Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap

Institutt for datateknologi, elektroteknologi og realfag

Prosjektnavn
Systemdokumentasjon

Versjon 2.0

REVISJONSHISTORIE

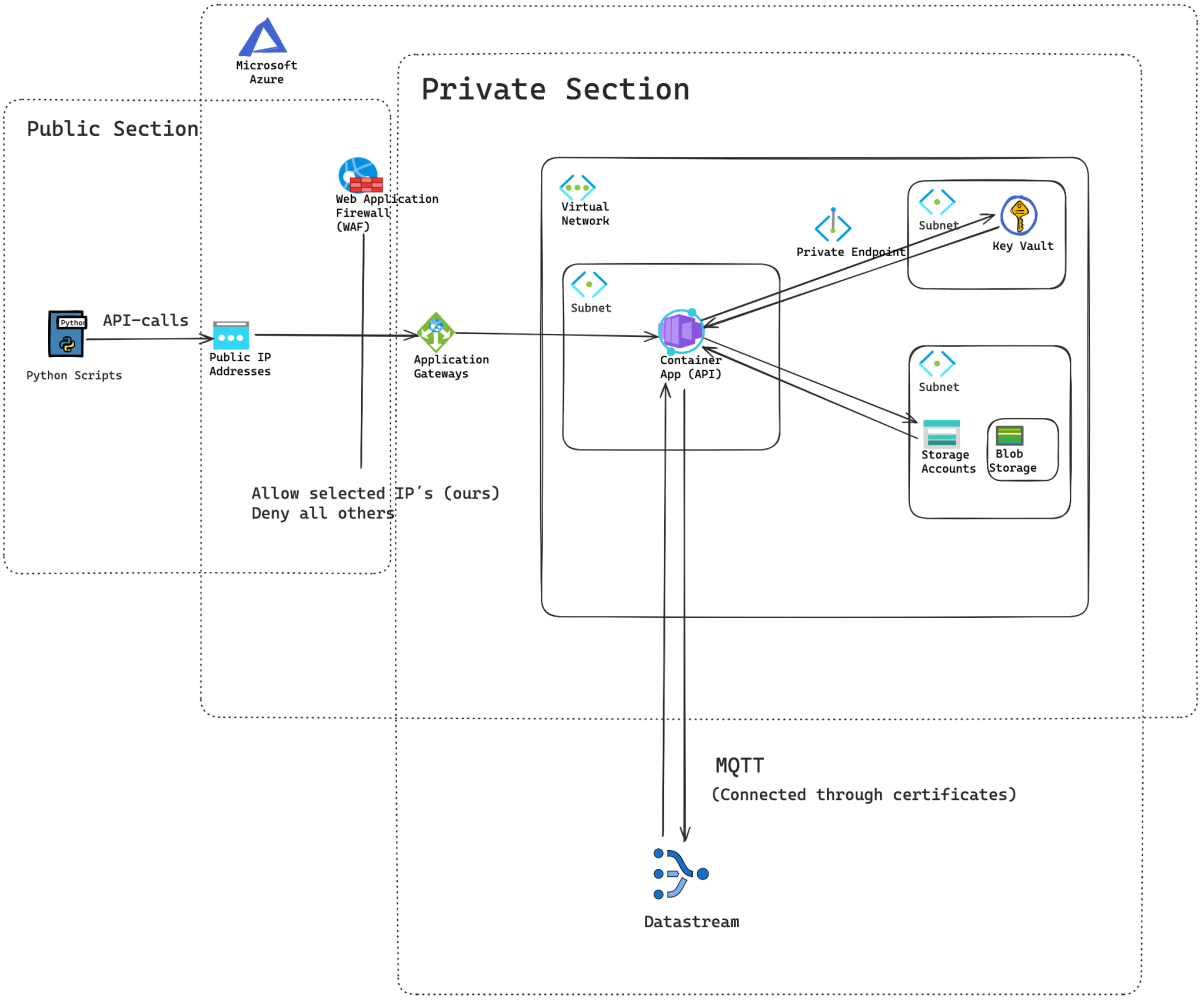
Dato	Versjon	Beskrivelse	Forfatter
10.04.2024	1.0	Fylt ut dokument utenom dokumentasjon av kildekode	Jonas B. Engen, Endre S. Nordtvedt, Jonas Vestbø
12.05.2024	2.0	Fylt ut dokumentasjon av kildekode	Jonas B. Engen

INNHOLDSFORTEGNELSE

1 INNLEDNING

Dokumentet er skrevet for å bedre forklare og vise hvordan systemarkitekturen for prosjektet ser ut. Her legges skjermbilder og enkle forklaringer ved.

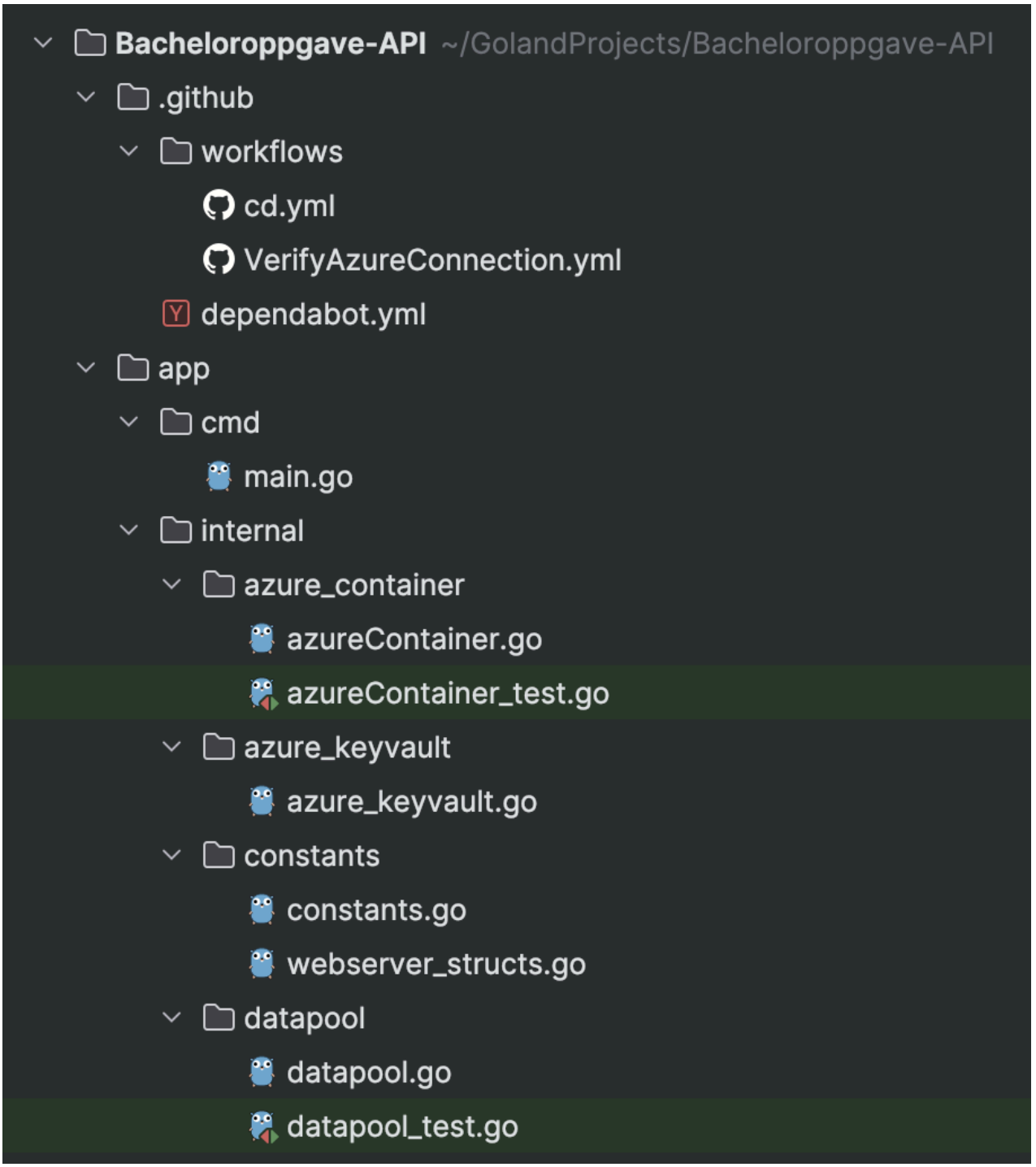
2 Arkitektur

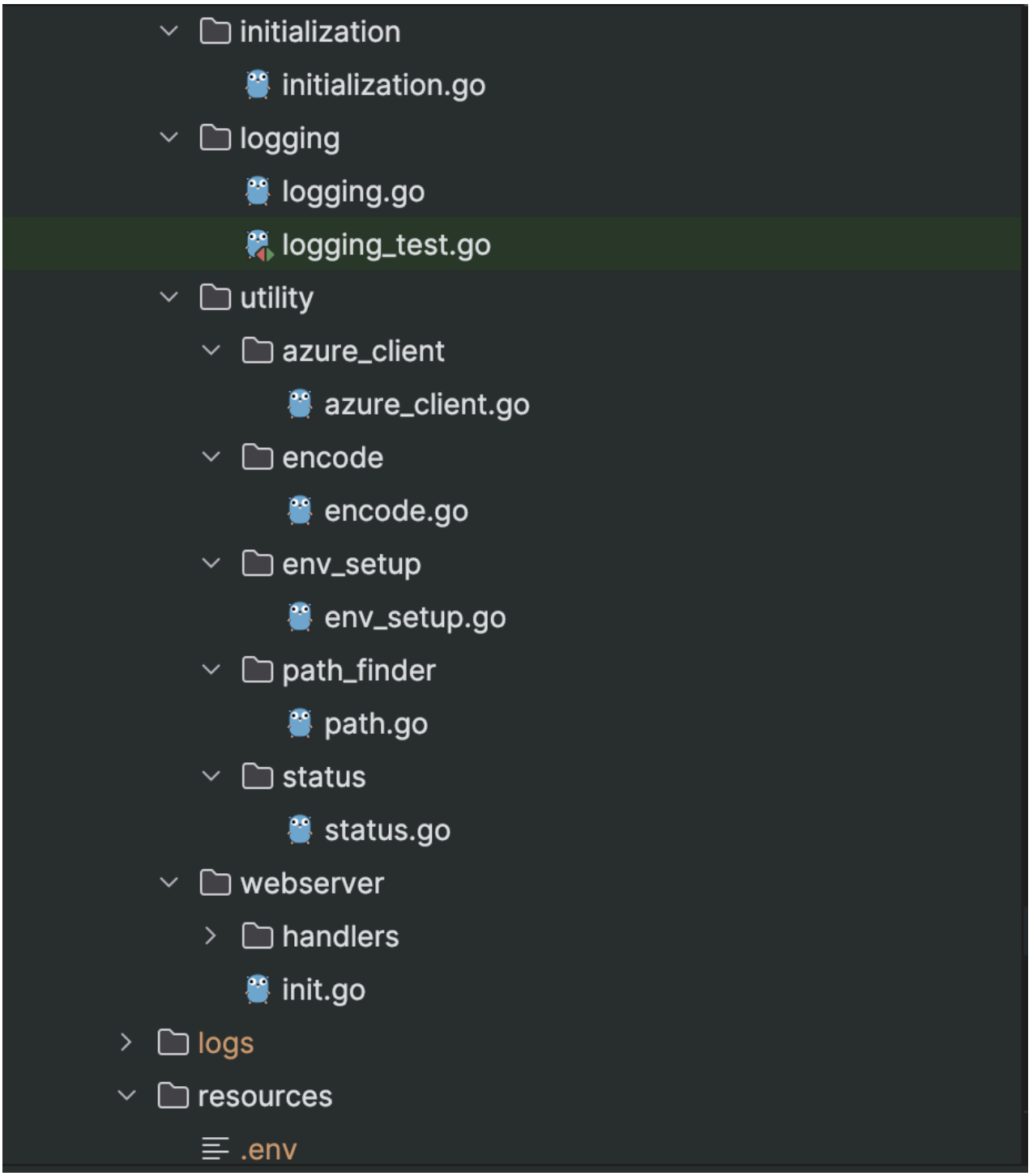


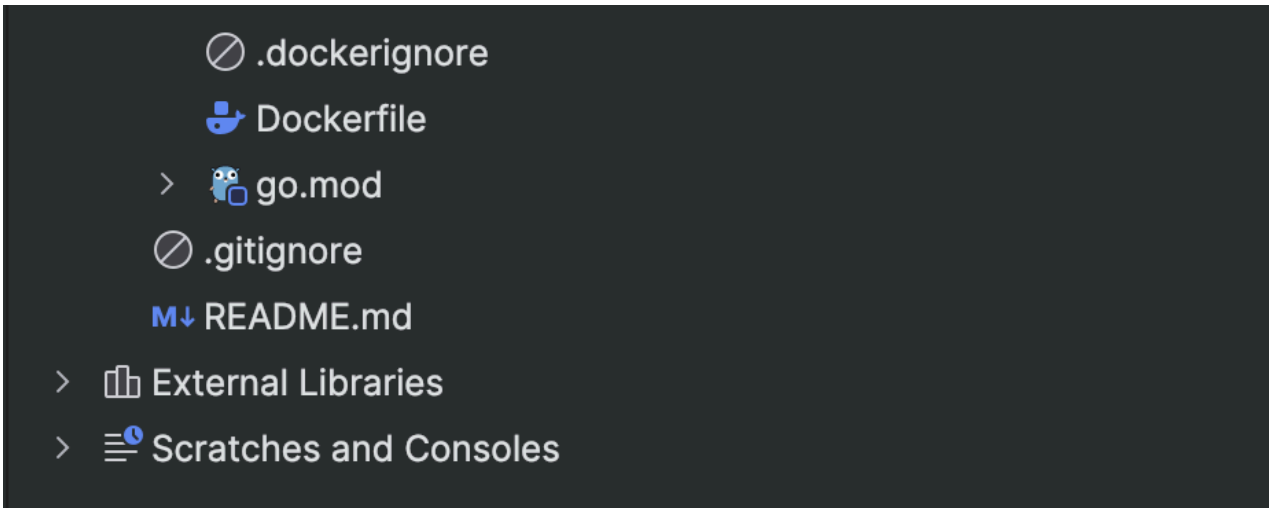
Struktur i og utenfor Azure med offentlig og privat seksjon.

3 PROSJEKTSTRUKTUR

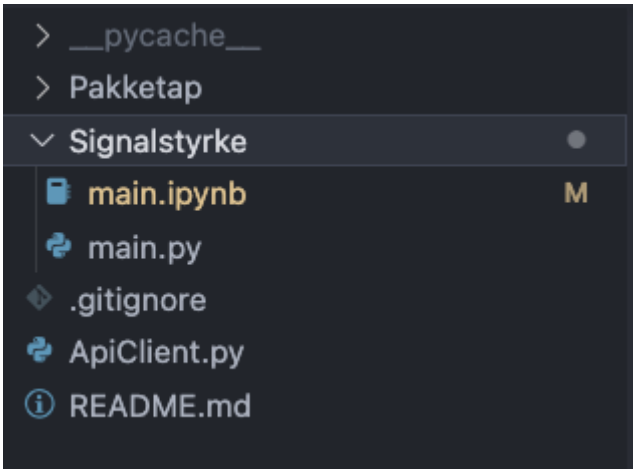
API programstruktur:



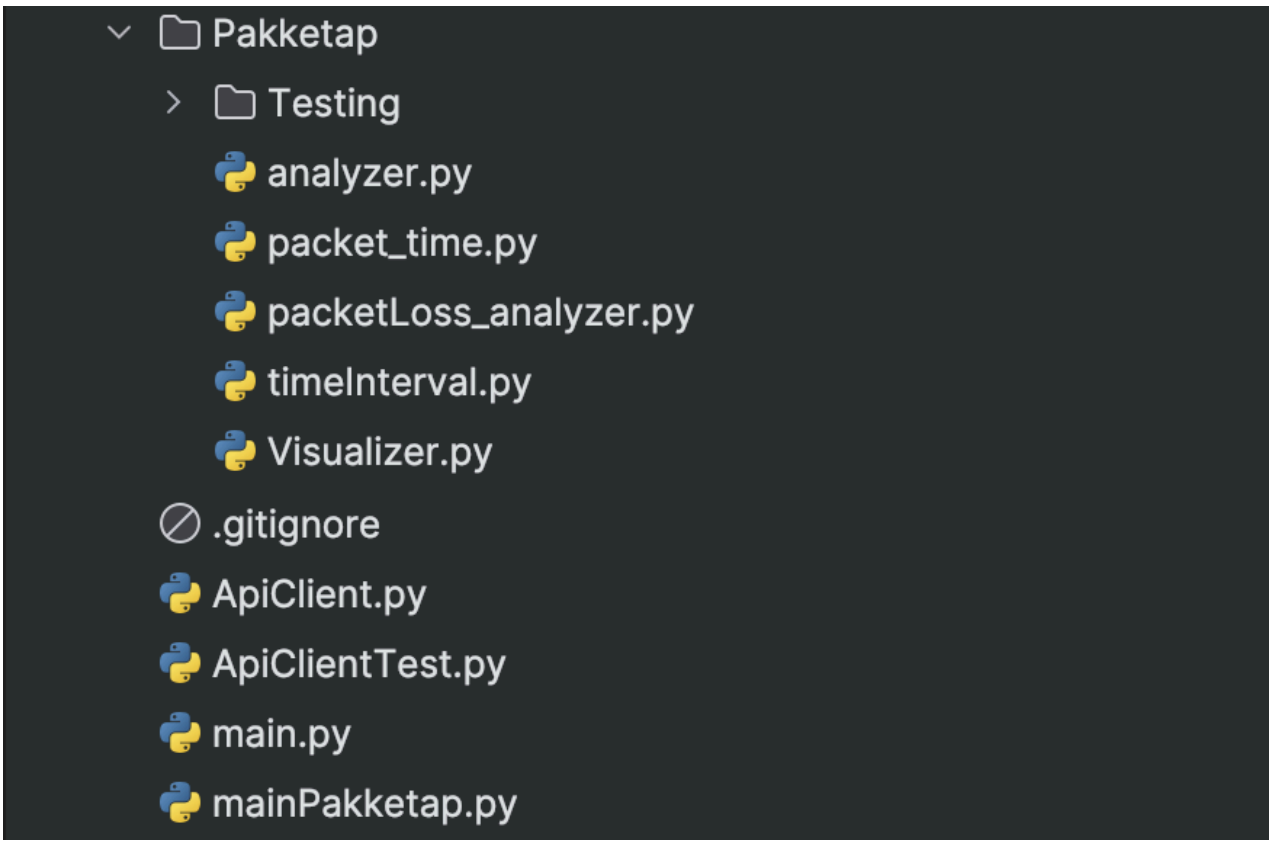

initialization
initialization.go
logging
logging.go
logging_test.go
utility
azure_client
azure_client.go
encode
encode.go
env_setup
env_setup.go
path_finder
path.go
status
status.go
webserver
handlers
init.go
logs
resources
.env



Signalstyrke evaluering arkitektur:

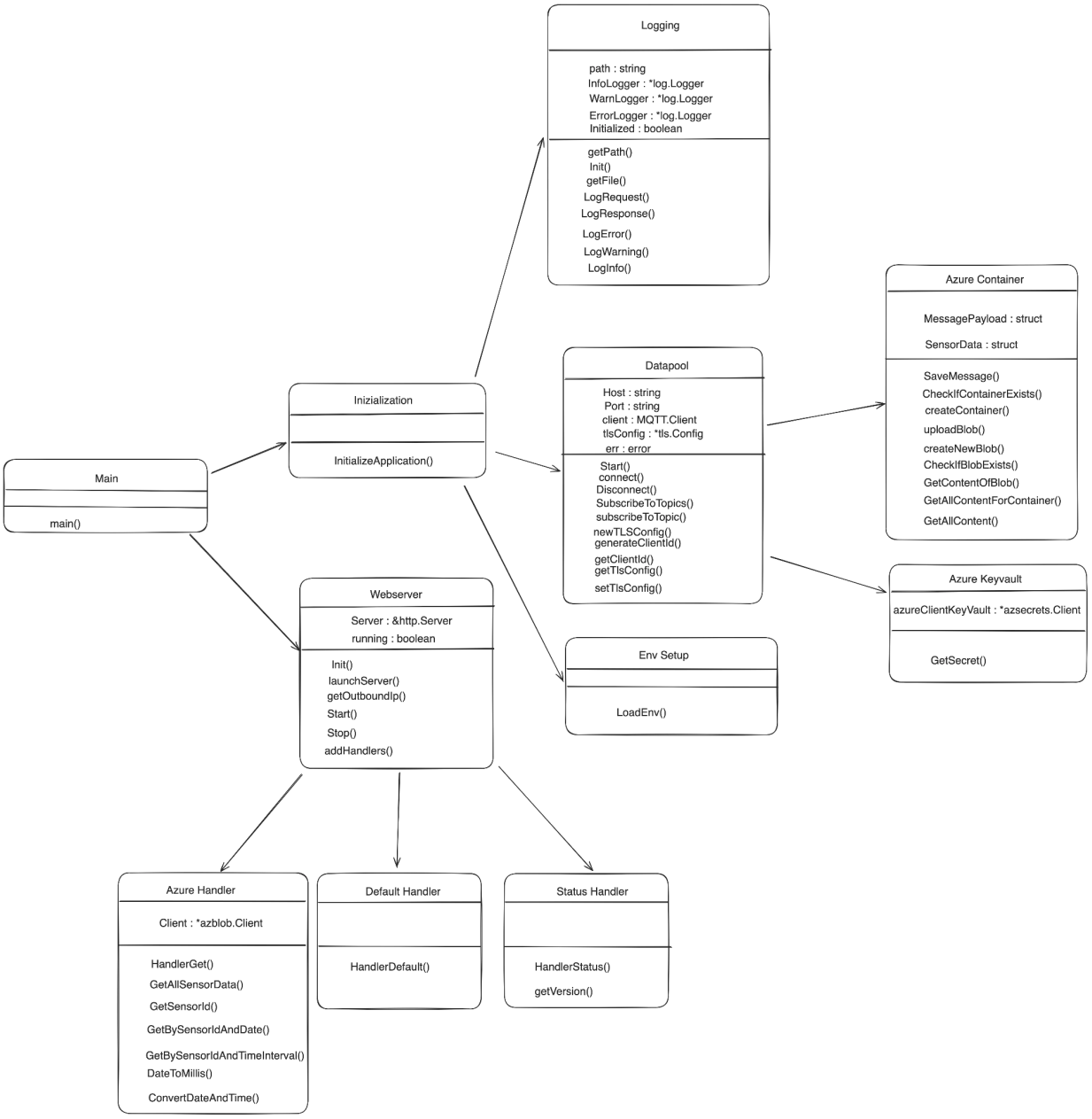


Pakketap arkitektur:

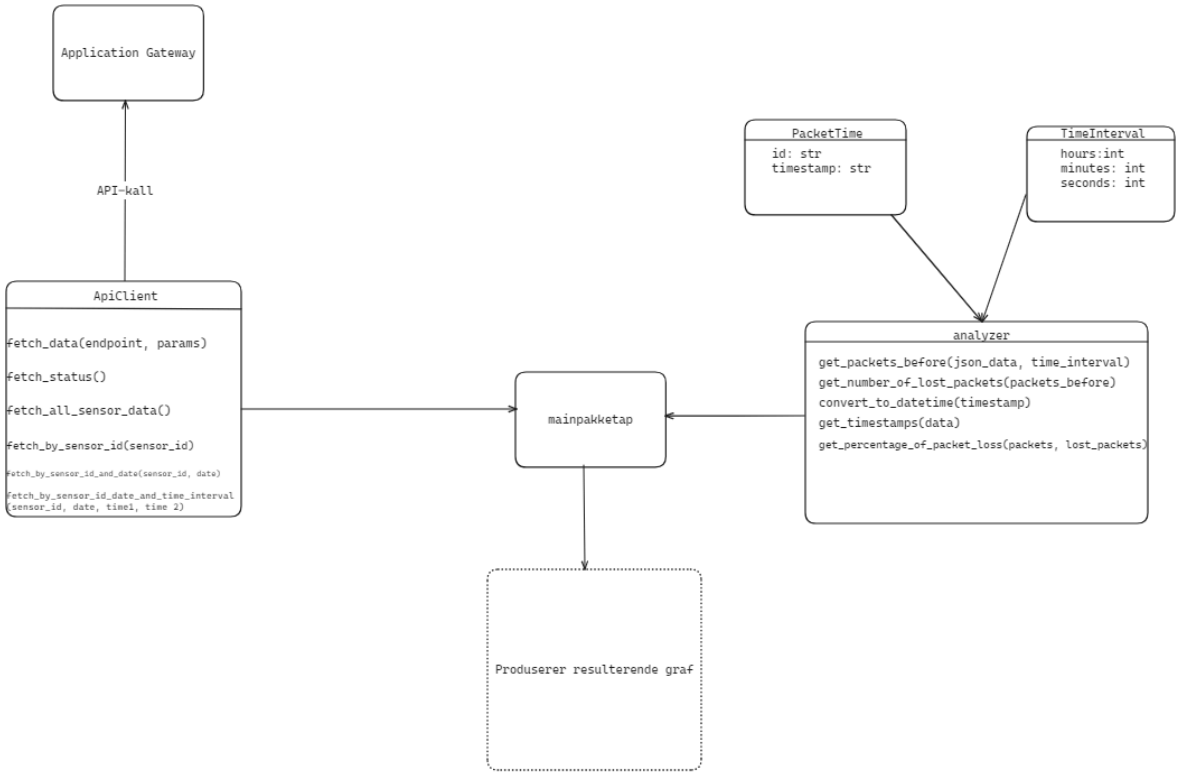


4 KLASSEDIAGRAM

Klassediagram API:

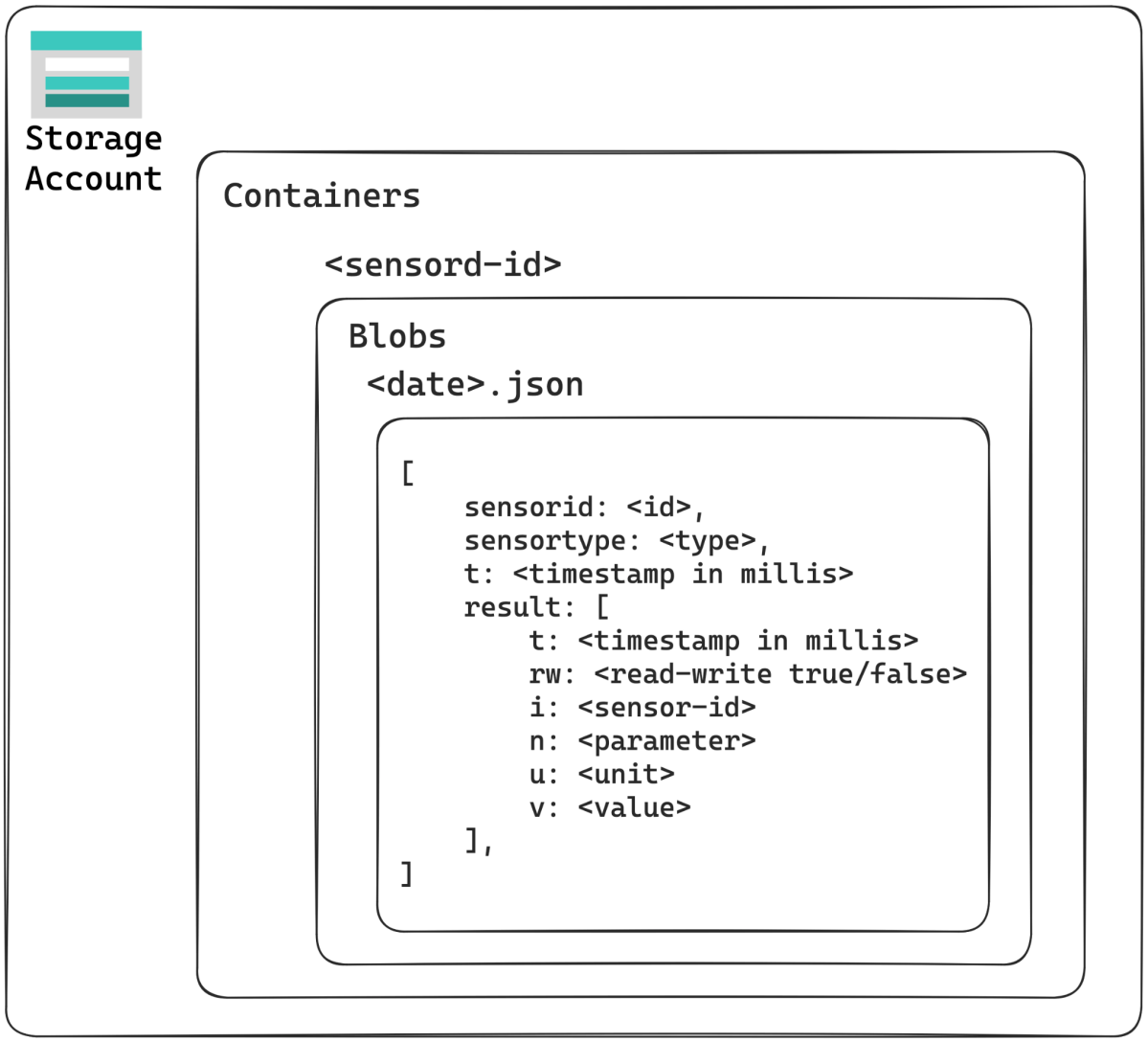


Klassediagram pakketap analyse:


Application Gateway
API-kall
ApiClient
fetch_data(endpoint, params)
fetch_status()
fetch_all_sensor_data()
fetch_by_sensor_id(sensor_id)
fetch_by_sensor_id_and_date(sensor_id, date)
fetch_by_sensor_id_date_and_time_interval(sensor_id, date, time1, time 2)
PacketTime
id: str
timestamp: str
TimeInterval
hours:int
minutes: int
seconds: int
analyzer
get_packets_before(json_data, time_interval)
get_number_of_lost_packets(packets_before)
convert_to_datetime(timestamp)
get_timestamps(data)
get_percentage_of_packet_loss(packets, lost_packets)
mainpakketap
Produserer resulterende graf

5 DATABASEMODELL

Vi har ikke valgt en database men en Storage Account i Azure. Dette minner litt om et filsystem mer enn en database.



Prosjektet har valgt løsningen ved å opprette mapper for hver sensor-id og inne i mappene så er det filer som tilsvarer hver dag, navngitt med datoen. I filene for hver dag så blir det lagt inn en JSON-liste hvor en inngang i listen tilsvarer en måling på et spesifikt klokkeslett på den dagen.

6 Server-tjenester

Gruppen har brukt en Application gateway i azure for å gjøre container appen tilgjengelig over nett. Det er satt på en WAF (Web Application Firewall) som bare tillater gitte ip-adresser

7 SIKKERHET

Som nevnt i punktet over så er det brukt en WAF for å begrense tilgang til Container appen (API-et), hvor det er lagt inn “allow” på gitte ip-adresser.

Det er også brukt et Key Vault i azure for å lagre sertifikatene som brukes for å koble seg på MQTT.

8 INSTALLASJON OG KJØRING

Pakker som trengs i API-et, ligger i “go.mod”-filen og installeres med “go mod download”

github.com/Azure/azure-sdk-for-go/sdk/storage/azblob v1.3.1
github.com/eclipse/paho.mqtt.golang v1.4.3
github.com/joho/godotenv v1.5.1
github.com/Azure/azure-sdk-for-go v68.0.0+incompatible // indirect
github.com/Azure/azure-sdk-for-go/sdk/azcore v1.9.2 // indirect
github.com/Azure/azure-sdk-for-go/sdk/azidentity v1.5.1 // indirect
github.com/Azure/azure-sdk-for-go/sdk/internal v1.5.2 // indirect
github.com/Azure/azure-sdk-for-go/sdk/keyvault/azsecrets v0.12.0 // indirect
github.com/Azure/azure-sdk-for-go/sdk/keyvault/internal v0.7.1 // indirect
github.com/AzureAD/microsoft-authentication-library-for-go v1.2.1 // indirect
github.com/golang-jwt/jwt/v5 v5.2.0 // indirect
github.com/google/uuid v1.6.0 // indirect
github.com/gorilla/websocket v1.5.1 // indirect
github.com/kylelemons/godebug v1.1.0 // indirect
github.com/pkg/browser v0.0.0-20240102092130-5ac0b6a4141c // indirect
golang.org/x/crypto v0.18.0 // indirect
golang.org/x/net v0.20.0 // indirect
golang.org/x/sync v0.1.0 // indirect
golang.org/x/sys v0.16.0 // indirect
golang.org/x/text v0.14.0 // indirect

For å kjøre API-et (og datastrøm) på egen maskin går man i terminal og går til prosjektet. Deretter skriver man “cd /app” og kjører deretter kommandoen “go run ./cmd”

For å kjøre scriptet som produserer graf som evaluerer pakketap brukes følgende kommando: python3 mainPakketap.py dersom du befinner deg i prosjekt rot mappen. Bibliotekene som kreves for å kjøre denne er:
requests, datetime og typing. Disse kommer ofte standard med alle python installasjoner. Men dersom de mangler kan de installeres med følgende kommando: “pip install requests datetime typing”. I tillegg er det brukt matplotlib, for å installere dette biblioteket brukes kommandoen “pip install matplotlib". Det finnes også en requirements.txt fil som er en liste over alle bibliotekene nødvendig for å kjøre scriptet. For å laste ned alle bibliotekene i denne fila kan kommandoen “pip install -r requirements.txt”

For å kjøre scriptet som skriver ut kommentarer og evaluerer SNR, rssi og SF kjøres med følgende kommando i terminalen: python signalstyrke/main.go. Ellers kan en trykke på run knappen med et “play” ikon som oftest oppe i høyre hjørnet for å kjøre scriptet.

Biblioteker importert til notebook som visualiserer signalstyrke:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
```

For å kjøre notebooken kan en initialisere dette med å skrive: jupyter notebook signalstyrke/main.ipynb i terminalen for å få notbooken i en web browser. Deretter kan en trykke på “run all” for å kjøre alle celler eller trykke på første celle og trykke run eller shift+enter for tastatursnarvei. Dette vil kjøre cellen du står i og hoppe videre til neste celle.

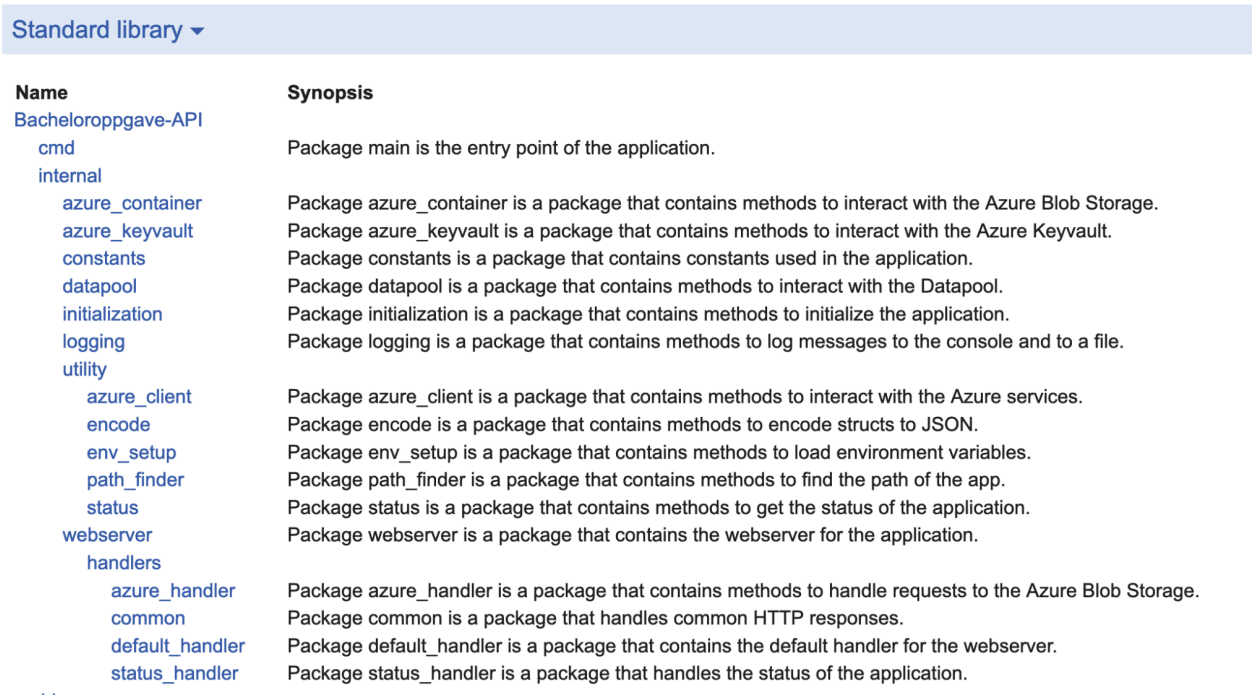
Pakkene som må installeres for å kjøre scriptet som analyserer pakketap, installeres med ‘pip install’:
matplotlib
requests
datetime.

9 DOKUMENTASJON AV KILDEKODE

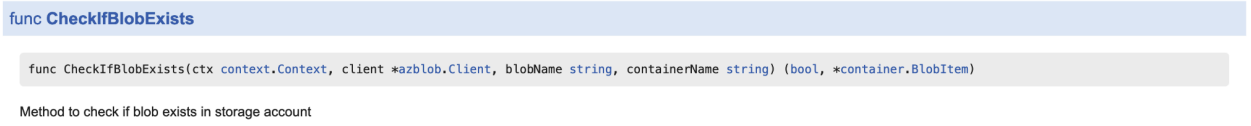
API (Go applikasjonen):

Her er hvordan man genererer kildekode for go applikasjonen:

1. Naviger til prosjektet også kjør 'cd /app' i kommandolinje
2. 'go get golang.org/x/tools/cmd/godoc' kjøres i kommandolinje
3. 'export PATH="$GOPATH/bin:$PATH"' kjøres i kommandolinje (kan hoppes over hvis punkt 3 fungerer med en gang)
4. 'godoc -http=:6060' kjøres i kommandolinje og gjør at man hoster en lokal webserver
5. Besøk 'http://localhost:6060/pkg/?m=all' i nettleser (?m=all brukes for å vise alle interne pakker. Disse ville ikke blitt vist ellers)



Figur 9.1: Skjermbilde av dokumentasjonen som er blitt generert



Figur 9.2: Skjermbilde av dokumentasjon av en metode som er blitt generert

Python:

Kan gjøres med ‘pydoc -p 8080’, funker ikke helt som ønsket på grunn av manglende tilgang til Azure ressurser.

10 KONTINUERLIG INTEGRASJON OG TESTING

Kontinuerlig Integrasjon:

På pull-requests til alle branches så kjører koden gjennom en workflow som heter build-app som installerer avhengigheter, kjører build, kjører "go vet -v ./…" som sjekker for errors og fjerner mistenkelig kode, så logger den inn på azure og åpner opp sin egen ip (nødvendig for testene), så kjører den gjennom testene med "go test -v -p 1 ./…" og til slutt fjerner ip-en sin fra azure.

På push til master-branchen så kjører den i gjennom det samme som over, men den kjører også andre workflows som å lage en release, pakke appen og deploye appen til azure.

Create-release lager bare en release tag som brukes når man pakker inn appen. I package-app blir mye av det samme som build-app men trenger i tillegg noen hemmeligheter og en Dockerfile.

Når dette er ferdig vil deploy-app deploye appen til Azure med gitte parametere.

Tester som er laget:

I API-et er stort sett alt testet. Testene som bruker hemmeligheter fra keyvault (i azure_keyvault_test.go og i datapool_test.go) er kommentert ut da de ikke funker i github action med environment variabler (disse funker lokalt).

11 REFERANSER