL)
  - JDBC ovladač
- Databázový konektor

Konfigurace

- Instalovat Eclipse
- Rozbalit Payara server
- Definovat server v IDE
  - Pro Glassfish nutné rozšíření Eclipse
- Instalovat databázový konektor
  - Např. do /lib serveru
  - Driver definitions v IDE
- Konfigurovat databázové spojení
  - V IDE i na serveru

Tvorba aplikace v Javě

Struktura aplikace

- Archivy s částmi aplikace:
  - MojeApp.war – webová aplikace
  - MojeKomponenta.jar – EJB komponenta
  - MojeKomponenta.ear – enterprise aplikace

- Archiv *.war obsahuje
  - Přeložené třídy
  - Knihovny (thin vs. thick war)
  - Webové soubory

Struktura aplikace (II)

- Kořenový adresář je / webu
- META-INF
  - Informace o archivu
- WEB-INF/lib
  - Potřebné knihovny (*.jar)
- WEB-INF/class
  - Přeložené třídy (*.class)

Konfigurace aplikace

- Deskriptor WEB-INF/web.xml
  - Jméno aplikace
  - Výchozí stránka
  - Mapování servletů
  - Definice filtrů
  - …

Vytvoření projektu

- New -> Dynamic Web Project
  - Cílové prostředí (Glassfish)
  - Podpora JavaServer Faces 2.x
  - Podpora JPA
- Konfigurace později
  - Project -> Properties
    - Project Facets
    - Přidat JAX-RS, apod.

Alternativna: Maven

- Maven – nástroj pro správu projektu
  - Získání závislostí, sestavení, …
- Konfigurace v souboru `pom.xml`
  - Parametry projektu, moduly
  - Závislosti
- Lze vyjít z připravené šablony
  - https://github.com/DIFS-Teaching/jsf-basic
- Přeložení a vytvoření distribučního archivu
  - `mvn clean package`

Java Bean

```
public class Person
{
    private long id;
    private String name;
    private String surname;
    private Date born;

    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
  ….
}
```

Persistence

- Pomocí anotací vytvoříme z třídy **entitu** persistence

```
@Entity
@Table(name = "person")
public class Person
{
    @Id
    private long id;
    private String name;
    private String surname;
    private Date born;
```

Generované ID

```
@Entity
@Table(name = "person")
public class Person
{
    @Id
    @GeneratedValue(strategy = IDENTITY)
    private long id;
    private String name;
    private String surname;
    private Date born;
...
```

Vztahy mezi entitami

- Anotace @OneToMany a @ManyToOne
- Nastavení mapování
  - V Eclipse pohled JPA
- Kolekce v Javě:
  - Collection<?>
    - List<?> (Vector, ArrayList, …)
    - Set<?> (HashSet, …)
    - Map<?, ?> (HashMap, …)s

Konfigurace presistence

- Soubor `persistence.xml`
  - Jméno jednotky persistence
  - Odkaz na data source na serveru
  - Případně další parametry pro mapování
    - Např. řízení automatického generování schématu

Implementace business operací

- Enterprise Java Beans (EJB)
  - Zapouzdřují business logiku aplikace
  - Poskytují business operace – definované rozhraní (metody)
  - EJB kontejner zajišťuje další služby
    - Dependency injection
    - Transakční zpracování
      - Metoda obvykle tvoří transakci, není-li nastaveno jinak

Vytvoření EJB

- Instance vytváří a spravuje EJB kontejner
- Vytvoření pomocí anotace třídy
  - `@Stateless` – bezstavový bean
    - Efektivnější správa – pool objektů přidělovaných klientům
  - `@Stateful` – udržuje se stav
    - Jedna instance na klienta
  - `@Singleton`
    - Jedna instance na celou aplikaci

Použití EJB

- Lokální
  - Anotace `@EJB` – kontejner dodá instanci EJB
- Vzdálené volání – dané rozhraní
  - Rozhraní definované pomocí @Remote

Propojení s JPA

- EJB implementují business operace
  - Často stačí stateless bean
- JPA rozhraní je reprezentováno objektem EntityManager
  - Dodá kontejner pomocí DI

Uložení objektu

```
@PersistenceContext
EntityManager em;


Person person = new Person();
person.setName(„karel“);
em.persist(person);
```

Změna objektu

```
@PersistenceContext
   EntityManager em;


   person.setName(“Karel”);
   em.merge(person);
```

Smazání objektu

```
@PersistenceContext
   EntityManager em;



em.remove(person);
```

Dotazování

- Query q = em.createQuery(“…”);
  q.setParameter(name, value);|
  q.setFirstResult(100);
  q.setMaxResuls(50);
  q.getResultList();

JPQL dotazy

- SELECT p FROM Person p
  WHERE p.name = “John”
- SELECT c FROM Car c
  WHERE c.reg LIKE :pref
- SELECT
  NEW myObject(c.type, count(c))
  FROM Car c
  GROUP BY c.type

Contexts and Dependency Injection (CDI)

- Obecný mechanismus pro DI mimo EJB
- Omezuje závislosti mezi třídami přímo v kódu
  - Flexibilita (výměna implementace), lepší testování, …
- Injektovatelné objekty
  - Třídy, které nejsou EJB
  - Různé vlastnosti pomocí anotací
- Použití objektu
  - Anotace @Inject
  - CDI kontejner zajistí získáni a dodání instance

CDI – Injektovatelné objekty

- Téměř jakákoliv Javovská třída
- Scope
  - @Dependent – vzniká pro konkrétní případ, zaniká s vlastníkem (default)
  - @RequestScoped – trvá po dobu HTTP požadavku
  - @SessionScoped – trvá po dobu HTTP session
  - @ApplicationScoped – jedna instance pro aplikaci
  - *Pozor na shodu jmen se staršími anotacemi JSF*
- Pokud má být přístupný z GUI (pomocí EL)
  - Anotace @Named

CDI – Dodání instancí

- Anotace @Inject
- Vlastnost (field)

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    @Inject
    private NumberGenerator numberGenerator;

}
```

CDI – Dodání instancí

- Konstruktor

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public ItemServlet(NumberGenerator numberGenerator) {
        this.numberGenerator = numberGenerator;
    }
    ...
}
```

CDI – Dodání instancí

- Setter

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public void setNumberGenerator(NumberGenerator numberGenerator)
    {
        this.numberGenerator = numberGenerator;
    }

    ...
}
```

Webové API – REST

REST

- Representational state transfer
- Jednoduchá metoda vzdálené manipulace s daty
- Reprezentuje CRUD (Create-Retrieve-Update-Delete) operace
  - Ale ve skutečnosti přistupujeme k **business vrstvě, ne přímo k datům**!
- Úzká vazba na HTTP
- Nedefinuje formát přenosu dat, obvykle JSON nebo XML

Využití HTTP

- Každá položka dat (nebo kolekce) má vlastní URI. Např.:
  - http://noviny.cz/clanky (kolekce)
  - http://noviny.cz/clanky/domaci/12 (jeden článek)
- Jednotlivé metody HTTP implementují příslušné operace s daty

Metody HTTP

- GET
  - Kolekce: získání seznamu položek
  - Entita: čtení entity (read)
- POST
  - Kolekce: přidání prvku do kolekce (create)
- PUT
  - Kolekce: nahrazení celého obsahu kolekce
  - Entita: zápis konkrétní entity (update)
- DELETE
  - Kolekce: smazání celé kolekce
  - Entita: smazání entity

Formát přenosu dat

- Není specifikován, záleží na službě
  - Obvykle JSON nebo XML (schéma záleží na aplikaci)
- Často více formátu k dispozici
  - Např.
    http://noviny.cz/clanky.xml
    http://noviny.cz/clanky.json
  - Využití MIME pro rozlišení typu, HTTP content negotiation

REST a Jakarta EE

- JAX-RS API součástí standardu
- Vytvoření služeb pomocí anotací
- Aplikační server zajistí funkci endpointu
  - Mapování URL a HTTP metod na Javovské objekty a metody
  - Serializace a deserializace JSON/XML na objekty
- Různé implementace
  - JAX-RS – Jersey (Glassfish), Apache Axis
  - Serializace – Jackson, gson, MOXy, …

REST v Javě

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String
                userName) {
        ...
     return someXmlString;
    }
}
```

Konfigurace

- JAX-RS servlet ve `web.xml`
  - Specifikace balíku s REST třídami nebo přímo výčet tříd
  - Specifikace JSON provider (Jackson)
- Alternativně
  - Třída odvozená od `javax.ws.rs.core.Application`
  - Konfigurace pomocí anotací

Klientská aplikace

- Zasílání REST požadavků
  - Jednotlivé JS frameworky mají vlastní infrastrukturu
- Prezentační logika
  - Navigace
  - Přechody mezi stránkami
  - Výpisy chyb, apod.

Autentizace v REST

- Protokol REST je definován jako ***bezstavový***
  - Požadavek musí obsahovat vše, žádné ukládání stavu na serveru
- To teoreticky vylučuje možnost použití sessions pro autentizaci
  - Technicky to ale možné je
  - Problém např. pro mobilní klienty
- Alternativy pro autentizaci:
  - HTTP Basic autentizace (nutné HTTPS)
  - Použití tokenu validovatelného na serveru – např. JWT
  - Složitější mechanismus, např. OAuth

HTTP Basic

- Standardní mechanismus HTTP, využívá speciální hlavičky
- Požadavek musí obsahovat hlavičku **Authorization**
  - Obsahuje jméno a heslo; nešifrované pouze kódované (base64)
    - **Je nutné použít HTTPS**
- Pro nesprávnou nebo chybějící autentizaci server vrací **401 Auhtorization Required**
  - V hlavičce **WWW-Autenticate** je identifikace oblasti přihlášení
  - Klient tedy zjistí, že je nutná autentizace pro tuto oblast

JSON Web Token (JWT)

- Řetězec složený ze 3 částí
  1. Header (hlavička) – účel, použité algoritmy (JSON)
  2. Payload (obsah) – JSON data obsahující id uživatele, jehopráva, expiraci apod.
  3. Signature (podpis) – pro ověření, že token nebyl podvržen nebo změněn cestou
- Tyto tři části se kódují (base64) a spojí do jednoho řetězce
  - **xxxxx.yyyyy.zzzzz**

Použití JWT

- Klient kontaktuje ***autentizační server*** a dodá autentizační údaje
  - Stejný server, jaký poskytuje API, nebo i úplně jiný (např. Twitter)
- Autentizační server vygeneruje podepsaný JWT a vrátí klientovi
- Klient předá JWT při každém volání API
  - Nejčastěji opět v hlavičce:
    Authorization: Bearer xxxxx.yyyyy.zzzzz
  - API ověří platnost, role uživatele může být přímo v JWT

Otázky?