



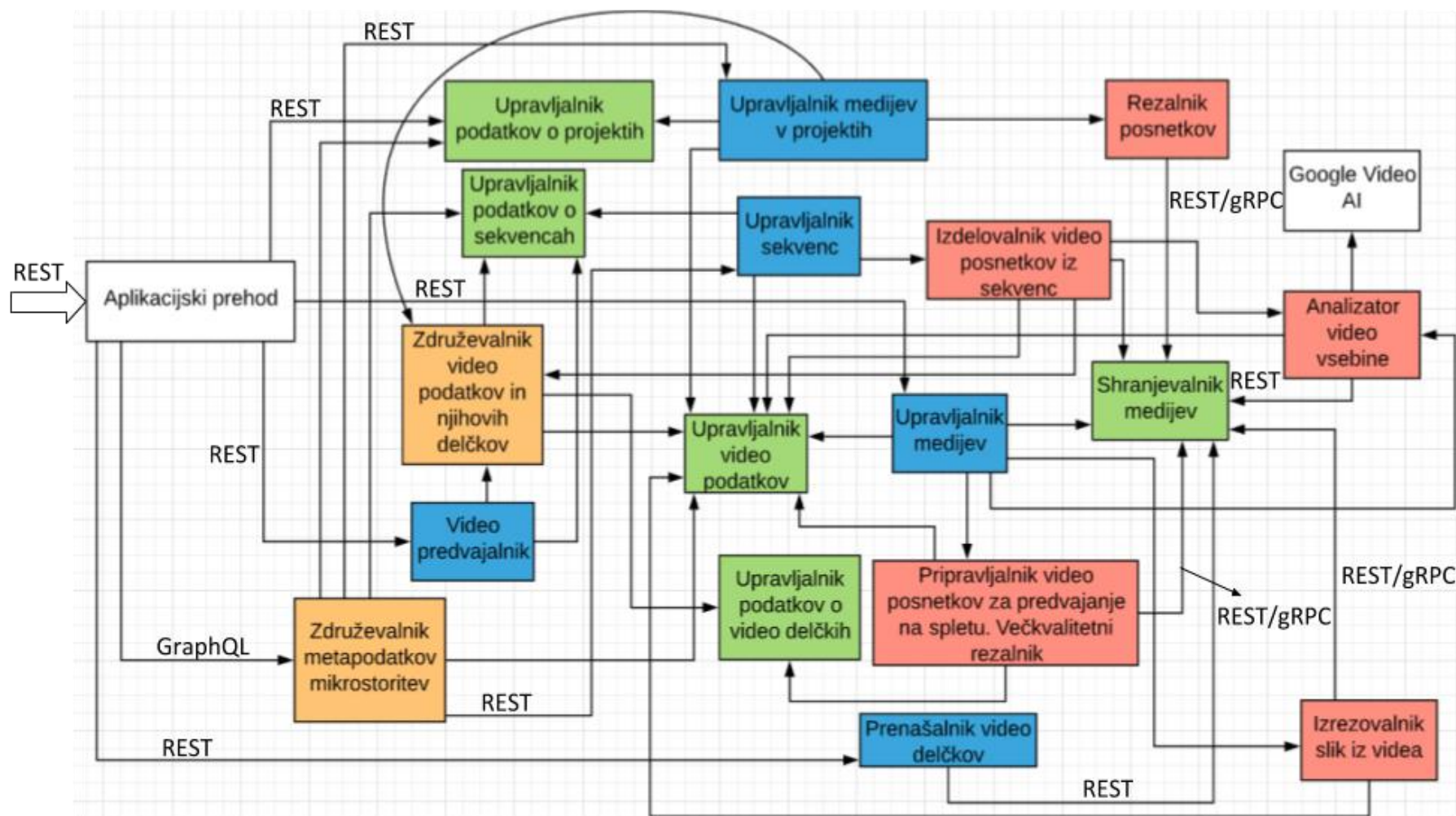
# Spletne storitve REST

# Spletna storitev

---

- Spletne storitve so aplikacijske komponente, ki komunicirajo po odprtih protokolih (predvsem z uporabo HTTP)
- Spletne storitve uporabimo predvsem kadar
  - načrtujemo omrežno interakcijo med raznolikimi avtonomnimi napravami, platformami in programskimi komponentami
  - želimo povezati obstoječe nezdružljive aplikacije na različnih platformah
  - želimo izdelati programske komponente, ki jih bo mogoče večkrat uporabiti

# Primer zaledja iz (mikro)storitev



# Izvedbe spletnih storitev

---

- Spletne storitve ločujemo predvsem glede na način dostopa do njih
- REST (Representational State Transfer)
  - razširljive, vitke spletne storitve, večinoma se uporabljajo za dostop odjemalcev do zaledja
- Klasične spletne storitve (SOAP/WSDL)
  - standardizirane, razširljive, boljša varnost in robustnost, uporabljajo se v porazdeljenih poslovnih aplikacijah
- Poizvedbeni jeziki ([GraphQL](#), Falcor, ...)
  - odjemalec sam pove, katere podatke rabi; uporabni za zmanjšanje kompleksnosti vmesnikov, potrebujejo le eno dostopovno točko (URL) do storitve
- Binarni protokoli ([gRPC](#), ...)
  - učinkoviti, uporabljajo se predvsem za komunikacijo med storitvami v zaledju
- Asinhroni vmesniki ([Websocket](#), MQTT, ...)
  - razrešujejo predvsem probleme protokola HTTP, predvsem sinhronost
- ...

# Arhitekturni vzorec REST

---

- Ne določa podrobnosti ampak le smernice izvedbe spletne storitve
  - delitev dela med *strežnikom in odjemalcem*
    - strežnik skrbi za shranjevanje podatkov (back-end), odjemalec pa za uporabniški vmesnik (front-end)
    - strežnik in odjemalec povezuje le aplikacijski programski vmesnik (API), tako da je razvoj obeh karseda neodvisen
  - programski vmesnik storitve mora biti *enoten in konsistenten* (podobna dejanja – npr. branje ali spreminjanje podatkov – se mora dati izvesti na enak način)
  - *nepovezavna komunikacija*: vsaka zahteva je samozadostna (vsebuje vse informacije za njeno uspešno izvršitev)
    - če potrebujemo sejo (npr. za overjanje), je za njeno vzdrževanje odgovoren odjemalec
  - *predpomnjenje* podatkov na strežniku in/ali odjemalcu, kadarkoli je to mogoče
  - *razslojena strežniška arhitektura* (ločeni aplikacijski, podatkovni, overitveni, ... strežnik)
  - poleg podatkov je mogoče odjemalcem posredovati tudi *programsko kodo*
- V realnosti so nekatere smernice pogosto kršene, a še vedno lahko govorimo o vzorcu REST

# Modeliranje informacij

---

- Osnovni informacijski element v arhitekturi REST je *vir* (angl. *resource*)
- Vir je lahko vse, kar lahko poimenujemo
  - oseba, izdelek, dokument, slika, storitev, ...
  - vir je lahko tudi zbirka virov
- Vsak vir mora imeti
  - globalni enolični identifikator (ID)
  - predstavitev (angl. *resource representation*)
- Predstavitev vira
  - predstavlja stanje vira v nekem trenutku
  - lahko vsebuje podatke o viru, metapodatke in povezave, ki omogočajo prehod v drugo stanje vira
- Formatu predstavitve vira pravimo *medijski tip* (angl. *media type*)
- Z viri manipulirajo odjemalci, strežnik jim mora to omogočiti

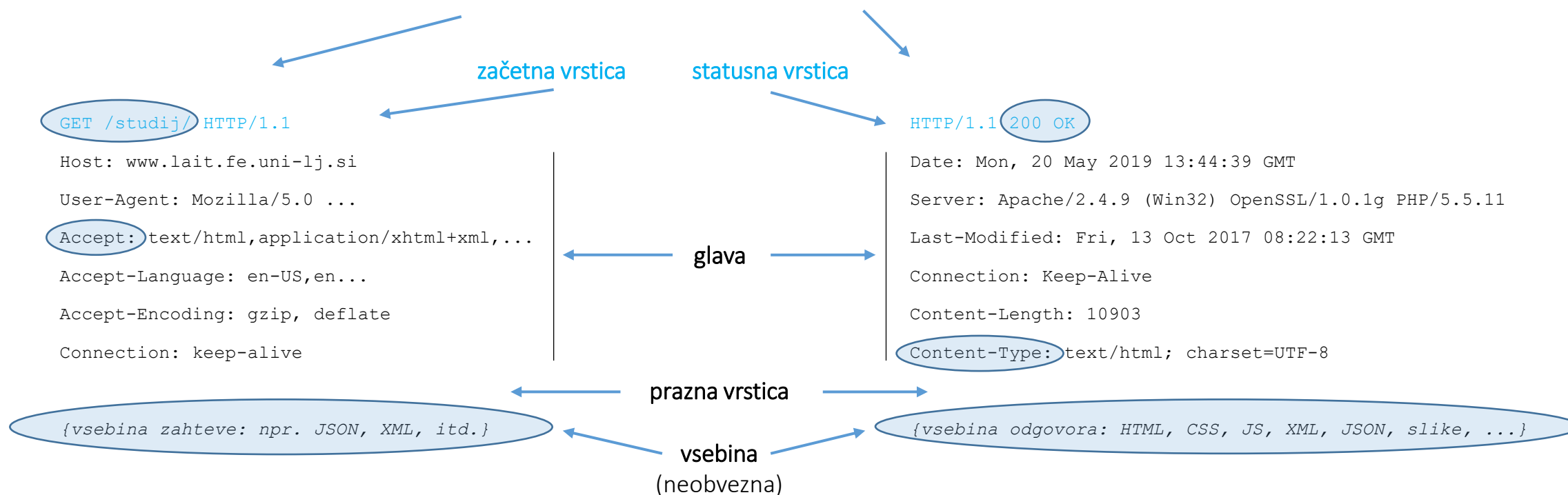
# Izvedba na podlagi protokola HTTP

---

- Spletne storitve REST so v veliki večini izvedene z uporabo protokola HTTP
- Viri so naslovljeni z URL naslovi
- Format predstavitev virov je določen z enim izmed MIME tipov (Multipurpose Internet Mail Extensions)
  - application/json, application/xml, ...
- Odjemalci manipulirajo z viri prek HTTP metod
  - izražajo dejanje, ki naj ga strežnik izvede na izbranem viru
  - GET, POST, PUT, DELETE, PATCH, ...
- Rezultat izvedbe dejanj je podan z enim izmed stanj v HTTP odgovoru
  - 200 (OK), 201 (Created), 204 (No Content), 400 (Bad request), 404 (Not Found), ...

# Protokol HTTP

- Osnovni protokol za komunikacijo med spletnim odjemalcem in strežnikom
- Tipičen protokol konceptov odjemalec-strežnik oz. zahteva-odgovor
  - dve vrsti sporočil: zahteva odjemalca in odgovor strežnika





# HTTP metode

---

## GET

- Zahteva po predstavitvi vira ali zbirke virov, določenih z URL naslovom v začetni vrstici zahteve
- Vsebina zahteve je prazna
- Metoda ne sme spreminjati vira, na katerega se sklicuje zahtevek
- Če vir, na katerega se sklicuje zahtevek, obstaja, strežnik odjemalcu posreduje odgovor s kodo 200 (OK) in predstavitvijo vira
- Če vir ne obstaja, posreduje strežnik odgovor s kodo 404 (Not Found)
- Če odjemalčeva zahteva ni pravilno tvorjena, posreduje strežnik odgovor s kodo 400 (Bad Request)

## POST

- Ustvari nov vir v *zbirki virov*, ki je določena z URL naslovom v začetni vrstici zahteve
- V vsebini zahteve podamo informacijo o MIME tipu priložene predstavitve vira
- Strežnik odjemalcu odgovori s kodo 201 (Created) in URL naslovom v glavi HTTP odgovora (polje Location)

# HTTP metode (2)

---

## PUT

- Nadomesti obstoječi vir, ki je določen z URL naslovom v začetni vrstici zahteve
- V vsebini zahteve podamo novo predstavitev vira v izbranem MIME tipu
- Če vir obstaja, strežnik posreduje odgovor s kodo
  - 200 (OK) in novo predstavitev vira v vsebini odgovora
  - 204 (No Content), če v odgovoru ne pošlje nove predstavitve vira
- Če vir še ne obstaja, lahko strežnik ustvari novega
  - veljajo podobna pravila kot pri POST
  - razlika med POST in PUT: zahtevek s POST se sklicuje na zbirko, zahtevek s PUT pa na nek vir znotraj zbirke

## PATCH

- Podoben kot PUT, le da samo delno posodobi obstoječi vir

## DELETE

- Izbriše vir, določen z URL naslovom v začetni vrstici zahteve
- Strežnik odgovori z 200 (OK), 202 (Accepted) ali 204 (No Content)

# Kode v HTTP odgovoru

---

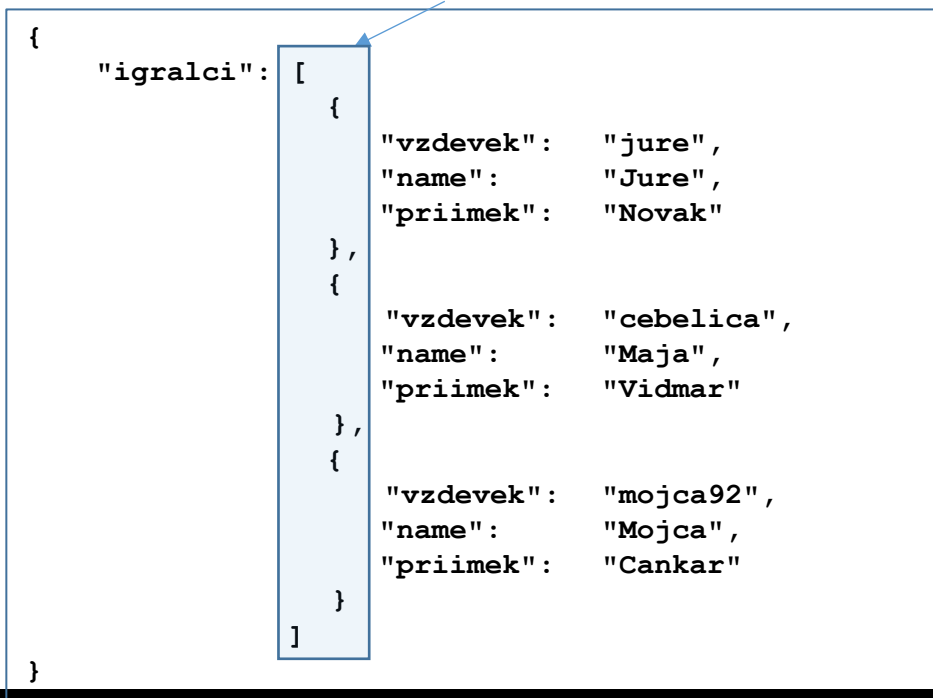
Kategorija	Opis
1xx	Informacije, povezane s komunikacijskim protokolom
2xx	Odjemalčeva zahteva je bila uspešno sprejeta
3xx	Odjemalec mora izvesti dodatna dejanja za dokončanje zahteve
4xx	Napaka odjemalca
5xx	Napaka na strežniku

- Najpogosteje uporabljene kode v storitvah REST
  - **200 (OK)** – Strežnik je uspešno izvedel odjemalčevo zahtevo. Odgovor mora vsebovati vsebino, ki je odvisna od HTTP metode v zahtevi (npr. GET – predstavitev zahtevanega vira, POST – opis ali rezultat dejanja).
  - **201 (Created)** – Strežnik je ustvaril nov vir. V glavi HTTP odgovora posreduje URL novega vira (v polju Location).
  - **204 (No Content)** – Podobno kot 200 (OK), le da odgovor nima vsebine ampak le statusno vrstico in glavo.
  - **400 (Bad Request)** – Splošna oznaka, ki nakazuje, da je odjemalec naredil napako v svoji zahtevi. Pogosto gre za napačno tvorjeno zahtevo ali posredovane nepravilne parametre.
  - **401 (Unauthorized)** – Odjemalec je poskušal izvesti neko dejanje brez predhodne avtorizacije (ali overjanja).
  - **404 (Not Found)** – Vir, na katerega se sklicuje odjemalec, ne obstaja.
  - **405 (Method Not Allowed)** – Uporabljena HTTP metoda pri viru, na katerega se sklicuje odjemalec, ni dovoljena.
  - **409 (Conflict)** – Konflikt s trenutnim stanjem vira, ki ga mora razrešiti odjemalec. Primer uporabe je kršitev tujih ključev v zbirki.
  - **500 (Internal Server Error)** – Splošna oznaka, ko pride do napake pri izvajanju programske kode na strežniku.

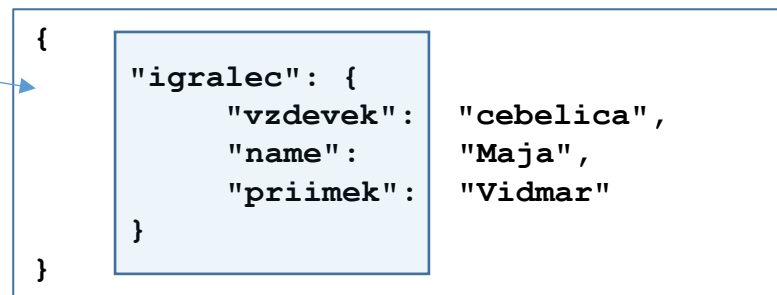
# Predstavitev virov v formatu JSON

- JSON (JavaScript Object Notation) je najbolj razširjen format za izmenjavo podatkov na Spletu
- Jezik sestavljata dve strukturi
  - objekti (zbirke parov ključ – vrednost)
  - polja (seznam vrednosti)

```
{
  "igralci": [
    {
      "vzdevek": "jure",
      "name": "Jure",
      "priimek": "Novak"
    },
    {
      "vzdevek": "cebelica",
      "name": "Maja",
      "priimek": "Vidmar"
    },
    {
      "vzdevek": "mojca92",
      "name": "Mojca",
      "priimek": "Cankar"
    }
  ]
}
```



```
{
  "igralec": {
    "vzdevek": "cebelica",
    "name": "Maja",
    "priimek": "Vidmar"
  }
}
```



# Izdelava programskega vmesnika REST



Izdelajte spletno storitev REST, ki ima sledeči programski vmesnik (API).  
Upoštevajte tudi pravilne vloge HTTP statusnih kod

Pot vstopne točke	Metoda	Medijski tip	Vsebina zahtevka	Opis
/igralci	GET	JSON	/	Pridobi podatke o vseh igralcih
/igralci/{vzdevek}	GET	JSON	/	Pridobi podatke o igralcu s posredovanim vzdevkom
/igralci	POST	JSON	{ "vzdevek": "____", "geslo": "____", "ime": "____", "priimek": "____", "email": "____" }	Doda novega igralca v zbirko igralcev
/igralci/{vzdevek}	PUT	JSON	{ "geslo": "____", "ime": "____", "priimek": "____", "email": "____" }	Posodobi igralca v zbirki
/igralci/{vzdevek}	DELETE	JSON	/	Zbriše igralca v zbirki

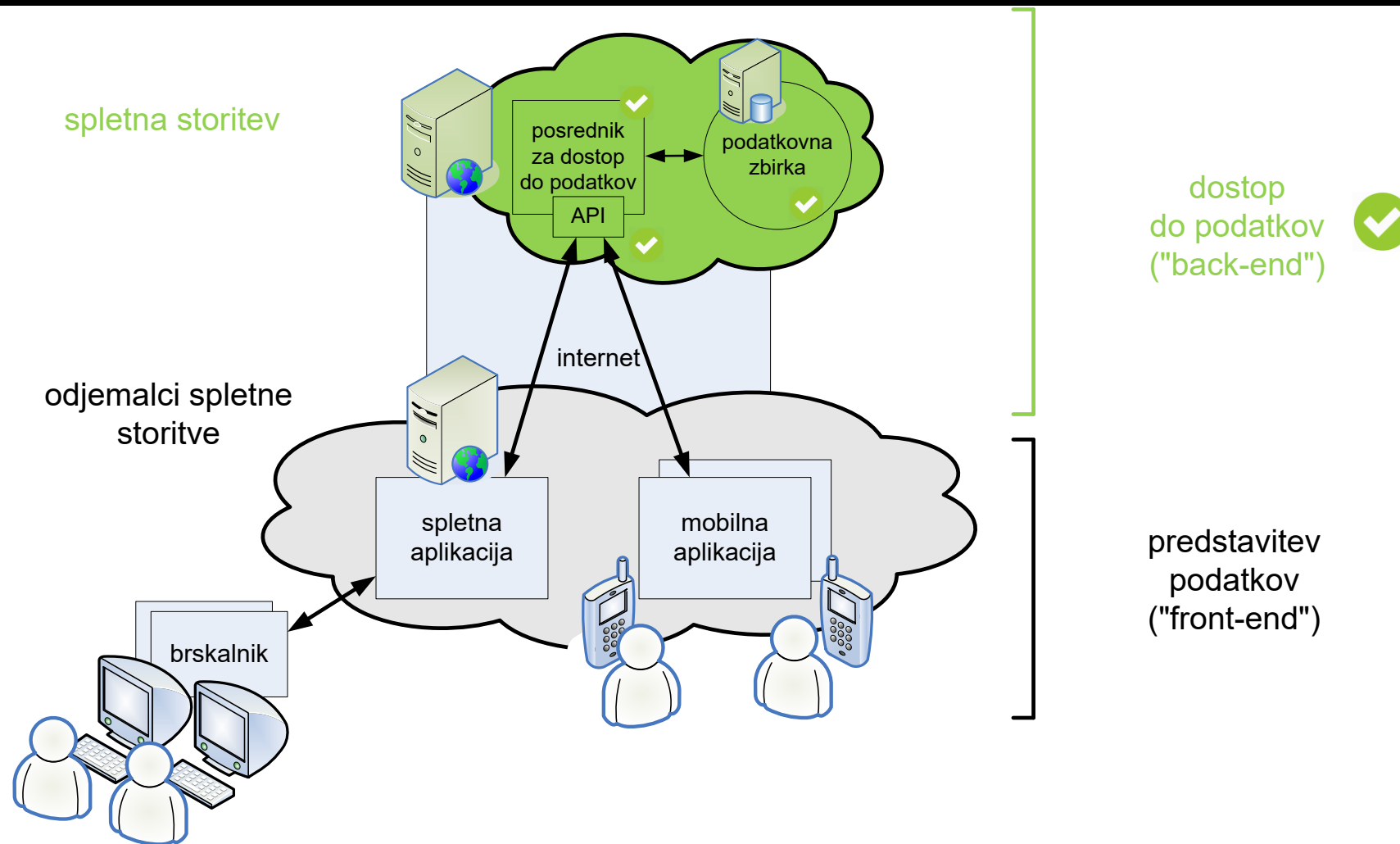
# Izdelava programskega vmesnika REST (2)

---

 Dopolnite izdelano spletno storitev še s sledečo funkcionalnostjo

Pot vstopne točke	Metoda	Medijski tip	Vsebina zahtevka	Opis
/igre/{vzdevek}	GET	JSON	/	Pridobi podatke o odigranih igrah igralca s posredovanim vzdevkom
/igre	POST	JSON	{ "vzdevek": "____", "tezavnost": "____", "rezultat": "____" }	Doda novo igro v zbirko odigranih iger
/lestvice/{tezavnost}	GET	JSON	/	Pridobi lestvico najboljših igralcev na posredovani težavnosti

# Zaledni del je končan



# Poenostavitve in nadgradnje

---

- Da bi izpostavili bistvene stvari, smo se poslužili nekaj poenostavitev
- Pri pridobivanju iger igralca bi bil pravilnejši URL vzorec `igre/?vzdevek=jure` namesto `igre/jure`
  - ker gre v bistvu za filtriranje zbirke virov (iger), 'vzdevek' pa ne določa ene izmed iger ampak enega igralca
- Nismo podrobneje preverjali (validirali) podatkov (dolžine, pravilnega formata e-mail naslova, ...)
  - lahko uporabite orodje [express-validator](#)
- Express-generator ustvari ogrodje v skladnji CJS, modernejša je skladnja ES
  - po ustvarjanju ogrodja je priporočljivo popraviti skladnjo predvsem pri uvozi in izvozi modulov
- Za samodejno pripravo dokumentacije REST vmesnika lahko uporabite orodje [Swagger](#)



# Overjanje

---

- Lastnosti vzorca REST
  - ...
  - *nepovezavna komunikacija*: vsaka zahteva je samozadostna (vsebuje vse informacije za njeno uspešno izvršitev)
    - če potrebujemo sejo (npr. za overjanje), je za njeno vzdrževanje odgovoren odjemalec
  - ...
- Možnosti: ob vsaki zahtevi se predstavimo z
  - uporabniškim imenom in geslom (npr. prek HTTP Basic authentication)
    - ni varno, uporabnik lahko podatke prestreže
  - ključem (angl. *API key*)
    - ključ dobimo od strežnika ob prijavi in je bolj ali manj nespremenljiv
    - lahko se pošilja v glavi HTTP (Bearer authentication) ali pa v telesu HTTP (npr. v obliki JSON)
  - žetonom (angl. *token*)
    - dobimo ga ob prijavi, veljavnost je časovno omejena
    - standardi: JWT, OAuth, ...
  - naprednejši načini ...
- V vsakem primeru je nujno tudi šifriranje podatkov (HTTPS = HTTP + TLS)

# Overjanje z JWT

---

- Priporočam uporabo žetonov [JWT](#) (JSON Web Token)
- Žetoni se lahko pošiljajo v URL naslovu, HTTP glavi ali telesu
- So majhni in samozadostni
- Uporabljajo se za overjanje, avtorizacijo in izmenjavo podatkov
- Če niso šifrirani, lahko vsak, z javnim ključem podpisnika (navadno strežnika), preveri njihovo veljavnost
- Hitra in preprosta implementacija
- Poiščite knjižnico za svojo platformo:
  - „Middleware“ [Passport.js](#) za Node.js
  - za ostale platforme pogledajte [sem](#)

# Ostala ogrodja za spletne storitve REST

---

- **JavaScript:** Express.js, NestJS, Fastify, Koa.js, Hapi.js, ...
- **Python:** Flask, DjangoREST Framework (DRF), FastAPI
- **Java in kotlin:** Spring Boot, Jersey, Jakarta EE / Microprofile
- **PHP:** Laravel, Lumen, Symfony
- **C#:** ASP.NET Core Web API
- **Ruby:** Ruby on Rails

# Domača naloga

---

- Nadgradite zbirko podatkov, ki jo boste uporabili v svoji aplikaciji, s spletno storitvijo

# Viri in literatura

---

- Knjiga Jim Webber, Savas Parastatidis, Ian Robinson: “REST in practice”
- [Learn REST: A RESTful Tutorial](#)
- [REST API Tutorial](#)
- [JSON Web Tokens](#)