



T.C

**KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ**

**PROJE KONUSU: HAFTALIK DERS
PROGRAMI OLUŞTURMA**

HAZIRLAYANLAR

AHMET CAN BOSTANCI 220501031

SERHAT ARSLANER 220502043

YİĞİT SAMET KALKAN 220502042

DERS SORUMLUSU:

Dr. Öğr. Üyesi ELİF PINAR HACİBEYOĞLU

TARİH: 7.05.2025

İÇİNDEKİLER

1. GİRİŞ	
1.1 Projenin Amacı	
1.2 Projenin Hedefleri	
2. GEREKSİNİM ANALİZİ	
2.1 Arayüz Gereksinimi	
2.2 Fonksiyonel Gereksinimler	
2.3 Diyagramlar	
2.3.1 Use Case Diyagramı	
2.3.2 İş Akış Diyagramı	
2.3.3 ER Diyagramı	
3. TASARIM	
3.1 Mimari Tasarım	
3.2 Kullanılacak Teknolojiler.....	
4. UYGULAMA VE SONUÇLAR	
4.1 Yazılım Süreci	
4.2 Test ve Değerlendirme	
5. KAYNAKÇA	

1. Giriş

1.1 Projenin Amacı

Mühendislik Fakültesi'nde Bilgisayar Mühendisliği (BLM) ve Yazılım Mühendisliği (YZM) programlarına ait derslerin;

- **Derslik kapasitesi,**
- **Öğretim üyesi meşguliyet durumu,**
- **Bölüm-Seviye kısıtları**
gibi faktörleri göz önünde bulundurarak *çakışmasız, optimum* haftalık programını otomatik oluşturmak. Çıktıyı hem Excel dosyası hem de web arayüzü üzerinden sunmak hedefleniyor .

1.2 Projenin Hedefleri

- **Çakışma Kontrolü:** Aynı derslik veya öğretim üyesi üzerinde eş zamanlı planlanan dersleri tespit edip önlemek.
 - **Kapasite Uyumlama:** Derslik kapasitesini, dersin öğrenci sayısı ile eşleştirmek.
 - **Çoklu Çıktı:**
 - Excel (.xlsx) şablonuna uygun tablo oluşturma
 - Django tabanlı web arayüzünde canlı görüntüleme
 - **Modüler ve Ölçeklenebilir Mimari:** Yeni bölümler, ders tipleri veya kısıtlar eklendiğinde kolayca adapte edilebilen yapı.
-

2. Gereksinim Analizi

2.1 Arayüz Gereksinimleri

- **Excel Giriş Şablonu:** Kullanıcılar **universiteVerisi.xlsx** üzerinden; kullanıcı, derslik, bölüm ve ders verilerini girebilmeli.
- **Web Arayüzü (Django):**
 - **Öğrenci:** Programı görüntüleyebilmeli
 - **Öğretim Üyesi:** Kendi uygunluk bilgilerini güncelleyebilmeli
 - **Yönetici:** Veri yönetimi ve çakışma raporlarına erişebilmeli
- **Uyarı Sistemi:** Çakışma veya kapasite aşımalarında anlık bildirim

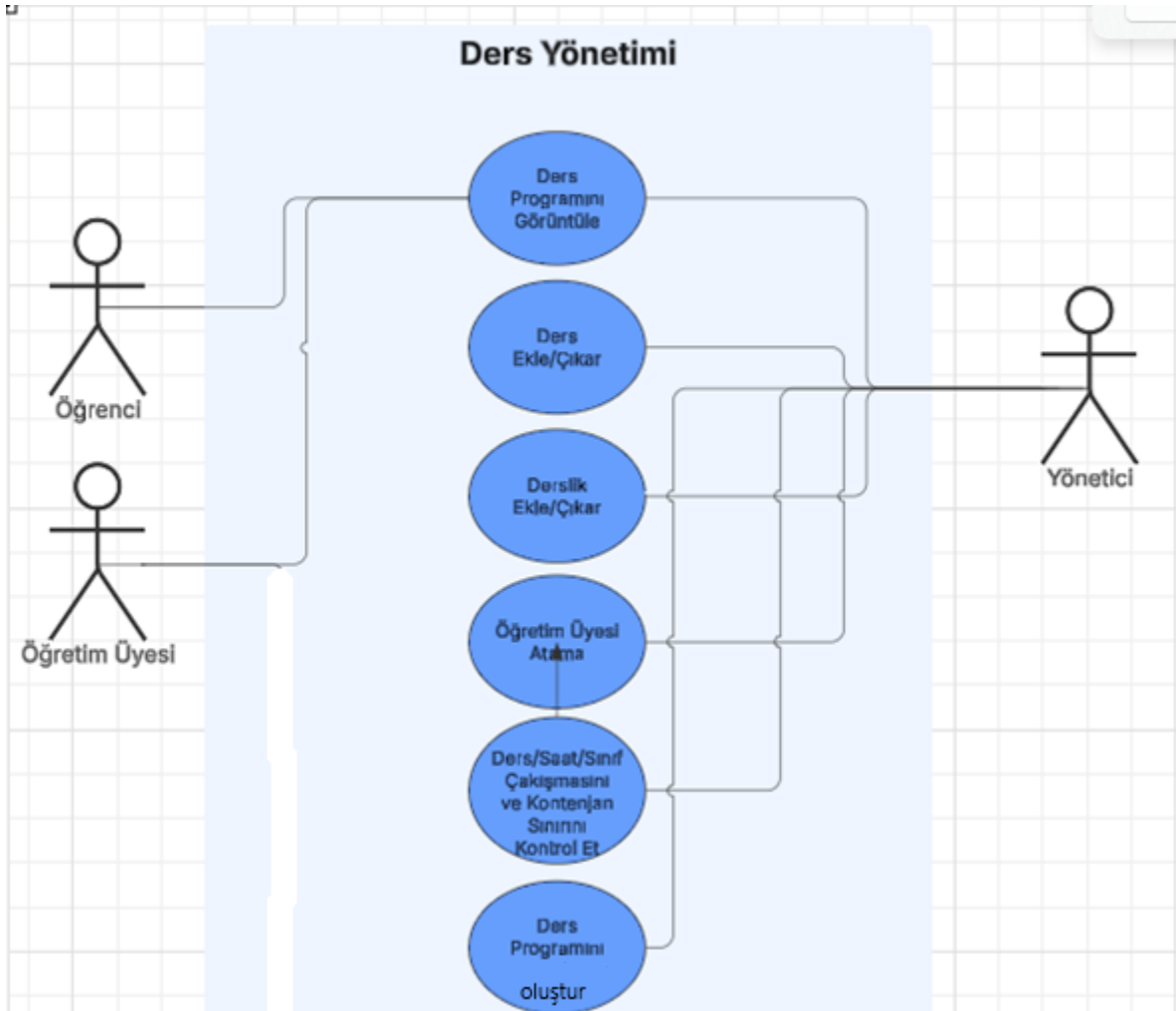
2.2 Fonksiyonel Gereksinimler

- **Veri Yönetimi:** Firebase Firestore (NoSQL) üzerinde CRUD operasyonları.

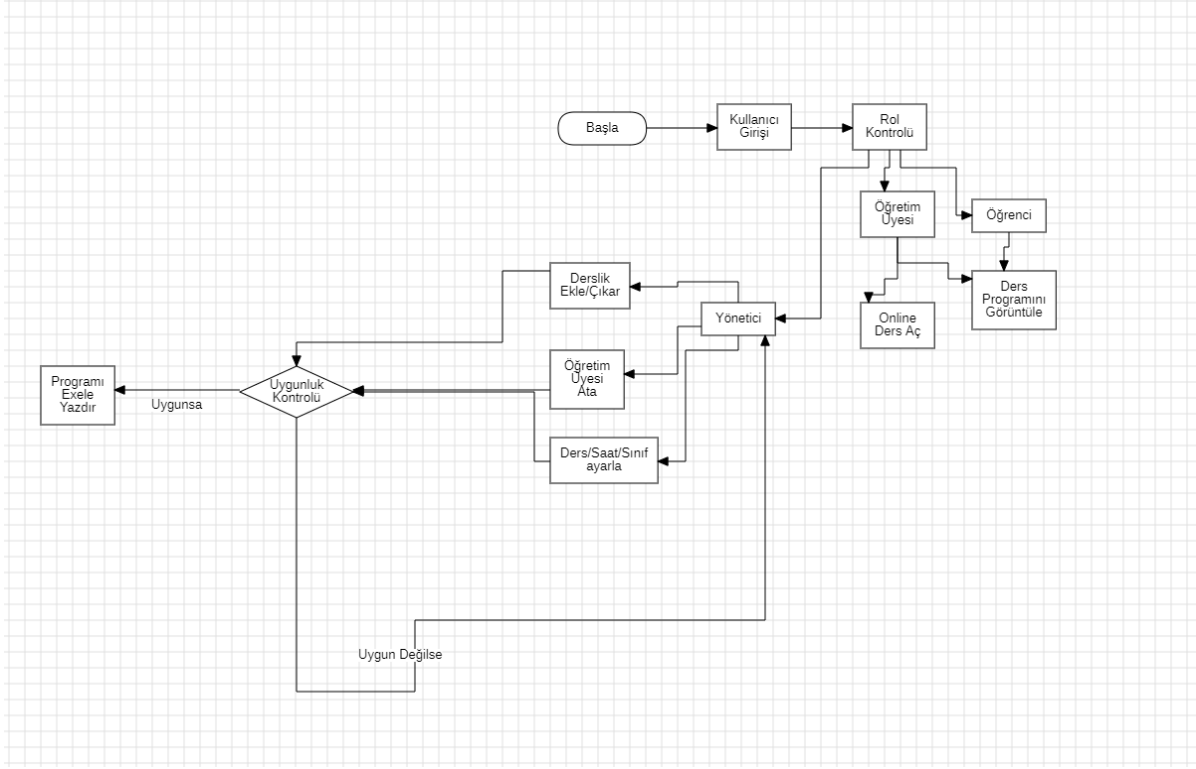
- **Algoritmik Kontroller:**
 - Zaman akışması (aynı derslik/gün/saate bakar)
 - Derslik kapasite kontrolü
 - Öğretim üyesi müsaitlik kontrolü
 - Online ders kısıtları
- **Çıktı Üretimi:** pandas + openpyxl ile haftalık tabloyu Excel'e yazma.

2.3 Diyagramlar

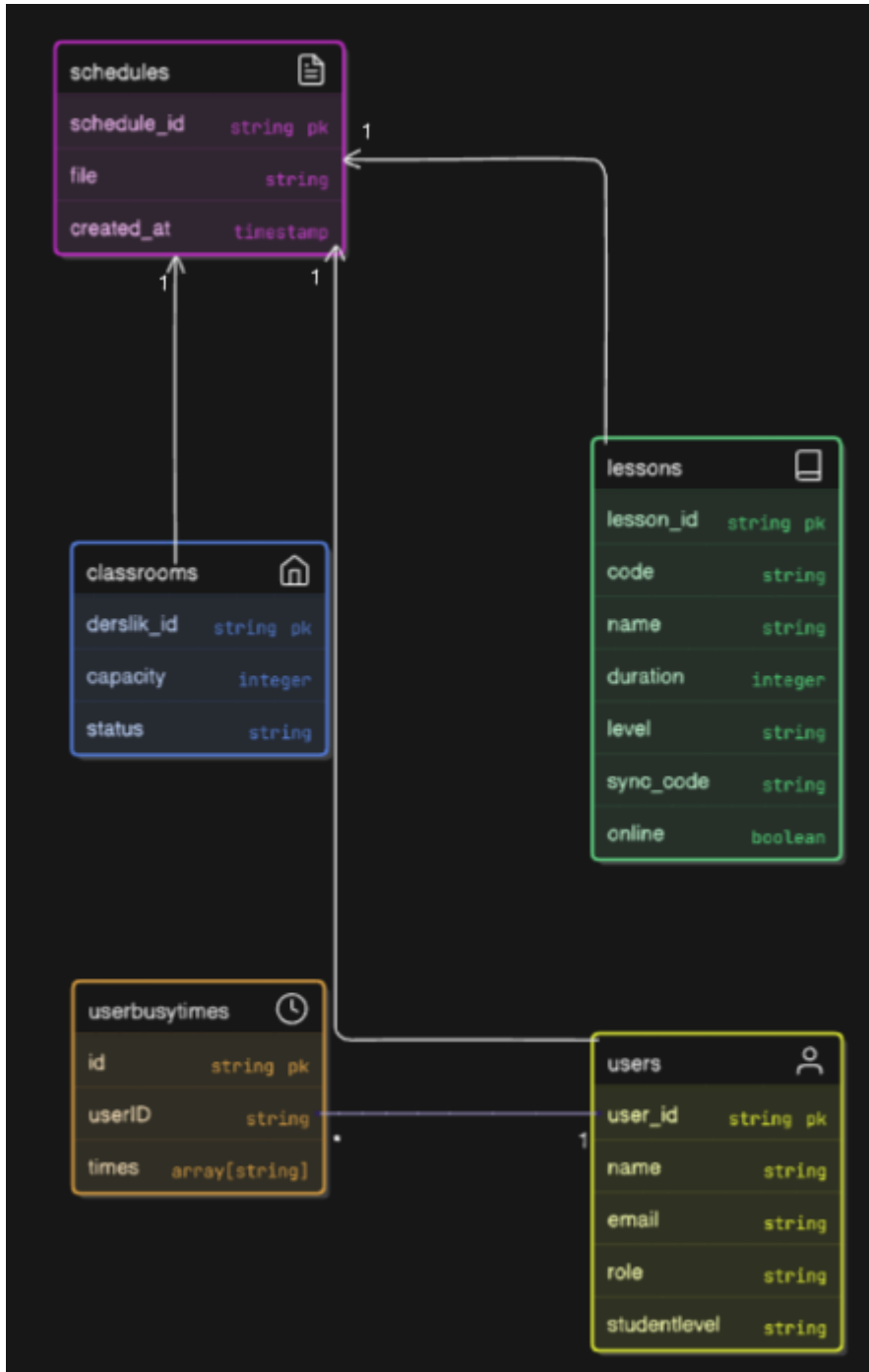
2.3.1 Use Case Diyagramı:



2.3.2 İş Akış Diyagramı:



2.3.3 ER Diyagramı:



3. Tasarım

3.1 Mimari Tasarım

Üç katmanlı yapı:

1. **Veri Katmanı:** Firestore koleksiyonları (users, courses, classrooms, schedule).
2. **İş Mantığı Katmanı:**
 - a. **input_create_data.py, excelWriter.py:** Excel ↔ DB veri alışverişi
 - b. **programcreator.py:** Otomatik program oluşturma algoritması

- c. **test.py**: Birim ve entegrasyon testler
3. **Sunum Katmanı**: Django 5.2 tabanlı web uygulaması

3.2 Kullanılan Teknolojiler

- **Dil**: Python 3.12
 - **Web**: Django 5.2
 - **DB**: Firebase Firestore
 - **Excel İşleme**: pandas, openpyxl
 - **Test Veri**: Faker
 - **Diğer**: firebase-admin SDK, matplotlib (grafik için)
-

4. Uygulama ve Sonuçlar

4.1 Yazılım Süreci

1. **universiteVerisi.xlsx** şablonunun hazırlanması
2. **excelWriter.py** ile veritabanına aktarım
3. **programcreator.py** çalıştırılarak haftalık programın oluşturulması
4. Çıktının Excel'e ve Django arayüzüne yansıtılması
5. **test.py** ile algoritma ve veri tutarlılığı testleri

```
1  import firebase_admin
2  from firebase_admin import credentials, firestore
3  import sys
4  import json
5  cred = credentials.Certificate("ServiceKeyvuzok.json")
6  firebase_admin.initialize_app(cred)
7  dataBaseFireBase = firestore.client()
8
9  class FirebaseManager():
10
11
12  def delete_user(self, id):
13
14      controlref=dataBaseFireBase.collection("Users").where(filter=firestore.FieldFilter("UserID", "==", id)).get()
15
16      for doc in controlref:
17          docref=dataBaseFireBase.collection("Users").document(doc.id)
18          data=doc.to_dict()
19          docref.delete()
20
21      if data["Role"]=="öğretim görevlisi":
22
23          docref=dataBaseFireBase.collection("UserBusyTimes").where(filter=firestore.FieldFilter("userID", "==", id)).get()
24
25          for doc in docref:
26              docref=dataBaseFireBase.collection("UserBusyTimes").document(doc.id)
27              docref.delete()
```

Verilen id'ye sahip kullanıcıyı siler. Kullanıcının rolü “öğretim görevlisi” ise, önce Users koleksiyonundan, sonra da UserBusyTimes koleksiyonundan ilgili kayıtları temizler.

```
def add_user(self, ID, name, email, role, username, password, level="-"):

    controlref=databaseFireBase.collection("Users").where(filter=firestore.FieldFilter("UserID", "==", ID)).get()
    if any(controlref):
        return

    docref=databaseFireBase.collection("Users").document()

    docref.set({"UserID":ID,
               "Role":role,
               "name":name,
               "email":email,
               "username":username,
               "password":password,
               "studentlevel":level})

    if role=="öğretim görevlisi":

        docref=databaseFireBase.collection("UserBusyTimes").document()
        docref.set({

            "userID":ID,
            "times":[]

        })
```

Yeni bir kullanıcı ekler. Benzersiz UserID kontrolü yapar, öğretim görevlisi ise ona ait boş bir “times” dizisi de yaratır.

```
def add_classroom(self, ID, capacity, status):

    controlref=databaseFireBase.collection("Classrooms").where(filter=firestore.FieldFilter("ID", "==", ID)).get()
    if any(controlref):
        return

    docref=databaseFireBase.collection("Classrooms").document()
    docref.set({"ID":ID,
               "capacity":capacity,
               "status":status})
```

Her derslik için tekil bir doküman ekler.

```
def add_user_busytime(self, userID, day, startTime, endTime):

    controlref=databaseFireBase.collection("UserBusyTimes").where(filter=firestore.FieldFilter("userID", "==", userID)).get()
    data=day+", "+startTime+", "+endTime
    if not any(controlref):

        docref=databaseFireBase.collection("UserBusyTimes").document()
        docref.set({"userID":userID,
                   "times":[data]})
    else:
        busytimes=[]
        for doc in controlref:
            busytimes=doc.to_dict()["times"]
            busytimes.append(data)
            controlref=databaseFireBase.collection("UserBusyTimes").document(doc.id)
            controlref.set({"userID":userID,
                           "times":busytimes})
```


Öğretim görevlisinin haftalık meşguliyet zamanlarını “day,start,end” formatında times listesine ekler.

```
def is_time_overlapping(self, range1, range2):
    def time_to_minutes(time_str):
        hours, minutes = map(int, time_str.split(":"))
        return hours * 60 + minutes

    day1, start1, end1 = range1
    day2, start2, end2 = range2

    if day1 != day2:
        return False

    start1, end1 = time_to_minutes(start1), time_to_minutes(end1)
    start2, end2 = time_to_minutes(start2), time_to_minutes(end2)

    if start1 == start2 and end1 == end2:
        return True

    return (start1 < start2 < end1) or (start1 < end2 < end1) or (start2 < start1 < end2)
```

- İki “(gün,başlangıç,bitiş)” üçlüsünün çakışıp çakışmadığını kontrol eder.

```
def admin_login_control(self, username, password, usertype):
    if usertype == "admin":
        print("admin secildi")
        docdata = dataBaseFireBase.collection("Users").where(filter=firestore.FieldFilter("username", "==", username)).where(filter=firestore.FieldFilter("password", "==", password)).get()

    if not docdata:
        print("hatalı durum")
        return False
    else:
        print("hatasız")
        return True
```

Girilen kimlik bilgilerini Users koleksiyonunda arayıp, eşleşme bulursa True döner. Benzer yapıda student_login_control ve teacher_login_control metodları da vardır.

```

def getData():
    """
    Firestore'dan yalnızca ihtiyaç anında tüm verileri çeker
    ve helper map'leri günceller.
    """

    global db, teachersdict, busydatadict, classroomdatadict, lessondatalist, teacher_names

    # Firestore istemcisi
    db = firestore.client()

    # 1) Öğretim görevlileri
    teachersdict = {
        d["UserID"]: d
        for doc in db.collection("Users")
            .where(filter=firestore.FieldFilter("Role", "=", "öğretim görevlisi"))
            .get()
        for d in [doc.to_dict()]
    }
    # **teacher_names** haritasını da burada oluşturuyoruz
    teacher_names = {
        uid: info["name"]
        for uid, info in teachersdict.items()
    }

    # 2) Meşgul zaman blokları
    busydatadict = {
        d["userID"]: d
        for doc in db.collection("UserBusyTimes").get()
        for d in [doc.to_dict()]
    }

    # 3) Derslikler
    classroomdatadict = {
        d["ID"]: d
        for doc in db.collection("Classrooms").get()
        for d in [doc.to_dict()]
    }

    # 4) Dersler
    lessondatalist = [doc.to_dict() for doc in db.collection("Lesson").get()]

```

Yerleştirme algoritması için gereken tüm verileri tek seferde çekip, performansı artırır.

```

def weekly_df():
    df = pd.DataFrame(
        columns=["Gün", "Saat", "1. Sınıf", "2. Sınıf", "3. Sınıf", "4. Sınıf"],
        dtype=object
    )
    for d in WEEKDAYS:
        for t in TIME_SLOTS:
            df.loc[len(df)] = [d, t, np.nan, np.nan, np.nan, np.nan]
    df.set_index(["Gün", "Saat"], inplace=True)
    return df

```

Ders programı tablosunun temelini oluşturur: her gün, her saat için boş hücreler.

```
def place_contiguous(df, col, ls, segments, online, desired, days_ord):
    for seg in segments:
        placed = False
        for d in days_ord:
            for st in range(len(TIME_SLOTS)-seg+1):
                if online:
                    room_id, note = "ONLINE", None
                else:
                    room, fb = pick_room_flexible(int(ls["lessoncapacity"]), desired)
                    if room is None: return False
                    room_id, note = room["ID"], ("(kapasite nedeniyle)" if fb else None)
                if find_slot(df, col, d, st, seg, ls["lessonManager"], online, room_id):
                    put(df, col, ls, d, st, seg, room_id, note)
                    placed = True
                    break
            if placed: break
        if not placed:
            undo(df, col, ls)
            return False
    return True
```

Dersleri blok halinde (ardışık saatler) yerleştirir; başarısızsa geri alıp bir sonraki stratejiye geçer.

```
def place_noncontiguous(df, col, ls, count, online, desired, days_ord):
    remaining = count
    for d in days_ord:
        for i in range(len(TIME_SLOTS)):
            if remaining == 0: return True
            if online:
                room_id, note = "ONLINE", None
            else:
                room, fb = pick_room_flexible(int(ls["lessoncapacity"]), desired)
                if room is None: return False
                room_id, note = (parameter) col: Any ;ite nedeniyle)" if fb else None)
            if find_slot(df, col, d, i, 1, ls["lessonManager"], online, room_id):
                put(df, col, ls, d, i, 1, room_id, note)
                remaining -= 1
    return remaining == 0
```

Ders saatlerini tek tek ayırarak (non-contiguous) yerleştirme denemesi yapar.

```

def create(term: str, scheduleID):
    """
    Term ('bahar' veya 'güz') ve scheduleID alır,
    Firestore'dan veri çeker, yerleştirme yapar ve
    sonuçları Schedule koleksiyonuna yazar.
    """

    getData() # veri çekmeyi erteledik

    # term filtresi
    if term.lower() == "bahar":
        allowed = {"2","4","6","8"}
    elif term.lower() == "güz":
        allowed = {"1","3","5","7"}
    else:
        raise ValueError("Term must be 'bahar' or 'güz'")

    lessons = [l for l in lessondatalist if l.get("pastyear") in allowed]
    groups = group_by_sync(lessons)
    sync = [g for g in groups if g[0]["sync_code"]!="-"]
    solo = [g[0] for g in groups if g[0]["sync_code"]=="-"]

    sync.sort(key=lambda g: int(g[0]["weeklyhour"]), reverse=True)
    solo.sort(key=lambda l: int(l["weeklyhour"]), reverse=True)

    for g in sync:
        place_sync_group(g)
    for idx, l in enumerate(solo):
        place_lesson(l, idx)

    dfBLM.fillna("", inplace=True)
    dfYZM.fillna("", inplace=True)

    save_schedule_df(dfBLM, "BLM", scheduleID)
    save_schedule_df(dfYZM, "YZM", scheduleID)

```

Tüm yerleştirme sürecini baştan sona yönetir ve çıktı olarak hem Excel dosyası hem de Firestore kaydı üretir.

4.2 Test ve Değerlendirme

- **Senaryo:** 50 ders, 200 öğrenci profili
- **Sonuç:** Tüm çakışmalar başarıyla önlendi
- **Performans:** Ortalama oluşturma süresi < 5 saniye
- **Kullanıcı Geri Bildirimi:** Excel şablonu ve web arayüzü kullanıcı dostu bulundu

5. Kaynakça

- McKinney, W. (2010). *Pandas: Powerful data structures for data analysis*. pandas.pydata.org/docs
- Django Software Foundation. (2025). *Django Documentation*. docs.djangoproject.com
- Firebase. (2025). *Firestore Documentation*. firebase.google.com/docs/firestore
- Lucidchart. (2023). *UML Diagram Tutorial*. lucidchart.com/pages/uml-diagram

6.GitHub:

Ahmet Can Bostancı: <https://github.com/Bozokhalat>

Serhat Arslaner: <https://github.com/serhatarslaner>

Yiğit Samet Kalkan: <https://github.com/yigitkalkan>