

**ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**



BLM3522

Bulut Bilişim Ve Uygulamaları

Ara Rapor

ENES VAROL - 18290068 - Github: <https://github.com/enesvarol189/BLM3522>

EMİN AYDIN - 22290099 - Github: <https://github.com/emnydn/BLM3522>

SEMİH BERKAN OKUTAN - 22290240 - <https://github.com/semihberkanokutan/BLM3522>

12/05/2025

ANLATIM VİDEOLARI VERİLEN GİTHUB REPO'LARININ İÇİNDEDİR

1. Gerçek Zamanlı Veri Akışı

Proje Amacı

Bu projede, endpoint üzerinden gerçek zamanlı veri toplanması, bu verilerin bulut platformuna aktarılması ve analiz edilmesi hedeflenmiştir. WebSocket kullanılarak veri akışı sağlanmış, veriler Redis üzerinde geçici olarak tutulmuş ve Google Cloud Pub/Sub üzerinden Dataflow BigQuery'ye aktarılmıştır.

1. Kullanılan Teknolojiler:

Backend Dili: Java (Spring Boot), Python

Protokol: WebSocket

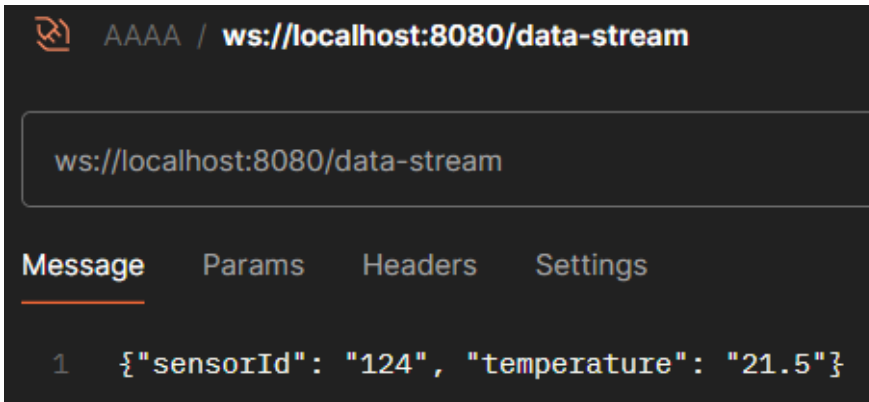
Veritabanı: Redis

Bulut Platformu: Google Cloud (Pub/Sub, Dataflow, BigQuery)

2. Sistem Mimarisi

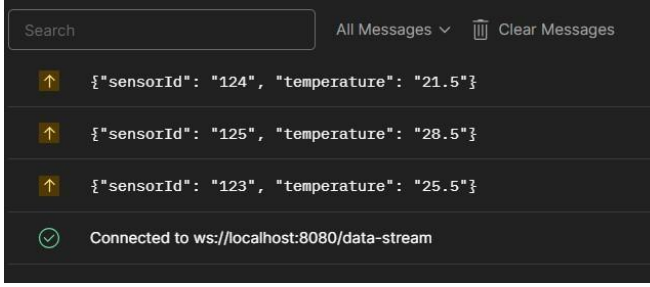
2.1. Veri Toplama Katmanı

Proje herhangi bir veri sunucusuyla entegre olacak şekilde hazırlanmıştır. WebSocket kullanılarak yazılan bu servis canlı olarak veri toplayabilir ve alınan bu verileri önce Redis'e kaydeder ve aynı anda Google Cloud Pub/Sub'ta ayrılan topic'e publish eder.



Şekilde görüldüğü gibi Postman üzerinden WebSocket'in endpoint'ine connection isteği atıldıktan sonra istenildiği kadar veri akışı sağlanabilir.

Örnek datalar:



WebSocket endpoint'i:

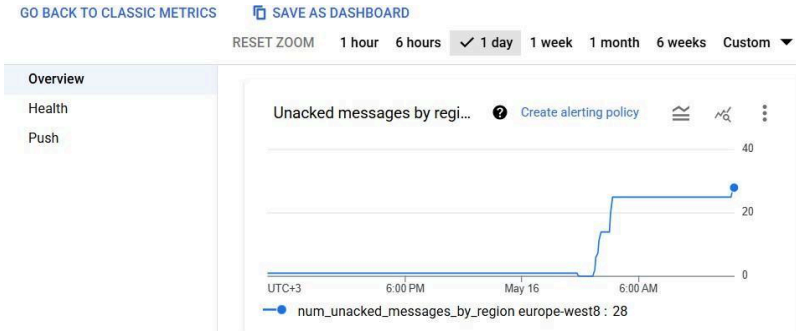
```
@Override
protected void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception {
    String data = message.getPayload();
    System.out.println("Received data: " + data);

    redisService.saveData(key:"latest-data", data);

    pubSubService.publishMessage(topic:"projects/upheld-chalice-459519-s3/topics/data-topic", data);
}
```

2.2. Bulut ve Veri İşleme Katmanı

Verilerin Pub/Sub topic'ine gönderilmesinin ardından, verilerin toplanması için bir subscription oluşturulur.



Şekilde verilerin işleme grafiğini görüyoruz. WebHook endpoint'i üzerinden alınan mesajlar doğrudan burada görülür.

Bu noktadan sonra verilerin BigQuery'ye yazılmasının 2 yöntemi vardır, projede her 2 yöntem kullanılmıştır ancak sadeliği açısından 2 yöntemle devam edilmiştir:

- Dataflow Pipeline'ı yazmak: Projenin Dataflow Pipeline katmanı için Python tercih edilmiştir. Bu Pipeline, verilen subscription üzerinden aldığı verileri ayıklayarak, BigQuery tablo yapısına hazır hale getirir, ve ardından bu verileri, verilen tabloya sırasıyla yazar.

```
streaming_pull_future = subscriber.subscribe(subscription_path, callback=pubsub_callback)
print(f"Listening for messages on {subscription_path}...")
```

- Doğrudan Pub/Sub topic Subscription'ı BigQuery'e bağlamak: Bu yöntem daha basittir, çünkü Dataflow için 2. Bir servis çalıştırmaya gerek kalmaz. Burada yapılan; subscription'ın, push operasyonu tipini doğrudan BigQuery'e yazacak şekilde ayarlamaktır. Bunun için ise subscription'dan sorumlu gCloud servis hesabının BigQuery'e yazma ve okuma yetkisi olmalıdır.

Delivery type ?

☐ Pull

☐ Push

☒ Write to BigQuery

A variant of the push operation. Select this option if you want Pub/Sub to deliver messages directly to an existing BigQuery table. [Learn more](#)

Project *

upheld-chalice-459519-s3

[BROWSE](#)

Dataset *

my_dataset

Table *

my_table

Bu proje kapsamında Python ile Dataflow Pipeline'ı yazılsa da sadeleği açısından ikinci yöntem tercih edilmiştir.

2.3. Analiz ve Görselleştirme Katmanı

Son aşama olarak, BigQuery'e yazılan verilen listelenebilir veya BigQuery'nin sunduğu herhangi bir data grafiği methoduyla görselleştirilebilir.

The screenshot shows the Google Cloud BigQuery console interface. At the top, there's a toolbar with buttons for 'Run', 'Save', 'Download', 'Share', and 'Schedule'. Below the toolbar, a SQL query is entered in a text area:

```
1 SELECT *
2 FROM `upheld-chalice-459519-s3.my_dataset.my_table`
3 LIMIT 1000
```

Below the query, a status message indicates: "This query will process 0 B when run." and "Processing location: US". At the bottom, there's a section titled "Query results" with tabs for "Job information", "Results", "Chart", "JSON", "Execution details", and "Execution graph". The "Chart" tab is currently selected.

2. Çift Katmanlı Web Uygulaması

Proje Amacı

Bu projemizde bir not tutma uygulaması yaptık. Frontend kısmında React, backend kısmında Django, veritabanı için PostgreSQL ve bulut kısmında da Google Cloud kullandık.

Projemizi frontend ve backend olmak üzere iki bölüme ayırdık. Daha sonra backend kısmını yapmaya başladık. Backend klasörünün içinde yine backend adında bir Django projesi oluşturduk, ardından notes adında bir app oluşturduk. Models.py içerisinde, title ve content olmak üzere veritabanımızın elemanlarını belirledik.

```
from django.db import models

# Create your models here.

class Note(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
```

Rest-framework ve bu modeli python nesnesinden JSON formata dönüştürebilmek için bir serializer oluşturmamız gerekiyordu. Bunun için serializers.py adında yeni bir python dosyası oluşturduk ve burda modeli serialize ettik.

```
from rest_framework import serializers
from .models import Note

class NoteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Note
        fields = '__all__'
```

Daha sonra API endpoint'lerini tanımlamak için views.py dosyası içinde viewset oluşturduk.

```
from rest_framework import viewsets
from .models import Note
from .serializers import NoteSerializer

# Create your views here.

class NoteViewSet(viewsets.ModelViewSet):
    queryset = Note.objects.all()
    serializer_class = NoteSerializer
```

Ardından backend projemizdeki urls.py dosyasında urlpattern ve router oluşturduk.

```
from django.conf import settings
from django.contrib import admin
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from notes.views import NoteViewSet
from django.conf.urls.static import static

router = DefaultRouter()
router.register(r'notes', NoteViewSet)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include(router.urls)),
]

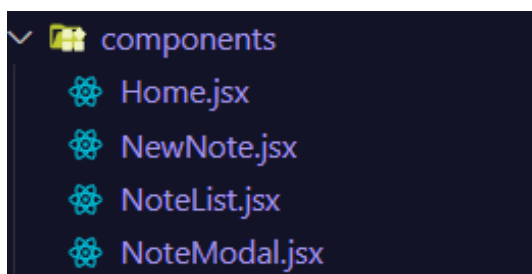
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Bütün bu uygulamaları django'nun işleyebilmesi için settings.py adında INSTALLED_APPS listesine rest-framework ve notes uygulamamızı ekledik.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'notes',
]
```

Tüm bunların ardından “python manage.py makemigrations” ve “python manage.py migrate” diyerek veritabanımızı ve yapısını migrate ettik.

Sırada frontend kısmını hazırlamaya başladık. Vite kullanarak bir react projesi oluşturduk. Componentlerimizi; anasayfa için Home.jsx, yeni bir not eklemek istediğimiz zaman gelecek olan form NewNote.jsx, oluşturduğumuz notları listeleyip ekranda görüntülemek için NoteList.jsx ve bir nota tıkladığımızda modal olarak ekranda görünmesi için NoteModal.jsx olarak components klasörü altında belirledik.



İlk olarak not formunu oluşturmaya başladık.

```
export default function NewNote({initialNote = null}) {
  return (
    <FormBuild onSubmit={handleSubmit}>
      <h1>{initialNote ? 'Edit the Note' : 'Add New Note'}</h1>
      <input
        type="text"
        ref={title}
        placeholder='Title'
      />
      <textarea
        ref={content}
        placeholder='Content'
      />
      <div>
        <button
          className='cancelBtn'
          type='button'
          onClick={handleCancel}
        >
          Cancel
        </button>
        <button
          className='saveBtn'
          type='submit'
        >
          Save
        </button>
      </div>
    </FormBuild>
  )
}
```

İnputlara girilen değerleri tutmak için useRef hook'unu kullandık. Form submit edildiğinde çalışması için onSubmit için handleSubmit adında bir fonksiyon oluşturduk.

```
const title = useRef()
const content = useRef()
const navigate = useNavigate()

const handleSubmit = async (e) => {
  e.preventDefault();
  if (!title.current.value.trim() || !content.current.value.trim()) return;
  try {
    await axios.post('https://notesapp-459915.uc.r.appspot.com/api/notes/',
      {title: title.current.value, content: content.current.value});
  } catch (err) {
    console.error('Not eklenemedi:', err);
  }
  title.current.value = ''
  content.current.value = ''
  navigate('/')
}

const handleCancel = () => {
  navigate('/')
}
```

handleSubmit fonksiyonu axios kullanarak veritabanımıza girdiğimiz title ve content değerlerini ekliyor. Burada her işlemin sıralı ve hatasız olarak ilerlemesi için asenkron fonksiyon kullandık. Navigate('/') kısmı bizi form submit olduktan sonra ana sayfaya yönlendiriyor. handleCancel fonksiyonu da cancel tuşuna bastığımız zaman çalışıyor ve hiçbir şey yapmadan direk bizi ana sayfaya yönlendiriyor.

Add New Note

Content

Cancel

Save

Formumuzun stili de bu şekilde. Bu projede bütün css stilleri için styled-components kullandık. Bu hem daha kolay css yazmamızı hem de stilleri react component'lerine daha kolay entegre edebilmemizi sağladı.

```
const FormBuild = styled.form`
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);
  background-color: #dedede;
  max-width: 600px;
  width: 600px;
  display: flex;
  flex-direction: column;
  gap: 1.2rem;
  padding: 24px;
  box-shadow: rgba(0, 0, 0, 0.35) 0px 5px 15px;
  border-radius: 5px;

  & h1 {
    text-align: center;
  }

  & input, textarea {
    outline: none;
    padding: 8px;
    font-size: 16px;
    border: 2px solid #6c757d;

    &:focus {
      border-color: #007bff;
    }
  }

  & .cancelBtn {
    background-color: none;
    color: #6c757d;
    border: 1px solid #6c757d;

    &: hover {
      background-color: #6c757d;
      color: #fff;
    }
  }

  & .saveBtn {
    background-color: #007bff;
    border: 1px solid #007bff;
    color: #fff;

    &: hover {
      background-color: #0069d9;
      color: #fff;
    }
  }
`
```

Styled-components kullanımı.

Daha sonra NoteModal component'ini oluşturduk.

```
import styled from "styled-components"
import { useRef, useImperativeHandle } from "react"

const NoteModalBuild = styled.dialog` ...

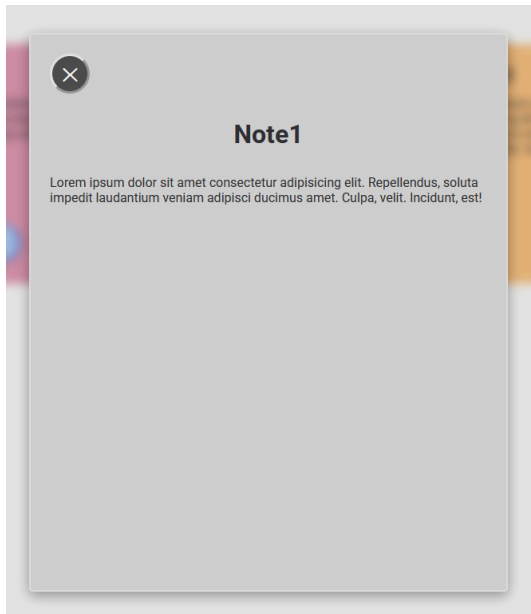
export default function NoteModal({modalTitle, modalContent, ref}) {
  const dialog = useRef()

  useImperativeHandle(ref, () => ({
    open: () => dialog.current.showModal(),
    close: () => dialog.current.close(),
  }));

  const handleClose = () => {
    dialog.current.close()
  }

  return (
    <NoteModalBuild ref={dialog} >
      <div>
        <button onClick={handleClose}>X</button>
        <h2>{modalTitle}</h2>
        <p>{modalContent}</p>
      </div>
    </NoteModalBuild>
  )
}
```

Bu component de useImperativeHandle kullanıyoruz ki diğer component'lerden dialogun showModal ve close fonksiyonlarını kullanabilelim. Ayrıca react'ın yeni sürümünde artık forwardRef kullanmamıza gerek yok, ref'i direk props olarak verebiliyoruz.



Modal yapımız da bu şekilde.

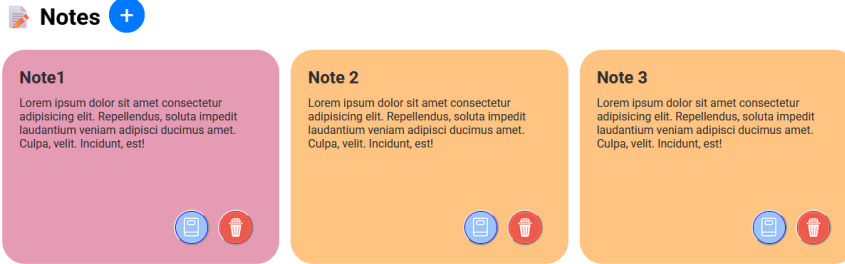
Sırada NoteList component'ini oluşturmak kaldı. O da aşağıdaki şekilde:

```
const NoteList = forwardRef(function NoteList({ notes, onDelete, onRead }, ref) {
  const colors = ['#BDDDE4', '#E69DB8', '#ADB2D4', '#C1CFA1', '#FFC785']

  return (
    <NoteListBuild>
      {Array.isArray(notes) &&
        notes.map((note) => (
          <NoteBuild
            key={note.id}
            style={{backgroundColor: colors[Math.floor(Math.random() * colors.length)]}}
            ref={ref}
          >
            <h2>{note.title}</h2>
            <p>{note.content}</p>
            <div className="btnGroup">
              <button className="readBtn" onClick={() => onRead(note.id)}>
                <img src={BookIcon} alt="" />
              </button>
              <button onClick={() => onDelete(note.id)} className="deleteBtn">
                <img src={TrashCanIcon} alt="" />
              </button>
            </div>
          </NoteBuild>
        ))}
    </NoteListBuild>
  );
});

export default NoteList
```

Burada notes array'ini map ile döndürerek bütün notları div tag'i içinde ekliyoruz. Notların arka plan rengi de rastgele colors dizininden seçiliyor. Burada forwardRef kullanmamıza gerek yoktu, ref'i props olarak verebilirdik fakat bu şekilde de kullanmak istedik.



Notlarımız da ekranda bu şekilde görünüyor. Kırmızı butona bastığımızda veri tabanından not siliniyor, mavi butona bastığımızda, daha önceki sayfalarda anlattığımız modal şeklinde not ekrana yansıyor.

Son olarak Home component'i kaldı. Bu component'te daha önce oluşturduğumuz bütün componentleri çağırıyoruz.

```

return (
  <HomeBuild>
    <div className='homeTop'>
      <h1> 📝 Notes</h1>
      <button onClick={handleNavigate}>+</button>
    </div>
    <NoteList
      notes={notes}
      onDelete={deleteNote}
      onRead={handleOpenNote}
      ref={activeNoteRef}
    />
    <NoteModal
      modalTitle={activeNote.activeTitle}
      modalContent={activeNote.activeContent}
      ref={dialog}
    />
  </HomeBuild>
);
}

export default App;

```

Bu component de fetchNotes ve deleteNote adında iki fonksiyonumuz var. fetchNotes, veritabanından notları çekiyor ve deleteNote da veritabanından notu siliyor. Bu işlemler için yine asenkron fonksiyon ve axios kullanıyoruz.

```

const fetchNotes = async () => {
  try {
    const res = await axios.get('https://notesapp-459915.uc.r.appspot.com/api/notes/');
    setNotes(res.data);
  } catch (err) {
    console.error('Couldnt fetch the data:', err);
  }
};

const deleteNote = async (id) => {
  try {
    await axios.delete(`https://notesapp-459915.uc.r.appspot.com/api/notes/${id}/`);
    fetchNotes();
  } catch (err) {
    console.error('Couldnt delete the data:', err);
  }
};

```

Notları useState hook'unu kullanarak set ediyoruz.

```
const [notes, setNotes] = useState([]);
```

Bu component de ayrıca modal modal fonksiyonumuz da bulunuyor.



```
const [activeNote, setActiveNote] = useState({
  activeTitle: null,
  activeContent: null,
})
const navigate = useNavigate()
const dialog = useRef()
const activeNoteRef = useRef()



const handleOpenNote = async (id) => {
  try {
    const res = await axios.get(`https://notesapp-459915.uc.r.appspot.com/api/notes/${id}/`);
    console.log(res.data)
    setActiveNote({
      activeTitle: res.data.title,
      activeContent: res.data.content
    })
    dialog.current.open()
  } catch (err) {
    console.error('Couldnt fetch the data:', err);
  }
}
```

Tıkladığımız nota göre activeTitle ve activeContent set edilerek modal componentine iletiliyor.


Şimdi ise projeyi buluta bağlama kısmı var. Bulut kısmında Google Cloud kullandık. Önce konsoldan notesapp adında bir proje başlattık. Daha sonra App Engine kullanarak bir app oluşturduk. Daha sonra shell üzerinden “gcloud init” diyerek bulutu initialize ettik.


Veritabanı için Cloud SQL kullanmamız gerekiyordu. Bunun için önce Cloud SQL auth proxy ile cloud’u authenticate ettik. Ardından Cloud SQL Admin API’yi etkinleştirdik. Daha sonra Google Cloud’un dökümanından takip ederek Cloud SQL Auth Proxy’i yükledik. Bütün bu işlemlerden sonra Cloud SQL’de PostgreSQL 15 kullanarak bir instance oluşturduk.

 **SQL** 


Overview  EDIT  IM

Primary instance

 **Overview**

 Cloud SQL Studio

All instances > notes-app-instance-id


 **notes-app-instance-id**

PostgreSQL 15


Ardından database kısmından veritabanımızı oluşturduk.

Databases

All instances > notes-app-instance-id

 **notes-app-instance-id**

PostgreSQL 15

 **CREATE DATABASE**

Name ↑	Collation	Character set	
notes-db	en_US.UTF8	UTF8	⋮
postgres	en_US.UTF8	UTF8	⋮

Daha sonra user ekledik.

	User name ↑	Authentication	Password status	
👤	postgres	Built-in	N/A	⋮
👤	user-postgres	Built-in	N/A	⋮

Ardından, daha önce indirdiğimiz cloud-sql-proxy ile sql'e bağlantı oluşturduk. Sonra ise projemize .env dosyası oluşturduk ve DATABASE_URL ve APPENGINE_URL'yi burada sakladık.

```
.env x
notes-app > backend > .env
1 DATABASE_URL=postgres://user-postgres:password@//cloudsql/notesapp-459915:us-central1:notes-app-instance-id/not
2 APPENGINE_URL=notesapp-459915.uc.r.appspot.com
```

Ardından settings.py dosyasını güncelledik.

```
env = environ.Env(DEBUG=(bool, False))
env_file = os.path.join(BASE_DIR, '.env')
env.read_env(env_file)
```

```
DATABASES = {"default": env.db()}
```

```
STATIC_URL = 'static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

Burada statik dosyaları topladık.

Daha sonra app.yaml dosyası oluşturduk.

```
app.yaml x
notes-app > backend > app.yaml
1 runtime: python312
2
3 handlers:
4 - url: /static
5   static_dir: staticfiles/
6
7 - url: /. *
8   script: auto
```

Daha sonra App Engine'in çalışabilmesi için main.py ekledik.

```
main.py x
notes-app > backend > main.py > ...
1 from backend.wsgi import application
2
3 app = application
```

En sonunda “gcloud app deploy” diyerek projemizi buluta deploy ederek tamamlamış olduk.