

Razvoj softvera 2 – Beleške za vežbe

Nikola Ajzenhamer

Sadržaj

1. UVOD U MIKROSERVISNE APLIKACIJE	2
1. O KURSU	2
2. O MIKROSERVISNIM APLIKACIJAMA	2
3. RAZVOJ CATALOG.API MIKROSERVISA	2
PRIPREMA MONGODB KONTEJNERA	2
KREIRANJE PROJEKTA I INSTALIRANJE PAKETA	3
RAZVOJ MIKROSERVISA	4
POKRETANJE I DEBAGIRANJE APLIKACIJE	5
2. VIŠESTRUKI MIKROSERVISI. KONTEJNERIZACIJA APLIKACIJE.	6
1. RAZVOJ BASKET.API MIKROSERVISA	6
PRIPREMA REDIS KONTEJNERA	6
KREIRANJE PROJEKTA I INSTALIRANJE PAKETA	6
RAZVOJ MIKROSERVISA	7
POKRETANJE I DEBAGIRANJE APLIKACIJE	7
2. KONTEJNERIZACIJA PROJEKATA	8
POKRETANJE PROJEKTA IZ KOMANDNE LINIJE	9
POKRETANJE PROJEKTA IZ VISUAL STUDIO ALATA	10
NEKE NAPOMENE	11
DOCKER DESKTOP	11
3. SINHRONA KOMUNIKACIJA IZMEĐU MIKROSERVISA POMOĆU GRPC PROTOKOLA	13
1. PRIPREMA POSTGRESQL I PGADMIN KONTEJNERA	13
ALAT PGADMIN4	14
2. KREIRANJE ZAJEDNIČKOG PROJEKTA ZA API I GRPC PROJEKTE	15
KREIRANJE PROJEKTA I INSTALACIJA PAKETA	15
RAZVOJ BIBLIOTEKE KLASA ZA DISCOUNT PROJEKTE	15
3. KREIRANJE API PROJEKTA	16
RAZVOJ MIKROSERVISA	16
POKRETANJE I DEBAGIRANJE APLIKACIJE	17
4. KREIRANJE GRPC PROJEKTA	17
RAZVOJ MIKROSERVISA	18
POKRETANJE I DEBAGIRANJE APLIKACIJE	19
5. KORIŠĆENJE GRPC PROJEKTA U BASKET MIKROSERVISU	20

1. Uvod u mikroservisne aplikacije

Tema ovih časova je upoznavanje studenata sa planom kursa i obavezama na kursu, kao i razvijanje jednostavnog mikroservisa sa MongoDB SUBP.

1. O kursu

- Sajt kursa: <http://rs2.matf.bg.ac.rs/>
- Prezentacija: <http://rs2.matf.bg.ac.rs/vezbe/o-kursu.pdf>
- O seminarским radovima: <http://rs2.matf.bg.ac.rs/seminarski-radovi/>
- Neophodni alati:
 - Visual Studio (Windows), Visual Studio for Mac (OSX), Visual Studio Code (Windows, OSX, Linux), JetBrains Rider (Windows, OSX, Linux)
 - .NET 5 (Windows, OSX, Linux)
 - Docker Desktop (Windows, OSX), Docker Server (Linux)

2. O mikroservisnim aplikacijama

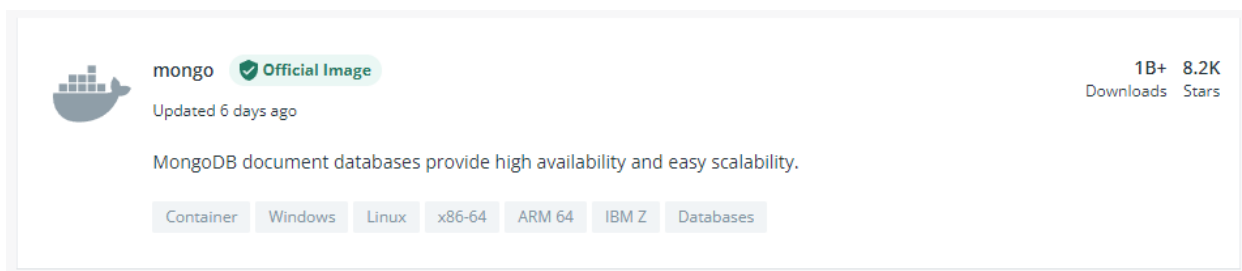
- Ukratko proći kroz tekst: <http://rs2.matf.bg.ac.rs/vezbe/ukratko-o-mikroservisima.pdf>

3. Razvoj Catalog.API mikroservisa

- Priprema MongoDB kontejnera
- Kreiranje projekta i instalacija paketa
- Razvoj mikroservisa
- Pokretanje i debugiranje aplikacije

Priprema MongoDB kontejnera

Na stranici <https://hub.docker.com/> uneti „mongo“ u polje za pretragu. Otvoriti sledeću stranicu:



Osnovni MongoDB pojmovi: baza dokumenata, dokumenti, jedinstveni ključevi (_id), kolekcije, indeksi (nad _id).

Pokretanje mongo kontejnera:

- Otvoriti Powershell
- **docker run --name mongo_catalog -p 27017:27017 -d mongo**

Ove naredbe ispisuju identifikator kontejnera u kojem je pokrenut MongoDB SUBP. Korisne docker naredbe:

- Ispisuje sve pokrenute kontejnere
 - **docker ps**
- Ispisuje sve kontejnere
 - **docker ps -a**
- Ispisuje samo identifikatore pokrenutih kontejnera

- **docker ps -q**
- Pokreće ugašeni kontejner
 - **docker start IDENTIFIKATOR_KONTEJNERA**
- Zaustavlja pokrenuti kontejner
 - **docker stop IDENTIFIKATOR_KONTEJNERA**
- Uklanja kontejner
 - **docker rm IDENTIFIKATOR_KONTEJNERA**
- Uklanja sve kontejnere sa sistema
 - **docker rm \$(docker ps -aq)**
- Kreira novi kontejner na osnovu slike čiji je naziv **IME_SLIKE** na hub.docker.com i pokreće ga
 - **docker run IME_SLIKE**
- Kreira novi kontejner na osnovu slike i pokreće ga, pri čemu mu daje ime **IME_KONTEJNERA**
 - **docker run --name IME_KONTEJNERA IME_SLIKE**
- Kreira novi kontejner na osnovu slike i pokreće ga, pri čemu se vrši preslikavanje portova, tako što se **UNUTRAŠNJI_PORT** preslikava na **SPOLJNI_PORT** u localhost-u
 - **docker run -p SPOLJNI_PORT:UNUTRAŠNJI_PORT IME_SLIKE**
- Kreira novi kontejner na osnovu slike i pokreće ga, ali u pozadini (terminal se ne blokira)
 - **docker run -d IME_SLIKE**
- Čita (i prati, ako se navede opcija -f) sve logove za pokrenuti kontejner
 - **docker logs -f IME_KONTEJNERA**
- Pokreće interaktivni terminal u kontejneru. Ovo je korisno za izvršavanje proizvoljnih naredba
 - **docker exec -it IME_KONTEJNERA /bin/bash**

Kada se prikačimo za mongo kontejner, dostupan nam je CLI alat **mongo** kojim možemo izvršavati proizvoljne naredbe za upravljanje mongo bazom. Neke osnovne komande ovog alata su:

- Prikazuje sve baze podataka
 - **show dbs**
- Bira bazu podataka **BAZA_PODATAKA** za koju će se odnositi sve dalje naredbe (odabrana BP biće dostupna kroz objekat **db**)
 - **use BAZA_PODATAKA**
- Čitanje svih dokumenata iz kolekcije **IME_KOLEKCIJE**
 - **db.IME_KOLEKCIJE.find({})**
- Unošenje novog dokumenta u kolekciju **IME_KOLEKCIJE**
 - **db.IME_KOLEKCIJE.insertOne({ name: 'Pera', prezime: 'Perić' })**
- Ažuriranje
 - **db.IME_KOLEKCIJE.updateOne({ name: 'Pera' }, { \$set: { izmenjen: true } })**
- Brisanje
 - **db.IME_KOLEKCIJE.deleteOne({ name: 'Pera' })**

Kreiranje projekta i instaliranje paketa

S obzirom da započinjemo razvoj „od nule“, potrebno je prvo da napravimo jedan *Solution* pre nego što kreiramo bilo koji projekat:

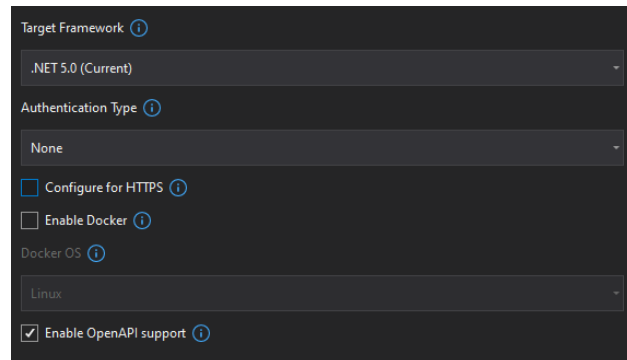
- File > New > Project
- Blank Solution
- Popuniti neophodnim podacima:
 - Solution name: **Webstore**
 - Location: **LOKACIJA_LOKALNOG_REPOZITORIJUMA**
 - Solution: **Create new solution**
- Create

Sada možemo da kreiramo nove projekte:

- Desni klik na ime *Solution-a*
- Add > New Project
- ASP.NET Core Web API
- Popuniti neophodnim podacima:
 - Project name: **Catalog.API**
 - Location: **LOKACIJA_LOKALNOG_REPOZITORIJUMA\Services\Catalog**
- Next
- Odabrati opcije kao na slici pored:
- Create

Pre nego što krenemo sa razvojem, potrebno je da instaliramo neophodne pakete. Otvoriti *NuGet Package Manager* sledećim koracima:

- Desni klik na naziv projekta
- Manage NuGet Packages



Prvo ćemo ažurirati sve pakete koji su do sada instalirani:

- Updates
- Select all packages
- Update

Zatim je potrebno otvoriti tab „Browse“ i tu pronaći i instalirati sledeće pakete:

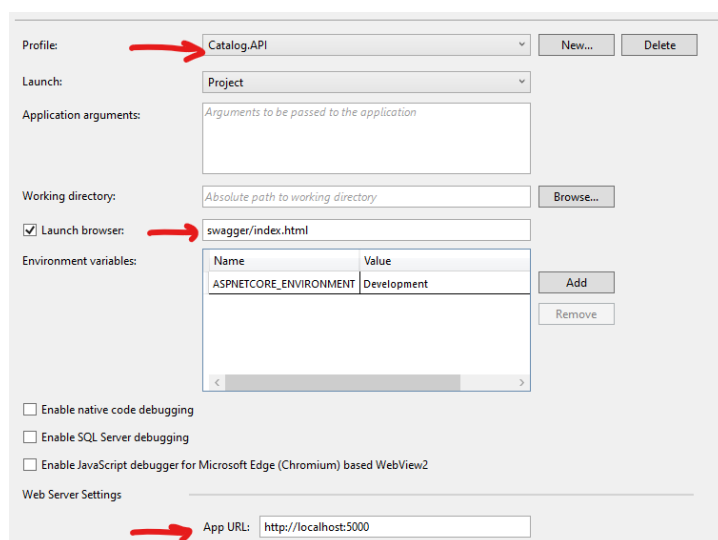
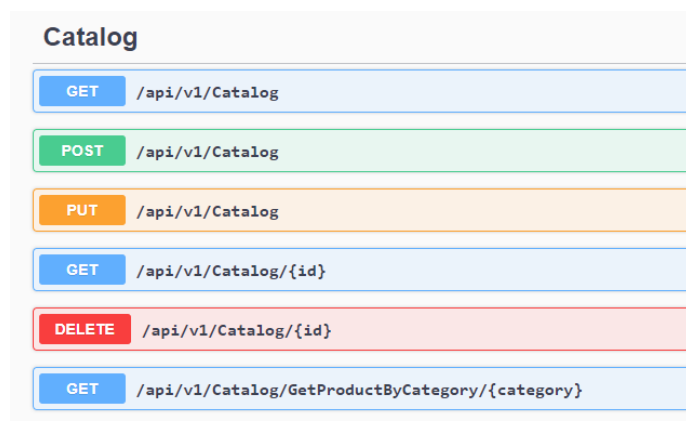
- MongoDB.Driver

Razvoj mikroservisa

API mikroservisa je opisan slikom pored.

Prvo podešavamo opcije za pokretanje aplikacije:

- Desni klik na naziv projekta
- Properties
- Debug
- Podesiti opcije sa slike ispod.



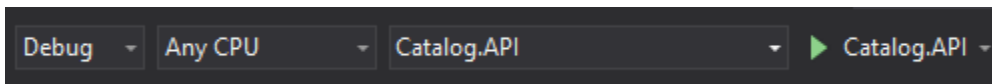
Zatim prelazimo na kodiranje, prema narednom redosledu:

- Entities
 - Product.cs
- Data
 - ICatalogContext.cs
 - CatalogContext.cs
 - CatalogContextSeed.cs
- Repositories
 - IProductRepository.cs
 - ProductRepository.cs
- Controllers
 - CatalogController.cs

Sada je preostalo da dodamo ubrizgavanje zavisnosti, što se nalazi u **Startup.cs** datoteci.

Pokretanje i debugiranje aplikacije

Iz gornjeg menija odabrati naredne opcije i pokrenuti aplikaciju:



Postaviti u nekom zahtevu tačku prekida i prolaziti kroz kod. Prikazivati Visual Studio okruženje za debugiranje.

2. Višestruki mikroservisi. Kontejnerizacija aplikacije.

Tema ovih časova je rad sa Redis, distribuiranom keš memorijom i kontejnerizacija projekata pomoću Docker i Docker Compose alata.

1. Razvoj Basket.API mikroservisa

Priprema Redis kontejnera

Na stranici <https://hub.docker.com/> uneti „redis“ u polje za pretragu. Otvoriti sledeću stranicu:



Osnovni Redis pojmovi: baza ključ-vrednost, keš memorija, prednosti i ograničenja (<https://redis.io/topics/faq>).

Pokretanje redis kontejnera:

- Otvoriti Powershell
- **docker run -d -p 6379:6379 --name redis_basket redis**

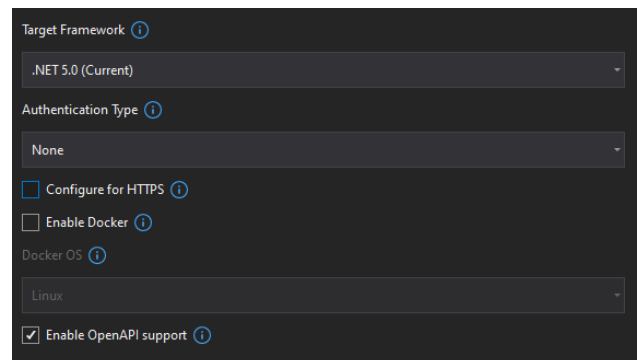
Kada se prikačimo za redis kontejner, dostupan nam je alat **redis-cli** kojim možemo izvršavati proizvoljne naredbe za upravljanje redis bazom. Neke osnovne komande ovog alata su:

- Provera da li je baza spremna (očekuje se odgovor PONG)
 - ping
- Postavljanje vrednosti
 - set **KLJUČ VREDNOST**
- Čitanje vrednosti
 - get **KLJUČ**

Kreiranje projekta i instaliranje paketa

Dodajemo novi projekat u okviru već napravljenog *Solution-a*:

- Desniklik na ime *Solution-a*
- Add > New Project
- ASP.NET Core Web API
- Popuniti neophodnim podacima:
 - Project name: **Basket.API**
 - Location:
LOKACIJA_LOKALNOG_REPOZITORIJUMA\Services\Basket
- Next
- Odabrali opcije kao na slici pored.
- Create



Pre nego što krenemo sa razvojem, potrebno je da instaliramo neophodne pakete. Otvoriti *NuGet Package Manager* sledećim koracima:

- Desni klik na naziv projekta
- Manage NuGet Packages

Prvo ćemo ažurirati sve pakete koji su do sada instalirani:

- Updates
- Select all packages
- Update

Zatim je potrebno otvoriti tab „Browse“ i tu pronaći i instalirati sledeće pakete:

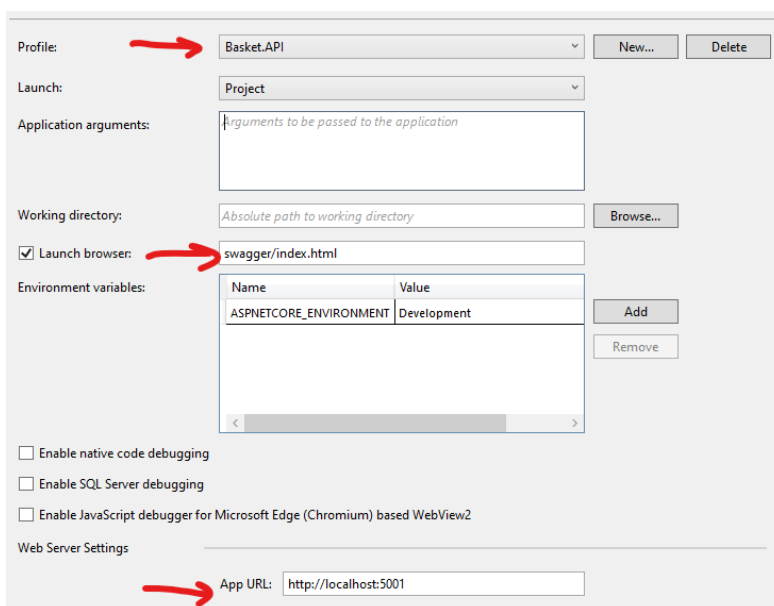
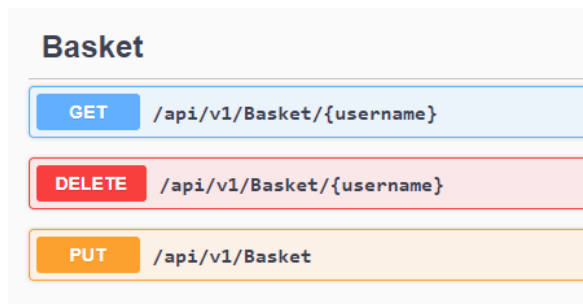
- Newtonsoft.Json
- Microsoft.Extensions.Caching.StackExchangeRedis

Razvoj mikroservisa

API mikroservisa je opisan slikom pored.

Prvo podešavamo opcije za pokretanje aplikacije:

- Desni klik na naziv projekta
- Properties
- Debug
- Podesiti opcije sa slike ispod.



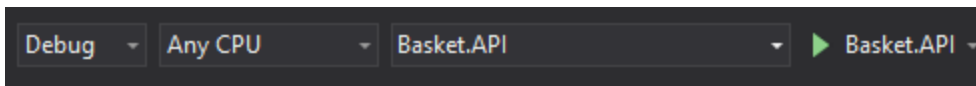
Zatim prelazimo na kodiranje, prema narednom redosledu:

- Entities
 - ShoppingCartItem.cs
 - ShoppingCart.cs
- Repositories
 - IBasketRepository.cs
 - BasketRepository.cs
- Controllers
 - BasketController.cs

Sada je preostalo da dodamo ubrizgavanje zavisnosti, što se nalazi u **Startup.cs** datoteci.

Pokretanje i debugiranje aplikacije

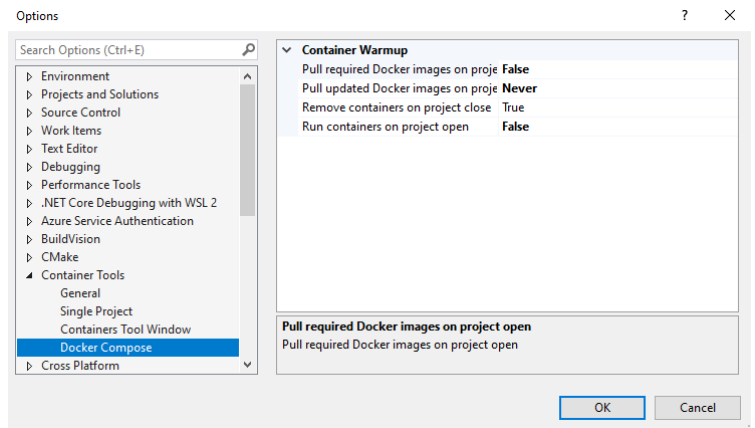
Iz gornjeg menija odabrati naredne opcije i pokrenuti aplikaciju:



2. Kontejnerizacija projekata

Pre nego što bilo šta uradimo, preporuka je da postavimo naredne opcije u Visual Studio alatu, kako bismo izbegli neka suvišna pokretanja Docker Compose alata:

- Tools > Options
- Otvoriti Container Tools grupu opcija
- Odabrati opciju Docker Compose
- Odabrati opcije sa naredne slike:



Dodavanje podrške za Docker Compose projektu:

- Desni klik na naziv projekta
- Add > Container Orchestrator Support
- Docker Compose
- Ok
- Linux
- Ok

Proći kroz generisani **Dockerfile** i objasniti neke najvažnije elemente, a zatim objasniti **docker-compose.yml** i **docker-compose.override.yml** datoteke.

Dodati naredne resurse u **docker-compose.yml** datoteku:

services:

catalogdb:

image: mongo

basketdb:

image: redis:alpine

volumes:

mongo_data:

Dodati naredne resurse u **docker-compose.override.yml** datoteku:

services:

catalogdb:

container_name: catalogdb

restart: always

ports:

- "27017:27017"

volumes:

- mongo_data:/data/db

basketdb:

container_name: basketdb

restart: always

ports:

- "6379:6379"

catalog.api:

container_name: catalog.api

environment:

- ASPNETCORE_ENVIRONMENT=Development
- "DatabaseSettings:ConnectionString=mongodb://catalogdb:27017"

depends_on:

- catalogdb

ports:

- "8000:80"

basket.api:

container_name: basket.api

environment:

- ASPNETCORE_ENVIRONMENT=Development
- "CacheSettings:ConnectionString=basketdb:6379"

depends_on:

- basketdb

ports:

- "8001:80"

Pokretanje projekta iz komandne linije

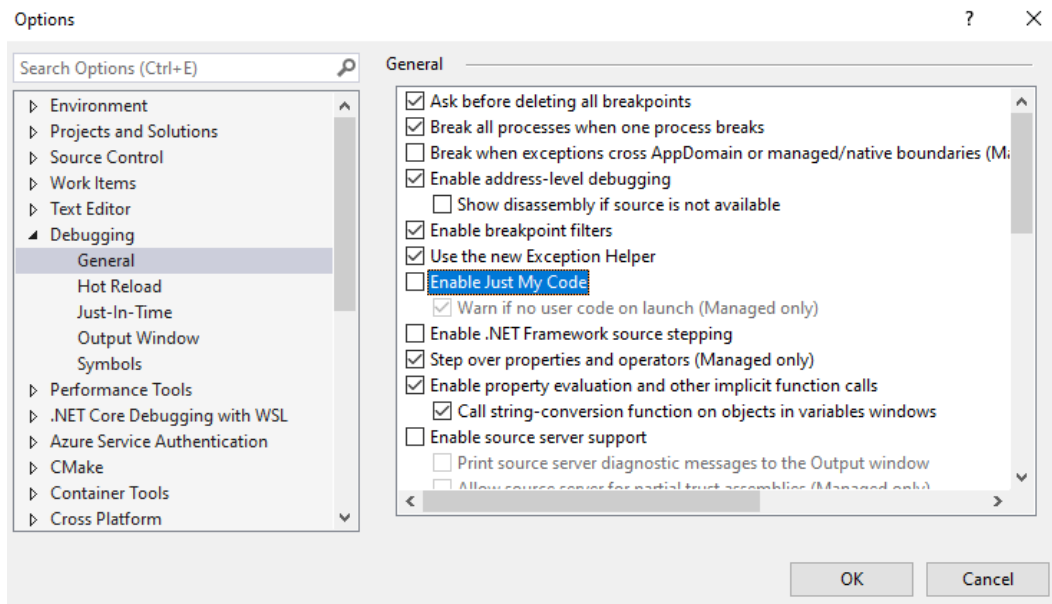
Pokretanje kontejnera iz komandne linije se vrši alatom **docker-compose** koja ima nekoliko važnih komandi (opcija **-d** označava da će se proces nastaviti u pozadini, kako se ne bi blokirao terminal):

- Izgradnja svih kontejnera
 - **docker-compose build**
- Podizanje svih kontejnera
 - **docker-compose up -d**
- Izgradnja i podizanje svih kontejnera
 - **docker-compose up --build -d**
- Zaustavljanje svih kontejnera
 - **docker-compose stop**
- Zaustavljanje i uklanjanje svih kontejnera
 - **docker-compose down**
- Specifikovanje datoteka koje se koriste za podizanje/spuštanje svih kontejnera
 - **docker-compose -f docker-compose.yml -f docker-compose.override.yml up --build -d**
 - **docker-compose -f docker-compose.yml -f docker-compose.override.yml**

Debugiranje projekata u kontejneru

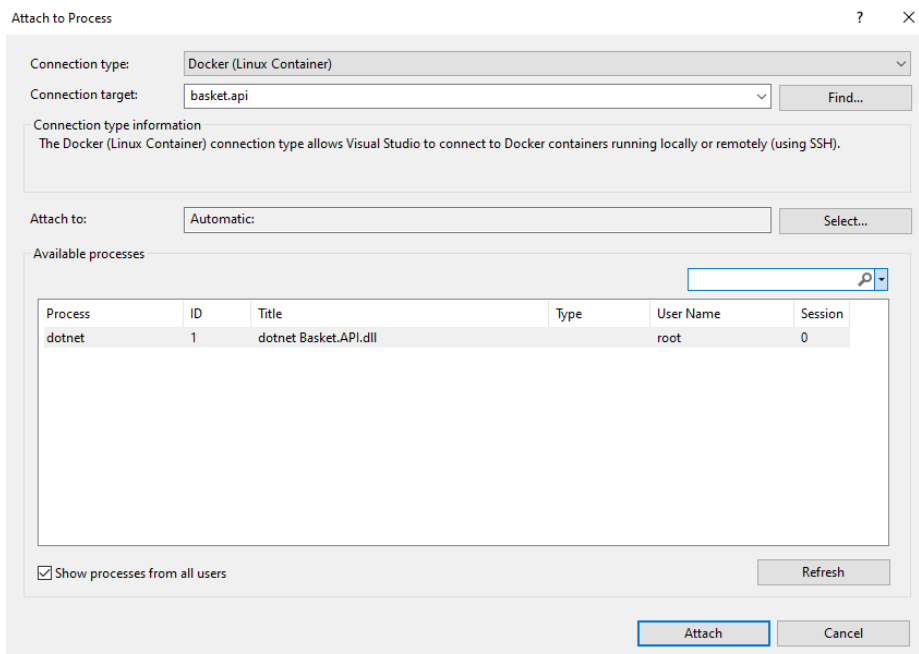
Kako bismo omogućili debugiranje projekata koji se pokreću u kontejneru, potrebno je da ručno *zakačimo* debager za proces koji se izvršava u kontejneru. Pre toga, potrebno je isključiti opciju „Enable Just My Code“ u podešavanjima, kako bismo instruisali Visual Studio da debugira i kod koji je izgrađen u kontejnerima:

- Tools > Options
- Debugging > General
- Isključiti opciju „Enable Just My Code“, kao na slici ispod



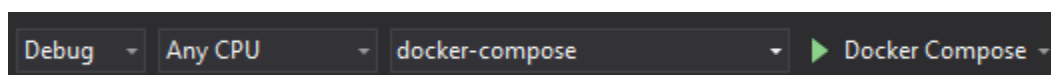
Sada možemo zakačiti debager:

- Debug > Attach to Process
- Za „Connection type“ odabrati Docker (Linux Container)
- Odabrati dugme Find
- Nakon nekoliko sekundi bi trebalo da se pojavi spisak svih podignutih kontejnera. Odabrati, na primer, basket.api projekat, pa dugme Ok
- U tabeli „Available processes“ bi trebalo da se pojavi „dotnet“ proces, kao na slici ispod
- Odabrati taj proces, pa dugme Attach
- Odabrati opciju „Managed (.NET Core for Unix)“, pa dugme Ok
- Posle nekoliko sekundi, debager će biti *zakačen* za proces



Pokretanje projekta iz Visual Studio alata

Nakon što smo napravili **docker-compose** projekat, potrebno je da odaberemo naredne opcije u glavnom meniju:



Klikom na pokretanje se vrše naredne akcije:

- Izgrađuje se kod iz *Solution-a*
- Izgrađuju se kontejneri
- Pokreću se kontejneri
- Pokreće se debager i automatski se zakači za izvršni kod

Neke napomene

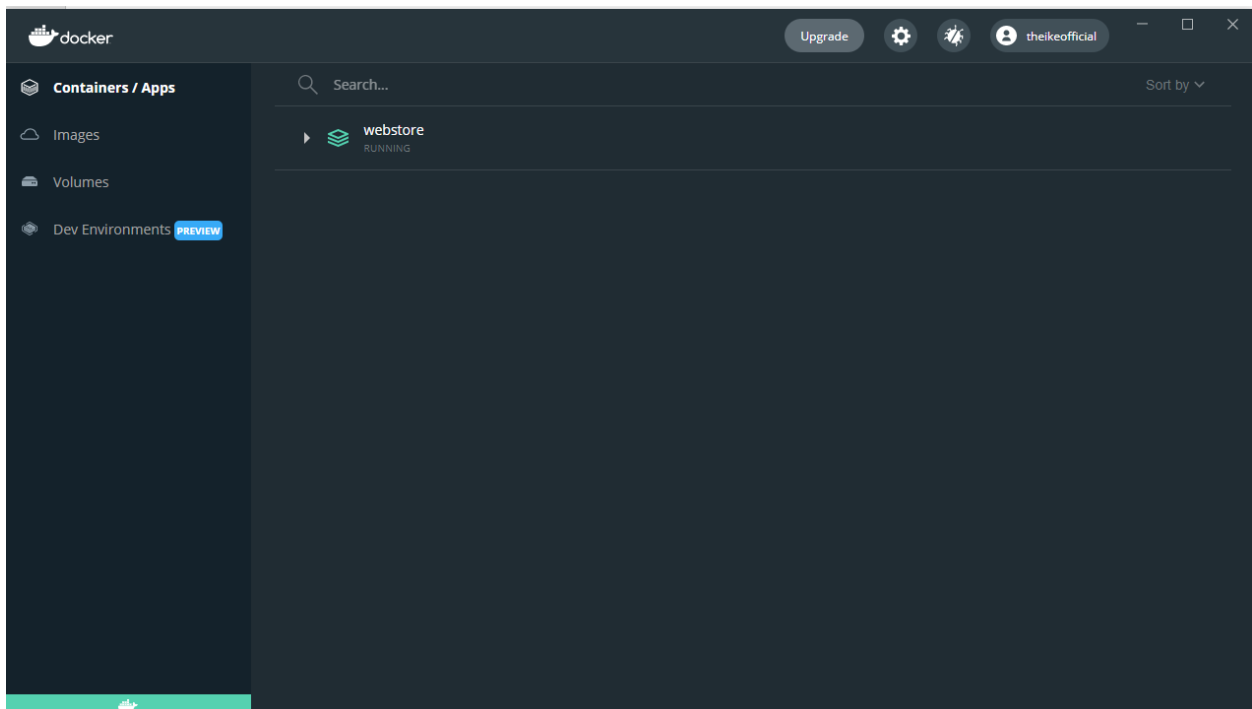
- Prednosti korišćenja su automatsko podizanje i automatsko debugiranje.
- Gašenjem projekta (bilo iz Visual Studio alata, gašenjem pregledača ili gašenjem terminala) se ne gase kontejneri, ali se ne mogu ugasiti iz **Docker Desktopa**, pa je neophodno uraditi **Build > Clean solution** iz glavnog menija – obavezno pre zatvaranja Visual Studio alata.
- Treba imati u vidu da će se konfiguracija prvo pročitati iz **appsettings.json** i **appsettings.Development.json** datoteka nego iz **docker-compose.yml** datoteka. Preporuka je da se prepisu sve promenljive okruženja iz Yaml datoteka u json pre pokretanja. Naravno, ovime se onemogućava pojedinačno pokretanje projekata iz Visual Studio alata.
- Debugiranje u kontejnerima je nešto sporije u odnosu na debugiranje aplikacija koje su pokrenute na host računaru, ali funkcioniše identično.

Docker Desktop

Alat Docker Desktop nam služi za upravljanje kontejnerima iz grafičke korisničke aplikacije. Na narednoj slici možemo videti prikaz nakon pokretanja. Sa strane možemo birati neke od tabova koji nam daju uvid u naredne elemente za rad sa kontejnerima:

- **Containers/Apps** prikazuje pregled svih kontejnera koji postoje na sistemu
- **Images** prikazuje pregled svih slika koji su dovučeni na sistemu
- **Volumes** prikazuje pregled svih „diskova“ koje kontejneri koriste za trajno skladištenje datoteka
- **Dev Environments** prikazuje pregled okruženja za razvoj za jednostavnu kolaboraciju razvijalaca softvera u timu

Nama će najznačajniji biti prvi pregled.



Kontejneri mogu biti pokrenuti pojedinačno, ili kao deo neke mreže, tj. orkestra kontejnera. Na slici ispod je prikazana orkestrizacija celokupne aplikacija *Webstore* koja se, u ovom trenutku, sastoji od četiri kontejnera. Sa

desne strane možemo videti zajedničke dnevnike, a klikom na konkretan kontejner, biće nam prikazan dnevnik samo za taj kontejner.

The screenshot shows the Docker Desktop application. On the left, the 'Containers / Apps' section is active, displaying a list of containers for the 'webstore' project. The containers are:

- catalogdb (mongo) - RUNNING - PORT: 27017
- basketdb (redis:alpine) - RUNNING - PORT: 6379
- catalog.api (catalogapi) - RUNNING - PORT: 8000
- basket.api (basketapi) - RUNNING - PORT: 8001

The right pane shows the logs for the selected container. The logs are as follows:

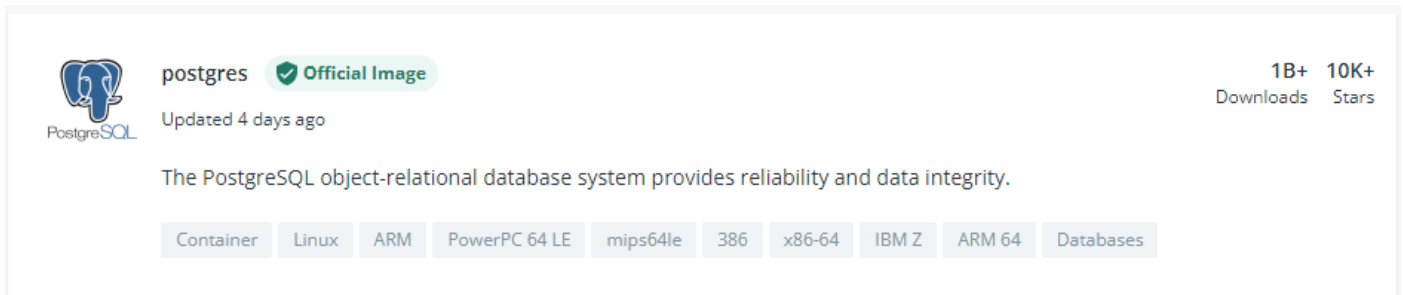
```
ta checkpoint timestamp: (0, 0) base write gen: 76"))
basket.api | info: Microsoft.Hosting.Lifetime[0]
basket.api | Now listening on: http://[::]:80
basket.api | info: Microsoft.Hosting.Lifetime[0]
basket.api | Application started. Press Ctrl+C to shut down.
basket.api | info: Microsoft.Hosting.Lifetime[0]
basket.api | Hosting environment: Development
basket.api | info: Microsoft.Hosting.Lifetime[0]
basket.api | Content root path: /app
catalog.api | info: Microsoft.Hosting.Lifetime[0]
catalog.api | Now listening on: http://[::]:80
catalog.api | info: Microsoft.Hosting.Lifetime[0]
catalog.api | Application started. Press Ctrl+C to shut down.
catalog.api | info: Microsoft.Hosting.Lifetime[0]
catalog.api | Hosting environment: Development
catalog.api | info: Microsoft.Hosting.Lifetime[0]
catalog.api | Content root path: /app
basketdb | 1:C 22 Oct 2021 15:25:24.647 # oO00oO00oO00o Redis is starting oO00oO00o
basketdb | 1:C 22 Oct 2021 15:25:24.647 # Redis version=6.2.6, bits=64, commit=000
00000, modified=0, pid=1, just started
basketdb | 1:C 22 Oct 2021 15:25:24.647 # Warning: no config file specified, using
the default config. In order to specify a config file use redis-
server /path/to/redis.conf
basketdb | 1:M 22 Oct 2021 15:25:24.647 * monotonic clock: POSIX clock_gettime
basketdb | 1:M 22 Oct 2021 15:25:24.648 * Running mode=standalone, port=6379.
basketdb | 1:M 22 Oct 2021 15:25:24.648 # Server initialized
basketdb | 1:M 22 Oct 2021 15:25:24.648 * Ready to accept connections
```

3. Sinhrona komunikacija između mikroservisa pomoću gRPC protokola

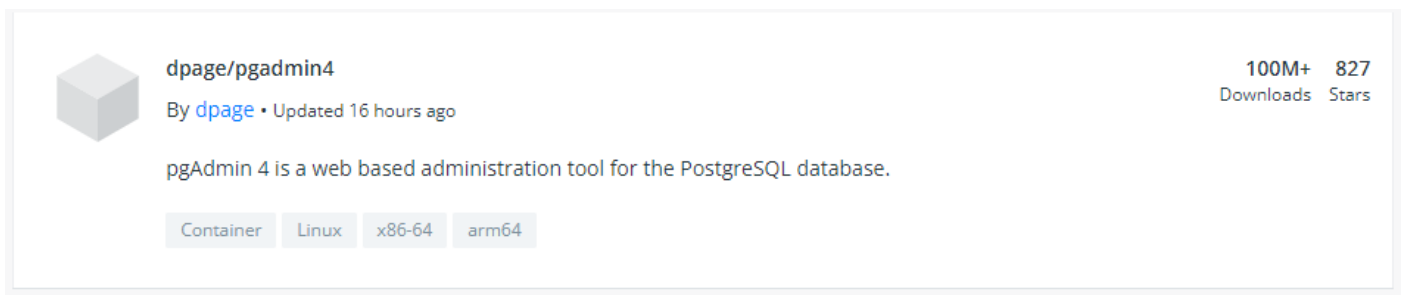
Tema ovih časova je ostvarivanje sinhronne komunikacije između mikroservisa implementiranjem gRPC protokola. Videćemo i rad sa PostgreSQL SUBP korišćenjem vrlo jednostavne biblioteke za ORP – Dapper.

1. Priprema PostgreSQL i pgadmin kontejnera

Na stranici <https://hub.docker.com/> uneti „postgres“ u polje za pretragu. Otvoriti sledeću stranicu:



Na stranici <https://hub.docker.com/> uneti „pgadmin“ u polje za pretragu. Otvoriti sledeću stranicu:



Dodati naredne resurse u **docker-compose.yml** datoteku:

services:

discountdb:

image: postgres

pgadmin:

image: dpage/pgadmin4

volumes:

mongo_data:

postgres_data:

pgadmin_data:

Dodati naredne resurse u **docker-compose.override.yml** datoteku:

services:

discountdb:

container_name: discountdb

environment:

- POSTGRES_USER=admin

- POSTGRES_PASSWORD=admin1234

- POSTGRES_DB=DiscountDb

restart: always

ports:

- "5432:5432"

volumes:

- postgres_data:/var/lib/postgresql/data/

pgadmin:

container_name: pgadmin

environment:

- PGADMIN_DEFAULT_EMAIL=razvoj.softvera.matf@gmail.com

- PGADMIN_DEFAULT_PASSWORD=admin1234

restart: always

ports:

- "5050:80"

volumes:

- pgadmin_data:/root/.pgadmin

Alat pgAdmin4

Nakon pokretanja kontejnera, alat je dostupan na adresi <http://localhost:5050>. Prijaviti se na sistem korišćenjem adrese elektronske pošte i lozinke koji su navedeni u datoteci iznad.

Dodavanje novog servera:

- Add New Server
- General
 - Name: **DiscountServer**
- Connection
 - Host name/address: **discountdb**
 - Port: **5432**
 - Maintenance database: **postgres**
 - Username: **admin**
 - Password: **admin1234**
 - Napomena za podatke iznad: username i password treba da budu isto kao u **docker-compose.override.yml** datoteci.
- Save

Podesiti inicijalnu shemu baze podataka DiscountDB:

- Tools > Query Tool
- Izvršiti naredni upit:

```
CREATE TABLE Coupon (  
  ID SERIAL PRIMARY KEY NOT NULL,  
  ProductName VARCHAR(24) NOT NULL,  
  Description TEXT,  
  Amount INT  
);
```

Sada kreirajmo nekoliko inicijalnih vrednosti:

```
INSERT INTO Coupon (ProductName, Description, Amount) VALUES ('iPhone X', 'iPhone Discount', 150);  
INSERT INTO Coupon (ProductName, Description, Amount) VALUES ('Huawei Plus', 'Huawei Discount', 110);
```

```
INSERT INTO Coupon (ProductName, Description, Amount) VALUES ('Xiaomi Mi 9', 'Xiaomi Discount', 75);
INSERT INTO Coupon (ProductName, Description, Amount) VALUES ('Samsung 10', 'Samsung Discount', 100);
```

Nakon svake izvršene naredbe koja menja shemu je potrebno osvežiti pogled u *Browser-u* kako bi se videle izmene.

2. Kreiranje zajedničkog projekta za API i gRPC projekte

S obzirom da koristimo iste resurse u dva projekta, bilo bi dobro da izdvojimo sve zajedničke stvari iz tih projekata u jednu biblioteku klasa. Ovime izbegavamo dupliciranje koda, ali polako uvodimo slojevitost u našim projektima. Ovu ideju ćemo detaljnije produbiti kada budemo govorili o čistoj arhitekturi.

Kreiranje projekta i instalacija paketa

Dodajemo novi projekat u okviru već napravljenog *Solution-a*:

- Desni klik na ime *Solution-a*
- Add > New Project
- Class library
- Popuniti neophodnim podacima:
 - Project name: **Discount.Common**
 - Location: **LOKACIJA_LOKALNOG_REPOZITORIJUMA\Services\Discount**
- Next
- Odabrati opciju **.NET 5.0 (Current)**
- Create

Pre nego što krenemo sa razvojem, potrebno je da instaliramo neophodne pakete. Otvoriti *NuGet Package Manager* sledećim koracima:

- Desni klik na naziv projekta
- Manage NuGet Packages

Prvo ćemo ažurirati sve pakete koji su do sada instalirani:

- Updates
- Select all packages
- Update

Zatim je potrebno otvoriti tab „Browse“ i tu pronaći i instalirati sledeće pakete:

- Npgsql
- Dapper
- AutoMapper.Extensions.Microsoft.DependencyInjection
- Microsoft.Extensions.Configuration
- Microsoft.Extensions.Configuration.Binder
- Microsoft.Extensions.DependencyInjection.Abstractions

Razvoj biblioteke klasa za Discount projekte

Kodiranje vršimo prema narednom redosledu:

- Entities
 - Coupon.cs
- Data
 - ICouponContext.cs
 - CouponContext.cs
- DTOs
 - BaseCouponDTO.cs

- BaselIdentityCouponDTO.cs
- CouponDTO.cs
- CreateCouponDTO.cs
- UpdateCouponDTO.cs
- Repositories
 - ICouponRepository.cs
 - CouponRepository.cs
- Extensions
 - DiscountCommonExtensions.cs

3. Kreiranje API projekta

Dodajemo novi projekat u okviru već napravljenog *Solution-a*:

- Desni klik na ime *Solution-a*
- Add > New Project
- ASP.NET Core Web API
- Popuniti neophodnim podacima:
 - Project name: **Discount.API**
 - Location:
LOKACIJA_LOKALNOG_REPOZITORIJUMA
\Services\Discount
- Next
- Odabrati opcije kao na slici pored:
- Create

Pre nego što krenemo sa razvojem, potrebno je da dodamo referencu na prethodno napravljeni projekat **Discount.Common**:

- Desni klik na naziv projekta
- Add > Project Reference
- Odabrati Discount.Common
- Ok

Razvoj mikroservisa

API mikroservisa je opisan slikom pored.

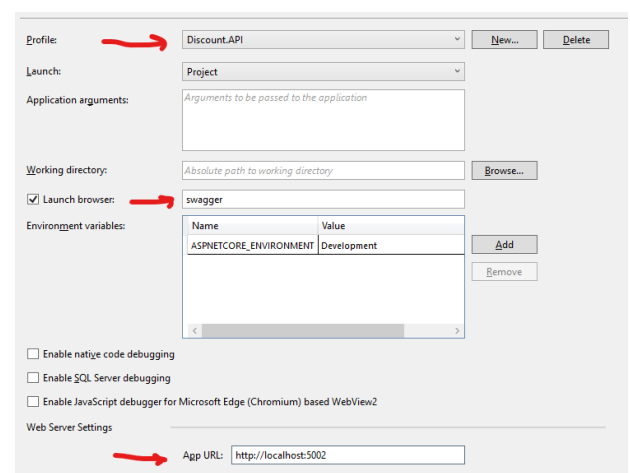
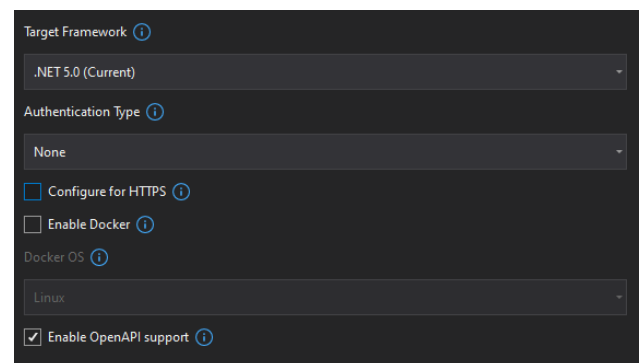
Prvo podešavamo opcije za pokretanje aplikacije:

- Desni klik na naziv projekta
- Properties
- Debug
- Podesiti opcije sa slike pored.

Zatim prelazimo na kodiranje, prema narednom redosledu:

- Controllers
 - DiscountController.cs

Sada je preostalo da dodamo ubrizgavanje zavisnosti, što se nalazi u Startup.cs datoteci.



Pokretanje i debugiranje aplikacije

Pokrenuti prvo **docker-compose** projekat iz terminala, a ujedno pokrenuti samo Discount.API projekat iz Visual Studio alata. Discount.API projekat bi trebalo da se poveže sa PostgreSQL bazom podataka ukoliko se u **appsettings.Development.json** datoteci doda naredna konfiguracija:

```
"DatabaseSettings": {  
  "ConnectionString": "Server=localhost;Port=5432;Database=DiscountDb;User Id=admin;Password=admin1234;"  
}
```

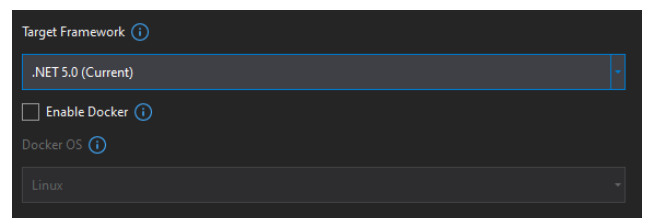
Sada dodati podršku za orkestrizaciju kontejnera za projekat Discount.API, pa pokrenuti sve kontejnere i testirati još jednom. Dopuniti **docker-compose.override.yml** datoteku narednim linijama:

```
discount.api:  
  container_name: discount.api  
  environment:  
    - ASPNETCORE_ENVIRONMENT=Development  
    - "DatabaseSettings:ConnectionString=Server=discountdb;Port=5432;Database=DiscountDb;User  
Id=admin;Password=admin1234;"  
  depends_on:  
    - discountdb  
  ports:  
    - "8002:80"
```

4. Kreiranje gRPC projekta

Dodajemo novi projekat u okviru već napravljenog *Solution-a*:

- Desni klik na ime *Solution-a*
- Add > New Project
- ASP.NET Core gRPC Service
- Popuniti neophodnim podacima:
 - Project name: **Discount.GRPC**
 - Location:
LOKACIJA_LOKALNOG_REPOZITORIJUMA
\Services\Discount
 - Next
- Odabrati opcije kao na slici pored.
- Create



Pre nego što krenemo sa razvojem, potrebno je da dodamo referencu na prethodno napravljeni projekat **Discount.Common**:

- Desni klik na naziv projekta
- Add > Project Reference
- Odabrati Discount.Common
- Ok

Sada ažurirajmo neophodne pakete:

- Desni klik na naziv projekta
- Manage NuGet Packages
- Updates
- Select all packages
- Update

Zatim je potrebno otvoriti tab „Browse“ i tu pronaći i instalirati sledeće pakete:

- Google.Protobuf
- Grpc.Core
- Grpc.Tools

Ako se i dalje pojavljuje greška u Build prozoru, samo očistiti i ponovo izgraditi projekat.

Razvoj mikroservisa

Prvo podešavamo opcije za pokretanje aplikacije:

- Desni klik na naziv projekta
- Properties
- Debug
- Podesiti opcije sa slike pored:

Da bismo implementirali gRPC protokol, potrebno je da postoje dve strane u komunikaciji: server i klijent.

Server je onaj mikroservis koji opslužuje druge nekim servisima, a klijent je onaj mikroservis koji koristi usluge nekih servera. U našem primeru, server je Discount.GRPC zato što on omogućava da se pretraže kuponi za dati proizvod, a Basket.API je klijent zato što on koristi tu uslugu pretrage kupona kako bi odredio konačnu cenu potrošačke korpe.

Za gRPC implementaciju su potrebne dve stvari:

- *proto buffer* datoteka koja opisuje gRPC servise
- C# klasa koja implementira te opisane servise

Generisanje *proto buffer* datoteke se radi narednim koracima:

- Desni klik na direktorijum *Protos*
- Add > New Item
- Protocol Buffer File
- Name: coupon.proto

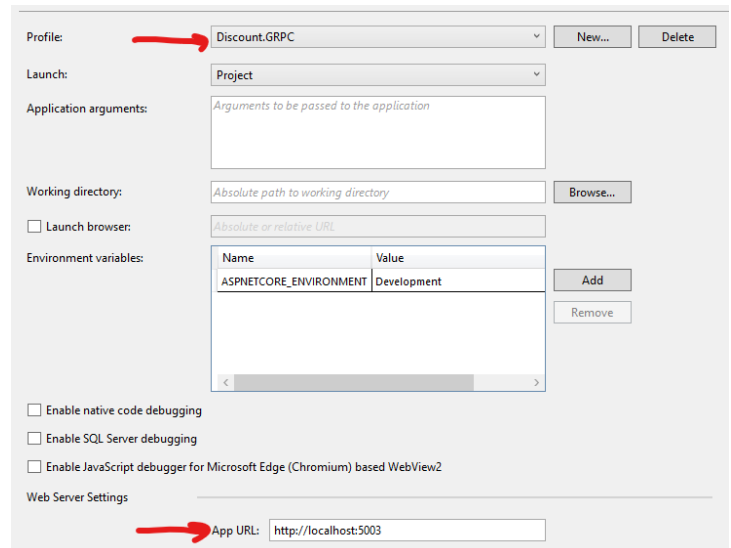
Sada je potrebno definisati servise koje želimo da omogućimo. U tu svrhu se koristi *proto3* sintaksa (više o ovom jeziku na <https://developers.google.com/protocol-buffers/docs/proto3>). Obavezno postaviti da se datoteka koristi za potrebe definisanja gRPC servera:

- Desni klik na *proto buffer* datoteku > Properties
- Build action: Protobuf compiler
- gRPC Stub Classes: Server Only

Kada se završi pisanje *proto buffer* datoteke, možemo generisati implementaciju narednim koracima:

- Desni klik na direktorijum *Services*
- Add > Class
- Name: CouponService
- Naslediti klasu **CouponProtoService.CouponProtoServiceBase**
- Desni klik na našu klasu > Generate overrides

Sada smo dobili potpise metoda koje implementiramo kako god želimo. Ova klasa predstavlja na neki način kontroler, samo što koristi gRPC protokol umesto HTTP.



Ne zaboraviti da je potrebno da se u **Startup.cs** doda rutiranje gRPC servisa u HTTP *pipeline*-u:

```
endpoints.MapGrpcService<CouponService>();
```

kao i ubrizgavanje zavisnosti i preslikavanje neophodnih modela:

```
services.AddDiscountCommonServices();
services.AddAutoMapper(configuration =>
{
    configuration.CreateMap<Coupon, CouponModel>().ReverseMap();
});
```

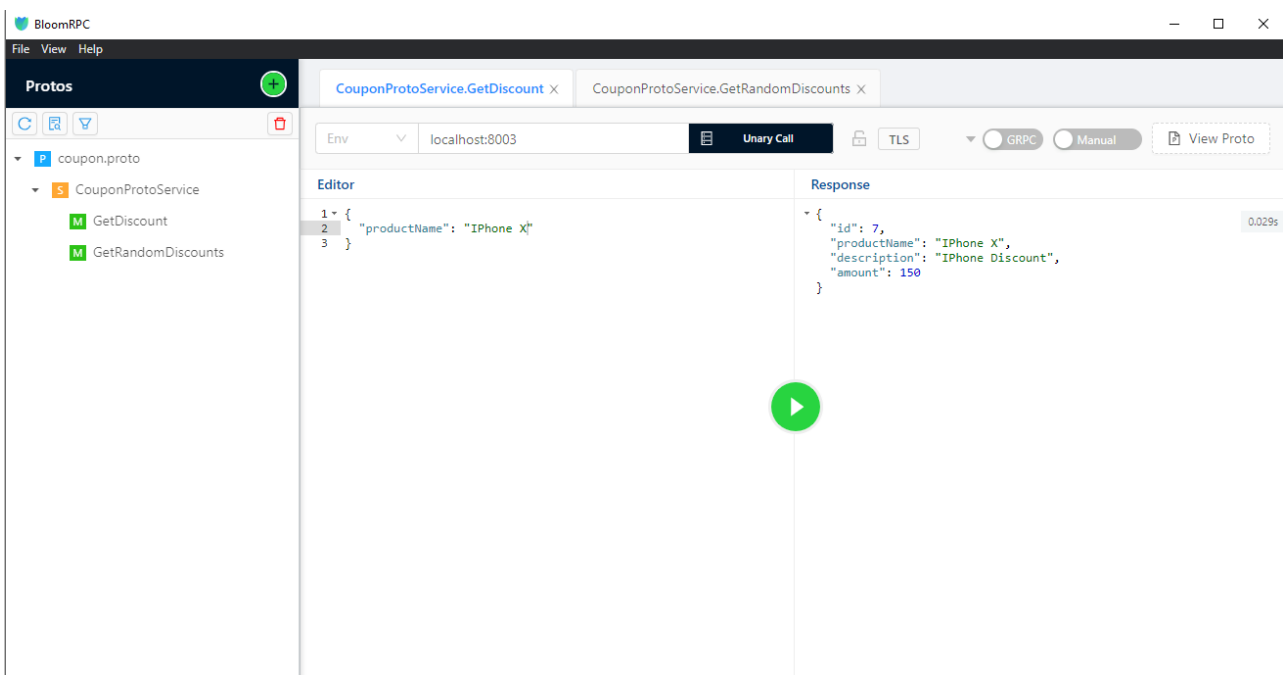
Pokretanje i debugiranje aplikacije

Pokrenuti prvo **docker-compose** projekat iz terminala, a ujedno pokrenuti samo Discount.GRPC projekat iz Visual Studio alata. Discount.GRPC projekat bi trebalo da se poveže sa PostgreSQL bazom podataka ukoliko se u **appsettings.Development.json** datoteci doda naredna konfiguracija:

```
"DatabaseSettings": {
  "ConnectionString": "Server=localhost;Port=5432;Database=DiscountDb;User Id=admin;Password=admin1234;"
}
```

Otvoriti BloomRPC alat, učitati *proto buffer* datoteku i testirati gRPC servis slanjem zahteva kao na narednoj slici. Pre prikazivanja slike napomenimo sledeće:

- Dok se testira projekat kad je pokrenut lokalno, u alatu upisati adresu **localhost:5003**
- Dok se testira projekat u kontejneru, u alatu upisati adresu **localhost:8003**



Sada dodati podršku za orkestrizaciju kontejnera za projekat Discount. GRPC, pa pokrenuti sve kontejnere i testirati još jednom. Dopuniti **docker-compose.override.yml** datoteku narednim linijama:

```
discount.grpc:
  container_name: discount.grpc
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - "DatabaseSettings:ConnectionString=Server=discountdb;Port=5432;Database=DiscountDb;User
Id=admin;Password=admin1234;"
  depends_on:
```

- **discountdb**

ports:

- "8003:80"

5. Korišćenje gRPC projekta u Basket mikroservisu

Vratimo se sada na Basket.API mikroservis kako bismo dodali neophodan kod za ostvarivanje gRPC komunikacije ka Discount.GRPC mikroservisu. Koraci koje je neophodno izvršiti su slični kao za server, ali neke stvari je moguće ubrzati:

- Registracija gRPC servisa
- Implementacija klijentske logike
- Dodavanje informacija o gRPC komunikaciji i ubrizgavanje zavisnosti

Da bismo registrovali novi gRPC servis, potrebno je da uradimo naredne korake:

- Desni klik na naziv Basket.API projekta
- Add > Connected Service
- U okviru „Service References (OpenAPI, gRPC, WCF Web Service)“ odabрати dugme „+“
- gRPC
- File > Browse
- Pronaći *proto buffer* datoteku koju smo implementirali na serveru. Trebalo bi da bude na putanji oblika **LOKACIJA_LOKALNOG_REPOZITORIJUMA\Services\Discount\Discount.GRPC\Protos\coupon.proto**
- Select the type of class to be generated: **Client**
- Finish
- Close

Sada možemo da napravimo klasu koju ćemo koristiti u ovom projektu, a koja se oslanja na automatski generisanog klijenta u gRPC komunikaciji:

- GrpcServices
 - CouponGrpcService.cs
- Controllers
 - BasketController.cs

Konačno, neophodno je da u **Startup.cs** datoteci navedemo podešavanja za gRPC komunikaciju. Najpre, potrebno je da navedemo koji gRPC servis „gađa“ generisani klijent, kao i da ubrizgamo klasu koju smo mi implementirali iznad:

```
services.AddGrpcClient<CouponProtoService.CouponProtoServiceClient>  
(o => o.Address = new Uri(Configuration["GrpcSettings:DiscountUrl"]));  
services.AddScoped<CouponGrpcService>();
```

Očigledno, neophodno je dodati adresu u konfiguraciju, jedanput u datoteci **appsettings.Development.json** i jedanput u **docker-compose.override.yml**. Dodatno, u drugoj datoteci je potrebno navesti da Basket.API mikroservis očekuje da Discount.GRPC servis bude pokrenut pre njega.

U **appsettings.Development.json** datoteci:

```
"GrpcSettings": {  
  "DiscountUrl": "http://localhost:5003"  
}
```

U **docker-compose.override.yml** datoteci:

```
basket.api:  
  container_name: basket.api
```

environment:

- ASPNETCORE_ENVIRONMENT=Development
- "CacheSettings:ConnectionString=basketdb:6379"
- "GrpcSettings:DiscountUrl=http://discount.grpc"

depends_on:

- basketdb
- discount.grpc

ports:

- "8001:80"

Za testiranje PUT zahteva za Basket.API mikroservis se može koristiti naredni JSON kod:

```
{
  "username": "rs2",
  "items": [
    {
      "quantity": 2,
      "color": "white",
      "price": 950,
      "productId": "602d2149e773f2a3990b47f5",
      "productName": "iPhone X"
    },
    {
      "quantity": 1,
      "color": "white",
      "price": 380,
      "productId": "602d2149e773f2a3990b47f9",
      "productName": "HTC U11+ Plus"
    }
  ]
}
```

Ono što možemo primetiti sa slike ispod jeste da se u odgovoru zahteva primenio kupon na uređaj „iPhone X“, ali da je uređaj „HTC U11+ Plus“ zadržao svoju originalnu cenu.

