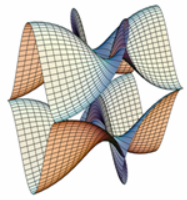


Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Predavanje 07 - Model-View-Controller. Object-relational mapping.

7. studenog 2025.

Sastavio: Zvonimir Bujanović

Model-View-Controller

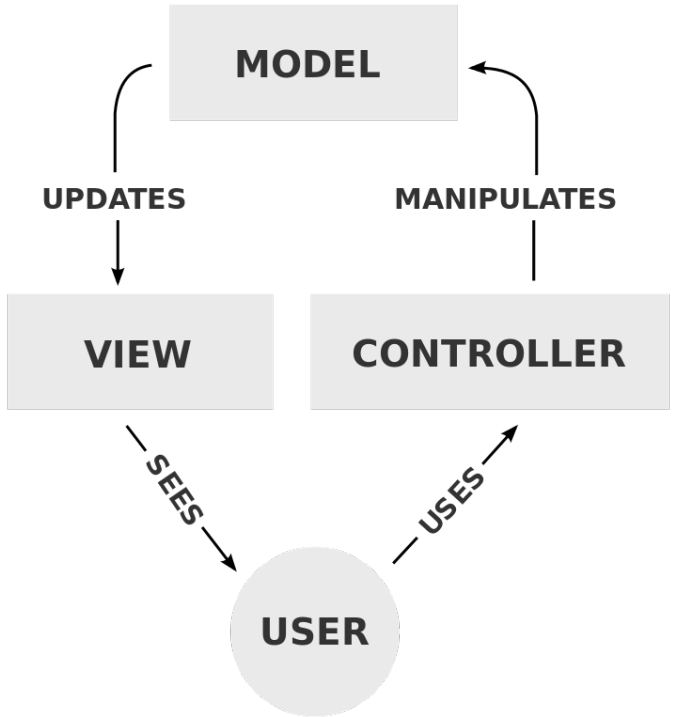
Problemi s dosadašnjim pristupom pisanju Flask aplikacija:

- Obrađivanje korisničkog ulaza (`GET`, `POST`...) je ispremiješano s logikom aplikacije.
- Teško je snaći se u kodu ako postoji puno ruta.
- Nemamo sustavnu strategiju proširivanja funkcionalnosti aplikacije (rute za korisnike, rute za artikle, rute za kupovinu...)
- Sve u jednoj datoteci: vrlo neprikladno za složenije aplikacije i više developera.

Ciljevi:

- *Separation of concerns*.
  - HTML je već potpuno odvojen od Python-a!
  - Obradu korisničkog ulaza treba potpuno odvojiti od logike.
- Omogućiti istovremen razvoj više developera na proširivanju funkcionalnosti aplikacije.

Model-View-Controller



- MVC je "arhitekturalni obrazac (*pattern*)" originalno namijenjen za implementaciju korisničkog sučelja (GUI).
- Prilagođen je i za protok podataka u web-aplikacijama.
- Brojni razvojni okviri (Django, Laravel, Symfony, Ruby on Rails, Java Spring MVC, . . .) su bazirani na MVC.

Model-View-Controller

- Mi ćemo ovdje izvesti od nule jednu moguću varijantu MVC frameworka.
- Za svaku iduću aplikaciju sve ove korake neće trebati raditi od početka. ⇝ jednostavno možemo uzeti gotovo konačno rješenje Zadatka 5 i samo prilagoditi sadržaje direktorija models/templates/controllers novoj aplikaciji.

Zadatak

Napišite web-aplikaciju koja simulira rad knjižnice.

- U bazi postoje tablice `users` (za svakog korisnika: `id`, `name`, `surname`), `books` (za svaku knjigu: `id`, `author`, `title`), `loans` (za svaku posudbu: `id`, `id_user`, `id_book`, `lease_end`).
- Omogućite ispis svih korisnika; za svakog korisnika možemo dobiti i popis knjiga koje je on posudio.
- Omogućite ispis svih knjiga u knjižnici; za svaku neka bude vidljivo je li i do kad je posuđena.

Bonus:

- Omogućite posuđivanje knjiga – za odabranog korisnika, napravite popis svih knjiga koje nisu posuđene, te mu omogućite da ih posudi.
- Omogućite vraćanje knjiga – za odabranog korisnika, uz popis knjiga koje je posudio stavite gumb za vraćanje knjige.
- Svaki korisnik ima i lozinku. Tek kad unese lozinku, može vidjeti svoje knjige, te ih razduživati i posuđivati nove.

Model

- Predstavlja logiku (web-)aplikacije i pripadne strukture podataka.
- Ovaj sloj je potpuno odvojiv i smislen i bez web-a.
- Isti model obično koristi više raznih pod-stranica iste web-aplikacije.
- Model obično implementira 3 aspekta:
  - Strukture podataka koje predstavljaju objekte s kojima se manipulira u aplikaciji (npr. `User`, `NewsArticle`, ...).
  - Načine za dohvaćanje tih podataka ("data mapper" – kako dohvatiti podatke iz baze, cache-a, diska...).
  - Sučelje (servis) za sve koji te podatke trebaju (“business logic”).
- Stoga, model:
  - Nema nikakvu interakciju s korisnikom aplikacije.
  - Ne koristi `GET`, `POST` i slične.
  - Ne sadrži HTML niti ikakav drugi prezentacijski kod.

Zadatak 1

- Napravite direktorij `library`.
- Skinite i u direktorij `library` odzipajte `prepare_db.zip`.
  - Pripremite datoteku `app.config`. Prilagodite podatke za spajanje na bazu.
  - Pokrenite aplikaciju i pristupite rutama `/create_tables` i `/seed_tables`. Ovo će napraviti tablice u bazi i popuniti ih nekim probnim podacima.
- U direktoriju napravite poddirektorije `models`, `controllers`, `templates`.
- U poddirektoriju `models` napravite datoteke:
  - `user.py` – klasa `User` koja ima članove `id`, `name`, `surname`, `password`, te konstruktor i metodu `__repr__`.
  - `libraryservice.py` – klasa `LibraryService` koja ima samo statičku funkciju `get_all_users()` koja vraća polje svih korisnika iz baze podataka.
- Testirajte `get_all_users` dodavanjem rute `/test_get_all_users`.

View

- Predstavlja sloj za vizualizaciju modela u nekom pogodnom formatu. U web-aplikacijama taj format je najčešće HTML, ali može biti i neki drugi (npr. JSON – kasnije).
- Tipično će svaka podstranica u web-aplikaciji biti predstavljena jednim view-om.
- Moguće je više puta koristiti iste view-ove za neke tipične radnje: ispis headera, menija, footera, formi koje se pojavljuju više puta i slično.
- Stoga, view:
  - Treba sadržavati skoro isključivo HTML.
  - Koristi Python na vrlo trivijalnoj razini – samo za ispis varijabli, ili iteraciju po polju za ispis varijabli.
  - Ne pristupa bazi podataka, datotekama na disku i slično.
  - Ako uopće pristupa modelu (što se može u potpunosti izbjeći), onda koristi isključivo read-only pristup – ne izmjenjuje model.
  - Služi samo za dohvaćanje i prikazivanje gotovih podataka. Ne radi nikakvu obradu.
  - Ne koristi `GET`, `POST` i slične.
- Jasno: u Flasku je svaki view zapravo Jinja template.

Jinja predlošci: blokovi

Jinja predlošci se mogu “include-ati” jedni u druge.

`layout.html`

```
<h2>Bok, {{ ime }}</h2>
{% block navigacija %}{% endblock %}
{% block sadržaj %}{% endblock %}
<h6>(c) PMF Zagreb
```

`index.html` ⇝ Efekt je sljedeći: kopira se sve iz `__layout.html` i tamo se uvrsti sadržaj blokova kako je definiran u `index.html`.

```
{% extends 'layout.html' %}

{% block navigacija %}
    <a href="/">Početna stranica</a>
{% endblock %}

{% block sadržaj %}
    <p>Ovo ide u sadržaj: {tekst}</p>
{% endblock %}
```

`app.py`

```
return render_template( 'index.html', ime='Mirko', tekst='Pozdrav od Mirka' );
```

Zadatak 2

U poddirektoriju `templates` napravite sljedeće datoteke:

- `_layout.html`
  - Sadrži html deklaraciju, do uključivo `<body>`.
  - Ispisuje naslov web-stranice, pretpostavite da je dostupan u varijabli `title`.
  - Slijedi Jinja blok naziva (npr.) `contents`.
  - Nakon toga ide `</body></html>`.
- `users_index.html`
  - Proširuje predložak `_layout.html`.
  - U Jinja blok-u `contents`:
    - Ispisuje u tablici sva imena i prezimena svih korisnika.
    - Pretpostavite da podaci o korisnicima dani u polju `users` objekata tipa `User`.

Provjerite radi li ispravno prikaz `users_index.html` dodavanjem testne rute.

Controller

- Obrađuje podatke poslane od strane korisnika, na temelju njih update-a model.
- Radi upit na model, na temelju odgovora priprema podatke koje će prikazati view.
- Dakle, controller je "ljepilo" koje povezuje model (aplikacijsku logiku) i view (prezentaciju podatka).

- Stoga, controller:
  - Pristupa `GET`, `POST` i ostalim podacima koje je poslao korisnik.
  - Stvara i koristi servise iz modela – pomoću njih može i dohvaćati podatke iz modela i mijenjati ih.
  - Ne radi direktno upite na bazu, nego koristi servise/data mappere iz modela.
  - Ne sadrži HTML niti ikakav drugi prezentacijski kod.
  - Na temelju podataka dobivenih od korisnika, odabire prikladni view i prikazuje ga.
- Vrlo često postoji 1-1 korespondencija između controllera i viewa.

Zadatak 3

U poddirektoriju `controllers` napravite datoteku `users_controller.py` u kojoj:

- Definirajte klasu `UsersController`.
- Klasa zasad ima jednu javno dostupnu funkciju `index` koja:
  - Dohvaća popis svih korisnika knjižnice koristeći `LibraryService` u listu `users`.
  - Popunjava varijablu `title` (sami odaberite što).
  - Renderira view/predložak `users_index.html` s gornjim podacima.

Provjerite radi li ispravno funkcija u kontroleru dodavanjem testne rute.

Router

Prilagodit ćemo ruter na sljedeći način:

- Ako korisnik pristupi ruti `/con/action`.
  onda ćemo pozvati funkciju `action` iz kontrolera `conController`.

- Ako korisnik pristupi ruti `/con`,
  onda ćemo pozvati funkciju `index` iz kontrolera `conController`.

- Dakle, ako se pristupa
  `/users/index`, ili
  `/users`,
  pozivamo funkciju `index` iz kontrolera `UsersController`.

Zadatak 4

Implementirajte usmjeravanje tako da radi kao na prethodnoj stranici.

- Preddefinirajte popis svih ruta (kontrolera i akcija) koje smiju biti posjećene.

Provjerite radi li ispravno prikaz svih korisnika knjižnice pristupom na `/users` i `/users/index`.

Zadatak 4 - Rješenje

```
1  from flask import abort;
2  import importlib;
3
4  # Popis dozvoljenih kontrolera i za svaki od njih dozvoljenih akcija.
5  ALLOWED_ROUTES = {
6      'users': ['index']
7  };
8
9  @app.route( '/<controller>/', defaults={'action': 'index'} )
10 @app.route( '/<controller>/<action>', methods=['GET', 'POST'] )
11 def dispatch( controller, action ):
12     if( controller not in ALLOWED_ROUTES
13             or action not in ALLOWED_ROUTES[controller] ):
14         abort( 404, f'Unknown controller {controller} and/or action {action}.' );
15
16     try:
17         # Dinamički importamo modul u kojem će se nalaziti odgovarajući kontroler.
18         module = importlib.import_module( f'controllers.{controller}_controller' )
19
20         # Odredimo ime klase traženog kontrolera.
21         controller_classname = f'{controller.capitalize()}Controller';
22         controller_class = getattr( module, controller_classname );
23
24         ...
```

Zadatak 4 - Rješenje

```
1         ...
2
3         # Instanciramo objekt pomoću klase spremljene u varijablu (!).
4         controller_instance = controller_class();
5
6         # Dohvatimo metodu (akciju) traženog imena iz tog objekta.
7         action_handle = getattr( controller_instance, action );
8
9         # Napokon, pozovemo tu metodu.
10        return action_handle();
11
12    except Exception as e:
13        abort( 500, str(e) );
```

Zadatak 5

- Dodajte controller `BooksController` i pripadne akcije:
  - `index` – za prikaz svih knjiga u knjižnici.
  - `search` – za pretraživanje knjiga po imenu autora.
  - `search_results` – za prikaz rezultata pretrage.

  Prilagodite, tj. proširite model.
- Dodajte controller `_404Controller` i pripadni view za prikaz nepostojećih stranica (tj. parova (controller, action)).
- U `templates/_layout.html` dodajte prikaz menija sa ponuđenim opcijama ispisa svih korisnika, knjiga, te pretraživanja knjiga.

Dodatno,

- Rute za pripremu baze prebacite na `/db/action`.
- Testne rute prebacite na `/test/action`.
- Ove rute bismo u praksi zaštitili da im može pristupiti samo admin, te da nisu dostupne u produkcijskoj verziji.

Zadatak 6

- U controlleru `Users`:
  - Modificirajte akciju `index` tako da kraj svakog korisnika u tablici možemo kliknuti na link "Prikaži posuđene knjige".
  - Klik na bilo koji od tih linkova vodi na `/users/show_loans` koji prikazuje u tablici sve knjige koje je kliknuti korisnik posudio, zajedno s datumom isteka posudbe.
- Dovršite sve ostale podzadatke početnog Zadatka.

Razmislite kako direktno u `app.py` ugraditi zaštitu nekih ruta da im ne može pristupiti neulogirani korisnik.

Flask Blueprints

⚠ Oprez

Flask ima vlastiti mehanizam za organizaciju složenih aplikacija: Blueprints.

- Pojedini *blueprint* implementira neku funkcionalnost (npr. autentifikaciju) tako da implementira neki zaokruženi skup ruta.
- Svaki blueprint je donekle sličan zasebnoj Flask aplikaciji.
- Blueprinti su onda komponente od kojih se onda modularno slaže aplikacija.
- Ponekad se isti blueprint može iskoristiti za više aplikacija.

Ovakav sustav ima drugačiju logiku organizacije od MVC.

- No postoji paralela između ruta koje pripadaju jednom kontroleru i jednog blueprinta.

Object-relationship mapping (ORM)

Tipično, podatke o svakoj pojedinoj klasi iz modela čuvamo u njezinoj pripadnoj tablici u bazi.

- Operacije poput "dohvati sve objekte neke klase" (knjige, korisnike), "dohvati objekt sa zadanim id-om", "dohvati sve objekte čiji zadani atribut ima zadanu vrijednost" želimo provoditi nad svim klasama iz modela.
- Moguće je implementirati automatsku konverziju između objekata neke klase i redaka u pripadnoj tablici iz baze podataka $\rightsquigarrow$ objektno-relacijsko preslikavanje (ORM).
- Standardno se to radi tako da su klase iz modela izvedene iz bazne, apstraktne klase `Model` u kojoj su implementirane sve funkcije za dohvaćanje i spremanje objekata u bazu.
- Dobra implementacija klase `Model` nije trivijalna.
- Na sljedećim slide-ovima pokazujemo primjer korištenja ORM paketa SQLAlchemy za Python, tj. Flaskovog paketa Flask-SQLAlchemy.

Object-relationship mapping (ORM) - Primjer

Funkcije `all` i `filter` su iz bazne klase `db.Model` i ne treba ih ponovno implementirati u klasama `User` i `Book`.

- Možemo u potpunosti izbjeći pisanje SQL upita!
- SQL upiti su skriveni od nas u klasi `db.Model`.

```
1  class User( db.Model ):
2      # Povezujemo klasu User s tablicom 'users' u bazi podataka
3      __tablename__ = 'users';
4
5      # Opis svakog stupca.
6      id = db.Column( db.Integer, primary_key=True );
7      name = db.Column( db.String(50) );
8      surname = db.Column( db.String(50) );
9      password = db.Column( db.String(255) );
10
11 # Dohvati sve korisnike iz baze u listu objekata tipa User.
12 users = User.query.all();
13
14 # Dohvati sve knjige autora "Asimov, Isaac"
15 books = Book.query.filter( Book.author == 'Asimov, Isaac' ).all();
```

Object-relationship mapping (ORM) - Primjer

Moguće je jednostavno opisati i odnose među klasama. Na primjer:

```
class Loan( db.Model ):
    __tablename__ = 'loans';

    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), primary_key=True);
    book_id = db.Column(db.Integer, db.ForeignKey('books.id'), primary_key=True);
    lease_end = db.Column( db.Date );

    # Za direktno dohvaćanje user-a koji je obavio tu posudbu.
    user = db.relationship( 'User' );

# Dohvati posudbu s id-om 1. Može i filter kao ranije.
loan = Loan.query.get( 1 );

# Ispiši ime korisnika koji je napravio tu posudbu.
print( loan.user.name );
```

Lako je realizirati i dvostrane veze, bez obzira jesu li 1-1, 1-N, N-N.
Na primjer, `user.loans` bi mogao vratiti sve posudbe korisnika `user`.