Pokročilé informační systémy

Architektury a principy napříč platformami

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

Třívrstvá architektura

- 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

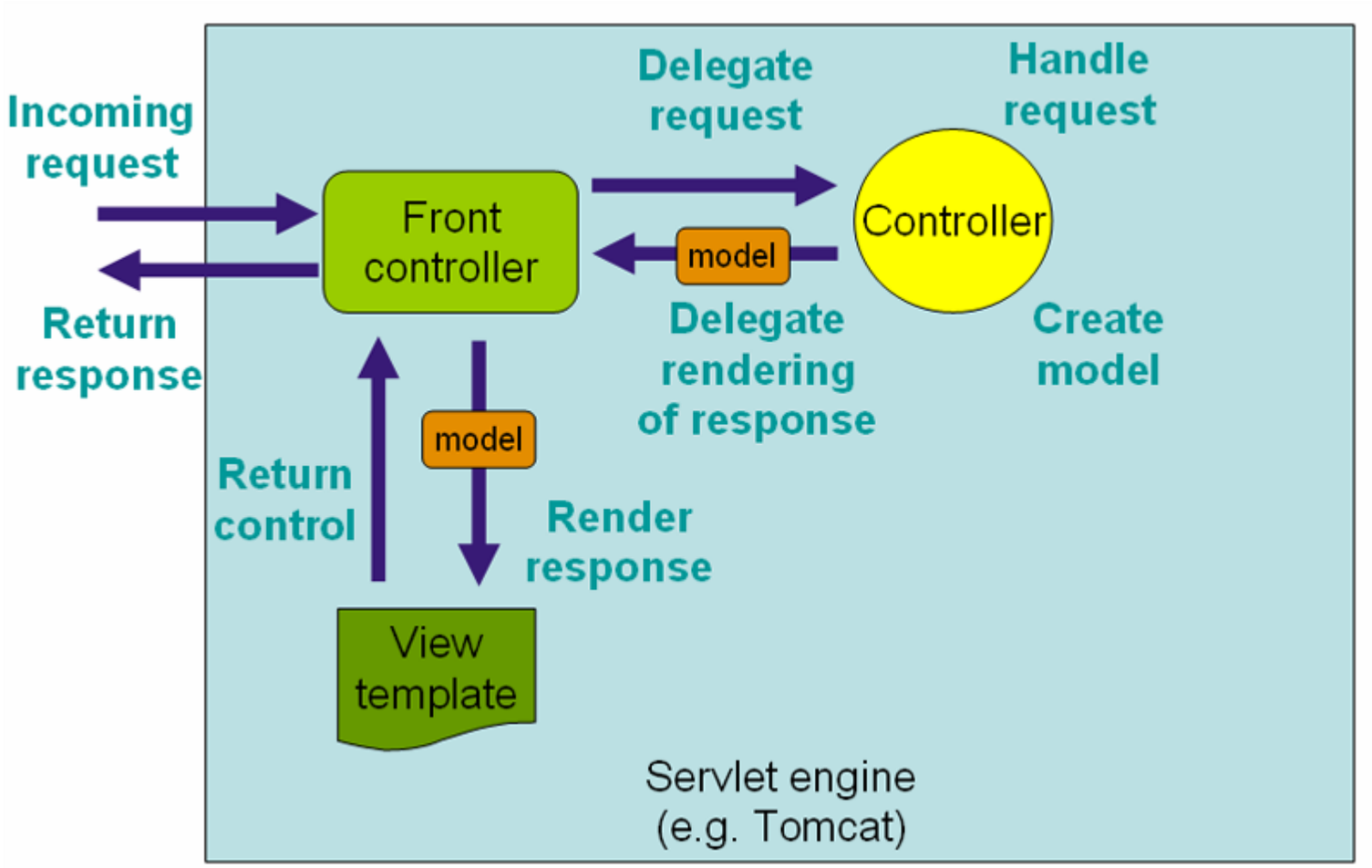
Java – alternativy

- Databázová vrstva
  - Hibernate ORM – implementuje JPA, ale i vlastní API
  - NoSQL databáze – Hibernate OGM, EclipseLink, …
- Business vrstva
  - Spring framework – alternativa EJB pro dependency injection, transakce, správa sezení, …
- Prezentační vrtstva
  - Spring MVC (controllers + JSP / Thymeleaf…)
  - Struts, Play!, …
  - https://www.dailyrazor.com/blog/best-java-web-frameworks/

Spring

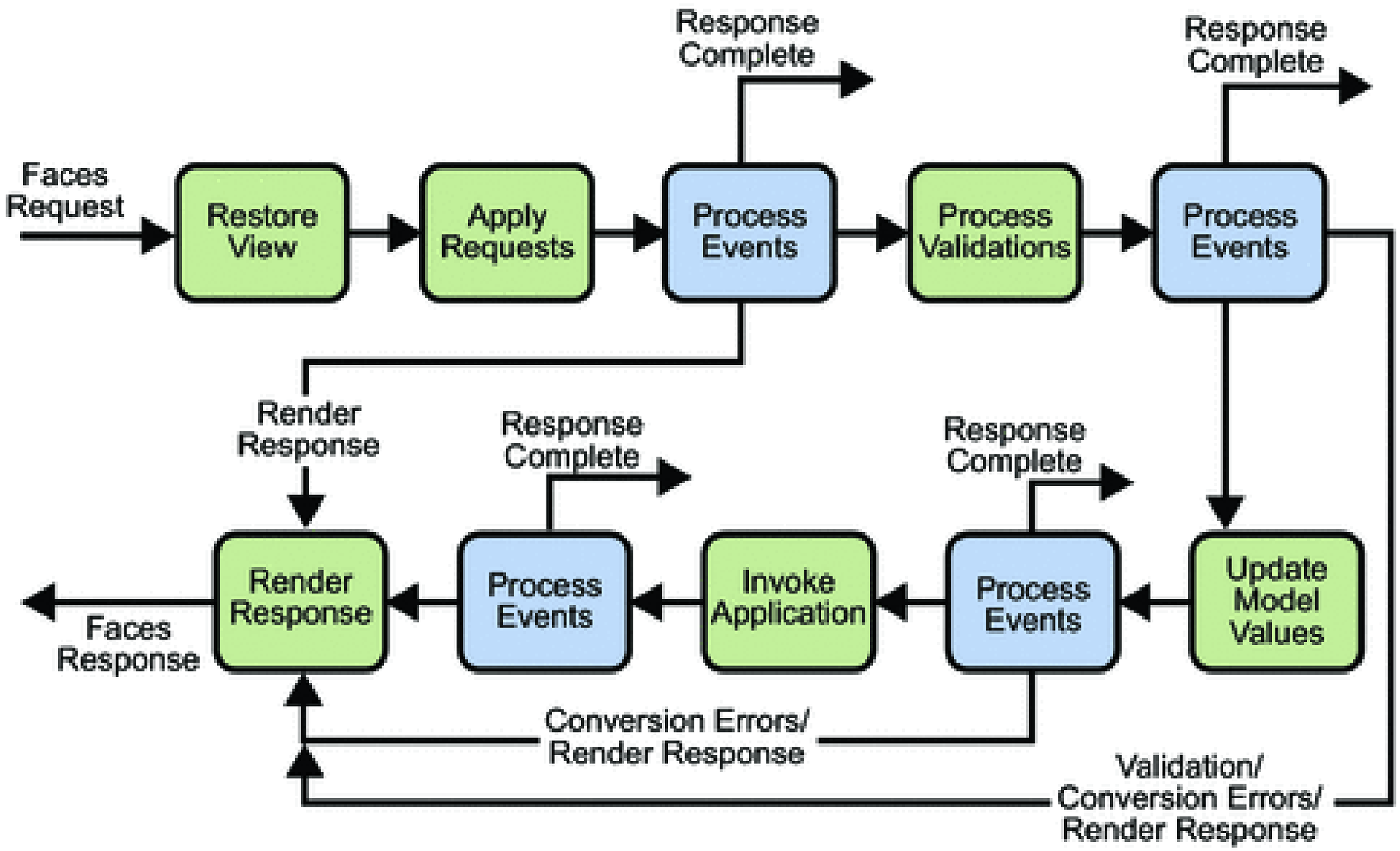
- Vznikl jako alternativa k EJB
  - Využití POJO místo (tehdy složitých) EJB
  - Omezení požadavků na infrastrukturu
- Modulární struktura, mnoho součástí
- Dependency injection
  - Podobně jako v J EE, anotace @Bean, @Autowire, …
  - Opět field, constructor, setter injection
- Spring MVC
  - Tradiční MVC přístup, bližší ostatním frameworkům
  - Ukázková aplikace
    https://github.com/spring-projects/spring-mvc-showcase

Zpracování požadavku Spring MVC



- Handler matching – anotace v controller třídách, vrací popis view (různé formáty) nebo přímo výsledný obsah
- View matching – výběr view podle výsledku (pokud je)
- https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html

Pro srovnání: JSF



Spring Boot

- Přístup „Vše je v aplikaci“ (včetně serveru)
  - Na rozdíl od Java EE – „Server umí vše“ (thin WARs)
- Usnadňuje vytvoření aplikace a konfiguraci závislostí
  - Maven nebo Gradle šablony
  - Spring moduly (MVC, Security, …), Thymeleaf, JPA,…
- Snadné vytvoření funkční aplikace
  - Třída reprezentující celou aplikaci
  - Konfigurace pomocí anotací
  - Spustitelná main() metoda
- https://www.baeldung.com/spring-boot-start

.NET

- .NET Core / .NET Framework
- Databázová vrstva
  - Entity Framework, (LINQ, Dapper, …)
    https://docs.microsoft.com/cs-cz/ef/core/modeling/
- Business vrstva (služby)
  - ASP.NET Core (dependency injection, middleware)
    https://docs.microsoft.com/en-us/aspnet/core/fundamentals/
- Webová vrstva
  - Razor (MVVM) – two-way data binding, HTML
    https://dotnet.microsoft.com/apps/aspnet/web-apps
  - ASP.NET Core MVC – logika+model v C#, view v HTML, REST, …
    https://docs.microsoft.com/cs-cz/aspnet/core/tutorials/first-mvc-app/start-mvc

Entity Framework – entita

```
[Table("Product")]
public class Product
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public int CategoryId { get; set; }

    [Required, StringLength(50)]
    public string Name { get; set; }

    [Required]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [Required]
    public int Stock { get; set; }

    [ForeignKey("CategoryId")]
    public virtual Category Category { get; set; }
}
```

PHP

- PHP je rozšiřující modul HTTP serveru
  - Žádný trvale běžící kontejner
  - + stabilita řešení
  - - efektivita, možnost udržovat stav napříč požadavky
- PHP Frameworky
  - Laravel – (MVC) https://laravel.com/
  - Symfony – (MVC) https://symfony.com/
  - Nette – (MVP) https://nette.org/
  - …
- Správa závislostí – composer

Databázová vrstva

- Různé vlastní přístupy
- Laravel
  - Fluent query builder – specifikace SQL dotazů v PHP
    DB::table('users')->where('name', 'John')->first();
  - Eloquent ORM
- Nette
  - Nette database – parametrizovatelné SQL dotazy
- **Doctrine** – pokročilé ORM, podobné JPA
  - Integrovatelné do všech frameworků

Doctrine: Entita

```php
<?php

use Doctrine\ORM\Annotation as ORM;

/**
 * @ORM\Entity @ORM\Table(name="products")
**/
class Product {

    /** @ORM\Id @ORM\Column(type="integer")
        @ORM\GeneratedValue **/
    protected $id;

    /** @ORM\Column(type="string") **/
    protected $name;



    // .. (other code)

}
```

Doctrine: Uložení objektu

```
$product = new Product();

$product->setName(„Tatranky“);



$entityManager->persist($product);
$entityManager->flush();


echo "Created Product with ID "
    . $product->getId() . "\n";
```

Business vrstva v PHP

- Obvykle v podobě služeb – services
- Framework poskytuje DI kontejner, který registruje služby
  - Procedurálně v PHP nebo externí konfigurační soubor
- Při vytváření controlleru framework dodá závislosti
  - Obvykle *constructor injection*
- Příklady
  - Laravel https://laravel.com/docs/5.8/container#resolving
  - Symfony https://symfony.com/doc/current/components/dependency_injection.html
  - Nette https://doc.nette.org/cs/2.4/dependency-injection

Zpracování požadavku v PHP

1. Požadavek na kořenový dokument (index.php)
2. Bootstrapping frameworku
   - Načtení konfigurace
   - Inicializace součástí, rozšíření, služeb (DI)
   - Obnova session
3. Dekódování parametrů požadavku
   - Směrování požadavku – routing
4. Volání aplikační logiky
   - Vytvoření instance controlleru
   - Volání metody podle požadavku (handler)
5. Vytvoření odpovědi (view rendering)

Zpracování požadavku v PHP – příklady

- Přiřazení controllerů k URL je definováno odděleně Route::get('user/{id}', 'UserController@show‘)
- Controller konfiguruje a vrací view
- https://laravel.com/docs/5.8/controllers

Zpracování požadavku v PHP – Symfony

- Controller je přiřazen k URL pomocí anotací
- Vrací objekt Response
  - Různé druhy, případně včetně obsahu, přesměrování, …
  - Případně sám zajišťuje použití šablon

- https://symfony.com/doc/current/controller.html

Zpracování požadavku v PHP – Nette

- Požadavek vyřizuje *presenter* (metoda `renderXyz(params)`)
- Presenter si sám spravuje model (není formalizováno)
- Předává data do view (template) nebo přímo odesílá odpověď (`sendResponse()`)
- https://doc.nette.org/cs/2.4/presenters

JavaScript – node.js

- Standardní řešení pro JS na serveru
- V8 JavaScript Engine + knihovny
- Procedurální implementace zpracování HTTP požadavků
  - Obdobně jako servlety
- Ukázka: https://nodejs.org/en/docs/guides/getting-started-guide/
- Správce balíků npm
  - Jednoduchá instalace závislostí (knihoven)

Databázová vrstva

- Knihovny pro podporu relačních DB serverů k dispozici v rámci platformy node.js
  - Např. MySQL
    https://expressjs.com/en/guide/database-integration.html#mysql
- Existují i ORM řešení
  - Např. Sequelize
    - Podpora MySQL, SQLite, PostgreSQL, MSSQL
    - https://github.com/sequelize/express-example

Sequelize

```
const User = sequelize.define('user', {
  firstName: {
    type: Sequelize.STRING
  },
  lastName: {
    type: Sequelize.STRING
  }
});

// Vytvoří tabulku
User.sync({force: true}).then(() => {
  // Table created
  return User.create({
    firstName: 'John',
    lastName: 'Hancock'
  });
});

// Dotaz
User.findAll().then(users => {
  console.log(users)
})
```

Business vrstva

- Implementace v JS, žádné standardní řešení
- Modularizace řešena na úrovni node.
- Případné DI řešení
  - https://www.npmjs.com/package/node-dependency-injection

Webová vrstva

- Velké množství frameworků s různými přístupy
  - http://nodeframework.com
- Express
  - Mapování HTTP požadavků na funkce v JS http://expressjs.com/en/guide/routing.html
  - Views pomocí několika template engines http://expressjs.com/en/guide/using-template-engines.html
- Full stack frameworky
  - Těsnější integrace s frontendem, např. Meteor

Mikroslužby (Microservices)

Architektura orientovaná na služby

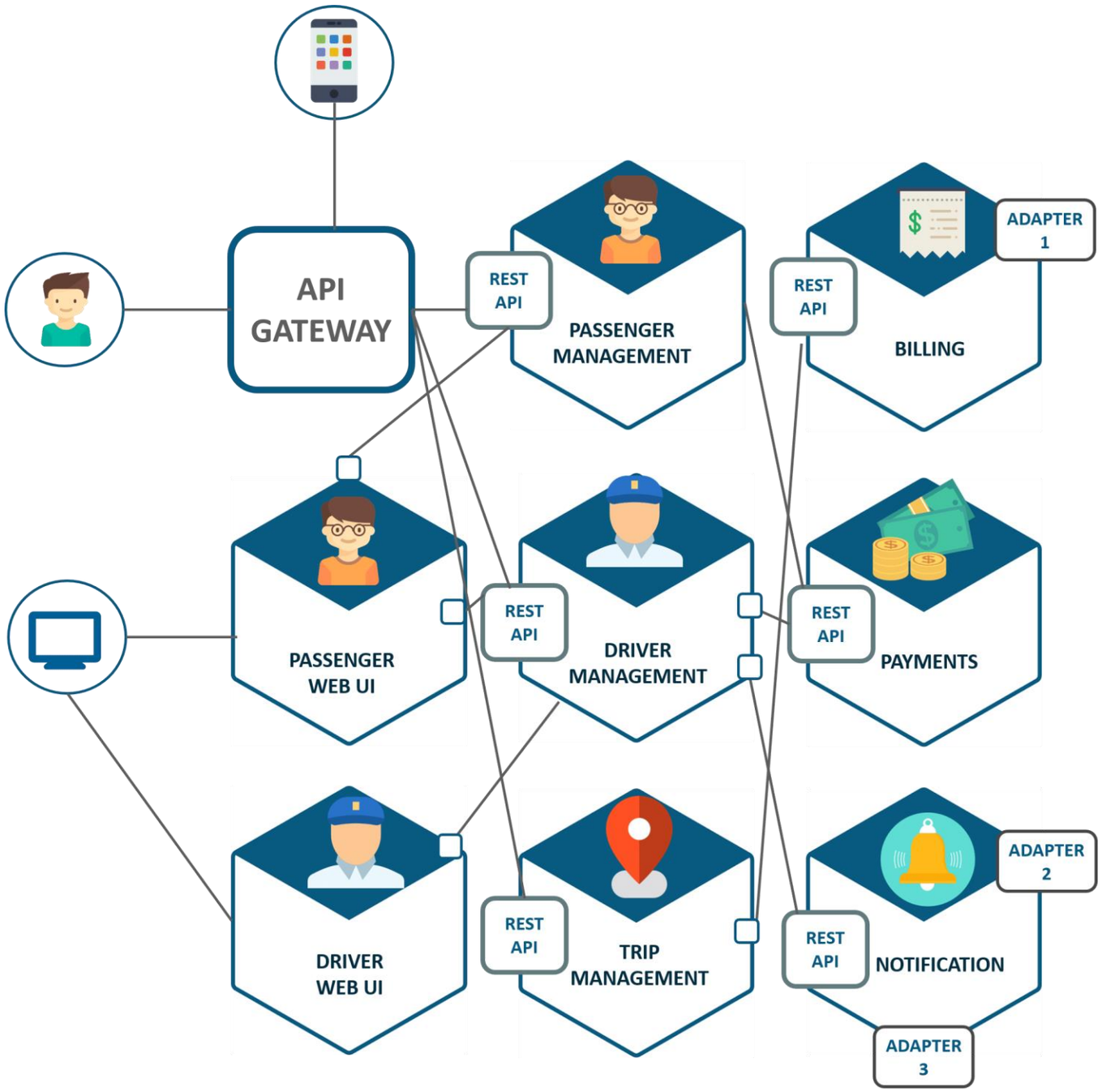
Monolitická architektura

- Jedna aplikace
  - Jedna databáze, webové (aplikační) rozhraní
  - Business moduly – např. objednávky, doprava, sklad, …
- Výhody
  - Jednotná technologie, sdílený popis dat
  - Testovatelnost
  - Rychlé nasazení – jeden balík
- Nevýhody
  - Rozměry aplikace mohou přerůst únosnou mez
  - Neumožňuje rychlé aktualizace částí, reakce na problémy
  - Pokud použité technologie zastarají, přepsání je téměř nemožné

Mikroslužby

- Aplikace je rozdělena na malé části
  - Vlastní databáze (nepřístupná vně)
  - Business logika
  - Aplikační rozhraní (REST)
- Typicky malý tým vývojářů na každou část (2 pizzas rule)
- Výhody
  - Technologická nezávislost
  - Snadné aktualizace, kontinuální vývoj
- Nevýhody
  - Testovatelnost – závislosti na dalších službách
  - Režie komunikace, riziko nekompatibility, řetězové selhání, …

Mikroslužby (příklad: Uber)



Vlastnosti mikroslužby

- Vnější API
  - Dostatečně obecné – reprezentuje logiku, ne např. schéma databáze (která je skrytá)
- Externí konfigurace
- Logování
- Vzdálené sledování
  - Telemetrie – metriky (počty volání apod.), výjimky
  - Sledování živosti (Health check)

V čem tvořit mikroslužby?

- V čemkoliv – spojovacím bodem je pouze API
- Node.js (+ express + MongoDB)
  - Populární rychlé řešení
- Java
  - Spring Boot
  - Ultralehké frameworky
    Např. Spark - https://github.com/perwendel/spark
  - Microprofile

Eclipse Microprofile

- https://microprofile.io/
- Standard založený na Java (Jakarta) EE
  - Podmnožina rozhraní JEE (např. CDI, JAX-RS, JSON-B)
  - Specifická rozhraní pro mikroslužby
    - Config, Health Check, Metrics, JWT Auth, REST Client, …
- Příklad aplikace
  - https://github.com/cicekhayri/payara-micro-javaee-crud-rest-starter-project

Microprofile – další API

- Config
  - Externí konfigurace služby – zdroje, priority, …
- Fault Tolerance
  - Řešení výpadků kvůli závislosti služeb
    - Timeout, Retry, …
- Health Check
  - Vzdálené zjištění živosti mikro služby

Microprofile – další API (II)

- JWT Authentication
- Metrics
  - Statistiky o využití služby – vzdálené měření výkonu
- OpenAPI
  - Generování formalizované dokumentace API služby
- REST Client
- Příklady
  https://github.com/payara/Payara-Examples/tree/master/microprofile

Aplikační rozhraní

Alternativy k REST

Standardizace API

- Předchůdci REST
  - Snaha o maximální standardizaci volání serverových služeb přes HTTP
  - Vznik komplikovaných standardů webových služeb (SOAP atd.)
  - Obtížně použitelné bez podpůrných technologií – omezení na konkrétní implementační platformy
- REST – zjednodušení v reakci na komplikovanost WS
  - Flexibilita, ale žádný standard – mnoho ad hoc řešení
- GraphQL

XML-RPC

- Předchůdce webových služeb
- Jednodušší – stále používané
- Definované XML zprávy pro předání parametrů i výsledku
- Podpora datových typů včetně polí, seznamů a struktur

XML-RPC volání

```
<?xml version="1.0"?>
 <methodCall>
   <methodName>trida.jePrvocislo</methodName>
   <params>
         <param>
            <value><int>1345</int></value>
         </param>
   </params>
</methodCall>
```

XML-RPC výsledek

```
<?xml version="1.0"?>
 <methodResponse>
        <params>
           <param>
              <value><boolean>0</boolean></value>
           </param>
        </params>
</methodResponse>
```

Volání v PHP

```
function xmlrpc($url, $method, $params, $types = array(), $encoding = 'utf-8') {
    foreach ($types as $key => $val) {
        xmlrpc_set_type($params[$key], $val);
    }
    $context = stream_context_create(array('http' => array(
        'method' => "POST",
        'header' => "Content-Type: text/xml",
        'content' => xmlrpc_encode_request($method, $params, array('encoding' => $encoding))
    )));
    return xmlrpc_decode(file_get_contents($url, false, $context), $encoding);
}
```

Webové služby (Web Services)

- Vzdáleně volané podprogramy umístěné na serveru
- Volání pomocí protokolu HTTP
  - Předání vstupních parametrů
  - Vrácení výsledku
- Předávají se XML data definovaná protokoly
  - WSDL – popis rozhraní služby
  - SOAP – volání služby

Popis rozhraní služby: WSDL

- Web Services Description Language
- Platformově nezávislý popis rozhraní
  - XML dokument
  - Využíva XML Namespaces a XML Schema
- Definuje
  - Názvy funkcí
  - Jejich parametry
  - Způsob volání (vstupní URL)

Příklad popisu služby

```
<message name="jePrvocisloRequest">
 <part name="cislo" type="xsd:long"/>
</message>
<message name="jePrvocisloResponse">
 <part name="return" type="xsd:boolean"/>
</message>
<portType name="Cisla">
 <operation name="jePrvocislo“ parametrOrder="cislo">
     <input message="m:jePrvocisloRequest"
name="jePrvocisloRequest"/>
        <output message="m:jePrvocisloResponse“
 name="jePrvocisloResponse"/>
 </operation>
</portType>
```

Příklad popisu služby (II)

```
<binding name="cislaSoapBinding" type="m:Cisla">
 ...
</binding>
<service name="CislaService">
 <port binding="m:cislaSoapBinding“ name="cisla">
    <wsdlsoap:address location="http://nekde.cz/cisla"/>
 </port>
</service>
```

Použití WSDL

- Na základě popisu lze automaticky generovat rozhraní v cílovém jazyce
  - „Stub“ - zástupnou metodu, která implementuje volání skutečné metody přes HTTP
- Rovněž lze generovat WSDL popis z rozhraní v cílovém jazyce
- Současné vývojové nástroje umožňuji vytvoření WS z funkce „jedním kliknutím“
  - Např. v Eclipse

Volání služby: SOAP

```
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    env:encodingStyle="
              http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xs="http://www.w3.org/1999/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <env:Header/>
  <env:Body>
   <m:jePrvocislo xmlns:m="urn:mojeURI">
     <cislo xsi:type="xs:long">1987</cislo>
   </m:jePrvocislo>
  </env:Body>
</env:Envelope>
```

Odpověď služby

```
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <env:Body>
    <ns1:jePrvocisloResponse
    xmlns:ns1="urn:mojeURI"
    env:encodingStyle="
            http://schemas.xmlsoap.org/soap/encoding/">
     <return xsi:type="xsd:boolean">true</return>
   </ns1:jePrvocisloResponse>
 </env:Body>
</env:Envelope>
```

Komplexní příklady

- WSDL
  http://www.w3.org/TR/wsdl20-primer/#basics-greath-scenario
- SOAP
  http://www.w3.org/TR/soap12-part0/#L1165
- W3C Web Services Activity
  http://www.w3.org/2002/ws/#documents

Vyhledání webových služeb

- Idea centrálního registru
  - Protokol UDDI (Universal Description, Discovery and Integration)
  - Poskytovatelé ukládají WSDL popisy, klienti prohledávají
- Přehled služeb daného poskytovatele
  - WSIL (Web Services Inspection Language)
  - Seznam služeb v souboru `/inspection.wsil`

Implementace WS

- Java EE
  - JAX-WS API součástí standardu
  - Vytvoření webových služeb pomocí anotací
  - Běží na JavaEE aplikačním serveru
- PHP
  - Rozšíření SOAP
  - Třídy SoapServer, SoapClient, ...

Java EE

- Součástí Java EE je *Java API for XML Web Services (JAX-WS)*
- Server
  - Definice služeb pomocí anotací tříd a metod
  - Automaticky generuje WSDL popis
- Klient
  - Automatické generování proxy třídy z WSDL popisu

Java EE Implementace

```
package helloservice.endpoint;

import javax.jws.WebService;
import javax.jws.webMethod;

@WebService
public class Hello {

    public Hello() {     }

    @WebMethod
    public String sayHello(String name) {
        return "Hello, " + name + ".";
    }
}
```

PHP

- Rozšíření SOAP
- Server
  - Třída **SoapServer**
  - Registruje třídy a metody implementující službu
- Klient
  - Třída **SoapClient**
  - Zpracuje WSDL a zpřístupní vzdálené metody

PHP Soap Server

```
<?php

  function sayHello($name) {
      return "Hello, " . $name;
  }

  $server = new SoapServer(null,
          array('uri' => "urn://helloservice/endpoint"));

  $server->addFunction('sayHello');
  $server->handle();

?>
```

PHP SOAP Klient

```
$client = new SoapClient(null,
   array('location' => "http://.../simple_server.php",
              'uri'      => "urn://my/namespace"));

  $name = "Karel";
  $result = $client->
                __soapCall("sayHello",array($name));

  print $result;
```

PHP s WSDL

```
$soap = new SoapClient(
   'http://api.search.live.net/search.wsdl');


print_r($soap->__getFunctions());
```

```
$ret = $soap->Search(…);
```

Popis služeb v REST

- Obdoba WSDL pro REST
- WADL (Web Application Description Language)
  - Založený na XML
  - Podpora v Javě – např. Payara
- OpenAPI https://swagger.io/specification/
  - Používá YAML (alternativně JSON)
  - Např. Payara http://localhost:8080/openapi/

GraphQL

- https://graphql.org/
- Motivace: klient (klienti) potřebují v různých situacích různá data
  - Např. stránka „seznam osob“ vs. „detail osoby“
    - REST endpoint vrací vždy stejnou strukturu
      - Redundance dat (nevyužijeme všechna data)
      - Více dotazů ((ne)efektivita, složitější logika klienta)
- Řešení GraphQL
  - Popis datového modelu API
  - Dotaz na API specifikuje požadovaný tvar odpovědi

GraphQL – datový model

- Datový model API (ne nutně serverové aplikace)
- Jednoduché datové typy: Int, Float, String, Boolean, ID, enum
- Uživatelské typy (*types*) = struktury
  - Vlastnosti (parametrizovatelné) – jméno, parametry, typ
  - Typy jednoduché, struktury (vztahy), kolekce
- Speciální typy reprezentující volání API (*root types*)
  - Query – čtení dat
  - Mutation – změna dat
- GraphQL Schema Definition Language (SDL)

GraphQL – definice typů

```
type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}
```

```
type Car {
  type: String!
  reg: String!
  owner: Person!
}
```

```
type Query {
  allPersons: [Person!]!
  findPerson(name: String!): Person!
}
```

```
type Mutation {
  createPerson(name: String!, age: Int!): Person!
}
```

GraphQL – dotazy

```
{
  allPersons {
    name
  }
}
```

```
{
  "allPersons": [
    { "name": "Jan" },
    { "name": "Karolína" },
    { "name": "Alice" }
  ]
}
```

```
{
  findPerson(name: "James") {
    name
    age
    cars {
      type
    }
  }
}
```

```
{
  "findPerson": {
    "name": "James",
    "age": 28,
    "cars": [
        { type: "Fiat" },
        { type: "Tesla" }
    ]
  }
}
```

GraphQL – modifikace

```
mutation {
  createPerson(name: "Bob", age: 36) {
    name
    age
  }
}
```

```
"createPerson": {
  "name": "Bob",
  "age": 36
}
```

GraphQL přes HTTP

- Jediné endpoint URL
- Odeslání přes GET
  `http://myapi/graphql?query={me{name}}`
- Odeslání přes POST
  - Data application/json
    `{"query": "{me{name}}"}`
  - Data application/graphql
    `{me{name}}`

Otázky?