

3. laboratorijska vježba - zadatak

Cilj 3. laboratorijske vježbe je upoznavanje dobrih praksi prilikom korištenja alata za verzioniranje koda (npr. Git), te početak izrade prvih dijelova podatkovnog cjevovoda (*engl. data pipeline*), kojeg ćete nadopunjavati novim komponentama u idućim laboratorijskim vježbama.

Azure DevOps i Git flow

Zadaci 1. - 4. su kreirani za upoznavanje sa osnovama sučelja Azure DevOps, te je 5. zadatak za upoznavanje s git flow-om u praksi. Pridržavajte se uputa za Git flow tijekom razvoja cijelog projekta.

1. Otiđite na stranicu: <https://azure.microsoft.com/en-us/services/devops/>, te kliknite sign in to Azure DevOps. Prijavite se sa svojim odgovarajućim podacima.

[Home](#) / [Services](#) / Azure DevOps

Azure DevOps

Plan smarter, collaborate better, and ship faster with a set of modern dev services.

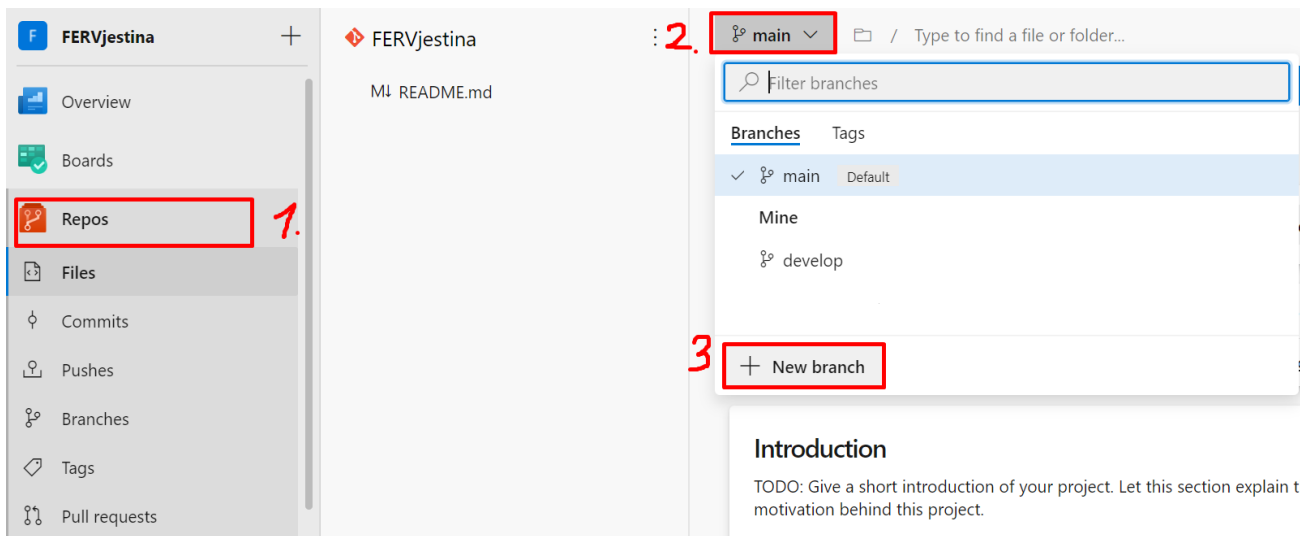
Start free

Start free with GitHub

Already have an account?

[Sign in to Azure DevOps >](#)

2. Jedna osoba iz tima trebala bi kreirati novi projekt te dodati ostale osobe kao "Members" na projektu.
 - a. Kliknuti "New Project"
 - b. Dati naziv projektu (preporuka: <ime_tima>, npr. Tim1)
 - c. Kliknuti Create
 - d. Kada vam se otvori *Dashboard* novokreiranog projekta kliknuti plavi gumb na kojem piše "Invite".
Napomena: Osobe moraju imati dodijeljenu **Basic** rolu kako bi svi mogli sudjelovati na projektu i vidjeti Repozitorij na kojem se nalazi kod. To se može i naknadno promijeniti tako da osoba koja je kreirala projekt klikne na projektne postavke (Organization settings) i svima dodijeli Basic rolu: <https://docs.microsoft.com/en-us/azure/devops/organizations/security/access-levels?view=azure-devops>
 - e. Dodajte sve svoje kolege iz tima u repozitorij
 - f. Dodane kolege neka provjere na svojim Azure DevOpsima jesu li dodani na projekt (trebao bi im se pokazati odmah na početnoj stranici nakon uspješne prijave)
3. Sada kada svi imate pristup istom repozitoriju, jedna osoba iz tima treba kreirati novu granu (*engl. branch*) i nazvati ju *develop*. Na *develop* granu idu gotove i testirane funkcionalnosti koje ste razvili na granama izvedenim iz *develop* grane. Tek kada je cijeli projekt gotov sve ćete spojiti (*engl. merge*) na *main* granu. Ovakav princip naziva se *Git flow*. Za detaljnije upute pogledati: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.
Struktura repozitorija trebala bi pratiti sljedeća pravila:
 - a. *Main* grana - sadrži samo u potpunosti gotov projekt
 - b. *Develop* grana - sadrži sve glavne release-ove projekta, odnosno na *develop*-u se u jednom trenutku nalazi zadnja funkcionalna verzija vašeg projekta. Kada projekt bude u potpunosti gotov *develop* granu ćete spojiti s *main* granom.
 - c. *Feature* grana - Svaka nova funkcionalnost koju razvijate radi se na tzv. *feature* grani. Kada je neka funkcionalnost gotova svoju *feature* granu spajate s *develop* granom.
4. Svaka osoba iz tima koja radi na nekoj novoj funkcionalnosti projekta trebala bi to raditi na grani izvedenoj iz *develop*. Novu granu možete napraviti klikom na Repos Main +New Branch ili iz komadne linije sljedećom naredbom:
`git checkout -b feature_branch_name`
Dogovorite se oko konvencije imenovanja *feature* grana, npr. *ft/ime_funkcionalnosti*.



5. Kako biste se upoznali s Azure DevOpsom kreirajte README datoteku na *develop* grani. Naizmjenično pull-ajte README s *develop*-a, kreirajte novu granu (nazivlje: ft/<naziv_funkiconalnosti, npr. ft/README_<vaše_ime>). Dodajte svoje ime na kraj README datoteke te pushajte svoje promjene.
6. Nakon što razvijete neku funkcionalnost, u ovome slučaju promjenu na README, potrebno je kreirati *pull request* na *develop* granu. Unutar Azure DevOps-a odaberite svoju granu na kojoj se nalazi vaš kod. Kliknite *Create a Pull Request*. Pobrinite se da vam piše pravilni naziv grane nad kojom ćete napraviti *pull request*. Dodajte naslov commit-a, opis, te dodajte u polje *Reviewers* svoj tim. Na kraju kliknite Create.

New pull request

🔗 jelenas_branch ▾ into 🔗 **develop** ▾ ⇅

Overview Files 19 Commits 2

Title

Nova funkcionalnost

Description

Opis nove funkcionalnosti

📎 Add commit messages

🔗 [Markdown supported.](#) Drag & drop, paste, or select files to insert. 🔗 [Link work items.](#)

@ # 🔗 📎 ✎ ▾ **B** *I* </> 🔗 ☰ ☰ ☰

Opis nove funkcionalnosti

Reviewers

Add required reviewers

👤 [FERVjestina]\FERVjestina Team X 🔍 Search users and groups to add as reviewers

Work items to link

Search work items by ID or title ▾

Tags

Create ▾

7. Ostale osobe iz tima sada trebaju pregledati vaš *pull request*, pregledati kod i promjene koje želite spojiti na *develop* granu te komentirati vaš kod ukoliko je potrebno. Nakon toga mogu odobriti *merge* na *develop* granu ili ako uoče nedostatke u kodu mogu odbiti *merge* s *develop* granom.

Projektni zadatak

U sklopu ovog projekta, kroz iduće 4 laboratorijske vježbe, kreirat ćete podatkovni cjevovod koji će primati podatke s nekog izvora podataka (engl. *ingest*), procesuirati te skladištiti obrađene podatke. Za izvor podataka koristit ćete podatke iz već preuzetog skupa podataka (2. vježba). Također ćete morati kreirati *CI/CD pipeline* koji služi za automatizaciju samog deploy procesa na *Azure Cloud*. Arhitektura projekta koji ćete razvijati kroz sljedeće vježbe je prikazana na donjoj slici.

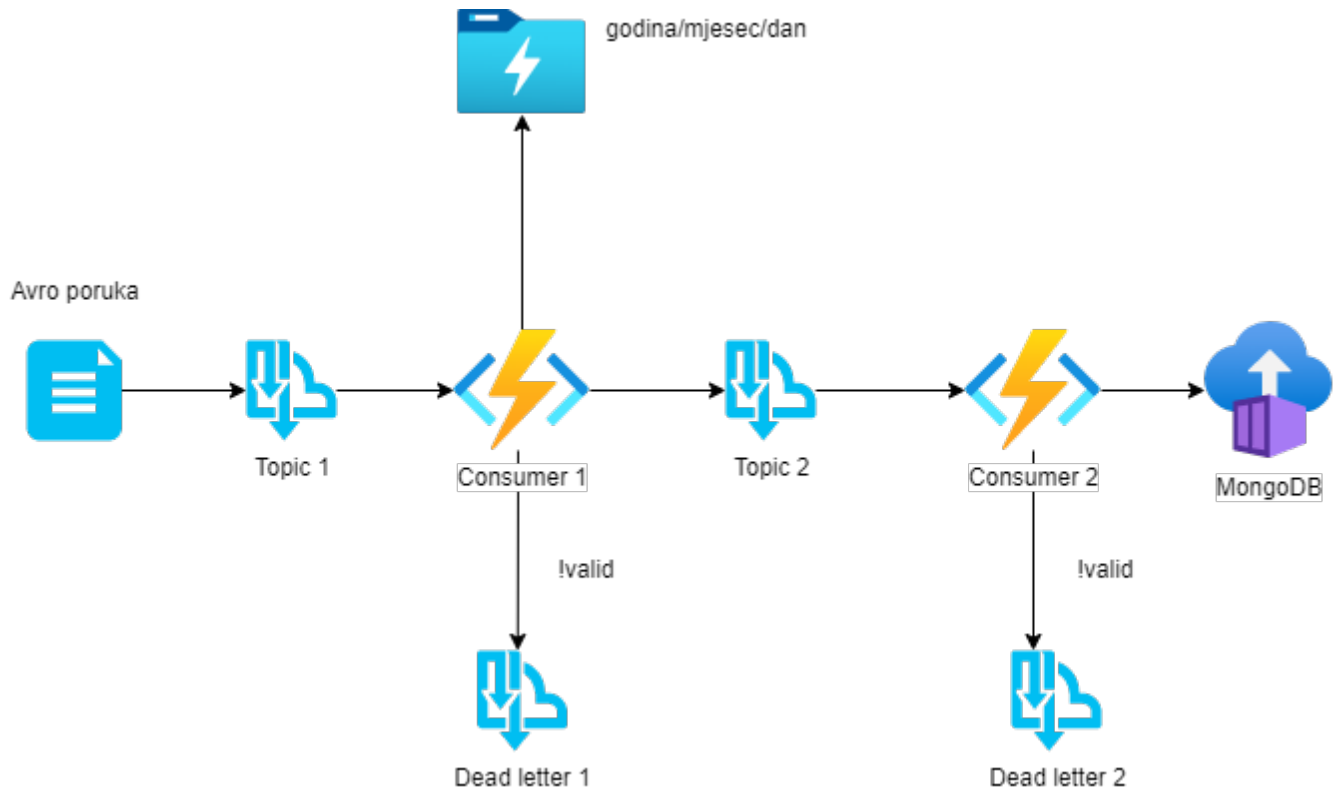
Consumer 1 čita Avro poruke s *Topic 1*, te radi validaciju Avro formata poruka. *Topic 1* predstavlja ulaznu točku u vaš podatkovni cjevovod. Poruke koje prođu validaciju šalju se na *Topic 2*, te se spremaju na *Data Lake* u izvornom formatu (engl. *raw format*). Struktura direktorija u koji se trebaju spremati poruke je u obliku *godina/mjesec/dan*. Spremanje poruka u izvornom formatu radi se u svrhu stvaranja sigurnosne kopije poruka. U slučaju da obrada poruke na *Consumeru*, u ovom slučaju *Consumeru 1*, završi nekom iznimkom, poruka se ponovno može dohvatiti u izvornom formatu s *Data Lake*-a, bez potrebe da *Producer* proces ili izvor koji šalje poruke mora ponovno

poslati tu istu poruku. Poruke koje nisu validne šalju se na *Dead letter 1* topic. Poruke koje su poslane na *Dead letter* se dalje mogu proučiti kako bi se uočila greška u poruci, te se dodatno mogu pokušati popraviti. U opsegu ovog projekta je samo da se neispravne poruke pošalju na *Dead letter*.

Consumer 2 čita poruke s *Topic 2* te radi validaciju poruka prema poslovnoj logici. S obzirom da prima poruke s *Topic 1*, sigurno je da su poruke u ispravnom formatu. Poruke koje su validne dalje sprema na *MongoDB* koristeći *lookup* funkcionalnost. U slučaju da u bazi postoji *Order* s istim *OrderID* kao primljena poruka, poruka koja sadrži točno jedan *OrderItem* pridodaje se na postojeći *Order* (join podataka). U slučaju da ne postoji *Order* s istim *OrderID*, kreira se novi *Order* i poruka se dodaje kao novi *Order* u bazu. Poruke koje nisu validne šalju se na *Dead letter 2* topic. *MongoDB* će se nalaziti na AKS clusteru kojeg ste konfigurirali u prvoj laboratorijskoj vježbi, te će proces deploy-a biti proveden kroz CI/CD pipeline koji također razvijate kontinuirano.

Napomena:

- Sljedeće vježbe su dosta kompleksne i podijelite se unutar tima tko će raditi određenu funkcionalnost
- Prilikom kreiranja resursa (Azure funkcije, *Topic* i sl.) držite se konvencije imenovanja s prethodnih laboratorijskih vježbi, kao biste na kraju lakše mogli ocijeniti jedni druge
- U daljnjem tekstu resursi su imenovani kako su označeni na dijagramu, kako biste znali na koji dio zadatka se upute odnose



1. Avro format poruka

- Proučite Avro format poruke.
Službena dokumentacija: <https://avro.apache.org/docs/current/index.html>
 - Pomoću Azure portala kreirajte Topic 1 i odgovarajući Subscription.
Napomena: Ako želite, ovdje možete koristiti i topic koji ste kreirali u prethodnoj laboratorijskoj vježbi.
 - Kreirajte Producera Avro poruka.
Za to možete iskoristiti kod iz prethodne laboratorijske vježbe kojim ste parsirane retke iz .csv datoteke slali kao poruke na topic. Modificirajte kod tako da prije slanja na topic, parsirane retke serijalizirate u Avro format.
 - Prijedlog: Za serijalizaciju i kasnije validaciju Avro poruka, možete koristiti sljedeći paket <https://github.com/hamba/avro>.
2. U ovoj laboratorijskoj vježbi krenut ćete s pisanjem koda za Consumera 1. Consumeri trebaju biti kreirani putem Azure funkcija (prisjetite se 1. laboratorijske vježbe). S obzirom da se kod piše u Go 1.18., trebate koristiti custom Go handlera za Azure: : <https://docs.microsoft.com/en-us/azure/azure-functions/functions-custom-handlers>. Custom handler je zapravo web server koji će obrađivati neke događaje (npr. dolazak poruke na *Topic*). Više o tome proučite na prethodnoj poveznici. Također, u sljedećem GitHub repozitoriju možete pronaći primjere implementacija custom handlera u Go-u koji vam mogu pomoći pri izradi ove vježbe: <https://github.com/Azure-Samples/functions-custom-handlers/tree/master/go>. Potrebno je ostvariti sljedeće funkcionalnosti:
- Nakon što poruka dođe na *Topic 1*, *Consumer 1* se treba automatski aktivirati, pročitati Avro poruku i validirati njenu shemu.

- U Visual Studio Code-u kreirajte Service Bus Topic trigger. Prvo klonirajte Git repozitorij koji ste kreirali u prvom dijelu laboratorijske vježbe. Kasnije sve promjene koje napravite dodajte prema uputama iz prvog dijela laboratorijske vježbe. Zatim unutar njega dodajte novu Azure funkciju:

- Jezik: Custom Handler
- Template: Azure Service Bus Topic trigger
- Naziv funkcije: po želji (npr. Tim1-Consumer1-labfer)

4. Service bus namespace: naziv namespace-a koji ste kreirali i koristili u prethodnoj i ovoj laboratorijskoj vježbi
5. Naziv topica: naziv topica kojeg ste kreirali/koristili u prethodnom zadatku
6. Naziv subscriptiona: naziv subscriptiona povezanog sa topicom koji koristite
- ii. Unutar repozitorija projekta dodajte novu *server.go* datoteku u kojoj će se nalaziti main funkcija i kod Consumera
 1. Prijedlog strukture projekta se nalazi na samom kraju teksta laboratorijske vježbe. U *server.go* datoteku dodajte sljedeći kod:

```
package main

import (
    "fmt"
    "log"
    "net/http"
    "os"
)

func handlerTopic1(w http.ResponseWriter, r *http.
Request) {
    // ovdje napišite kod za Consumer 1 koji e se izvršiti
    kad na topic 1 doe nova poruka
}

func main() {
    p, ok := os.LookupEnv("FUNCTIONS_CUSTOMHANDLER_PORT")
    if !ok {
        p = "8080"
    }
    addr := fmt.Sprintf(":%s", p)
    http.HandleFunc("/ServiceBusTopicTrigger1",
handlerTopic1)
    http.ListenAndServe(addr, nil)
    log.Printf("Listening on address: %s
/ServiceBusTopicTrigger1", addr)
}
```

- iii. Napišite funkciju koja će validirati Avro poruke, na način da će provjeravati imaju li odgovarajuću schemu. Funkcija će se pozivati unutar handler funkcije.
- iv. Nadopunite kod handler funkcije tako da ostvaruje navedenu funkcionalnost Consumera 1.
- v. **Napomena:** FUNCTIONS_CUSTOMHANDLER_PORT je port pomoću kojega Functions host poziva custom handlera i njega Azure Functions runtime postavlja u env varijablu - vi ga ne morate sami postavljati.
- b. *Ukoliko schema poruke nije validna, poruka se treba proslijediti na Dead letter 1 topic/queue.*
 - i. Pomoću Azure Portala kreirajte Dead letter 1 topic ili Dead letter 1 queue (Azure Storage queue).
 - ii. Nadopunite kod handler funkcije tako da ostvaruje navedenu funkcionalnost.
 - iii. Za prosljeđivanje poruke na drugi topic ili queue, koristite *output bindings*: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=csharp>.
- c. *Ukoliko je schema poruke validna, poruka se treba spremi na Azure Data Lake Gen2 u raw Avro formatu.*
 - i. U ovom dijelu zadatka možete koristiti spremište podataka koje ste kreirali u 1. laboratorijskoj vježbi. Direktorij unutar spremište podataka u koji ćete spremati poruke organizirajte prema sljedećoj strukturi: direktorij/godina /mjesec/dan.
 - ii. Nadopunite kod handler funkcije tako da ostvaruje navedenu funkcionalnost.
 - iii. Za spremanje poruke u Data Lake Gen2, također koristite *output binding*.
- d. *Nakon što je schema spremjena na Azure Data Lake Gen2, potrebno ju je proslijediti na Topic 2.*
 - i. Pomoću Azure Portala kreirajte Topic 2 i dodajte mu novi Subscription.
 - ii. Nadopunite kod handler funkcije tako da ostvaruje navedenu funkcionalnost.
 - iii. Za prosljeđivanje poruke na Topic 2, također koristite *output binding*.

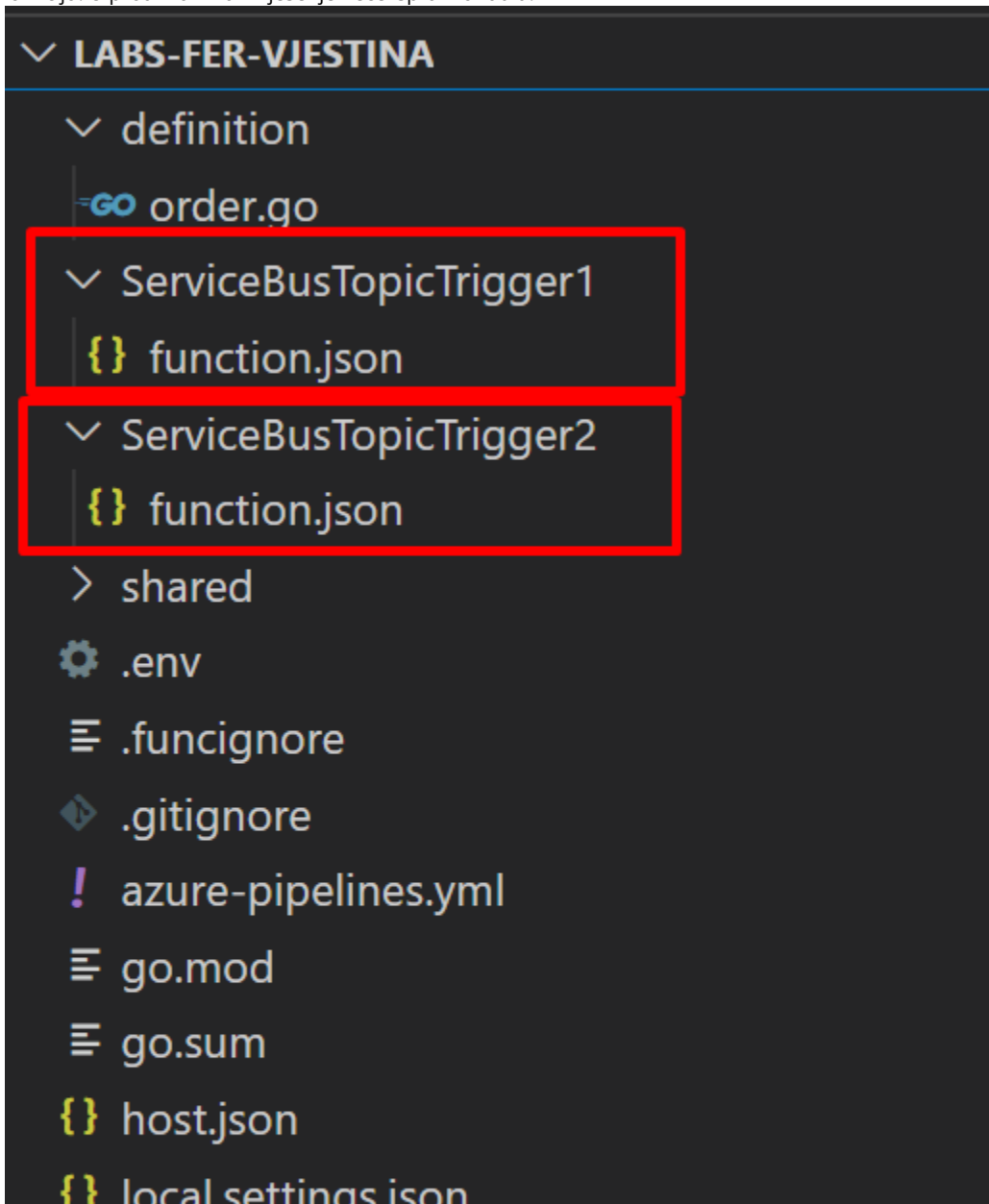
3. Pokrenite *Consumer 1* funkciju lokalno u *VS Code* terminalu naredbom `func start` te istestirajte funkcionalnost tako da šaljete *Avro* poruke na *Topic 1*. Provjerite spremaju li se validne poruke na *Data Lake* i *Topic 2*. Pokušajte poslati i poruku koja nije odgovarajućeg formata.


Dodatno


Sljedeća slika prikazuje prijedlog kako biste mogli organizirati direktorij svog programskog rješenja. Najvažniji dijelovi su:


- Datoteka `server.go` koja sadrži `main` funkciju koja je ulazna točka u vaš program, te funkcije koje će obrađivati zahtjev kada se dogodi neki događaj (engl. *event*), odnosno `trigger`
- Direktoriji `ServiceBusTopicTrigger1` i `ServiceBusTopicTrigger2` koji su direktoriji Azure funkcija. Ime direktorija odgovara imenu funkcije. Ako ste imenovali vaše *Consumer* funkcije drugačije, direktoriji će vam se drugačije zvati, tako da ovo shvatite samo kao primjer.
 - Svaka funkcija ima svoju datoteku `function.json` koja sadrži definiciju na koji događaj (engl. *event*) se funkcija okida
- `host.json` datoteka koja određuje na koji web server se šalju zahtjevi za obradom događaja
 - U prvoj laboratorijskoj vježbi ste također razvijali jednostavan *Custom Handler* u Go te modificirali `host.json`, stoga se možete podsjetiti
- Direktorij `definition` sadrži datoteku u kojoj su definicije Go struktura koje koristite u svome programskom rješenju
- Direktorij `shared` može sadržavati neke pomoćne funkcije

Prijedlog organizacije shvatite samo kao prijedlog da vam olakša početak izrade samog projekta, po potrebi možete modificirati strukturu kako vam odgovara. Jedino se pridržavajte principa da imate direktorij za svaku *Azure* funkciju u kojoj se nalazi `function.json` za točno tu funkciju. U protivnom vam rješenje neće ispravno raditi.



 README.md

 server.exe

 server.go