Programska potpora komunikacijskim sustavima

8. predavanje, 10. svibnja 2023.

doc. dr. sc. Jelena Božek

Arhitekturni stil REST

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

- Predavanja izradio prof. dr. sc. Krešimir Pripužić, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Sadržaj predavanja

- Općenito o REST-u
- Ograničenja koja uvodi REST
- Resursi u REST-u
- CRUD resursa u REST-u korištenjem metoda HTTP zahtjeva
- Primjer REST-usluge

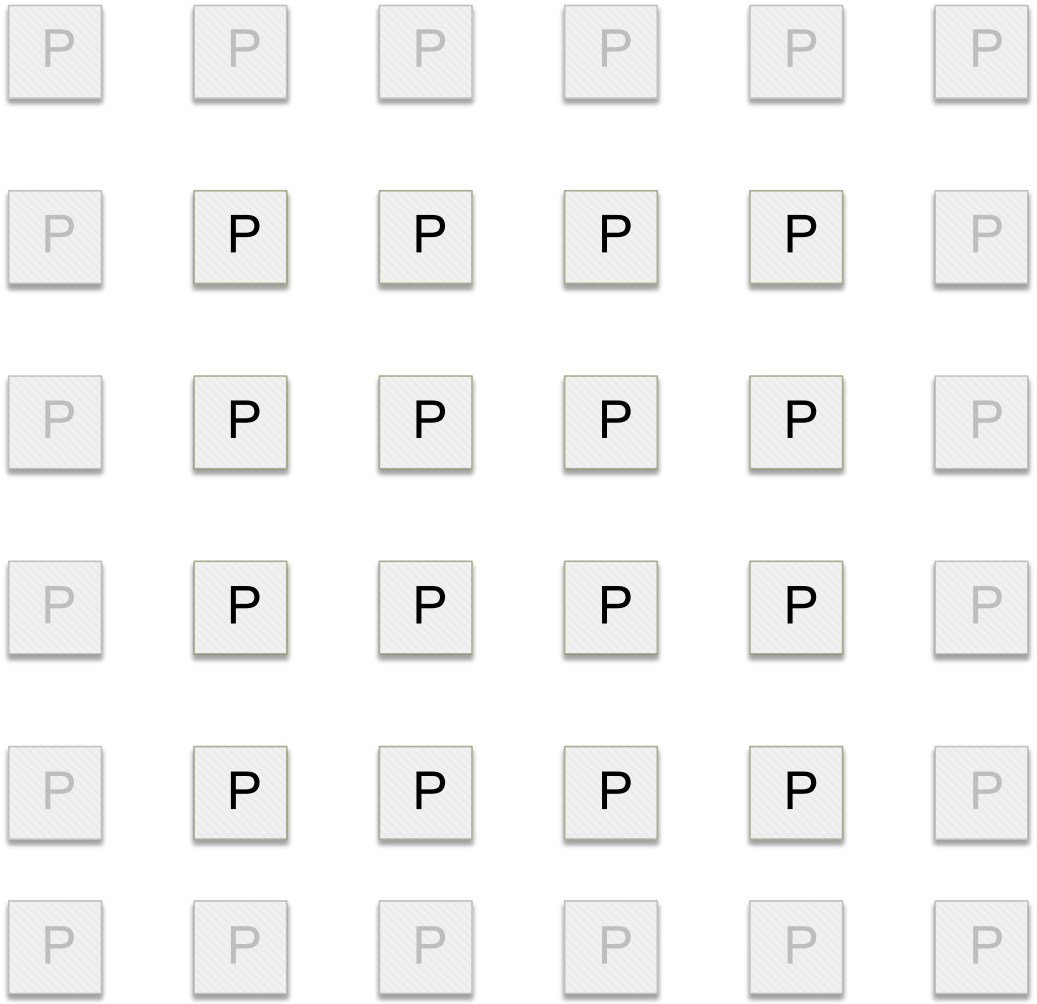
REST (*Representational State Transfer*)

- Hrvatski prijevod: **prijenos prikaza stanja resursa**
- Pojam je 2000. godine skovao Roy Fielding u svojoj doktorskoj disertaciji „Architectural Styles and the Design of Network-based Software Architectures”
- Nije standard već je **arhitekturni stil** za *World Wide* ***Web*** (WWW)
  - Neovisan je o konkretnoj implementaciji tj. tehnologiji
  - Uvodi ograničenja kojih se trebamo držati pri razvoju i implementaciji programskih komponenti na Webu
  - Definira način na koji trebaju komunicirati raspodijeljeni procesi na Webu
- RESTful API (*application programming interface*) je API koji poštuje ograničenja arhitekturnog stila REST i omogućava interakciju s RESTful web-uslugama

Web-usluga ≠ web-aplikacija

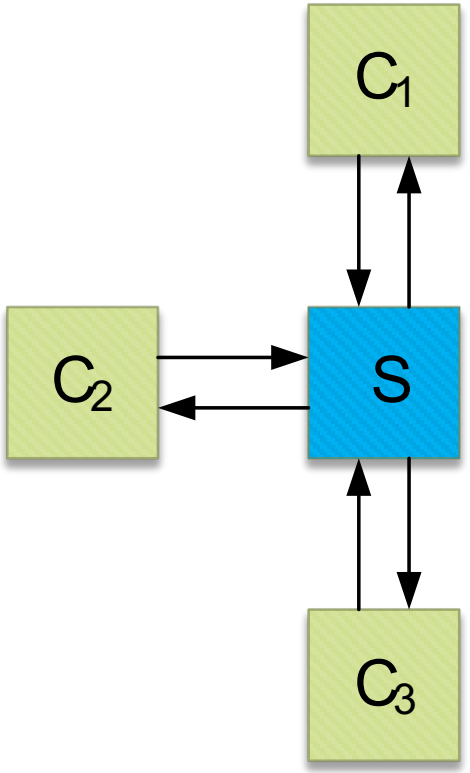
- Web-aplikacija
  - Tema prošlotjednog predavanja
  - Aplikacija kojoj se pristupa putem web-preglednika koji radi na računalu krajnjeg korisnika
  - Namijenjena je ljudima
- Web-usluga
  - Tema današnjeg predavanja
  - Usluga koja omogućuje međusobnu interakciju **raspodijeljenih procesa** na Webu
  - Namijenjena je (procesima na) računalima

Raspodijeljeni procesi na Webu



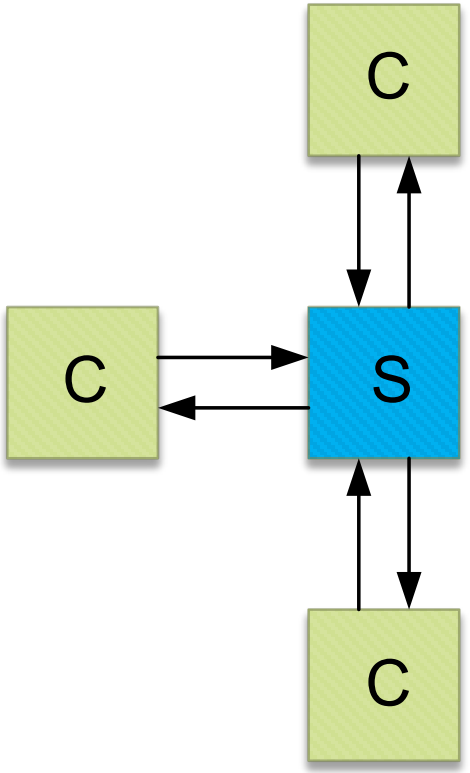
Ograničenje 1: model klijent-server

- Jasna podjela odgovornosti (*separation of concerns, SoC*) na klijenta i poslužitelja
  - Ne rade svi sve
  - Odgovornost za korisničko sučelje je razdvojeno od odgovornosti za pohranu podataka
- Prednosti ovog ograničenja
  - Pojednostavljen je razvoj programskih komponenti
  - Poboljšava se skalabilnost usluge
  - Olakšane su programske nadogradnje



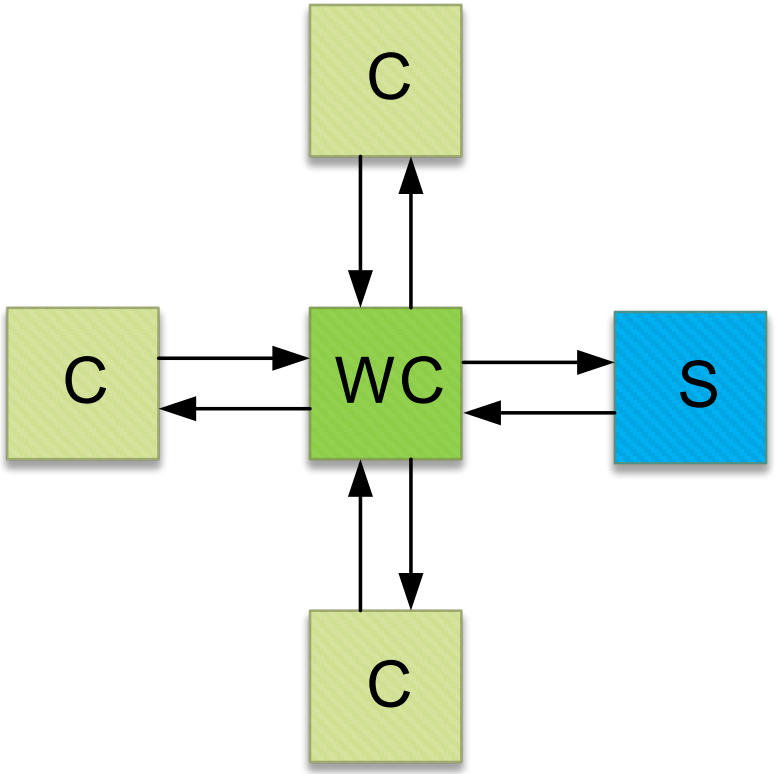
Ograničenje 2: Nema stanja

- Poslužitelj ne pamti stanje komunikacije s klijentom
  - Poslužitelj ne pravi razliku između klijenata i njihovih pojedinačnih zahtjeva
  - Svaki zahtjev mu je neovisan o prethodnim zahtjevima
  - Zahtjev u sebi sadrži sve informacije koje su potrebne poslužitelju da napravi odgovor
- Prednosti ovog ograničenja
  - Poslužitelj je rasterećen jer ne mora pamtiti stanje komunikacije (*session*) sa svakim pojedinim klijentom
  - Poboljšane su performanse i skalabilnost poslužiteljske komponente



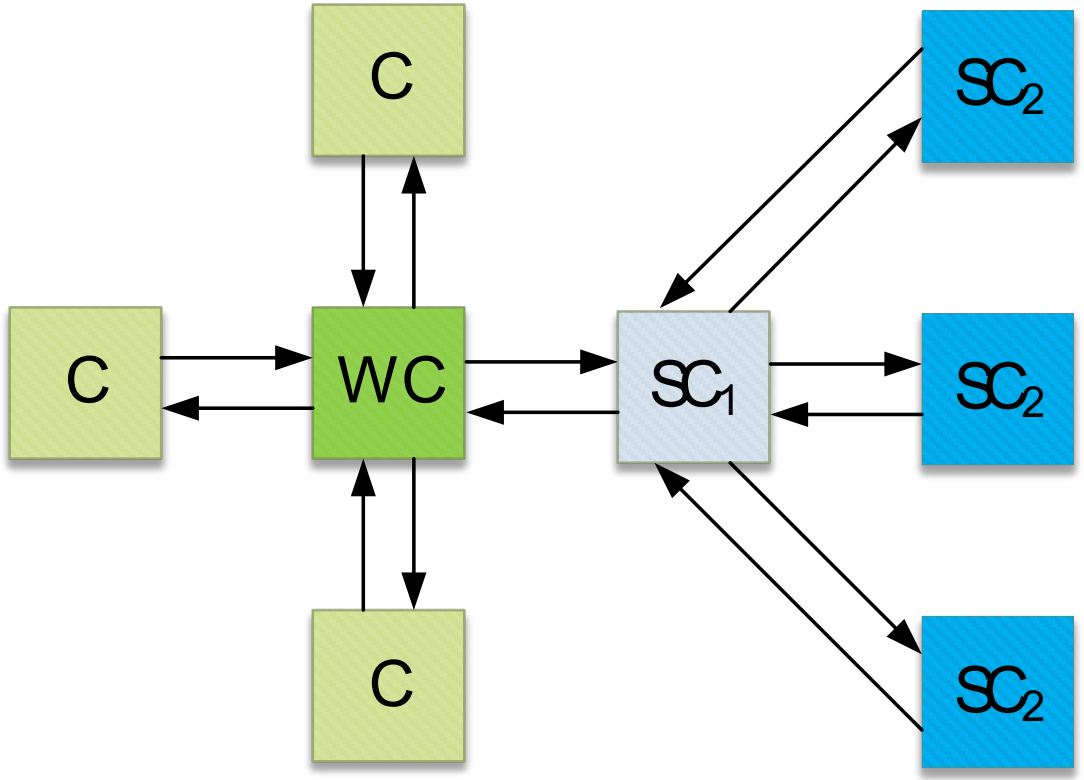
Ograničenje 3: Pamtljivi odgovori

- Odgovori moraju moći biti pamtljivi (*cacheability*) tj. moraju se moći pohraniti u priručno spremište
  - Privremena spremišta se mogu nalaziti kod klijenata ili kod posebnih spremišta na Webu (*Web Cache, WC*) koja su smještena između klijenata i poslužitelja
  - Svaki odgovor mora sadržavati informaciju o tome smije li se privremeno pohraniti ili ne
- Prednosti ovog ograničenja
  - Poslužitelj je rasterećen jer ne mora odgovarati na sve zahtjeve
  - Poboljšane su performanse i skalabilnost poslužiteljske komponente



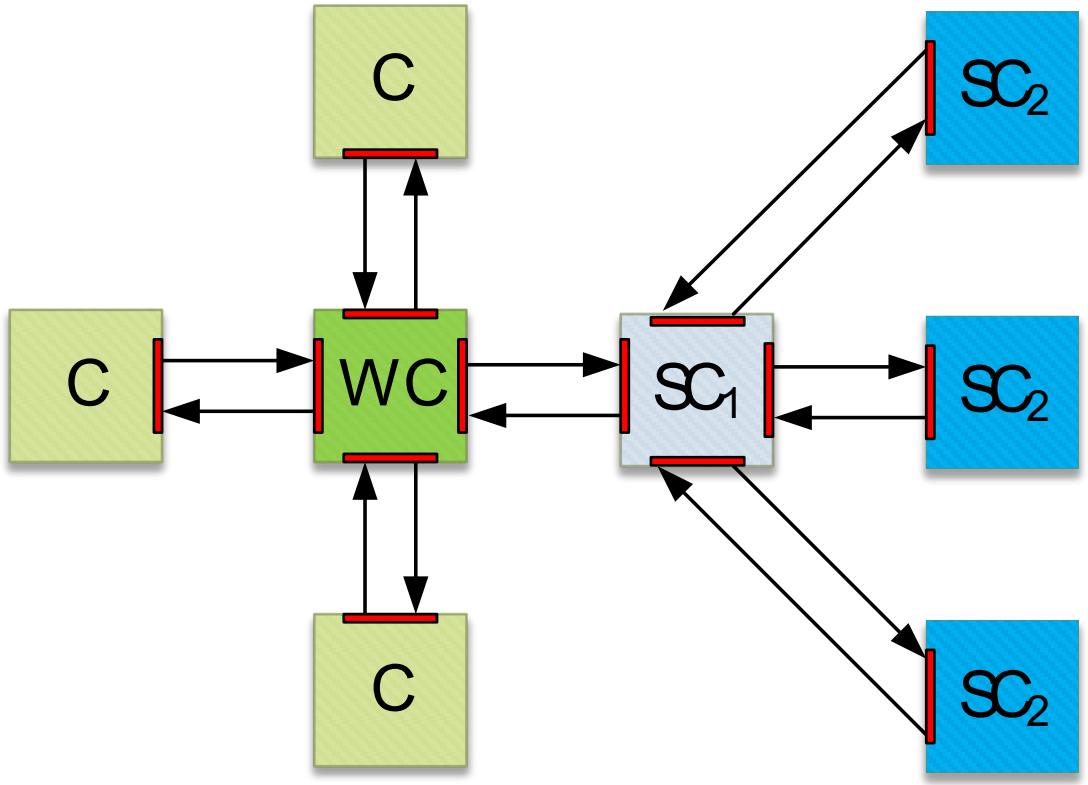
Ograničenje 4: Slojevitost poslužitelja

- Arhitektura poslužiteljske komponente može biti slojevita
  - Između poslužitelja i klijenta se moraju moći nalaziti poslužiteljske komponente kao što su zastupnik (*proxy*) ili uravnoteživač opterećenja (*load balancer*)
  - Slojevitost arhitekture poslužitelja ne smije zahtijevati izmjene programskog koda poslužitelja i klijenata
- Prednosti ovog ograničenja
  - Uravnoteživanjem opterećenja su poboljšane performanse i skalabilnost poslužiteljskog dijela



Ograničenje 5: Uniformo sučelje

- Sučelje između komponenti treba biti uniformno
  - Identifikacija resursa
  - Manipulacija resursa njihovom reprezentacijom
  - Samoopisujuće poruke
  - Princip HATEOAS
- Prednosti ovog ograničenja
  - Olakšan razvoj programskih komponenti



Identifikacija resursa

- U praksi se koristi URI (*Uniform Resource Identifier*)
- Sintaksa URI-ja:

```
scheme ":" ["//" authority] path ["?" query] ["#" fragment]
authority = [userinfo "@"] host [":" port]
```

- Primjeri:
  - http://xyz.com/users/ - popis svih korisnika
  - http://xyz.com /users/1 - korisnik s identifikatorom 1
  - http://xyz.com /users/1/name - ime korisnika s identifikatorom 1

Reprezentacija resursa

- Pojedini resurs može imati više reprezentacija (xml, html, json, jpg)

application/xml

```
<user>
   <id>1</id>
   <name>Ivo</name>
   <surname>Ivić</surname>
</user>
```

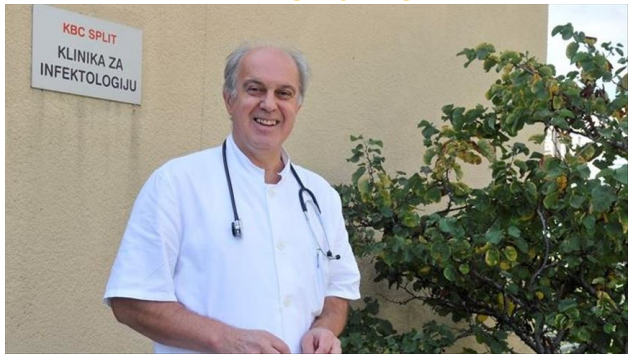
application/json

```
{
    "id":1,
    "name":"Ivo",
    "surname":"Ivić"
}
```

text/html

```
<p class="user">
    <span class="id">1</span>
    <span class="name">Ivo</span>
    <span class="surname">Ivić</span>
</p>
```

image/jpeg



Samoopisujuće poruke

- Svaka poruka koja se razmjenjuje opisuje način na koji je treba parsirati tj. pročitati i protumačiti
- U praksi se koristi tip medija (*media type*) u standardnom obliku kako ga klasificira organizacija Internet Assigned Numbers Authority (IANA)
- Sintaksa tipa medija:

```
type "/" [tree "."] subtype ["+" suffix]* [";" parameter]
```

- Primjeri tipa medija
  - `application/json`
  - `application/xml`
  - `application/html`
  - `image/jpeg`

Princip HATEOAS

- HATEOAS (*Hypermedia As The Engine Of Application State*)

- Osnovno znanje o hipermediji je dovoljno za interakciju s poslužiteljem
- Aplikacijski poslužitelj klijentu daje informacije koristeći hipermediju

- Hipermedija je nelinearni medij informacije koji uključuje grafiku, audio, video, tekst i poveznice (*hyperlinks*)

```
HTTP/1.1 200 OK

{
    "account": {
        "account_number": 12345,
        "balance": {
            "currency": "usd",
            "value": 100.00
        },
        "links": {
            "deposits":
"/accounts/12345/deposits",
            "withdrawals":
"/accounts/12345/withdrawals",
            "transfers":
"/accounts/12345/transfers",
            "close-requests":
"/accounts/12345/close-requests"
        }
    }
}
```

CRUD

- REST u praksi koristi metode HTTP zahtjeva da bi radio CRUD operacije nad resursima

HTTP request method	CRUD
GET	**READ**
HEAD	
POST	**CREATE**
PUT	**(FULL) UPDATE**
DELETE	**DELETE**
CONNECT	
OPTIONS	
TRACE	
PATCH	**(PARTIAL) UPDATE**

Svojstva metoda HTTP zahtjeva

- Sigurna (*safe*) - bez posljedica za podatke
- Indempodentna (*idempotent*) - može se izvršavati više puta
- Pamtljiv odgovor (*cachable*)

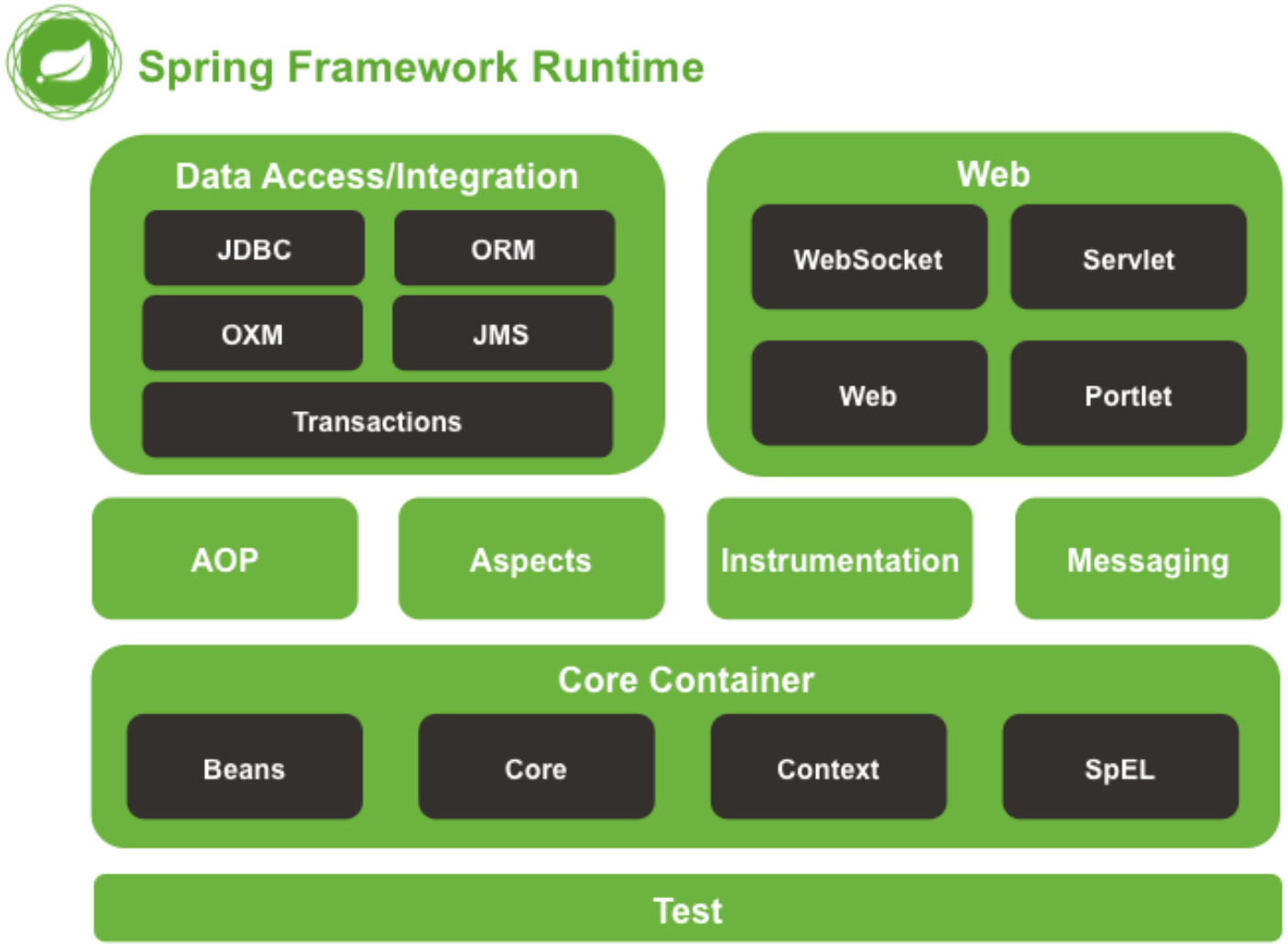
No (RESTful API)

HTTP request method	Request has payload body	Response has payload body	Safe	Idempotent	Cacheable
GET	**Optional**	**Yes**	**Yes**	**Yes**	**Yes**
HEAD	*Optional*	*No*	*Yes*	*Yes*	*Yes*
POST	**Yes**	**Yes**	**No**	**No**	**Yes**
PUT	**Yes**	**Yes**	**No**	**Yes**	**No**
DELETE	**Optional**	**Yes**	**No**	**Yes**	**No**
CONNECT	*Optional*	*Yes*	*No*	*No*	*No*
OPTIONS	*Optional*	*Yes*	*Yes*	*Yes*	*No*
TRACE	*No*	*Yes*	*Yes*	*Yes*	*No*
PATCH	**Yes**	**Yes**	**No**	**No**	**No**

Spring - http://spring.io

- Radni okvir za lakši razvoj web-aplikacija
- Bavi se konfiguracijom objekata u sustavu
- Upravlja poslovnim objektima kao običnim objektima (*POJO – plain old java objects*)
- Brine se za kreiranje objekata (*IoC – inversion of control*)
- Povezuje kreirane objekte (*IoC, wiring up, dependency injection*)
- Upravlja njihovim životnim ciklusom
- Složene veze između objekata se definiraju u XML-u ili pomoću bilješki (*annotation*)
- Odvaja poslovnu logiku od mehanizama za ispravan rad sustava (transakcije, logiranje, ...)
- Vrlo je složen za početnika jer ima puno stvari ugrađeno

Springova arhitektura - dokumentacija



Spring Boot

- Jedan od Springovih podprojekata
  - http://projects.spring.io/spring-boot/
- Pojednostavnjuje korištenje Springa
- Podržava:
  - Automatsku konfiguraciju
  - Pretraživanja klasa na putu (*path*)
- Aplikacija ima manje koda
- Kod web-aplikacija - omogućuje izradu samostalnih aplikacija
  - Web-poslužitelj zapakiran u jar
  - Jednostavnije instaliranje
  - Aplikacija spremna za produkcijsku okolinu

Primjer: Persons

- Popis nastavnika (osoba)
- Dohvaćanje i spremanje u memorija poslužitelja

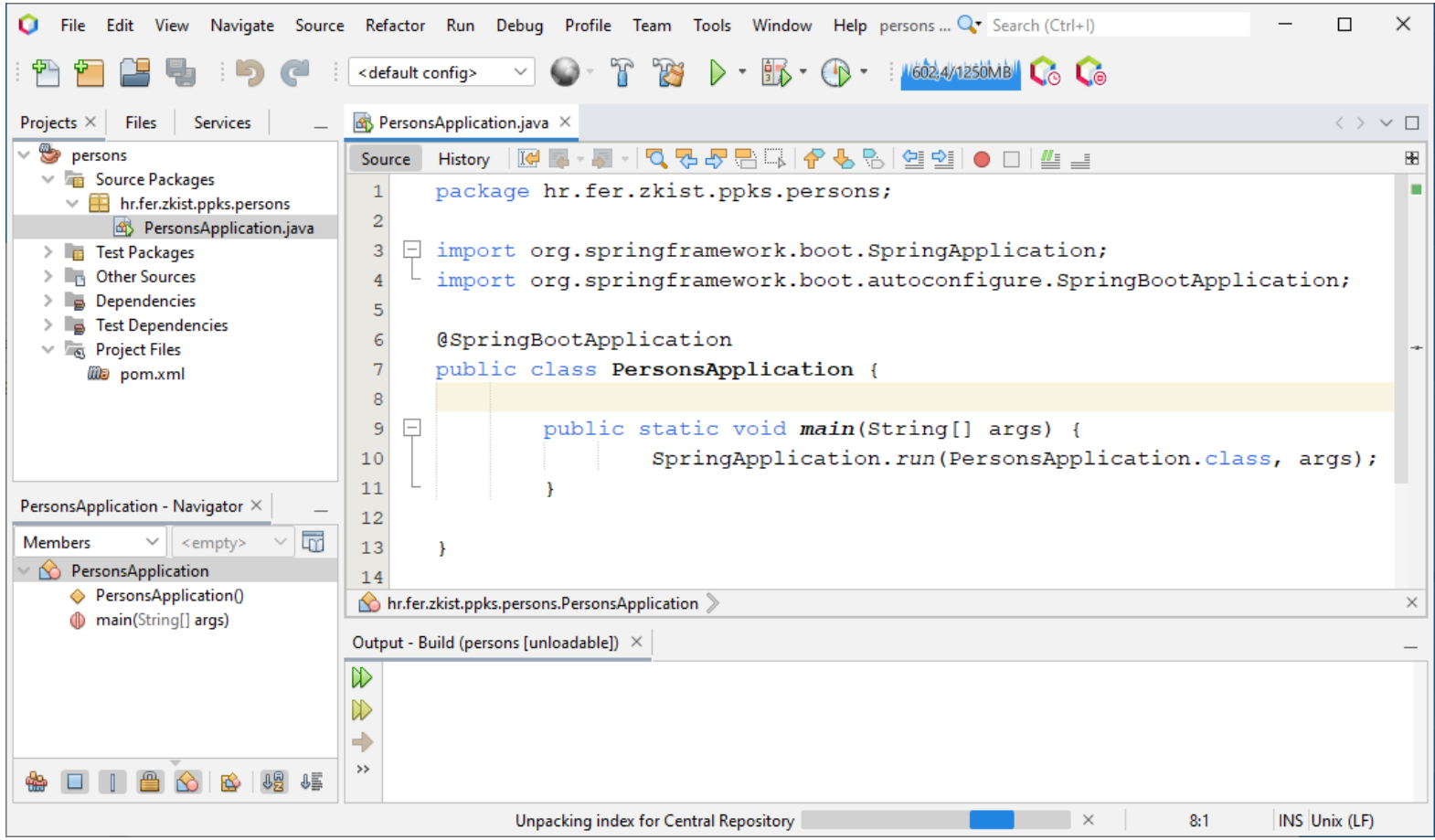
- Prikaz u JSON-u (REST)
  - uređivanje (dodavanje, promjena, brisanje) podataka

Stvaranje Spring Boot projekta

- Otvoriti stranicu https://start.spring.io
- Odabrati:
  - **Project:** **Maven** Project (u primjeru u repozitoriju je Gradle Project)
  - **Language:** Java
  - **Spring Boot:** 2.6.7 (ili novije 3.0.6)
- Popuniti **Project Metadata**:
  - **Group:** hr.fer.zkist.ppks
  - **Artifact:** persons
  - **Name:** persons
  - **Package name:** hr.fer.zkist.ppks.persons
- Odabrati:
  - **Packaging:** Jar
  - **Java Version:** 17
- Odabrati **Dependencies**:
  - **Spring Web**
  - **REST repositories**
  - **Spring HATEOAS**
- Kliknuti na **Generate Project**
- Uvesti projekt u razvojnu okolinu (Apache NetBeans, Eclipse, IntelliJ IDEA)

Spring Boot projekt

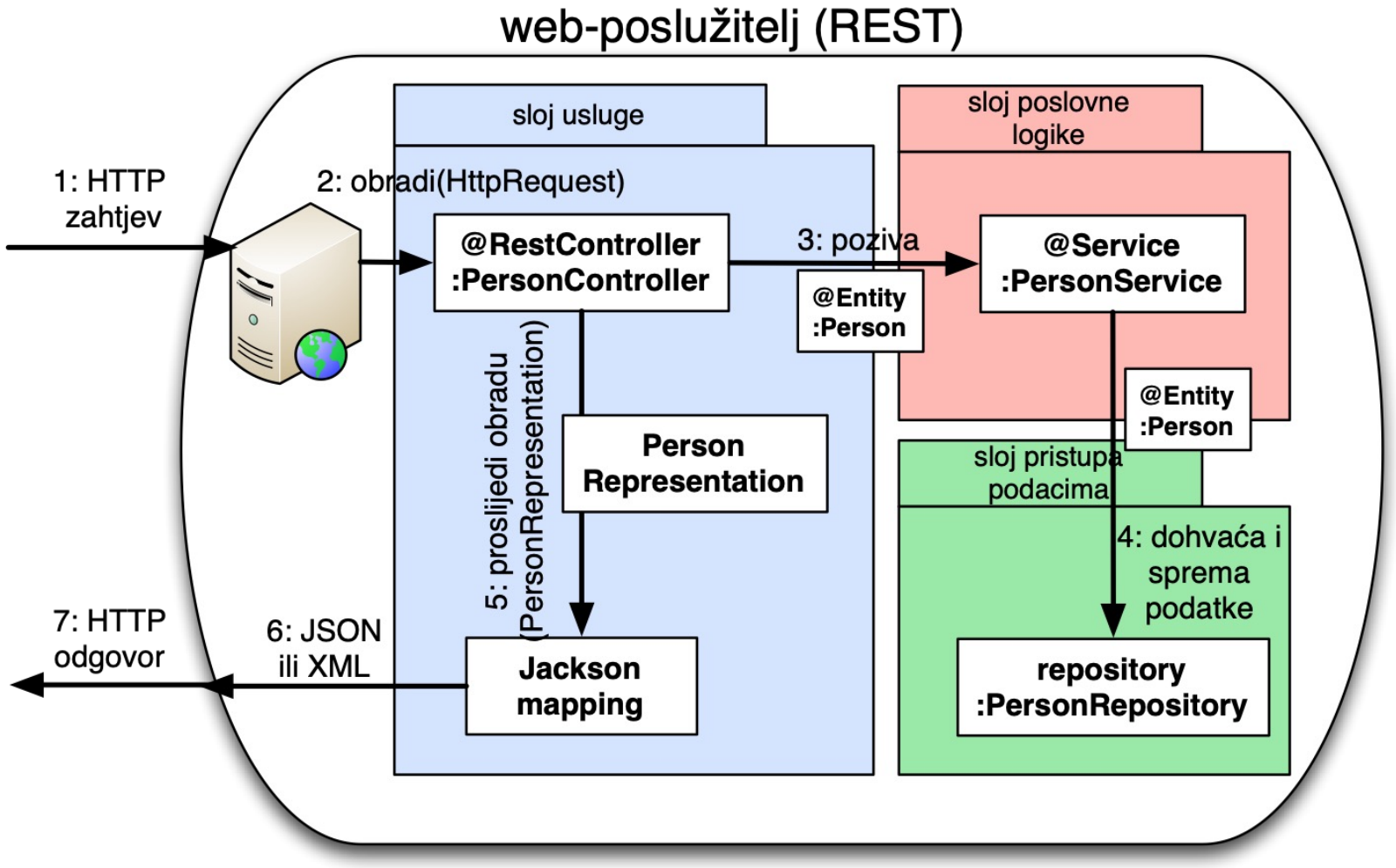
- struktura projekta
- pogledati:
  - klasa `PersonsApplication`
    - klasa koja se pokreće
  - datoteka `pom.xml`
    - skripta za izgradnju



Primjer RESTful web-usluge

resurs	podržane metode	šalje	svrha
/persons	GET		vraća listu osoba
	POST	osoba	stvara novu osobu
/persons/{id}	GET		vraća osobu s ID-om
	DELETE		briše osobu
	PUT	osoba	mijenja podatke o osobi

Arhitektura RESTful web-usluge



Primjer upita i odgovora [1]

- GET /persons
- odgovor: 200 OK


```
[
 {
   "id": 0,
   "firstName": "Goran",
   "lastName": "Delač",
   "phone": "01/6129-549",
   "room": "D-339-2"
 },
 {
   "id": 1,
   "firstName": "Marin",
   "lastName": "Šilić",
   "phone": "01/6129-549",
   "room": "D-339-2"
 }
]
```

Primjer upita i odgovora [2]

- POST /persons
- sadržaj zahtjeva:

```
{
    "id": 4,
    "firstName": "Mislav",
    "lastName": "Grgić",
    "phone": "01/6129-851",
    "room": "C11-06"
}
```

- odgovor: 201 Created
- header: "location" : "http://localhost:8080/persons/4"

Primjer upita i odgovora [3]

- GET /persons/4
- odgovor: 200 OK

```
{
    "id": 4,
    "firstName": "Mislav",
    "lastName": "Grgić",
    "phone": "01/6129-851",
    "room": "C11-06"
}
```

Primjer upita i odgovora [4]

- GET /persons/5
- odgovor: 404 Not Found


```
{
  "timestamp": "2022-05-10T13:48:15.450+00:00",
  "status": 404,
  "error": "Not Found",
  "path": "/persons/4"
}
```


Primjer upita i odgovora [5]

- PUT /persons/4

```
{
    "id": 4,
    "firstName": "Mislav",
    "lastName": "Grgić",
    "phone": "01/6129-999",
    "room": "C11-06"
}
```

- odgovor: 204 No Content

Primjer upita i odgovora [6]

- PUT /api/persons/100

```
{
    "id": 100,
    "firstName": "Mislav",
    "lastName": "Grgić",
    "phone": "01/6129-851",
    "room": "C11-06"
}
```

- odgovor: 304 Not Modified
- header: "etag" : "Person with id 100 not found."

Primjer upita i odgovora [7]

- DELETE /persons/2
- odgovor: 204 No Content

Primjer upita i odgovora [8]

- DELETE /persons/100
- odgovor: 404 Not Found

Klasa Person

```
public class Person implements Serializable {

    private final long id;
    private String firstName, lastName, phone, room;

    public Person(long id, String firstName, String lastName, String phone, String
room) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.phone = phone;
        this.room = room;
    }

    //getters and setters
}
```

Klasa PersonFactory

```
public class PersonFactory {

    public static List<Person> generate() {
        List<Person> list = new LinkedList<>();

        list.add(new Person(0, "Goran", "Delač", "01/6129-549", "D-339-2"));
        list.add(new Person(1, "Marin", "Šilić", "01/6129-549", "D-339-2"));
        list.add(new Person(2, "Marin", "Vuković", "01/6129-658", "C07-04"));
        list.add(new Person(3, "Krešimir", "Pripužić", "01/6129-745", "C08-17"));

        return list;
    }
}
```

Klasa PersonService [1]

```
@Service
public class PersonService {

    private final List<Person> persons = PersonFactory.generate();

    public List<Person> get() {
        Collections.sort(persons, Comparator.comparing(Person::getId));
        return persons;
    }

    public void insert(Person person) {
        persons.add(person);
    }
```

Klasa PersonService [2]

```
public boolean delete(long id) {
    Iterator<Person> iterator = persons.iterator();

    while (iterator.hasNext()) {
        if (iterator.next().getId() == id) {
            iterator.remove();
            return true;
        }
    }

    return false;
}
```

Klasa PersonService [3]

```
    public boolean update(Person person, long id) {
        Iterator<Person> iterator = persons.iterator();
        boolean updated = false;
        while (iterator.hasNext()) {
            if (iterator.next().getId() == id) {
                iterator.remove();
                updated = true;
                break;
            }
        }
        if (updated)
            persons.add(person);

        return updated;
    }
}
```

Klasa PersonController [1]

```
@RestController
public class PersonController {

    @Autowired
    private PersonService personService;

    @GetMapping("/persons")
    public List<Person> get() {
        return personService.get();
    }
```

Klasa PersonController [2]

```
    @PostMapping("/persons")
    public ResponseEntity insert(@RequestBody Person person) {
        personService.insert(person);

        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.LOCATION,
linkTo(methodOn(PersonController.class).get(person.getId())).toStri
ng());
        ResponseEntity response = new ResponseEntity(headers,
HttpStatus.CREATED);
        return response;
    }
```

Klasa PersonController [3]

```
    @GetMapping("/persons/{id}")
    public Person get(@PathVariable("id") long id) {
        Person person = personService.get(id);
        if (person != null) {
            return person;
        } else {
            throw new ResourceNotFoundException("Person
with id " + id + " not found.");
        }
    }
```

Klasa PersonController [4]

```
    @PutMapping("/persons/{id}")
    public ResponseEntity update(@RequestBody Person person,
@PathVariable("id") long id) {
        if (personService.update(person, id)) {
            return ResponseEntity.noContent().build();
        } else {
            return ResponseEntity.status(HttpStatus.NOT_MODIFIED)
                    .eTag("Person with id " + id + " not found.")
                    .build();
        }

    }
```

Klasa PersonController [5]

```
    @DeleteMapping(value = "/persons/{id}")
    public ResponseEntity delete(@PathVariable("id") long
id) {
        if (personService.delete(id)) {
            return ResponseEntity.noContent().build();
        } else {
            throw new ResourceNotFoundException("Person
with id " + id + " not found.");
        }
    }
}
```

Kako vratiti XML reprezentaciju resursa

- U `build.gradle` dodati:

```
dependencies {
    implementation 'com.fasterxml.jackson.dataformat:jackson-dataformat-xml’
}
```

- U klasi `PersonsController` promijeniti bilješku (*annotation*) `@GetMapping`:

```
@GetMapping(value = "/persons", produces = { "application/json", "application/xml" })
//@GetMapping("/persons")
```

- U zaglavlje HTTP zahtjeva dodati: **Accept** **application/xml**

Sigurnost RESTful web-usluga

- Moguće je postići značajnu razinu sigurnosti
  - Korištenje protokola HTTPS umjesto HTTP
  - Korištenje autentifikacije korisnika

- Detalji su izvan opsega ovog predavanja