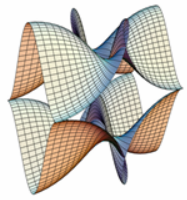


Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Predavanje 12 - REST. Single page applications.

23. siječnja 2026.

Sastavio: Zvonimir Bujanović

REST

- Protokol HTTP je *stateless*.
  - Svaka komunikacija klijenta sa serverom je potpuno nezavisna sa svim prethodnim komunikacijama.
  - Session je mehanizam iznad HTTP-a kojim se na serveru čuva korisnički kontekst (npr. login) ⇝ REST naglašava da svaki zahtjev mora nositi sve što treba za obradu (npr. token).

- REST - Representational State Transfer
  - Arhitekturalni stil za izgradnju mrežnih aplikacija.
  - Stateless, klijent-server, *cacheable* komunikacijski protokol.
  - Koristi razne HTTP zahtjeve (GET, POST, PUT, DELETE) za slanje, čitanje i brisanje podataka.
    ⇝ implementira sve četiri CRUD (Create/Read/Update/Delete) operacije (termin iz baza podataka).
  - Definira sučelje koje se sastoji od: resursa, standardnih HTTP metoda, status kodova, medijskih tipova (JSON).

REST

Komponente i svojstva REST arhitekture:

- REST API se sastoji od mreže resursa. Svaki pojedini resurs identificira jedan URL.

  Primjeri:

  - `http://www.parts-depot.com/parts/` – kolekcija resursa
  - `http://www.parts-depot.com/parts/00345` – jedan resurs

- Pristup resursu korištenjem HTTP metode:
  - GET - Vraća reprezentaciju resursa (ili kolekcije) u odgovarajućem formatu (npr. JSON) [read].
  - PUT - Zamijeni resurs (ili cijelu kolekciju) novim resursom poslanim klijentskim zahtjevom [update].
  - POST - Dodaj novi resurs u kolekciju (obično se ne koristi za cijele kolekcije) [create].
  - DELETE - Obriši resurs (ili cijelu kolekciju) [delete].
  - PATCH - Parcijalni update (umjesto PUT).

REST

- Ne postoji "stanje konekcije" niti nešto slično session-u.
  - Svaka interakcija klijenta sa serverom je posve neovisna o prethodnima.
  - Svaki novi zahtjev klijenta treba sadržavati sve informacije potrebne za njegovo izvršavanje i ne smije ovisiti o prethodnim interakcijama sa serverom.
  - Postoje rješenja kako riješiti autentifikaciju i autorizaciju klijenta (bez da se uvijek isponova šalju username i password).
    - `Authorization: Bearer <token>` (npr. JWT - JSON Web Token).
    - Cookie + CSRF.

- Za API koji implementira ove komponente i svojstva kažemo da je RESTful.

- Primjeri:
  Paypal, X, Google Translate, Flickr, Dropbox, Azure Maps.

- REST API tipično ima rute oblika `/api/v1/...` koje omogućavaju verzioniranje.
- Kod detekcije greške se tipično vraća JSON (`error: '...'`).

REST - Implementacija u Flasku

```
@app.route( '/api/v1/books/<int:id>', methods=['GET'] )
def books_id_get( id ):
    # Napravi upit na bazu i pokušaj dohvatiti info o knjizi s danim id.
    book = BookService.get( id );

    if( book == None ):
        return jsonify( {'error': 'Book not found'} ), 404;
    else:
        return jsonify( book );
```

U REST-u postoje standardizirani HTTP response kodovi:

- 200 - OK
- 201 - Created (uspješan POST + Location header)
- 204 - No Content (uspješan DELETE ili PUT bez tijela odgovora)
- 400 - Bad Request (kriva sintaksa ili druga generička greška)
- 401 - Unauthorized (treba dati autentifikaciju za pristup)
- 403 - Forbidden (zabranjen pristup bez obzira na autentifikaciju)
- 404 - Not Found (resurs ne postoji)
- 405 - Method Not Allowed (metoda nije dozvoljena za resurs)
- 410 - Gone (resurs više ne postoji)
- 429 - Too Many Requests (prijeđen dozvoljeni broj upita)

Single page web-applications (SPA)

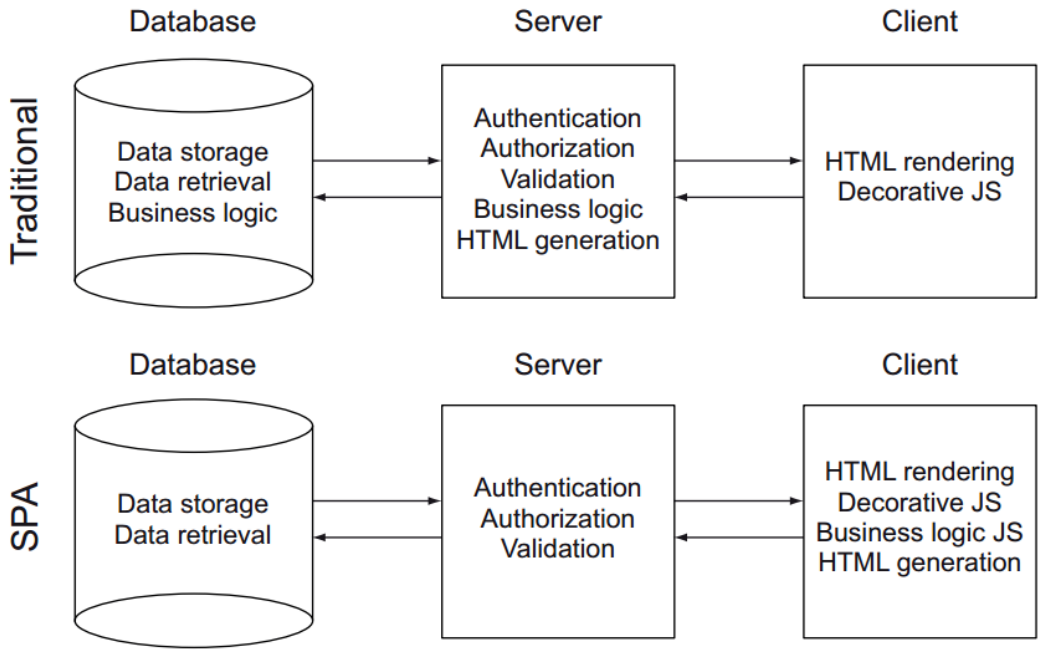
- U posljednje vrijeme je velik trend tranzicije web-aplikacija u model "Single page application".
  - Sav HTML, CSS i JavaScript se učita u prvom pristupu serveru.
  - Ovisno o korisnikovim akcijama, dodatni resursi se dinamički učitavaju i dodaju na stranicu, bez ponovnog učitavanja cijele stranice (npr. Ajax upitom na REST API).
  - Cilj je pružiti dojam tradicionalnih desktop aplikacija.

- U SPA, web-server često služi samo za:
  - Slanje statičkih datoteka (index.html, JavaScript, CSS)
  - REST API (JSON)
  - Autentifikaciju + izdavanje tokena

- Čak se i routing radi na klijentu (History API):
  - SPA može promijeniti URL bez reload-a.
  - Server treba biti podešen da uvijek vrati index.html.

Single page web-applications (SPA)

- U SPA, sva logika web-aplikacije se pomiče sa servera na klijenta [slika iz knjige: Minkowski, Powell: "SPA"]



SPA ⇜⇝ REST API: tko što radi?

1. Browser učita `index.html`, `app.js`, `CSS`.
2. SPA (na klijentu) inicijalizira UI i registrira evente (click, input...).
3. Kao reakciju na događaje radi `fetch('/api/v1/...')`
4. REST API vraća JSON + HTTP status ⇝ SPA ažurira prikaz (bez ponovnog učitavanje stranice).

Podjela odgovornosti:

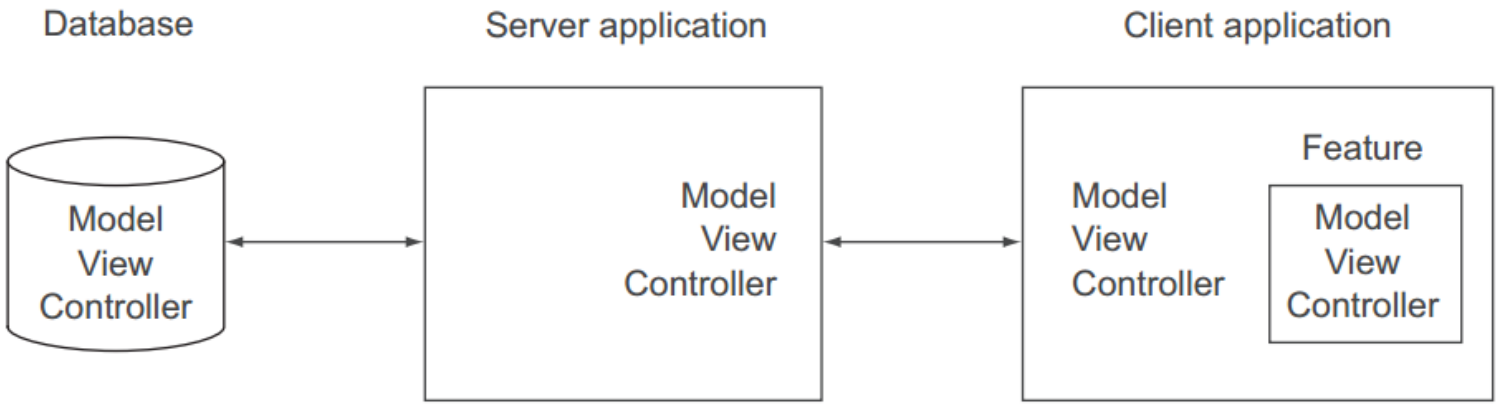
- SPA (klijent): prikaz UI, osvježavanje stanja UI, navigacija/routing, validacija “za korisnika".
- REST API (server): poslovna logika, validacija “za sigurnost”, autentifikacija/autorizacija, rad s bazom.

```
1  // app.js
2  async function getBooks() {
3      let response = await fetch('/api/v1/books');
4      return await response.json();
5  }
6
7  button.addEventListener( 'click', async function() {
8      let books = await getBooks(); // Bez await, getBooks vraća Promise!
9      renderBooks( books );
10 } );
```

Single page web-applications (SPA)

- Povećanje kompleksnosti klijentskog programa zahtijeva bolju organizaciju JavaScript koda.
  - Razdvojiti programsku logiku od dinamičkog generiranja HTML-a (prezentacije).
  - Razdvojiti reakciju na input korisnika od programske logike i HTML-a.
  - Déjà vu?

- MVC i prijatelji (MVVM) na klijentskoj strani.
  - Angular, React, Vue, Svelte.



JavaScript moduli (ESM)

JavaScript moduli omogućavaju organizaciju koda, na primjer:

- `api.js` - Komunikacija s REST API (fetch, status kodovi, JSON).
- `ui.js` - Interakcija sa HTML DOM-om, izgradnja korisničkog sučelja.
- `app.js` - Reakcije na događaje (click/input), spaja API i UI.

Korištenje JavaScript modula:

- HTML: `<script type="module" src="app.js"></script>`
- Unutar modula: `export/import`

```
1  // api.js
2  export async function getBooks() { ... }
3
4  // ui.js
5  export function renderBooks(books) { ... }
6
7  // app.js
8  import { getBooks } from './api.js';
9  import { renderBooks, showError } from './ui.js';
10
11 button.addEventListener('click', async function() {
12     renderBooks( await getBooks() );
13 } );
```

Vue

- vue.js
  - "Bibioteka za izgradnju interaktivnih web-sučelja".
  - Omogućuje jednostavno i reaktivno povezivanje podataka i komponenti sučelja.
  - Fokusira se na View sloj, no u njemu se mogu implementirati čitave SPA u skladu sa obrascem MVVM.
- Napravit ćemo samo dva vrlo jednostavna primjera da pokažemo osnovne koncepte.
- Koristit ćemo Vue bez tzv. *build step-a*, kao običnu JavaScript biblioteku.

- Resursi za učenje:
  - Službena dokumentacija.
  - Izvrsni video-tutorial.
  - Nekoliko praktičnih primjera.
  - w3schools.

Vue - Primjeri

Primjer 1:

- Cilj:
  - Unosom ispravnog imena u input prikazuje se gumb, u protivnom se prikazuje informacija o grešci.
- Ilustrira:
  - Data-binding (reaktivne komponente);
  - Event-handling u vue.js.

Primjer 2:

- Cilj:
  - TODO lista, zadaci se mogu dodavati i označavati kao obavljeni.
- Ilustrira:
  - Iteriranje po listama podataka;
  - Custom komponente;
  - Vezu na REST API u konačnoj verziji.

Što nakon kolegija Razvoj web-aplikacija?

Nadogradnja onoga što već znamo (Flask + JavaScript/REST).

- Autentifikacija i autorizacija (session vs token, role-based pristup).
- Validacija podataka + rukovanje greškama + logging.
- Baza: modeli/relacije, migracije, seed podaci (SQLAlchemy).
- Testiranje (pytest + Flask test client).
- Osnove *deploymenta* (gunicorn + reverse proxy, konfiguracija).

Aktualne web-tehnologije.

- ES moduli + "build tool" (npr. Vite): organizacija koda i tijek razvoja.
- SPA okvir (Vue/React): komponente, routing, state management.
- TypeScript: sigurniji JavaScript za veće projekte.
- Server Side Rendering/Generation (npr. Next.js).
- Real-time (WebSockets): chat, multiplayer...
- Docker + CI/CD (GitHub Actions): proces isporuke aplikacije.
- Drugi *backend* sustavi: Laravel (PHP), Django, FastAPI (Python), Blazor (C#), Java Spring, Node.js/Express ...
- Kolegij "Razvoj grafičkih korisničkih sučelja".