

2024-2025 YILI BAHAR DÖNEMİ
BULUT BİLİŞİM VE UYGULAMALARI
PROJE ÖDEVLERİ TESLİM DOKÜMANI



AD – SOYAD

HÜSEYİN TINAZTEPE:

YUSUF TUNÇ:

BORA KOCABIYIK:

ÖĞRENCİ NUMARASI

21290360

22290071

21290270

GitHub Linki: <https://github.com/hsyntinaztepe/cloud-project-data>

PROJE 1 - YouTube Linki: **Çift Katmanlı Web Uygulaması (Web API + Frontend)**

Bora - <https://youtu.be/EyVPyuHyZCw>

Hüseyin - <https://youtu.be/uzP7cAMsIx4>

Yusuf - https://youtu.be/2Uld3BLzK_E

PROJE 2 - YouTube Linki: **Akıllı Veri Analitiği ve Makine Öğrenmesi Uygulaması**

Bora - <https://youtu.be/P8G7fpsn-ig>

Hüseyin - <https://youtu.be/Lqkzts8Yn48>

Yusuf - <https://youtu.be/hDo2oQ2puFU>

PROJE 3 - YouTube Linki: **E-Ticaret Uygulaması (Otomatik Ölçeklendirme ve Yönetim)**

Bora - <https://youtu.be/K70y9Q2krTs>

Hüseyin - <https://youtu.be/9z-mOofC4VY>

Yusuf - <https://youtu.be/Lglhx8VGsTo>

PROJE 4 - YouTube Linki: **Gerçek Zamanlı Veri Akışı ve İşleme (IoT Uygulaması)**

Bora - <https://youtu.be/tA8p7Vt9Aio>

Hüseyin - <https://youtu.be/nQBN8iOe3TE>

Yusuf - <https://youtu.be/ZfUPqlrDnx8>

PROJE 1: Çift Katmanlı Web Uygulaması (Web API + Frontend)

SkyNotes Tam Yıgın Not Yönetim Uygulaması Proje Raporu

1. Proje Genel Bakış

1.1 Proje Amacı

SkyNotes, kullanıcıların dijital not yönetimini kolaylaştırmak için geliştirilmiş bir web uygulamasıdır. Proje, modern web geliştirme teknolojilerini kullanarak kullanıcı dostu bir not yönetim sistemi oluşturmayı hedeflemektedir.

2. Teknoloji Yığını

2.1 Frontend Teknolojileri

SkyNotes'un frontend geliştirme süreci, kullanıcı deneyimini merkeze alan bir yaklaşım ile ilerletildi. Proje başlangıcında Vite ile hızlı ve modern bir React projesi kurularak temeller atıldı. İlk olarak, uygulamanın genel mimari yapısı ve bileşen hiyerarşisi planlandı. React Router DOM kullanılarak sayfa navigasyonu için sağlam bir altyapı oluşturuldu.

Tailwind CSS ile stil oluşturma süreci, hızlı ve responsive bir tasarım sağlamak üzere yapılandırıldı. Her bileşen, kullanıcı deneyimini optimize edecek şekilde detaylı olarak tasarlandı. Geliştirme sürecinde sürekli olarak bileşenlerin yeniden kullanılabilirliği ve kod tekrarından kaçınma prensipleri gözetildi.

- React: Kullanıcı arayüzü geliştirmek için modern JavaScript kütüphanesi
- React Router DOM: Sayfa yönlendirme ve navigasyon için
- Tailwind CSS: Hızlı ve esnek stil oluşturma için utility-first CSS framework'ü
- React Icons: Kullanıcı arayüzü için zengin icon seti

```

14 const Home = () => {
15   const [openAddEditModal, setOpenAddEditModal] = useState({
16     isShown: false,
17     type: "add",
18     data: null,
19   });
20
21   const [showToastMsg, setShowToastMsg] = useState({
22     isShown: false,
23     message: "",
24     type: "add",
25   });
26
27   const [allNotes, setAllNotes] = useState([[]]);
28   const [userInfo, setUserInfo] = useState(null);
29   const [loading, setLoading] = useState(true);
30   const [isSearch, setIsSearch] = useState(false);
31
32   const navigate = useNavigate();
33
34 >   const handleEdit = (noteDetails) => { ... };
35
36 >   const showToastMessage = (message, type) => { ... };
37
38 >   const handleCloseToast = () => { ... };
39
40 >   const handleClearSearch = () => {
41     setIsSearch(false);
42     getAllNotes();
43   };
44
45 >   const handleSearch = (query) => {
46     setIsSearch(true);
47     const filteredNotes = allNotes.filter((note) =>
48       note.title.toLowerCase().includes(query.toLowerCase())
49     );
50
51     setAllNotes(filteredNotes);
52   };
53
54 >   // Get user info
55   const getUserInfo = async () => { ... };
56
57 >   // Get all notes
58   const getAllNotes = async () => { ... };
59
60 >   // Delete note
61   const deleteNote = async (data) => { ... };
62
63 >   // Search note
64   const onSearchNote = async (query) => { ... };
65
66 >   // Pinned note
67   const updateIsPinned = async (noteData) => { ... };
68
69 >   useEffect(() => {
70     getAllNotes();
71     getUserInfo();
72     return () => {};
73   }, []);

```

The screenshot shows a code editor interface with a file tree on the left and a code preview on the right.

File Tree:

```

    < frontend\skynotes-frontend
      < src
        < assets
        < components
          < Cards
            NoteCard.jsx
            ProfileInfo.jsx
          < EmptyCard
            EmptyCard.jsx
          < Input
            PasswordInput.jsx
            TagInput.jsx
          < Navbar
            Navbar.jsx
          < SearchBar
            SearchBar.jsx
          < ToastMessage
            Toast.jsx
          < pages
            < Home
              AddEditNotes.jsx
              Home.jsx
            < Login
              Login.jsx
            < NotFound
              NotFound.jsx
            < SignUp
              SignUp.jsx
            tempCodeRunnerFile.jsx
          < utils
            axiosInstance.js
            constants.js
            helper.js

```

Code Preview:

```

const handleLogin = async (e) => {
  e.preventDefault();

  if (!validateEmail(email)) {
    setError("Please enter a valid email address.");
    return;
  }

  if (!password) {
    setError("Please enter the password.");
    return;
  }

  setError("");

  // Login API call
  try {
    const response = await axiosInstance.post("/login", {
      email: email,
      password: password,
    });

    // Successful login response
    if (response.data && response.data.accessToken) {
      localStorage.setItem("token", response.data.accessToken);
      navigate("/dashboard");
    }
  } catch (error) {
    setError(error.message);
  }
};

```

2.2 Backend Teknolojileri

- Node.js: Sunucu tarafı JavaScript çalışma ortamı
- Express.js: Web uygulaması ve API geliştirme framework'ü
- Mongoose: MongoDB ile etkileşim için Object Data Modeling (ODM) kütüphanesi
- JSON Web Token (JWT): Güvenli kullanıcı kimlik doğrulama mekanizması
- dotenv: Ortam değişkenlerini yönetme
- cors: Cross-Origin Resource Sharing yönetimi

Backend geliştirme süreci, güvenli ve ölçeklenebilir bir API mimarisi oluşturmaya odaklandı. Node.js ve Express.js kombinasyonu, hızlı ve esnek bir sunucu tarafı çözümü sundu. Mongoose ORM kullanılarak MongoDB ile olan veri etkileşimi optimize edildi ve veri modellemesi için güçlü bir altyapı kuruldu.

Kullanıcı ve not modellerinin tasarılarında, veri bütünlüğü ve güvenliği ön planda tutuldu. JWT (JSON Web Token) ile kimlik doğrulama mekanizması uygulandı, bu sayede kullanıcı oturumları güvenli bir şekilde yönetildi. Her API endpoint'i, hem güvenlik hem de performans açısından detaylı olarak test edildi.

```
backend > index.js > authenticateToken
1  require("dotenv").config();
2
3  const config = require("./config.json");
4  const mongoose = require("mongoose");
5
6  mongoose.connect(config.connectionstring);
7
8  const User = require("./models/user.model");
9  const Note = require("./models/note.model");
10
11 const express = require("express");
12 const cors = require("cors");
13 const app = express();
14
15 const jwt = require("jsonwebtoken");
16 const { authenticateToken, authenticateToken } = require("./utilities");
17
18 app.use(express.json());
19
20 app.use(
21   cors({
22     origin: "*",
23   })
24 );
25
26
27 // Create Account
28 > app.post("/create-account", async (req, res) => {
29   ...
30 });
31
32 // Login
33 > app.post("/login", async (req, res) => {
34   ...
35 });
36
37 // Get User
38 > app.get("/get-user", authenticateToken, async (req, res) => {
39   ...
40 });
41
42 // Add Note
43 > app.post("/add-note", authenticateToken, async (req, res) => {
44   ...
45 });
46
47 // Edit Note
48 > app.put("/edit-note/:noteId", authenticateToken, async (req, res) => {
49   ...
50 });
51
52 // Delete Note
53 > app.delete("/delete-note/:noteId", authenticateToken, async (req, res) => {
54   ...
55 });
56
```

```

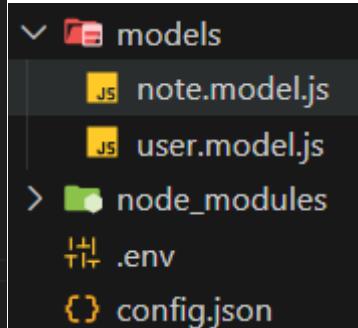
253 // Get All Notes
254 > app.get("/get-all-notes", authenticateToken, async (req, res) => { ...
271 });
272
273 // Update isPinned Value
274 > app.put("/update-note-pinned/:noteId", authenticateToken, async (req, res) => { ...
301 });
302
303 // Search notes
304 > app.get("/search-notes/", authenticateToken, async (req, res) => { ...
334 });
335
336 app.listen(8000);
337
338 module.exports = app;

```

```

backend > ls utilities.js > [o] <unknown> > authenticateToken
1 const jwt = require("jsonwebtoken");
2
3 function authenticateToken(req, res, next) {
4   const authHeader = req.headers["authorization"];
5   const token = authHeader && authHeader.split(" ")[1];
6
7   if (!token) return res.sendStatus(401);
8
9   jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, user) => {
10     if (err) return res.sendStatus(401);
11     req.user = user;
12     next();
13   });
14 }
15
16 module.exports = [
17   authenticateToken,
18 ];
19

```



2.3 Veritabanı

- MongoDB: NoSQL, esnek ve ölçülebilir bulut tabanlı veritabanı

```

backend > {} config.json > ...
1 {
2   "connectionString": "mongodb+srv://borak:TIfrzPJbBMw6krxq@skynotes.gd8
3 }

```

The screenshot shows the MongoDB Atlas interface. On the left, the sidebar displays the project 'Bora's Org' and various service integrations. The main area shows a single database named 'test'. Under the 'test' database, there are two collections: 'notes' and 'users'. The 'users' collection is currently selected. The results table shows four documents with the following data:

```

[{"_id": "ObjectId('681fbccf95a73db72894ble5')", "fullName": "Bora", "email": "kullanici@posta.com", "password": "güçlüparola", "createdAt": "2025-05-10T20:50:28.532+00:00", "__v": 0}, {"_id": "ObjectId('68208c5115fb0439c48d9ff7f')", "fullName": "Bora", "email": "aob.co", "password": "1", "createdAt": "2025-05-11T11:34:18.132+00:00", "__v": 0}

```

3. Uygulama Özellikleri

3.1 Kullanıcı İşlevselligi

- Kullanıcı kayıt ve giriş sistemi
- Not oluşturma, düzenleme ve silme
- Not arama fonksiyonu
- Notları sabitleme (pin) özelliği
- Okunabilir tarih formatı
- Bilgilendirme mesajları (Toast)
- Boş not listesi için özel kullanıcı arayüzü

4. Geliştirme Süreci

4.1 Proje Kurulumu

- Proje npm create vite@latest kullanılarak oluşturuldu
- Tailwind CSS resmi dokümantasyonuna göre entegre edildi
- Google Fonts'dan Noto Sans fontu kullanıldı
- Tarih biçimlendirme için moment kütüphanesi entegre edildi

4.2 Frontend Geliştirme Aşamaları

- Sayfa ve bileşen tasarımlı
- React Router DOM ile navigasyon
- Tailwind CSS ile stil oluşturma
- Özel bileşenler geliştirildi:
 - Navbar
 - Giriş ekranı
 - Şifre girişi bileşeni
 - Arama çubuğu
 - Not ekleme/düzenleme modali
 - Etiket girişi bileşeni

4.3 Backend Geliştirme Aşamaları

- Node.js, Express.js ve Mongoose ile API geliştirme
- Kullanıcı ve Not modelleri oluşturma
- API endpoint'leri hazırlama:
 - Kullanıcı kayıt
 - Kullanıcı girişi

- Not oluşturma
- Not düzenleme
- Not silme
- Not getirme
- Not sabitleme

4.4 Entegrasyon

- Frontend ve backend arasında API bağlantıları kuruldu
- Kullanıcı bilgileri backend'den çekilip görüntüülendi
- Kullanıcıya özel notlar listelendi
- Arama fonksiyonu entegre edildi
- Bilgilendirme mesajları (toast) eklendi

5. Sonuç

SkyNotes projesi, modern web geliştirme teknolojilerini kullanarak kullanıcı merkezli, esnek bir not yönetim uygulaması oluşturmayı başarmıştır. Proje, frontend ve backend teknolojilerinin entegrasyonunu gösteren kapsamlı bir örnek teşkil etmektedir.

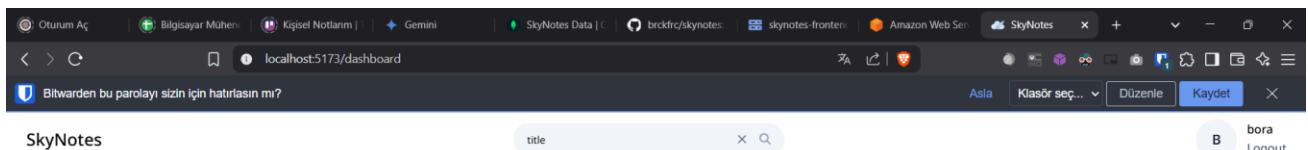
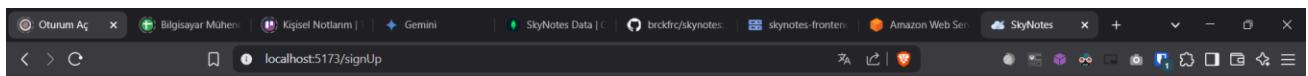
Screenshot of a web browser showing the SkyNotes application interface. The URL is <https://bau18u8ajp.eu-central-1.awsapprunner.com>. The page displays a list of notes:

- Test note5** (11 May 2025)
lorem ipsum5
#tag06
- testando 123** (11 May 2025)
1 2 3 4 5 6 7 8 9
#12 #34
- new title** (11 May 2025)
new contentnew contentnew contentnew content new
contentnew
#newtag1
- 123** (11 May 2025)
a
#16

A large blue plus button (+) is located in the center-right area.

Screenshot of a web browser showing the SkyNotes login screen. The URL is <localhost:5173/login>. The page features a "Login" form with fields for Email and Password, and a "Login" button. Below the form is a link for new users: "Not registered yet? [Create an Account](#)". The top right corner of the window says "Welcome!"





new title 11 May 2025 new content new content new content new content new content new content #newtag1
longtitlelongtitlelongtitlelongtitlelongtitlelongtitlelongtitle 11 May 2025 longcontent longcontent longcontent longcontent longcontent #longtaglongtaglongtaglongtaglongtaglongtaglongtaglongtag



PROJE 2: Akıllı Veri Analitiği ve Makine Öğrenmesi Uygulaması

Proje Konusu: AWS kullanarak telefon özelliklerini bulunduran veri setini eğiterek fiyat aralığı tahmini yapan modeli buluta deploy ettik ve buradan API oluşturarak veritabanı uygulamasında veri analizi gerçekleştirdik.

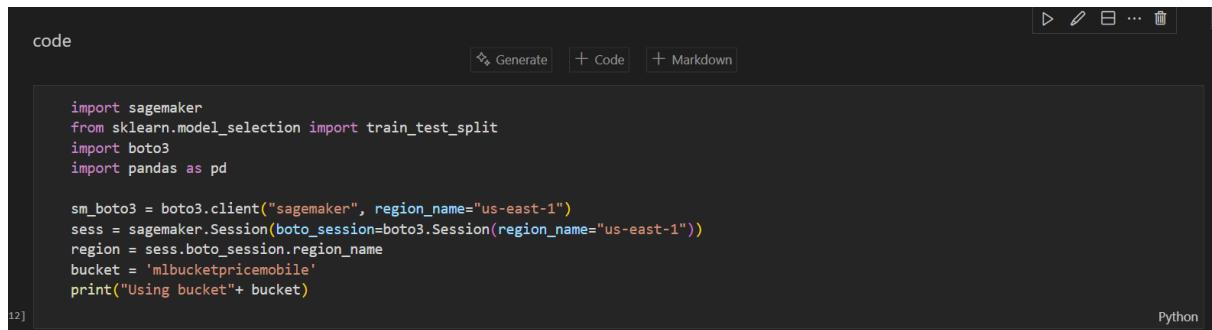
Bu projede;

Backend dili: Python, Node.js

Makine Öğrenmesi Kütüphaneleri,

Veri Tabanı: MongoDB,

Bulut Platformu: AWS (S3 Bucket, Sagemaker, Lambda, API Gateway)



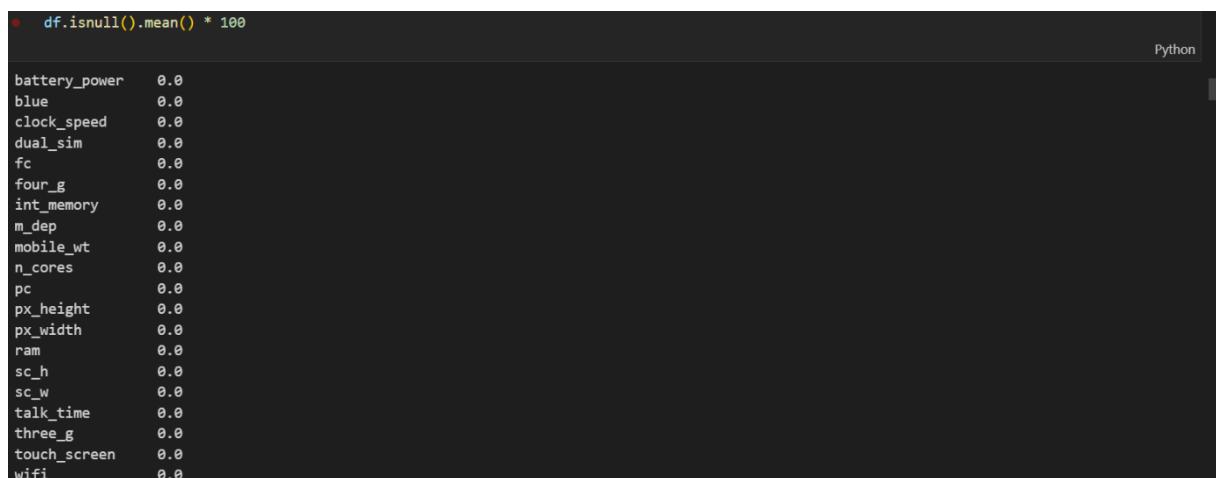
```
code
Generate + Code + Markdown
```

```
import sagemaker
from sklearn.model_selection import train_test_split
import boto3
import pandas as pd

sm_boto3 = boto3.client("sagemaker", region_name="us-east-1")
sess = sagemaker.Session(boto_session=boto3.Session(region_name="us-east-1"))
region = sess.boto_session.region_name
bucket = 'nlbucketpricemobile'
print("Using bucket"+ bucket)
```

12] Python

S3 Bucket oluşturarak localdeki kod ile bağlantı sağlandı.



```
df.isnull().mean() * 100
```

Python

```
battery_power      0.0
blue                0.0
clock_speed         0.0
dual_sim            0.0
fc                  0.0
four_g              0.0
int_memory          0.0
m_dep               0.0
mobile_wt           0.0
n_cores             0.0
pc                  0.0
px_height           0.0
px_width            0.0
ram                 0.0
sc_h                0.0
sc_w                0.0
talk_time           0.0
three_g             0.0
touch_screen        0.0
wifi                0.0
```

Elimizde bulunan dataset'i analiz edilerek eksiklikler gözlemlendi.

```
trainX = pd.DataFrame(X_train)
trainX[label] = y_train

testX = pd.DataFrame(X_test)
testX[label] = y_test
```

Python

```
trainX.to_csv("train-V-1.csv", index=False )
testX.to_csv("test-V-1.csv", index=False )

sk_prefix = "sagemaker/mobile_price_classification/sklearncontainer"
trainpath = sess.upload_data(
    path ="train-V-1.csv", bucket=bucket, key_prefix=sk_prefix
)

testpath = sess.upload_data(
    path ="test-V-1.csv", bucket=bucket, key_prefix=sk_prefix
)
print(trainpath)
print(testpath)
```

Python

Python

Bucket' a var olan train.csv ile test.csv dosyaları yüklandı.

```
1
2 √ from sklearn.ensemble import RandomForestClassifier
3 from sklearn.metrics import accuracy_score, classification_report
4 import pandas as pd
5 import numpy as np
6 import joblib
7 import os
8 import json
9 from io import StringIO
10 import argparse
11
12
13 √ def model_fn(model_dir):
14
15     clf = joblib.load(os.path.join(model_dir, "model.joblib"))
16
17     return clf
18
19
20 √ def input_fn(request_body, request_content_type):
21
22     if request_content_type == "application/json":
23         data = json.loads(request_body)
24         return pd.DataFrame(data)
25     else:
26         raise ValueError(f"Desteklenmeyen content type: {request_content_type}")
27
28
29     if content_type == "application/json":
30         return json.dumps({"prediction": prediction.tolist()})
31     else:
32         raise ValueError(f"Desteklenmeyen content type: {content_type}")
33
34
35     if __name__ == "__main__":
36
37         parser = argparse.ArgumentParser()
38         parser.add_argument("--n_estimators", type=int, default=100)
39         parser.add_argument("--random_state", type=int, default=0)
40         parser.add_argument("--max_depth", type=int, default=None)
41
42         parser.add_argument("--model-dir", type=str, default=os.environ.get("SM_MODEL_DIR"))
43         parser.add_argument("--train", type=str, default=os.environ.get("SM_CHANNEL_TRAIN"))
44         parser.add_argument("--test", type=str, default=os.environ.get("SM_CHANNEL_TEST"))
45
46         parser.add_argument("--train-file", type=str, default="train-V-1.csv")
47         parser.add_argument("--test-file", type=str, default="test-V-1.csv")
```



```

X_train = train_df[features]
X_test = test_df[features]
y_train = train_df[label]
y_test = test_df[label]

model = RandomForestClassifier(
    n_estimators=args.n_estimators,
    random_state=args.random_state,
    max_depth=args.max_depth
)
model.fit(X_train, y_train)

model_path = os.path.join(args.model_dir, "model.joblib")
joblib.dump(model, model_path)

y_pred = model.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

```

Randomforest ile elimizde bulunan dataseti eğitmek için ve daha sonra model eğitiminde kullanılması için script.py üzerine overwrite edildi.

- ```

from sagemaker.sklearn.model import SKLearnModel
from time import gmtime, strftime

model_name = "Custom-sklearn-model-" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
model = SKLearnModel(
 model_data=artifact,
 role="arn:aws:iam::767141477485:role/MLsagemaker",
 entry_point="script.py",
 framework_version=FRAMEWORK_VERSION,
 sagemaker_session=sagemaker_session
)
```

Aws üzerinde IAM rolü belirlendi ve bağlantı kurması için koda eklendi. Arından script.py ile model eğitildi.

```

● from time import gmtime, strftime
import time

endpoint_name = f"mobile-price-classifier-{int(time.time())}"
print("EndpointName={}".format(endpoint_name))

predictor = model.deploy(
 initial_instance_count=1,
 instance_type="ml.m4.xlarge",
 endpoint_name=endpoint_name,
 sagemaker_session=sagemaker_session
)

```

Bu kısımda Sagemaker için endpoint oluşturuldu.

**Roller (23) Bilgi**  
IAM rolü, kısa süreler için geçerli olan kimlik bilgilerine ve belirli izinlere sahip olacak şekilde oluşturabileceğiniz bir kimliktir. Roller, güvenliğiniz varlıklar tarafından üstlenilebilir.

| Rol adı                           | Güvenilir varlıklar    | Son etkinlik  |
|-----------------------------------|------------------------|---------------|
| ml_price_calculator-role-dg0rchny | AWS Hizmeti: lambda    | 8 gün önce    |
| MLSagemaker                       | AWS Hizmeti: sagemaker | 6 dakika önce |

**ml\_price\_calculator**

**İşlev genel bakış Bilgi**

Diyagram | Şablon

**Açıklama**

Son değiştirilme  
2 hafta önce

İşlev ARN'si  
arnaws:lambda:us-east-1:767141477485:function:ml\_price\_calculator

İşlev URL'si | Bilgi

Kod kaynağı Bilgi

Şuradan yükle ▾

Ardından, Lambdaya IAM üzerinden sagemaker erişim rolü verildi. Ardından bu endpoint Lambda'ya bağlandı.

The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar with navigation links like 'APIs', 'Stages', 'Authorizers', etc. The main area is titled 'Resources' and shows a single resource path: '/api-ml-price-calculator'. Below it, a 'POST' method is listed. The 'Resource details' section shows the path as '/' and the 'Resource ID' as 'w38hftlink'. The 'Methods (0)' section indicates 'No methods' defined. There are buttons for 'Update documentation', 'Deploy API', 'Delete', and 'Create method'.

API Gateway kullanılarak MongoDB için RESTAPI oluşturuldu ve deploy edildi

A terminal window showing Python code for interacting with the REST API. The code imports requests and pymongo, sets up a URL and headers, and defines a payload dictionary with various mobile phone specifications. To the right of the terminal, a screenshot of the AWS Lambda function configuration is visible, showing the function name 'ML-Price-Calculator' and the 'Handler' field set to 'lambda\_function.lambda\_handler'.

```
mongo.py > ...
1 import requests
2 from pymongo import MongoClient
3 import json
4
5
6 url = "https://1hfgp7cu95.execute-api.us-east-1.amazonaws.com/prod/api-ml-price-calculator"
7 headers = {"Content-Type": "application/json"}
8
9
10 payload = {
11 "battery_power": 1454,
12 "blue": 1,
13 "clock_speed": 0.5,
14 "dual_sim": 1,
15 "fc": 1,
16 "four_g": 0,
17 "int_memory": 34,
18 "m_dep": 0.7,
19 "mobile_wt": 83,
20 "n_cores": 4,
21 "pc": 2,
22 "px_height": 250,
23 "px_width": 1033,
24 "ram": 3419,
25 "sim": 1}
```

```
30 "wifi": 0
31 }
32
33
34 try:
35 response = requests.post(url, json=payload, headers=headers)
36 response.raise_for_status()
37
38
39 result = response.json()
40
41 body = json.loads(result["body"])
42 print("Prediction sonucu:", body)
43
44 except requests.exceptions.RequestException as e:
45 print("Veri çekme hatası:", e)
46 exit()
47
48
49 try:
50 client = MongoClient("mongodb+srv://useradmin:admin@mlprojecluster.dl7if0o.mongodb.net/?retryWrites=true&w=majority&appName=m+mlproje")
51 db = client["awsmlproje"]
52 collection = db["awsmlprojecollection"]
53 print("MongoDB'ye bağlandı.")
54 except Exception as e:
55 print("MongoDB'ye bağlanma hatası:", e)
56 exit()
57
58 try:
59 payload_with_prediction = payload.copy()
60 payload_with_prediction["prediction"] = body["prediction"][0]
61 collection.insert_one(payload_with_prediction)
62 print("Veri MongoDB'ye kaydedildi.")
63 except Exception as e:
64 print("Veri kaydetme hatası:", e)
65
```

Bu kısımda MongoDB ile AWS API bağlantısı sağlandı ardından local kodda bu bağlantı gösterildi. Test verisi için veri yollandı ve ardından kod çalıştırılarak API kullanıldı

The screenshot shows the Compass interface for a MongoDB database named 'awsmlproje' and a collection named 'awsmlprojecollection'. The top navigation bar includes 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A button for 'INSERT DOCUMENT' is located in the top right. Below the navigation is a search bar with the placeholder 'Type a query: { field: 'value' }' and buttons for 'Reset', 'Apply', and 'Options'. The main area displays the results of a query: 'QUERY RESULTS: 1-1 OF 1'. The document returned is a mobile phone specification:

```
_id: ObjectId('68166e41804982760795e13e')
battery_power : 1454
blue : 1
clock_speed : 0.5
dual_sim : 1
fc : 1
four_g : 0
int_memory : 34
m_dep : 0.7
mobile_wt : 83
n_cores : 4

pc :
px_height : 250
px_width : 1033
ram : 3419
sc_h : 7
sc_w : 5
talk_time : 5
three_g : 1
touch_screen : 1
wifi : 0
prediction : 3
```

Sonuç olarak, MongoDB'ye Tahmin ve test verisi gönderildi ve gözlemlendi.

## **PROJE 3: E-Ticaret Uygulaması (Otomatik Ölçeklendirme ve Yönetim)**

**Proje Konusu: Microsoft Azure ile E-Ticaret sitesi**

<https://nice-sand-04de3bb0f.6.azurestaticapps.net>

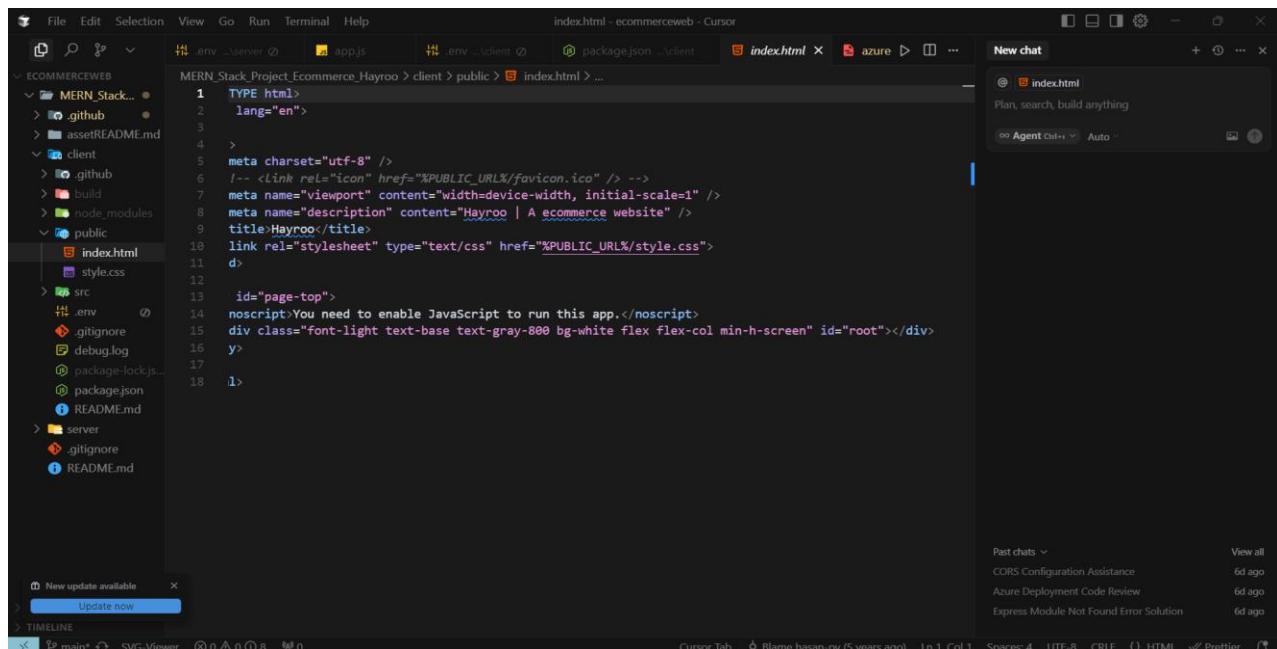
**Bu projede;**

**Frontend dili: JavaScript,**

**Backend dili: Node.js,**

**Veri Tabanı: MongoDB,**

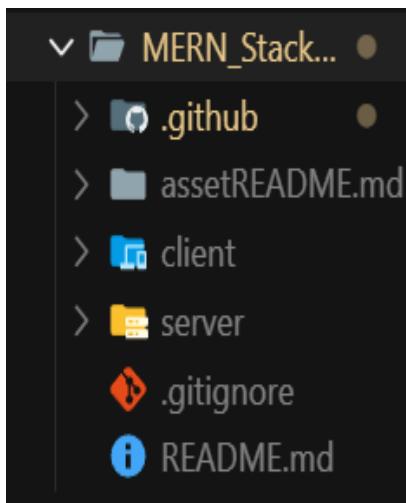
**Bulut Platformu: Microsoft Azure (Azure App Services)**



```
TYPE html>
lang="en">
>
meta charset="utf-8" />
!-- <link rel="icon" href="%PUBLIC_URL%/favicon.ico" /> -->
meta name="viewport" content="width=device-width, initial-scale=1" />
meta name="description" content="Hayroo | A ecommerce website" />
title:Hayroo</title>
link rel="stylesheet" type="text/css" href="%PUBLIC_URL%/style.css">
d>
id="page-top">
noscript>You need to enable JavaScript to run this app.</noscript>
div class="font-light text-base text-gray-800 bg-white flex flex-col min-h-screen" id="root"></div>
y>
l>
```

Bu projede, Cloud bağlantı yapılabilecek hazır website taslağı kullanıldı. Src:

[https://github.com/hasan-py/MERN\\_Stack\\_Project\\_Ecommerce\\_Hayroo](https://github.com/hasan-py/MERN_Stack_Project_Ecommerce_Hayroo)



server dosyası backendi barındırırken (node.js),  
client dosyası frontendi barındırmaktadır. (Html, Css,  
Javascript)

The screenshot shows the Microsoft Azure portal interface for the 'ecommerce-cloudproject' resource group. The left sidebar shows various navigation options like Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, and Help. The main area displays the following resources:

| Name                     | Type             | Location    | Actions |
|--------------------------|------------------|-------------|---------|
| ecommerce-frontend       | Static Web App   | West Europe | ...     |
| ecommerce-frontendgithub | Static Web App   | East US 2   | ...     |
| ecommerce-plan           | App Service plan | West Europe | ...     |
| ecommercecloud           | App Service      | West Europe | ...     |

İlk olarak Microsoft Azure üzerinde dosyaların barınacağı Resource Group oluşturuldu.

Microsoft Azure

Search resources, services, and docs (G+ /)

Copilot

tinaztepe124@gmail.com VARSAYILAN DIZIN

Home >

ecommerce-plan Linux plan

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events (preview)

Resource visualizer

Settings

Monitoring

Automation

Help

https://portal.azure.com/#

Search

Delete Send us your feedback

App Service Plan metrics are not applicable to Free and Shared SKU

Essentials

Resource group (move) : ecommerce-cloudproject

Status : Ready

Location : West Europe

Subscription (move) : Azure subscription 1

Subscription ID : 7db76202-793c-44a4-a7c9-631894610496

Pricing plan : F1

Instance count : 1

App(s) / Slots : 1 / 0

Operating System : Linux

Zone redundant : Disabled

Tags (edit) : Add tags

CPU Percentage

Memory Percentage

Data In

JSON View

Devamında, Linux tabanlı plan hizmete ayrıldı ve resource group bağlantısı yapıldı.

Microsoft Azure

Search resources, services, and docs (G+ /)

Copilot

tinaztepe124@gmail.com VARSAYILAN DIZIN

Home >

ecommercecloud Web App

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Recommended services (preview)

Resource visualizer

Deployment

Settings

Performance

App Service plan

Download tools Add or remove favorites by pressing CtrlL+Shift+F

Search

Browse Stop Swap Restart Delete Refresh Download publish profile Reset publish profile Share to mobile ...

Essentials

Resource group (...) : ecommerce-cloudproject

Status : Running

Location (move) : West Europe

Subscription (move) : Azure subscription 1

Subscription ID : 7db76202-793c-44a4-a7c9-631894610496

Default domain : ecommercecloud.azurewebsites.net

App Service Plan : ecommerce-plan (F1: 1)

Operating System : Linux

Health Check : Not Configured

Tags (edit) : Add tags

Properties Monitoring Logs Capabilities Notifications Recommendations

Web app

Name : ecommercecloud

Publishing model : Code

Runtime Stack : Node - 20-Its

Domains

JSON View

Bu kısımda, Azure Web App Service oluşturuldu.

ecommercecloud.azurewebsites.net

okunaklı hale getir □

{"message": "E-commerce API is running"}

Azure Web App Service üzerinde localdeki kodun backend kısmı ile App Service API ile bağlı. Backend sunucuya yüklendi.

Microsoft Azure    Upgrade    Search resources, services, and docs (G+)

Home > ecommerce-frontendgithub Static Web App

View app in browser Refresh Delete Manage deployment token Send us your feedback

Overview    JSON View

Activity log    Resource group (...) : ecommerce-cloudproject

Access control (IAM)    Subscription (move) : Azure subscription 1

Tags    Subscription ID : 7db76202-793c-44a4-a7c9-631894610496

Diagnose and solve problems    Location : Global

Resource visualizer    Sku : Free

Settings    Tags (edit) : Add tags

Monitoring    Deployment history

Automation    Get started

Help

View your application

Status: Ready    Environment: Production    Domain: https://nice-sand-04de3bb0f6.azurestaticapps.net    Hosting plan: Free

Visit your site

https://portal.azure.com/#@finaztepe124@gmail.com/microsoft.com/resource/subscriptions/7db76202-793c-44a4-a7c9-631894610496/resourceGroups/ecommerce-cloudproject/providers/Microsoft.Web/sites/ecommerce-frontendgithub

Microsoft Azure üzerinde Static Web App oluşturuldu. Localdeki frontend kodu Github aracılığıyla deploy edildi.

Actions    New workflow

All workflows

Azure Static Web Apps CI/CD    Azure Static Web Apps CI/CD

Management

Caches    Attestations    Runners    Usage metrics    Performance metrics

CORS configuration updated    main    last week (2m 4s)

CORS configuration updated    main    last week (2m 7s)

Updated API URL configuration    main    last week (2m 16s)

Updated API URL configuration    main    last week (2m 6s)

Update build script to use export for Linux environment    main    last week (2m 30s)

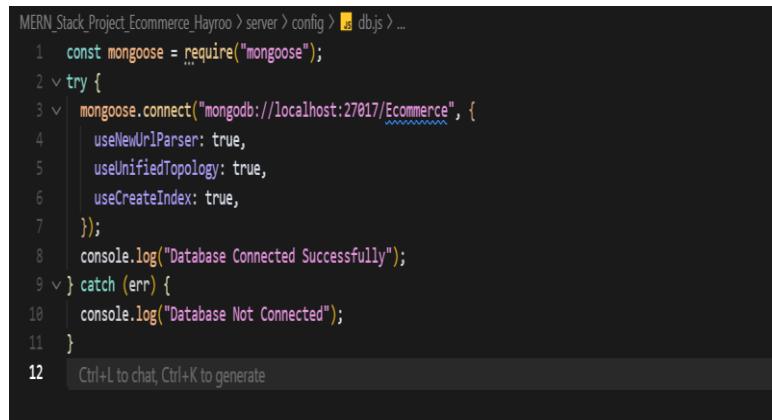
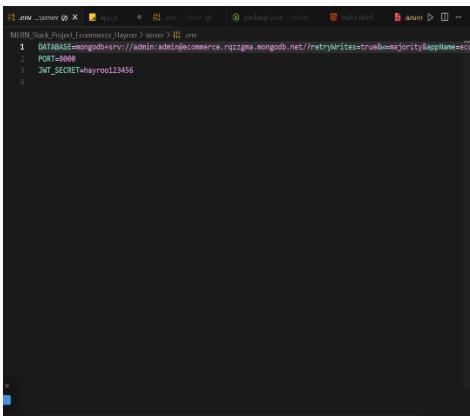
Update build script to use export for Linux environment    main    last week (2m 7s)

```
MERN_Stack_Project_Ecommerce_Hayroo > server > app.js > ...
52 |)
53 }
54 .catch((err) => {
55 console.log("Database Connection Error:", err);
56 console.log("Connection String:", process.env.DATABASE);
57 });
58
59 // Middleware
60 app.use(morgan("dev"));
61 app.use(cookieParser());
62 app.use(
63 cors({
64 origin: "https://nice-sand-04de3bb0f.6.azurestaticapps.net",
65 credentials: true,
66 })
67);
68 app.use(express.static("public"));
69 app.use(express.urlencoded({ extended: false }));
70 app.use(express.json());
1 REACT_APP_API_URL=https://ecommercecloud.azurewebsites.net
2
```

Projenin backend kısmı frontend kısmına Cloud üzerinde bağlanması için localde bağlantı sağlandı.

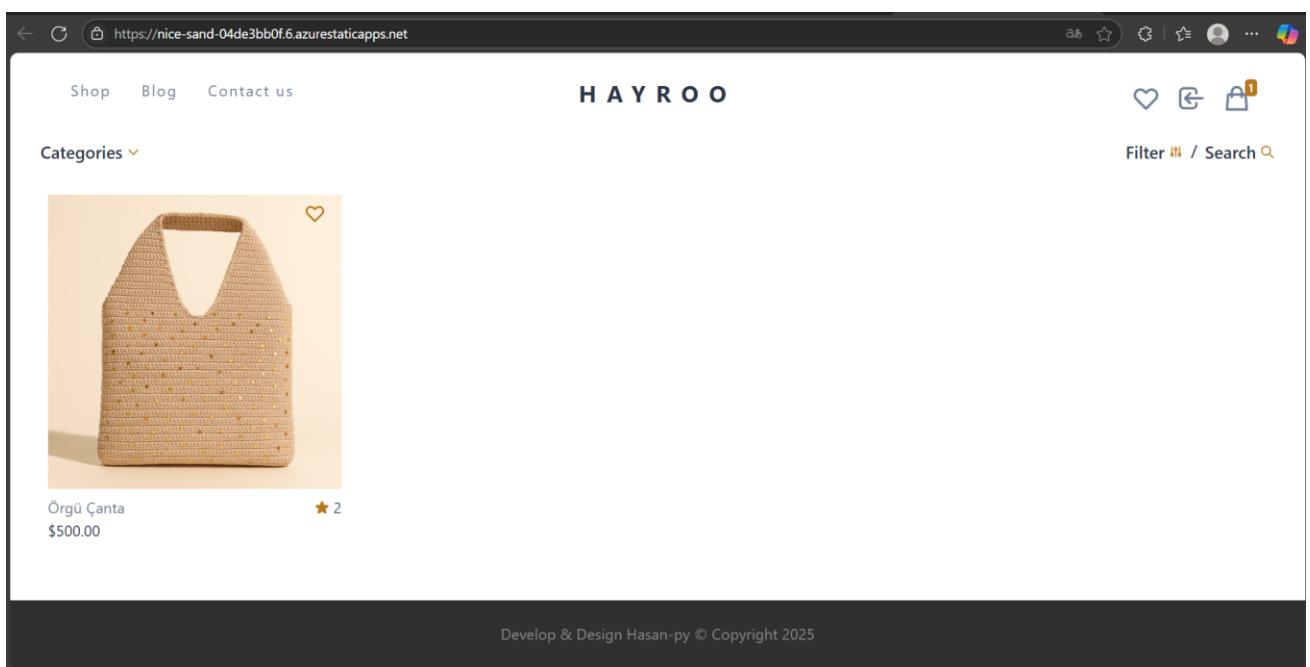
The screenshot shows the MongoDB Atlas Data Services interface. On the left sidebar, under the 'Clusters' section, the 'ecommerce' cluster is selected. Under 'DATABASE', the 'test' database is selected, and within it, the 'categories' collection is shown. The main panel displays the 'test.categories' collection details, including storage size (36KB), logical data size (207B), total documents (1), and index size (36KB). Below this, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A search bar at the top right says 'Generate queries from natural language in Compass'. A query result table shows one document:

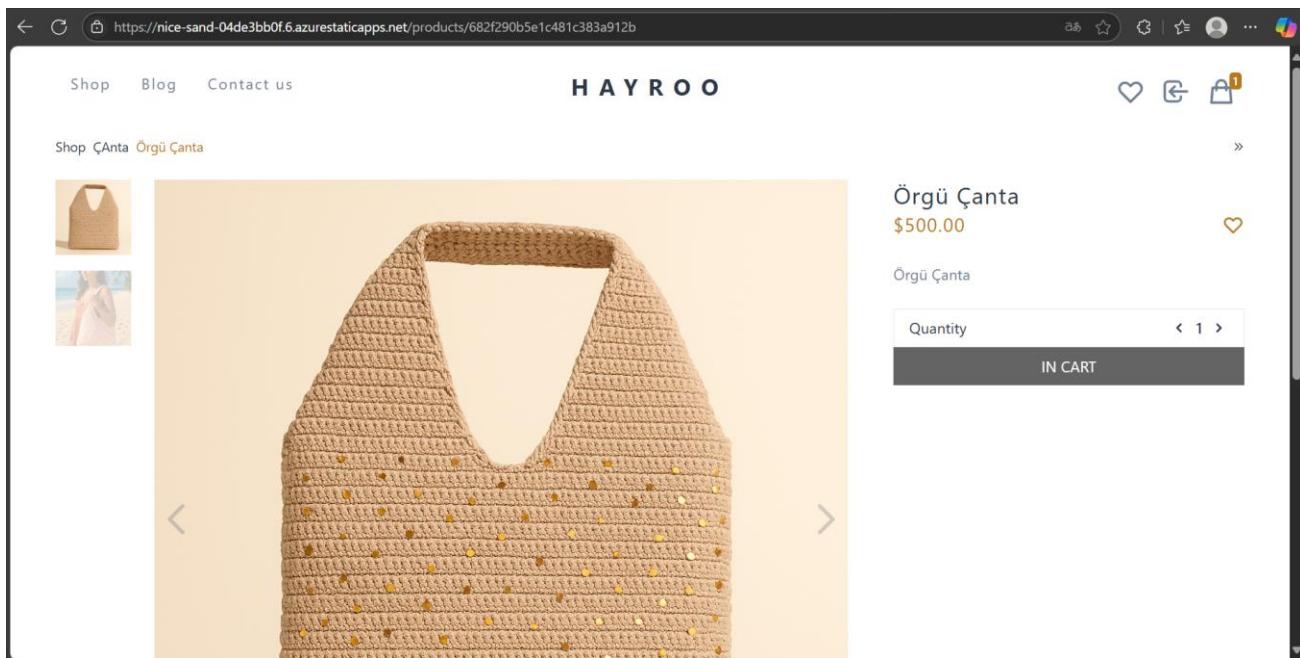
| QUERY RESULTS: 1-1 OF 1                                                      |  |
|------------------------------------------------------------------------------|--|
| <code>_id: ObjectId('682f28e55e1c481c383a9125')</code>                       |  |
| <code>cName : "Çanta"</code>                                                 |  |
| <code>cDescription : "Çanta modelleri"</code>                                |  |
| <code>cStatus : "Active"</code>                                              |  |
| <code>cImage : "1747921125840_ChatGPT Image 17 May 2025 12_21_03.png"</code> |  |
| <code>createdAt : 2025-05-22T13:38:45.950+00:00</code>                       |  |
| <code>updatedAt : 2025-05-22T13:38:45.950+00:00</code>                       |  |
| <code>_v : 0</code>                                                          |  |



```
MERN_Stack_Project_Ecommerce_Hayroo > server > config > db.js > ...
1 const mongoose = require("mongoose");
2 v try {
3 v mongoose.connect("mongodb://localhost:27017/Ecommerce", {
4 useNewUrlParser: true,
5 useUnifiedTopology: true,
6 useCreateIndex: true,
7 });
8 console.log("Database Connected Successfully");
9 v } catch (err) {
10 console.log("Database Not Connected");
11 }
12 Ctrl+L to chat, Ctrl+K to generate
```

Database olarak kullanılması için MongoDB bağlantısı sağlandı.





A screenshot of the admin dashboard for the HAYROO platform. The dashboard features a sidebar with 'Dashboard', 'Categories', 'Product', and 'Order' sections. The main area includes performance metrics for 'Customers' (7% ↑) and 'Orders' (10% ↑), and a section for managing 'Shop Slider Images' with an 'Upload File' button and a message stating 'No slide image found'. Below this is a section for 'Today's Orders 0' with a table header for 'Products', 'Image', 'Status', 'Order Address', and 'Ordered at'. A message at the bottom states 'Total 0 orders found'.

Ve bulut üzerinde çalışan e-ticaret sitesi kullanıma açık.

# Sensora - IoT Sensör Verisi Gerçek Zamanlı İşleme Projesi

## 1. PROJE GENEL BAKIS

### 1.1 Proje Amacı

Bu proje, Google Cloud Platform (GCP) üzerinde gerçek zamanlı IoT sensör verilerinin toplanması, işlenmesi ve saklanması için tam entegre bir sistem geliştirmeyi amaçlamaktadır. Sistem, Cloud Functions, Pub/Sub, Firestore ve BigQuery teknolojilerini kullanarak ölçeklenebilir bir veri işleme pipeline'ı oluşturmaktadır.

### 1.2 Seçilen Proje

#### Proje 2: Gerçek Zamanlı Veri Akışı ve İşleme (IoT veya WebSocket Uygulaması)

Kullanılan teknolojiler:

- Backend Dil:** Python
- Protokol:** Pub/Sub (Google Cloud mesajlaşma servisi)
- Veritabanları:** Firestore (NoSQL), BigQuery (Veri ambarı)
- Bulut Platformu:** Google Cloud Platform

## 2. SİSTEM MİMARİSİ

### 2.1 Genel Mimari

```
[Sensor Veri Simulatoru] → [Pub/Sub Topic] → [Cloud Function] → [Firestore + BigQuery]
```

### 2.2 Bileşen Detayları

#### 1. Veri Kaynağı (Sensör Simülatörü)

- Sıcaklık ve nem verilerini simüle eder
- 5 saniye aralıklarla veri üretir
- JSON formatında Pub/Sub'a gönderir

- Asenkron mesaj kuyruğu
- Yüksek throughput desteği
- Güvenilir mesaj teslimi

#### 3. Cloud Function (process\_sensor\_data)

- Serverless işlem birimi
- Pub/Sub trigger ile otomatik tetikleme
- Python 3.13 runtime

#### 4. Veri Depolama

- Firestore:** Gerçek zamanlı erişim için
- BigQuery:** Analitik sorgular için

#### 2. Pub/Sub Topic (iot-sensor-data)

### 3. TEKNİK UYGULAMA DETAYLARI

#### 3.1 Proje Kurulumu ve Konfigürasyon

##### Google Cloud Proje Ayarları

- Proje ID: `sensora-460619`
- Bölge: `europe-west3` (Frankfurt)
- Etkinleştirilen API'ler:
  - o Cloud Functions API
  - o Pub/Sub API
  - o Firestore API
  - o BigQuery API

##### Gerekli Servisler ve Kaynaklar

```
Projeye konfigürasyonu
gcloud config set project sensora-460619

Cloud Function depolama
gcloud functions deploy process_sensor_data \
 --entry-point process_sensor_data \
 --runtime python313 \
 --trigger-topic iot-sensor-data \
 --region europe-west3 \
 --project sensora-460619
```

Google Cloud projesi `sensora-460619`, Frankfurt bölgesinde (`europe-west3`) bulunuyor.

Cloud Functions, Pub/Sub, Firestore ve BigQuery API'leri etkinleştirildi.

Sağdaki görseldeki komutla proje seçilirken, `gcloud functions deploy` komutuyla Python 3.13 kullanarak `process_sensor_data` fonksiyonu, `iot-sensor-data` Pub/Sub konusu tetikleyicisiyle ilgili bölgede dağıtılmıyor.

Bu yapılandırma, sensör verilerinin bulut ortamında işlenmesini sağlıyor.

#### 3.2 Veri Akış Pipeline'sı

##### Sensör Veri Simülatörü

##### (`sensor_data_publisher.py`)

```
Ana özellikler:
- Rastgele sıcaklık verisi (20-30°C)
- Rastgele nem verisi (%40-60)
- Sabit cihaz ID (sensor-001)
- Unix timestamp ile zaman damgası
- 5 saniye aralıklıkla sürekli veri gönderimi
Örnek Veri Formatı:
{
 "temperature": 25.34,
 "humidity": 52.18,
 "device_id": "sensor-001",
 "timestamp": 1716590123
}
```

##### Cloud Function İşleme Mantığı (`main.py`)

```
Temel işlevler:
1. Pub/Sub mesajını decode etme
2. JSON parsing ve validasyon
3. Önceki veri ile karşılaştırma (%5 değişim kontrolü)
4. Firestore'a kaydetme
5. BigQuery'ye paralel kaydetme
6. Hata yönetimi ve loglama
```

##### Akıllı Filtreleme Algoritması:

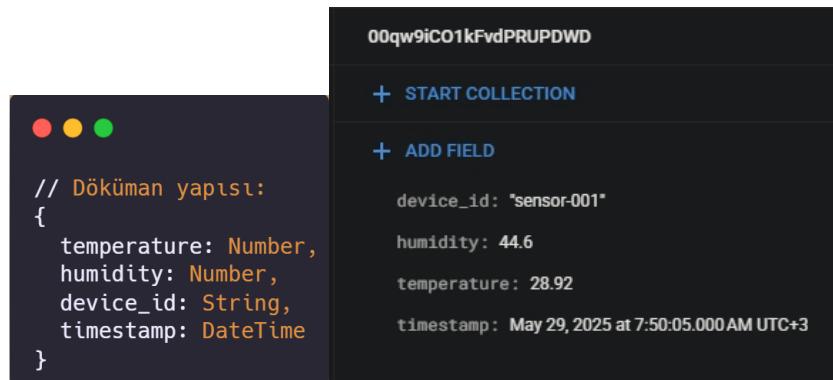
- Önceki ölçüm ile karşılaştırma
- %5'ten az değişimde veri yazılmaz
- Gereksiz veri kirliliğini önler
- Depolama maliyetini optimize eder

Sensör Veri Simülatörü, cihaz kimliği ve zaman damgasıyla birlikte 5 saniyede bir rastgele sıcaklık ve nem verisi üretip JSON formatında gönderir.

Cloud Function, Pub/Sub'dan gelen veriyi alır, doğrular ve önceki veriye göre %5'ten az değişiklik varsa kaydetmez. Geçerli veriler Firestore ve BigQuery'ye kaydedilir, hatalar ise loglanarak yönetilir. Bu filtreleme yöntemi, gereksiz veri girişini engelleyip depolama maliyetini azaltır.

### 3.3 Veri Depolama Stratejisi

#### Firestore Koleksiyonu (sensor\_readings)



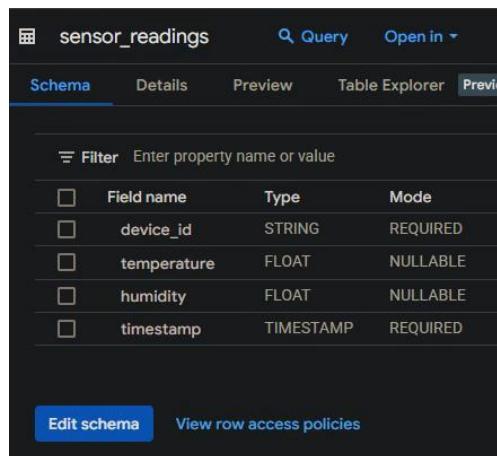
The screenshot shows a document in the Firestore database under the collection 'sensor\_readings'. The document ID is '00qw9iCO1kFvdPRUPDWD'. The document contains the following data:

```
// Döküman yapısı:
{
 temperature: Number,
 humidity: Number,
 device_id: String,
 timestamp: DateTime
}
```

The data fields are:

- device\_id: "sensor-001"
- humidity: 44.6
- temperature: 28.92
- timestamp: May 29, 2025 at 7:50:05.000 AM UTC+3

#### BigQuery Tablosu (sensora\_dataset.sensor\_readings)



The screenshot shows the schema for the 'sensor\_readings' table in BigQuery. The table has the following columns:

| Field name  | Type      | Mode     |
|-------------|-----------|----------|
| device_id   | STRING    | REQUIRED |
| temperature | FLOAT     | NULLABLE |
| humidity    | FLOAT     | NULLABLE |
| timestamp   | TIMESTAMP | REQUIRED |

Sensör verileri, hızlı erişim için Firestore koleksiyonunda saklanır. Aynı veriler, detaylı analiz için BigQuery'de tablo olarak tutulur. Böylece gerçek zamanlı işlem ve uzun dönemli analiz dengelenir.

---

## 4. DEPLOYMENT VE TEST SÜREÇLERİ

### 4.1 Geliştirme Ortamı Kurulumu



```
Sanal ortam oluşturma
python -m venv venv
.\venv\Scripts\activate

Kimlik doğrulama
set GOOGLE_APPLICATION_CREDENTIALS=%PROJECT_CREDENTIALS_PATH%

Bağımlılıklar
pip install google-cloud-pubsub
pip install google-cloud-firestore
pip install google-cloud-bigquery
pip install functions-framework
```

Geliştirme ortamı için Python sanal ortamı oluşturulur ve aktif hale getirilir. Gerekli Google Cloud kütüphaneleri (pubsub, firestore, bigquery) ile functions-framework paketi yüklenir.

Ayrıca, kimlik doğrulama için servis hesabı anahtarı ortam değişkeni olarak ayarlanır. Bu adımlar, uygulamanın bulut servislerine güvenli ve sorunsuz bağlanması sağlar.

## 4.2 Cloud Function Deployment

```
Deployment komutu ve çıktı:
%PATH%\sensora_app\cloud_function_code>
gcloud functions deploy process_sensor_data
--entry-point process_sensor_data
--runtime python313
--trigger-topic iot-sensor-data
--region europe-west3
--project sensora-460619

Başarılı deployment:
✓ [Build] Build ID: 6875294f-8e7d-4958-9e39-ffab4954bf53
✓ [Service] Cloud Run service oluşturuldu
✓ [Trigger] Pub/Sub trigger yapılandırıldı
```

### Deployment Özellikleri:

- Runtime: Python 3.13
- Memory: 256MB
- Timeout: 60 saniye
- Max Instances: 6
- Generation: 2nd Gen (Varsayılan)

Cloud Function, Python 3.13 runtime ile belirtilen bölgede ve iot-sensor-data Pub/Sub konusu tetikleyicisiyle dağıtılmıştır.

Başarılı deploy sonrası Cloud Run servisi oluşturulur ve trigger aktif hale getirilir. Fonksiyon, 256MB bellek, 60 saniye zaman aşımı ve maksimum 6 instance ile çalışır.

Bu yapılandırma, güvenilir ve ölçeklenebilir veri işleme sağlar.

## 4.3 Test ve Doğrulama

### Fonksiyon Testleri

1. **Local Test:** Functions Framework ile yerel test
2. **Integration Test:** Pub/Sub mesaj gönderimi
3. **End-to-End Test:** Veri akışının tamamı

### Log Analizi

```
Cloud Function logları
gcloud functions logs read process_sensor_data --region=europe-west3

Örnek başarılı log:
"Received message: {'temperature': 25.34, 'humidity': 52.18, 'device_id': 'sensor-001'}"
"Data saved to Firestore. Document ID: abc123..."
"Data inserted into BigQuery successfully."
```

Fonksiyonun doğruluğu üç aşamada test edilir. Öncelikle, Functions Framework kullanılarak yerelde çalışması kontrol edilir. Ardından, Pub/Sub'a mesaj gönderilerek entegrasyon testi yapılır.

Son olarak, veri akışının tamamı test edilerek uçtan uca doğrulama sağlanır.

Testler sırasında oluşan loglar, gcloud functions logs read komutuyla incelenir. Başarılı örnek loglar, mesaj alındığını ve verinin Firestore ile BigQuery'ye başarılı şekilde kaydedildiğini gösterir.

---

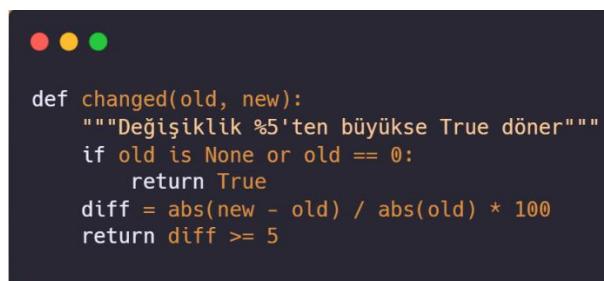
## 5. PERFORMANS VE OPTİMİZASYON

### 5.1 Sistem Performansı

Test ortamında mesaj işleme süresi yaklaşık 200-500 ms arasında değişir ve saniyede 5-10 mesaj işlenebilir. Hata oranı testlerde %1 olarak gözlemlenmiş olup, Cloud Functions SLA kapsamında %99.9 üzeri erişilebilirlik sağlanır.

### 5.2 Optimizasyon Stratejileri

#### Akıllı Veri Filtreleme



```
def changed(old, new):
 """Değişiklik %5'ten büyükse True döner"""
 if old is None or old == 0:
 return True
 diff = abs(new - old) / abs(old) * 100
 return diff >= 5
```

#### Hata Yönetimi

- Try-catch blokları ile exception handling
- Detaylı loglama ve monitoring
- Pub/Sub retry mekanizması  
(RETRY\_POLICY\_DO\_NOT\_RETRY)

Sistemde akıllı veri filtreleme kullanılarak, önceki veriye göre %5'ten az değişiklik gösteren veriler kaydedilmez.

Hata yönetimi için try-catch bloklarıyla istisnalar yakalanır, detaylı loglama yapılır ve Pub/Sub mesajlarının tekrar denemeleri kontrol edilir.

Bu sayede performans artarken hata yönetimi ve izleme güvence altına alınır.

---

## 6. GÜVENLİK VE YETKİ YÖNETİMİ

Proje, minimal yetki prensibine uygun olarak IAM rolleri ve servis hesaplarıyla yönetilir;

Cloud Functions, Pub/Sub, Firestore ve BigQuery erişimleri ayrı ayrı yetkilendirilir ve kimlik doğrulama JSON anahtar dosyasıyla sağlanır.

### 6.1 IAM Rollerleri

- **Cloud Functions Invoker:** Pub/Sub trigger için
- **Pub/Sub Publisher:** Veri gönderimi için
- **Firestore User:** Veritabanı erişimi
- **BigQuery Data Editor:** Veri ambarı erişimi

### 6.2 Servis Hesabı

- Compute Engine default service account kullanımı
- Minimal yetki prensibi (principle of least privilege)
- JSON key file ile kimlik doğrulama

## 7. MONİTORİNG VE LOGLAMA

### 7.1 Cloud Function Monitoring

- **Invocation Count:** Fonksiyon çağrı sayısı
- **Execution Time:** Ortalama çalışma süresi
- **Error Rate:** Hata oranı izleme

```
Detaylı loglama örnekleri:
print(f"Received message: {sensor_data}")
print(f"Data saved to Firestore. Document ID: {doc_ref.id}")
print("Data inserted into BigQuery successfully.")
print("Değişiklik %5 altında, veri yazılmadı.")
```

### 7.2 Loglama Stratejisi

## 8. VERİ GÖRSELLEŞTİRME VE DASHBOARD

### 8.1 Looker Studio Entegrasyonu

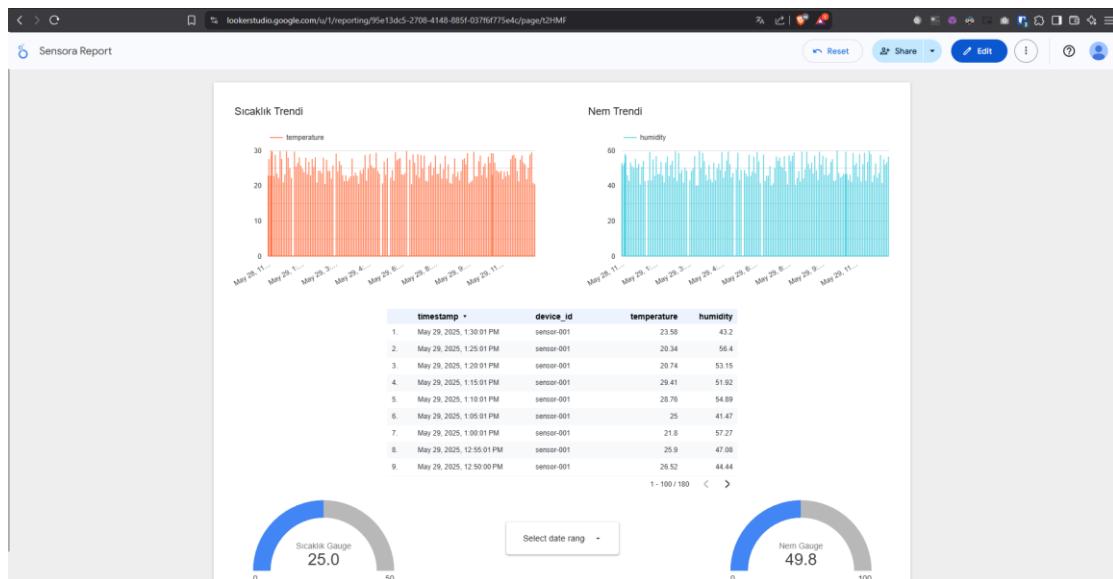
Proje kapsamında toplanan sensör verilerinin gerçek zamanlı izlenmesi ve analiz edilmesi için Google Looker Studio ile entegre bir dashboard oluşturulmuştur.

#### Dashboard Özellikleri

- **Gerçek Zamanlı Veri:** BigQuery veri kaynağından canlı veri akışı
- **Zaman Serisi Grafikleri:** Sıcaklık ve nem değerlerinin zaman içindeki değişimi
- **Cihaz Bazlı Filtreleme:** Device ID'ye göre veri filtreleme imkanı
- **İstatistiksel Özetter:** Ortalama, minimum, maksimum değerler
- **Trend Analizi:** Günlük, saatlik trend görüntümleri

#### Görselleştirme Faydaları

- **Operasyonel İzleme:** Sensör durumlarının anlık takibi
- **Trend Analizi:** Uzun dönemli veri eğilimlerinin belirlenmesi
- **Anomali Tespiti:** Normal dışı değerlerin görsel tespiti
- **Raporlama:** Periyodik raporların otomatik oluşturulması



## 9. PROJE DOSYA YAPISI

```
sensora_app/
├── {account_key}.json # GCP Service Account Key
├── sensor_data_publisher.py # Ana veri gönderici script
├── venv/ # Python sanal ortam
└── schema.json # BigQuery tablo şeması tanımı
├── cloud_function_code/
│ ├── main.py # Ana fonksiyon kodu
│ └── requirements.txt # Python bağımlılıkları
└── publisher_function/
 ├── main.py # Alternatif publisher
 └── requirements.txt # HTTP trigger fonksiyonu
 # Bağımlılıklar
```

---

## 10. ÖĞRENİLEN DERSLER VE ÇÖZÜLEN SORUNLAR

### 10.1 Teknik Zorluklar

1. **Cloud Functions Gen2 Geçişi:** Otomatik olarak 2nd gen'e geçiş
2. **IAM Yetkilendirme:** Servis hesabı rollerinin doğru yapılandırılması
3. **Timestamp Conversion:** Unix timestamp → DateTime dönüşümü
4. **BigQuery Schema:** Tablo şemasının doğru tanımlanması

### 10.2 Çözüm Yaklaşımları

1. **Dokümantasyon Takibi:** GCP dökümanlarından aktif yararlanma
2. **Yapay Zeka Destekli Yardım:** Dokümantasyon yorumu, hata çözümü ve yapılandırma
3. **Log-Driven Development:** Detaylı loglama ile hata ayıklama
4. **Incremental Testing:** Adım adım test ve doğrulama
5. **Error Handling:** Kapsamlı exception yönetimi

---

## 11. SONUÇ VE GELECEK PLANLAR

### 11.1 Başarılı Hedefler

- ✓ Google Cloud üzerinde tam fonksiyonel IoT pipeline
- ✓ Gerçek zamanlı veri işleme ve depolama
- ✓ Kapsamlı hata yönetimi ve loglama
- ✓ Ölçeklenebilir serverless mimari
- ✓ Çoklu veri deposu entegrasyonu (Firestore + BigQuery)

### 11.2 Sistem Avantajları

- **Serverless:** Otomatik ölçekleme, yönetim gerekmeyez
- **Cost-Effective:** Kullanım bazlı ücretlendirme
- **High Availability:** Cloud provider SLA garantileri
- **Real-time:** Düşük latency ile veri işleme
- **Scalable:** Yüksek throughput desteği

### **11.3 Gelecek Geliştirmeler (Roadmap)**

- **Cloud Scheduler:** Otomatik veri gönderimi
  - **Anomaly Detection:** ML tabanlı anomalilik tespiti
  - **Multi-Device Support:** Çoklu sensör desteği
  - **Real-time Alerts:** Threshold bazlı uyarı sistemi
- 

## **12. REFERANSLAR VE KAYNAKLAR**

### **12.1 Kullanılan Teknolojiler**

- **Google Cloud Functions:** Sunucusuz (serverless) hesaplama hizmeti
- **Google Cloud Pub/Sub:** Mesajlaşma servisi
- **Google Cloud Firestore:** NoSQL veritabanı
- **Google Cloud BigQuery:** Veri ambarı (data warehouse)
- **Google Cloud IAM:** Yetkilendirme ve erişim kontrolü
- **Cloud Logging (Stackdriver):** Loglama ve izleme hizmeti
- **Docker:** Uygulamaların konteyner içinde taşınabilir şekilde çalışması için
- **Python 3.13:** Backend geliştirme dili

### **12.2 Önemli Linkler**

- [Cloud Run Functions - Resmi Dökümantasyon](#)
  - [Pub/Sub - Resmi Dökümantasyon](#)
  - [Firestore - Resmi Dökümantasyon](#)
  - [BigQuery - Resmi Dökümantasyon](#)
  - [IoT ve Pub/Sub Entegrasyonu - ChirpStack](#)
  - [Firestore ve BigQuery Entegrasyonu - GCP Rehberi](#)
-