Objektno orijentirano programiranje - praktikum u .NET okolini

Predavanje 03

Arhitektura web servisa (API)

- Korisnik **šalje upit** (engl. *request*) prema poslužitelju putem:
  - Grafičkog sučelja (desktop, web ili mobilna aplikacija)
  - Sučelja naredbenog retka (engl. c*ommand-line interface*)

- Poslužitelj **šalje odgovor** (engl. *response*) na upit koristeći neki izvor podataka (bazu podataka ili pozadinsku uslugu, tj. servis)
  - Odgovor postaje vidljiv korisniku preko korištenog sučelja

Tipovi web servisa

- **REST** (Representational State Transfer)
  - Koristi standardni **HTTP** protokol
  - Dozvoljava različite formate podataka (preferirano **JSON**)

- **SOAP** (Simple Object Access Protocol)
  - Koristi **XML** kao format podataka
  - Standardni protokol za razmjenu poruka (lošije performanse i veća kompleksnost od REST-a, ali veća sigurnost)

REST

- Osigurava interoperabilnost na Internetu (**RESTful API**)
  - Različiti tipovi aplikacija mogu međusobno komunicirati
- Koristi HTTP glagole za kreiranje upita:
  - **POST** – kreiranje novog resursa (**C**reate)
  - **GET** – dohvaćanje resursa (**R**ead)
  - **PUT** ili **PATCH** – ažuriranje resursa (**U**pdate)
  - **DELETE** – brisanje resursa (**D**elete)
- Koristi predefinirane **„stateless”** operacije
  - Svaki HTTP upit je izoliran (ne pamti stanje)

Primjer RESTful API upita i odgovara

- **Upit**:
  `GET /api/users`

- **Odgovor**:
  `200 OK`

```
{ "data": [
    {
      "id": 1,
      "email": "pero@mail.com",
      "first_name": "Pero",
      "last_name": "Peric"
    }
]}
```

Korištenje RESTful API-ja u jeziku C#

- Postoji više različitih C# biblioteka zaduženih za korištenje RESTful API-ja, a neke od poznatijih su:
  - HttpWebRequest
  - WebClient
  - HttpClient
  - **RestSharp**
  - ServiceStack
  - Flurl

Mapiranje JSON objekata na C# modele (1/2)

JSON objekt

```
{
   "id": 1,
   "email": "pero@mail.com",
   "first_name": "Pero",
   "last_name": "Peric"
}
```

C# model

```
class User
{
   public int Id { get; set; }
   public string Email { get; set; }
   public string FirstName { get; set; }
   public string LastName { get; set; }
}
```

Mapiranje JSON objekata na C# modele (2/2)

- Postoji više različitih C# biblioteka zaduženih za mapiranje JSON objekata na C# modele, a neke od poznatijih su:
  - **Newtonsoft.Json**
  - AutoMapper

Asinkrono dohvaćanje podataka

Sinkroni rad

- Svaki zadatak mora biti završen kako bi idući mogao započeti
- Zadatak se izvršava na jednoj dretvi (engl. *thread*)

Asinkroni rad

- Idući zadatak može započeti za vrijeme izvršavanja postojećeg zadatka
- Zadatak se može izvršavati na više dretvi istovremeno

Prednosti asinkronog rada

- **Više zadataka** može se izvršavati **istovremeno** što generalno rezultira **boljim performansama**
  - Performanse su definitivno bolje ako se zadaci izvršavaju na računalu koje ima više procesorskih dretvi i/ili jezgri, ili više procesora
  - Ako raspolažamo s jednom procerskom dretvom, potrebno je raditi mijenjanje konteksta (engl. *context switching*) što usporava ukupne performanse
- U slučaju aplikacija s grafičkim sučeljem (npr. Windows Forms), asinkronost omogućuje održavanje **responzivnosti aplikacije**
  - Moguće obraditi događaj (npr. Click) za vrijeme dohvata podataka

Asinkroni rad u jeziku C# (generalno)

- Koncept se temelji na korištenju klase **Task**
  - Omogućuje apstrakciju pisanja asinkronog kôda
  - Task može biti omotač (engl. *wrapper*) oko bilo kojeg tipa podatka

- Korištenje ključnih riječi **async** i **await**
  - **async** definira asinkronu metodu (izvršava se u zasebnoj dretvi)
  - **await** definira operator koji čeka da se asinkrona metoda izvrši, i potom dohvaća podatak koji smo „omotali” u Task
    - `await` pokreće asinkronu metodu tako da ne blokira dretvu iz koje je pozvan
    - Ako ne koristimo `await`, izvršavanje programa se nastavlja nakon poziva asinkrone metode (nema čekanja rezultata)

Asinkroni rad u jeziku C# (specifično za aplikacije s grafičkim sučeljem)

- Korištenje kontrole **BackgroundWorker**
  - Događaji definirani na kontroli:
    - **DoWork** (pozadinska dretva)
      - Definira zadatak koji se izvodi na pozadinskoj dretvi
      - Započinje pozivom metode **RunWorkerAsync** na instanci BackgroundWorkera
    - **ProgressChanged** (glavna dretva)
      - Definira promjenu u zadatku koji se izvodi na pozadinskoj dretvi
      - Započinje pozivom metode **ReportProgress** na instanci BackgroundWorkera
    - **RunWorkerCompleted** (glavna dretva)
      - Definira dovršen zadatak koji se izvodio na pozadinskoj dretvi
      - Zadatak može biti uspješan, neuspješan, a može biti i otkazan (engl. *cancel*) postavljanjem svojstva **Cancel** (definirano na **DoWorkEventArgs**) na vrijednost **true**

Usporedba asinkronog rada u jeziku C#

async / await

- Jednostavniji rad ako je potrebno samo odraditi zadatak na pozadinskoj dretvi
- Bolje performanse

BackgroundWorker

- Ugrađeni mehanizam objavljivanja promjena u izvršavanju zadatka
- Ugrađeni mehanizam otkazivanja započetog zadatka